Parallel Programming Languages 1 - OpenMP

H. Kim
(hjunkim@skku.edu)

College of Information and Communication Engineering
Sungkyunkwan University
Shared Memory Parallel Programming in the Multi-Core Era

- Desktop and laptop
  - 2, 4, 8 cores and more and more

- Shared memory hardware accelerators
  - NVIDIA’s GTX980Ti GPU has 2816 cores
  - Upcoming Intel Xeon Phi has up to 72 cores
OpenMP?

• API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran

• OpenMP consists of a set of
  • Compiler directives – ex) #pragma omp parallel
  • Runtime library – ex) omp_set_num_threads(N), omp_get_thread_num()
  • Environment variables – ex) OMP_NUM_THREADS
OpenMP Hello World

• Hello world program – “hello.c”
• $gcc –fopenmp –o hello hello.c

Sequential Program

```c
#include <stdio.h>
int main(void) {
    printf("Hello World\n");
    return 0;
}
```

Parallel Program

```c
#include <stdio.h>
#include <omp.h>
int main(void) {
    #pragma omp parallel
    {
        printf("Hello World\n");
    }
    return 0;
}
```
OpenMP is usually used to parallelize loops

- Find the most time consuming loops
- Split them up between threads

```
#include <stdio.h>
int main(void) {
    int i, N=1000;
    double A[N], B[N], C[N];
    for (i=0; i<N; i++) {
        A[i] = B[i] + C[i];
    }
    return 0;
}
```

```
#include <stdio.h>
#include <omp.h>
int main(void) {
    int i, N=1000;
    double A[N], B[N], C[N];
    #pragma omp parallel for
    for (i=0; i<N; i++) {
        A[i] = B[i] + C[i];
    }
    return 0;
}
```
• SPMD: Single Program Multiple Data

Thread 0

```c
#include <stdio.h>
#include <omp.h>
int main(void) {
    int i, N=1000;
    double A[N], B[N], C[N];
    #pragma omp parallel for
    for (i=0; i<250; i++) {
        A[i] = B[i] + C[i];
    }
    return 0;
}
```

Thread 1

```c
#include <stdio.h>
#include <omp.h>
int main(void) {
    int i, N=1000;
    double A[N], B[N], C[N];
    #pragma omp parallel for
    for (i=250; i<500; i++) {
        A[i] = B[i] + C[i];
    }
    return 0;
}
```

Thread 3

```c
#include <stdio.h>
#include <omp.h>
int main(void) {
    int i, N=1000;
    double A[N], B[N], C[N];
    #pragma omp parallel for
    for (i=750; i<1000; i++) {
        A[i] = B[i] + C[i];
    }
    return 0;
}
```
OpenMP Fork-and-Join Model

N=1000;
#pragma omp parallel for
for (i=0; i<N; i++) {
  A[i] = B[i] + C[i];
}

M=500;
#pragma omp parallel for
for (i=0; i<N; i++) {
  A[i] = B[i] + C[i];
}
**Reduction**

- Reduction(operator | list)

- Reduction generates private copy of the list for every threads, then stores final result into a shared variable

```c
int main(void) {
    sum = 0;
    #pragma omp parallel for reduction (+:sum)
    for (i=0; i<N; i++) {
        sum = sum + A[i];
    }
}
```
Calculate “Pi” Using Monte Carlo Simulation

- Square (width, height = 1 = radius ‘r’)
- A minimum distance from an origin to circular arc is 1
- If a distance from origin to random (x,y) is < 1, then (x,y) is placed within a circular arc
- else, (x,y) is placed outside of a circular arc

\[ \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4} = \frac{\text{# of coordinates within a circle}}{\text{total # of coordinates}} \]
Calculate “Pi” Using Monte Carlo Simulation (cont.)

- Private(var1, var2, …, varN)

**Sequential Program**

```c
rnd = 1.0 / (double)RAND_MAX;
for (i=0; i<LOOP_ITERATION; i++) {
    x = (double)rand_r(&state) * rnd;
    y = (double)rand_r(&state) * rnd;
    if (x*x + y*y < 1) {
        hits++;
    }
}
```

**Parallel Program**

```c
rnd = 1.0 / (double)RAND_MAX;
#pragma omp parallel \ 
private(x,y,state) reduction(+:hits)
{
    #pragma omp for
    for (i=0; i<LOOP_ITERATION; i++) {
        x = (double)rand_r(&state) * rnd;
        y = (double)rand_r(&state) * rnd;
        if (x*x + y*y < 1) {
            hits++;
        }
    }
}
```
Private

• A variable that can be accessed by all the threads in the thread group has **shared**

• A variable that can only be accessed by a single thread has **private**

• The default scope for variables declared before a parallel block is **shared**
Summary

• OpenMP is great for parallel programming
  • Sequential programs can be enhanced with OpenMP directives, leaving the original program essentially intact
  • We learnt basic OpenMP with “hello world, for loop, reduction, Monte Carlo” examples

• For further detail, go to http://www.openmp.org
• John Conway’s game of life
  • Cellular automation

• Universe
  • Infinite 2D orthogonal grid of square cells
  • Cell has 2 states: live or dead
  • At every step, state is decided depending on 8 neighbor cells

• Rules
  • Any live cell with live_neighbors # < 2 or live_neighbors # > 3 will die
  • Any live cell with 2 ≤ live_neighbors #≤ 3, will live
  • Any dead cell with live_neighbors # = 3 will come to life
3D Game of Life

• Cubic cells in 3-D universe
  • One cell will have 26 \((3^3 - 1)\) neighbors

• Rules
  • \(N = \text{live_neighbors} \#\)
  • If a live cell, with \(N < D1\) or \(N > D2\), will die
  • If a dead cell, with \(L1 < N < L2\), will come to life
  • Otherwise, unchanged
Programming

• 3-D game of life in two programming models
  • OpenMP
  • MPI

• Input file
  • Size of universe, D1, D2, L1, L2, steps to run
  • Initial seeds of cells

• Output
  • Measured time
    • Report on monitor in microseconds (using gettimeofday() function)
    • Core work of simulation excluding file read/write, initialization
  • Output file
    • Resulting universe of cells (the same format of initial seeds of cells)
Input Example

- Input
  - 3 x 3 x 3 universe input
  - D1=5, D2=20, L1=10, L2=15
  - 1000 steps

```bash
$ cat input.life
3 5 20 10 15 1000
010
010
101
110
100
000
001
001
101
```

- Diagram showing cell(x,y,0), cell(x,y,1), cell(x,y,2)
Submission

• Submit through “icampus”
• Due by 4/11(Mon), midnight (23:59:59 pm)
• Submission.tar
  • 2 directories for source codes (OpenMP, MPI)
  • Each directory contains
    • Only source code of your work
      • No binary/object code, input file, output file
    • Makefile
    • README: a short description of your optimization work and how to make and run your program