OpenMP History

- **What is it?**
  - Extensions to Fortran, C and C++ for portable SMP programming
    - higher-level than POSIX threads
  - Mainly Fortran comment directives and C/C++ pragmas
    - can be ignored by a non-OpenMP compiler
- **Why is it?**
  - Many compiler vendors (particularly Fortran) had vendor-specific SMP directive support
    - not portable and with subtle differences in meaning
  - Previous attempts at standardization had failed

OpenMP History (continued)

- **Primary OpenMP participants**
  - Compaq, HP, IBM, Intel, KAI, SGI, Sun
- **U.S. Dept. of Energy ASCI Program**
- **OpenMP Fortran API, Version 1.0, published Oct. 1997**
- **OpenMP C API, Version 1.0, published Oct. 1998**

OpenMP Directives

- **PARALLEL regions**
  - Most basic parallel construct
  - Code enclosed by PARALLEL/END PARALLEL is executed redundantly by a "team" of threads
    - Can be basis for coarse-grained parallelism
  - Initial thread that encountered directive becomes "master" of team

    - !$OMP PARALLEL PRIVATE(CHUNK,LB,UB), SHARED(A)
    - CHUNK = ...; LB = ...; UB = ...
    - CALL WORK(A(LB:UB))
    - !$OMP END PARALLEL

  - IF clause condition determines whether to execute in parallel

OpenMP Work-sharing Directives

- **Work-sharing constructs**
  - Contains block of code that threads of a PARALLEL region execute co-operatively
    - All threads must encounter work-sharing at same time
  - DO (Fortran) / FOR (C)
    - Iterations of loop are split amongst threads, in chunks determined by a scheduling algorithm (user-specified or default)

```
!$OMP DO PRIVATE(S) SHARED(A, B)
      DO I = 1, 100
        S = B(I)
        A(I) = S*(S+1)
      END DO
!$OMP END DO
```

OpenMP Work-sharing Directives (continued)

- Work-sharing DO scheduling algorithms
  - Designed to balance work load amongst threads or maintain cache locality
  - Work is assigned to threads in chunks

  - !$OMP DO SCHEDULE(schedule-type[, chunk-size])
  - DO I = 1, 100
  - ...
  - END DO

OpenMP Work-sharing Directives (continued)

- STATIC
  - Threads receive chunks of iterations in thread order, round-robin
  - Good if every iteration contains same amount of work
  - May help keep parts of an array in a particular processor's cache
- DYNAMIC
  - Thread receives chunks as thread becomes available for more work
  - Default chunk size is 1
  - Good for load-balancing

OpenMP Work-sharing Directives (continued)

- GUIDED
  - Thread receives chunks as the thread becomes available for work
  - Chunk size decreases exponentially, until it reaches the chunk size specified (default is 1)
  - Balances load and reduces number of requests for more work
- RUNTIME
  - Schedule is determined at run-time by OMP_SCHEDULE
  - Useful for experimentation

OpenMP Work-sharing Directives (continued)

- **Ex: loop with 100 iterations and 4 threads**
  - SCHEDULE(STATIC)

Thread	0	1	2	3
Iteration	1-25	26-50	51-75	76-100

  - SCHEDULE(DYNAMIC, 15)

Thread	0	1	3	2	1	1	2
Iteration	1-15	16-30	31-45	46-60	61-75	76-90	91-100

  - SCHEDULE(GUIDED, 8)

Thread	0	1	2	3	3	2	3	1
Iteration	1-25	26-44	45-58	59-69	70-77	78-85	86-93	94-100

OpenMP Work-sharing Directives (continued)

- **SECTIONS**
  - Blocks of code are split amongst threads - task parallel style
  - A thread might execute more than one block or no blocks

```
!$OMP SECTIONS
!$OMP SECTION
      CALL TASK1
!$OMP SECTION
      CALL TASK2
!$OMP SECTION
      CALL TASK3
!$OMP END SECTIONS
```

OpenMP Work-sharing Directives (continued)

- **SINGLE**
  - Block of code is executed by a single thread

```
!$OMP SINGLE
      GLOBAL_CNTR = GLOBAL_CNTR + 1
      PRINT *, GLOBAL_CNTR
!$OMP END SINGLE
```

OpenMP Work-sharing Directives (continued)

- **NOWAIT clause**
  - Normally threads encounter a barrier synchronization at end of work-sharing construct
  - NOWAIT on END DO/END SECTIONS/END SINGLE specifies that threads that complete assigned work can proceed

OpenMP Combined Directives

- **Combined directives**
  - PARALLEL DO and PARALLEL SECTIONS
  - Same as PARALLEL region containing only DO or SECTIONS work-sharing construct

```
!$OMP PARALLEL DO
      DO I = 1, 100
        A(I) = B(I)
      END DO
```

OpenMP Data Scoping

- **Data scoping**
  - PRIVATE
    - Each thread has its own copy of the specified variable
    - Variables are undefined after worksharing region
  - SHARED
    - Threads share a single copy of the specified variable
  - DEFAULT
    - A default of PRIVATE, SHARED or NONE can be specified
    - No default(private) in C)
    - Note that loop counters are always PRIVATE by default; everything else is SHARED by default

OpenMP Data Scoping (continued)

```
REAL :: A(100), T1, T2
!$OMP PARALLEL DO SHARED(A),
  PRIVATE(T1,T2)
    DO I = 1, 100
      T1 = F(I); T2 = G(I)
      A(I) = SQRT(T1**2 + T2**2)
    END DO
    ! Each thread has private copy of T1 & T2
    ! Threads share A - write to distinct
    ! elements
```

OpenMP Data Scoping (continued)

- **Data scoping (continued)**
  - REDUCTION
    - Each thread has its own copy of the specified variable
    - Can appear only in reduction operation
    - All copies are "reduced" back into the original master thread's variable

OpenMP Data Scoping (continued)

```
SUM = 0
!$OMP PARALLEL DO REDUCTION(+:SUM)
      DO I = 1, 100
        SUM = SUM + A(I)
      END DO
! Each thread's copy of SUM is added
! to original SUM at end of loop
```