INSTITUTE OF COMMUNICATION, INFORMATION AND PERCEPTION TECHNOLOGIES

Scuola Superiore Sant'Anna

OpenMP and GPU Programming

OpenMP Exercises

Emanuele Ruffaldi

PERCeptual RObotics Laboratory, TeCIP
Scuola Superiore Sant'Anna
Pisa,Italy
e.ruffaldi@sssup.it

April 6, 2016

Task Dependencies

OpenMP allows for creating dependencies between tasks for moving from a purely parallel model to a data flow model. The result is that the task structure moves from hierarchical fork-join to a Direct Acyclic Graph (DAG).
The depend clause specifies how a variable, or array slide, is produced or consumed by the task. The dependent-type can be in,out or inout followed by a list of variables. The newly generated task that specifies a in dependency will wait sibling tasks with out or inout.

```
for (int i = 0; i < T; ++i) {
    #pragma omp task shared(x, ...) depend(out: x) // T1
    preprocess_some_data(...);
    #pragma omp task shared(x, ...) depend(in: x) // T2
    do_something_with_data(...);
    #pragma omp task shared(x, ...) depend(in: x) // T3
    do_something_independent_with_data(...);
}
```

Manual Code Protection

In some cases the resource protection provided by the shared/private mechanism of tasks is not enough.
For this reason OpenMP provides two constructs that supports such a protection. The first is the one of **critical** section that protects the execution of code: only one task at time executes a given code region.
Then we have atomic directives for controlling the access and the operations to variables.

Exercises

- Fibonacci (basic tasking)
- Mandelbrot Set (looping)

Tools: GCC >4.9
Source code will be made available on Github
`https://github.com/eruffaldi/course_openmpgpu`.

Reference

- Environment Variables
  - OMP_NUM_THREADS
  - OMP_DYNAMIC
- Functions
  - omp_set_num_threads
  - omp_set_dynamic
- Directives
  - parallel
  - parallel for (with schedule, unordered, collapse)
  - single
  - critical
  - task/taskwait/taskgroup/taskyield
  - cancel taskgroup

Parallel Fibonacci

Compute the Fibonacci sequence using a recursive parallel scheme based on tasks. Then estimate timing. Use the following stub:

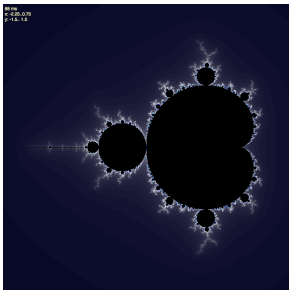
```
#include <omp.h>
#include <stdio.h>

// omp_get_wtime()

int fib(int n)
{
    // usi openmp
}

int pfib(int q)
{
    // ... something ...
}

int main(int argc, char const *argv[])
{
    int n = pfib(atoi(argv[1]));
    printf("hello %d\n",n);
    return 0;
}
```

Mandelbrot Set

A Mandelbrot Set is the most known fractal that is built based on the termination situation of a recursive function. In particular the recursive function:

$$z[k+1] = z[k]^2 + c \quad (1)$$
$$z[1] = (0,0) \quad (2)$$

The iteration is stop when the complex modulus is larger than a threshold (e.g. 100) or a number of iterations have been reached (e.g. 256)
Then the point is colored based on the number of iterations with the convention of 0 if the maximum iteration is reached.
Javascript example online: here.
For saving use stb_image_writer.h library.



Visualizing For Schedule

The schedule clause of the parallel for allows for specifying the distribution across the threads. While performing the Mandlebrot set it is possible to display the different scheduling policies. The images on the right show the following cases with 8 threads and collapse 2.

- Top-left static 50
- Top-right dynamic 50
- Bottom-left guided 50
- Bottom-right auto