Smart Memory Tile Architecture

Nuwan Jayasena

Mark Horowitz,
Lance Hammond, Ron Ho, Paul Lee,
Ken Mai, Alex Solomatnikov, Vicky Wong
Outline

• Global Overview
• Tile Architecture
  – Memory System
  – Processor
• Current Status
• Future Work
Global Architecture Review

- 64 tiles on a chip
- 4 tiles grouped into a “quad”
- Quads communicate over global network
- 1GHz in a commodity 0.10µm process, 20 FO4 cycle
- Could replace some tiles/quads with embedded DRAM
Global Network

- Can fit 16 128b buses per edge
- Congestion makes 6 buses more realistic
  - One-hop buses in and out
  - Two-hop buses in and out
  - Must accommodate neighbors’ two-hop buses
- 2 transfers per cycle
- Peak BW in/out of quad 1.8Tbps
- Bisection BW 6 Tbps

Assuming 32λ per bit and 40% power/clock overhead
Global Network Interface

- Shared among tiles of a quad
- Performs several functions
  - Network routing
  - Cache control
    - Handle cache miss requests from local processors
    - Maintain state for outstanding cache miss requests
    - Handle cache coherence requests
  - Gather/scatter and other DMA transfers
  - Synchronization support
    - Maintain state for fine grain synchronization
    - Synchronized stall and enable of local quad’s processors
Quad Interconnect

- Four 64b broadcast buses
  - Phase-pipelined
  - Statically or dynamically allocated

- Off-tile on-quad memory access latency is 5 cycles
Tile Architecture

- Each tile has processing, memory, and communication
- Tile edge is 2.5mm (0.10 µm technology)
- Tile area can fit a MIPS R5000 or 4MB embedded DRAM
Memory System Goals

An efficient reconfigurable memory system

- Reconfigurability
  - Caches
  - Stream register files & FIFOs
  - Indexable local (scratchpad) memory

- Low overhead
  - Re-use memory cell arrays
  - Add minimum features to peripheral circuits
Tile Memory System

- 16 independent, phase-pipelined 8KB SRAM mats
- Each word = 64b data, 4b control, 1b valid
- Reconfigurable read-modify-write and control logic
- Head/tail pointers and strides for hardware FIFOs
- Inter-mat control network for passing control information
Example Cache Configuration

- 2-way set associative cache
- Write-back
- 2 words per line
- LRU replacement

Line Dirty
Line Valid
LRU

Word Dirty
Word Valid
Tile Crossbar Interconnect

- Dynamically routed, phase-pipelined crossbar
- Allows for aggregation of mats into larger memory structures
- Latency is 1/2 cycle per traversal
Latency and Bandwidth Summary

• Latency
  – On-tile = 2 cycles
  – On-quad = 5 cycles

• Peak bandwidth with 1GHz clock
  – To/from tile memories
    • 16GB/s per mat
    • 128GB/s per tile memory system (limited by xbar ports)
  – To/from tile
    • 64GB/s (4 QI ports at 2 64b xfers per cycle)
  – Quad network bisection bandwidth
    • 64GB/s (4 qbuses at 2 64b xfers per cycle)
Tile Processor Design Goals

• Support a wide range of application classes by exploiting various levels of parallelism
  – ILP, TLP, DLP

• Ability to trade off resource utilization vs. flexibility

• Small area and ease of implementation
  – Simple microarchitecture
    • No out of order issue
  – Hardware reuse

→ Achieve these goals by a coarse-grain reconfigurable compute engine
Exploiting Parallelism

• Instruction-level parallelism
  – Quad-issue VLIW
    • Issue up to 4 instructions per cycle
    • Statically scheduled instruction packets w/ dynamic interlocks
    • Intended for applications with high ILP but unpredictable control flow
  – Microcoded control
    • Issue up to 10 operations per cycle
    • Statically scheduled
    • Intended for applications with high ILP and compute requirements and regular control structures

• Data Parallelism
  – Four-way SIMD (microcoded control)
    • All tiles of quad execute the same microcode stream in lock-step
    • All operations, including communication statically scheduled
Exploiting Parallelism (Contd.)

- Thread-level parallelism
  - Coarse-grain threads
    - Each tile concurrently executes 2 independent threads
    - Splits tile processor resources in half
    - Each thread is dual-issue
    - Intended for applications with low ILP
  - Fine-grain threads
    - 4 hardware contexts per coarse-grain thread
    - Low-overhead context switching on long latency (memory) operations
    - Intended for applications with low ILP but lots of TLP
Resource Utilization vs. Flexibility

• Compute density vs. ease of programming & global addressability
  – Microcoded mode
    • Very high compute density
    • High bandwidth to on-tile memory using local addresses
    • Restricted set of programming constructs
    • Off-tile data prefetches decoupled in to gathers/scatters
  – VLIW and threaded modes
    • Lower compute density
    • Lower bandwidth to globally addressed data
    • Simple, familiar programming models
    • Multi-context mode increases utilization via low-overhead context switching

• Flexibility for any execution mode to utilize global and local memory accesses subject to bandwidth and other tradeoffs
Instruction Encodings

• RISC
  – Used for multi-threaded and VLIW modes
  – 32-bit instructions similar to MIPS
  – Pipeline:


• Microcode
  – Very wide instructions (256 bits each)
  – Explicitly controls register files and result busses
  – In SIMD, instructions broadcast over entire quad leading to longer pipeline:
Tile Processor

- 64-bit processor
- Instruction and data accesses interleaved on to tile crossbar on alternate phases of the clock
Instruction Fetch Unit

- Two instruction fetch units per tile
  - Operate independently for multi-threaded modes
  - Ganged together for wide issue (VLIW and microcode) modes
  - Each with 64 or 128 bits/cycle BW

- Each fetch unit supports four contexts
  - Per context state
    - PC
    - Instruction FIFO
    - Thread state
    - Etc.
  - Fetches interleaved at cycle granularity
  - Instruction buffers decouple fetch & execution
Branch Prediction

- Two fully-associative BTBs
  - Function independently for multi-threaded modes
  - Combined for wide-issue modes
  - Each with 32 entries
- Integrated 2-bit predictors
  - Useful mostly for wide-issue modes
- Partial tag (10 bits) storage to minimize area
Context Switch Mechanism

• PC and instruction buffer are replicated to achieve fast context switching
• After a context switch
  – The instruction fetch unit starts fetching for the new active context
  – Instructions are issued from the instruction buffer of the new active context
  – Instructions returned from the I$ are tagged with the context ID so that they are written into the correct instruction buffer
## Context Switch Conditions

- **Conditions**
  1. Static cache miss prediction (generated by compiler)
  2. Predicted-hit operations that miss in the cache
  3. Exceptions

- **Latency**
  - Instruction fetch unit tries to buffer at least 2 instructions for each context
  - Best case latency is 0 cycle for condition 1, and 3 cycles for conditions 2 & 3

### Table 1: Context Switch Conditions

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>T+1</th>
<th>T+2</th>
<th>T+3</th>
<th>T+4</th>
<th>T+5</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>IF1</td>
<td>IF2</td>
<td>DC</td>
<td>RF</td>
<td>EX1</td>
<td>EX2</td>
</tr>
<tr>
<td>X+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X+2</td>
<td></td>
<td>IF1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>DC</td>
<td>RF</td>
<td>EX1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y+1</td>
<td>IF2</td>
<td>DC</td>
<td>RF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y+3</td>
<td>IF1</td>
<td>IF2</td>
<td>DC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>T+1</th>
<th>T+2</th>
<th>T+3</th>
<th>T+4</th>
<th>T+5</th>
<th>T+6</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>IF1</td>
<td>IF2</td>
<td>DC</td>
<td>RF</td>
<td>EX1</td>
<td>EX2</td>
<td></td>
</tr>
<tr>
<td>X+1</td>
<td></td>
<td>IF1</td>
<td>IF2</td>
<td>DC</td>
<td>RF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X+2</td>
<td></td>
<td></td>
<td>IF2</td>
<td>DC</td>
<td>RF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X+3</td>
<td>IF1</td>
<td>IF2</td>
<td>DC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X+4</td>
<td></td>
<td>IF1</td>
<td>IF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X+5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>DC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y+1</td>
<td>IF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y+2</td>
<td>IF1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table above shows the context switch conditions for different time points. The conditions include static cache miss prediction, predicted-hit operations that miss in the cache, and exceptions. The latency for each condition is also provided, with the best case being 0 cycles for condition 1 and 3 cycles for conditions 2 & 3.
Decode/Issue Unit

- Reconfigurable decode unit
- Logical to physical register mapping
- Issue/pipeline logic is fixed

<table>
<thead>
<tr>
<th>From instruction fetch unit</th>
<th>Register mapping</th>
<th>Pipeline registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decoder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decoder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decoder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC update</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Bypass, stall and datapath control logic
- To datapath
Multi-Context Support

- Each register file is partitioned among the hardware contexts
  - A thread control micro-thread (TCMT) is always resident and perform thread control functions
- Instructions in the pipeline are tagged with the context ID
  - Context ID is used to name the thread that suffers a cache miss or an exception
- Decode unit maps logical register numbers into physical register numbers
Universal Datapath

- 64-bit datapath
- Each RF has 32 entries
- Two register files in front of multiplier to support multiply-add
- Two sets of buses – result buses & register data busses

Diagram of Universal Datapath:

- ALU L/S branch unit
- FP adder/Integer ALU
- FP/integer multiplier/divider/sqrt
- RF
- Reciprocal ROM
- FP/integer multiplier/divider/sqrt
- FP adder/Integer ALU
- ALU L/S branch unit
Datapath – VLIW Mode

- Issue 4 RISC instructions per cycle:
  - 2 floating point
  - 2 integer
- Integer and floating point register files are mirrored on both clusters
Datapath – Dual Thread Mode

• Each cluster executes its own thread

• Each thread can issue up to 2 instructions per cycle
  – 1 floating point
  – 1 integer
Datapath – Microcode Mode

• Similar to Imagine:
  – Local register file in front of each functional unit

• Up to 10 operations per cycle:
  – 4 floating point or integer arithmetic operations
  – 2 integer/load/store/branches
  – 4 stream accesses
Load/Store Unit

- Supports memory operations for all execution modes
  - Cache accesses
  - Stream and FIFO accesses
  - Indexed local memory accesses
  - Uncached global accesses
  - Synchronization operations

- Combine primitive memory system ops to implement complex ops
  - e.g. a cache access consists of two mat operations – tag compare followed by read/write or miss handling
- Opcode translator generates one or more memory system opcodes based on processor request
- Address mapper performs mode-specific address translations
Cache Miss Handling

• Cache miss detection
  – Detect by looking at “match” signal from tag mats
  – Stall processor (single context) or context switch (multi-context)
  – Send cache service request to cache controller

• Cache miss completion
  – Load/store unit receives cache controller response
  – Unstall processor and complete register write (single context) or hardware or software completion (multi context)
Cache Miss Completion – Multi-Context Mode

- Load/store unit receives cache controller response
- If thread is still resident in hardware, completion handled in hardware
- If thread is unloaded from hardware, completion in software
  - TCMT calculates the location of the thread in the thread table and updates data register
  - TCMT adds the thread to the ready queue
Multi-Context Memory Structures

- Thread table is mapped to one or more mats configured as local memory
  - Saving/restoring registers does not incur cache misses
- Context loading and unloading done in software, by the TCMT
  - Instructions and data used by the TCMT are stored in local memory
- Ready queue and free list are implemented as hardware FIFO
  - Ready queue: a list of thread IDs of threads that are ready to run
  - Free list: a list of unused entries in the thread table
Multi-Context Mode
Exceptions and Interrupts

• Exception raised in the processor is handled by excepting thread
  – Datapath squashes instructions of the excepting thread
  – Each context has its own exception-related special registers

• Interrupts are handled by TCMT
  – The interrupt is forwarded to the software event queue and the TCMT is invoked
  – TCMT determines which thread should handle the interrupt and updates context state to start handler

• Memory exceptions are treated as interrupts and handled by TCMT
Peak Performance

- Peak compute rates at 1GHz
  - Microcode mode
    - 10 GOPS/sec (64-bit)
    - 8 GFLOPS/sec (single-precision)
  - VLIW and threaded modes
    - 4 GOPS/sec (64-bit)
    - 4 GFLOPS/sec (single precision)
Bandwidth Hierarchy

- For Microcode mode
- Roughly an order of magnitude BW increase at each level
- Global BW limited by conservative assumptions on global network interface

16 GB/s (per quad)

Global/Off-chip Memory

128 GB/s (per quad)

Local (on-quad) memory

1024 GB/s (per quad)

Processor Registers

Compute units
Current Status

• Microarchitecture
  – Tile memory system, tile & quad networks, instruction fetch, decode/issue, and datapath architecture defined
  – Memory interfaces (load/store and cache controller) functionality defined
  – Architectural mapping of VLIW, dual-thread, multi-context, and microcode (streaming) execution modes largely complete.

• Implementation
  – Tile memory test chip in progress

• Simulation infrastructure
  – Standalone tile/quad simulators available for scalar and microcode modes
Future Work

- **Microarchitecture**
  - Complete architectural definition of memory interface units
  - Refine exception and interrupt handling framework
  - Generate architecture specification document
  - Define global network architecture

- **Instruction set**
  - Define bit encodings
  - Specify special and privileged instructions for reconfiguration etc.

- **Reconfigurability**
  - Define dynamic re-configuration framework
Future Work (Contd.)

• Implementation
  – Tape out memory test chip (April 2002)
  – Explore implementation options for tile processor

• Simulation infrastructure
  – Integrate threaded and streaming tile simulators
  – Extend threaded simulator to support multiple contexts
Backup Slides
Reconfiguration

- Points of reconfigurations
  - Memory system
    - Use of control bits in memory mats
    - Quad and tile network arbitration scheme
    - Inter-mat control network
  - Processor memory interface
    - Mapping of CPU to memory opcodes
    - Address mapping table
  - Processor
    - Execution mode

- Cost of reconfiguration
  - Order 100 cycles to reconfigure tile memory and processor
  - Order 10000 cycles to flush and re-fetch tile memory data

- Key to dynamic reconfiguration is to avoid flushing memory
  - Support mixed memory models
  - Memory structures that can be reconfigures without full flush?
Example FIFO Configuration

- Probe head valid status
- Conditional read, valid/pointer update

- 2 memory operations for a FIFO operation
  - Probe valid status
  - Conditional memory operation (based on valid status)

- Example of FIFO head read
Memory System Issues

• What reconfigurability is useful?

• Naming and address translation
  – Stay virtual on the quad (translation in network interface)

• How to reduce overhead of interconnect network?
  – Phase-pipeline
  – Share data lines between read/write/compare
  – Use low-swing signaling

• Arbitration for memory and network resources

Use testchip as a vehicle to explore these issues ...
## Execution Mode Summary

<table>
<thead>
<tr>
<th>Mode</th>
<th>Instruction Encoding</th>
<th>Targeted Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>VLIW</td>
<td>RISC</td>
<td>ILP</td>
</tr>
<tr>
<td>Dual-thread</td>
<td>RISC</td>
<td>Course-grain TLP</td>
</tr>
<tr>
<td>(single context)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dual-thread</td>
<td>RISC</td>
<td>Fine-grain TLP</td>
</tr>
<tr>
<td>(multi-context)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microcode</td>
<td>Microcode</td>
<td>ILP</td>
</tr>
<tr>
<td>(single tile)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microcode</td>
<td>Microcode</td>
<td>ILP, Data</td>
</tr>
<tr>
<td>(SIMD)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Instruction Fetch Configurability

• Current configurability
  – Select one of the 3 modes (VLIW, multi-threaded, microcode)
  – # of contexts per thread (multi-threaded mode only)
  – Instruction fetch latency (microcode mode only)
  – Memory and cache controller interface

• Potential points of future configurability
  – Fetch loop and instruction width
  – Context selection scheme (multi-threaded mode only)
  – Instruction buffer control
Load/Store Unit Requirements

- Flexible memory system
  - Primitive operations used to implement more complex operations
    e.g. Cache reads are two mat operations (read, compare) followed by appropriate
    steps according to match value & control bits
  - Need mapping of each memory instruction to sequence of primitive
    operations.
- Tasks needed for each instruction
  - Opcode translation
  - Address translation
  - Control information (which return bits to look at, etc.)
Memory Instructions

- Memory instructions to support
  - Cache accesses
    - Load, store, prefetch
    - Test-and-set
    - Synchronization primitives (future/synchronized load/store)
  - Direct local memory accesses
    - Thread tables for multithreaded mode
    - Scratchpad for microcode(stream) mode
  - FIFO accesses
    - Stream accesses for microcode mode
    - Task queue for multithreaded mode
  - Uncached load/stores
  - Etc. (Sync, config writes, …)

- Need to translate these instructions into (possibly multiple) requests to appropriate memory mats or global interface
Fine-Grain Synchronization

A: Sync $LD X$

Wait bit set A waiting X

X A Sync

Re-enable A

B: Sync $ST X$

Processor

Thread A

Thread B

Memory

F/E W

Cache controller

A: Sync $LD X$

Re-enable A
Hardware Support

- Program states of a hardware context
  - A subset of integer and floating-point general purpose registers, and condition code registers
  - PC, exception PC and instruction buffers
  - A set of thread status registers

- Instruction fetch unit
  - Makes context switch decisions and performs context switches
  - Prefetches instructions for inactive contexts during idle fetch cycles

- Decode unit
  - Identifies memory operations with long predicted latency
  - Maps logical register numbers to physical register numbers
Hardware Support (2)

- **Datapath**
  - Each instruction in the pipeline is tagged with a context ID
  - Supports nullifying of instructions belonging to a specific context

- **Load/store unit**
  - Signals the instruction fetch unit to switch context on cache miss
  - Supports synchronization operations and waiting mechanisms

- **Memory system**
  - Provides local storage for thread states and thread control structures
  - Supports fine-grain synchronization operations and waiting mechanisms
  - Maintains additional information in outstanding memory request record to facilitate load instruction completion and thread re-enabling
Thread Creation and Termination

• Number of threads in a tile is limited by the number of memory mats devoted to the thread table

• Steps for thread creation
  1. Allocate a stack and update appropriate OS data structures
  2. Remove a thread table entry (TTE) from the free list
  3. Initialize appropriate control and data register values of TTE
  4. Add the address of TTE (thread ID) to the ready queue

• Steps for thread termination
  1. Free the stack and update appropriate OS data structures
  2. Add the address of TTE to the free list
  3. Change the status of the context to invalid
  4. Context switch
Fine-Grain Synchronizations

- Full/empty (F/E) bit is stored in the cache coherence directory when the word is not in cache.
- If any waiting bit is set when a cache line is invalidated, the cache controller re-enables associated stalled threads as the full/empty state may have been changed.
- Types of synchronization operations

<table>
<thead>
<tr>
<th></th>
<th>Load</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Before</td>
<td>After</td>
</tr>
<tr>
<td>Sync</td>
<td>Full</td>
<td>Empty</td>
</tr>
<tr>
<td>Future</td>
<td>Full</td>
<td>Full</td>
</tr>
</tbody>
</table>
SIMD Streaming Mode Overview

• Achieve high compute density and high memory bandwidth

• Intended for applications with regular memory access and control flow, and amenable to static analysis

• A quad in streaming configuration is controlled as a coprocessor by a VLIW or threaded tile

• Stream gather/scatter performed by global network interface
Local Resource Allocation

- Dynamically tradeoff stream, indexable, and microcode storage
- Quad and tile network BW statically partitioned
  - Instruction broadcast
  - Local memory access (up to 4 words/cycle/tile)
  - Local communication (up to 2 words/cycle/tile)
  - Stream gather/scatter
Streaming Mode
Communication Hierarchy

- Quad network bandwidth statically partitioned
  - Stream transfers
  - Inter-tile communication
  - (Microcode broadcast)

- Tile crossbar bandwidth allocated by compiler
  - Local memory access
  - Inter-tile communication

Diagram:

- Global/Off-chip Memory
- Quad Network
  - NI
  - Global Network
  - Quad Network
    - Tile Mem
    - Tile Mem
    - Tile Mem
    - Tile Mem
    - Quad Network
    - LRFs
    - LRFs
    - LRFs
    - LRFs
    - Tile XB
    - Tile XB
    - Tile XB
    - Tile XB
    - Arith. Units
    - Arith. Units
    - Arith. Units
    - Arith. Units
    - Tile XB
Microcode Broadcast

- Microcode distributed on all 4 tiles
- Each tile reads a neighbor’s portion of microcode
- All other tiles grab the instruction word from quad network
Memory Testchip Goals

• Take design from architecture to full implementation
  – Make sure all i’s are dotted and t’s are crossed

• Quantify overheads of reconfigurability
  – Delay
  – Power
  – Area

• Exercise reconfigurability of architecture
  – Caches
  – Streaming FIFOs
  – DMA and cache control
Testchip Floorplan

- 8 2KB (512x32b) memory mats
- Tile crossbar with 4 inputs
  - 2 CPU ports
  - 2 quad bus ports
- Tapeout April 1, 2002 in TSMC 0.18µm technology

Area estimate assumes a 0.18µm technology