**Merrimac Multi-node Memory Model**

**Merrimac Memory Architecture**

Merrimac has a distributed shared memory system where any node can reach any memory location in the system, but remote accesses have higher latency and lower bandwidth.

Merrimac has a relaxed memory consistency model with safety nets, hardware or software synchronization primitives and segmentation for memory protection and interleaving. Segment based cache coherence is proposed to extract higher memory bandwidth on multi-node systems with manageable hardware complexity.

**How Merrimac Supports UPC**

Memory consistency model (MCM)
- Formal specification of memory semantics
- Sequential consistency vs. relaxed consistency in general
- Merrimac support
  - Stream operations including memory accesses are issued by scalar processor,
  - Scalar processor supporting relaxed consistency with safety nets

Synchronization
- UPC has split-phase barrier and memory locks
- Merrimac support
  - Dedicated barrier network and/or atomic memory operations like fetch&op
  - Atomic memory operations will be semaphores for memory locking

Shared address space
- UPC has shared memory and private variables
- Affinity determines physical belonging of shared variables in memory
- Merrimac support
  - Allocating shared and physical variables into different segments
  - Affinity by address interleaving in segments

**Cache Coherency in Merrimac**

Purpose of stream and scalar cache
- Stream cache : bandwidth amplification
- Scalar cache : latency reduction

Cache coherency
- How to avoid stale copies in a multi-node system
  - Traditional approaches : snooping & directory based protocol

Segment based cache coherency mechanism
- Segment has four bits specifying
  - Stream-cache-memory-uncacheable
  - Stream-cache-network-uncacheable
  - Scalar-cache-memory-uncacheable
  - Scalar-cache-network-uncacheable

Each cache line has four bits specifying
- Stream cache valid bits helping segment based cache gang-flush and gang invalidate
- Compiler detects cachable streams and specifies options

Cacheable streams : streams whose access patterns have spatial and/or temporal locality – ex. vector in sparse-matrix-vector multiplication
- Streams used for many nodes can be cached in multiple nodes without coherence problem if the streams are read-only
- Stream processor can’t access scalar cache but scalar processor can access stream cache.

**Reliability Support**

Merrimac Fault Detection
- Even compute clusters shadow computation
  - Results compared when stored to SRF or transferred to other clusters
  - SRF protected with ECC
- All SRF in use – lanes are 2x wide
- Shadowing can be tuned off for increased productivity in small configuration
- ECC on buses and memories
- Replicated Control
  - Scalar processor
  - Microcontroller (microcode-store not replicated)
  - Stream controller (scalar interface)
- Address generators
  - Control parts of the memory and network interfaces

Merrimac Fault Detection
- ECC can mask single-bit faults in memories and buses
- Checkpoint all transient state every \( T_c \) time
- Upon unmasked fault
  - Diagnose failed component
  - Repair if diagnostic failed (hot swap)
- Rollback state
- Restart computation
- Expected slow-down <2%

**Summary of Multi-node Memory Model**

Distributed shared memory system (DSM)
- NUMA : non uniform memory access
- from a processing node, memory access to a different location might have different bandwidth and latency

Segmentation
- Address translation from virtual address to physical address
- One segment can be interleaved over multiple nodes
- Cache access policy changes over time on the unit of segment

Synchronization
- Hardware barrier or software implementation

**UPC Features**

- UPC (Unified Parallel C)
  - An explicit parallel extension of ANSI C considering DSM systems
  - Programmers are presented with single program multiple data execution model
  - private and shared memory space
  - each variable physically associated with a single processor
  - represents what programmers expect from the DSM systems

Extensions from ANSI C
- Explicitly parallel execution model
- Shared address space
- Synchronization primitives
- Memory consistency model

**Status:** October 2004