Pthreads

High Performance Computing

Master in Applied Artificial Intelligence

Nuno Lopes

Motivation

- Library for developing parallel applications using shared memory.
- Assumes a POSIX-compliant operating system as its base.
- Library can be embedded in any programming language, usually C.
- The use of threads is achieved by invoking functions from the library.

Compilation and Execution of Pthreads Programs

- Include library headers:
  - #include <pthread.h>
- Linker option:
  - -lpthread
  - gcc - lpthread hello.c -o hello

Pthread API to create and join threads

```
pthread_create(
        pthread_t* thread_p
        const pthread_attr_t* attr_p
        void* (*start_routine)(void*)
        void* arg_p );
```

- thread_p: thread object reference,
- attr_p: creation attributes, NULL
- start_routine: function to execute
- arg_p: function argument

- Generic function header:

void* start_routine(**void*** args_p);

Pthread API to create and join threads

```
pthread_join(
  pthread_t thread       /* in */,
  void** ret_val_p       /* out */ );
```

Example Incremental Application

```
// global vars
long long n;        // set to number of iterations
long long thread_count;    // set to number threads
long long sum;  // global sum value

// thread operation
void* Increment(void* rank) {
        long my_rank = (long) rank;
        long long my_n = n/thread count; // even division
        long long my_first_i = my_n * my_rank;
        long long my_last_i = my_first_i + my_n;
        printf(“Thread %ld range: %ld to %ld\n”, rank, my_first_i, my_last_i);
        for( i=my_first_i; i<my_last_i;i++)
                sum += 1;
```

Example Increment Application

```
void* Increment(void* rank);

int main(int argc, char* argv[]) {
  long thread;
  pthread_t* thread_handles;
  thread_count = strtol(argv[1], NULL, 10);
  thread_handles = malloc (thread_count * sizeof(pthread_t));
  ...
  for (thread = 0; thread < thread_count; thread++)
    pthread_create(&thread_handles[thread], NULL, Increment, (void*) thread);
  ...
  for (thread = 0; thread < thread_count; thread++)
    pthread_join(thread_handles[thread], NULL);
  ...
  printf (“Final value: %ld\n”, sum);
  free(thread_handles);
```

Mutual Exclusion Mechanism

Mutual Exclusion Functions

- Request to access shared resource

```
pthread_mutex_lock ( pthread_mutex_t* mutex_p /* in/out */);
```

- Release of resource:

```
pthread_mutex_unlock ( pthread_mutex_t* mutex_p /* in/out */);
```

- Creation of *mutex* auxiliary structures:

```
pthread_mutex_init (
    pthread_mutex_t* mutex_p                /* out */,
    const pthread_mutexattr_t* attr_p       /* in */);
```

- Release of auxiliary structures:

```
pthread_mutex_destroy ( pthread_mutex_t* mutex_p /* in/out */);
```

- Auxiliary structure:

```
pthread_mutex_t
```

Mutual Exclusion Example

```
// global variables
pthread_mutex_t mutex;

// main init
pthread_mutex_init( &mutex, NULL);

// thread function
void* thread_function(void* arg)

    pthread_mutex_lock( &mutex );
    ...
    pthread_mutex_unlock( &mutex );
```

Exercises

Deterministic Incremental

- Using as base the incremental code example, implement a parallel program that uses multiple threads to increment a shared global variable *n* times.

PI Value Estimation

- Based on the Gregory-Leibniz Series, it is possible to estimate the value of PI:

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + (-1)^n \frac{1}{2n+1} + \cdots\right).$$

$$\pi = 4\left[1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right] = 4\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}.$$

- Implement a parallel program to use multiple threads.