Lab2
Pthread & OpenMP

18 Oct 2018
Parallel Programming

SLURM quick reference

```
srun [flags] ./prog
=========== or ===========
#!/bin/bash
#SBATCH [flags]
srun ./prog  # (MPI)
./prog       # (non-MPI)
--------- run with: --------
sbatch job.sh
```

[flags]:

-p debug or batch

-N number of nodes

-n number of processes

-c CPUs per process

-t additional time limit

-J name of job

Outline

- **Pthread**
  - **Hello world**
  - Mutex
  - Condition variable
- OpenMP
- OpenMP + MPI

Running pthread programs on apollo

Type man pthread_create in terminal to see this

```
SYNOPSIS
       #include <pthread.h>

       int pthread_create(
           pthread_t *thread, const pthread_attr_t *attr,
           void *(*start_routine) (void *), void *arg);

       Compile and link with -pthread.
```

Running pthread programs on apollo

`cp /home/ta/lab2/hello_pthread.c .`

compile

```
gcc hello_pthread.c -o hello_pthread -pthread
```

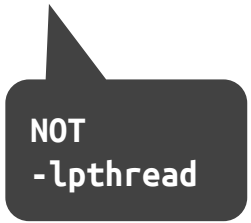
execute

```
srun -c4 -n1 ./hello_pthread 4
```

`-c4` means 4 CPUs per process

`-n1` means 1 process

You can use sbatch as well



Outline

- **Pthread**
  - Hello world
  - **Mutex**
  - Condition variable
- OpenMP
- OpenMP + MPI

Pthread Lock/Mutex Routines

- To use mutex, it must be declared as of type **pthread_mutex_t** and initialized with **pthread_mutex_init()**
- A mutex is destroyed with **pthread_mutex_destroy()**
- A critical section can then be protected using **pthread_mutex_lock()** and **pthread_mutex_unlock()**
- Example:

```
#include "pthread.h"
pthread_mutex_t   mutex;
pthread_mutex_init (&mutex, NULL);
pthread_mutex_lock(&mutex);          // enter critical section

      Critical Section

pthread_mutex_unlock(&mutex);        // leave critical section
pthread_mutex_destroy(&mutex);
```

specify default attribute for the mutex

Mutex

---

man pthread_mutex_init

```
#include <pthread.h>
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;


int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

man pthread_mutex_lock

Mutex:
[Practice 1] approximate π using pthread

```
gcc pi_pthread.c -o pi_pthread -pthread -lm
```

code filename

executable filename

we're using pthread

```
srun -c4 -n1 ./pi_pthread 4 500000000
```

number of threads

number of points

Outline

- **Pthread**
  - Hello world
  - Mutex
  - **Condition variable**
- OpenMP
- OpenMP + MPI

Condition Variables (CV)

- CV represent some condition that a thread can:
  - Wait on, until the condition occurs; or
  - Notify other waiting threads that the condition has occurred
- Three operations on condition variables:
  - wait() --- Block until another thread calls signal() or broadcast() on the CV
  - signal() --- Wake up one thread waiting on the CV
  - broadcast() --- Wake up all threads waiting on the CV
- In Pthread, CV type is a pthread_cond_t
  - Use **pthread_cond_init()** to initialize
  - **pthread_cond_wait (&theCV, &somelock)**
  - **pthread_cond_signal (&theCV)**
  - **pthread_cond_broadcast (&theCV)**

Condition variable

```
#include <pthread.h>
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_wait(
        pthread_cond_t *restrict cond,
        pthread_mutex_t *restrict mutex);
```

man pthread_cond_broadcast

man pthread_cond_timedwait

Condition variable

`pthread_cond_signal`

Make 1 thread that called **`pthread_cond_wait`** to continue

`pthread_cond_broadcast`

Make all threads that called **`pthread_cond_wait`** to continue

`pthread_cond_wait`

Wait for some other thread to call **`pthread_cond_signal`**

[Practice 2] Condition variable

Compile and execute pthread_cond.c under lab2/

you should see →
which is incorrect

```
Threads have been created
Enter 4 values
Values filled in array are
0
0
0
0
```

Modify the program using condition variable so it will wait until you input the values then print

Outline

- Pthread
  - Hello world
  - Mutex
  - Condition variable
- **OpenMP**
- OpenMP + MPI

Running OpenMP programs on apollo: example (/home/ta/lab2)

---

compile

```
gcc hello_omp.c -o hello_omp -fopenmp
```

execute

```
srun -c4 -n1 ./hello_omp
```

-c4 means 4 CPUs per process

-n1 means 1 process

You can use sbatch as well

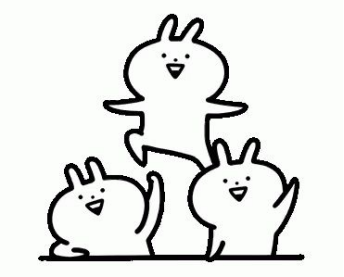
Try different number of threads!

OpenMP automatically detects number of CPUs from SLURM (affinity) So we don't have to specify it again

Count prime numbers: sequential version

```
gcc -lm prime.c -o prime
srun ./prime 1000
srun ./prime 10000000
```

Count prime numbers:
[Practice 3] OpenMP



1. Modify the sequential prime.c with openmp

2. Try to see the effect of changing
   dynamic/static **scheduling**
   chunk size
   number of **threads**

[example commands]

```
gcc -lm prime_omp.c -o prime_omp -fopenmp
srun -c4 -n1 ./prime_omp 10000000
```

Outline

- Pthread
  - Hello world
  - Mutex
  - Condition variable
- OpenMP
- **OpenMP + MPI**

Hybrid MPI and OpenMP program

```
mpicc hello_hybrid.c -o hello_hybrid -fopenmp
```

We're using MPI

We're using OpenMP

```
srun -c3 -n2 ./hello_hybrid
```

3 threads

2 processes

Hybrid MPI and OpenMP program: Hello World

```
srun -c 3 -n2 -N2 ./hello_hybrid
Hello apollo32: rank  0/ 2, thread  0/ 3
Hello apollo32: rank  0/ 2, thread  1/ 3
Hello apollo32: rank  0/ 2, thread  2/ 3
Hello apollo33: rank  1/ 2, thread  0/ 3
Hello apollo33: rank  1/ 2, thread  1/ 3
Hello apollo33: rank  1/ 2, thread  1/ 3
```

Hybrid MPI and OpenMP program: **[Practice 4]** Approximate π

Use MPI and OpenMP to approximate π

(You can refer to your lab1b code)