OpenMP Programming

Aiichiro Nakano

Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Chemical Engineering & Materials Science
Department of Biological Sciences
*University of Southern California*

Email: anakano@usc.edu

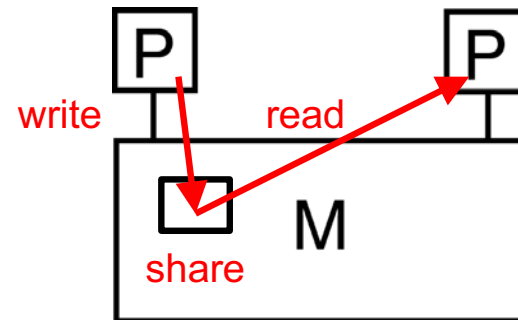**Goal:** Use multiple cores in a computing node *via* multithreading

OpenMP

- **Portable application program interface (API) for shared-memory parallel programming based on multi-threading by compiler directives**

- **OpenMP = <u>Open</u> specifications for <u>Multi Processing</u>**

- **OpenMP homepage**
  `www.openmp.org`

- **OpenMP tutorial**
  `computing.llnl.gov/tutorials/openMP`

- **Process: an instance of program running**

- **Thread: a sequence of instructions being executed, possibly sharing resources with other threads within a process**
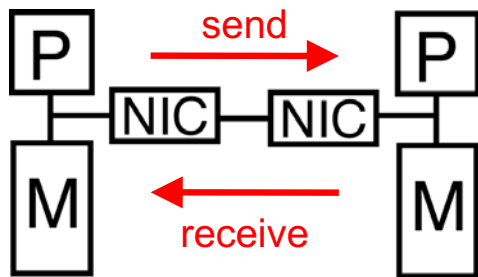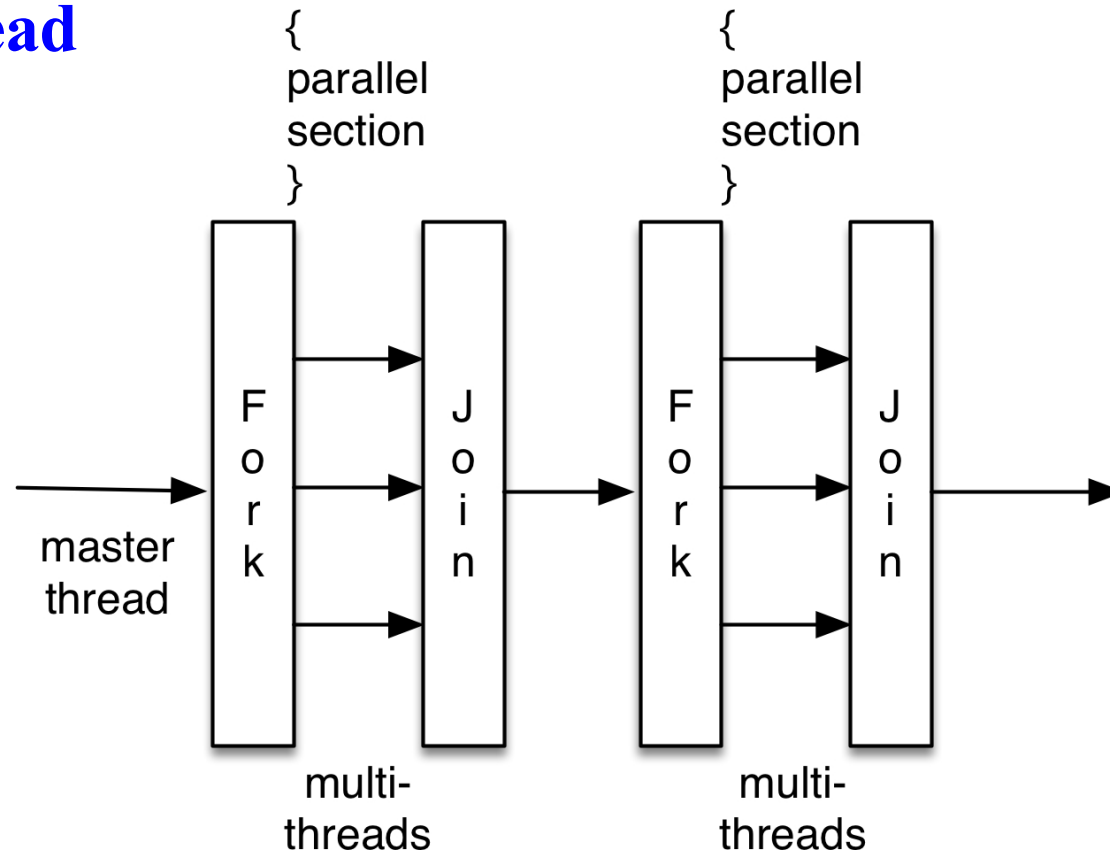

MPI (distributed memory)   **OpenMP (shared memory)**

OpenMP Programming Model

Fork-join parallelism

- **Fork:** master thread spawns a team of threads as needed

- **Join:** when the team of threads complete the statements in the parallel section, they terminate synchronously, leaving only the master thread

- **OpenMP threads communicate by sharing variables**

OpenMP Example: `omp_example.c`

```c
#include <stdio.h>
#include <omp.h>
void main () {
  int nthreads,tid;
  nthreads = omp_get_num_threads();          ← Get the number of threads
  printf("Sequential section: # of threads = %d\n",nthreads);
  /* Fork multi-threads with own copies of variable */
  #pragma omp parallel private(tid)
  {
                                             ← Each threads gets a private variable
    /* Obtain & print thread id */
    tid = omp_get_thread_num();              ← Get my thread ID: 0, 1, ...
    printf("Parallel section: Hello world from thread %d\n",tid);
    /* Only master thread does this */
    if (tid == 0) {
      nthreads = omp_get_num_threads();
      printf("Parallel section: # of threads = %d\n",nthreads);}
  } /* All created threads terminate */
}
```

parallel section

- **Obtain the number of threads & my thread ID** (*cf*. **MPI_Comm_size & MPI_Comm_rank**)

- **By default, all variables are shared unless selectively changing storage attributes using private clauses**

OpenMP Example: `omp_example.c`

- **Compilation on `carc.usc.edu`**
  `gcc` -o omp_example omp_example.c **–fopenmp**

- **Slurm script**
  ```
  #!/bin/bash
  #SBATCH --nodes=1
  #SBATCH --ntasks-per-node=1    1 process per computing node
  #SBATCH --cpus-per-task=2      2 cores (threads) per process
  #SBATCH --time=00:00:59
  #SBATCH --output=omp_example.out
  #SBATCH -A anakano_429
  export OMP_NUM_THREADS=2  ←——————  Set the # of threads using environment parameter
  ./omp_example
  ```

- **Output**
  ```
  Sequential section: # of threads = 1
  Parallel section: Hello world from thread 1
  Parallel section: Hello world from thread 0
  Parallel section: # of threads = 2
  ```

Setting the Number of Threads

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

void main () {
  int nthreads,tid;
  omp_set_num_threads(2);
  nthreads = omp_get_num_threads();
  printf("Sequential section: # of threads = %d\n",nthreads);
  /* Fork multi-threads with own copies of variable */
  #pragma omp parallel private(tid)
  {
    /* Obtain & print thread id */
    tid = omp_get_thread_num();
    printf("Parallel section: Hello world from thread %d\n",tid);
    /* Only master thread does this */
    if (tid == 0) {
      nthreads = omp_get_num_threads();
      printf("Parallel section: # of threads = %d\n",nthreads);
    }
  } /* All created threads terminate */
}
```

- **Setting the number of threads to be used in parallel sections within the program (no need to set OMP_NUM_THREADS); see `omp_example_set.c`**

OpenMP Programming Model

- **OpenMP is typically used to parallelize (big) loops**

- **Use synchronization mechanisms to avoid race conditions (*i.e.*, the result changes for different thread schedules)**

- **Critical section: only one thread at a time can enter**

```
#pragma omp parallel
{
  ...
   #pragma omp critical
   {
     ...
   }
   ...
}
```

Threads wait their turn— only one at a time executes the critical section
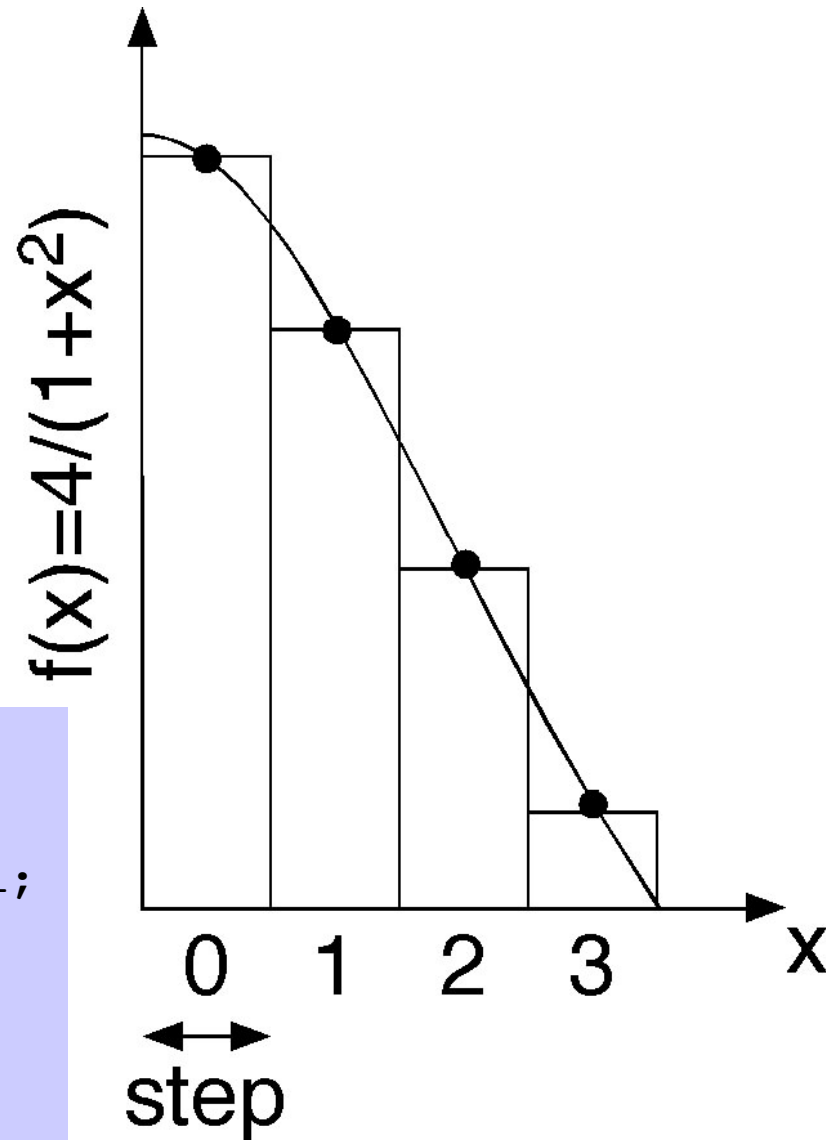
Example: Calculating π

- **Numerical integration**

$$\int_0^1 \frac{4}{1+x^2}\, dx = \pi$$

- **Discretization:**

$\Delta = 1/N$: `step = 1/NBIN`

$x_i = (i+0.5)\Delta \ (i = 0,\ldots,N\text{-}1)$

$$\sum_{i=0}^{N-1} \frac{4}{1+x_i^2}\Delta \cong \pi$$


```c
#include <stdio.h>
#define NBIN 100000
void main() {
  long long i; double step,x,sum=0.0,pi;
  step = 1.0/NBIN;
  for (i=0; i<NBIN; i++) {
    x = (i+0.5)*step;
    sum += 4.0/(1.0+x*x);}
  pi = sum*step;
  printf( "PI = %f\n" ,pi);
}
```

# OpenMP Program: `omp_pi_critical.c`

```c
#include <stdio.h>
#include <omp.h>
#define NBIN 100000
void main() {
  double step,sum=0.0,pi;
  step = 1.0/NBIN;
  # pragma omp parallel
  {
    int nthreads,tid; long long i;
    double x;
    nthreads = omp_get_num_threads();
    tid = omp_get_thread_num();
    for (i=tid; i<NBIN; i+=nthreads) {
      x = (i+0.5)*step;
      #pragma omp critical
      sum += 4.0/(1.0+x*x);
    }
  }
  pi = sum*step;
  printf("PI = %f\n",pi);
}
```

Shared variables

Private (local) variables

This has to be atomic

Thread-private variables: Either declare private or define within a parallel section

tid = 0
tid = 1

0 1 0 1 0
0 1 2 3 4 = NBIN-1
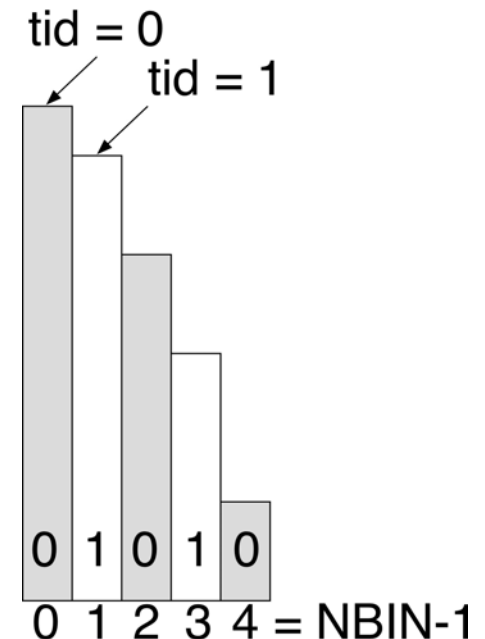
Race Condition

- **Race condition: Output is dependent on the sequence or timing of how multiple threads are executed**
- **Race condition arises if the read & write operations below are not atomic (a set of operations is atomic if they are executed without being interrupted by other operations)**

```
sum = sum + 1;
```

write   **read**
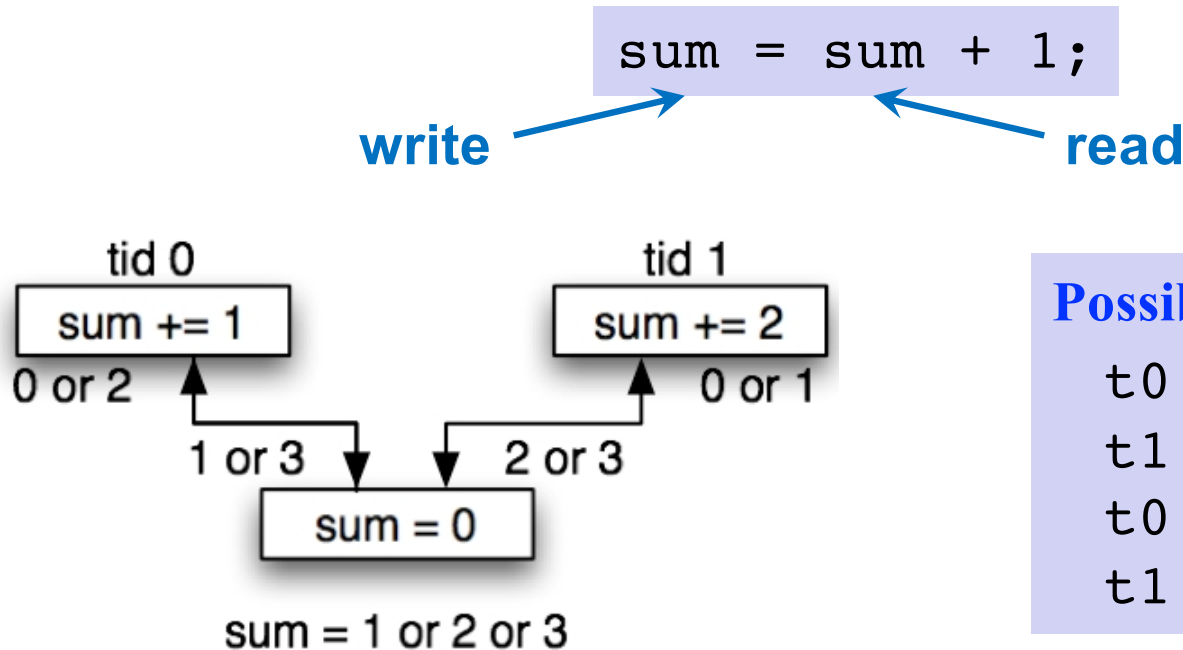
Possible scenarios

```
t0  r  0        t0  r  0
t1  r  0        t0  w  1
t0  w  1        t1  r  1
t1  w  2        t1  w  3
```

Critical Section

- **Critical section degrades scalability, *cf.* Amdahl's law**

$$T_P = fT_1 + (1-f)\frac{T_1}{P}$$

$$S_P = \frac{T_1}{T_P} = \frac{1}{f + \frac{1-f}{P}} \rightarrow \frac{1}{f} \quad (P \rightarrow \infty)$$

```
for (i=tid; i<NBIN; i+=nthreads) {
  x = (i+0.5)*step;
  #pragma omp critical        f ~ 0.5
  sum += 4.0/(1.0+x*x);
}
```

- **How to get rid of the critical section?**

Avoid Critical Section: `omp_pi.c`

Data privatization: Give each thread a dedicated accumulator

```c
#include <stdio.h>
#include <omp.h>
#define NBIN 100000
#define MAX_THREADS 8
void main() {
  int nthreads,tid;
  double step,sum[MAX_THREADS]={0.0},pi=0.0;
  step = 1.0/NBIN;
  #pragma omp parallel private(tid)
  {
    long long i;
    double x;
    nthreads = omp_get_num_threads();
    tid = omp_get_thread_num();
    for (i=tid; i<NBIN; i+=nthreads) {
      x = (i+0.5)*step;
      sum[tid] += 4.0/(1.0+x*x);
    }
  }
  for(tid=0; tid<nthreads; tid++) pi += sum[tid]*step;
  printf("PI = %f\n",pi);
}
```

Array of partial sums for multi-threads

Private accumulator

Inter-thread reduction

Avoid Critical Section: "Wrong" Way

```c
#include <stdio.h>
#include <omp.h>
#define NBIN 100000
void main() {
  double step,sum=0.0,pi;
  step = 1.0/NBIN;
  # pragma omp parallel
  {
    int nthreads,tid
    long long i;
    double x;
    nthreads = omp_get_num_threads();
    tid = omp_get_thread_num();
    for (i=tid; i<NBIN; i+=nthreads) {
      x = (i+0.5)*step;
      // #pragma omp critical
      sum += 4.0/(1.0+x*x);
    }
  }
  pi = sum*step;
  printf("PI = %f\n",pi);
}
```

omp_pi_noncritical.c

Everything You Learned About Parallel Computing is
Wrong for Machine Learning!

Prof. Kunle Olukotun (Stanford)
(Sep. 28, 2017 at USC)

**HOGWILD!: A Lock-Free Approach to Parallelizing
Stochastic Gradient Descent**

F. Niu *et al.*, NeurIPS11

```
[anakano@hpc-login3 src]$ ./omp_pi_critical
PI = 3.141593
[anakano@hpc-login3 src]$ ./omp_pi_noncritical
PI = 0.558481          16-thread run
```

Load Balancing

- **Interleaved assignment of loop-index values to threads balances the loads among the threads**

```
for (i=tid; i<NBIN; i+=nthreads) {
        ...
}
```

A bad example

Most Widely Used Construct

- **OpenMP for:** Distribute the loop iterations across the threads; can be combined with OpenMP parallel to achieve multithreading in just one line.

```c
#include <omp.h>
#include <stdio.h>
#define NBIN 100000
void main() {
  long long i;
  double step,x,sum=0.0,pi;

  step = 1.0/NBIN;
  omp_set_num_threads(2);
  # pragma omp parallel for private (i,x) reduction(+:sum)
  for (i=0; i<NBIN; i++) {
    x = (i+0.5)*step;
    sum += 4.0/(1.0+x*x);
  }
  pi = sum*step;
  printf("PI = %f\n",pi);
}
```

Reduction clause performs automatic thread reduction

- **OpenMP parallelization is very easy!**

Where to Go from Here

- **OpenMP tutorial introducing most constructs**
  https://computing.llnl.gov/tutorials/openMP

- **OpenMP 4.5 has added many constructs to support modern hardware architectures**

  `#pragma omp target`: **Offload computation to accelerators like graphics processing units (GPUs)**

  `#pragma omp simd`: **Explicit control over single instruction multiple data (or vector) operations**

  https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf