Parallel Programming Models Update

Greg Fischer, Charlie Carroll
Programming Environment

Compiler-based Models

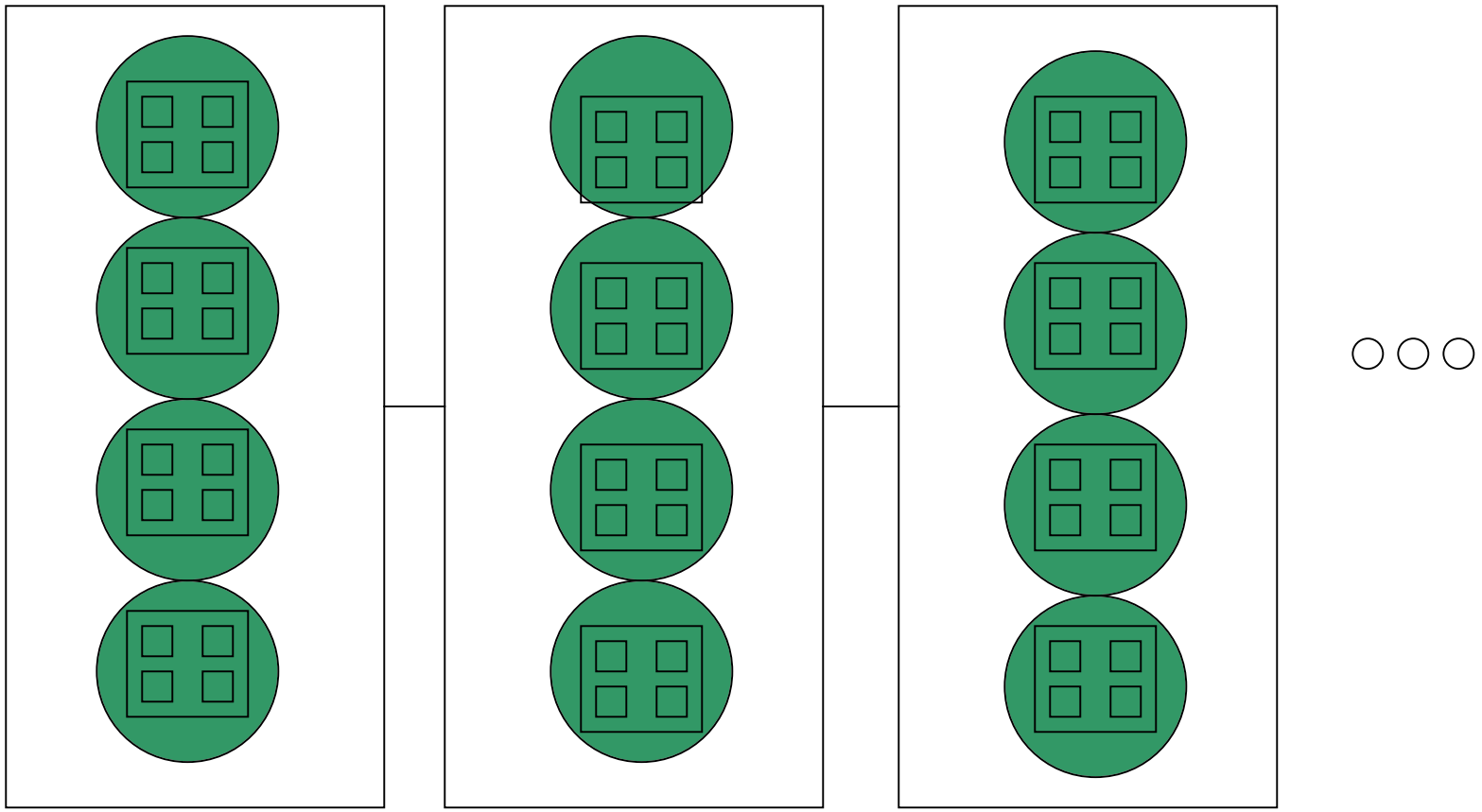
- Co-array Fortran
- UPC
- OpenMP

Co-array Fortran, UPC – Primary Features

- http://www.co-array.org/, http://upc.gwu.edu/

- Distributed Memory (NUMA)
- Multiple Instantiations of Program (SPMD)
- Communication through shared, distributed arrays
- Syntax-based
- Improved performance
- Improved programmer productivity

Distributed Memory Execution



Co-array Fortran – Status

- Currently Available on X1: T3E-Equivalent
- PE 5.1, 5.2: No additional features planned

UPC - Status

- Currently Available on X1: Subset UPC
- PE 5.1: Full UPC
  - All Shared Types
  - Pointers to all shared types
  - upc_forall
  - upc_global_alloc, upc_free
  - Other UPC v1.1 intrinsics
  - "Unoptimized" Caution Messages

OpenMP - Primary Features

- http://www.openmp.org/

- Shared Memory (UMA) Model
- Single Instantiation of Program
- Multiple Threads
- Directive-based
- Work Distribution of Loops to Threads

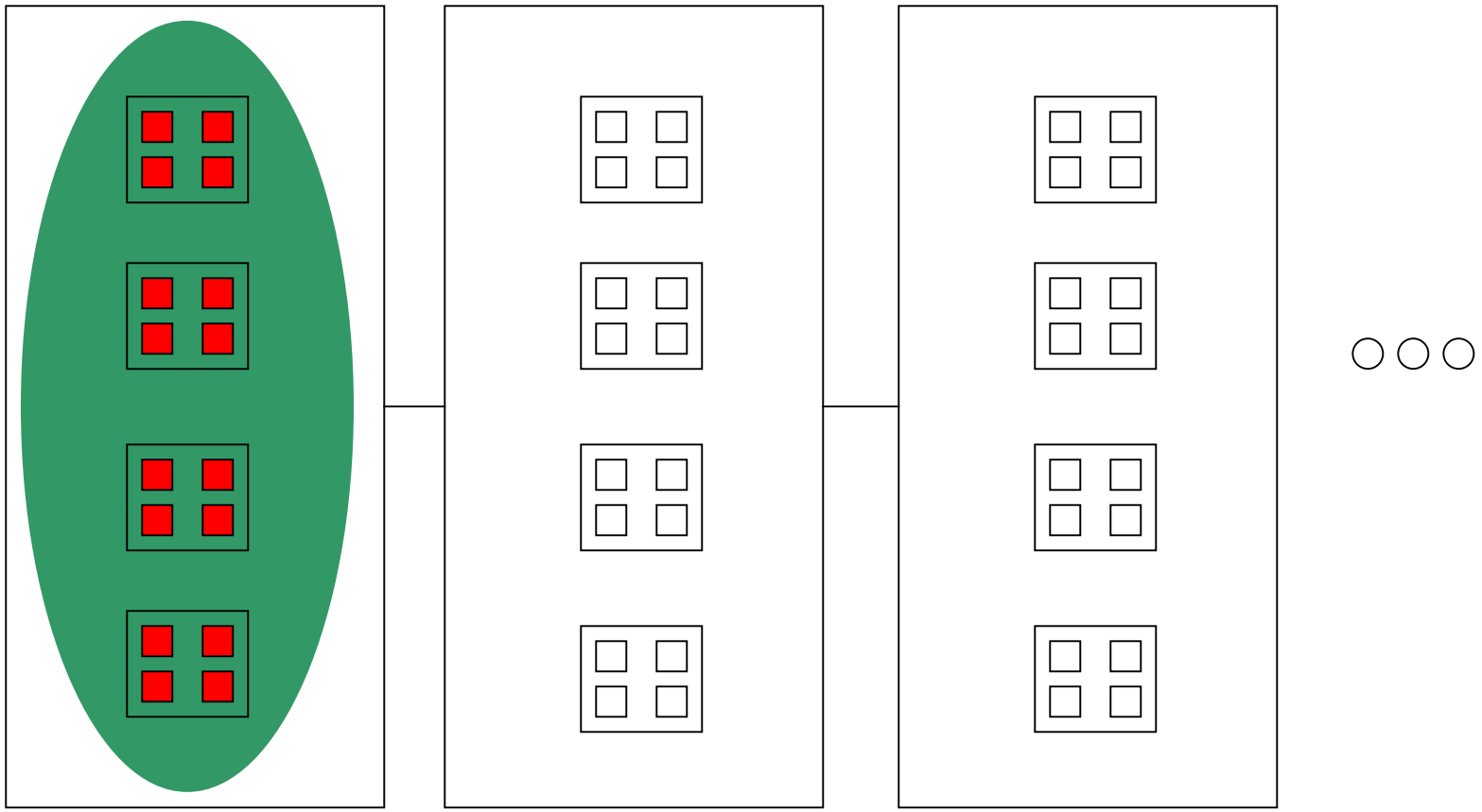
OpenMP - Status

- First Available in PE 5.1
- OpenMP 2.0 Compliance (including WORKSHARE)
- Fortran, C/C++

OpenMP - Node Programming

- Pthread Implementation
  - Single UNICOS Process
  - Single X1 Node
- Shared, UMA Memory Model
- Largest X1 UMA Component: Node
- Full Node Power to Single-image Applications

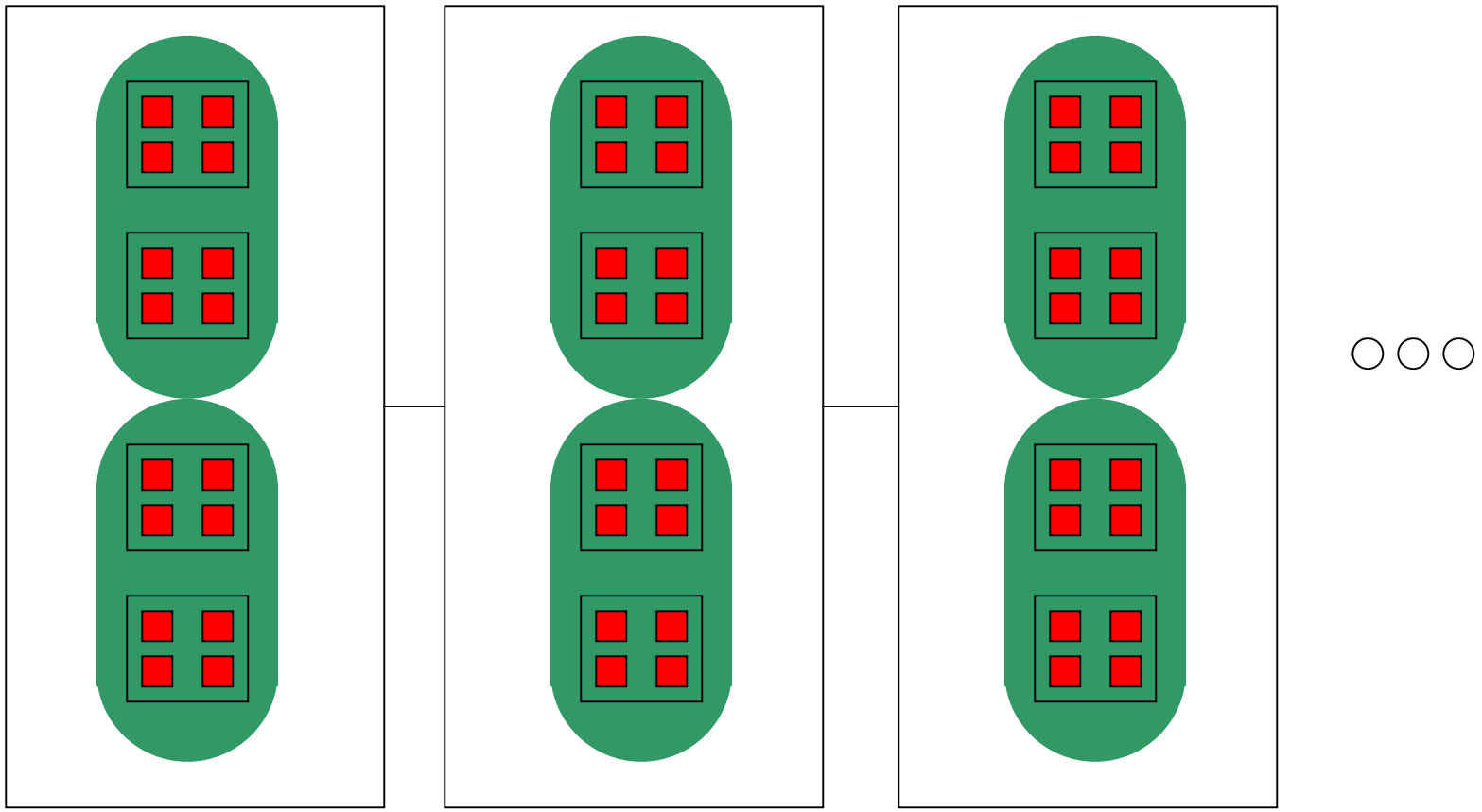
OpenMP – Node Execution



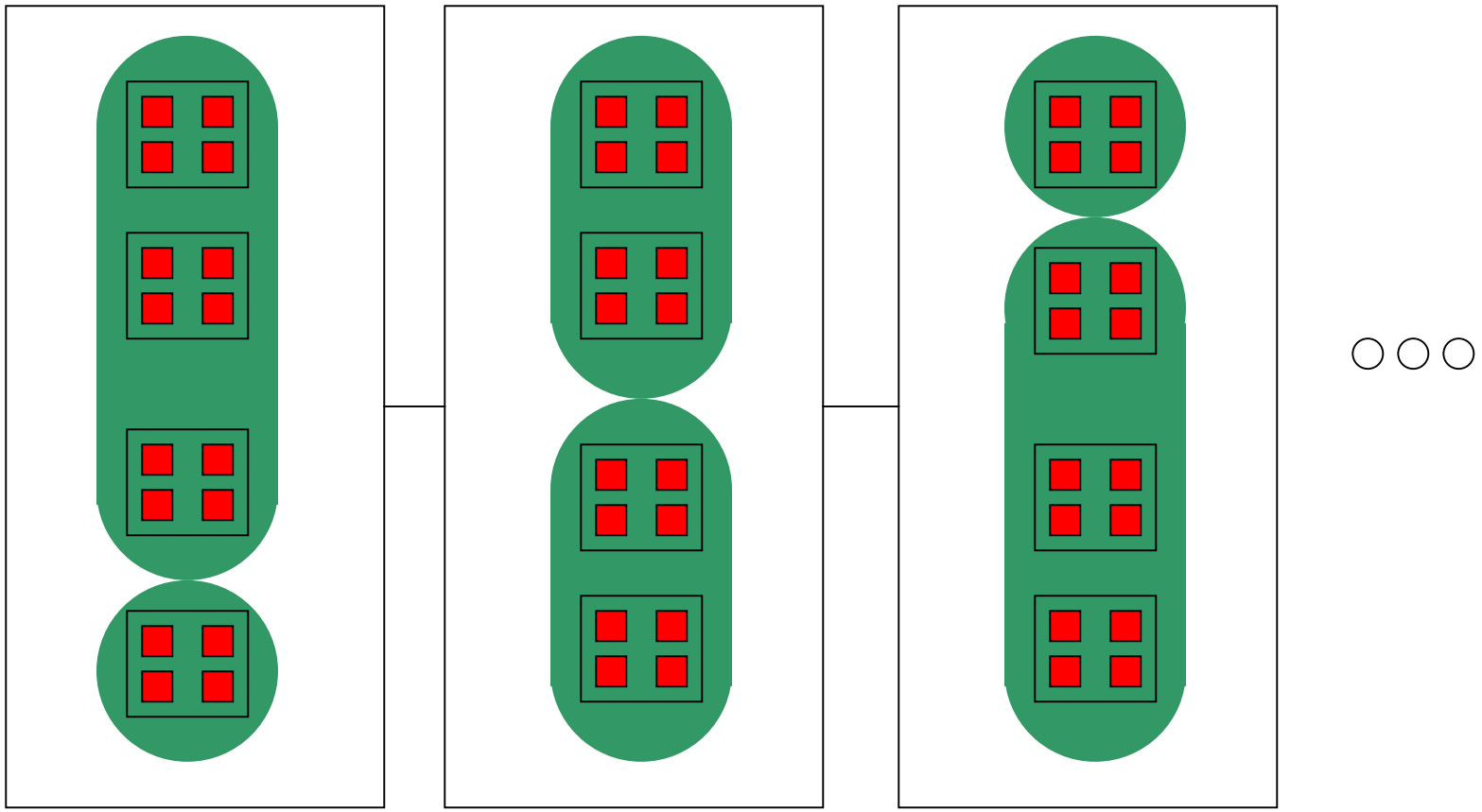
OpenMP - Hybrid Programming

- Distributed Memory and OpenMP
- OpenMP applied to PE
  - Multi-level Parallelism
  - Heterogeneous-sized PEs for Irregular Grid Applications

Hybrid Execution



Hybrid Execution - Heterogeneous



OpenMP - X1 Resource Scheduling

- aprun -d <integer>
- guarantees gang-scheduled processors
- guarantees performance
- oversubscription still possible

OpenMP - X1 SSP Mode

- available in PE 5.0
- allows SSPs to be "processors"
- will map to SSP:
  - MPI process
  - UPC Thread
  - Co-array Fortran Image
  - OpenMP Thread

OpenMP - MSP mode vs. SSP mode

- X1 E-cache
  - Shared cache between SSPs
  - Two-ways
  - Key: avoid conflict between SSPs
  - Solution: parallel loop distribution
    - Multi-streaming
    - OpenMP “DO” or “for” directives

OpenMP - MSP mode vs. SSP mode

- Fine-grain Parallelism
  - AKA Loop-level Parallelism
  - Work distribution via loop distribution with OpenMP “DO” or “for” directive
  - Use SSP mode

OpenMP - MSP mode vs. SSP mode

- Coarse-grain Parallelism
  - AKA Functional Parallelism
  - Parallel Regions Contain Calls to Sizable Subroutine
  - Works on Threadprivate Data (no loop distribution)
  - Use MSP mode, multi-streaming

OpenMP Performance

- Disclaimer
- Overhead measurements
- X1 compared to SV1
- Two kernels from NAS Parallel Benchmarks

Disclaimer

- OpenMP is not ready to ship
- We've collected some preliminary data
- Performance is not (yet) our highest priority
- But it will be soon
- The numbers will certainly change

Overhead

- Methodology from J.M. Bull

  at University of Edinburgh (URL in paper)
- Overhead = $T_P - T_S / p$
  - T_S is time to execute on a single thread
  - P is the number of threads
  - T_P is time to execute on p threads
  - Example: 100s of work spread to 4 threads should take 25s; extra time is overhead

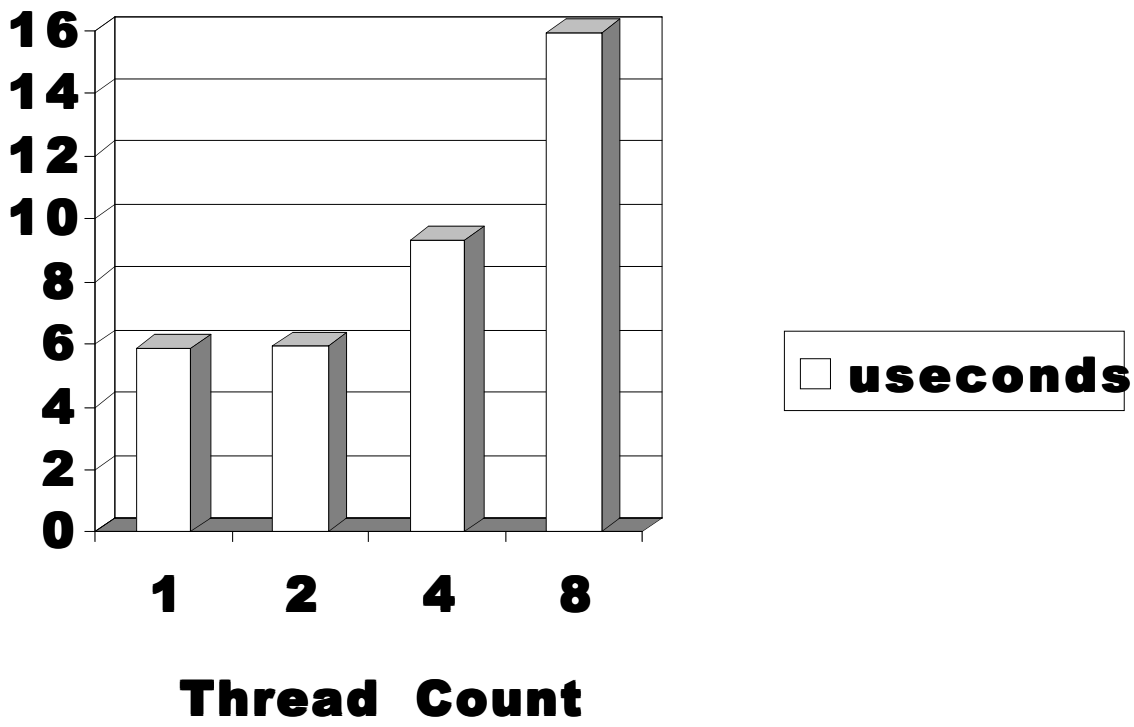
Reference Code


```
do j=1,numiter
  call delay(delaylength)
end do
```

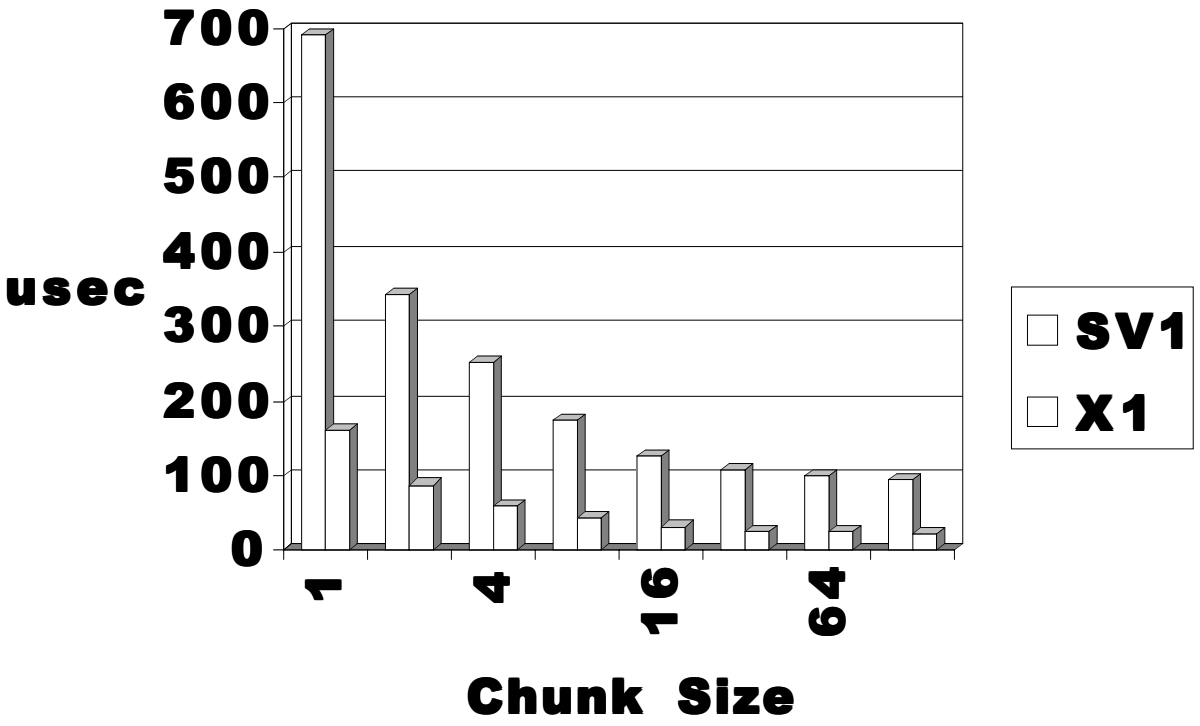
Test Code


```
!$OMP PARALLEL
  do j=1,numiter
!$OMP DO
    do I=1,omp_get_num_threads()
      call delay(delaylength)
    end do
!$OMP END DO
  end do
!$OMP END PARALLEL
```

Parallel Do Overhead



Dynamic Scheduling Overhead



NAS Parallel Benchmark CG Kernel

Thread Count	Speedup
1	1.0
2	2.0x
4	3.9x
8	7.9x
16	15.5x

MSP mode ran in 70% of 4-thread time

MSP mode = 5.6 OpenMP threads

NAS Parallel Benchmark MG Kernel

Thread Count	Speedup
1	1.0
2	1.9x
4	3.1x
8	7.0x
16	12.3x

MSP and 4-thread times nearly identical

Performance Conclusions

- It's early; much work still to do
- Numbers will change, mostly to the good
- X1 compares well to SV1
- OpenMP scales well on good codes