Introduction to High-Performance Computing

Session 06

Introduction to OpenMP (II)

OpenMP

- OpenMP is a parallel programming model
  - based on shared memory with workload distribution among threads
  - uses mainly compiler directives and a few additional library routines

so far we have seen:

- how to compile OpenMP programs
- how to run OpenMP programs in a job script
- how to create parallel regions

next topic is how to distribute the workload among the threads

Work Sharing Directives

- parallel region to create a team of threads
  - every thread executes the same code
  - example

```
const int N=1000000;
double x[N];
#pragma omp parallel
{
  int threadID = omp_get_thread_num();

  for(int i=0; i<N; i++)
    x[i] = 1./double(threadID+1);
}
```

  - every thread does the same work (and there is a race condition)

Work Sharing Directives

- parallel region to create a team of threads
  - every thread executes the same code
  - example

```
const int N=1000000;
double x[N];
#pragma omp parallel
{
  int threadID = omp_get_thread_num();
  #pragma omp for
  for(int i=0; i<N; i++)
    x[i] = 1./double(threadID+1);
}
```

  - now every thread does a chunk of the work (and there is no race condition)

Work Sharing Directives

- parallel region to create a team of threads
  - every thread executes the same code
  - example

```
const int N=1000000;
double x[N];
#pragma omp parallel for
{
  for(int i=0; i<N; i++)
    x[i] = 1./(i+1.);
}
```

  - directive can be separated or combined as needed

Work Sharing Directives

- usable in parallel regions
- directives to specify how the work is distributed
- no synchronization at entry, only at exit (disable with nowait)
- directives

– for	split a loop into parallel tasks
– sections/section	defines a task for one thread
– single/master	one/master thread only, no synchronization
– critical	executed by one thread at a time
– …	

- additional clauses e.g. to further specify distribution of work

Example: Mean of Random Numbers

- how to parallelize the program Random.cpp with OpenMP?
  - e.g. the calculation of the mean value

```
// calculate mean value
double mean=0;
for (int i=0; i<NSIZE; i++)
   mean += vec[i];
mean /= NSIZE;
```

Example: Mean of Random Numbers

- how to parallelize the program Random.cpp with OpenMP?
  - e.g. the calculation of the mean value

```
// calculate mean value
double mean=0;
#pragma omp parallel shared(mean)
{
   double mean_loc=0;
   #pragma omp for
   for (int i=0; i<NSIZE; i++)
      mean_loc += vec[i];
   #pragma omp critical
   mean += mean_loc;
}
mean /= NSIZE;
```

OpenMP Directive `critical`

- only one thread at a time can execute critical code block
  - in the example

    ```
    #pragma omp critical
    mean += mean_loc;
    ```

    this ensures mean is calculated without race condition
  - overhead for synchronization and serialization of code block
  - a faster alternative is provided by the atomic directive

    ```
    #pragma omp atomic
    mean += mean_loc;
    ```

  - has limitation on the expressions (critical is more general)

OpenMP `reduction` Clause

- an alternative (optimal?) solution can be obtained with the reduction clause

```
// calculate mean value
double mean=0;
#pragma omp parallel reduction(+:mean)
{
   #pragma omp for
   for (int i=0; i<NSIZE; i++)
      mean += vec[i];
}
mean /= NSIZE;
```

  – no need of critical section and private variable mean_loc

OpenMP Clauses

- the behavior of OpenMP directives can be adjusted using clauses
  - e.g. the following clauses can be used with the for directive:

```
private(list)
firstprivate(list)
lastprivate(list)
```
} how data is treated

```
reduction(reduction-identifier:list)
```
} compiler creates reduction operation

```
schedule([modifier [,modifier]:]kind[, chunk_size])
collapse(n)
ordered[(n)]
```
} how work of loop is distributed among threads

```
nowait
```
} no implicit barrier at the end of loop construct

Code Portability

- it is often desirable to have the same code file being used for serial and OpenMP parallel code
  - use conditional compilation, e.g.

```
#ifdef _OPENMP
  double wt1 = omp_get_wtime();
#endif
```

  - pragmas only have effect when OpenMP option is used at compile time
  - code becomes more difficult to read

OpenMP Summary

- standard for easy shared memory parallelization
- uses compiler directives and some library functions
- based on threads and a fork-join model
- incremental parallelization
- serial and parallel code in one source file
- difference between shared and private data is important
- be careful about race conditions

Exercises

Calculate Pi in Parallel

- modify the program Pi.cpp so that it parallelizes the computation of Pi with OpenMP
  - add a parallel region to the code
  - parallelize the loop so that each thread computes a part of sum (integral)
  - combine the partial sums for the final answer
  - also add a wall clock timer (omp_get_wtime()) and compare the change in CPU and wall clock time for different number of threads