OSSCOM

Partnership with Enterprises Towards Building Open Source Software Communities and Rejuvenation of Technical Education and Innovation

Grant agreement no.: No 544520-TEMPUS-1-2013-1-DE-TEMPUS-JPHES

Parallel Programming Using OpenMP

Enrique Arias, UCLM

Beirut, Lebanon, 3rd December 2016

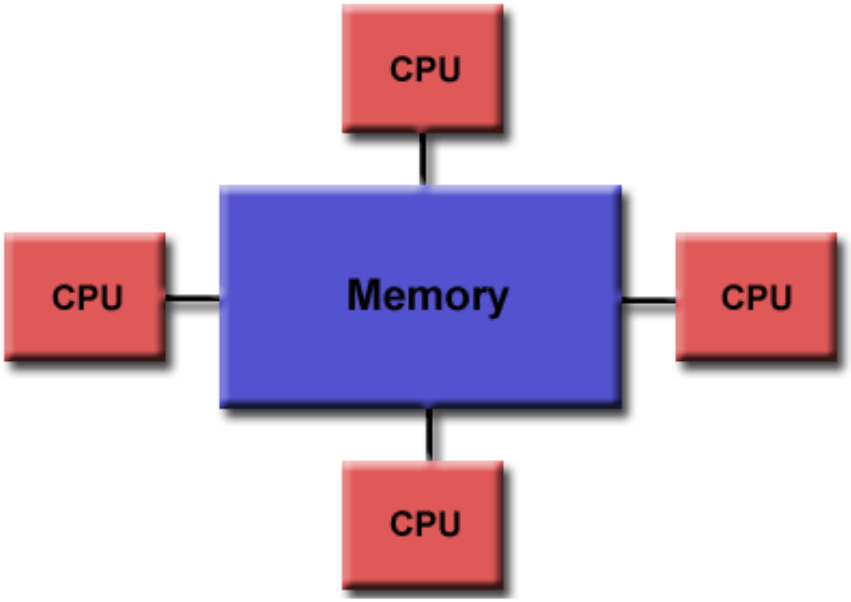
UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Tempus

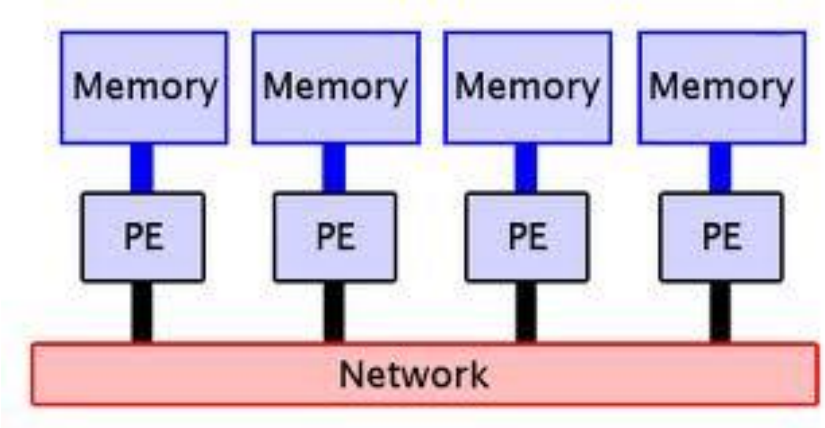
Contents

Contents

Parallel Platforms



Shared



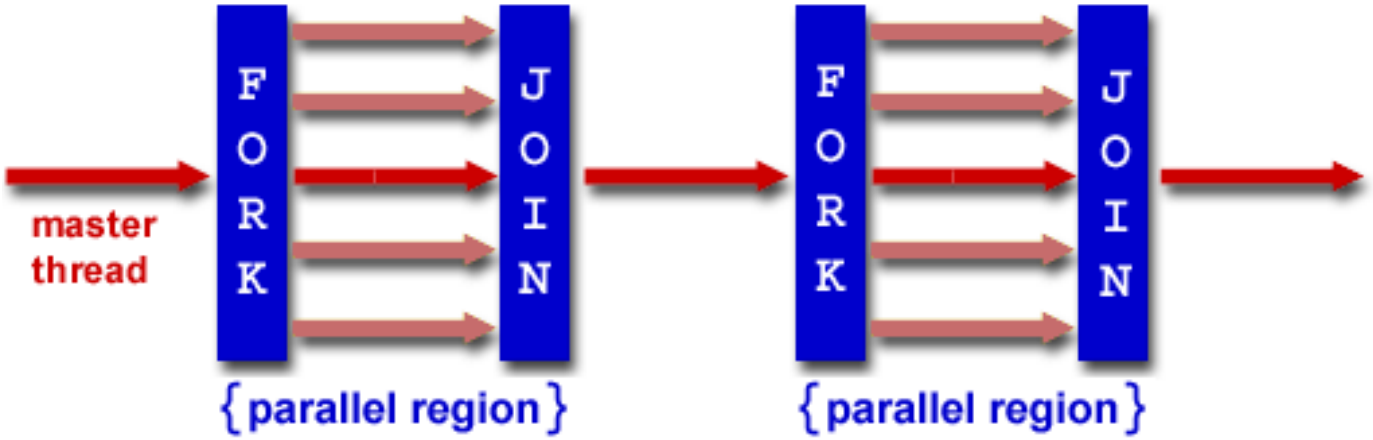
Distributed

What Is OpenMP?

- Open Multi-Processing
- A *standard* API used to explicitly direct multi-threaded, shared memory parallelism (C/C++, Fortran)
- Compiler directive-based (#pragma).
- Support for concurrency, synchronization, and data handling
- High level primitives (OpenMP) vs. low level primitives (Pthreads)

Programming Model

- OpenMP follows a fork-join model



OpenMP Code Structure

```
#include <omp.h>

main ()  {
int var1, var2, var3;
    …  Serial code …

// Beginning of parallel section. Fork a team of threads.
// Specify variable scoping
#pragma omp parallel private(var1, var2) shared(var3)
{
        … Parallel section executed by all threads …
        … All threads join master thread and disband …
}
    … Resume serial code …
}
```

OpenMP Directives

#pragma omp *directive-name [clause,…]*

{ *structured block of code* }

Examples:

#pragma omp parallel default(shared) private(beta,pi)

#pragma omp parallel for schedule(static)

Contents

- Introduction
- Parallelizing loops and sections
- Synchronization
- Runtime Library Routines

Parallel Region Construct

#pragma omp parallel [clause ...]
if (scalar_expression)
private (list)
shared (list)
default (shared | none)
firstprivate (list)
reduction (operator: list)
copyin (list)
num_threads (integer-expression)

{ structured_block }

How Many Threads?

1. Evaluation of the ***if*** clause (parallel/only master)
2. Setting of the ***num_threads*** clause
3. Use of the ***omp_set_num_threads()*** library function
4. Setting of the ***OMP_NUM_THREADS*** environment variable
5. Implementation default (number of CPUs)

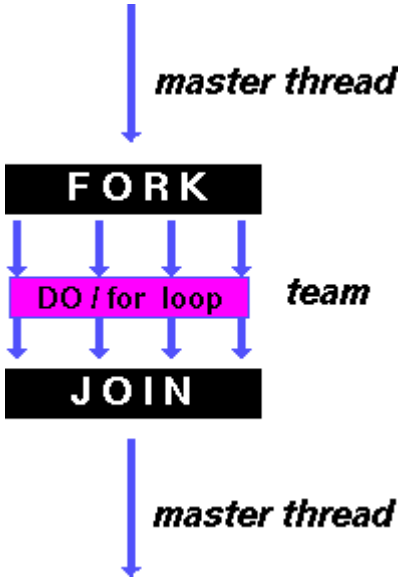
Data Scope Clauses

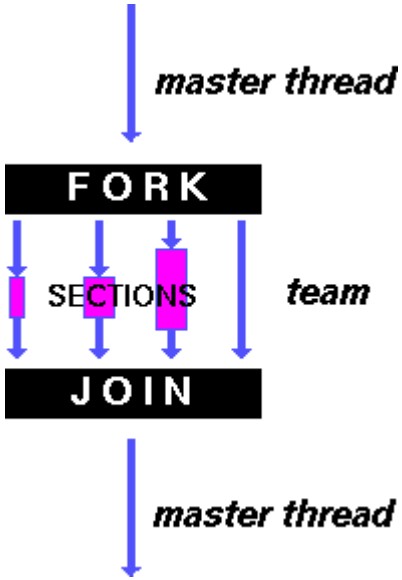
- *private (list):* private to each thread (uninitialized).
- *shared(list):* shared among all threads
- *default (shared | none):* default scope
- *firsprivate (list):* initialized according to the value of their original objects before entry the parallel region
- *lastprivate (list):* performs a copy from the last loop iteration or section to the original variable object.
- *reduction (operator:list):* thread private copy + final reduction operation (arithmetic, logic)

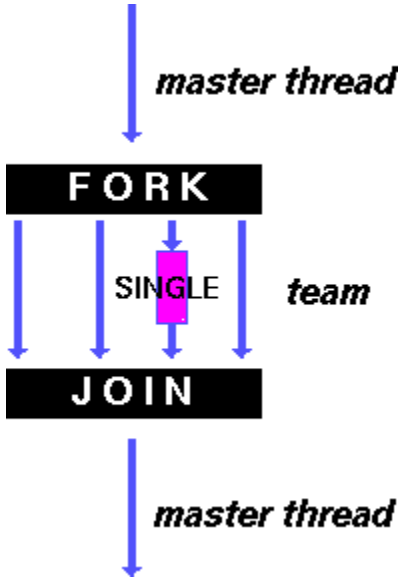
Parallel Region Example

```
int nthreads, tid;

#pragma omp parallel private(tid)
{
   /* Obtain and print thread id */
   tid = omp_get_thread_num();
   printf("Hello World from  thread = %d\n", tid);

   /* Only master thread does this */
   if (tid == 0){
     nthreads = omp_get_num_threads();
     printf("Number of threads = %d\n", nthreads);
   }
}
```

Work Sharing Constructs







#pragma omp for #pragma omp sections #pragma omp single

They must be defined **inside** a parallel region

for Directive

```
#pragma omp for [clause ...]
        schedule (type [,chunk])
        ordered
        private (list)
        firstprivate (list)
        lastprivate (list)
        shared (list)
        reduction (operator: list)
        collapse (n)
        nowait

    for_loop
```

for Directive Clauses

- *schedule:* how iterations of the loop are divided among the threads
  - *static*
  - *dynamic*
  - *guided*
  - *runtime:* scheduling decision is deferred until runtime by variable OMP_SCHEDULE
- *ordered:* iterations of the loop must be executed as they would be in a serial program
- *nowait:* threads do not synchronize at the end of the parallel loop

for Static Schedule

- Loop iterations are divided into pieces of size chunk and then statically assigned to threads.
- If chunk is not specified, the iterations are evenly (if possible) divided contiguously among the threads.

Examples (3 threads and 10 iterations):

```
#pragma omp for schedule(static)
#pragma omp for schedule(static,2)
#pragma omp for schedule(static,4)
```

for Dynamic Schedule

- Loop iterations are divided into pieces of size chunk, and dynamically scheduled among the threads; when a thread finishes one chunk, it is dynamically assigned another.
- The default chunk size is 1.

Examples (3 threads and 10 iterations):

```
#pragma omp for schedule(dynamic)
#pragma omp for schedule(dynamic,2)
#pragma omp for schedule(dynamic,4)
```

Guided Schedule

- A large chunk of contiguous iterations are allocated to each thread dynamically.
- The chunk size decreases exponentially with each successive allocation to a minimum size specified in the parameter chunk.
- The default minimum chunk size is 1.

sections Directive

```
#pragma omp sections [clause ...]
                    private (list)
                    firstprivate (list)
                    lastprivate (list)
                    reduction (operator: list)
                    nowait
{
    #pragma omp section
                    structured_block
    #pragma omp section
                    structured_block
}
```

single Directive

#pragma omp single [clause ...]
private (list)
firstprivate (list)
nowait

structured_block

Threads in the team that do not execute the single directive, wait at the end of the enclosed code block, unless a nowait clause is specified.

Combined Parallel Work Sharing Constructs

```
#pragma omp parallel
   #pragma omp for schedule(dynamic,2)
```

is equivalent to

```
#pragma omp parallel for schedule(dynamic,2)
```

The same for :

```
#pragma omp parallel sections
```

Contents

Synchronization Constructs

- *#pragma omp barrier*
  - synchronizes all threads
- *#pragma omp master*
  - code block executed only by the master thread
  - no implied barrier associated
- *#pragma omp critical [name]*
  - critical section for a block of code
  - different names for different critical sections
- *#pragma omp atomic*
  - critical section for only one statement

Example of Synchronization Constructs

```
#pragma omp parallel for
for (i=0;i<DIM;i++)
{
    #pragma omp atomic
    a++;

      #pragma omp critical
      {
          b++;
          if (b>LIMIT) c++;
      }
}
```

Contents

- Introduction
- Parallelizing loops and sections
- Synchronization
- Runtime Library Routines

Runtime Library Routines for Synchronization

- void omp_init_lock(omp_lock_t *lock)
- void omp_destroy_lock(omp_lock_t *lock)
- void omp_set_lock(omp_lock_t *lock)
- void omp_unset_lock(omp_lock_t *lock)

Example of Locking and Unlocking Routines

```
omp_lock_t lock;

omp_init_lock(&lock);

#pragma omp parallel for
for (i=0;i<DIM;i++)
{
        omp_set_lock(&lock);
        a++;
        omp_unset_lock(&lock);
}

omp_destroy_lock(&lock);
```

Other Runtime Library Routines

- void omp_set_num_threads(int num_threads)
- int omp_get_num_threads(void)
- int omp_get_thread_num(void)
- int omp_in_parallel(void) True/False
- double omp_get_wtime(void)

Example of omp_get_wtime

```
double init,end;

init=omp_get_wtime();
#pragma omp parallel
        … some parallel work ….

end=omp_get_wtime();

printf("Time: %f seconds\n",(end-init));
```

Contents

- Introduction
- Parallelizing loops and sections
- Synchronization
- Runtime Library Routines

Thank you for your attention!

This project is kindly funded by the
European Comission‘s Tempus IV program

Tempus