OpenMP Data Scoping (continued)

- **Data scoping (continued)**
  - FIRSTPRIVATE
    - Like PRIVATE, but copies are initialized using value from master thread's copy
  - LASTPRIVATE
    - Like PRIVATE, but final value is copied out to master thread's copy
    - For DO: last iteration; SECTIONS: last section
  - *THREADPRIVATE*
    - *Procedures called within a thread share data that is not visible to other threads*
    - *Provides a way of declaring common blocks that are specific to a thread*

C OpenMP API

- **Functionality provided is similar to that for Fortran**
  - Include file: #include <omp.h>
  - Format is a pragma
    - #pragma omp parallel for schedule(static) private(j)
    - #pragma omp parallel
    - #pragma omp critical
  - Pragmas apply to a block of code, no “end” pramgas

C OpenMP API (continued)

- Some differences in spellings

```
#pragma omp parallel for private(j)
for(i=0; i < 100; i++){
  j = f(i);
  a[i] = j*j;
 }
```

- Nested/recursive locks allowed
  - same thread can set acquire a lock arbitrary number of times
- Threadprivate makes file-scope variable local to a thread

C OpenMP API (continued)

- No "default(private)" clause
- Conditional compilation only allowed via the _OPENMP macro

C Examples

```
#pragma omp parallel
#pragma omp critical
        {
                thread_bind();
        }


#pragma omp parallel for schedule(static) private(j)
    for( i=i1;i<=i2;i++)
        for(j=j1;j<=j2;j++)
          psi[i][j]=new_psi[i][j];
}
```

OpenMP Data Scoping (continued)

- **Data scoping (continued)**
  - FIRSTPRIVATE
    - Like PRIVATE, but copies are initialized using value from master thread's copy
  - LASTPRIVATE
    - Like PRIVATE, but final value is copied out to master thread's copy
    - For DO: last iteration; SECTIONS: last section

OpenMP Data Scoping (continued)

```
!$OMP PARALLEL DO LASTPRIVATE(TEMP)
      DO I = 1, 100
        TEMP = F(I)
      END DO
      PRINT *, 'F(100) == ', TEMP
      ! TEMP is equal to F(100) at end of loop
```

OpenMP Data Scoping (continued)

```
 I = 17
!$OMP PARALLEL SECTIONS FIRSTPRIVATE(I)
!$OMP SECTION
      ! Each thread has its own I variable,
      ! and I is equal to 17 here
      DO J = I, 100
        B(J) = J
      END DO
!$OMP SECTION
      DO I = 1, 100
        SUM = SUM + A(I)
      END DO
!$OMP END PARALLEL SECTIONS
```

OpenMP Synchronization Directives

- **Synchronization and MASTER constructs**
  - MASTER
    - Only the master thread executes the enclosed block of code
  - CRITICAL
    - Only one thread at a time executes the enclosed block
    - May specify a name that is common to multiple CRITICALs

```
!$OMP PARALLEL PRIVATE(MYDATA)
!$OMP   CRITICAL(MYSECT)
        READ *, MYDATA
!$OMP   END CRITICAL(MYSECT)
      ...
!$OMP END PARALLEL
```

OpenMP Synchronization Directives (continued)

- **BARRIER**
  - Threads in a team wait until entire team reaches the barrier

```
!$OMP PARALLEL
!$OMP DO REDUCTION(+:S)
      DO I = 1, 100
        S = S + F(I)
      END DO
!$OMP END DO NOWAIT
      ...
!$OMP BARRIER
       ! Can now use S
!$OMP END PARALLEL
```

OpenMP Synchronization Directives (continued)

- **ORDERED**

  The enclosed block of code is executed in per iteration order

```
!$OMP PARALLEL DO ORDERED SCHEDULE(DYNAMIC)
      DO I = 1, 100
        IF(A(I) > 100.0) THEN
!$OMP     ORDERED
            S = S + A(I)
!$OMP END ORDERED
        END IF
      END DO
```

OpenMP Synchronization Directives (continued)

- **ATOMIC**
  - Specifies an atomic update of a variable can be used, if available

```
!$OMP PARALLEL SHARED(S)
        ...
!$OMP ATOMIC
      S = S + 1.0
        ...
!$OMP END PARALLEL
```

OpenMP Synchronization Directives (continued)

- **FLUSH example**

```
!$OMP PARALLEL SHARED(WORK_READY)
      IF (OMP_GET_THREAD_NUM() == 0) THEN
        CALL SETUP_WORKQUEUES
        WORK_READY = .TRUE.
!$OMP    FLUSH(WORK_READY)
      ELSE
100     CONTINUE
!$OMP     FLUSH(WORK_READY)
       IF(.NOT. WORK_READY) GO TO 100
      CALL START_WORKING
      END IF
!$OMP END PARALLEL
```

OpenMP Synchronization Directives (continued)

- **FLUSH {, list-of-vars}**
  - Forces memory synchronization, flushing/loading of values of variables to/from memory, etc.
  - Can build broadcast or point-to-point communication primitives

OpenMP Conditional Compilation

- **Conditional compilation**
  - Introduced with a !$, C$ or *$ trigger

```
!$OMP PARALLEL PRIVATE(I)
!$      I = 0
        CALL SUB(I)
```

  - !$OMP END PARALLEL
  - Or can use _OPENMP macro in cpp

```
!$OMP PARALLEL PRIVATE(I)
#idef _OPENMP
        I = 0
#endif
!$OMP END PARALLEL
```

OpenMP Environment Variables

- **Environment variables**
  - OMP_SCHEDULE
    - specifies the default scheduling algorithm to use for DO
  - OMP_NUM_THREADS
    - specifies the default number of threads to use
  - OMP_DYNAMIC
    - specifies whether the number of threads can be changed automatically at execution time by the implementation

OpenMP Environment Variables (continued)

- **OMP_NESTED**
  - permits or disables nested parallelism
  - when disabled, a parallel region encountered while another region is active creates a new team consisting of a single thread

OpenMP Library

- **Library procedures**
  - Execution environment procedures
    - Used to control and query the parallel environment
    - OMP_SET_NUM_THREADS, OMP_GET_NUM_THREADS, OMP_GET_MAX_THREADS, OMP_GET_THREAD_NUM, OMP_GET_NUM_PROCS, OMP_IN_PARALLEL, OMP_SET_DYNAMIC, OMP_GET_DYNAMIC, OMP_SET_NESTED, OMP_GET_NESTED

OpenMP Library (continued)

- **Lock procedures**
  - Used to implement locks at a lower, more flexible, level than CRITICAL construct
  - OMP_INIT_LOCK, OMP_DESTROY_LOCK, OMP_SET_LOCK, OMP_UNSET_LOCK, OMP_TEST_LOCK