Modern Vector Systems
A Crash Course

James B. White III (Trey)
whitejbiii@ornl.gov

Boulder, CO
February 6, 2003

Acknowledgement

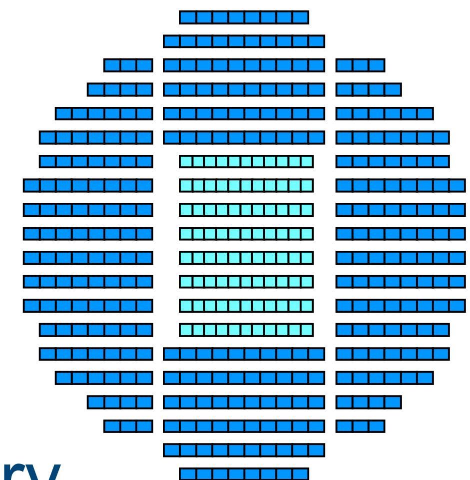
Research sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

Modern vector systems

- Earth Simulator
- Cray X1
- Performance implications

Earth Simulator (ES)

- World's biggest cluster
- 640 nodes
  - 5120 processors
  - 40 TF
  - 10 TB distributed memory
- Crossbar interconnect



ES node

- NEC SX-6
- 8 vector processors
  - 64 GF
- 16 GB of shared memory
  - 256 GB/s bandwidth

ES processor

- 500 MHz / 1 GHz
- 4-way super-scalar unit
  - 64 kB each data & instruction caches
- Vector unit
  - 8 GF
  - No cache
  - 32 GB/s memory bandwidth

ES vector unit

- 72 vector registers
- 256 values (64-bit) per register
- 8 vector pipes of 6 types
  - Types: add, multiply, divide, logical, mask, load/store
  - One instruction drives all 8 pipes
  - 256 values split among 8 pipes
  - Some pipe types work concurrently

ES interconnect

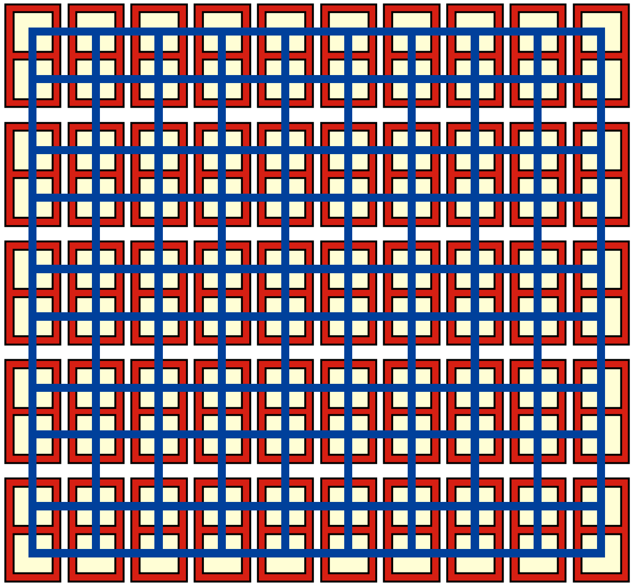
- Crossbar - full bisection bandwidth
  - No need for contiguous nodes
- 12.3 GB/s each way per node
- Hardware-enhanced data transfer
  - 3D subarrays
  - Scatter/gather

Modern vector systems

- Earth Simulator
- **Cray X1**
- Performance implications

Cray X1

- World's biggest single system?
- 4096 processors, one system image
- 50+ TF
- 4-way SMP nodes
- Globally addressable memory among nodes

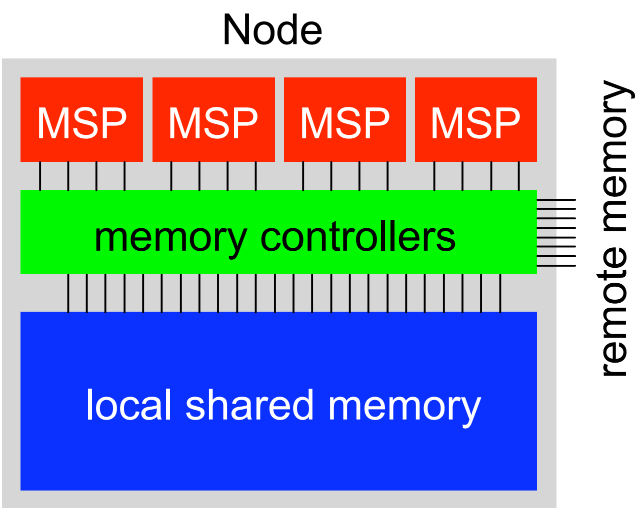


ORNL X1 plans

- 32 processors (MSPs) before April
  - 128 GB of memory
- 256 processors this FY
  - 1 TB of memory
  - 32 TB of local disk
- *Plenty of room for 4096 processors!*

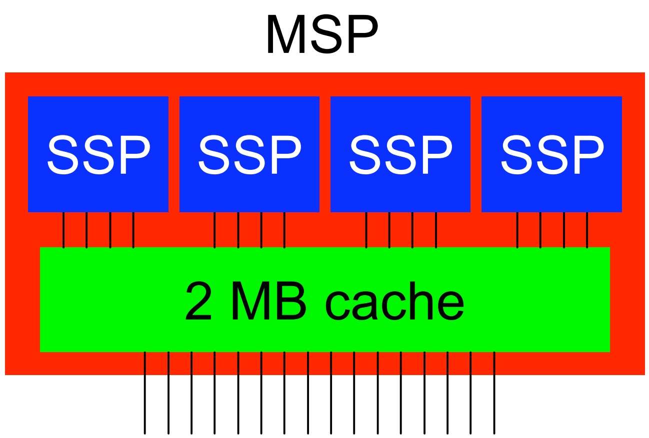
X1 node

- 4 multi-streaming processors (MSPs)
- 51 GF
- 16 GB shared memory (ORNL)
- 200 GB/s bandwidth
  - 150 GB/s to local MSPs
  - Extra for remote access



X1 MSP

- 4 single-streaming processors (SSPs)
- 12.8 GF (25.6 GF single precision)
- 2 MB shared cache
  - 51 GB/s load, 26 GB/s store
  - 4-word cache line
  - Extra bandwidth for scatter/gather

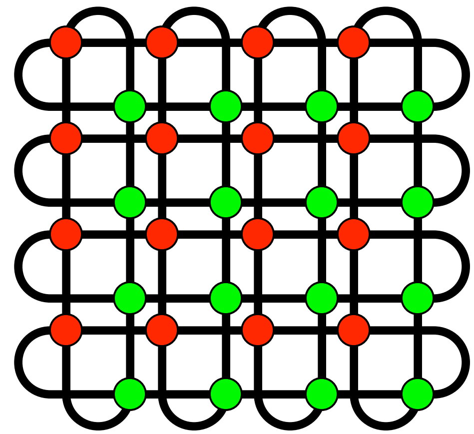


X1 SSP

- Decoupled scalar and vector units
- 400 MHz 2-way super-scalar unit
  - 16 kB each data & instruction caches
- 800 MHz 2-pipe vector unit
  - 32 registers, 64 values (64-bit) each
  - 3 Functional Unit Groups:
    Add, multiply, divide/`sqrt`

X1 interconnect

- 3D torus
- 12.8 GB/s/MSP
- Memory to memory
- Globally addressable memory
  - Load/store remote memory



Globally addressable memory

- Load/store memory on any node
- Remote address translation
  - On memory's node, not at processor
  - Avoids TLB misses
  - Requires contiguous processors
- Cache coherent
  - Only cache local memory

Parallel models

- Multistreaming within MSP
- OpenMP within node (late 2003)
- Between nodes (or processors)
  - MPI-1 two-sided message passing
  - MPI-2 one-sided communication
  - SHMEM one-sided communication
  - Co-Array Fortran remote memory

X1 libraries

- FFTs
  - Complex, real-complex, complex-real
  - Single-MSP 1D, multiple 1D, 2D, 3D
  - Parallel 2D, 3D
- Single-MSP BLAS, LAPACK, sparse direct solver
- Parallel BLACS, ScaLAPACK

Performance implications

- Optimization priorities
- Vectorization
- Multistreaming
- Communication

Optimization priorities

- Earth Simulator
  - Long vector loops (>10x)
  - Large-grain messages (2x)
- Cray X1
  - Moderate vector loops (10x)
  - Multistream (4x)
  - Low-latency communication (2x)
  - Cache blocking (<2x)

Performance implications

- Optimization priorities
- **Vectorization**
- Multistreaming
- Communication

Vectorization

- One vector instruction = many loop iterations
- Needs enough loop iterations
  - 256 for ES, 64 or 256 for X1
  - Fewer iterations = lower efficiency
- No procedure calls
- No loop-carried data dependencies
  - Some exceptions (reductions)

Vectorization: What compilers can do

- Array notation
- Scalar temporary variables
- Re-arrange loop nests
- Reductions, (un)pack, scatter/gather
- Fuse loops and array statements
- Inline procedures
- `if` statements within loops
  - Vector masks, some loss of efficiency

Vectorization: What compilers can’t do

- Make short vector loops efficient
- Make stride-1 (or -0) scatter/gather efficient (Cray X1)
- Know that index arrays don’t repeat
- Effectively inline many levels down

Vectorization: How you can help

- Assert that a loop is concurrent
  - `!dir$ concurrent`
  - Index vectors don't repeat
- Change array temporaries to scalar
  - Can remove dependencies
- Break up the big outer loop
  - To move it inside multiple inner loops
- Move loops inside procedure calls

Vectorization: Loopmark listings

- What vectorized, what didn't, and why?

```
679.                          ndayc = 0
680.  Vs------------<          do i=1,ncol
681.  Vs                          if (coszrs(i) > 0.0_r8) then
682.  Vs                             ndayc = ndayc + 1
683.  Vs                             idayc(ndayc) = i
684.  Vs                          end if
685.  Vs------------>          end do
```

```
ftn-6205 f90: VECTOR File = radcswmx.F90, Line = 680
  A loop starting at line 680 was vectorized with a single
   vector iteration.
```

What about C/C++?

- Same optimizer as Fortran
- Loopmark listings by Fall 2003
- `!dir$` becomes `#pragma _CRI`
- Standard Template Library

Performance implications

- Optimization priorities
- Vectorization
- **Multistreaming**
- Communication

Multistreaming

- Is an MSP one or four processors?
- One!
  - Fast synchronization
  - Shared cache
  - Can act like one 8-pipe processor
- Four!
  - Each SSP can operate independently
  - OpenMP-like directives

Multistreaming: What compilers can do

- Most vectorizable loops
- Most array syntax
- Nested loops with no dependencies
- Rearrange loop nests for vectorization within multistreaming
- Short loops

Multistreaming: What compilers can't do

- `pack/unpack` (not yet)
- Loops with:
  - Procedure calls
  - Dependencies
- Always choose the right loop to vectorize versus multistream

Multistreaming: How you can help

- Directives, directives, directives
  - `!dir$ concurrent`
  - `!dir$ preferstream`
  - `!dir$ ssp_private` (procedure calls)
  - Cray Streaming Directives (CSDs)
    - Much like OpenMP

Performance implications

- Optimization priorities
- Vectorization
- Multistreaming
- **Communication**

ES communication

- OpenMP within a node?
- MPI between nodes
- High bandwidth
- Moderate latency
  - Try to aggregate messages

X1 communication

- OpenMP within a node?
  - Late 2003
- MPI-1
- One-sided communication
  - For latency-sensitive operations

One-sided communication

- MPI-2 library
  - Complicated interface
  - Higher overhead?
- SHMEM library
  - Vendor specific
- Co-Array Fortran (and UPC)
  - Lowest latency
  - Vendor specific
- Intermix with each other and MPI-1

Optimization strategy (X1)

- Functional port
- Iterate
  - Loopmark and profile
  - Vectorize and multistream
- Tune communication

References

- “Overview of the Earth Simulator.” Earth Simulator exhibit, SC2002.
- James L. Schwarzmeier. “Cray X1 Architecture and Hardware Overview.” ORNL Cray X1 Tutorial, November 2002.
- Nathan Wichmann. “Coding for Performance on the Cray X1.” ORNL Cray X1 Tutorial, November 2002.
- John M. Levesque. “Message-Passing Paradigms.” ORNL Cray X1 Tutorial, November 2002.

Some CCSM challenges

- CAM vectorization
  - Day/night column physics
  - Cloud radiation
- CLM vectorization
- Multistreaming strategy

Vectorization: Day/night

- Issue
  - Day/night computation disparity
  - Half of columns per chunk are day
- Strategies
  - Index array (current CAM)
    - Stride-1 scatter/gather - bad on X1?
  - Vector mask (<50% efficiency)
  - Sort columns within a chunk - stride 1

Vectorization: Cloud-radiation algorithm

- Find maximum-overlap regions
  - Variable number of regions per column
  - Variable height & thickness per region
- Sort cloud levels within a region
  - Variable-length sorts
  - Mostly very short, 1-4 elements
  - Some beyond 20 elements

Cloud vectorization strategy?

- Create index array for all regions in a chunk of columns
- Do first few sort iterations on the region vector
  - Mask out sorted regions
- Create new index array and iterate
- Finish long sorts with scalar code

CLM vectorization issues

- Tree data structures
- Depth-first tree traversal
  - Good data locality
  - Shortest possible inner loops
  - Deepest possible loop nests

CLM vectorization strategy?

- Tree structures point into flat arrays
  - Minor code modifications
- Create new index arrays
  - *e.g.* for each PFT
- Rewrite critical loop nests
  - Perform full-breadth loops
  - Use index arrays

Multistreaming: CCSM Strategy

- Augment vectorization?
  - Also works well for ES
- Mimic OpenMP using CSDs?
  - Easy code modifications
  - Reduce chunk size by 4
  - Potential for better scalability

Questions?
Answers?