
Merrimac Multi-Node Memory Model

Jung Ho Ahn
April 20, 2004

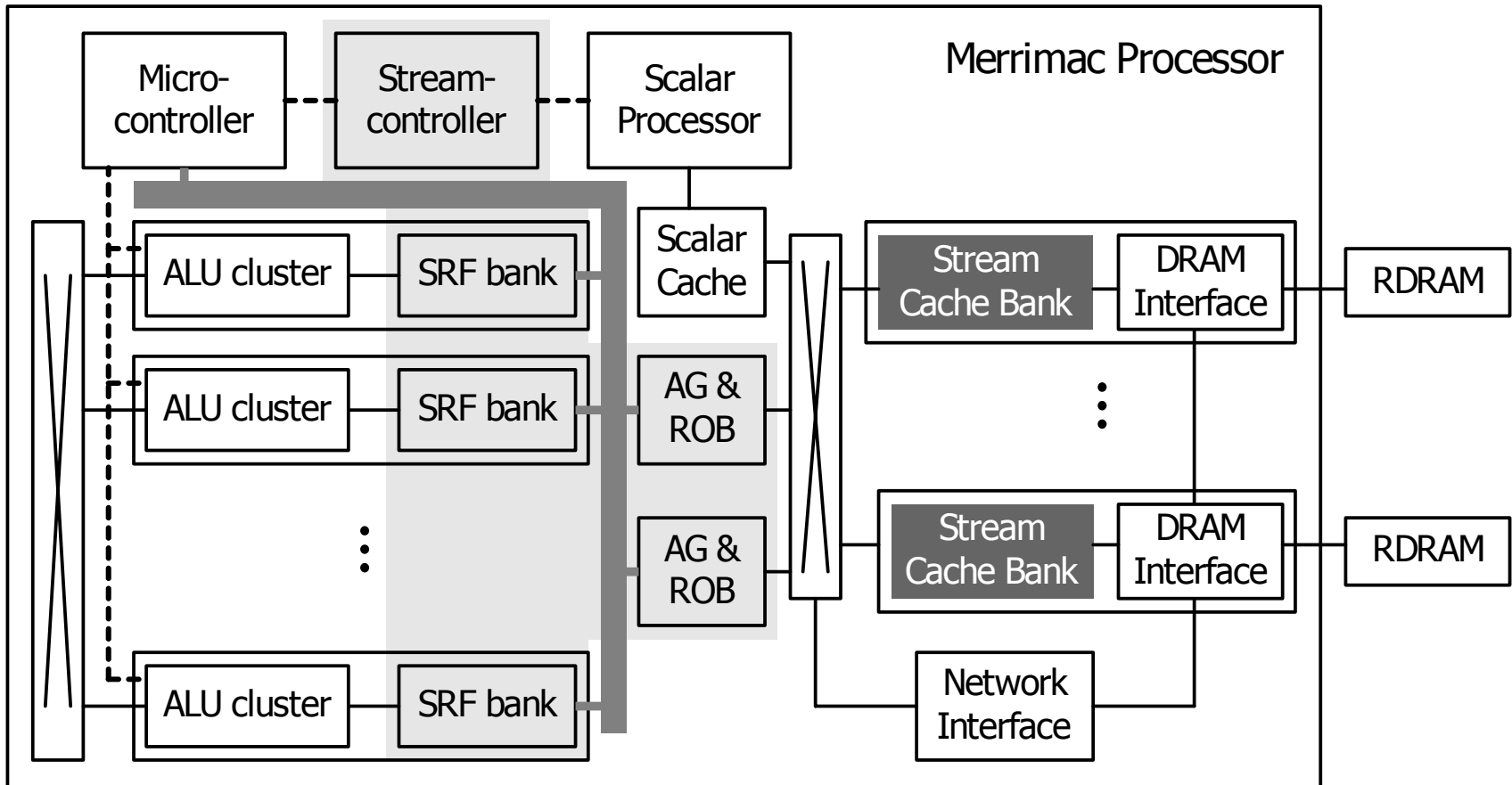
Outline

- Review of single node memory system
- Architectural features in multi-node
- Summary of UPC extensions
- How multi-node memory model supports UPC
- Features not in UPC – Cache coherence/transition

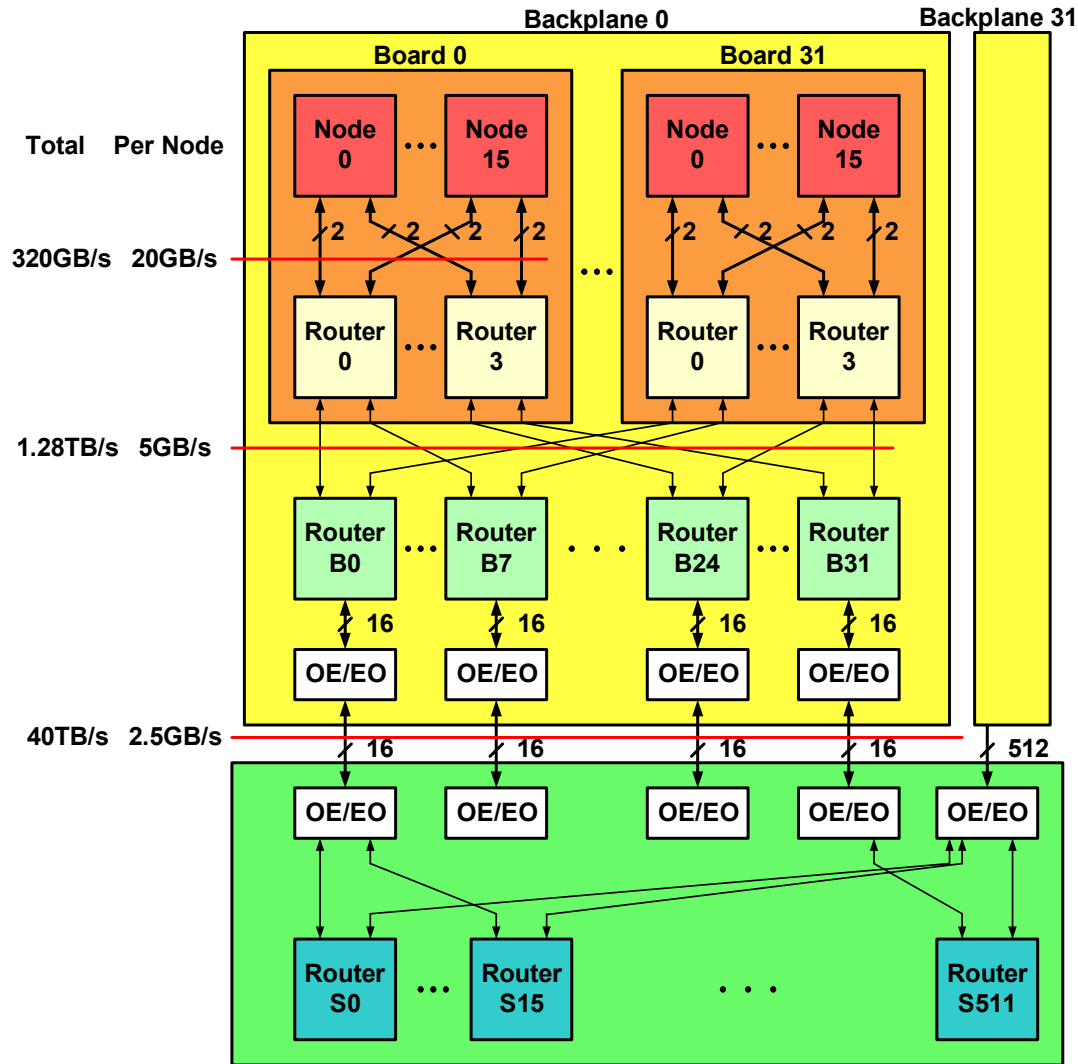
Quick review of single node memory

- Design focus
 - more priority on higher throughput rather than less latency
- Memory system consists of
 - 2 address generator, 2 reorder buffer
 - More than one word processed
 - Reorder buffer enabled Memory Access Scheduler
 - 8 stream cache bank, 16 DRAMs
 - interleaved by memory address
 - Stream Cache
 - Bandwidth amplifier
 - DDR II SDRAM or RDRAM
 - Stream and scalar unit share the same memory space

Single node Merrimac architecture



BW hierarchy & interconnection network



Summary of multi-node memory model

- Distributed shared memory system (DSM)
 - NUMA : non uniform memory access
 - lower bandwidth, higher latency for remote data accesses
- Segmentation
 - From virtual address to physical address
 - one segment can cover the memory area of multiple node
 - interleave factor
 - cache access policy changing over time on the unit of segment
- Synchronization
 - Hardware barrier or software implementation
 - using atomic memory operations (AMOs) like fetch&op

Introducing UPC

- UPC
 - Unified Parallel C
 - An explicit parallel extension of ANSI C
 - Considering distributed shared memory (DSM) system
- Why introduce UPC?
 - Brook will be extended adopting UPC features
 - One of representative parallel extension for DSM systems
 - What people expect from the DSM systems
 - Merrimac don't have to support all the constructs of UPC efficiently.

Summary of UPC

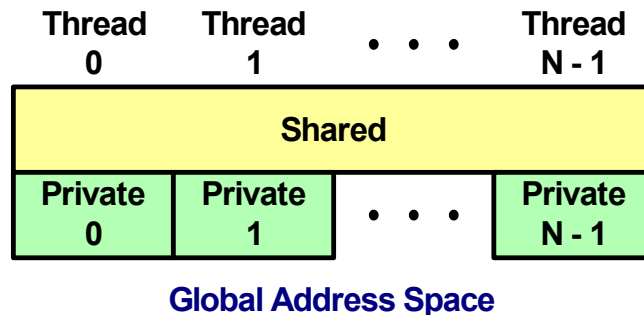
- The programmer is presented with
 - One single share & partitioned address space
 - Each variable physically associated with a single processor
 - SPMD : single program multiple data
- Constructs UPC extends ANSI C with
 - Explicitly parallel execution model
 - Shared address space
 - Synchronization primitives
 - Memory consistency model

UPC : Explicitly parallel execution model

- Execution model
 - Single Program Multiple Data
 - Similar to message passing style of programming (MPI)
 - Over multiple nodes/processors/threads (UPC term)
 - Collection of threads in a single global address space
 - Affinity : the physical association between data items and UPC threads
 - Threads have shared memory space and private memory space
- How Merrimac memory system supports?
 - Execution model is for the control units
 - Not directly related to the memory system
 - Scalar unit will deal with it

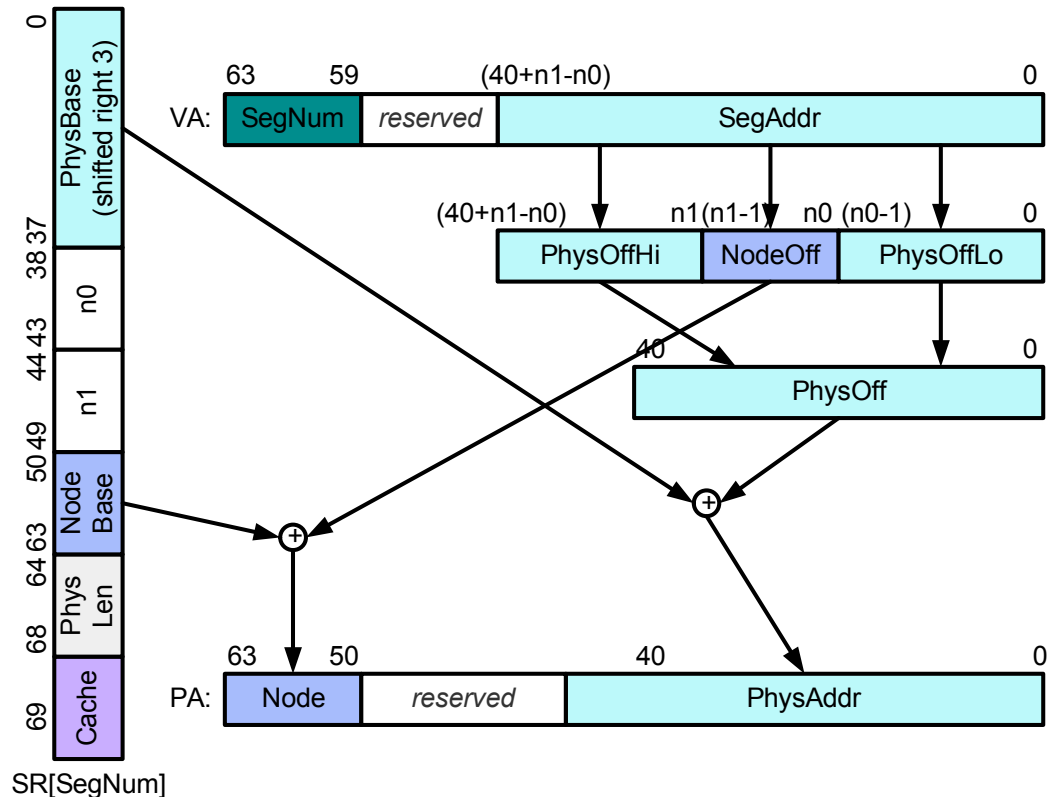
UPC : Shared address space

- A variable : either shared or private
 - New C type qualifier “shared”
 - Private
 - memory address viewable to the local thread only
 - Shared
 - memory address viewable to all threads
 - part of memory physically belonging to the local node is determined by affinity
- How Merrimac memory system supports?
 - Segmentation



Segmentation

- What segmentation supplies
 - Separate and protect memory spaces
 - Memory address interleaving
- Segments
 - Segments are disjoint each other
 - One stream belongs to one segment



UPC : Shared address space (cont.)

- Affinity

- scalar data : affinity with thread 0
- shared int `d[3][THREADS]`
 - `[THREADS]` = number of threads = 4
 - Can be distributed by segmentation
- shared `[3]` int `E[4][THREADS]`
 - `[3]` = layout qualifier
 - Cannot be distributed by segmentation

Thread 0	Thread 1	Thread 2	Thread 3
d[0][0]	d[0][1]	d[0][2]	d[0][3]
d[1][0]	d[1][1]	d[1][2]	d[1][3]
d[2][0]	d[2][1]	d[2][2]	d[2][3]

Thread 0	Thread 1	Thread 2	Thread 3
E[0][0]	E[0][3]	E[1][2]	E[2][1]
E[0][1]	E[1][0]	E[1][3]	E[2][2]
E[0][2]	E[1][1]	E[2][0]	E[2][3]
E[3][0]	E[3][3]		
E[3][1]			
E[3][2]			

UPC : Memory consistency model

- Memory consistency model
 - Formal specification of memory semantics
 - “how the memory system should behave with regard to ordering of reads and writes by different processors to different memory addresses”
 - Sequential consistency vs. relaxed consistency
- Sequential Consistency
 - Lamport ('71)
 - “The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program”
 - Makes hardware optimization harder
 - Between a write and a following read operation by write buffers
 - Overlapping write operations over network
 - Non-blocking read operations

UPC : Memory consistency model (cont.)

- Relaxed consistency model
 - Distinguished by the amount of ordering relaxation
 - Write to read, write to write, read to read, read to write to different locations
 - Different architectures/systems have different definition
 - MIPS : weak ordering
 - PowerPC
 - Alpha server
 - Cray machines
 - Safety nets exist to give further restriction to memory access ordering

UPC : Memory consistency model (cont.)

- Memory consistency model in UPC
 - How to order shared operations
 - Strict memory references
 - follow sequential consistency model
 - one paper claimed that strict consistency is weaker than sequential consistency.
 - it appears to 'all' threads that the strict references within the same thread appear in the program order
 - Relaxed memory references
 - follow local consistency model
 - it appears to the 'issuing' thread that all shared references within the thread appear in the program order
- How Merrimac memory system supports?
 - Relaxed consistency : determined by scalar processor selection
 - With safety net RC can be SC

UPC : Synchronization primitives

- What UPC provides
 - Split-phase barriers
 - Notify & Wait pairs
 - Inserting private tasks between notify and wait instructions might improve performance
 - Barrier – notify + wait in one instruction
 - Memory locks
 - Protecting shared data against multiple writes
 - `upc_lock`, `upc_lock_attempt`, `upc_unlock` over shared data
- How Merrimac memory system supports?
 - Barriers : dedicated barrier network or MCM + AMO
 - Locks : MCM + AMO

Cache coherency in Merrimac

- Cache coherency
 - How to avoid stale copies in multi-node system
 - Coherency defines what value is returned on a read
 - Consistency defines when it is available
 - Snooping & Directory based protocol
 - Two traditional approaches
 - Snooping : not scalable
 - Directory based : hard/expensive to implement especially in large scale parallel system
- Segment based cache coherency mechanism
 - One cache state per segment at one time
 - State transition requires proper cache actions like flush
 - Multiple copies on a memory address only if it is safe
 - Programmer/Compiler can set the cache state
 - potential for better performance

Segment cache states

- Segment can have one of following states
 - Not Cacheable (NC)
 - Locally Cacheable (LC)
 - Only stream cache can be used for the local memory references
 - Exclusively Cacheable (EC)
 - Programmer/compiler should guarantee no stale data in cache
 - Read Only Cacheable (ROC) : both stream/scalar cache
 - Scalar Locally Cacheable (SLC)
 - Only scalar processor access caches, cache contains local copies
 - Remotely Scatter-Add Cacheable (RSAC)
 - Enables combining partial scatter-add result in remote nodes
- Notes
 - LC also allows scatter-add operations
 - Remote caching

Necessary actions in cache state transition

- State transition
 - Should be protected by safety nets
 - Whole segment should be affected
 - Kind of actions
 - Flush
 - Invalidate (from ROC)
 - Flush remote data only (to LC)
 - Invalidate remote data only (to LC)
 - Combine remote data (from RSAC)

Cache state transition table

From To	NC	LC	EC	ROC	SLC	RSAC
NC		No Action	No Action	No Action	No Action	No Action
LC	Flush + Invalidate		No Action	Flush	No Action	No Action
EC	Flush + Invalidate	Flush + Invalidate remote data		Flush	Flush + Invalidate remote data	Flush + Invalidate remote data
ROC	Invalidate	Invalidate remote data	Invalidate remote data		Invalidate remote data	Invalidate remote data
SLC	Flush + Invalidate	No Action	No Action	Flush		No Action
RSAC	Combine remote, Flush + Invalidate local data	Combine remote data	Combine remote data	Combine remote, Flush local data	Combine remote data	

Memory system diagram

