High Performance Computing (Nov 12)

Question 1

Run the OpenMP program on Slide 16 with changing the number of threads. How does the execution time change with the thread count?

Answer

The code calls the sleep() function 16 times. At each function call, the process sleeps for 1 second. As a result, the process is expected to take 16 seconds in the case of single-thread execution. If the process is executed with N threads, each thread simultaneously calls the function and thus sleeps for 1 second at the function call. In theory, hence, the execution time will be 16/N seconds.

Sample job script (run.sh)

```
---------------------------------------------------------------------------
#!/bin/sh
#PBS -q lx_edu

cd $PBS_O_WORKDIR

export OMP_NUM_THREADS=1
time ./a.out
export OMP_NUM_THREADS=2
time ./a.out
export OMP_NUM_THREADS=4
time ./a.out
export OMP_NUM_THREADS=8
time ./a.out
export OMP_NUM_THREADS=16
time ./a.out
---------------------------------------------------------------------------
```

Execution results (run.sh.eXXXXXX)

```
---------------------------------------------------------------------------
real    0m16.004s
user    0m0.001s
```

```
sys     0m0.000s

real    0m8.003s
user    0m0.002s
sys     0m0.000s

real    0m4.003s
user    0m0.002s
sys     0m0.001s

real    0m2.003s
user    0m0.000s
sys     0m0.007s

real    0m1.004s
user    0m0.020s
sys     0m0.003s
```

The time command shows the total execution time (wall clock time) as “real,” the CPU time (spent by the program itself) as “user,” and the execution time spent by the OS as “sys.” The sample program calls the sleep() function 16 times to increase the total execution time. But the function call just sleeps the process, and thus does not spend the CPU time or the OS time for the execution.

Each thread sleeps at the same time. As a result, the execution time decreases with the thread count. In this example, the runtime overhead is negligibly small, and the execution time decreases almost perfectly with the thread count.

Question 2

Write a code to check the behavior of reduction code in Slide 22, and paste it.

Answer

A simple example to check the behavior is as follows.

```
#include <stdio.h>
#include <unistd.h>
```

```
#define N 256

 int main(int ac,char* av)
 {
   int i,j,x;

   x=0;
 #pragma omp parallel for reduction(+:x),private(j)
   for(i=0;i<N;i++){
     for(j=0;j<N;j++){
       x+=j;
   }}
   printf("%d¥n",x);
   return 0;
 }
```

You can check the behavior by enabling and disabling OpenMP directives. It is enabled only when the command line option of -fopenmp is given at compilation.

```
$ gcc -fopenmp omp2.c
$ ./a.out
8355840
$ gcc omp2.c
$ ./a.out
8355840
```

As you can see, the computation result does not change, meaning that the reduction clause works correctly.