Parallel Processing

Lecture 9
Guy Tel-Zur

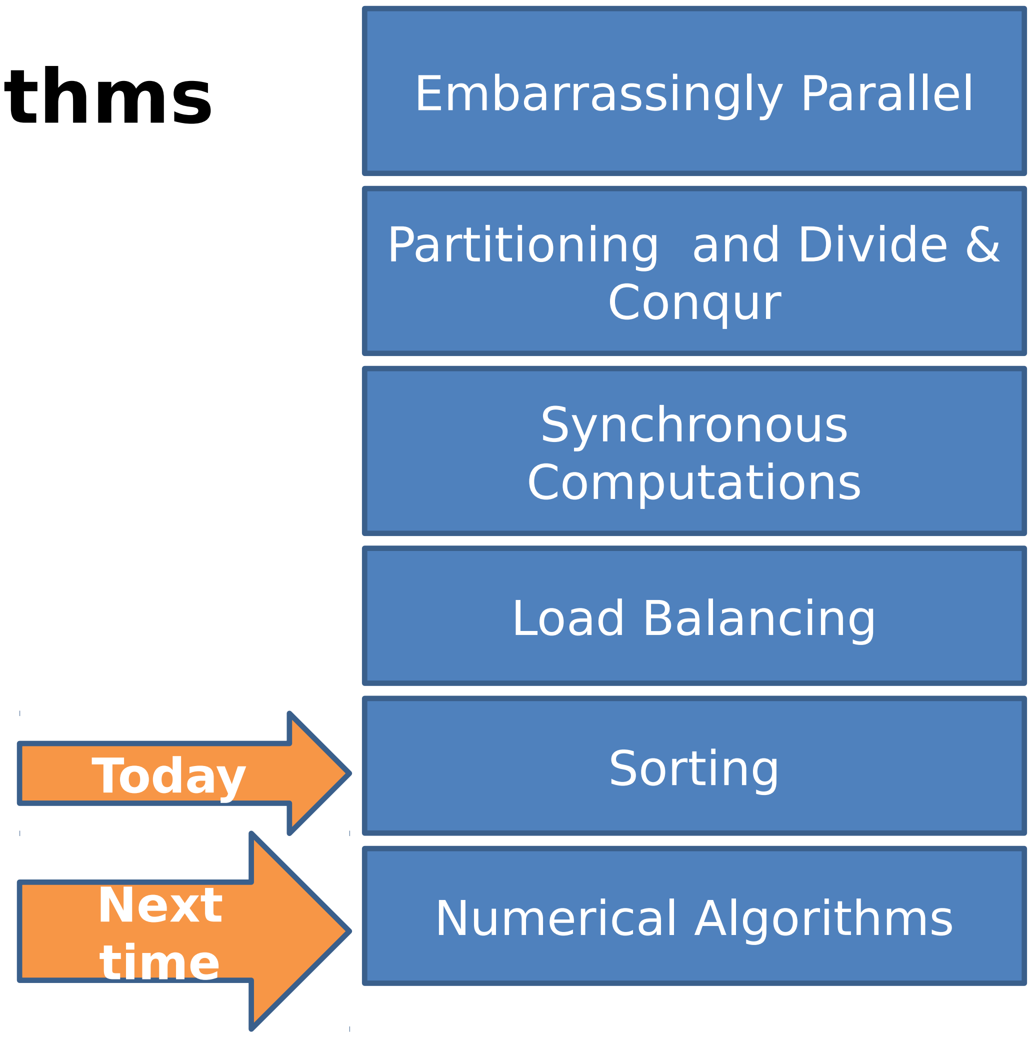
Version: 20/12/2015

Today’s agenda

- Final Presentations status
- Continue with OpenMP
- CilkPlus
- Parallel Matlab
- Sorting Algorithms (slides10)
- Load Balancing (slides7)
- Home assignment #3

The Course Roadmap

Algorithms

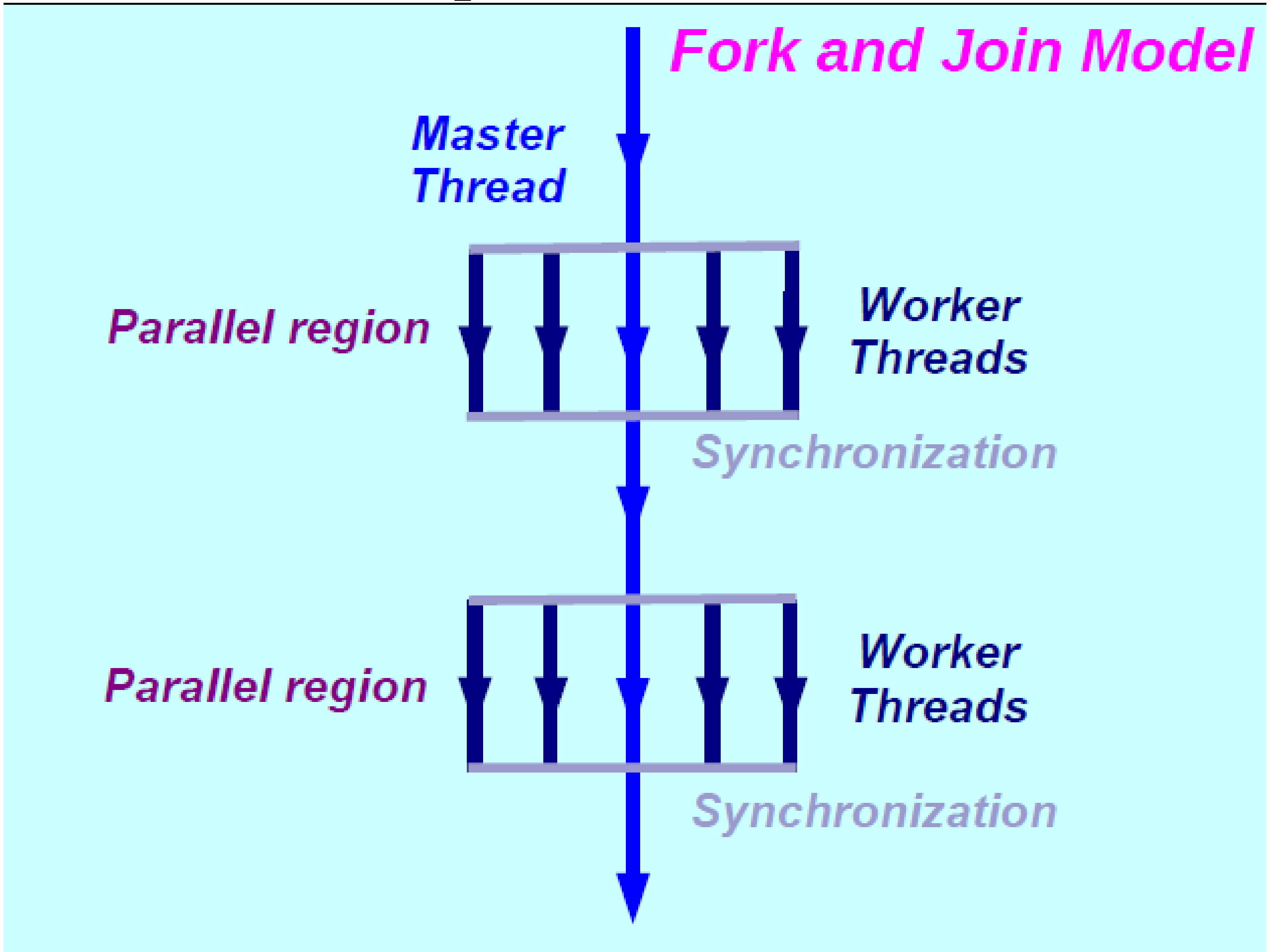


MPI tip – shortcuts using define

```
#define MASTER 0
#define Bcast(send_data, count, type)
MPI_Bcast(send_data, count, type, MASTER,
MPI_COMM_WORLD) //root --> MASTER
#define Finalize() MPI_Finalize()
#define Init(x,y) MPI_Init(x,y)
#define Rank(x) MPI_Comm_rank(MPI_COMM_WORLD, x)
#define Size(x) MPI_Comm_size(MPI_COMM_WORLD, x)
```

Caution: Name Space !

The OpenMP Model



Exercise 2: A simple SPMD pi program

```
#include <omp.h>
static long num_steps = 100000;        double step;
#define NUM_THREADS 2
void main ()
{             int i, nthreads;  double pi, sum[NUM_THREADS];
              step = 1.0/(double) num_steps;
              omp_set_num_threads(NUM_THREADS);
   #pragma omp parallel
   {
             int i, id,nthrds;
            double x;
            id = omp_get_thread_num();
            nthrds = omp_get_num_threads();
            if (id == 0)   nthreads = nthrds;
              for (i=id, sum[id]=0.0;i< num_steps; i=i+nthrds) {
                          x = (i+0.5)*step;
                          sum[id] += 4.0/(1.0+x*x);
              }
    }
              for(i=0, pi=0.0;i<nthreads;i++)pi += sum[i] * step;
}
```

Promote scalar to an array dimensioned by number of threads to avoid race condition.

Only one thread should copy the number of threads to the global value to make sure multiple threads writing to the same address don't conflict.

This is a common trick in SPMD programs to create a cyclic distribution of loop iterations

Exercise 3: SPMD Pi without false sharing

```
#include <omp.h>
static long num_steps = 100000;        double step;
#define NUM_THREADS 2
void main ()
{            double  pi;          step = 1.0/(double) num_steps;
             omp_set_num_threads(NUM_THREADS);
#pragma omp parallel
{
             int i, id,nthrds;    double x, sum;
           id = omp_get_thread_num();
           nthrds = omp_get_num_threads();
           if (id == 0)   nthreads = nthrds;
             id = omp_get_thread_num();
           nthrds = omp_get_num_threads();
             for (i=id, sum=0.0;i< num_steps; i=i+nthreads){
                         x = (i+0.5)*step;
                         sum += 4.0/(1.0+x*x);
             }
          #pragma omp critical
                    pi += sum * step;
}
}
```

Create a scalar local to each thread to accumulate partial sums.

No array, so no false sharing.

Sum goes "out of scope" beyond the parallel region … so you must sum it in here. Must protect summation into pi in a critical region so updates don't conflict

Exercise 4: solution

```
#include <omp.h>
static long num_steps = 100000;        double step;
#define NUM_THREADS 2
void main ()
{           int i;    double x, pi, sum = 0.0;
            step = 1.0/(double) num_steps;
            omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for private(x) reduction(+:sum)
            for (i=0;i< num_steps; i++){
                     x = (i+0.5)*step;
                     sum = sum + 4.0/(1.0+x*x);
            }
            pi = step * sum;
}
```

For good OpenMP implementations, reduction is more scalable than critical.

i private by default

Note: we created a parallel program without changing any code and by adding 4 simple lines!

Parallel Programmers love Monte Carlo algorithms

Embarrassingly parallel: the parallelism is so easy its embarrassing.

Add two lines and you have a parallel program.

```
#include “omp.h”
static long num_trials = 10000;
int main ()
{
  long i;     long Ncirc = 0;      double pi, x, y;
  double r = 1.0;   // radius of circle. Side of squrare is 2*r
  seed(0,-r, r);  // The circle and square are centered at the origin
  #pragma omp parallel for private (x, y) reduction (+:Ncirc)
  for(i=0;i<num_trials; i++)
  {
    x = random();        y = random();
    if ( x*x + y*y) <= r*r)   Ncirc++;
   }

   pi = 4.0 * ((double)Ncirc/(double)num_trials);
   printf("\n %d trials, pi is %f \n",num_trials, pi);
}
```

Compiler notes: Visual Studio

- Start “new project”
- Select win 32 console project
  - Set name and path
  - On the next panel, Click “next” instead of finish so you can select an empty project on the following panel.
  - Drag and drop your source file into the source folder on the visual studio solution explorer
  - Activate OpenMP
    - Go to project properties/configuration properties/C.C++/language … and activate OpenMP
- Set number of threads inside the program
- Build the project
- Run “without debug” from the debug menu.