OpenMP Tutorial

What is OpenMP?

- OpenMP - abbreviation for **Open M**ulti-**P**rocessing
- API for multi-threaded, shared memory parallelism
- Designed for multi processor/core shared memory machines – UMA/NUMA accesses
- Three primary components are:

  Compiler directives
  Runtime library routines
  Environment variables

- Supports C/C++, Fortran. OpenMP like interface for Java - JOMP

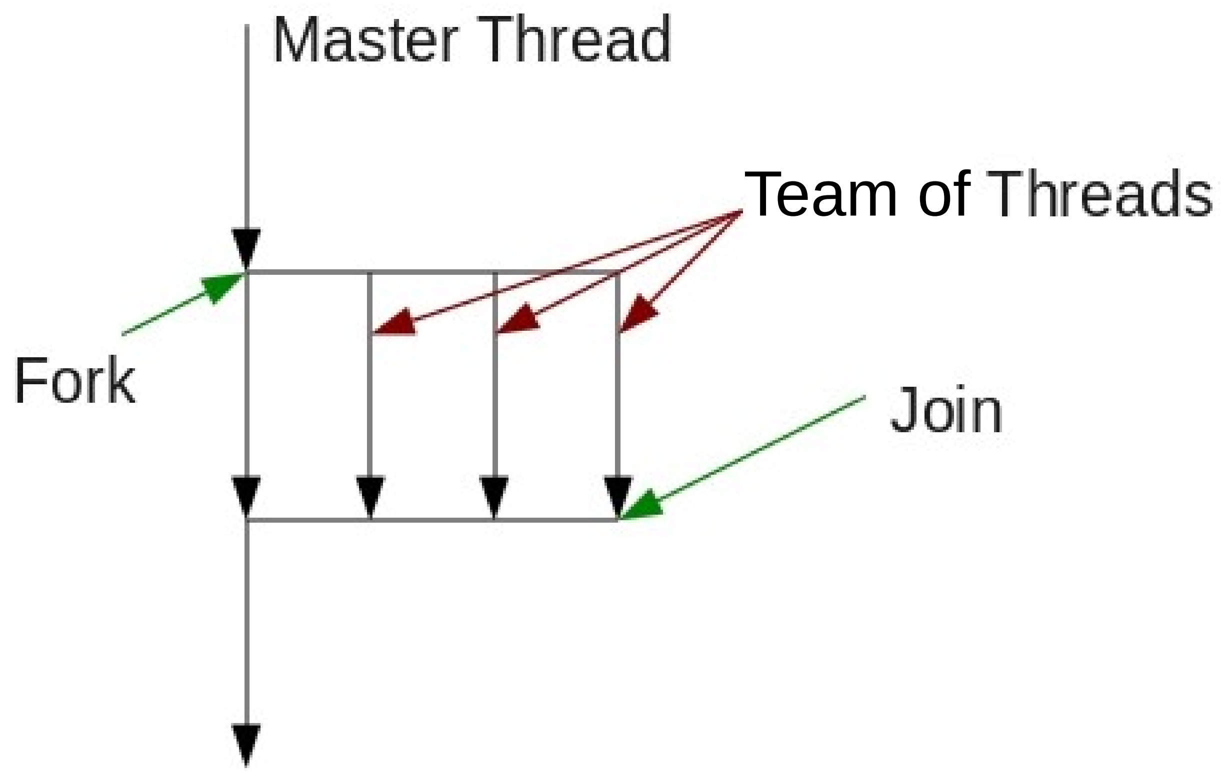
Note: Synchronizing Input/Output is programmer's responsibility

Programming Model

- Thread based parallelism
- Explicit parallelism – programmer has full control
- Fork - Join Model of execution

Each thread is assigned a unique id within team.

Master thread has id 0.



Hello World!

```c
#include <omp.h>
#include <stdio.h>
int main(void) {
omp_set_num_threads(10);
int tid = 0;
#pragma omp parallel private(tid)
{
    tid = omp_get_thread_num();
    if(tid%2==0)
      printf("Hello World from thread %d\n", tid);
}
printf("%d is the tid value here\n", tid);
return 0;
}
```

Output (Parallel, Sequential)

Hello World from thread 2
Hello World from thread 4
Hello World from thread 0
Hello World from thread 6
Hello World from thread 8
0 is the tid value here

For gcc/g++ compilers, use the flag fopenmp

gcc a.c -fopenmp

OpenMP API Overview

- Compiler directives

  #pragma omp directive-name [clause, ...] newline

  The parallel directive

  #pragma omp parallel [clause,..] newline structured_block

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

  - the thread forks a team of threads with itself as master
  - an implicit barrier at the end of parallel section causes the threads to join

- Clauses specify the conditions imposed during parallel section execution
  - Data scoping clauses – shared/private/lastprivate
  - Loop iteration divisions – schedule
  - No. of threads to create – num_threads

```
#pragma omp parallel for       \
  default(shared) private(i)   \
  schedule(static,chunk)       \
  num_threads(5)
```

- **if** clause – conditional parallelization. If the condition is false, the region is executed serially.

- No. of threads in a parallel region is determined by the following in precedence order:

   1. **if** clause evaluation
   2. Setting of the num_threads clause
   3. Using omp_set_num_threads() library function
   4. **OMP_NUM_THREADS** environment variable
   5. By default - usually the number of CPUs on a node.

- Runtime Library routines – defined in omp.h. Few examples are:

   omp_get_dynamic() - True, if dynamic adjustment of no. of threads created is enabled
   omp_set_dynamic() - enables alteration of no. of threads created
   omp_get_nested() - True if nested parallel regions enabled
   omp_set_nested() - enables nested parallel regions
   omp_get_num_threads() - gets number of threads

Work Sharing Constructs

Divides execution of enclosed region amongst the team members. No new threads are spawned.
for – parallel execution of iterations - of the loop immediately following it. The parallel region to be initialized earlier.

```
#pragma omp for [clause ...]  newline
            schedule (type [,chunk])
            ordered
            nowait
  for_loop
```

Things to note:
integer loop variable,
no *break* or *goto* in loop,
chunk per thread given by schedule, "dependencies" across iterations?

```c
#define N 1000
#define chunk_size 100
int main() {

    omp_set_dynamic(0);
    omp_set_num_threads(10);
    int i = 0, res = 0, tid;
    #pragma omp parallel private(tid)
    {
        #pragma omp for \
        schedule (static, chunk_size) \
        reduction(+: res)
        for (i = 0; i < N; i++) {
                tid = omp_get_thread_num();
            res = res + i;
            printf("%d %d\n", tid, res);

        }
    }
}
```

schedule(type [,chunk]) – describes how iterations of loop are divided among the threads.

static – iterations are divided in to parts of size 'chunk' and statically assigned to threads. Iterations are evenly divided contiguously if chunk not specified.

dynamic – same as above but dynamically allocated to threads

guided – similar to dynamic, but chunk size is defined as follows:

    Initial value proportional to # iterations / #threads
    Next values proportional to # remaining_iterations / #threads

auto/runtime – decision left to compiler/deferred till runtime. OMP_SCHEDULE

The chunk size should be a loop invariant integer expression.

nowait – threads do not synchronize at the end of parallel loop

ordered – iterations of loop executed as they would be in a serial program

reduction(operator: list) – how multiple local copies of a variable from threads are combined to a single variable at master when threads exit. The operator can be +, *, -, &, |, ^, && and ||

sections **–** a non-iterative work sharing construct. Independent **section** directives are nested within it. Each section is assigned to one thread.

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

 #pragma omp section   newline

    structured_block

 #pragma omp section   newline

    structured_block
}
```

```
#pragma omp parallel shared(a,b,c,d) private(i)
  {

   printf("Total no. of threads %d\n", omp_get_num_threads());
  #pragma omp sections nowait
    {

    #pragma omp section
    {
      printf("Executed by %d\n",omp_get_thread_num());
      for (i=0; i < N; i++)
        c[i] = a[i] + b[i];
    }
    #pragma omp section
    {
      printf("Executed by %d\n",omp_get_thread_num());
      for (i=0; i < N; i++)
        d[i] = a[i] * b[i];

    }
    }  /* end of sections */

  }  /* end of parallel section */
```

Op: Total no. of threads 4
Executed by 0
Total no. of threads 4
Executed by 1
Total no. of threads 4
Total no. of threads 4

Note:

OpenMP pragmas must be encountered by all threads in a team or none at all, hence a condition based execution is not allowed.

Also lookup:

- task, single, master, flush, threadprivate, copyin directives
- Combining/nesting the work sharing constructs/directives
- Environment variables: OMP_DYNAMIC, OMP_NESTED and OMP_NUM_THREADS
- omp_get_num_procs(), omp_in_parallel(), omp_get_max_threads()

Data Scope Attribute Clauses

These classes determine how the variables are scoped.

private - variables declared are private/local to the thread.

firstprivate - same as private, with automatic initialization of the variables in its list.

lastprivate - same as private, with the last loop iteration or section updating the value of the original variable object.

shared - variables in its list are shared among all threads in team

default - specify default scope for all variables in a parallel region

In a nested loop scenario for(i...) {.. for(j...) {.. for(k...) { } } } what happens with shared(j) ?

Synchronization

To impose order constraints on threads and control access to shared data, synchronization constructs are used.

- critical – specifies a region of code that should be executed by only one thread at a time. #pragma omp critical [(name)] newline structured_block The optional name enables multiple different critical regions to exist. Critical sections with same name are treated as the same region. All unnamed critical sections are treated as the same section.

```
#pragma omp parallel shared(max)
 {
     #pragma omp critical (FindMax)
  {
      if(max < n)    max = n;
   }
 }
```

Nesting of same name critical sections are not allowed to avoid potential deadlock scenarios.

- barrier – synchronizes all threads in the team. All/none threads in the team must execute barrier. A thread waits till all the threads reach the barrier. #pragma omp barrier newline

- atomic - allows a specific memory location to be updated atomically (applies to the statement immediately following it). Preferred for single memory updates over critical which is used for set of statements. #pragma omp atomic newline <statement>

- ordered – iterations of enclosed loop to be executed in original serial order. Only one thread allowed in the section at a time.

Can I use atomic for x = x+2;
Implication of ordered clause?

```
#pragma omp parallel
{
  #pragma omp for ordered
  {
    for(i = 0; i < N; i ++) {
      #pragma omp ordered
      A[i] =  A[i-1]*2;
     }
   }
 }
```

Lock Routines – for low level synchronization / access control of all thread visible variables.

omp_init_lock() - initializes a lock associated with a lock variable
omp_set_lock() - to acquire a lock ; wait if already in use
omp_unset_lock() - to release a lock
omp_test_lock() - to set a lock ; doesn't block if unavailable
omp_destroy_lock() - frees the lock variable from any locks

Read – Nested locks

```
omp_lock_t lock_var;
omp_init_lock(lock_var);
#pragma omp parallel private (result,tid)
{
    tid = omp_get_thread_num();
    result = do_calc(tid);
    omp_set_lock(&lock_var);
    printf("%d %d",result,tid);
    omp_set_lock(&lock_var);
}
omp_destroy_lock(&lock_var);
```

References

- OpenMP - https://computing.llnl.gov/tutorials/openMP/
- A “Hands-on” introduction to OpenMP - http://openmp.org/mp-documents/omp-hands-on-SC08.pdf
- OpenMP FAQs - http://openmp.org/openmp-faq.html
- Guide into OpenMP (C++): http://bisqwit.iki.fi/story/howto/openmp/
- OpenMP: a Standard for Directive Based Parallel Programming - Section 7.10 – Introduction to Parallel Computing by Ananth Grama, Anshul Gupta, George Karypis and Vipin Kumar