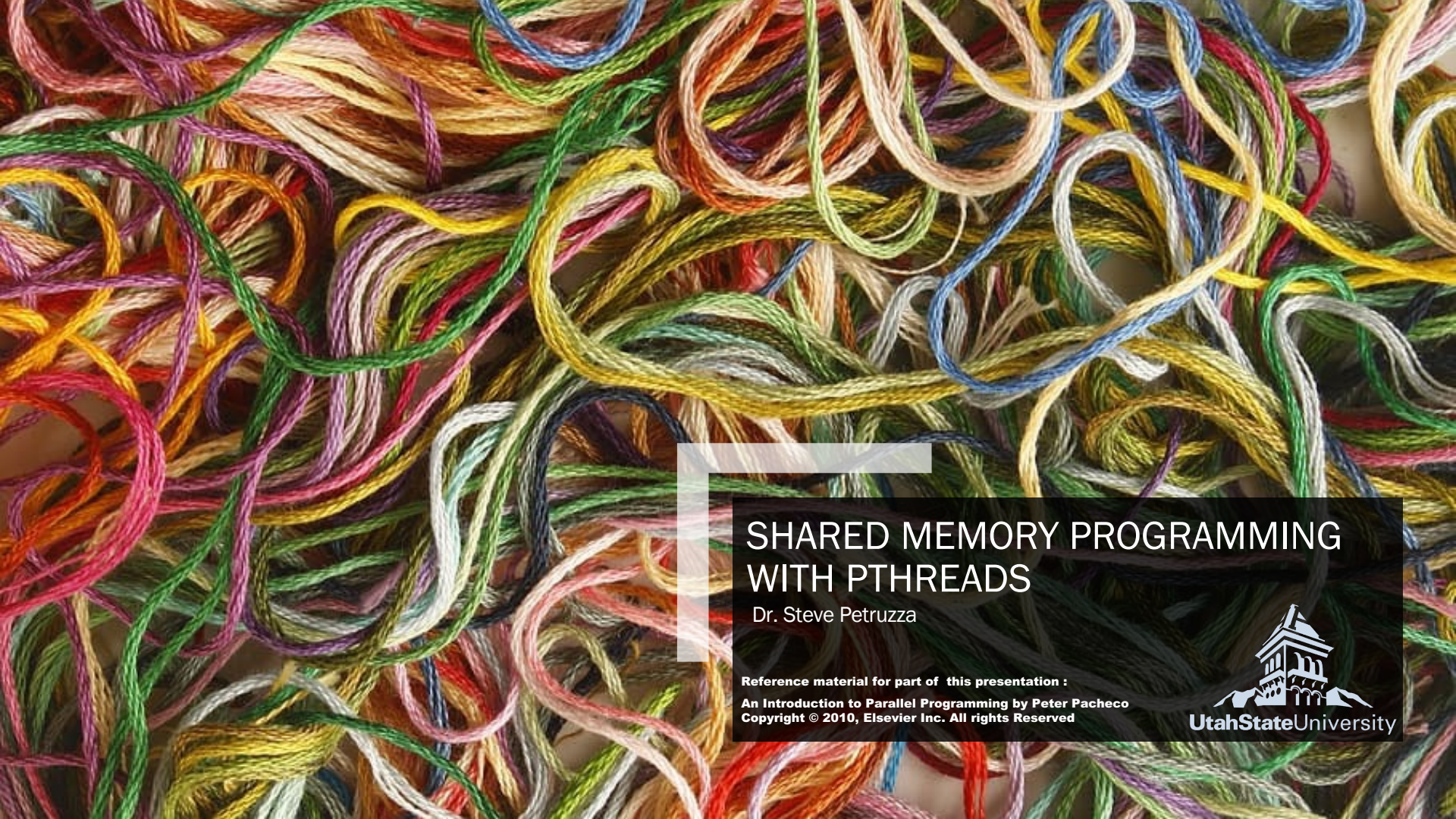


SHARED MEMORY PROGRAMMING WITH PTHREADS

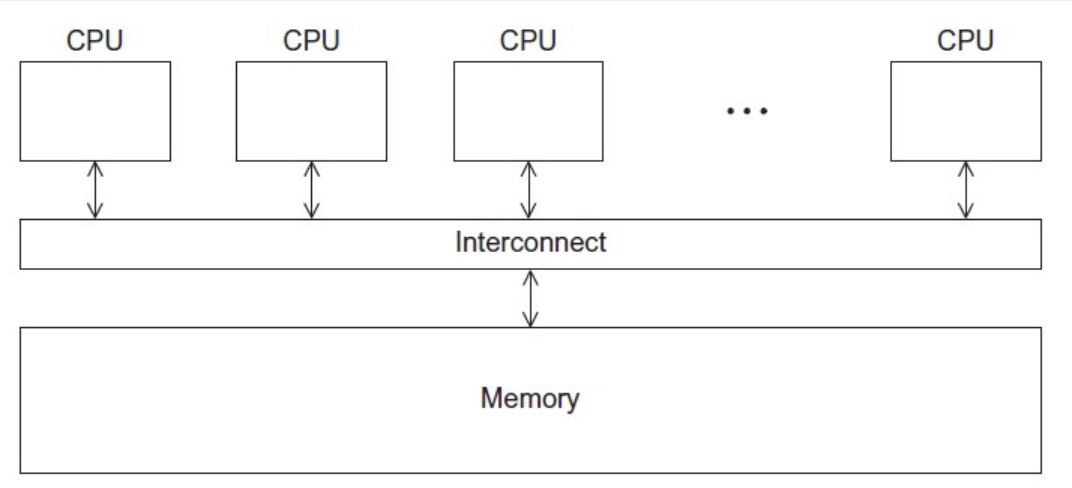
Dr. Steve Petruzza

Reference material for part of this presentation :

An Introduction to Parallel Programming by Peter Pacheco

UtahStateUniversity

A Shared Memory System



Processes and Threads

- A process is an instance of a running (or suspended) program.
- Threads are analogous to a “light-weight” process.
- In a shared memory program a single process may have multiple threads of control.

POSIX®Threads

- Also known as Pthreads.
- A standard for Unix-like operating systems.
- A library that can be linked with C programs.
- Specifies an application programming interface (API) for multi-threaded programming.

Hello World! (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

/* Global variable:  accessible to all threads */
int thread_count;

void *Hello(void* rank);  /* Thread function */

int main(int argc, char* argv[]) {
   long       thread;  /* Use long in case of a 64-bit system */
   pthread_t* thread_handles;

   /* Get number of threads from command line */
   thread_count = strtol(argv[1], NULL, 10);

   thread_handles = malloc (thread_count*sizeof(pthread_t));
```

declares the various Pthreads functions, constants, types, etc.

Hello World! (2)

```
   for (thread = 0; thread < thread_count; thread++)
      pthread_create(&thread_handles[thread], NULL,
          Hello, (void*) thread);

   printf("Hello from the main thread\n");

   for (thread = 0; thread < thread_count; thread++)
      pthread_join(thread_handles[thread], NULL);

   free(thread_handles);
   return 0;
}  /* main */
```

Hello World! (3)

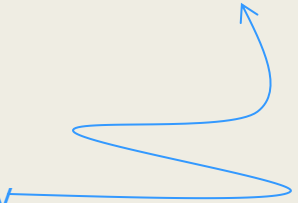
```
void *Hello(void* rank) {
   long my_rank = (long) rank;  /* Use long in case of 64-bit system */

   printf("Hello from thread %ld of %d\n", my_rank, thread_count);

   return NULL;
}  /* Hello */
```

Caveat

- The Pthreads API is only available on POSIX systems — Linux, MacOS X, Solaris, HPUX, …
- Online compilers:
  - *https://www.onlinegdb.com/online_c++_compiler*
  - *https://rextester.com/l/c_online_compiler_gcc*

Compiling a Pthread program

```
gcc –g –Wall –o pth_hello pth_hello.c –lpthread
```



Running a Pthreads program

./ pth_hello <number of threads>

./ pth_hello 1

Hello from the main thread
Hello from thread 0 of 1

./ pth_hello 4

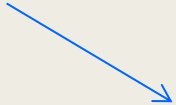
Hello from the main thread
Hello from thread 0 of 4
Hello from thread 1 of 4
Hello from thread 2 of 4
Hello from thread 3 of 4

Global variables

- Can introduce subtle and confusing bugs!
- Limit use of global variables to situations in which they're really needed.
  - *Shared variables.*

Starting the Threads

pthread.h



pthread_t

One object for each thread.

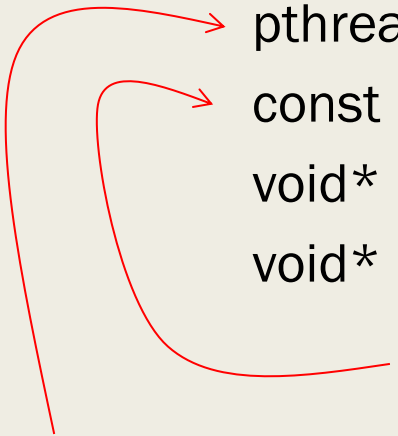
```
int pthread_create (
        pthread_t*  thread_p /* out */ ,
        const pthread_attr_t*  attr_p /* in */ ,
        void*  (*start_routine ) ( void ) /* in */
,
        void*  arg_p /* in */ ) ;
```

pthread_t objects

- Opaque
- The actual data that they store is system-specific.
- Their data members aren't directly accessible to user code.
- However, the Pthreads standard guarantees that a pthread_t object does store enough information to uniquely identify the thread with which it's associated.

A closer look (1)

```
int pthread_create (
      pthread_t*  thread_p /* out */ ,
      const pthread_attr_t*  attr_p /* in */ ,
      void*  (*start_routine ) ( void ) /* in */ ,
      void*  arg_p /* in */ ) ;
```



We won't be using, so we just pass NULL.

Allocate before calling.

A closer look (2)

```
int pthread_create (
        pthread_t*  thread_p /* out */ ,
        const pthread_attr_t*  attr_p /* in */ ,
        void*  (*start_routine ) ( void ) /* in */ ,
        void*  arg_p /* in */ ) ;
```

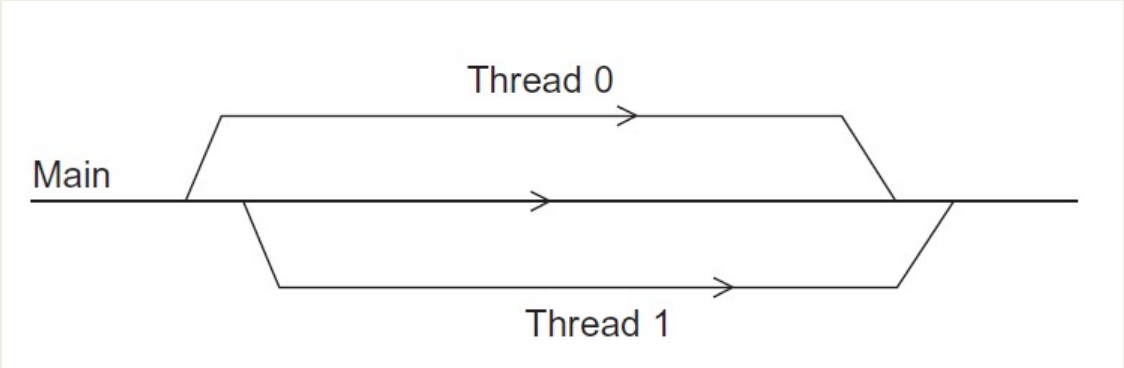
Pointer to the argument that should be passed to the function *start_routine*.

The function that the thread is to run.

Function started by pthread_create

- Prototype:
  void* thread_function (void* args_p) ;
- Void* can be cast to any pointer type in C.
- So args_p can point to a list containing one or more values needed by thread_function.
- Similarly, the return value of thread_function can point to a list of one or more values.

Running the Threads



Main thread forks and joins two threads.

Stopping the Threads

- We call the function pthread_join once for each thread.
- A single call to pthread_join will wait for the thread associated with the pthread_t object to complete.

Threads in C++11

- Easier to program (object oriented implementation)
- You can use the same library on Windows (using MinGW compiler)

C++11 Threads

C++11 threads

```cpp
#include <iostream>
#include <thread>

//This function will be called from a thread

void call_from_thread() {
    std::cout << "Hello, World" << std::endl;
}

int main() {
    //Launch a thread
    std::thread t1(call_from_thread);

    //Join the thread with the main thread
    t1.join();

    return 0;
}
```

C Pthreads

```cpp
#include <iostream>
#include <pthread.h>

//This function will be called from a thread

void *call_from_thread(void *) {
    std::cout << "Launched by thread" << std::endl;
    return NULL;
}

int main() {
    pthread_t t;

    //Launch a thread
    pthread_create(&t, NULL, call_from_thread, NULL);

    //Join the thread with the main thread
    pthread_join(t, NULL);
    return 0;
}
```

Linux: g++ -std=c++11 -lpthread file_name.cpp
OSX: clang++ -std=c++11 -stdlib=libc++ file_name.cpp

Different ways to define a thread “function” in C++11

Thread using function pointer

```
#include <iostream>
#include <thread>

void thread_function()
{
    for(int i = 0; i < 10000; i++);
        std::cout<<"thread function Executing"<<std::endl;
}

int main()
{

    std::thread threadObj(thread_function);
    for(int i = 0; i < 10000; i++);
        std::cout<<"Display From MainThread"<<std::endl;
    threadObj.join();
    std::cout<<"Exit of Main function"<<std::endl;
    return 0;
}
```

Thread using function object

```
#include <iostream>
#include <thread>
class DisplayThread
{
public:
    void operator()()
    {
        for(int i = 0; i < 10000; i++)
            std::cout<<"Display Thread Executing"<<std::endl;
    }
};

int main()
{
    std::thread threadObj( (DisplayThread()) );
    for(int i = 0; i < 10000; i++)
        std::cout<<"Display From Main Thread "<<std::endl;
    std::cout<<"Waiting For Thread to complete"<<std::endl;
    threadObj.join();
    std::cout<<"Exiting from Main Thread"<<std::endl;
    return 0;
}
```

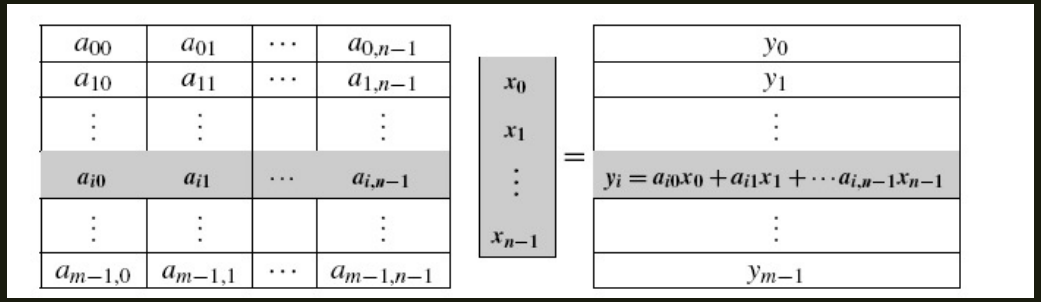
Different ways to define a thread “function” in C++11 (2)

Thread using Lambda Function

```
#include <iostream>
#include <thread>
int main()
{
    int x = 9;
    std::thread threadObj([]{
            for(int i = 0; i < 10000; i++)
                std::cout<<"Display Thread Executing"<<std::endl;
            });

    for(int i = 0; i < 10000; i++)
        std::cout<<"Display From Main Thread"<<std::endl;

    threadObj.join();
    std::cout<<"Exiting from Main Thread"<<std::endl;
    return 0;
}
```

Syntax of a Lambda Function

```
[ capture clause ] (parameters) -> return-type
{
   definition of method
}
```



$$\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,n-1} \\ \vdots & \vdots & & \vdots \\ a_{i0} & a_{i1} & \cdots & a_{i,n-1} \\ \vdots & \vdots & & \vdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_i = a_{i0}x_0 + a_{i1}x_1 + \cdots a_{i,n-1}x_{n-1} \\ \vdots \\ y_{m-1} \end{bmatrix}$$

MATRIX-VECTOR MULTIPLICATION IN PTHREADS

Serial pseudo-code

```
/* For each row of A */
for (i = 0; i < m; i++) {
    y[i] = 0.0;
    /* For each element of the row and each element of x */
    for (j = 0; j < n; j++)
        y[i] += A[i][j]* x[j];
}
```

$$y_i = \sum_{j=0}^{n-1} a_{ij} x_j$$

Using 3 Pthreads

Thread	Components of y
0	y[0], y[1]
1	y[2], y[3]
2	y[4], y[5]

thread 0

```
y[0] = 0.0;
for (j = 0; j < n; j++)
   y[0] += A[0][j]* x[j];
```

general case

```
y[i] = 0.0;
for (j = 0; j < n; j++)
   y[i] += A[i][j]*x[j];
```

Pthreads matrix-vector multiplication

```
void *Pth_mat_vect(void* rank) {
   long my_rank = (long) rank;
   int i, j;
   int local_m = m/thread_count;
   int my_first_row = my_rank*local_m;
   int my_last_row = (my_rank+1)*local_m - 1;

   for (i = my_first_row; i <= my_last_row; i++) {
      y[i] = 0.0;
      for (j = 0; j < n; j++)
         y[i] += A[i][j]*x[j];
   }

   return NULL;
}  /* Pth_mat_vect */
```