

CSE 260

Lecture 18

Irregular Problems: sparse matrices
NUMA architecture and programming

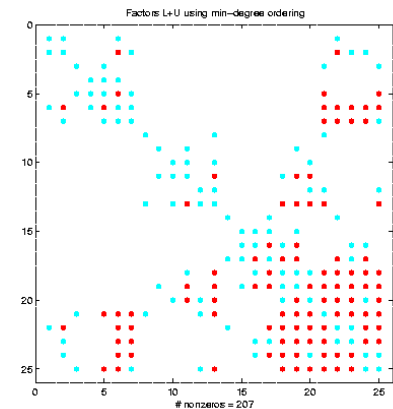
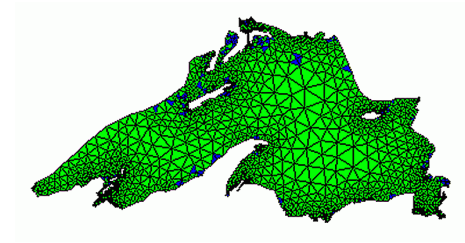
Announcements

Today's lecture

- Irregular Problems
 - ♦ Sparse matrices
- NUMA

Classifying application domains – Colella's “7 Dwarfs”

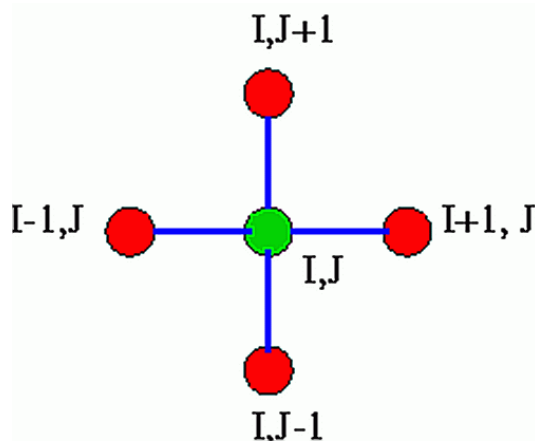
- ◆ Classify applications according to a pattern of data access and operations on the data: important in HPC
- ◆ Structured grids
 - Image processing, simulations
- ◆ Dense linear algebra:
 - Gaussian elimination, SVD, many others
- ◆ N-body methods
- ◆ Sparse linear algebra
- ◆ Unstructured Grids
- ◆ Spectral methods (FFT)
- ◆ Monte Carlo



Sparse Matrices

- A matrix where we employ knowledge about the location of the non-zeroes
- Consider Jacobi's method with a 5-point stencil

$$u'[i,j] = (u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i,j+1] - h^2 f[i,j]) / 4$$



Discrete Poisson Problem on 4-by-4 Grid

$$\begin{bmatrix}
 4 & -1 & & & \\
 -1 & 4 & -1 & & \\
 & -1 & 4 & -1 & \\
 & & -1 & 4 & -1 \\
 & & & -1 & 4
 \end{bmatrix}
 \begin{bmatrix}
 U(1,1) \\
 U(2,1) \\
 U(3,1) \\
 U(4,1) \\
 U(1,2) \\
 U(2,2) \\
 U(3,2) \\
 U(4,2) \\
 U(1,3) \\
 U(2,3) \\
 U(3,3) \\
 U(4,3) \\
 U(1,4) \\
 U(2,4) \\
 U(3,4) \\
 U(4,4)
 \end{bmatrix}
 =
 \begin{bmatrix}
 b(1,1) \\
 b(2,1) \\
 b(3,1) \\
 b(4,1) \\
 b(1,2) \\
 b(2,2) \\
 b(3,2) \\
 b(4,2) \\
 b(1,3) \\
 b(2,3) \\
 b(3,3) \\
 b(4,3) \\
 b(1,4) \\
 b(2,4) \\
 b(3,4) \\
 b(4,4)
 \end{bmatrix}$$

Web connectivity Matrix: 1M x 1M

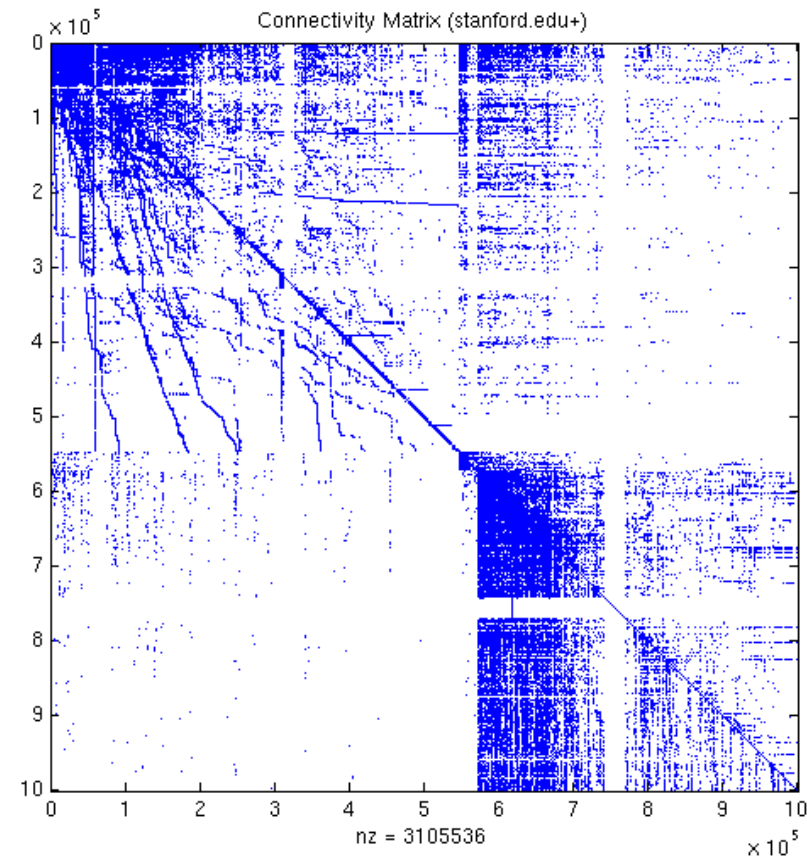
1M x 1M submatrix of the web connectivity graph, constructed from an archive at the Stanford WebBase

3 non-zeroes/row

$$\begin{aligned}\text{Dense: } 2^{20} \times 2^{20} &= 2^{40} \\ &= 1024 \text{ Gwords}\end{aligned}$$

$$\begin{aligned}\text{Sparse: } (3/2^{20}) \times 2^{40} \\ &= 3 \text{ Mwords}\end{aligned}$$

Sparse representation saves a factor of 1 million in storage



Jim Demmel

Circuit Simulation

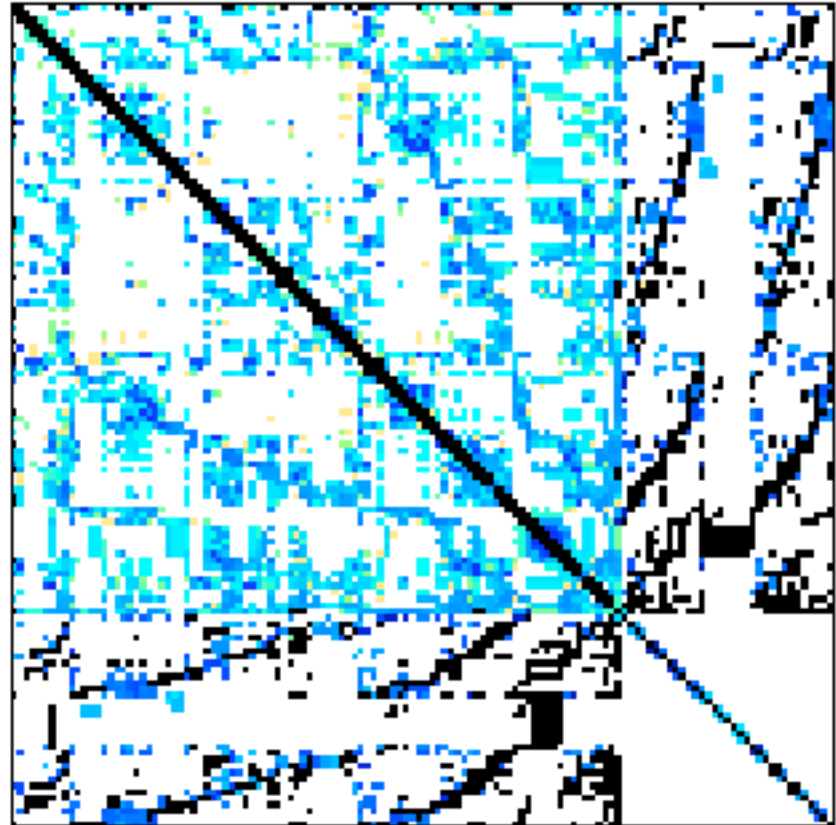
Motorola Circuit

$170,998^2$

958,936 nonzeros

.003% nonzeros

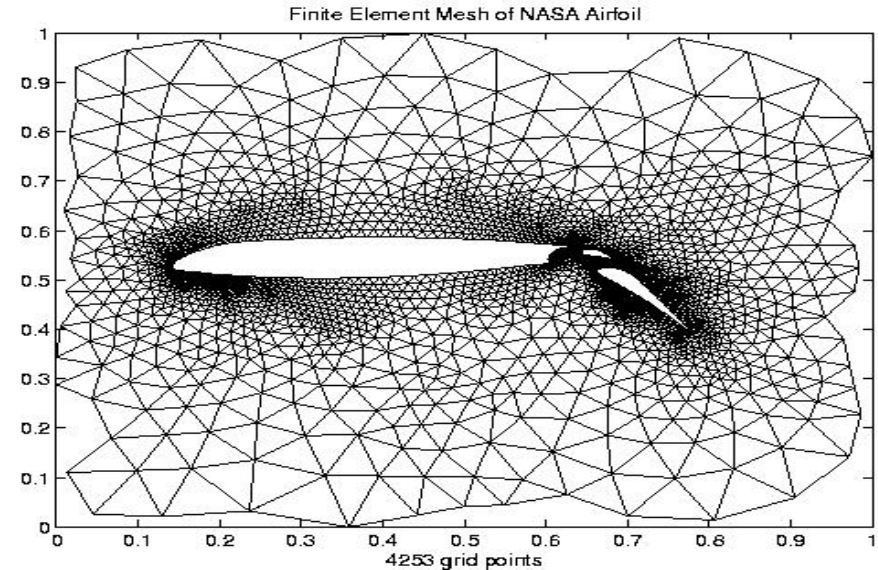
5.6 nonzeros/row



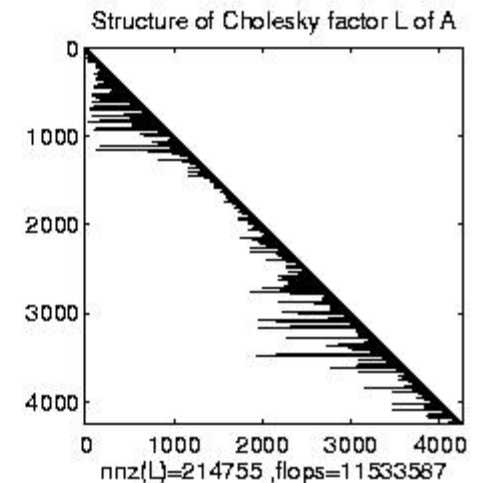
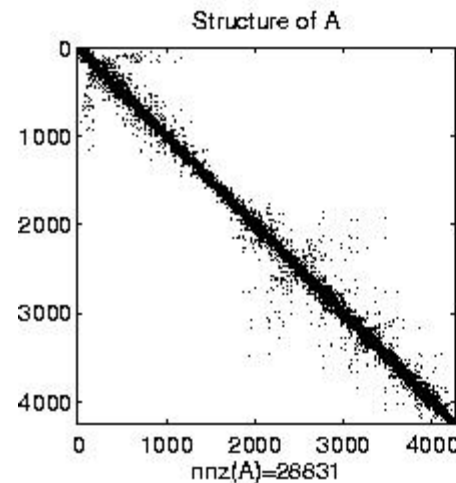
www.cise.ufl.edu/research/sparse/matrices/Hamm/scircuit.html

Generating sparse matrices from unstructured grids

- In many applications of sparse matrices, we generate the matrix from a large data input file
- “Unstructured” meshes, e.g. finite element method
- We can apply direct mesh updates, but more common to generate the equivalent sparse matrix

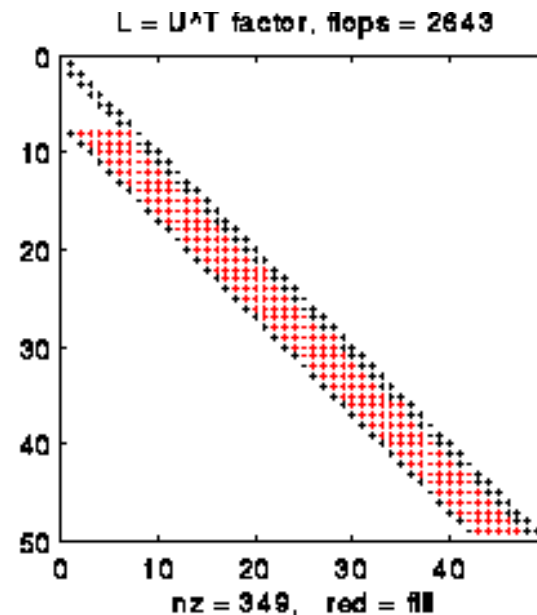
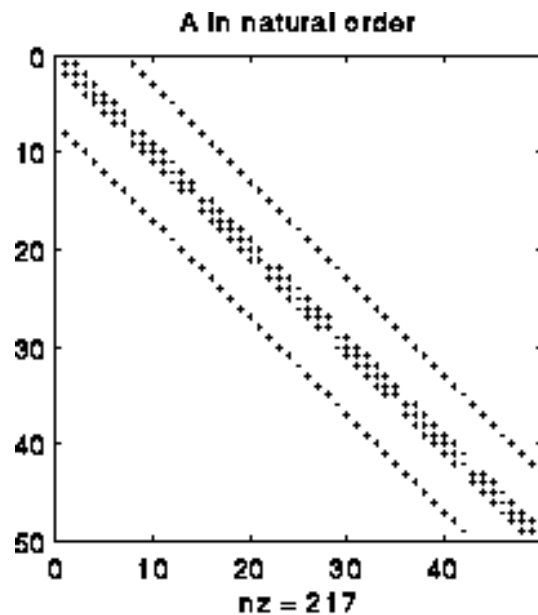


A 2D airfoil



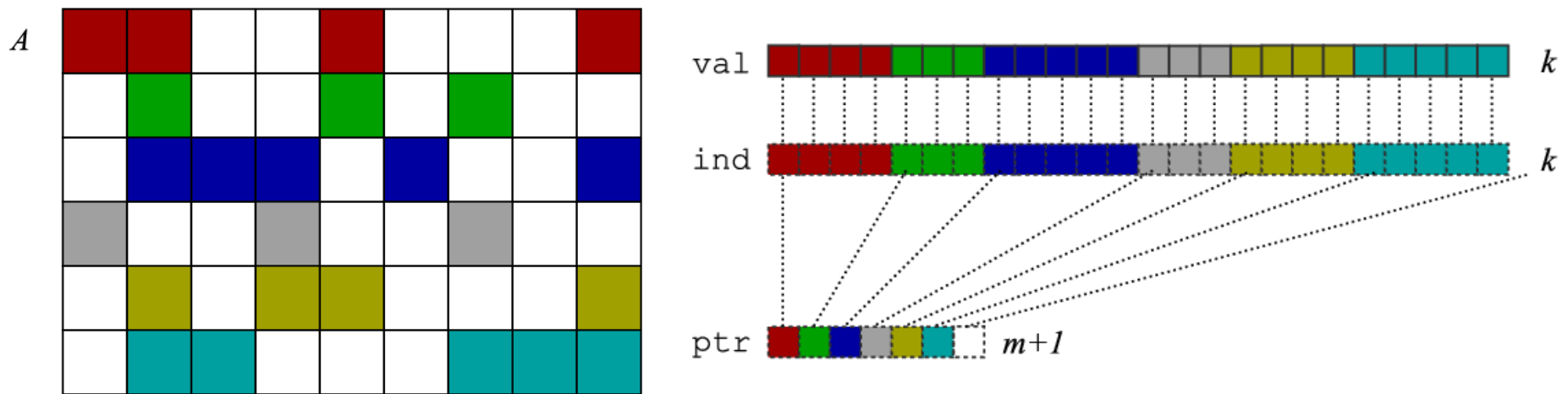
Fill in

- When factorizing (e.g. LU) a sparse matrix, *fill in* can occur: a **zero becomes a non-zero**
- Computation & storage costs
- Matrix reordering can reduce fill in



Sparse Matrix Vector Multiplication

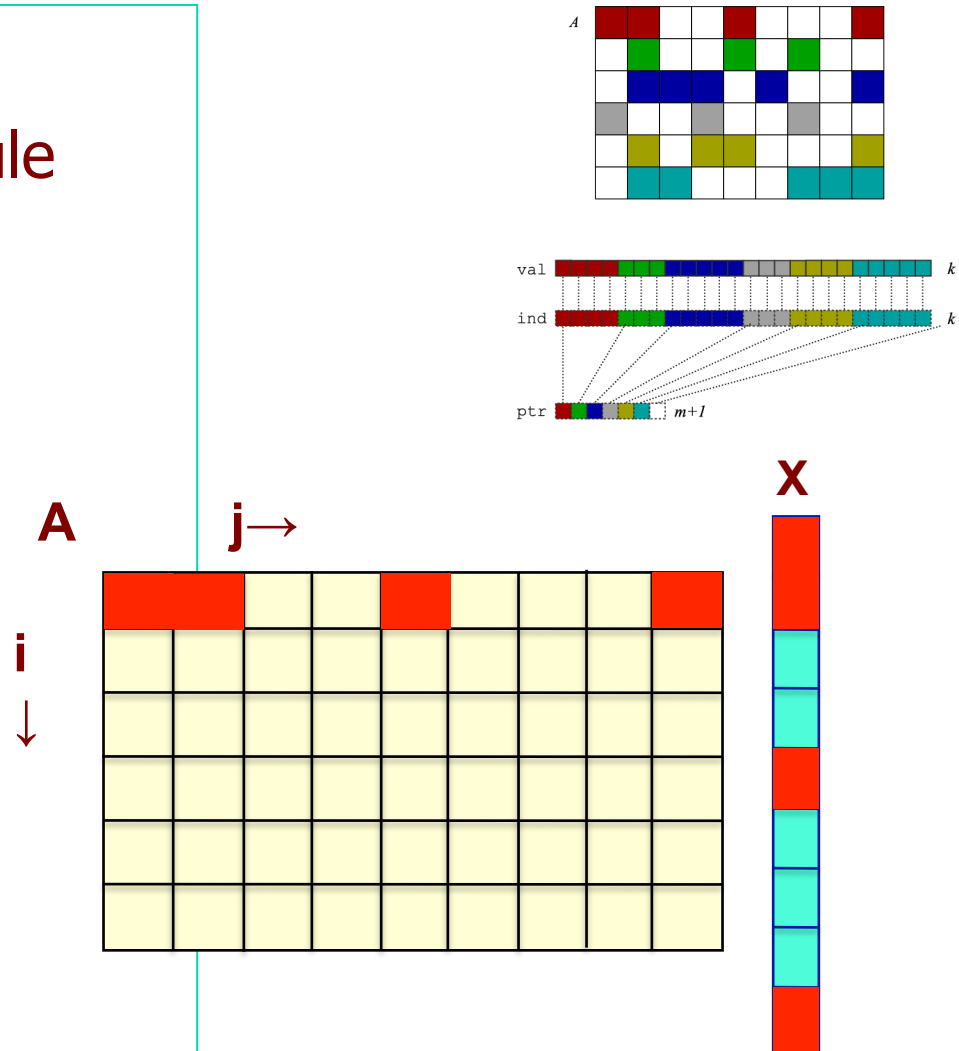
- Important kernel used in linear algebra
- Assume $x[]$ fits in memory of 1 processor
 $y[i] += A[i,j] \times x[j]$
- Many formats, common format for CPUs is Compressed Sparse Row (CSR)



Jim Demmel

Sparse matrix vector multiply kernel

```
// y[i] += A[i,j] × x[j]
#pragma parallel for schedule
    (dynamic,chunk)
for i = 0 : N-1    // rows
    i0= ptr[i]
    i1 = ptr[i+1] - 1
    for j = i0 : i1    // cols
        y[ ind[j]] +=
            val[ j ] *x[ ind[j] ]
    end j
end i
```

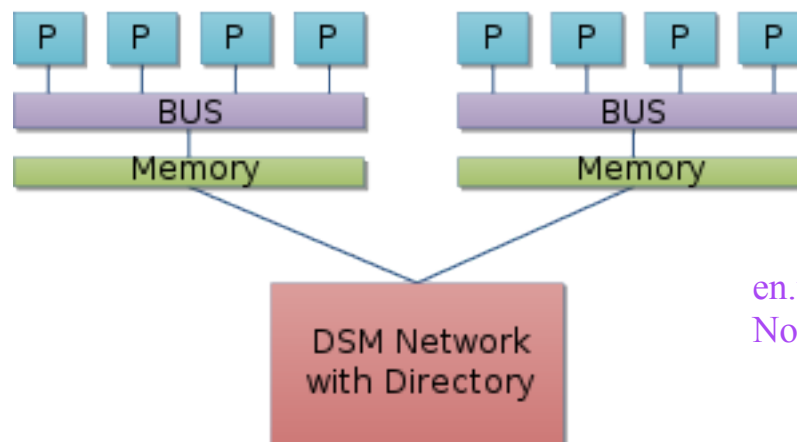


Today's lecture

- Sparse matrices
- **NUMA**
- Example NUMA Systems
 - ◆ Cray XE-6
 - ◆ SGI
- Performance programming

NUMA Architectures

- The address space is global to all processors, but memory is physically distributed
- Point-to-point messages manage coherence
- A **directory** keeps track of sharers, one for each block of memory
- Stanford Dash; NUMA nodes of the Cray XE-6, SGI UV, Altix, Origin 2000



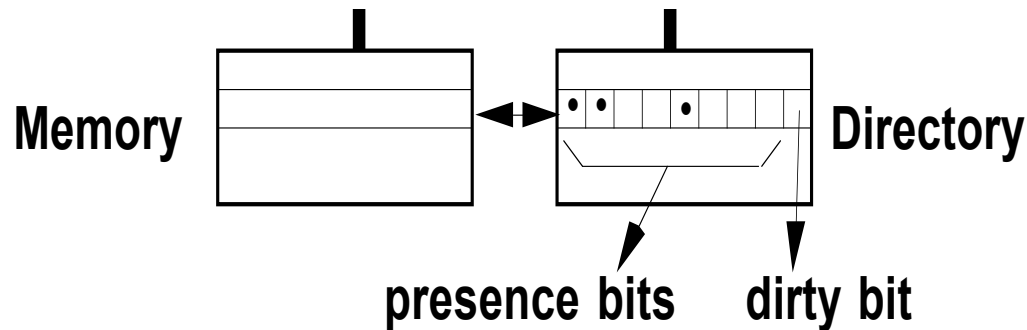
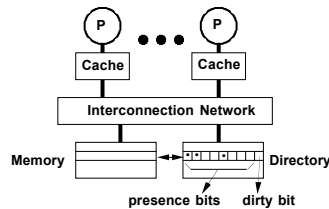
[en.wikipedia.org/wiki/
Non-Uniform_Memory_Access](http://en.wikipedia.org/wiki/Non-Uniform_Memory_Access)

Some terminology

- Every block of memory has an associated **home**: the specific processor that physically holds the associated portion of the global address space
- Every block also has an **owner**: the processor whose memory contains the actual value of the data
- Initially home = owner, but this can change ...
- ... if a processor other than the home processor writes a block

Inside a directory

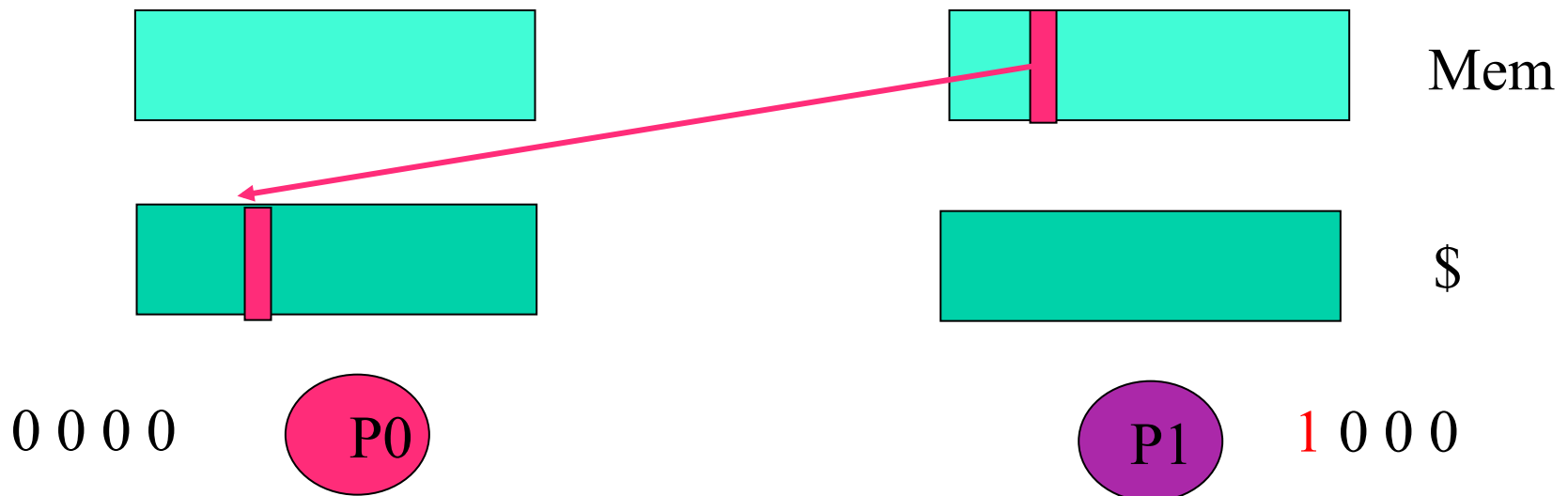
- Each processor has a 1-bit “sharer” entry in the directory
- There is also a dirty bit and a PID identifying the owner in the case of a dirt block



*Parallel
Computer
Architecture,
Culler, Singh,
& Gupta*

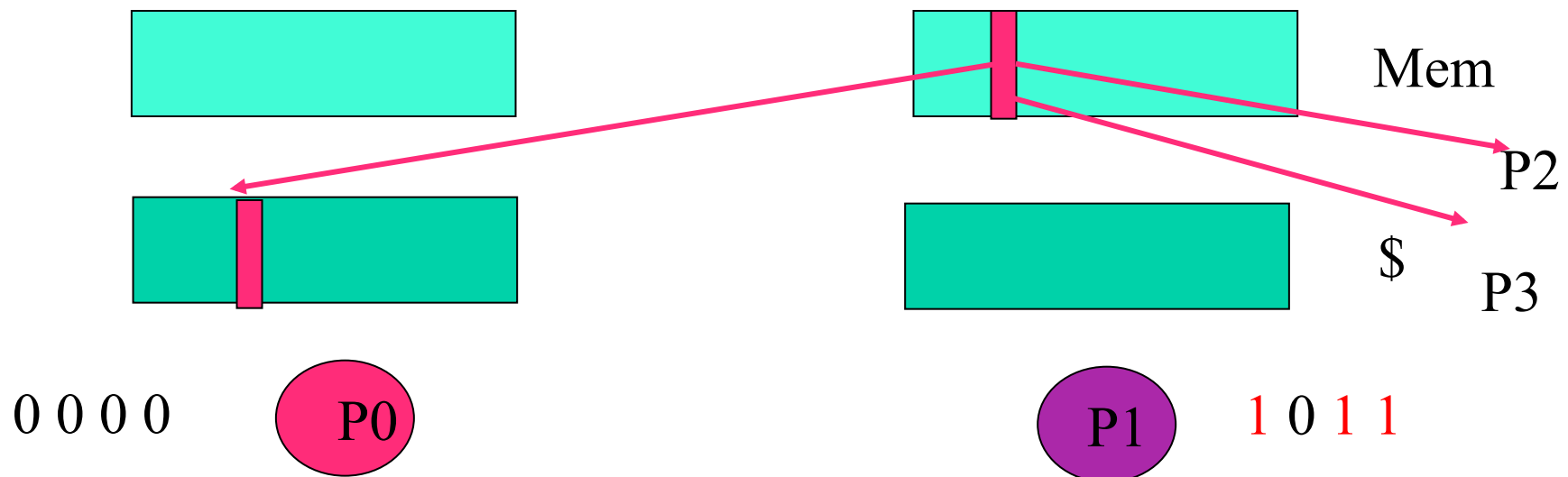
Operation of a directory

- P0 loads **A**
- Set directory entry for **A** (on P1) to indicate that P0 is a sharer



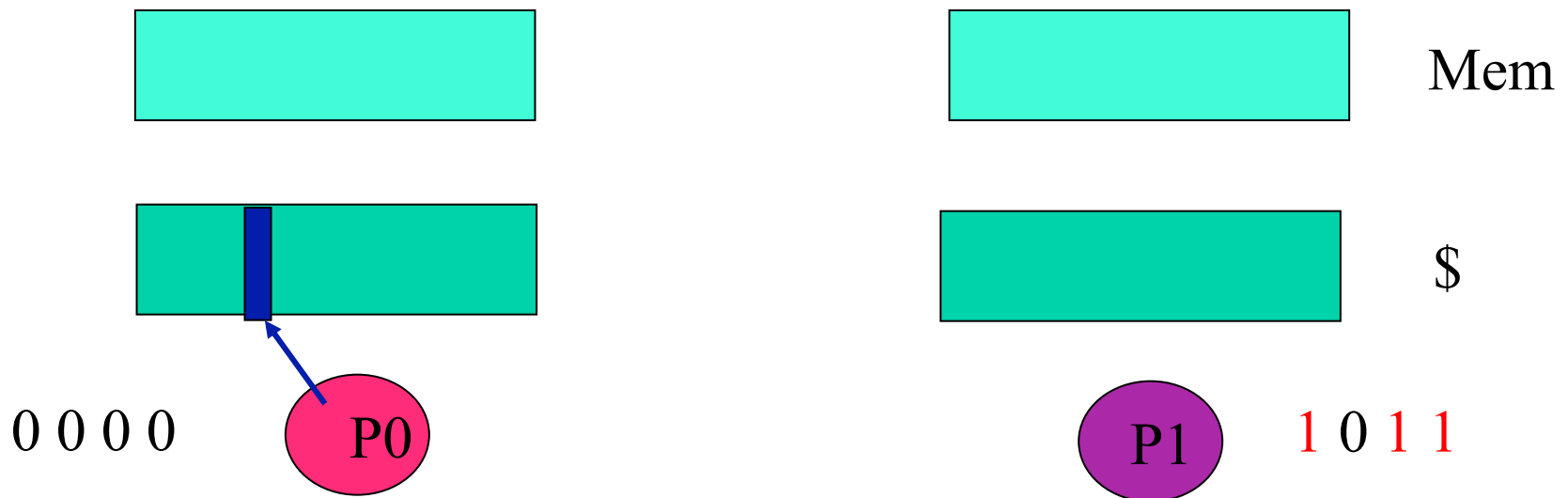
Operation of a directory

- P2, P3 load **A** (not shown)
- Set directory entry for **A** (on P1) to indicate that P0 is a sharer



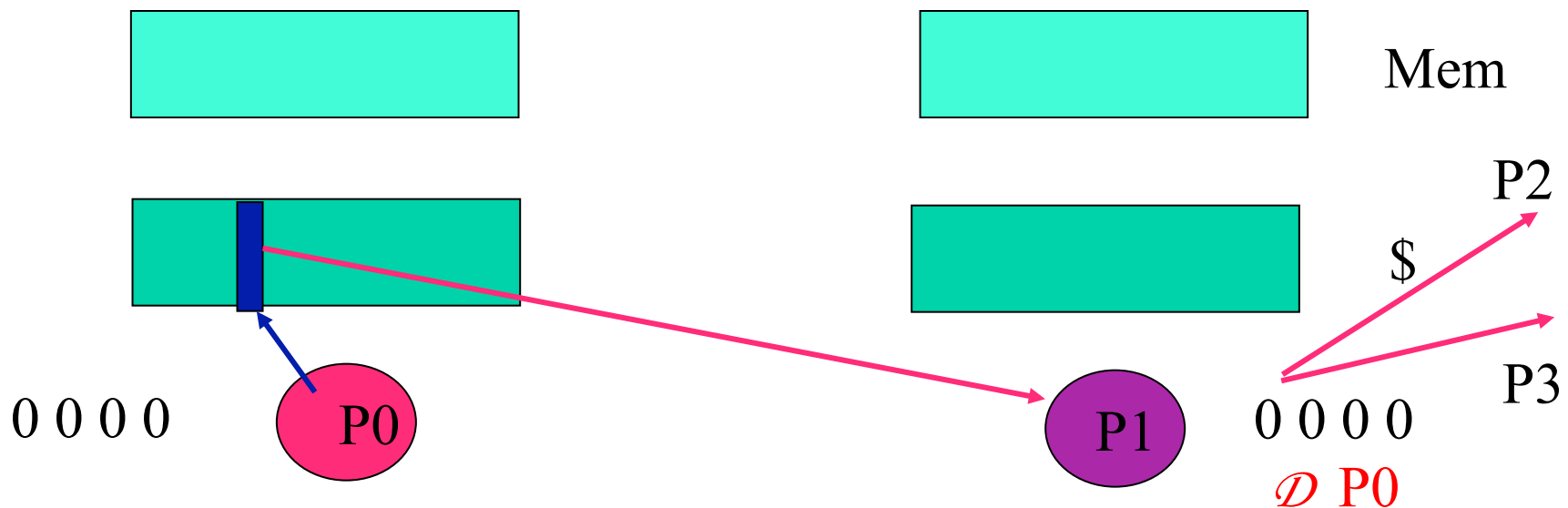
Acquiring ownership of a block

- P0 writes *A*
- P0 becomes the owner of *A*



Acquiring ownership of a block

- P0 becomes the owner of A
- P1's directory entry for A is set to *Dirty*
- Outstanding sharers are invalidated
- Access to line is blocked until all invalidations are acknowledged

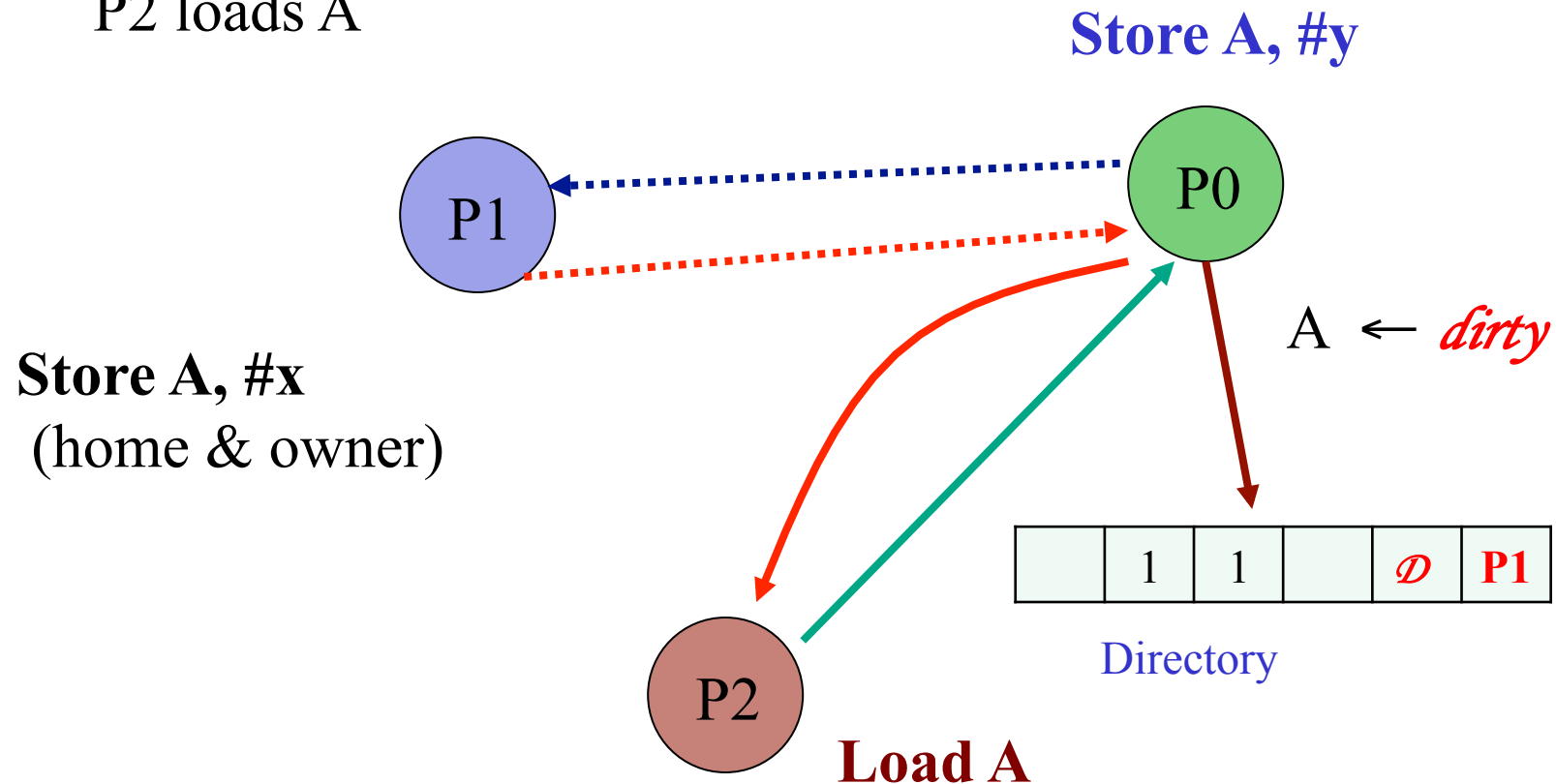


Change of ownership

P0 stores into A (home & owner)

P1 stores into A (becomes owner)

P2 loads A



Forwarding

P0 stores into A (home & owner)

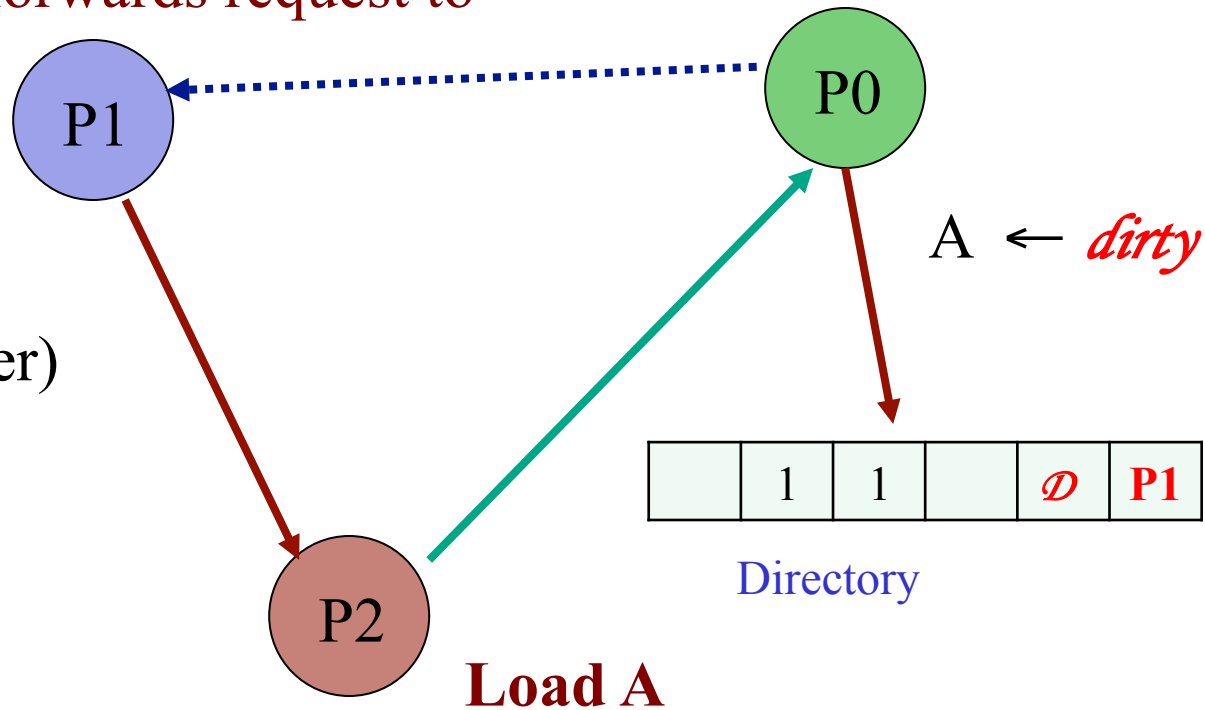
P1 stores into A (becomes owner)

P2 loads A

home (P0) forwards request to
owner (P1)

Store A, #x
(home & owner)

Store A, #y



Performance issues

- False sharing
- Locality, locality, locality
 - ◆ Page placement
 - ◆ Page migration
 - ◆ Copying v. redistribution
 - ◆ Layout

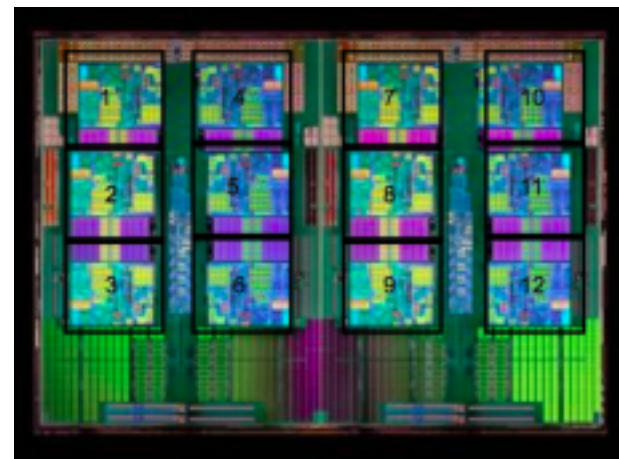
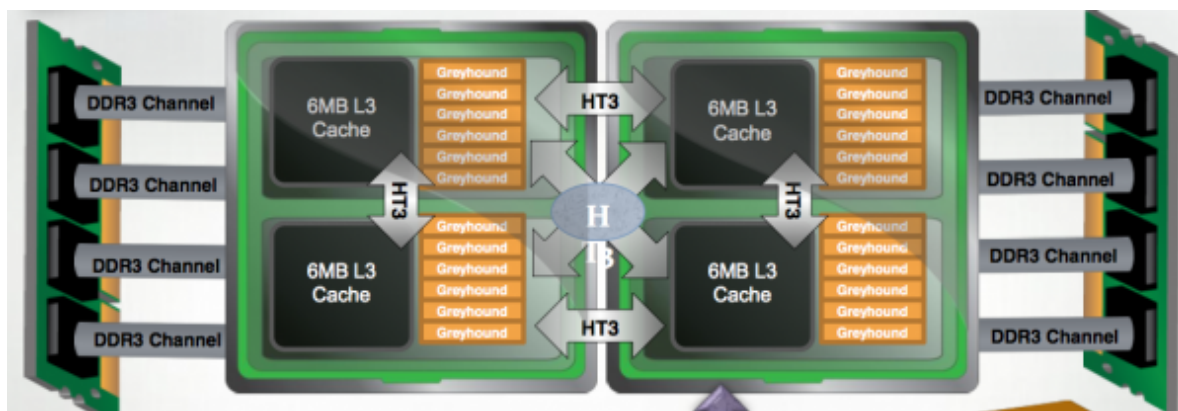
Today's lecture

- Sparse matrices
- NUMA
- Example NUMA Systems
 - ◆ Cray XE-6
 - ◆ SGI
- Performance programming

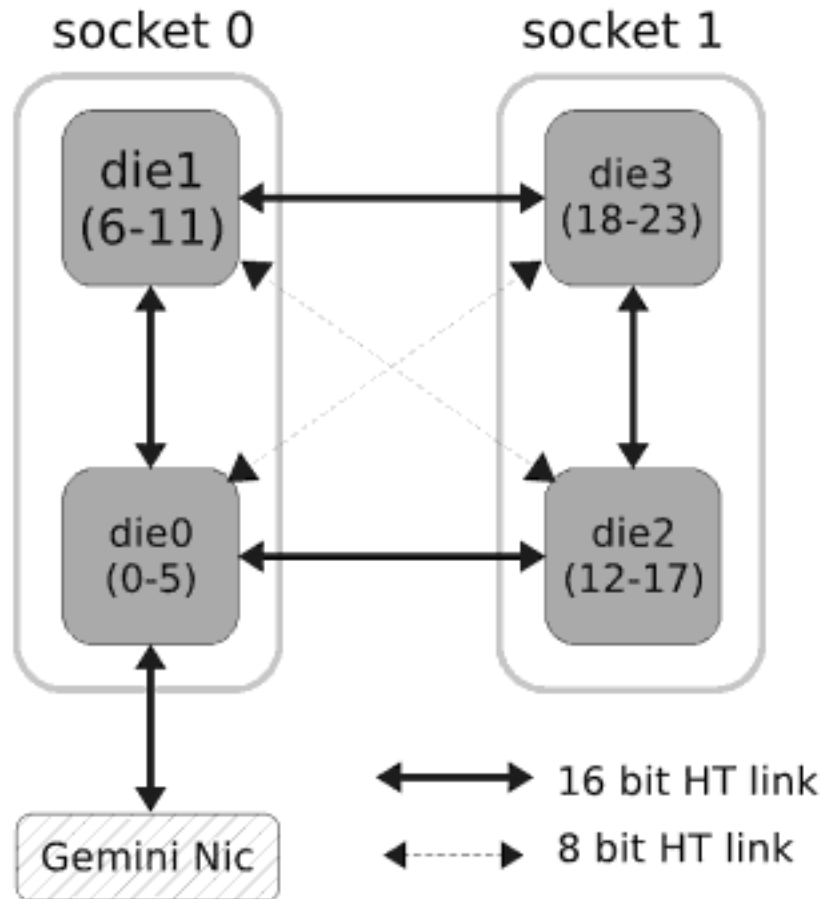
Cray XE6 node

- 24 cores sharing 32GB main memory
- Packaged as 2 AMD Opteron 6172 processors “Magny-Cours”
- Each processor is a directly connected Multi-Chip Module: two hex-core dies living on the same socket
- Each die has 6MB of shared L3, 512KB L2/core, 64K L1/core
 - ♦ 1MB of L3 is used for cache coherence traffic
 - ♦ Direct access to 8GB main memory via 2 memory channels
 - ♦ 4 Hyper Transport (HT) links for communicating with other dies
- Asymmetric connections between dies and processors

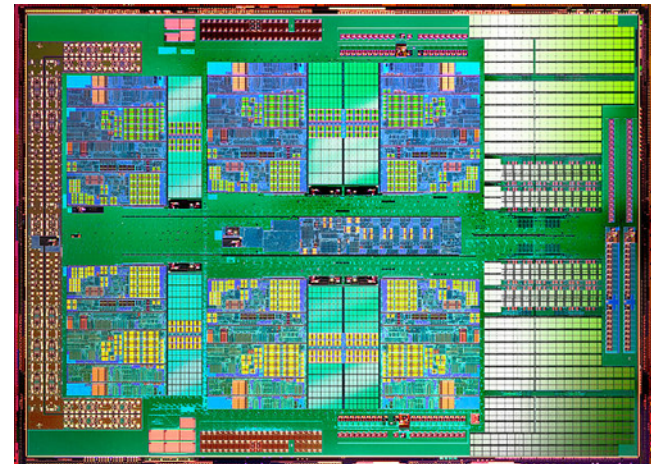
www.nersc.gov/users/computational-systems/hopper/configuration/compute-nodes/



XE-6 Processor memory interconnect (node)

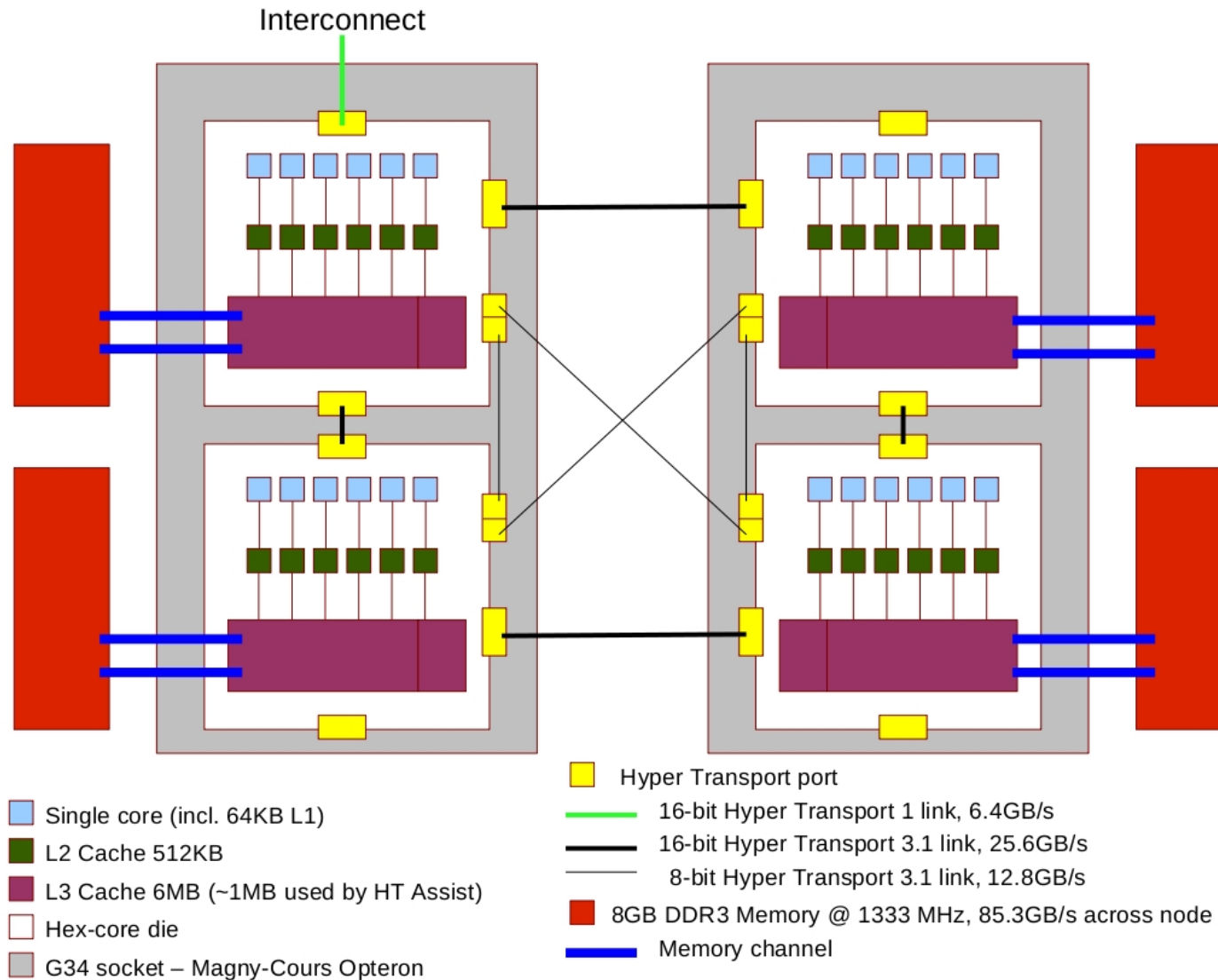


- Asymmetric connections between dies and processors
 - Direct access to 8GB main memory via 2 memory channels
 - 4 Hyper Transport (HT) links for communicating with other dies



<http://www.hector.ac.uk/cse/documentation/Phase2b/#arch>

XE-6 Processor memory interconnect (node)

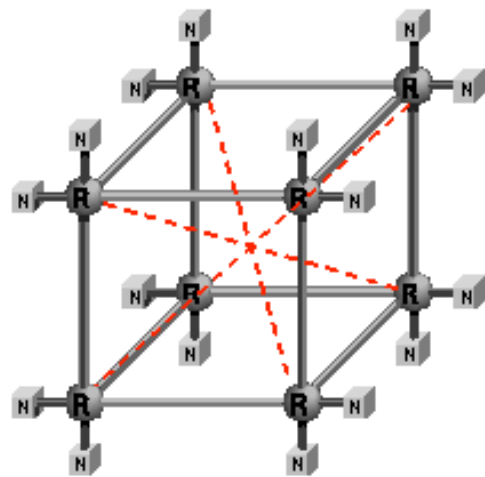


<http://www.hector.ac.uk/cse/documentation/Phase2b/#arch>

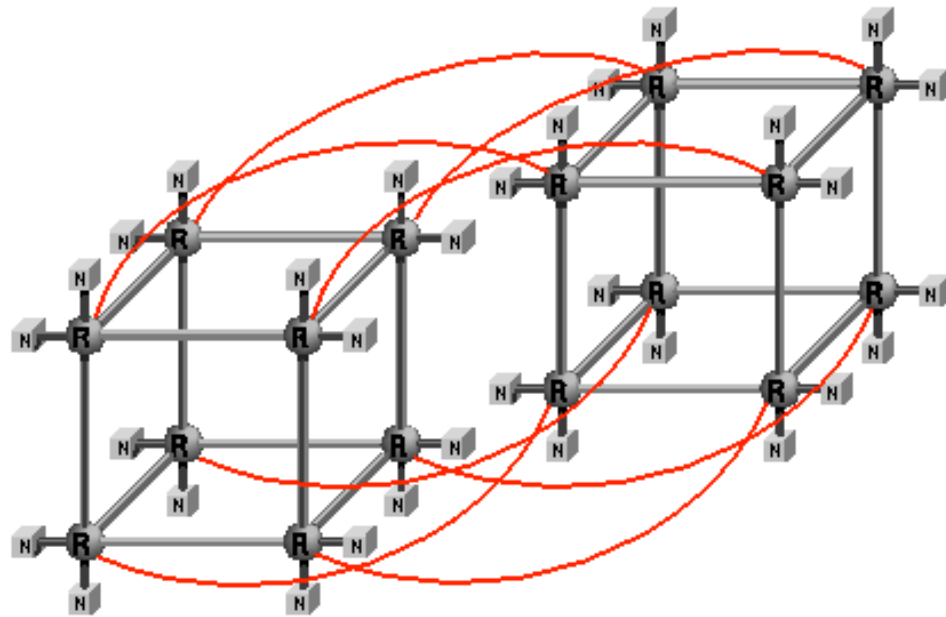
Today's lecture

- Sparse Matrices
- Example NUMA Systems
 - ♦ Cray XE-6
 - ♦ **SGI Origin 2000**
- Performance programming

Origin 2000 Interconnect

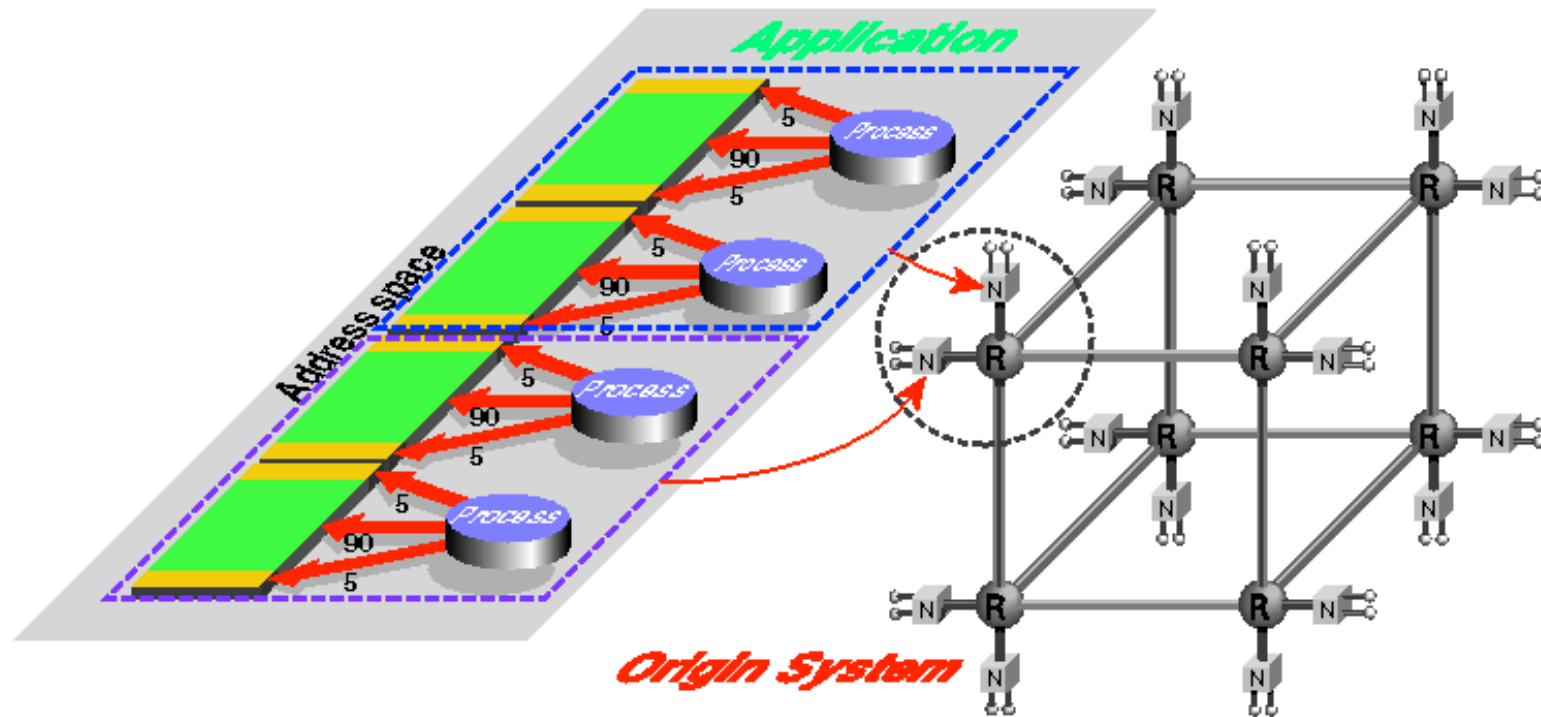


32 processor
system

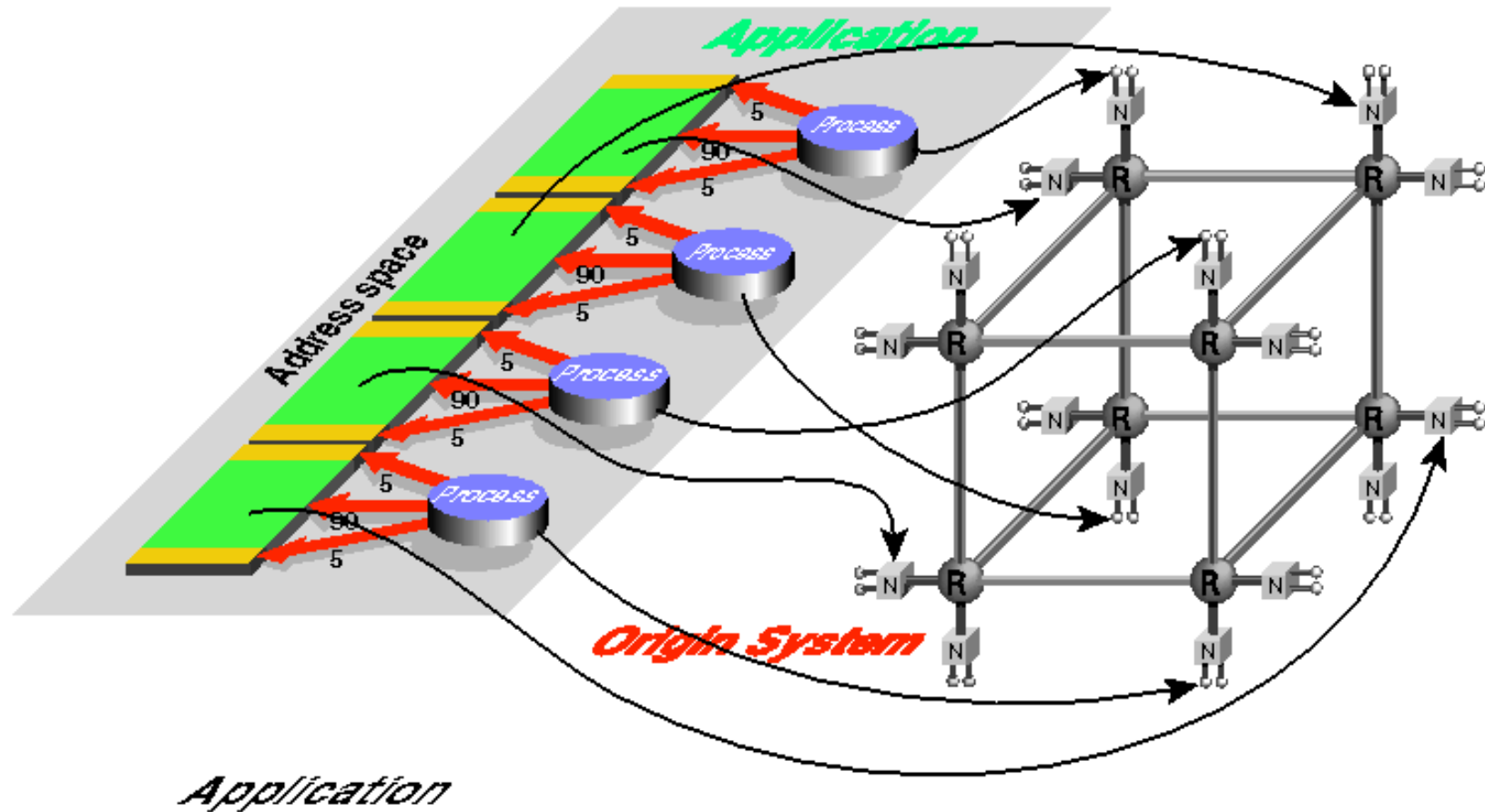


64 processor
system

Locality



Poor Locality



Today's lecture

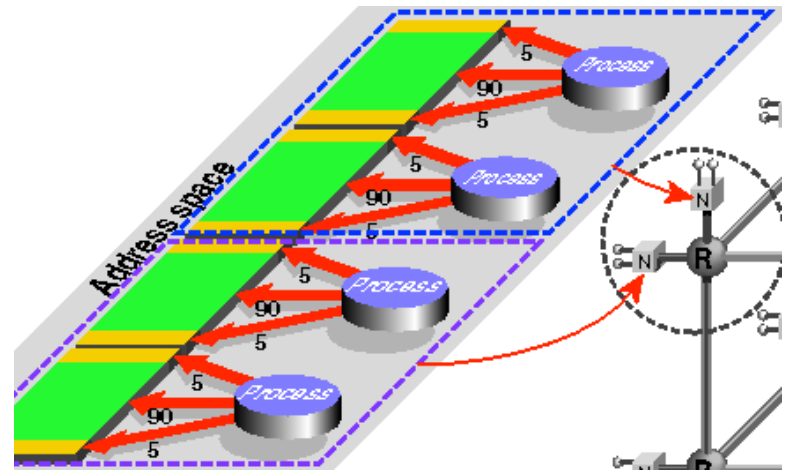
- Sparse matrices
- NUMA
- Example NUMA Systems
 - ◆ Cray XE-6
 - ◆ SGI
- **Performance programming**

Quick primer on paging

- We group the physical and virtual address spaces into units called *pages*
- Pages are backed up on disk
- Virtual to physical mapping done by the Translation Lookaside Buffer (TLB), backs up page tables set up by the OS
- When we allocate a block of memory, we don't need to allocate physical storage to pages; we do it on demand

Remote access latency

- When we allocate a block of memory, which processor(s) is (are) the owner(s)?
- We can control memory locality with the same kind of data layouts that we use with message passing
- Page allocation policies
 - ◆ First touch
 - ◆ Round robin
- Page placement and Page migration
- Copying v. redistribution
- Layout



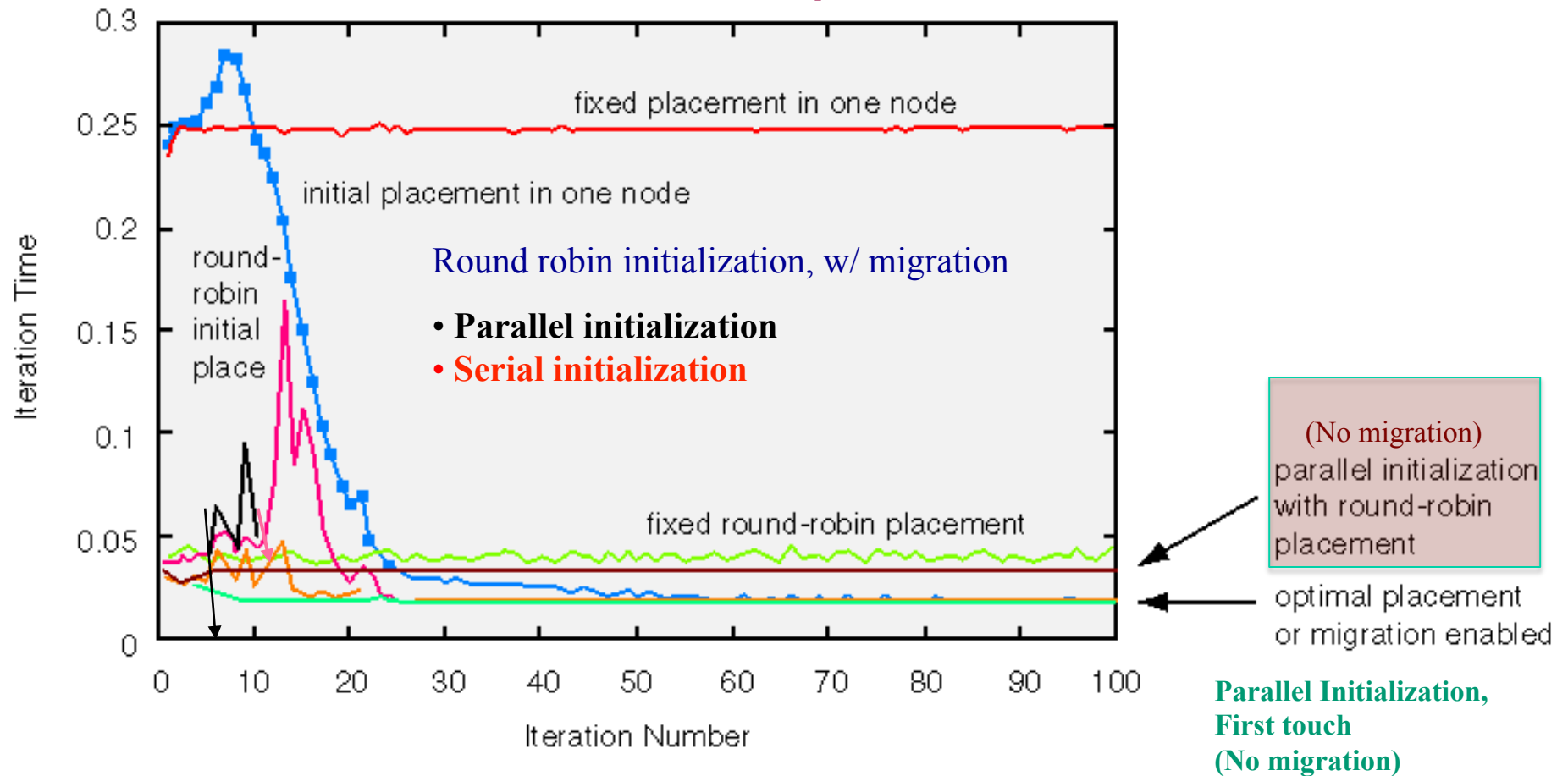
Example

- Consider the following loop

```
for r = 0 to nReps-1
  for i = 0 to n-1
    a[i] = b[i] + q*c[i]
```

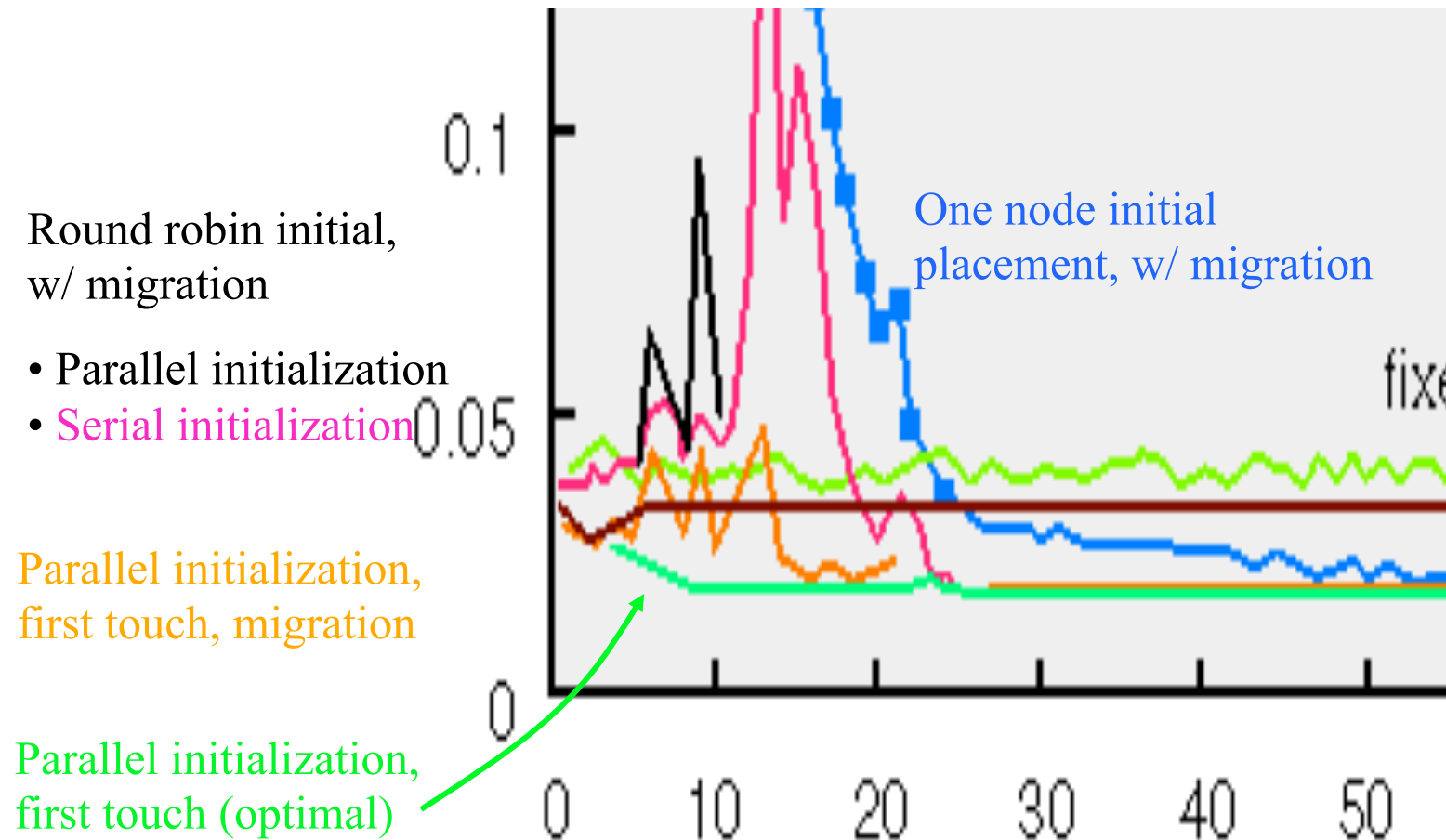
Page Migration

$$a[i] = b[i] + q * c[i]$$

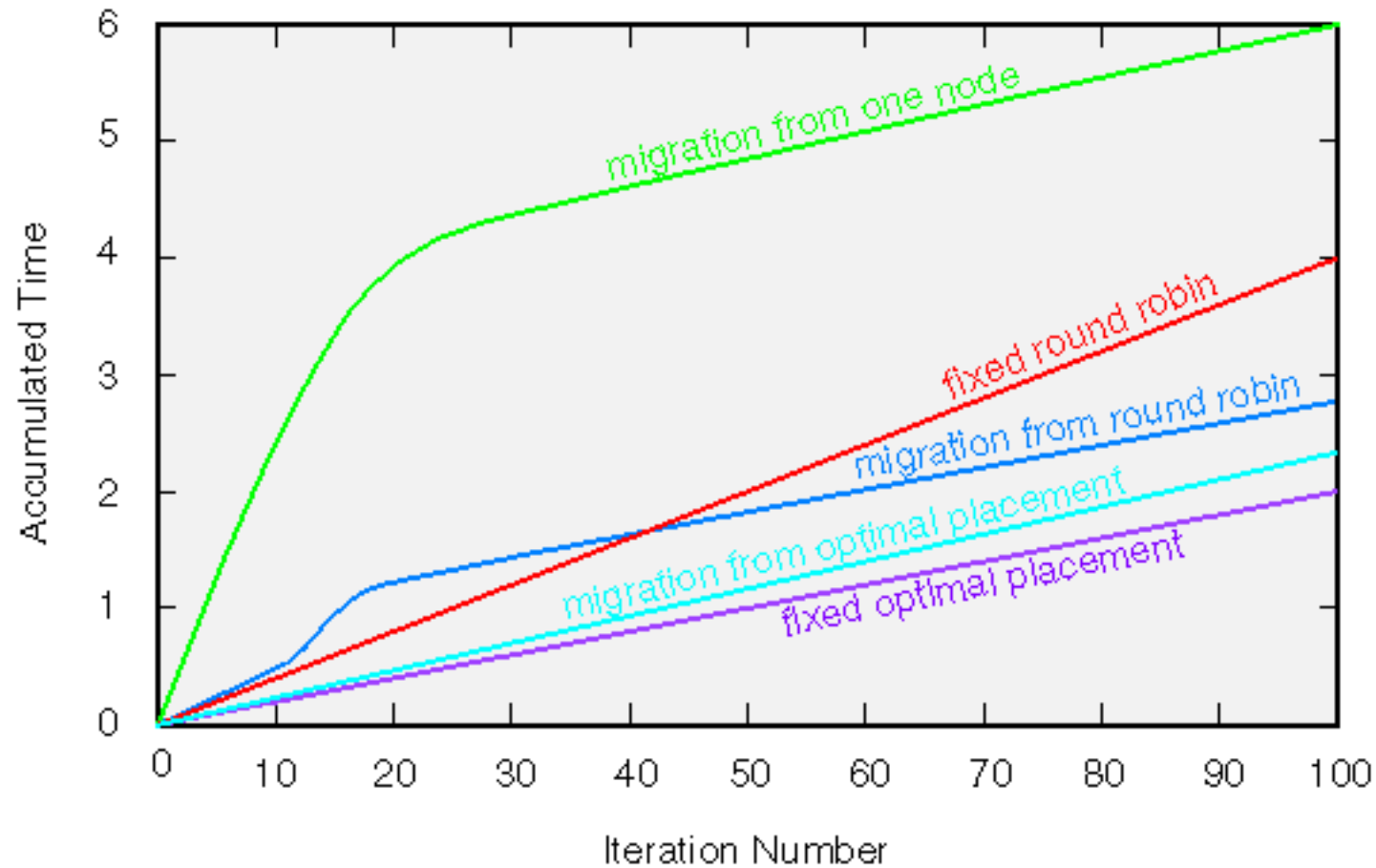


techpubs.sgi.com/library/tp1/cgi-bin/getdoc.cgi/0650/bks/SGI_Developer/books/OrOn2_PfTune/sgi_html/ch08.html#id5224855

Migration eventually reaches the optimal time

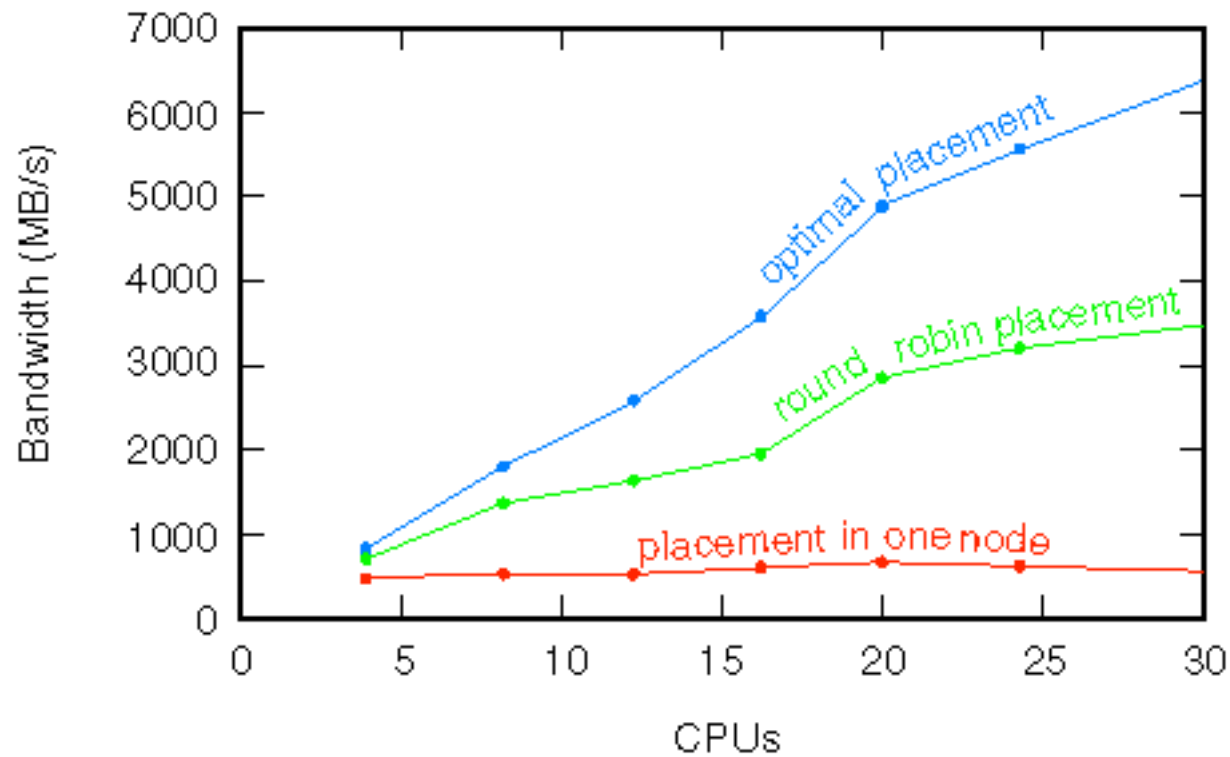


Cumulative effect of Page Migration



Bisection Bandwidth and Latency

$$a(i) = b(i) + q * c(i)$$



Parallelization via the compiler

```
integer n, lda, ldr, lds, i, j, k
real*8 a(lda,n), r(ldr,n), s(lds,n)
!$OMP PARALLEL DO private(j,k,i), shared(n,a,r,s),
schedule(static)
do j = 1, n
  do k = 1, n
    do i = 1, n
      a(i,j) = a(i,j) + r(i,k)*s(k,j)
    enddo
  enddo
enddo
```

AMD Phenom's memory hierarchy

Dedicated L1

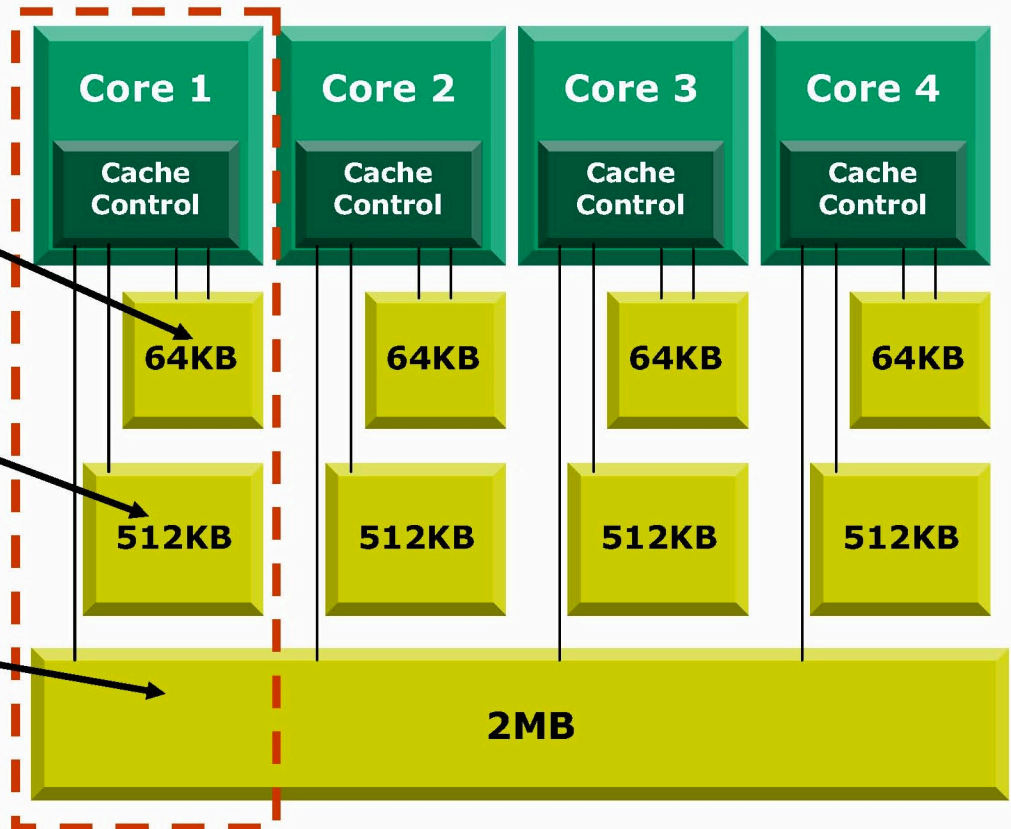
- Locality keeps most critical data in the L1 cache
- Lowest latency
- 2 loads per cycle

Dedicated L2

- Sized to accommodate the majority of working sets today
- Dedicated to eliminate conflicts common in shared caches

Shared L3 – NEW

- Victim-cache architecture maximizes efficiency of cache hierarchy
- Fills from L3 leave likely shared lines in the L3
- Sharing-aware replacement policy



6MB L3 on cseclass 01 and 02

http://developer.amd.com/media/gpu_assets/Develop_Brighton_Justin_Boggs-2.pdf

NUMA awareness

- When we allocate memory, a NUMA processor uses the *first-touch policy*, unless told otherwise
- Be sure to use *parallel for* when initializing data
 $x = \text{new double}[n], y, z = \text{new double}[n];$
`#pragma omp parallel for`
`for (i=0; i< n; i++) {x[i]=0; y[i] = ...; z[i] = ...}`
`#pragma omp parallel for`
`for (i=0; i<n; i++) {x[i] = y[i] + z[i]; }`

