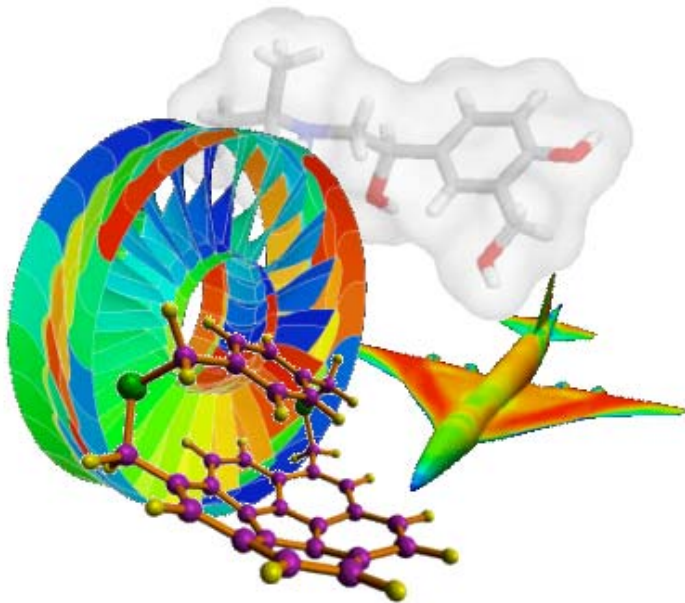


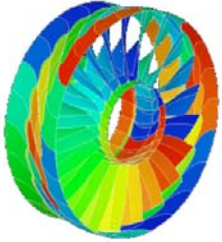
CME342/AA220/CS238 - Parallel Methods in Numerical Analysis

Shared memory programming II



April 20, 2005

Lecture 10

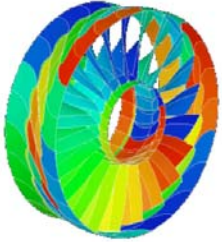


OpenMP

OpenMP is an API for writing multithreaded applications:

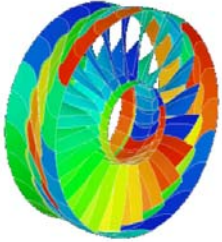
- A set of compiler directives and library routines for parallel application programmers
- Makes it easy to create multithreaded programs in Fortran, C and C++
- Supported by major hardware and software vendors

<http://www.openmp.org>



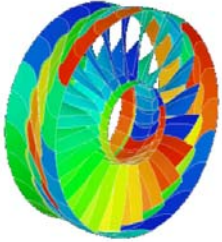
OpenMP

- Fine grained parallelism (at loop level)
- Coarse grained parallelism:
- Compiler directives, library and environment variables *extend* base language
- NOT automatic parallelization
- Since the constructs are directives, an OpenMP program can be compiled by compilers that do not support OpenMP



OpenMP's constructs

- OpenMP's constructs fall in 5 categories:
 - Parallel regions
 - Synchronization
 - Worksharing
 - – Data environment
 - Runtime functions/environment variables



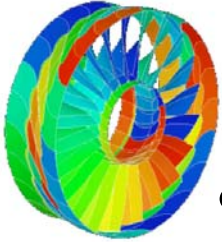
Data scope

SHARED - variable is shared by all processors

PRIVATE - each processor has a private copy of a variable

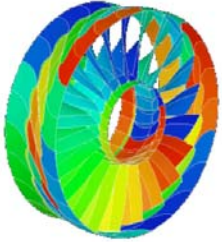
```
!$OMP PARALLEL DO SHARED(A,B,C,N) PRIVATE(I)
  do I=1,N
    C(I) = A(I) + B(I)
  enddo
!$OMP END PARALLEL DO
```

All CPUs have access to the same storage area for A, B, C and N, but each loop needs its own private value of the loop index I.



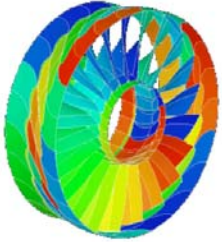
Default Storage Attributes

- Global variables are **SHARED** among threads
 - FORTRAN: **COMMON** blocks, **SAVE** variables, **MODULE** variables
 - C: File scope variables, static
- Stack variables in sub-programs called from parallel regions are **PRIVATE**



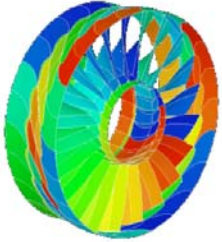
```
program sort
common/input/a(10)
integer index(10)
call input
!$OMP PARALLEL
    call work
!$OMP END PARALLEL
print *,index(1)
....
```

```
subroutine work
common/input/a(10)
real dummy(10)
integer count
save count
.....
```



Storage Attributes

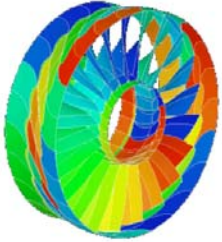
- Storage attributes can be changed using the following clauses:
 - SHARED
 - PRIVATE
 - FIRSTPRIVATE
 - THREADPRIVATE
- The value of a PRIVATE variable inside a parallel loop can be transmitted to a global value outside the loop with:
 - LASTPRIVATE
- The default status can be modified with:
 - DEFAULT (PRIVATE | SHARED | NONE)



PRIVATE clause

- PRIVATE(var) creates a local copy of var for each thread
 - The value is uninitialized
 - Private copy is not storage associated with the original

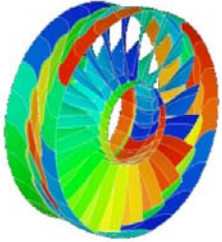
```
program wrong
  is=0
C$OMP PARALLEL DO PRIVATE(IS)
  do j=1,1000
    is=is+j    !is is not initialized
  end do
C$OMP END PARALLEL DO
  print *,is  !this is still the original is
```



FIRSTPRIVATE clause

- `FIRSTPRIVATE(var)` creates a local copy of `var` for each thread with the corresponding value from the master thread

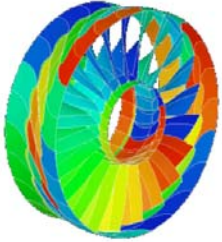
```
program wrong2
  is=0
  C$OMP PARALLEL DO FIRSTPRIVATE(IS)
    do j=1,1000
      is=is+j      ! is now 0
    end do
  C$OMP END PARALLEL DO
  print *,is      ! still the original is
```



LASTPRIVATE clause

- LASTPRIVATE(var) passes the value of a PRIVATE variable from the last iteration to a global variable

```
program wrong3
  is=0
  C$OMP PARALLEL DO FIRSTPRIVATE(IS)
  C$OMP* LASTPRIVATE(IS)
    do j=1,1000
      is=is+j      !is is now 0
    end do
  C$OMP END PARALLEL DO
  print *,is
```



Example

```
program wrong3
  is=0
  C$OMP PARALLEL DO FIRSTPRIVATE(IS)
  C$OMP* LASTPRIVATE(IS)
  do j=1,1000
    is=is+j      !is is now 0
  end do
  C$OMP END PARALLEL DO
  print *,is
```

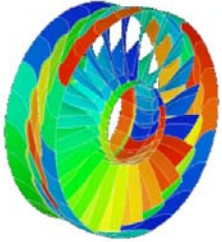
Serial output

```
junior:~> a.out
          500500
```

Parallel output:

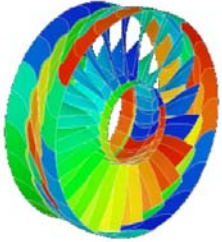
```
junior:~> setenv OMP_NUM_THREADS 2
junior:~> a.out
          375250
```

```
junior:~> setenv OMP_NUM_THREADS 4
junior:~> a.out
          218875
```



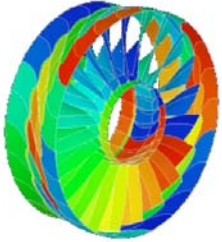
REDUCTION clause

- REDUCTION(op: list) :local copies are reduced into a single global copy at the end of the construct using the specified operation
 - The variables in list must be shared.
 - Inside a parallel or worksharing construct a local variable is made and properly initialized.



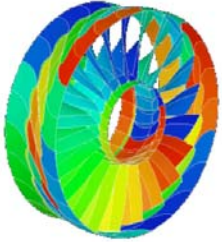
THREADPRIVATE clause

- Make global data private to a thread:
 - FORTRAN: COMMON blocks
 - C: file scope and static variables
 - Each thread has its own copy of the common block.



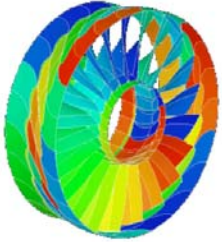
OpenMP's constructs

- OpenMP's constructs fall in 5 categories:
 - Parallel regions
 - Synchronization
 - Worksharing
 - Data environment
 - – Runtime functions/environment variables



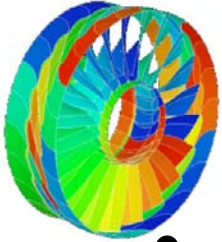
Runtime functions

- Run time library can be used to control and query the parallel execution environment
 - OMP_SET_NUM_THREADS()
 - OMP_GET_NUM_THREADS()
 - OMP_GET_THREAD_NUM()
 - OMP_GET_NUM_PROCS()
 - OMP_IN_PARALLEL(): *return true or false*



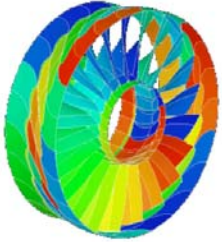
Programming errors

- Shared memory parallel programming is a mixed bag
 - It saves programmer from mapping data onto multiple processors
 - It may introduce new errors coming from unanticipated shared resource conflicts



Common SMP errors

- Race conditions:
 - The outcome of the program is dependent on the detailed timing of the threads (or the number of threads)
- Deadlock
 - Threads lock up waiting on a locked resource that will never become available
- Livelock
 - Multiple threads working individual tasks which the ensemble can't finish



Race conditions

- The result varies unpredictably based on detailed order of execution for each section
- Wrong answers produced without warning

```
C$OMP PARALLEL SECTIONS
```

```
  A=B+C
```

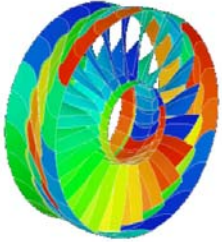
```
C$OMP SECTION
```

```
  B=A+C
```

```
C$OMP SECTION
```

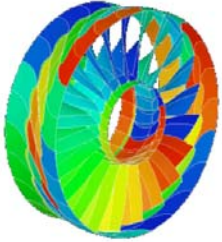
```
  C=B+A
```

```
C$OMP END PARALLEL SECTIONS
```



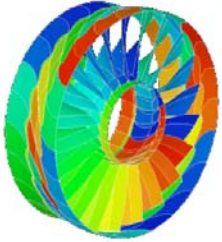
OpenMP traps

- Are you using thread-safe libraries?
- I/O inside a parallel region can interleave unpredictably
- Private variables can mask global ones
- Understand when shared memory is coherent.
When in doubt use FLUSH
- NOWAIT removes implied barriers



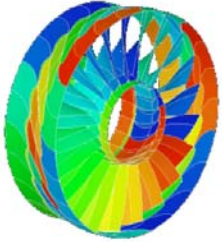
Portable Sequential Equivalence

- What is Portable Sequential Equivalence (PSE)?
 - A program is sequentially equivalent if its results are the same with one thread or many threads
 - For a program to be portable (runs the same on different platforms/compilers) it must execute identically when the OpenMP constructs are used or ignored
- Strong SE: bitwise identical results
- Weak SE: equivalent mathematically, not bitwise identical



Portable Sequential Equivalence

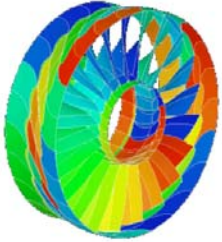
- Strong SE:
 - Locate all cases where a shared variable can be written by multiple threads
 - The access to the variable must be protected
 - If multiple threads combine results into a single value, enforce sequential order
- Weak SE:
 - Floating point arithmetic is not associative and not commutative
 - In most cases no particular grouping is mathematically preferred so why choose the sequential order?



Example

The summation into RES occurs one thread at a time, but in any order so the result is not bitwise equivalent to the sequential one.

```
C$OMP PARALLEL PRIVATE(I,TMP)
C$OMP DO
    DO I=1,NDIM
        TMP=FOO(I)
C$OMP CRITICAL
        CALL COMBINE(TMP,RES)
C$OMP END CRITICAL
    END DO
C$OMP END PARALLEL
```



OpenMP or MPI?

- Do you need total portability?
MPI
- Do you need to use hundreds of processors?
MPI
- Do you have access to DSM memory machines?
OpenMP
- Do you want to be ready for the next generation of computers?
MPI and OpenMP