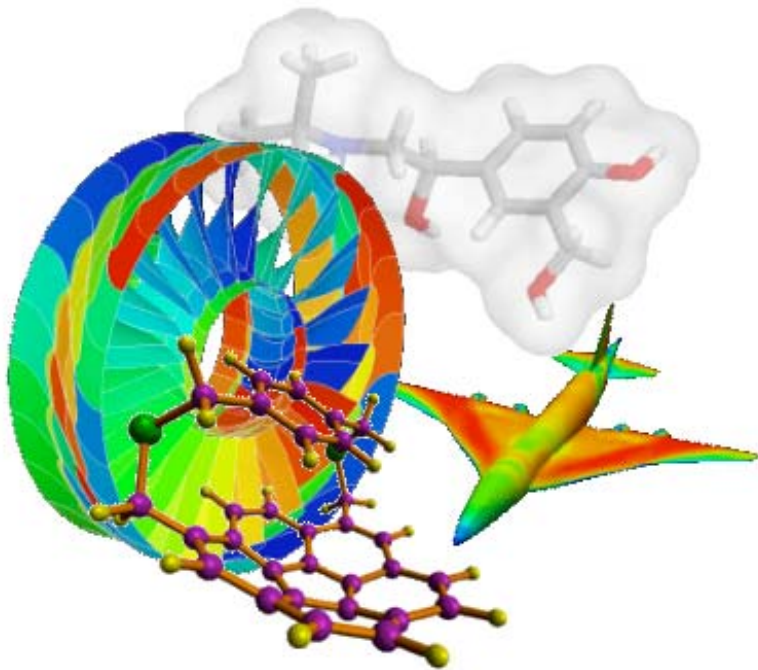


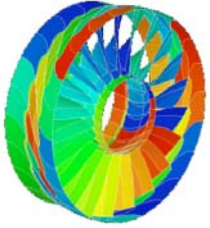
CME342/AA220/CS238 - Parallel Methods in Numerical Analysis

## Shared memory programming III



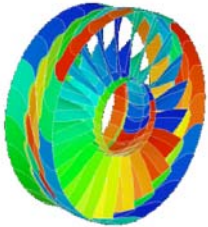
*April 22, 2005*

*Lecture 11*



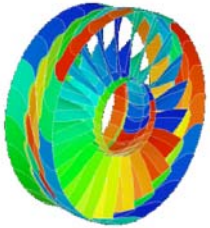
## OpenMP scalability

- **Memory is often the limit to achievable performance of a shared memory program**
- **On scalable architectures, the latency and bandwidth of memory accesses depend on the locality of those accesses**
- **In achieving good speedup of a shared memory program, data locality is an essential element**



## Data distribution - DSM

- **Initial data distribution determines on which node of a Distributed Shared Memory machine the memory is placed**
  - first touch or round-robin system policies
  - data distribution directives
  - explicit page placement
- **Work sharing, e.g., loop scheduling, determines which thread accesses which data**
- **Cache friendliness determines how often main memory is accessed**



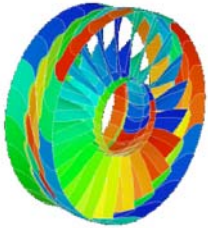
## False sharing

- **Contention is an issue specific to parallel loops, e.g., *false sharing of cache lines***
- **Cache friendliness =**  
**high locality of references**  
**+**  
**low contention**



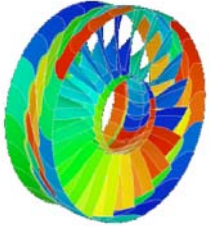
## NUMA

- **Memory hierarchies exist in single-CPU computers and Symmetric Multiprocessors (SMPs)**
- **Distributed shared memory (DSM) machines based on Non-Uniform Memory Architecture (NUMA) add levels to the hierarchy:**
  - *local memory* has low latency
  - *remote memory* has high latency



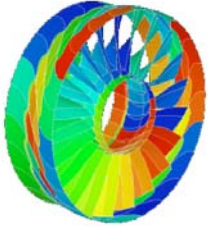
## Origin 2000 memory

<b>Level</b>	<b>Latency (cycles)</b>
register	0
primary cache	2...3
secondary cache	8...10
local main memory & TLB hit	75
remote main memory & TLB hit	250
main memory & TLB miss	2000
page fault	$10^6$



## Page locality

- **An ideal application has full *page locality*: pages accessed by a processor are on the same node as the processor, and no page is accessed by more than one processor (no page sharing)**
- **Twofold benefit:**
  - » **low memory latency**
  - » **scalability of memory bandwidth**



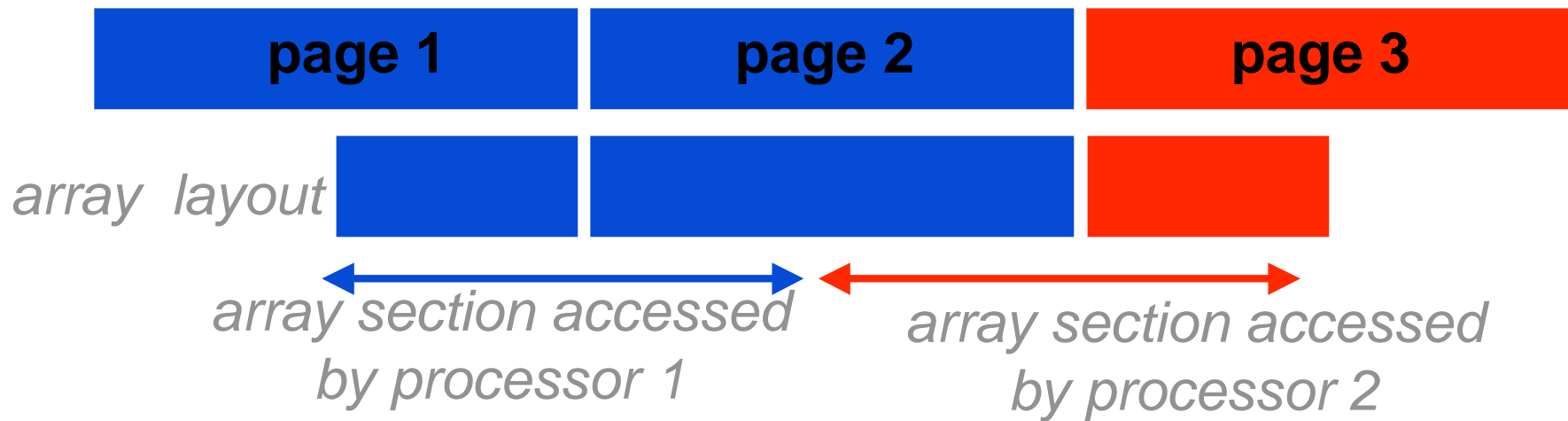
## Page locality

- **The benefits of page locality are more important for programs that are *not* cache friendly**
- **Several data placement strategies for improving page locality**
  - » **system based placement**
  - » **data initialization and directives (system specific, not in the OpenMP standard)**

# Page locality



- Consider an array whose size is twice the size of a page, and which is distributed between two nodes
- Page 1 and page 2 are located on node 1, page 3 is on node 2



- Page 2 is shared by the two processors, due to the array not starting on a page boundary

# Page locality on IRIX



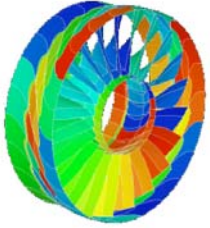
**IRIX has two page placement policies:**

- ***first-touch***: the process which first references a virtual address causes that address to be mapped to a page on the node where the process runs

- ***round-robin***: pages allocated to a job are selected from nodes traversed in round-robin order

**IRIX uses first-touch, unless**

**setenv `_DSM_ROUND_ROBIN`**



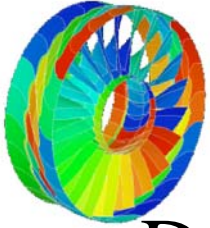
# Achieving page locality

- a) ***Parallel data initialization***, using OpenMP parallel work constructs such as **parallel do**, combined with operating system's *first-touch* placement policy
  - no data distribution directives needed
  - can be used with page migration
- b) **IRIX *round-robin* page placement**
  - no change of code needed
- c) **IRIX page migration**



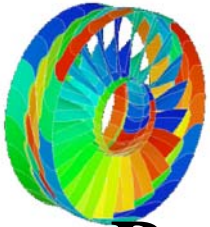
# Achieving good performance

- **Optimize single-CPU performance**
  - maximize cache reuse
  - eliminate cache misses
  - compiler flags
- **Parallelize as high a fraction of the work as possible**
  - preserve cache friendliness
  - avoid synchronization and scheduling overhead:
    - partition in few parallel regions,
    - avoid reduction, single and critical sections,
    - use static scheduling
  - partition work to achieve load balancing
- **Check correctness of parallel code**
  - run OpenMP compiled code first on one thread, then on several threads



## Using an SPMD approach

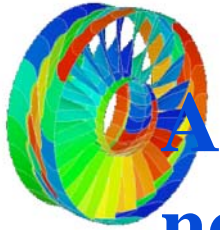
- Data is distributed explicitly among processes
- With message passing, e.g., MPI, where no data is shared, data is explicitly communicated
- Synchronization is explicit or embedded in communication
- With parallel regions in OpenMP, both SPMD and data sharing are supported



## Using an SPMD approach

**Potentially higher parallel fraction than with loop parallelism**

- » The fewer parallel regions, the less overhead
- » More explicit synchronization needed than for loop parallelization
- » Does not promote incremental parallelization and requires manually assigning data subsets to threads

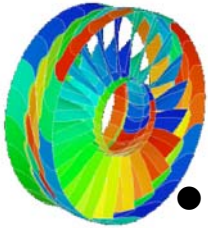


# SPMD example

A single parallel region, no scheduling needed, each thread explicitly determines its work

```
program mat_init
implicit none
integer, parameter::N=1024
real A(N,N)
integer :: iam, np
iam = 0
np = 1
!$omp parallel private(iam,np)
  np = omp_get_num_threads()
  iam = omp_get_thread_num()
! Each thread calls work
  call work(N, A, iam, np)
!$omp end parallel
end
```

```
subroutine work(n, A, iam, np)
integer n, iam, n
real A(n,n)
integer :: chunk,low,high,i,j
chunk = (n + np - 1)/np
low = 1 + iam*chunk
high=min(n,(iam+1)*chunk)
do j = low, high
  do I=1,n
    A(I,j)=sqrt(real(i*i+j*j))
  enddo
enddo
return
```



# Summary of OpenMP constructs

## ● **Parallel Region**

`c$omp parallel`                      `#pragma omp parallel`

## ● **Worksharing**

`c$omp do`                              `#pragma omp for`

`c$omp sections`                      `#pragma omp sections`

`c$single`                              `#pragma omp single`

`c$workshare`                          `#pragma workshare`

## ● **Data Environment**

◆ directive: `threadprivate`

◆ clauses: `shared`, `private`, `lastprivate`, `reduction`, `copyin`, `copyprivate`

## ● **Synchronization**

◆ directives: `critical`, `barrier`, `atomic`, `flush`, `ordered`, `master`

## ● **Runtime functions/environment variables**