2068-17

Advanced School in High Performance and GRID Computing - Concepts and Applications

30 November - 11 December, 2009

Introduction to OpenMP

A. Kohlmeyer

University of Pennsylvania
Philadelphia
USA

Introduction to OpenMP

Advanced School in High Performance and GRID Computing

Axel Kohlmeyer

ICMS

Institute for Computational Molecular Science

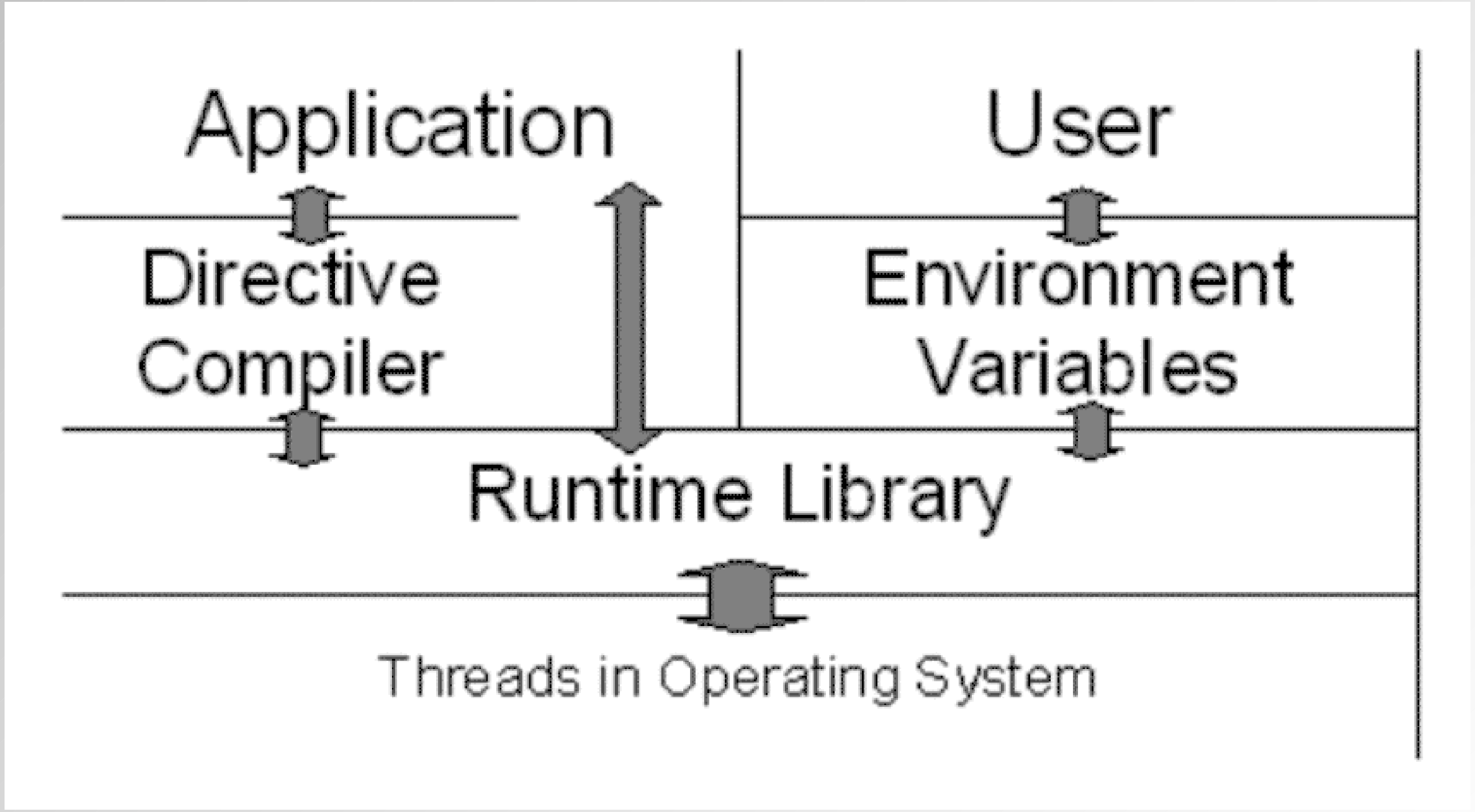
OpenMP

TEMPLE UNIVERSITY®

Overview

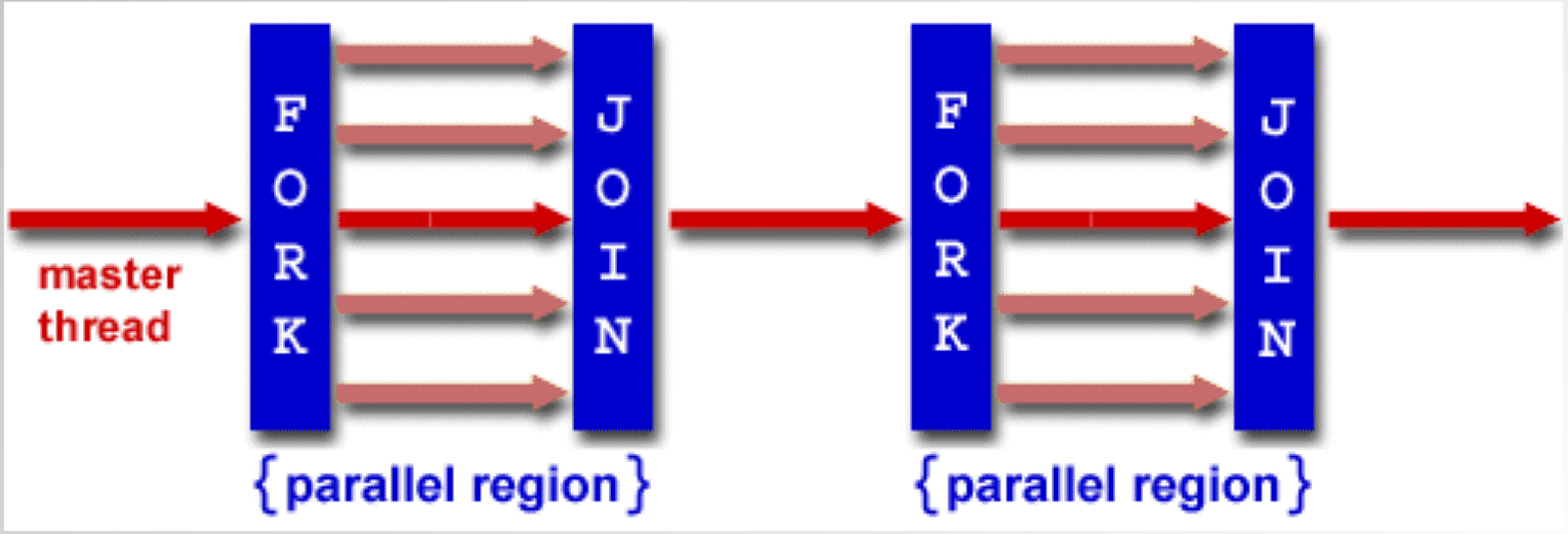
- Fine grained (loop) parallelism
- For shared memory SMP machines
- Directive based parallelization:
  Code should compile unaltered in serial mode
- Fortran 77/95 and C/C++ interface
- Incrementally parallelize a serial program
- Independent from and orthogonal to MPI
- http://www.openmp.org

Architecture



Execution Model

Fork-Join model on thread based machines



Directives Example: Fortran

```
          PROGRAM HELLO
          INTEGER VAR1, VAR2, VAR3
      Serial code
          ...
!$OMP PARALLEL PRIVATE(VAR1,VAR2) SHARED(VAR3)
      Section executed in parallel by multiple threads
          ...
!$OMP END PARALLEL
      Resume serial code
          END
```

Directives Example: C

```
#include <omp.h>
int main(int argc, char **argv) {
    int var1, var2, var3;
Serial code
#pragma omp parallel private(var1,var2) shared(var3)
  {
 Section executed in parallel by multiple threads
  }
Resume serial code
   return 0;
}
```

OpenMP

Parallel Region

```
      PROGRAM HELLO
      INTEGER NTHREADS, TID, OMP_GET_NUM_THREADS,
     +   OMP_GET_THREAD_NUM
!$OMP PARALLEL PRIVATE(TID)
     TID = OMP_GET_THREAD_NUM()
     PRINT *, 'Hello World from thread = ', TID
     IF (TID .EQ. 0) THEN
       NTHREADS = OMP_GET_NUM_THREADS()
       PRINT *, 'Number of threads = ', NTHREADS
     END IF
!$OMP END PARALLEL
      END
```

Loop Parallelization

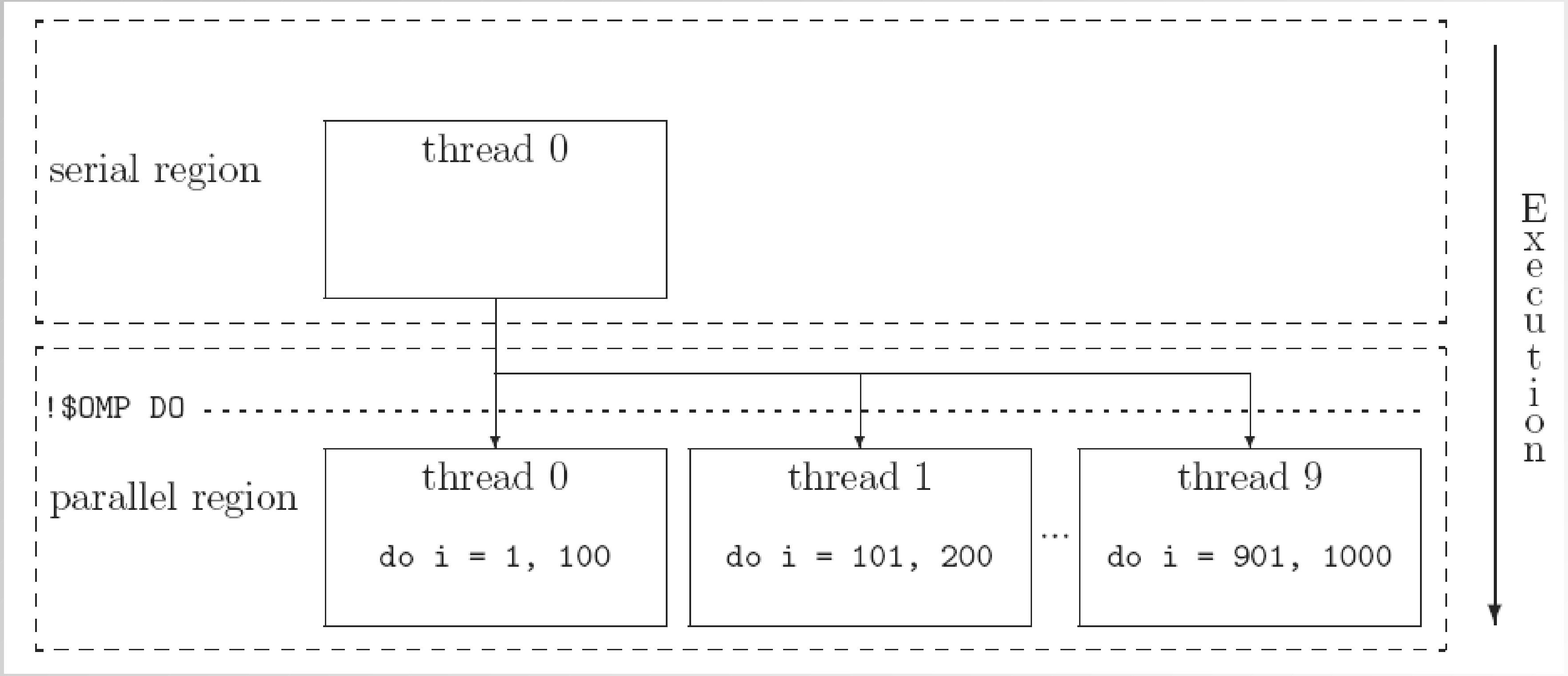
```
      PROGRAM VEC_ADD_DO
      INTEGER I
      REAL*8 A(1000), B(1000), C(1000)
      DO I = 1, 1000
        A(I) = I * 1.0d0
        B(I) = A(I)*2.0d0
      ENDDO
!$OMP PARALLEL DO SHARED(A,B,C) PRIVATE(I)
      DO I = 1, 1000
         C(I) = A(I) + B(I)
      ENDDO
!$OMP END PARALLEL
      END
```

Remove data dependency between threads. Each thread will have its own copy of “I”.

Outside of the parallel region the value of “I” is undefined.

OpenMP

Loop Parallelization, cont'd



Reduction Operation

```
      PROGRAM VEC_ADD_DO
      INTEGER I
      REAL*8 A(1000), B
      DO I = 1, 1000
        A(I) = I * 1.0d0
      ENDDO
!$OMP PARALLEL DO SHARED(A) PRIVATE(I) REDUCTION(+:B)
      DO I = 1, 1000
         B = B + A(I)
      ENDDO
!$OMP END PARALLEL
      END
```

Each thread will do part of the sum and the result from the threads will be combined into one final sum.

Non-Parallelizable Operation

```
      PROGRAM VEC_ADD_DO
      INTEGER I
      REAL*8 A(1000),B(1000),C(1000)
      ...
!$OMP PARALLEL DO SHARED(A,B) PRIVATE(I)
      DO I = 2, 999
        C(I) = 0.25d0*(A(I-1)+A(I+1))-0.5d0*A(I)
        B(I) = 0.25d0*(C(I-1)+C(I))+0.5d0*A(I)
      ENDDO
!$OMP END PARALLEL
      END
```

A step of the iteration depends of the result of a previous step, but with threading, we cannot know if that result is already available.

Race Condition

```
#if defined(_OPENMP)
#pragma omp parallel for default(shared) schedule(static)\
            private(i,j) reduction(+:epot)
#endif
    for(i=0; i < natoms-1; ++i) {
        for(j=i+1; j < natoms; ++j) {
            d=r[j] - r[j];
            d=d*d;
            if (d < rcutsq) {
                rinv = 1.0/sqrt;
                r6=rinv*rinv*rinv;
                ffac = (12.0*c12*r6 - 6.0*c6)*r6*rinv;
                epot += r6*(c12*r6 - c6);
                f[i] += ffac;
                f[j] -= ffac;
            }
        }
    }
```

The “i” loop index will be distributed across multiple threads, so the “j” on some thread may be the same number as “j” or “i” on some other thread.

Race Condition

```
#pragma omp parallel for default(shared) schedule(static)\
            private(i,j) reduction(+:epot)
    for(i=0; i < natoms-1; ++i) {
        for(j=i+1; j < natoms; ++j) {
            d=r[j] - r[j];
            d=d*d;
            if (d < rcutsq) {
                rinv = 1.0/sqrt;
                r6=rinv*rinv*rinv;
                ffac = (12.0*c12*r6 - 6.0*c6)*r6*rinv;
                epot += r6*(c12*r6 - c6);
#pragma omp critical
              {
                f[i] += ffac;
                f[j] -= ffac;
              }
            }
        }
    }
```

The critical directive will guarantee, that only one thread at a time, will execute this part of the code.
Problem: not parallel => slow

Race Condition Cont'd

```
#pragma omp parallel for default(shared) schedule(static)\
            private(i,j) reduction(+:epot)
    for(i=0; i < natoms-1; ++i) {
        for(j=i+1; j < natoms; ++j) {
            d=r[j] - r[j];
            d=d*d;
            if (d < rcutsq) {
                rinv = 1.0/sqrt;
                r6=rinv*rinv*rinv;
                ffac = (12.0*c12*r6 - 6.0*c6)*r6*rinv;
                epot += r6*(c12*r6 - c6);
#pragma omp atomic
                f[i] += ffac;
#pragma omp atomic
                f[j] -= ffac;
            }
        }
    }
```

The “atomic” directive will protect a single memory location. Much less overhead than “critical”, but requires support from processor hardware.

Race Condition Cont'd

```
#pragma omp parallel for default(shared) schedule(static)\
              private(i,j) reduction(+:epot)
     for(i=0; i < natoms; ++i) {
          for(j=0; j < natoms; ++j) {
               If (i==j) continue;
               d=r[j] - r[j];
               d=d*d;
               if (d < rcutsq) {
                    rinv = 1.0/sqrt;
                    r6=rinv*rinv*rinv;
                    ffac = (12.0*c12*r6 - 6.0*c6)*r6*rinv;
                    epot += 0.5*r6*(c12*r6 - c6);
                    f[i] += ffac;
                 }
               }
          }
     }
```

The race condition can be completely avoided by changing the loop, but now we have twice the compute work to do. Overall, this is **still** faster.

How To Activate OpenMP

- Compile with special flags:
  - Intel: -openmp
  - PGI: -mp
  - GNU: -fopenmp
- Set number of threads:
  - Environment: $OMP_NUM_THREADS
  - Function: omp_set_num_threads()
  - Implementation default
- For optimal performance, use with threaded, <u>and</u> re-entrant BLAS/LAPACK library (MKL)

OpenMP vs. MPI

- OpenMP does not require code layout change... in principle, but it may help a lot
- Thread creation/delete overhead on SMP
- OpenMP requires shared memory
- Fine grained, local changes
- MPI + OpenMP = 2-level parallelization most efficient on cluster of SMP nodes
- No MPI call within OpenMP block