HPF Features for Locality Control on CC-NUMA Architectures

Barbara Chapman
University of Houston

Contents

- CC-NUMA Architectures
- Language Extensions for CC-NUMA Systems
- Programming for Scalable Performance
- Summary

CC-NUMA Platforms

- Emulate true shared memory systems
  - globally addressable memory
  - hardware support for cache consistency
- Increasingly built and deployed
  - HP, SGI, Compaq, Sun…
- Increasing size of individual systems
  - 1024 processor SGI Origin 3000 soon to be delivered

New AlphaServer GS System

- CC-NUMA machine built from 4-processor building blocks ("quads") interconnected with a fast switch that delivers 1.6GB/s in + 1.6GB/s out = 3.2GB/s total per quad, with remote latency less than 3:1 even under heavy load!
- Each quad is a UMA SMP, with 4*1.6 = 6.4GB/s total bandwidth between processors and memory
- Processors: Up to 32 Alpha EV67@ 729Mhz (initially)
    - Dual floating point pipelines; quad integer pipelines

COMPAQ
Better answers

CC-NUMA Programming Issues

- Memory hierarchy
  - cache, local and remote
- Performance impact
  - keep data in cache
  - penalties for true and false sharing of cache lines
  - network contention

CC-NUMA Programming

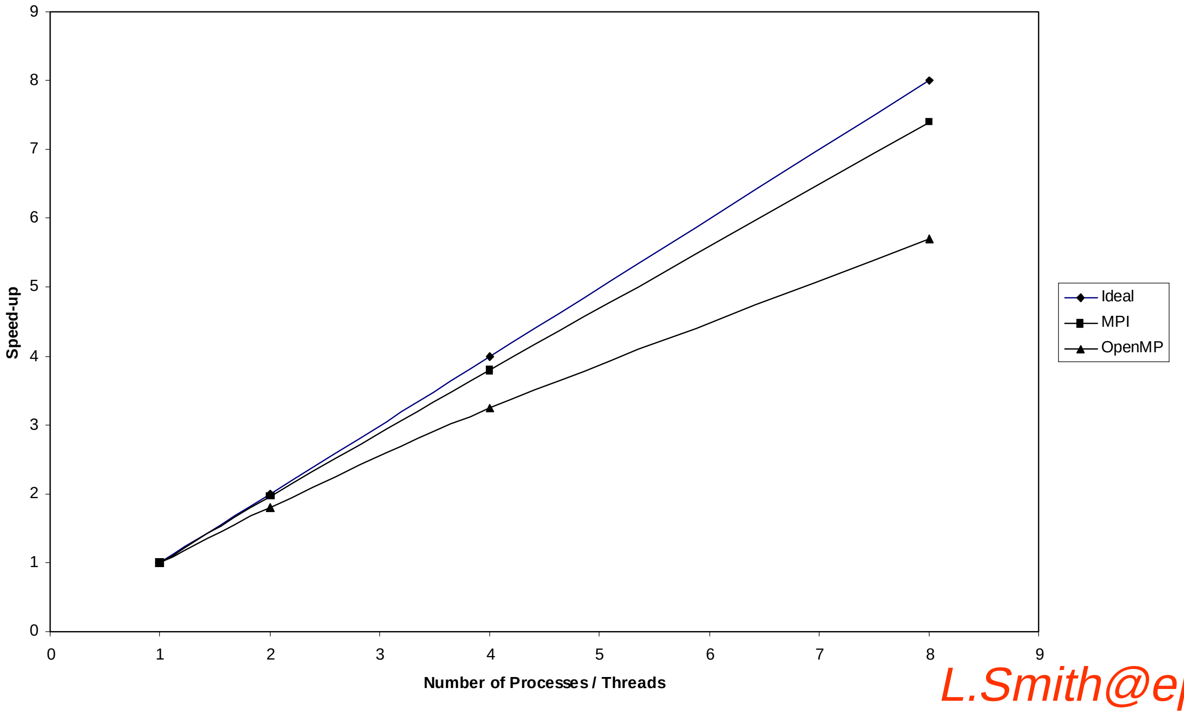
- A variety of programming models used
  - MPI
  - OpenMP
  - HPF
  - MPI and OpenMP, HPF and OpenMP
- But shared memory is what they emulate

OpenMP

- OpenMP de facto standard for shared memory work distribution
  – available for Fortran, C and C++
- OpenMP application development
  – easy, fast, incremental
  – code maintenance benefits
  – … but optimization is hard

Fast OpenMP Parallelization

- QMC on SGI Origin 2000, 40 195MHz R10000 processors
  - Access to 8 processors



L.Smith@epcc.ed.ac.uk

OpenMP on CC-NUMAs

- No features to support CC-NUMA
- Vendors acknowledge need for data locality control at node level
  - first touch allocation policy
  - automatic page migration
  - page-based mappings
  - HPF-style element mappings
  - association of work with location of data

SGI OpenMP CC-NUMA Extensions

- Allocate *cache pages* to memory on nodes
  - DISTRIBUTE, ONTO, DYNAMIC, page_place
  - inaccurate, but preserves illusion of true shared memory
- Allocate *data* to processors in HPF style
  - DISTRIBUTE_RESHAPE, ONTO, query intrinsics
  - accurate, but destroys illusion of shared memory
  - translates references to (processor, offset)
- Assign loop iterations to thread
  - AFFINITY (like ON HOME), NEST

User-Directed Page Migration

- Two new directives:

```
!dec$ omp migrate_next_touch(<variable-
list>)
!dec$ omp place_next_touch  (<variable-
list>)
```

- migrate_next_touch marks pages containing **any part** of a variable in the list for migration to the quad of the thread that next touches the page.
- place_next_touch marks pages containing **only** data belonging to a variable in the list for migration to the quad of the thread that next touches the page; <u>the contents of the page(s) are discarded</u>.

COMPAQ Better answers

Extensions to Compaq Fortran OpenMP Language

- Add data, computation layout directives to specify:
  - On which quad data is placed
  - On which quad a loop iteration is placed
- Add "NUMA" directive to control computation placement:

```
!dec$ omp numa
!$omp parallel do
```

- The NUMA directive modifies the following PARALLEL DO to schedule iterations based on layout and usage of data in loop

COMPAQ Better answers

LU Example With Data Layout

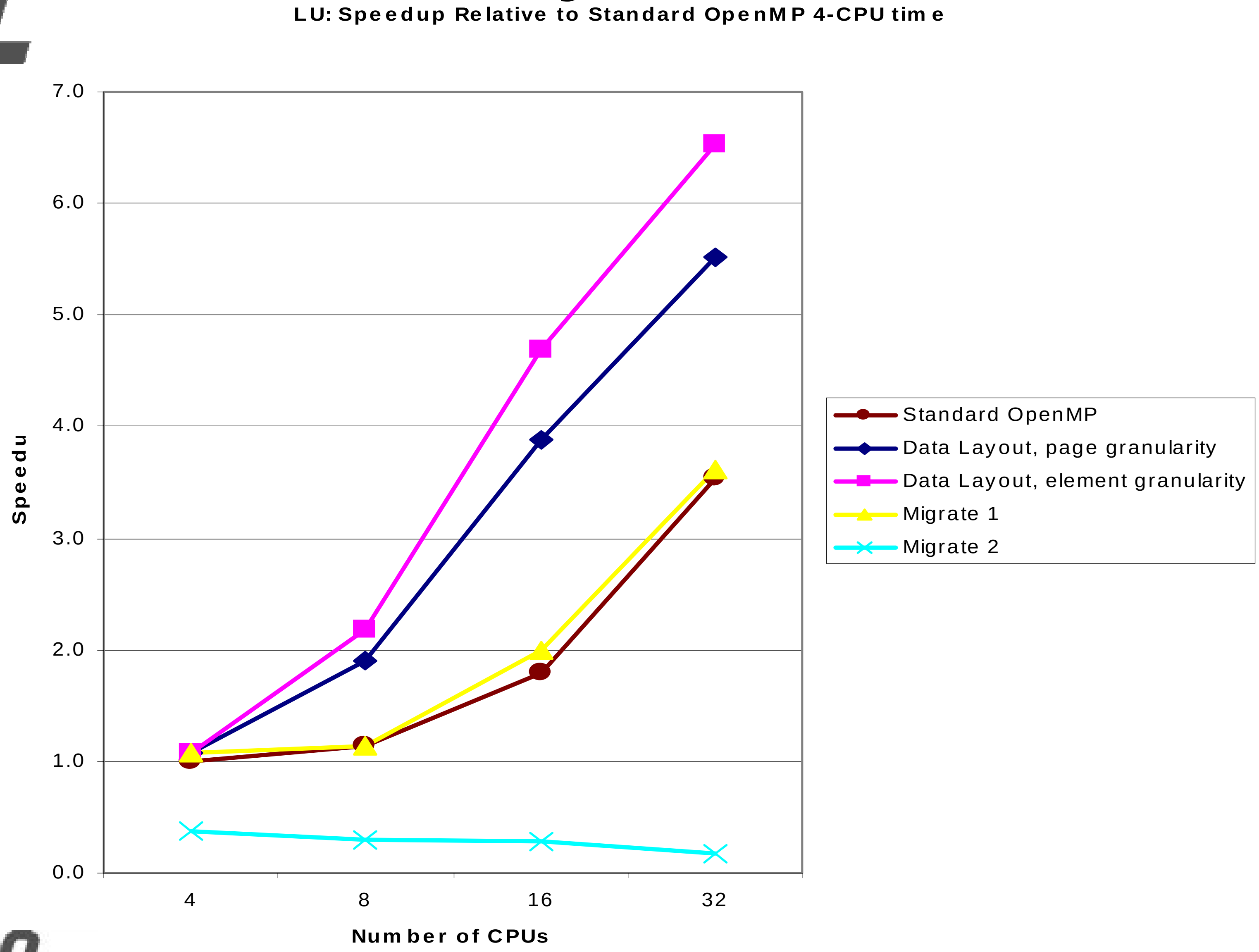
```
integer, parameter              :: n=1024
real(kind=8)                    :: a(n,n)
!dec$ distribute (*,cyclic) :: a(n,n)
. . .

do k=1,n-1
   do m = k+1, n
     a(m,k) = a(m,k) / a(k,k)
 end do

 !dec$ omp numa
 !$omp parallel do private(i)
 do j = k+1, n
    do i = k+1, n
         a(i,j) = a(i,j) - a(i,k) *a(k,j)
    end do
 end do
end do
```

Better answers

Preliminary Results with LU



COMPAQ
Better answers

Data Layout Directive Summary

- Data and computation placement directives:
  - DISTRIBUTE, REDISTRIBUTE
  - ALIGN
  - ON
  - TEMPLATE
  - MEMORIES*
  - [NO]SEQUENCE
- Can do complex layouts, including blocked [by chunks], round-robin [by chunks], partial replication, full replication

Directives taken from High Performance Fortran, which carefully figured out how to make them work with Fortran 90/95 features

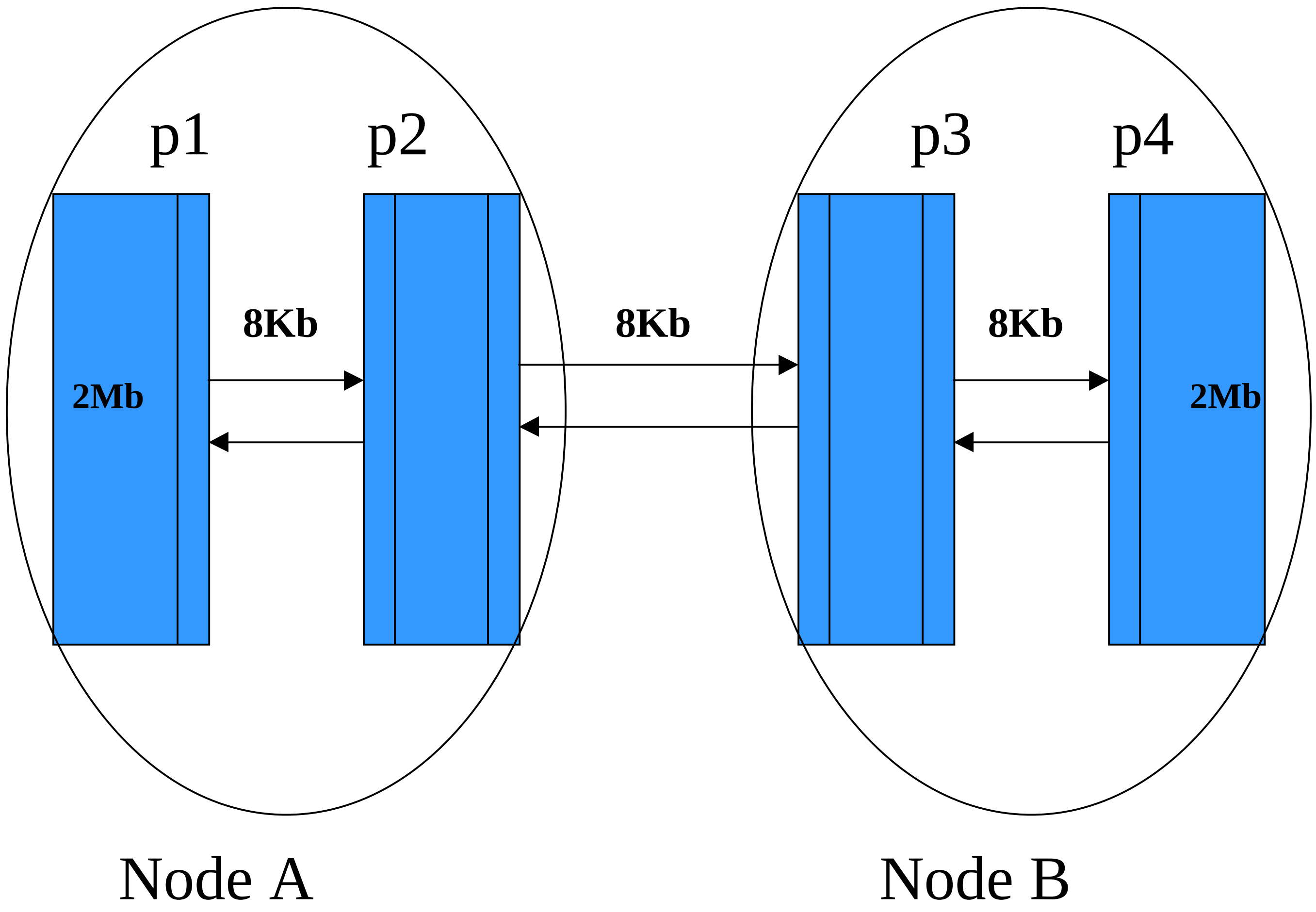
*MEMORIES equivalent to HPF's PROCESSORS directive

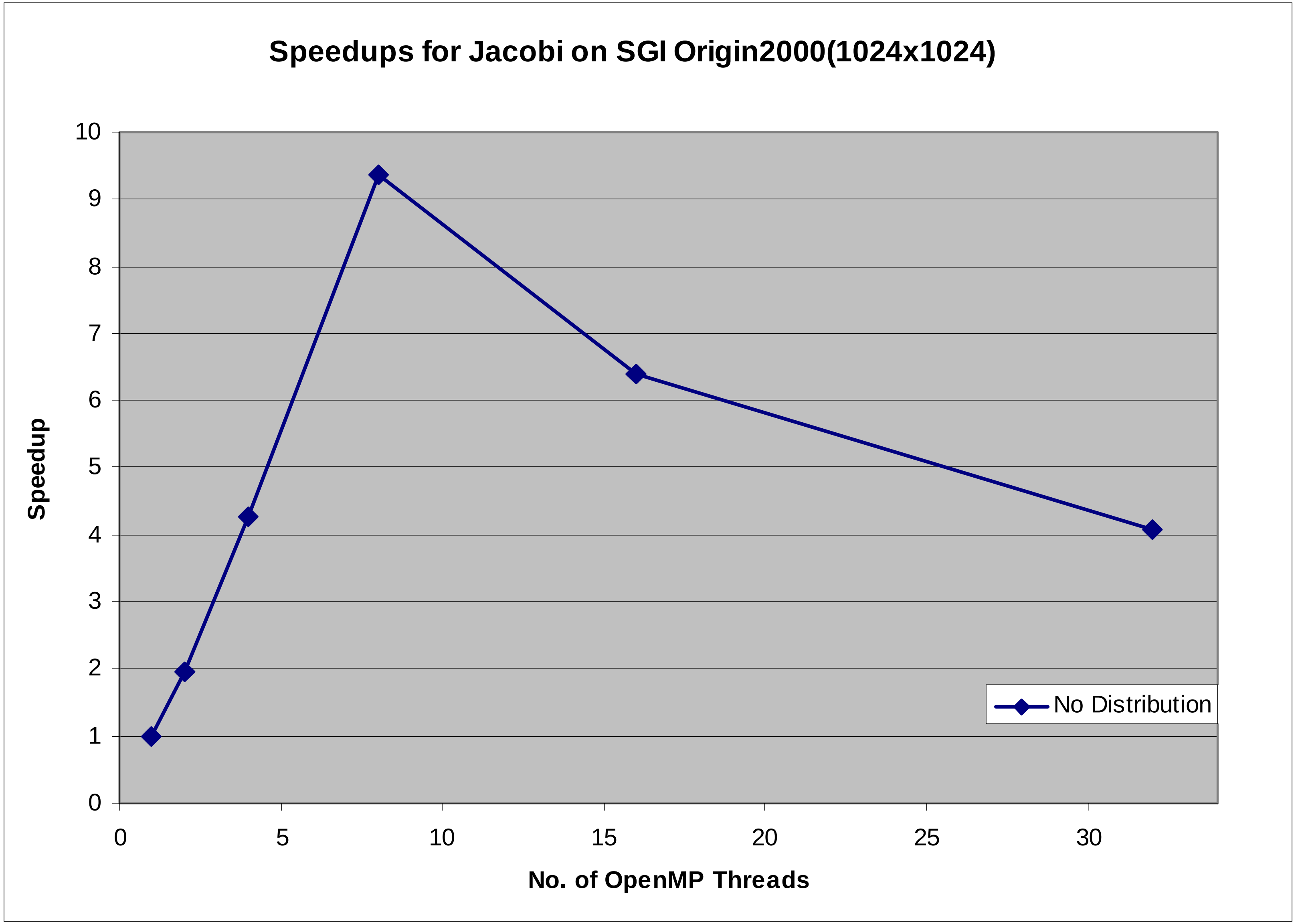
COMPAQ Better answers

OpenMP Jacobi on Origin

```
!$OMP Parallel Shared ( b, a, sum )
 ………..
!$OMP DO
do j = 1, n
 do i = 1, n
   a (i,j) = ( b(i-1,j) + b(i+1,j) + b(i,j-1) + b(i,j+1) ) * 0.25
 enddo
enddo
```

- First touch data allocation distributes second dimension of a, b in BLOCK fashion

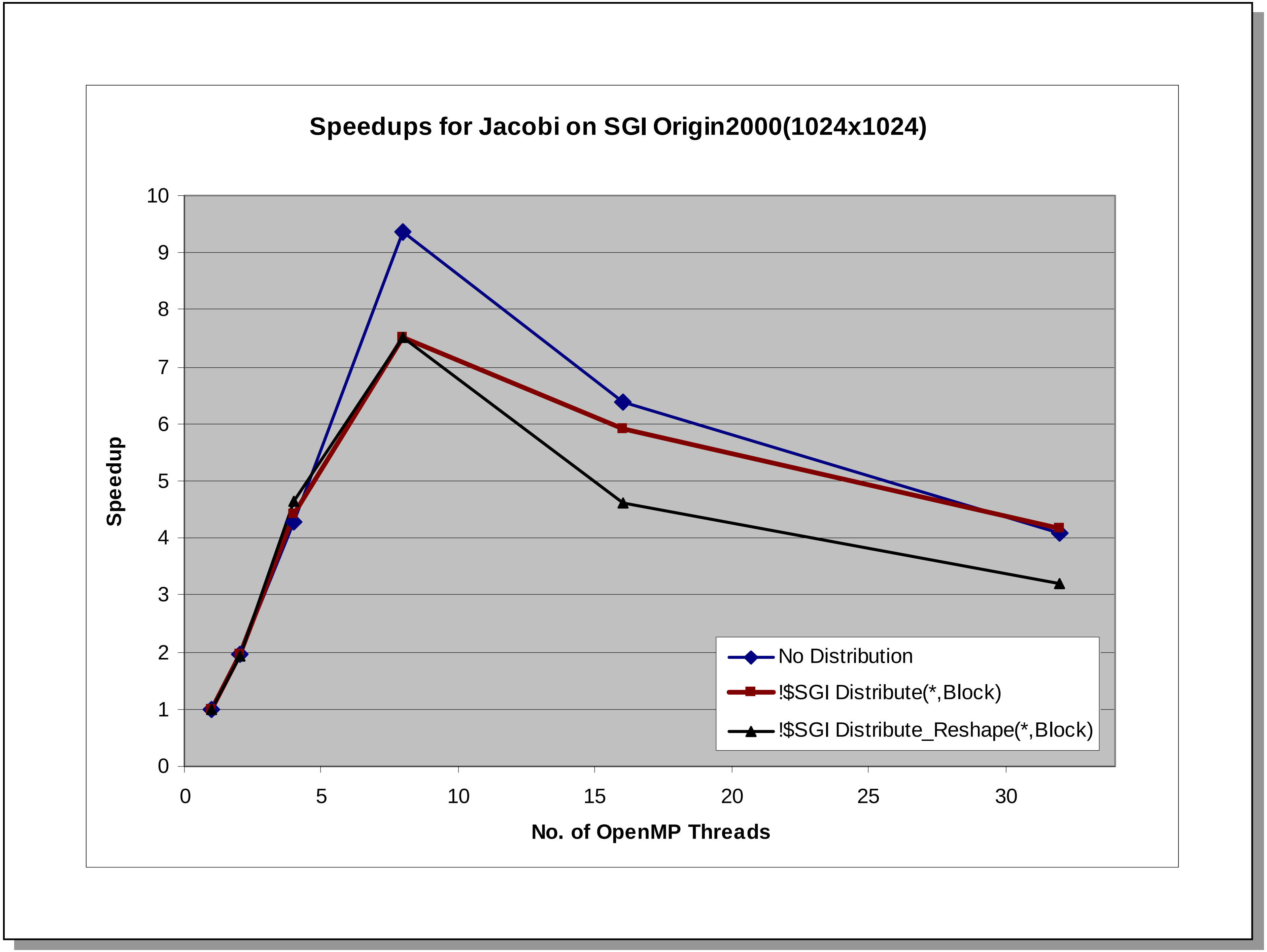

p1
p2
p3
p4
8Kb
8Kb
8Kb
2Mb
2Mb
Node A
Node B


Speedups for Jacobi on SGI Origin2000(1024x1024)
Speedup
10
9
8
7
6
5
4
3
2
1
0
0
5
10
15
20
25
30
No. of OpenMP Threads
No Distribution

OpenMP Jacobi on Origin

```
!$SGI DISTRIBUTE_RESHAPE b(*,block), a(*,block)
!$OMP PARALLEL SHARED ( b, a, sum )
 ………..
!$OMP DO
do j = 2, n
 do i = 1, n
  a (i,j) = b(i-1,j) + …
 enddo
enddo
```

- Data is mapped explicitly to processors
- This is the same mapping as first touch

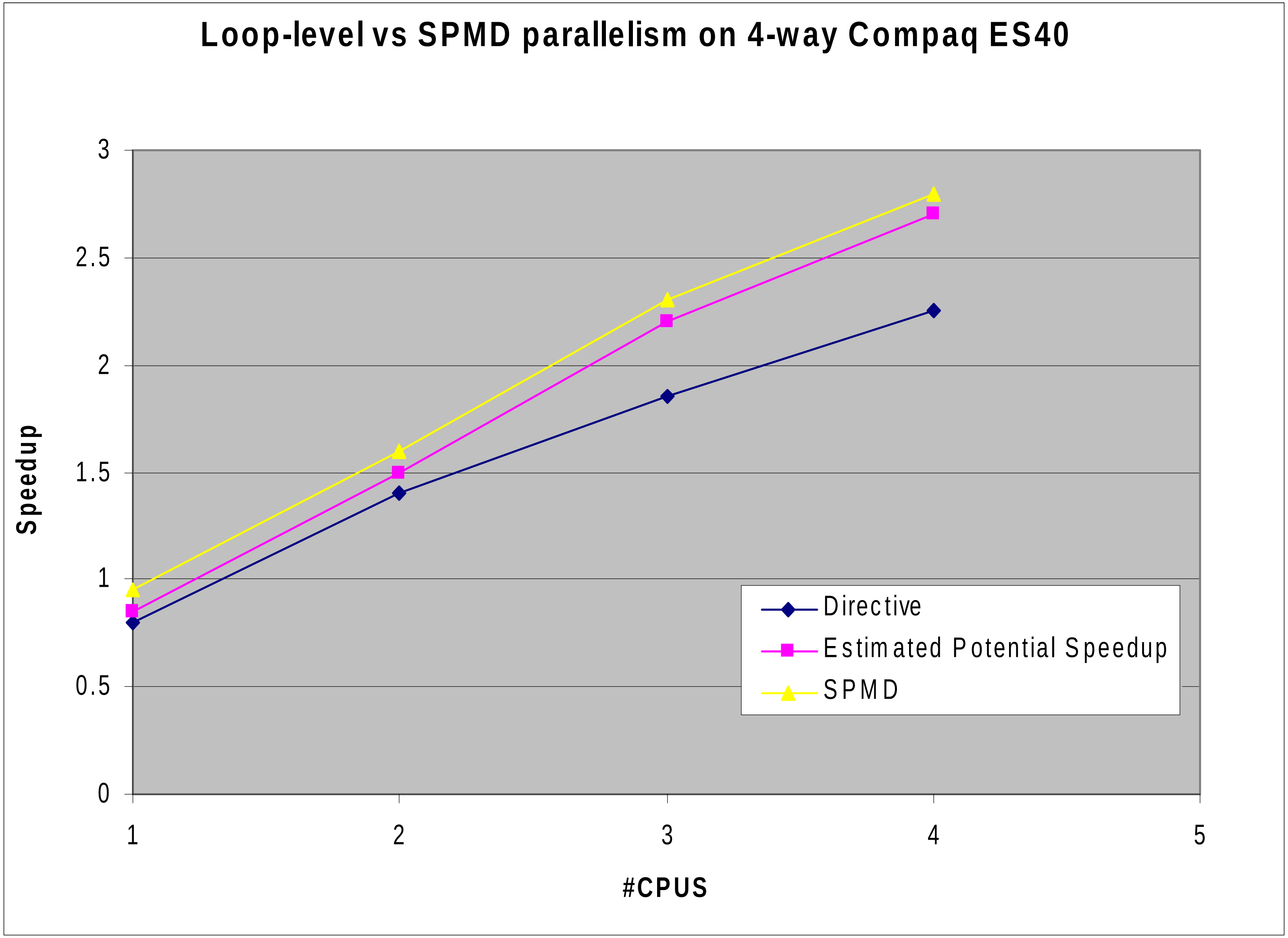

Speedups for Jacobi on SGI Origin2000(1024x1024)
Speedup
No. of OpenMP Threads
0
1
2
3
4
5
6
7
8
9
10
0
5
10
15
20
25
30
No Distribution
!$SGI Distribute(*,Block)
!$SGI Distribute_Reshape(*,Block)

Improving Scalability

- Minimize number of variables accessed by more than 1 processor
- Separate frequently updated variables from others
- Aggregate related frequently updated variables

OpenMP SPMD Parallelization

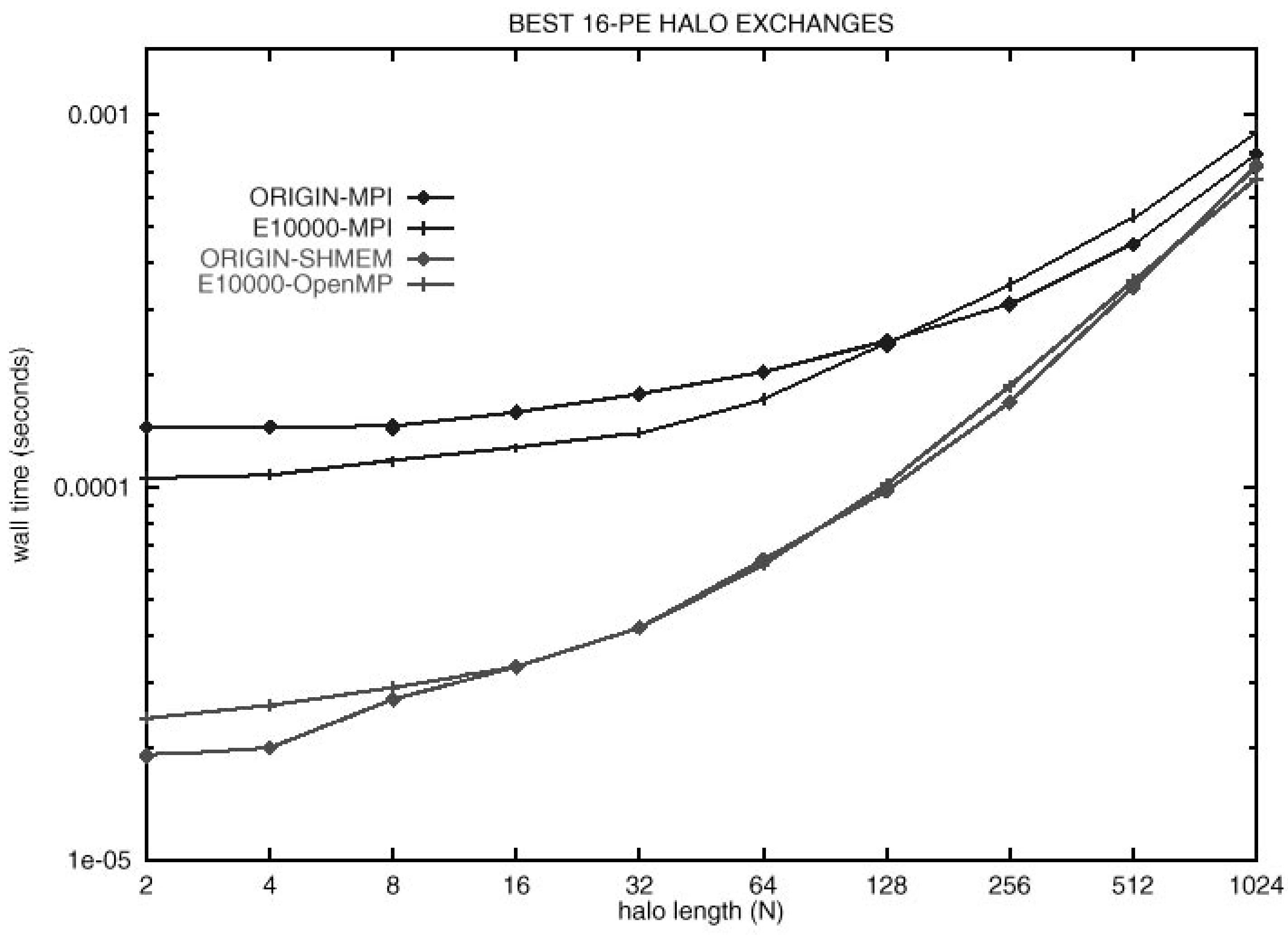
- Distribute arrays among threads, privatize
- Create buffers to store data shared between two or more threads
- Copy data to and from buffers as needed
- Insert necessary synchronization



Loop-level vs SPMD parallelism on 4-way Compaq ES40
Speedup
3
2.5
2
1.5
1
0.5
0
1
2
3
4
5
#CPUS
Directive
Estimated Potential Speedup
SPMD

SPMD Programming Style

- NLOM, NCOM Ocean Models
  - several parallel versions developed at Naval Research Lab
- Developed HALO benchmark to compare OpenMP and MPI on range of architectures
  - OpenMP significantly outperformed MPI
- OpenMP code is now preferred version
  - *scales close to linearly* up to 112 nodes on Origin 2000
  - MPI to 28 nodes

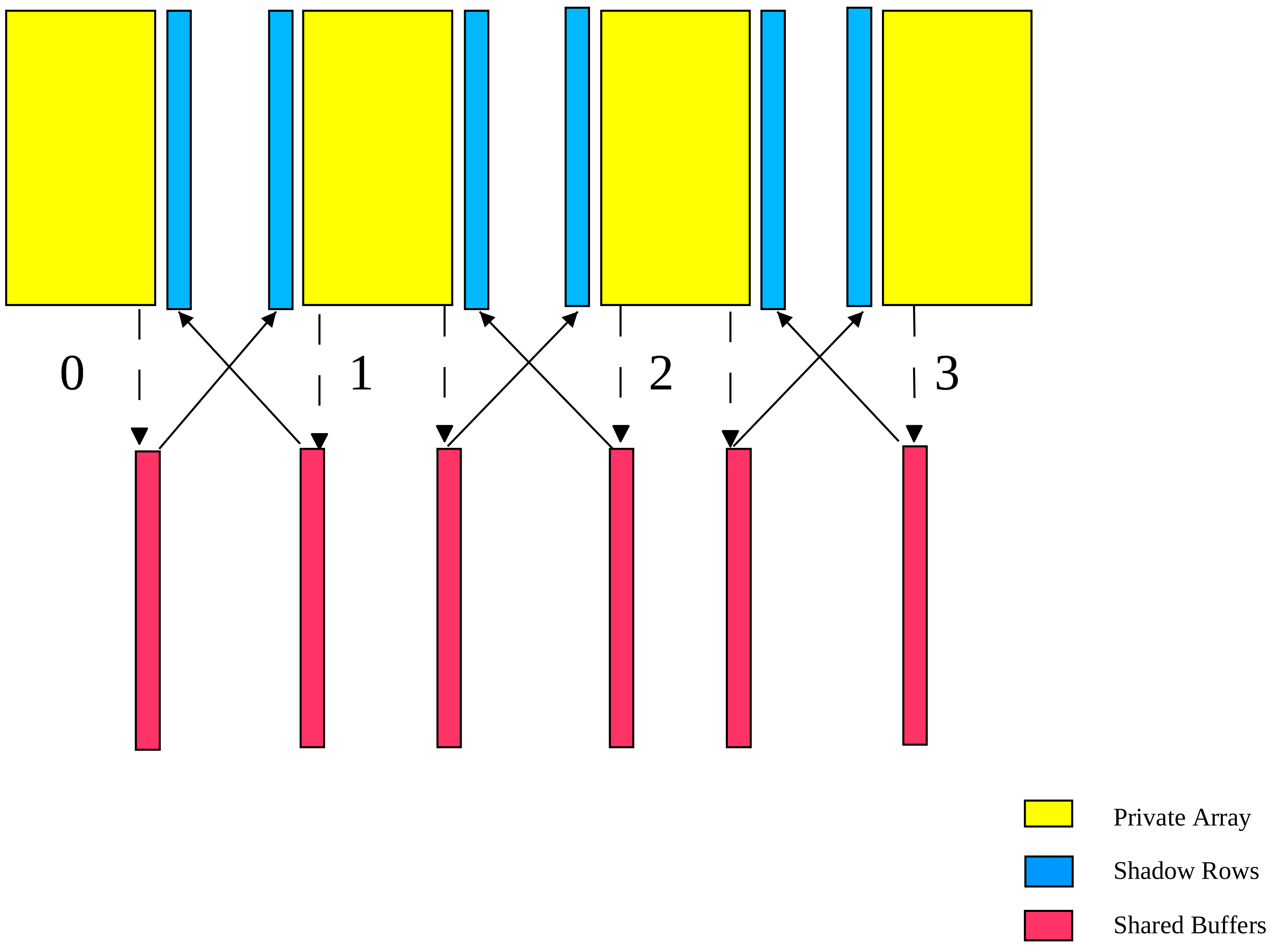

BEST 16-PE HALO EXCHANGES
ORIGIN-MPI
E10000-MPI
ORIGIN-SHMEM
E10000-OpenMP
wall time (seconds)
0.001
0.0001
1e-05
2
4
8
16
32
64
128
256
512
1024
halo length (N)

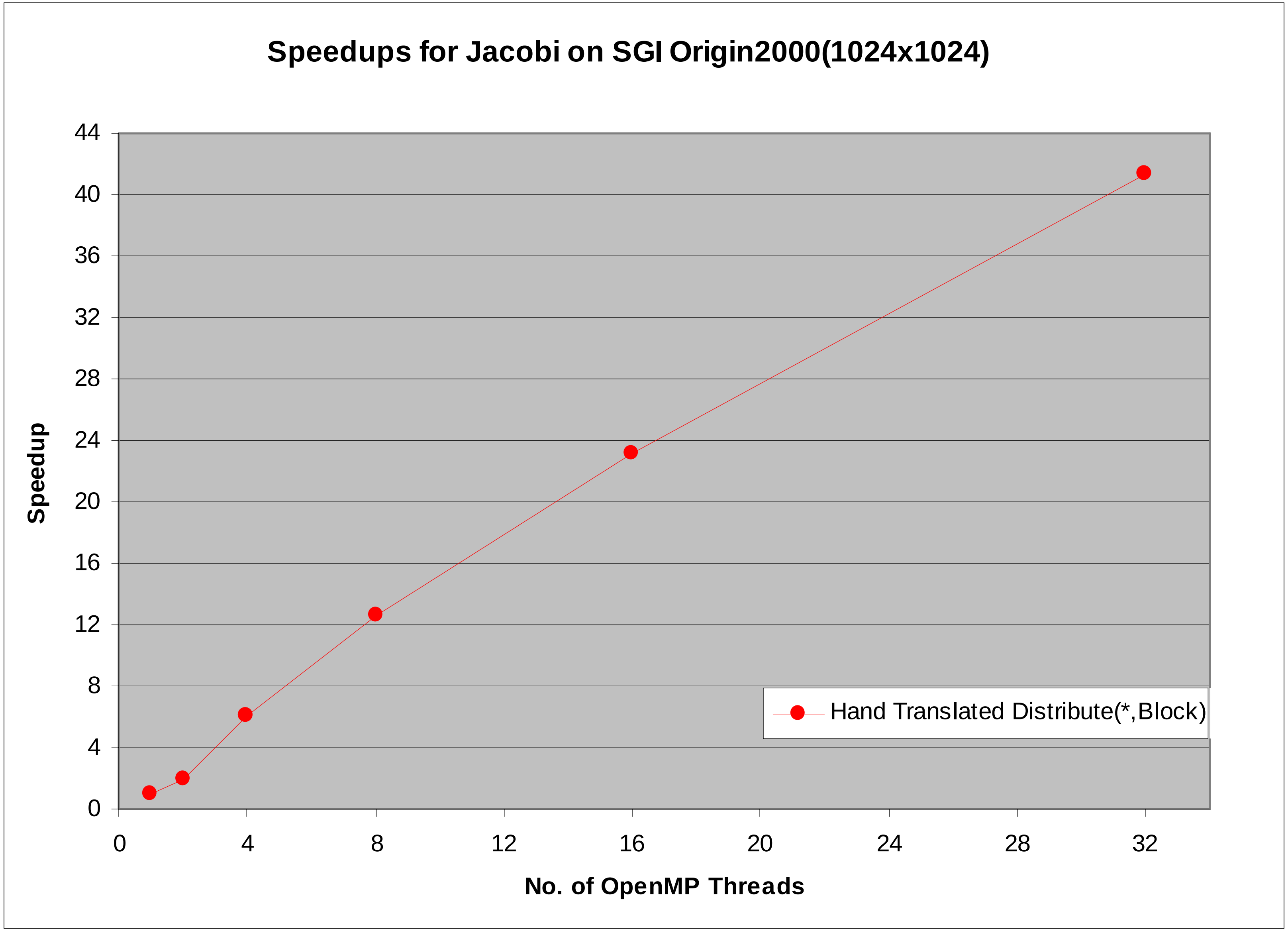
OpenMP Jacobi on Origin

```
!$OMP Parallel Shared (sum, bufleft, bufright ) &
!$OMP PRIVATE ( a, b, threadnum, mylb1, myub1, ..)
 . . . . . .
do i = 1, n
 bufleft ( i, threadnum ) = b ( i, 1 )
end do
 . . . . .
do j =mylb1, myub1
 do i =mylb2, myub2
  a (i,j) = b(i-1,j) + …
 . . . . .
```

- Private arrays (include shadow region)
- Buffers used to share data

Data Decomposition for Private Version





Speedups for Jacobi on SGI Origin2000(1024x1024)
Speedup
0
4
8
12
16
20
24
28
32
36
40
44
0
4
8
12
16
20
24
28
32
No. of OpenMP Threads
Hand Translated Distribute(*,Block)

OpenMP Jacobi on Origin

```
!$OMP Parallel Shared (sum, bufleft, bufright ) &
!$OMP PRIVATE ( a, b, threadnum, mylb1, myub1, ..)
 . . . . . .
do i = 1, n
 bufleft ( i, threadnum ) = b ( i, 1 )
end do
 . . . . .
do j =mylb1, myub1
 do i =mylb2, myub2
  a (i,j) = b(i-1,j) + …
 . . . . .
```

- It is generally hard work to write this code

OpenMP Jacobi on Origin

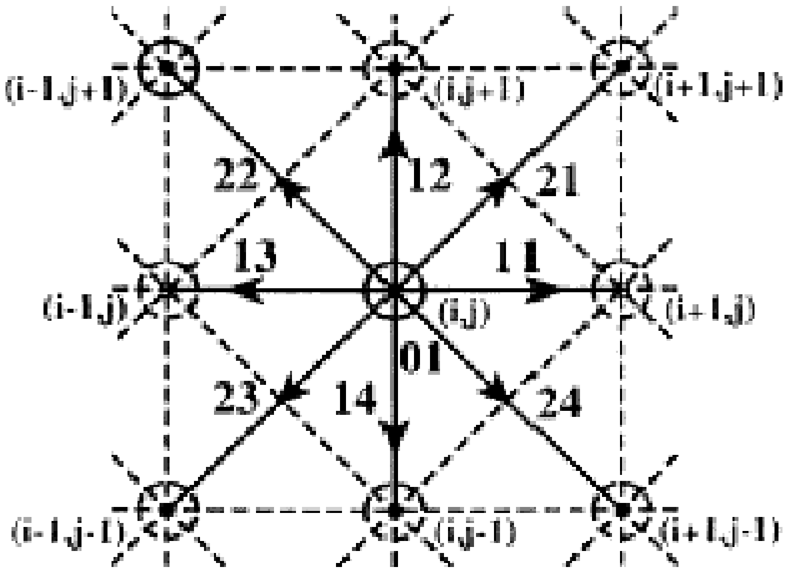
```
!$NMP DISTRIBUTE A (*,BLOCK), B(*, BLOCK)
!$NMP SHADOW B ( 0, 1:1 )
!$OMP Parallel Shared ( a, b, sum)
. . . . . . .
do j = 1, n
 do i = 1, n
  a (i,j) = b(i-1,j) + …
 enddo
enddo
```

- Data is distributed, work mapped accordingly
- Compiler generates private arrays, buffers and code to copy data to and from buffers

Lattice-Boltzmann Equation (LBE)

- LBE code supplied by L.S. Luo, NASA Langley
- Finite difference equations
- Update is 2-d Jacobi using data from 8 neighboring points
- But data associated with neighboring points is also updated

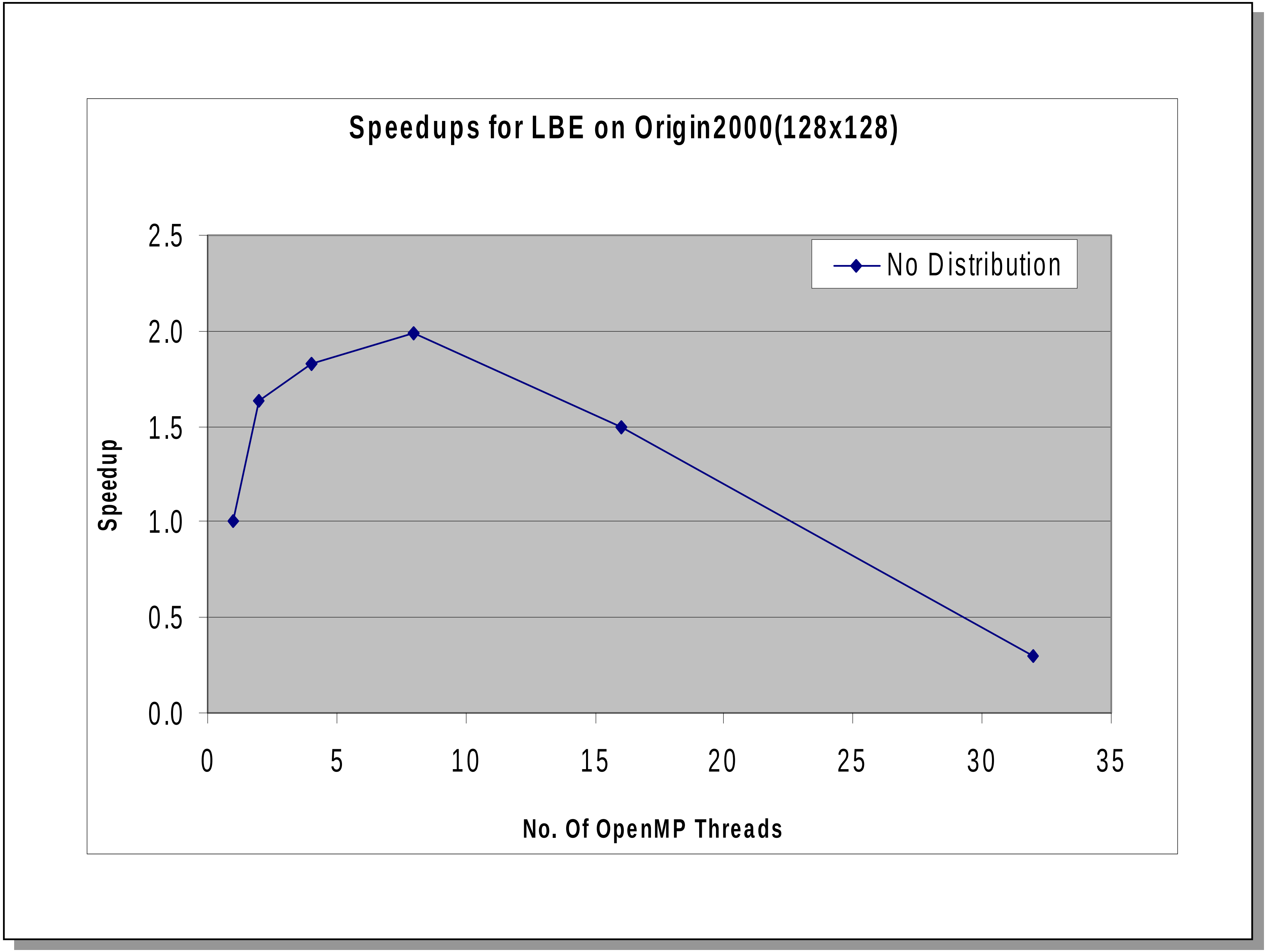
Discretization of velocities for the 9-bit LBM

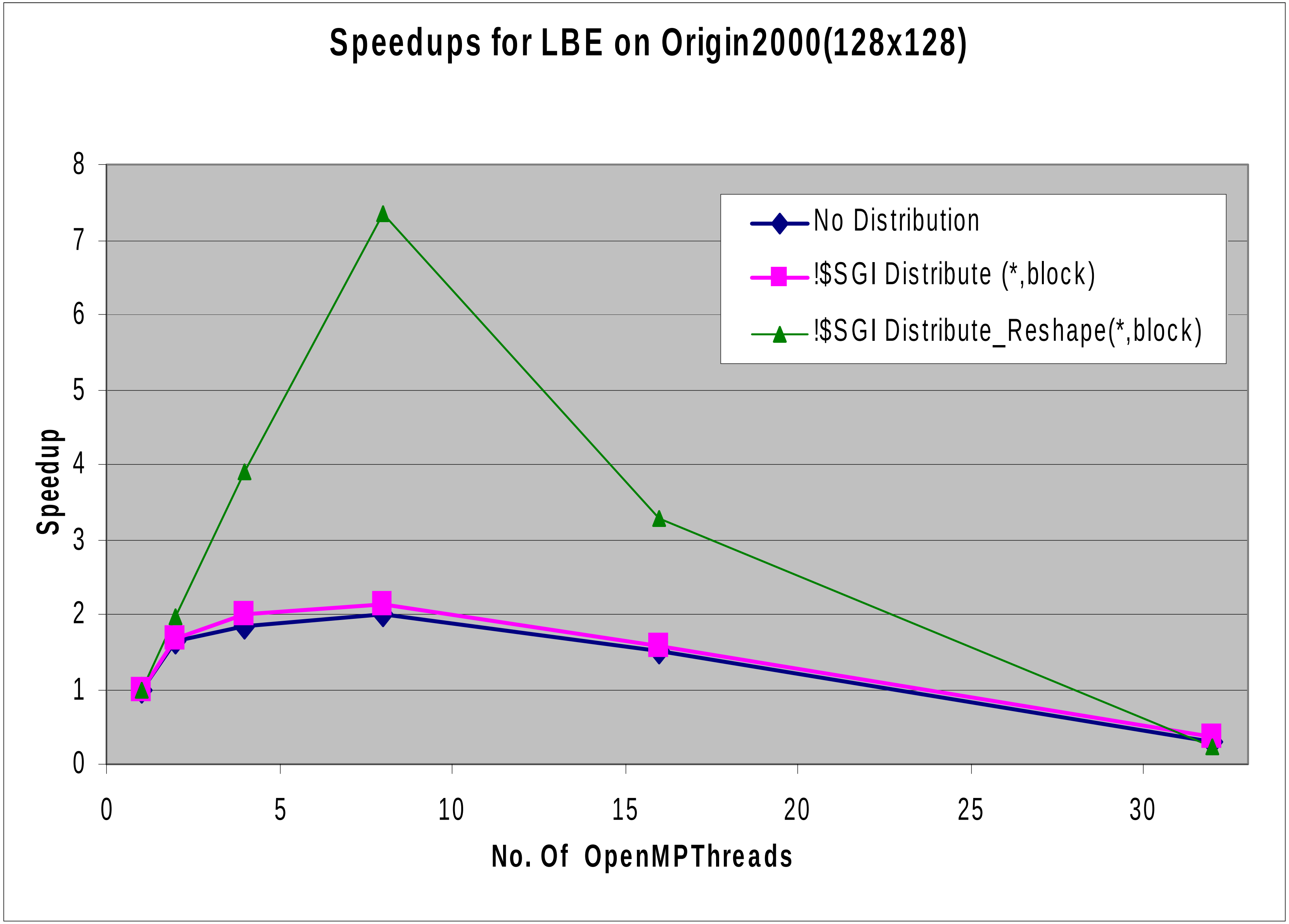


Lattice-Boltzmann Equation

```
!$SGI DISTRIBUTE F ( *, *, BLOCK), FOLD(*, *, BLOCK)
!$OMP Parallel Shared ( f, fold )
!$OMP DO
do j = 1, n
 do i = 1, n
   f( i, 0, j )  =  fold ( i, 0, j) + …
   f(i+1, 1, j) = fold ( i, 1, j) + …
   f( i, 2, j+1) = fold ( i, 2, j) + …
   f( i, 4,  j-1) =  fold ( i, 4, j) + …
   . . . .
 enddo
enddo
```

- Multiple processors write cache lines of f
- Test size small: decreasing accuracy of distribution

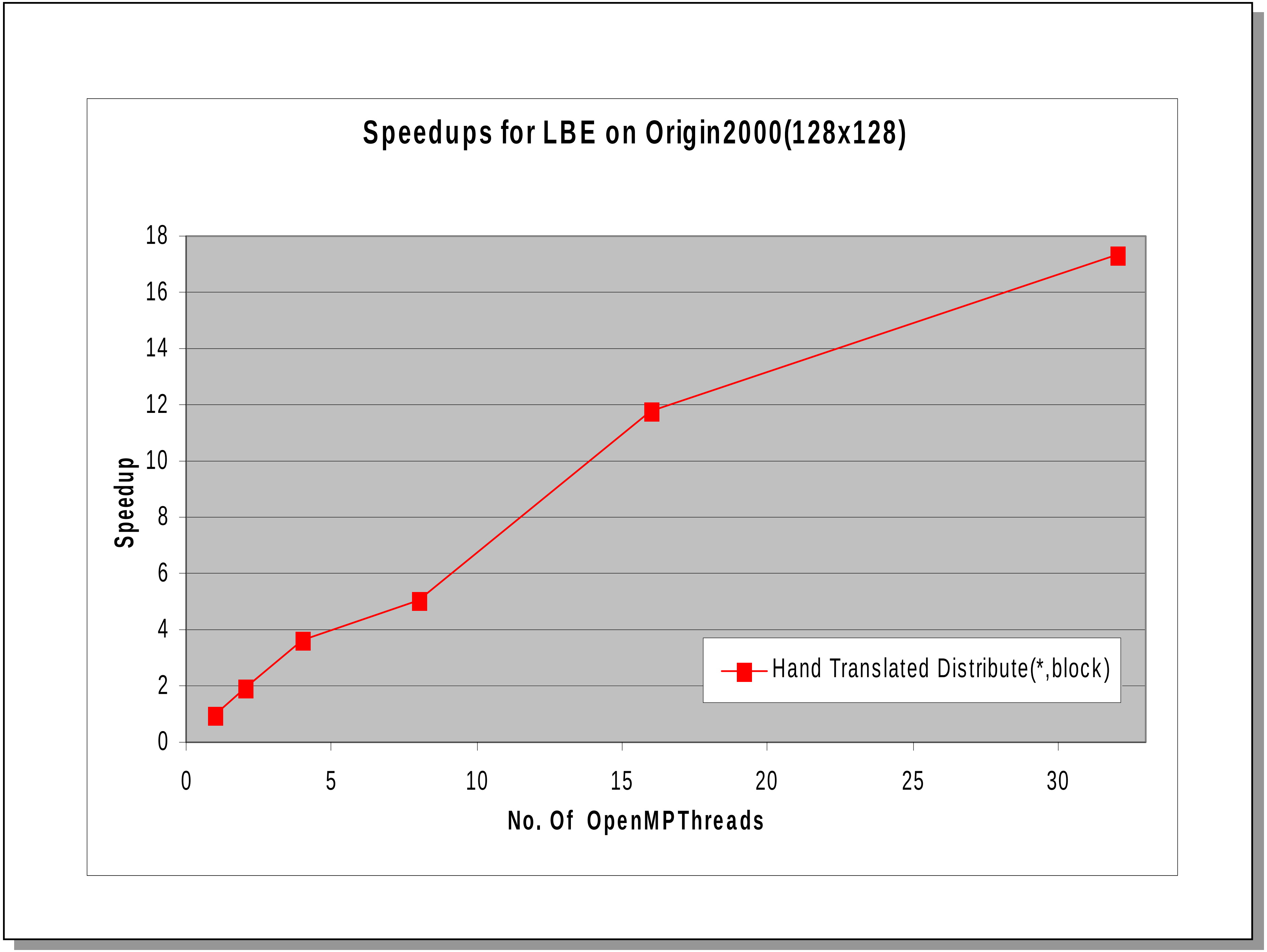

Speedups for LBE on Origin2000(128x128)
No Distribution
Speedup
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
30
35
No. Of OpenMP Threads



Speedups for LBE on Origin2000(128x128)
Speedup
No. Of OpenMPThreads
No Distribution
!$SGI Distribute (*,block)
!$SGI Distribute_Reshape(*,block)
0
1
2
3
4
5
6
7
8
0
5
10
15
20
25
30

Lattice-Boltzmann Equation

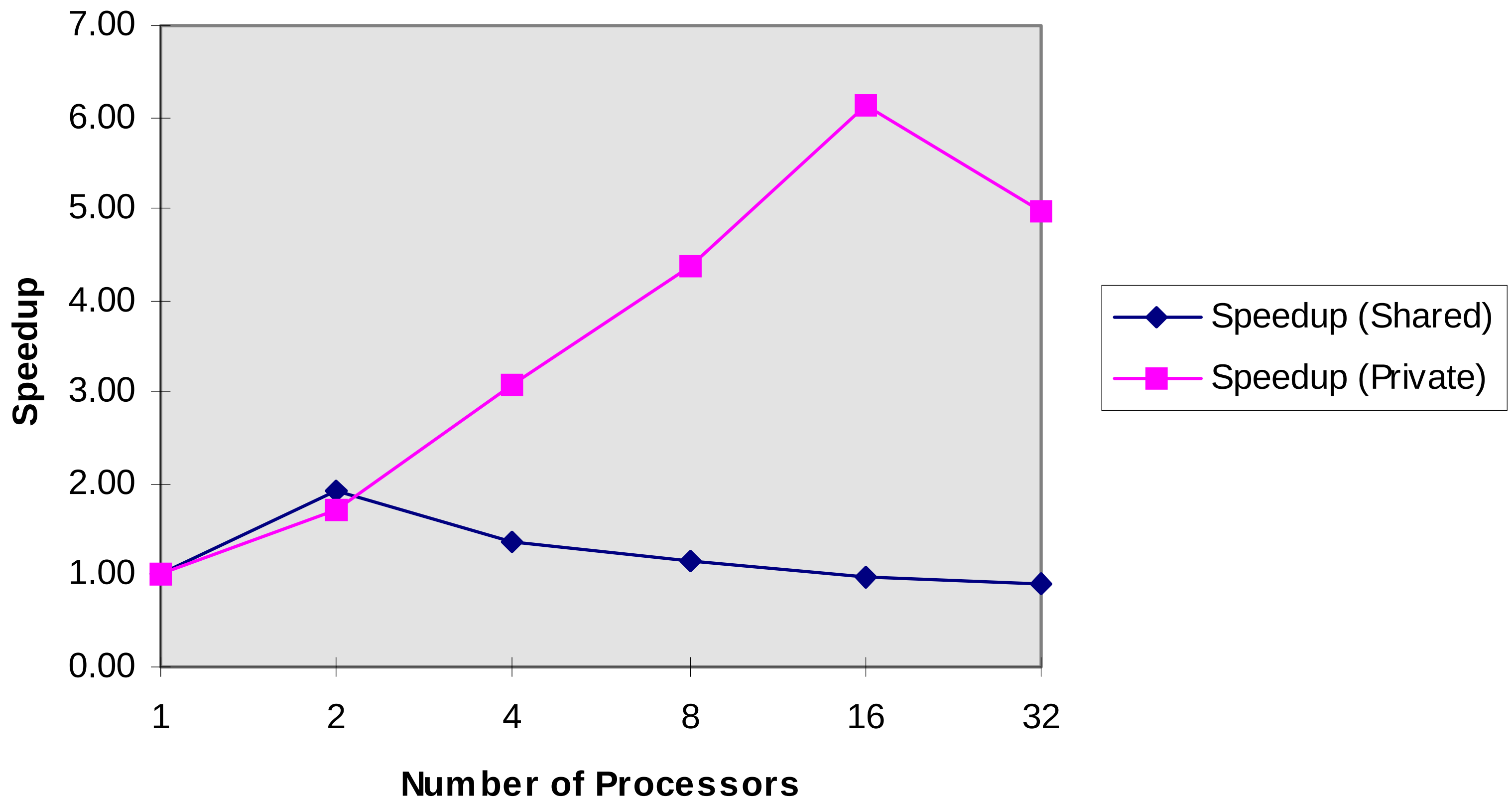
```
!$NMP DISTRIBUTE F ( *, *, BLOCK), FOLD(*, *, BLOCK)
!$NMP SHADOW F ( 0, 0, 1:1 )
!$OMP Parallel Shared (f, fold)
. . . . . .
!$OMP DO
do j = 1, n
 do i = 1, n
    f( i, 0, j )  =  fold ( i, 0, j) + …
    f(i+1, 1, j) = fold ( i, 1, j) + …
    f( i, 2, j+1) = fold ( i, 2, j) + …
    f( i, 4,  j-1) =  fold ( i, 4, j) + …
    . . . .
 enddo
enddo
```


Speedups for LBE on Origin2000(128x128)
Speedup
18
16
14
12
10
8
6
4
2
0
0
5
10
15
20
25
30
No. Of OpenMPThreads
Hand Translated Distribute(*,block)

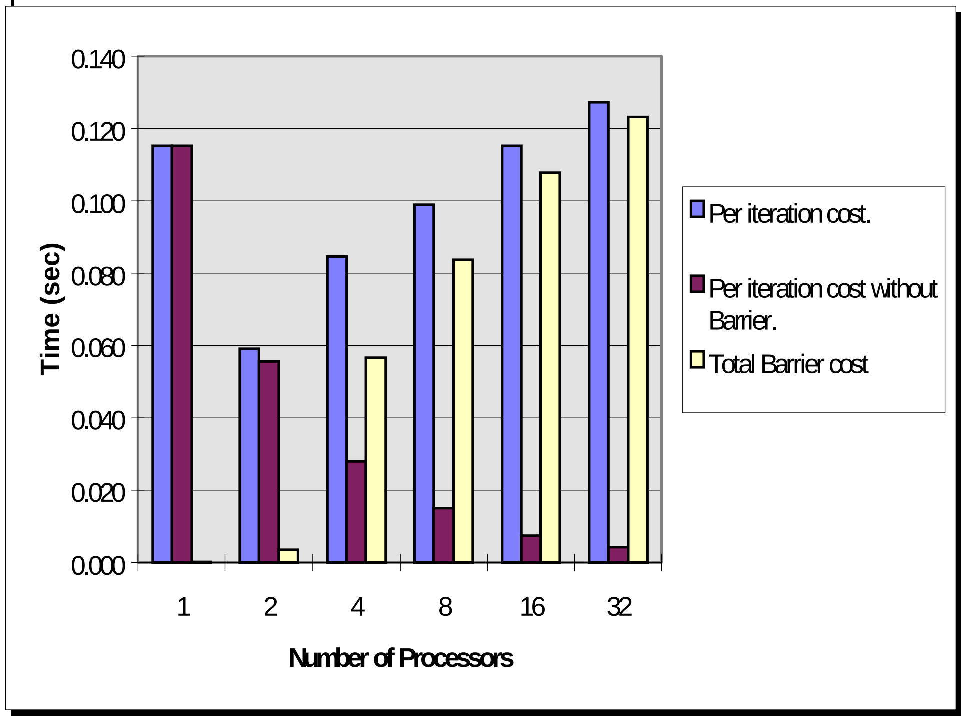
SPMD Style on Software DSM

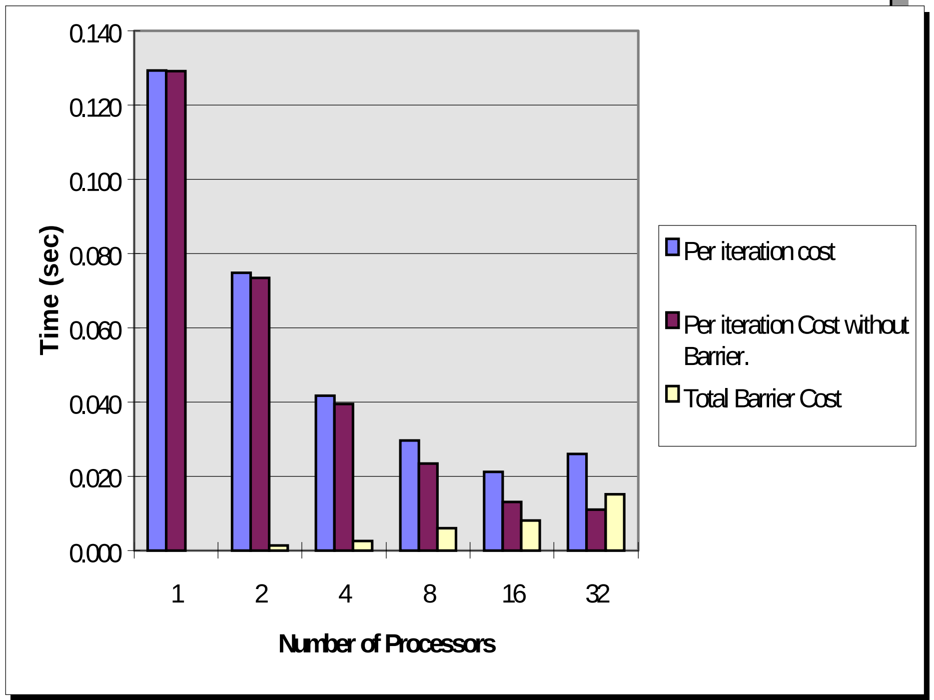
- Tested on SP2 with TreadMarks also
- Slides show Jacobi example
- Shared version: arrays declared as shared, system handles references
- Private version: private copies of local part of decomposed array, buffers hold shared parts of array

Speedups: Shared & Private Versions



Per-Iteration Cost





Shared

Private

Data/Work Locality Features

- Vendors provide user-level directives
- But features differ considerably
  - markedly different sets of extensions
  - translation, rules at subroutine boundaries…
- Do not necessarily provide scalable performance
- Do not give much support for irregular computations
  - GEN_BLOCK might be modest improvement

HPF for Locality (and more)

- SPMD programming style provides scalability on CC-NUMA systems
- Not easy for user to create SPMD code
- Could be generated via HPF-like translation

Issues in Combining Features

- Incremental development
- Storage and sequence association
- Which data distribution features are "enough"?
- Mappings to nodes or processors?
- Simplify procedure interface?

Summary

- OpenMP popular on SMPs, ccNUMAs
- Lacks facilities for expressing data locality, alignment of thread and data
- HPF features for data/work locality can be used with OpenMP
- Translation scheme generates SPMD OpenMP code with high performance