Cray X1 Overview

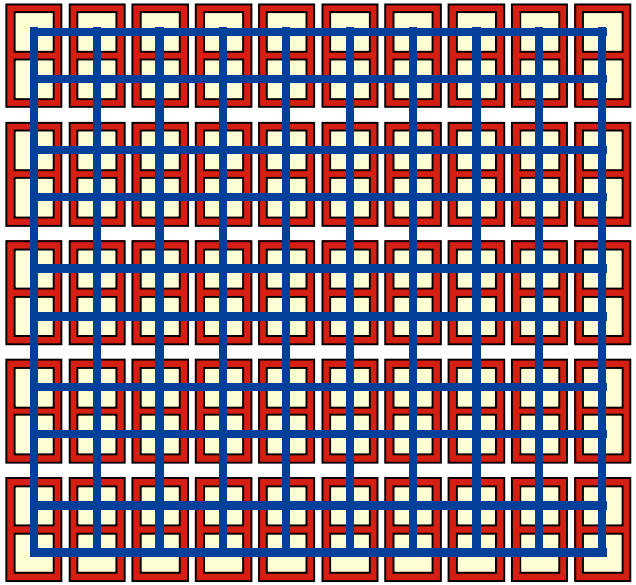
Computational Chemistry Workshop

James B. White III (Trey)

trey@ornl.gov

Cray X1

- Up to 4096 processors
- One system image
- 50+ TF
- 4-way SMP nodes
- Globally addressable memory among nodes

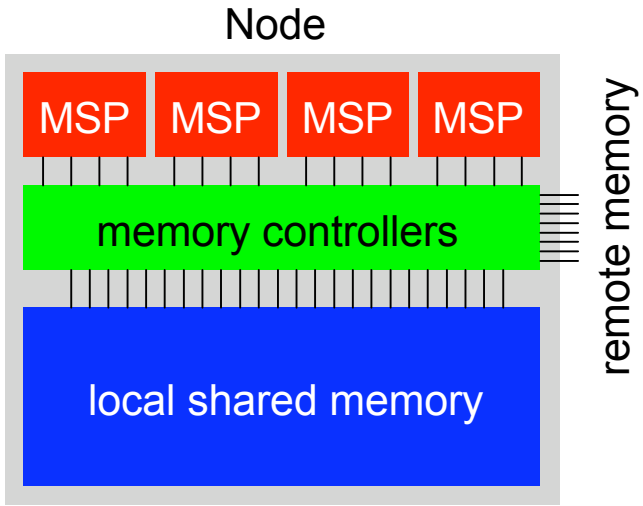


Phoenix

- Soon 512 processors (MSPs)
  - 2 TB of memory
- Upgrade in winter 2005
  - 1024 X1e processors
  - Same 2 TB of memory

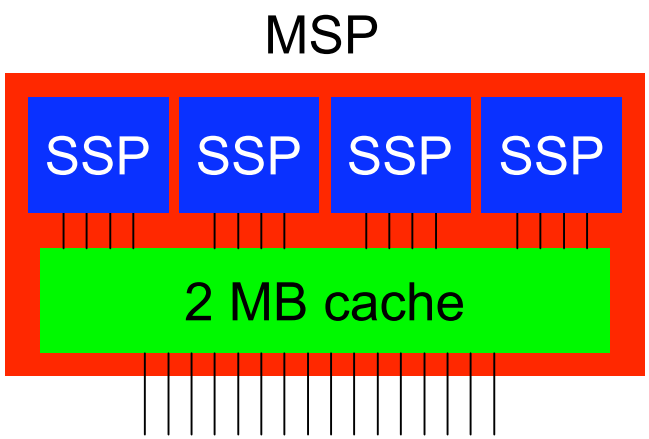
X1 node

- 4 multi-streaming processors (MSPs)
- 51 GF
- 16 GB shared memory (ORNL)
- 84 GB/s STREAM bandwidth
- Additional bandwidth available 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

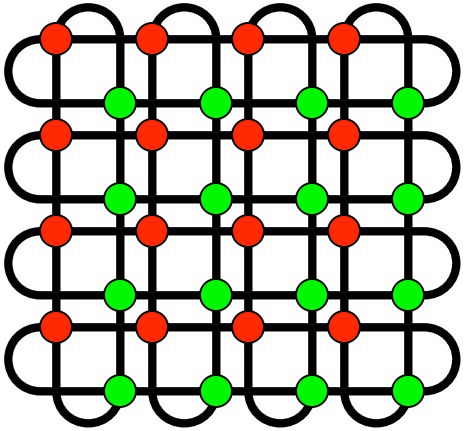


X1 SSP

- 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

- Similar to 3D torus
  - Not full bisection bandwidth
- 12 GB/s measured MPI bandwidth
- Memory to memory
- Globally addressable 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
- Between nodes (or processors)
  - MPI-1 two-sided message passing
  - MPI-2 one-sided communication
  - SHMEM one-sided communication
  - Co-Array Fortran remote memory
  - Direct load/store using pointers

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

- Moderate vector loops (10x)
- Multistream (4x)
- Low-latency communication (2x)
- Register blocking (<2x)
- Cache blocking?

Performance implications

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

Vectorization

- One vector instruction = many loop iterations
- Needs enough loop iterations
  - 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 (one level down)
- `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
- Know that index arrays don't repeat
  -
```
do j = 1, n
   x(i(j)) = x(i(j)) + …
```
- Effectively inline many levels down

Vectorization: How you can help

- Assert that a loop is concurrent
  - `!dir$ concurrent`
  - Index arrays 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
- `!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**

X1 communication

- OpenMP within a node
- 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

Cray X1 Overview

Computational Chemistry Workshop

James B. White III (Trey)

trey@ornl.gov