COMP1001

Computer Systems

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk

Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

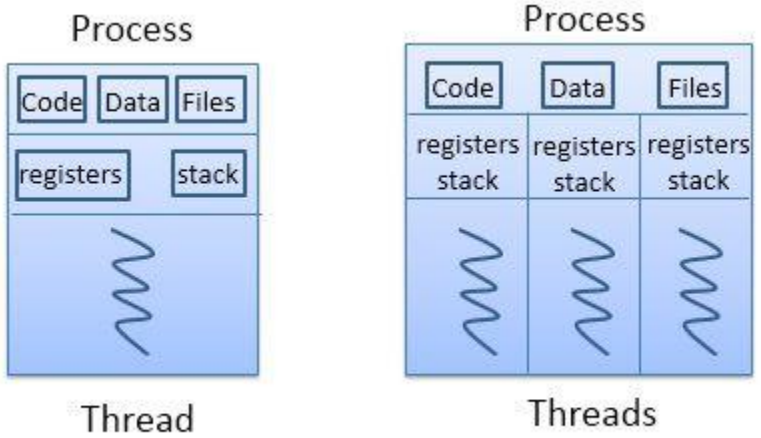
School of Computing
(University of Plymouth)

Outline

- Introduction to Threads
- Introduction to Pthreads
- Introduction to OpenMP

What is difference between thread, process and program?

- **Program:**
  - Program is an executable file containing the set of instructions written to perform a specific job on your computer
  - For example, *skype.exe* is an executable file containing the set of instructions which help us to run *skype*
- **Process:**
  - Process is an executing instance of a program
  - For example, when you double click on the *skype.exe* on your computer, a process is started that will run the skype program
- **Thread:**
  - Thread is the smallest executable unit of a process
  - For example, when you run skype program, OS creates a process and starts the execution of the main thread of that process
  - A process can have multiple threads
  - All threads of the same process share memory of that process



Multithreading

Processes and Threads

- A process is a program in memory along with its dynamically allocated storage (heap), its stack storage and its execution context (state of registers and instruction pointer)
- A process might have more than one threads – multithreaded process

- A process can be broken down into
  1. Text segment, heap segment, static data segments
  2. Stack , instruction pointer and registers
     - Each thread has its own stack, instruction pointer and registers
     - Each thread has its own thread ID

Threads – Advantages of using threads

- Threads are light compared to processes
  - The OS does not have to create a new memory map (recall previous lecture) for a new thread, as it does for a process
  - The OS does not have to keep track of open files amongst different threads
  - Switching between threads of the same process has less overhead than process switch
  - A multithreading application scales better by adding more CPU cores
- Programming is easier as all the threads share global variables

POSIX Threads

- The POSIX thread libraries are a standards based thread API for C/C++.
- It allows to create a multiprocessing flow.
- It is most effective on multi-core systems where the process flow can be scheduled to run on multiple cores and thus improve performance.
- Creating multiple threads require less overhead than "forking" or spawning a new process because the system does not create a new memory map.
- Performance gains are also found on uniprocessor systems which exploit latency in I/O and other system functions which may halt process execution.
  - One thread may execute while another is waiting for I/O or some other system latency

Create a new Thread using pthreads

- Pthreads are **defined** by using:
  - ***Pthread_t*** my_thread;
  - We must include the ***#include <pthread.h>*** library
- A thread is **created** via
  - *Pthread_t* my_thread;

    pthread_create(&my_thread, NULL, func, arg);

  - int ***pthread_create***(pthread_t * thread, *//this is the thread ID*

    const pthread_attr_t * attr, *//we will set this to NULL (default thread attributes)*

    void * (*start_routine)(void *), *//pointer to the function to be threaded. Function has a single argument: pointer to void.*

    void *arg); *//pointer to argument of function. To pass multiple arguments, send a pointer to a structure.*

Pthreads - Exit and Join

- **Exit a thread.** A thread can exit by
  - calling ***pthread_exit*** or
  - ***pthread_join()***

- **Join two threads**
  - By calling Pthread_join( ) command, the program will wait here until the thread finishes

Pthreads example (1)


```
int main() {

    pthread_t thread1, thread2, thread3, thread4;

    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int  iret1, iret2;

    iret1 = pthread_create( &thread1, NULL, print_mes, (void*) message1);
    iret2 = pthread_create( &thread2, NULL, print_mes, (void*) message2);

    pthread_join( thread1, NULL); //wait until thread1 finishes
    pthread_join( thread2, NULL);  //wait until thread2 finishes

    printf("Thread 1 completed, returned: %d\n",iret1);
    printf("Thread 2 completed, returned: %d\n",iret2);

return 0; }
```

```
void *print_mes( void *ptr )
{
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}
```

Thread1 runs its own print_mes()

Thread2 runs its own print_mes()

Pthreads example (2)


```
void *print_mes ( void *ptr )
{
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}
```

- this function takes as input a pointer to void (pointer to anything).
- It returns a pointer to void too.
- A void pointer is a pointer that has no associated data type with it.
- A void pointer can hold address of any type and can be typecasted to any type.

Pthreads example (3)
What does this program print?

- The 1st time we run the program it generates:

Program's output:

Thread1
Thread2
Thread1 completed, returned: 0
Thread2 completed, returned: 0

- The 2nd time we run the program it generates:

Program's output:

Thread2
Thread1
Thread1 completed, returned: 0
Thread2 completed, returned: 0

- ***Each thread runs its own copy of the function, at any order.***

What if we delete the join() function?


```
int main() {

    pthread_t thread1, thread2, thread3, thread4;

    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int  iret1, iret2;

    iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1); //create
thread1 which will run its own print_message_function()
    iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);

    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);

    printf("Thread 1 completed, returned: %d\n",iret1);
    printf("Thread 2 completed, returned: %d\n",iret2);

return 0; }
```

```
void *print_message_function( void *ptr )
{
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}
```

Program's output:

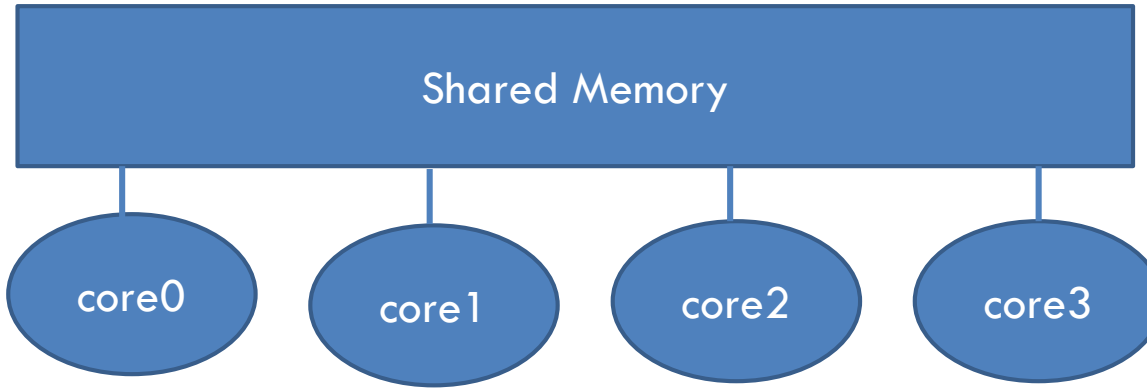
Thread1 completed, returned: 0
Thread2 completed, returned: 0
Thread2
Thread1

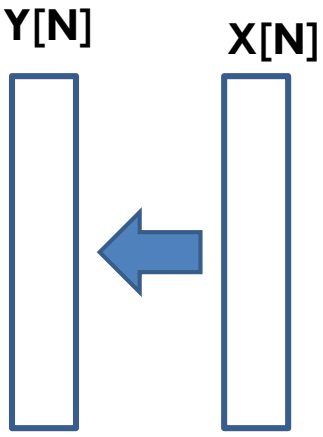
Wrong output

How can we parallelize this program using pthreads? sqrt.c program


```
#define N 1000
float X[N], Y[N];
int i, j;

for (j=0; j<N; j++)
   Y[i] = sqrt( X[j] ) ;
```

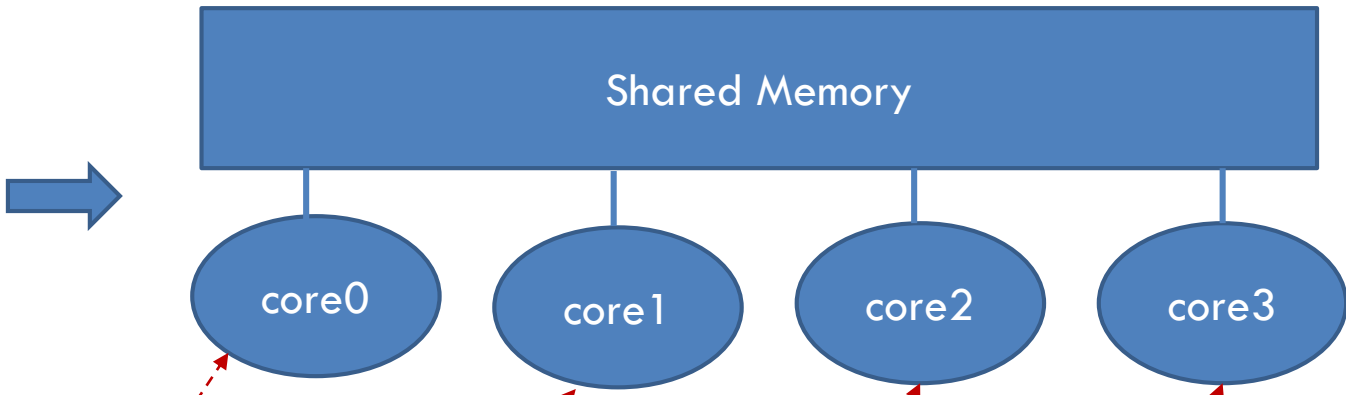




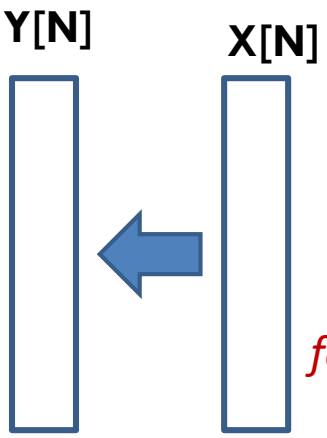
How can we parallelize this program using pthreads? sqrt.c program


```
#define N 1000
float X[N], Y[N];
int i, j;

for (j=0; j<N; j++)
   Y[i] = sqrt( X[j] ) ;
```



- There are 4 cores, so let's use 4 threads
- We can split the loop into four equal parts and each core executes its part




```
for (j=0; j<250; j++)
   Y[i] = sqrt( X[j] ) ;
```

```
for (j=250; j<500; j++)
   Y[i] = sqrt( X[j] ) ;
```

```
for (j=500; j<750; j++)
   Y[i] = sqrt( X[j] ) ;
```

```
for (j=750; j<1000; j++)
   Y[i] = sqrt( X[j] ) ;
```

How can we parallelize this program using pthreads? sqrt.c program


```
...
for (thread_num = 0; thread_num < NUM_THREADS; thread_num++)
    pthread_create(&thread_handles[thread_num], NULL, sqrt, (void*) thread_num);
.....

void *sqrt (void* thread_num) {

  long my_rank = (long) thread_num;
  int i, j;
  int local = N/NUM_THREADS;

  int starting_row = my_rank * local;
  int ending_row = starting_row + local - 1;

  for (i = starting_row; i <= ending_row; i++) {
       Y[i] = sqrt ( X[j] ) ;    }

  return 0;
}
```

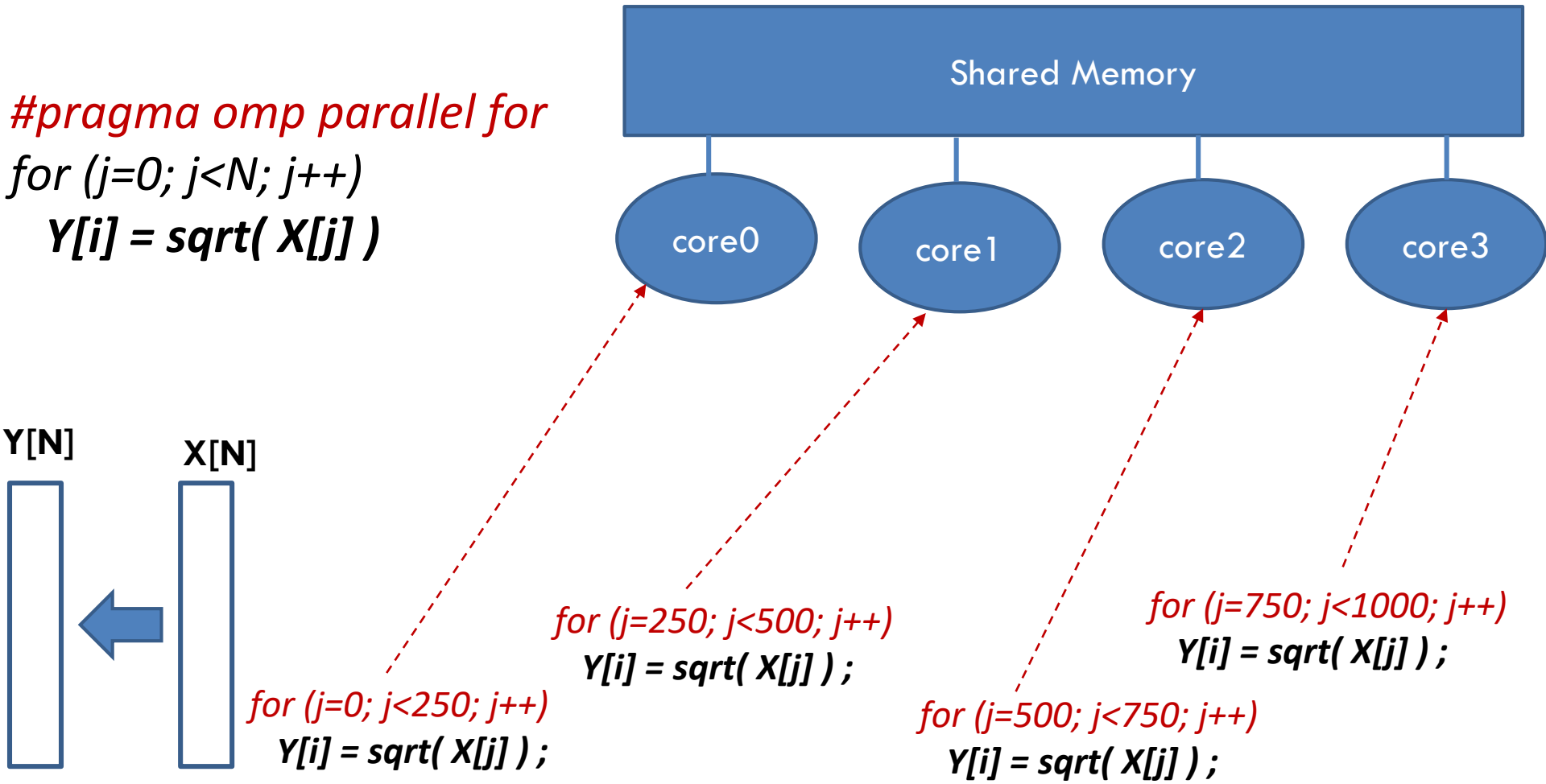
- ***Create 'NUM_THREADS' threads where they all execute sqrt() function***
- *Let's assume N=1000 and NUM_THREADS=4*

Thread	Starting row	Ending row
0	0	249
1	250	499
2	500	749
3	750	999

How can we parallelize the sqrt program by using OpenMP framework ?

- This program yields the same behaviour as the previous one

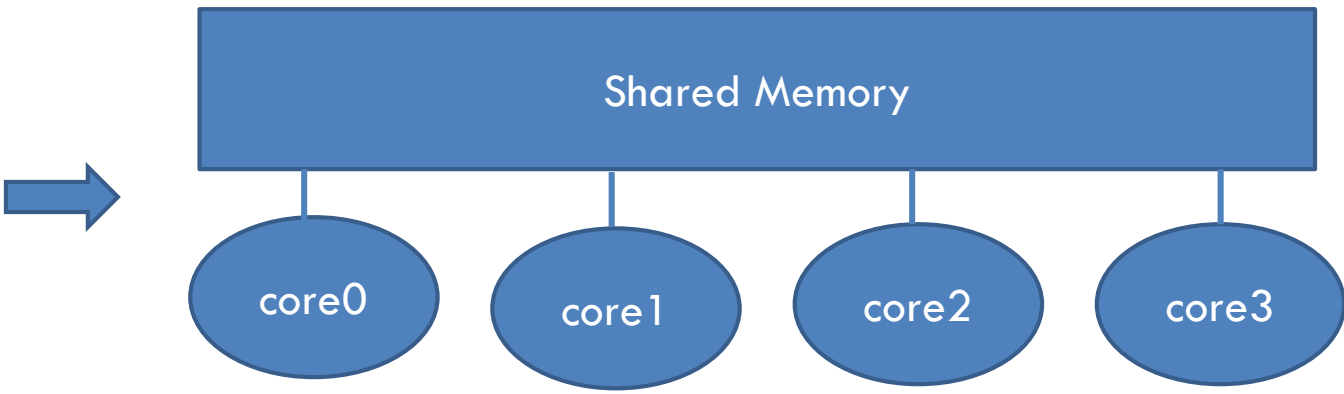
```
#pragma omp parallel for
for (j=0; j<N; j++)
   Y[i] = sqrt( X[j] )
```

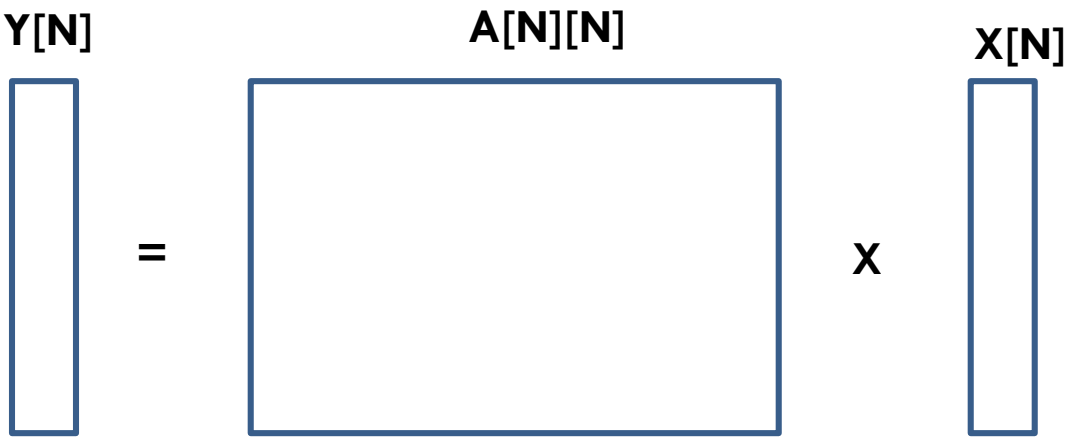


How can we parallelize this program using pthreads? MVM_pthreads.c program


```
float A[N][N];
float X[N], Y[N];
int i,j;

for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```



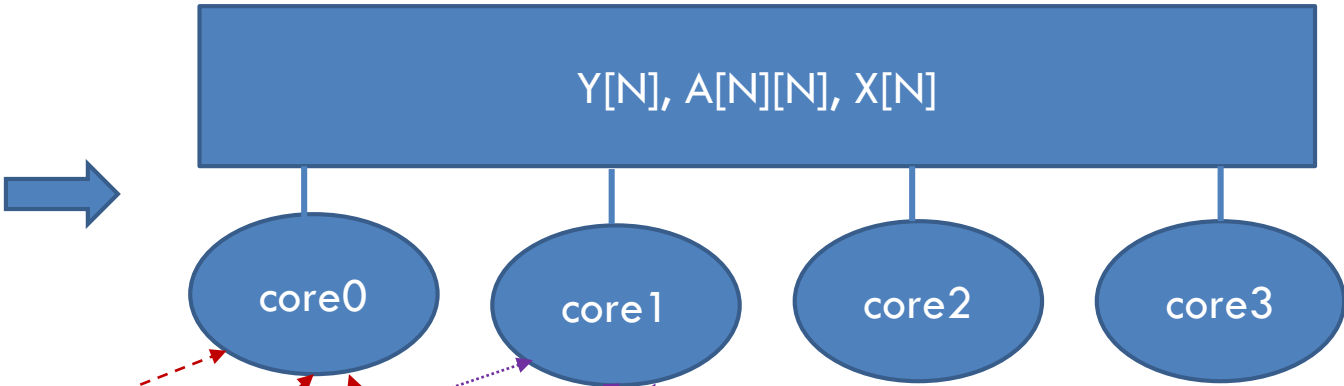


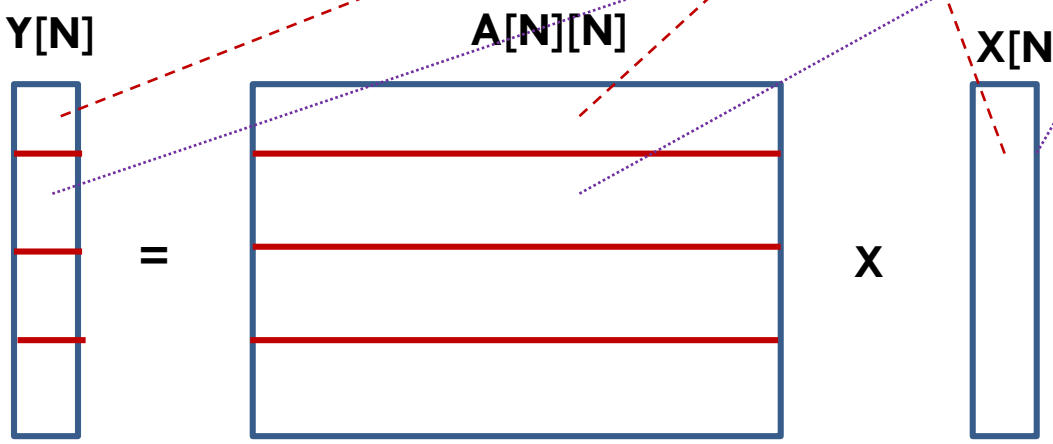
How can we parallelize this program using pthreads? MVM_pthreads.c program

We can split i loop into four equal parts, one for each core

```
float A[N][N];
float X[N], Y[N];
int i,j;

for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```



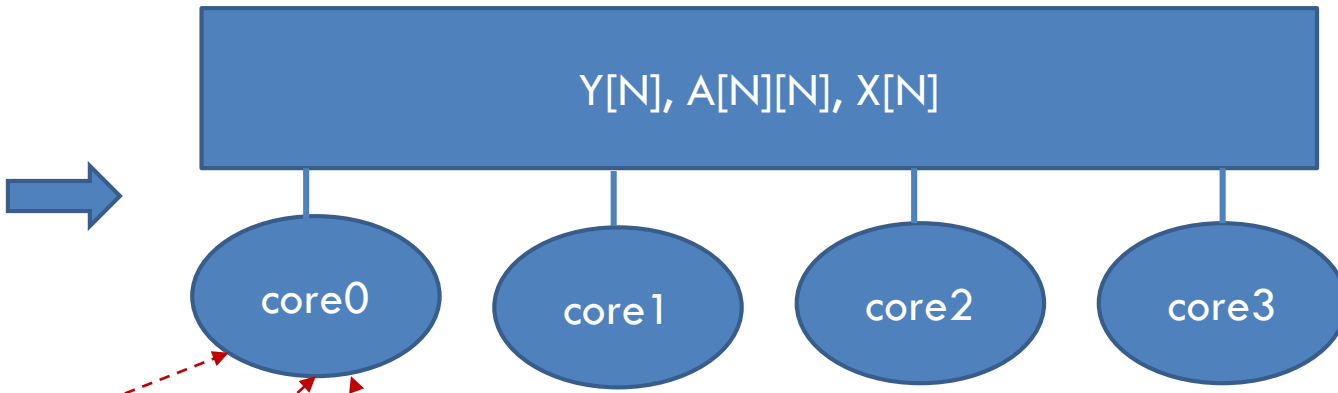


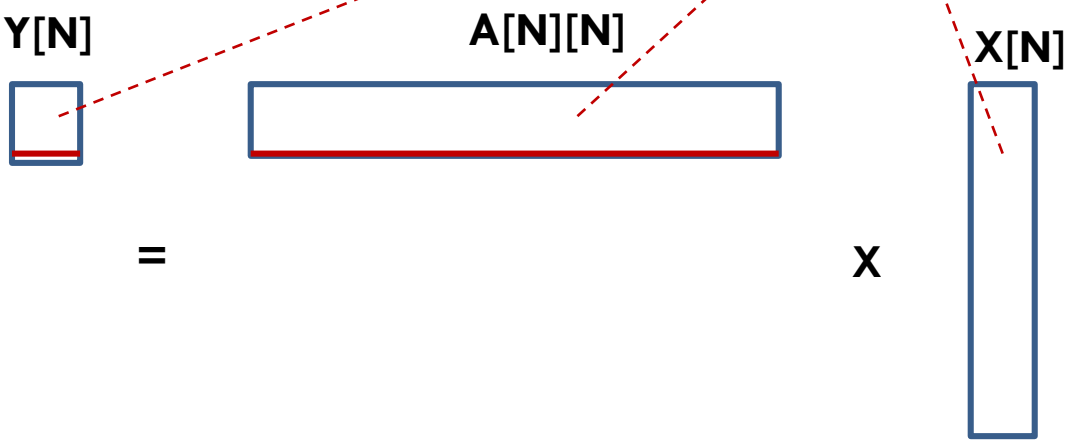
How can we parallelize this program using pthreads? MVM_pthreads.c program

We can split i loop into four equal parts, one for each core

```
float A[N][N];
float X[N], Y[N];
int i,j;

for (i=0; i<N/4; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```



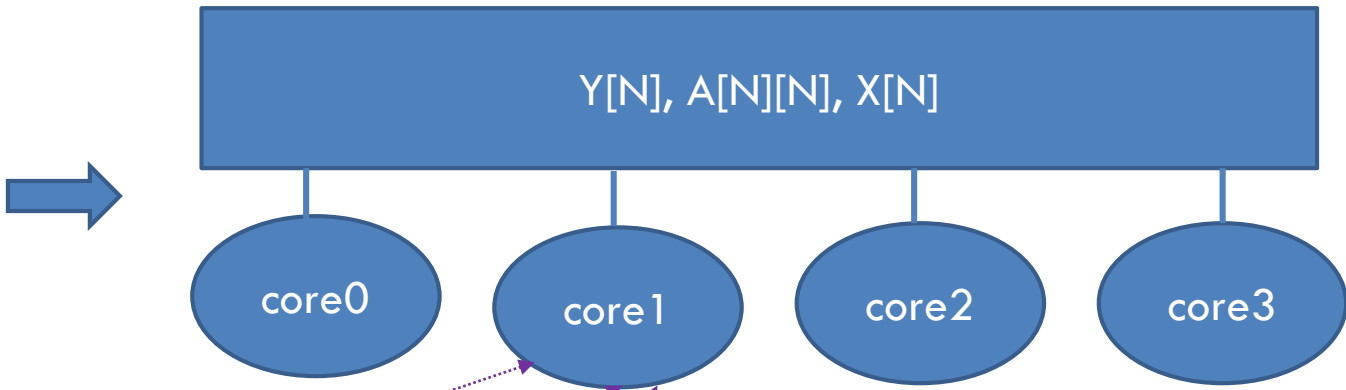


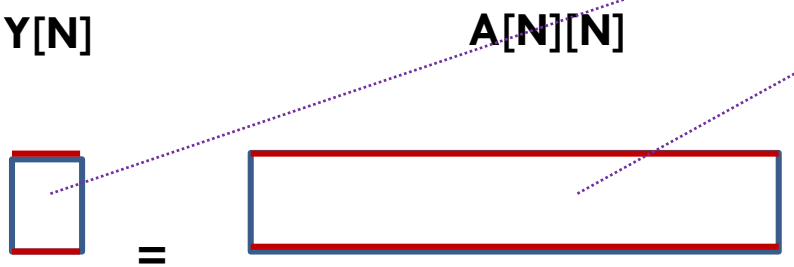
How can we parallelize this program using pthreads? MVM_pthreads.c program

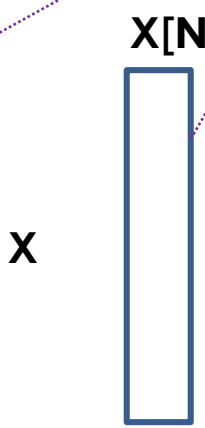
We can split i loop into four equal parts, one for each core

```
float A[N][N];
float X[N], Y[N];
int i,j;

for (i=N/4; i<N/2; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```





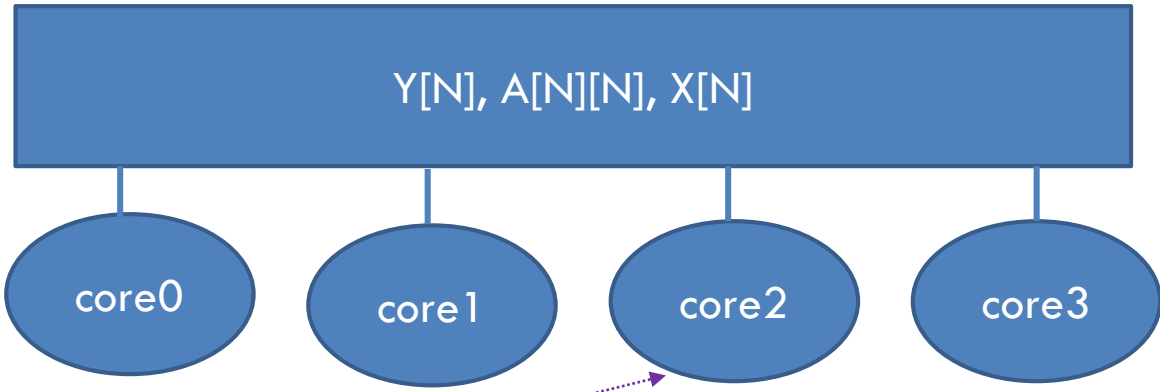


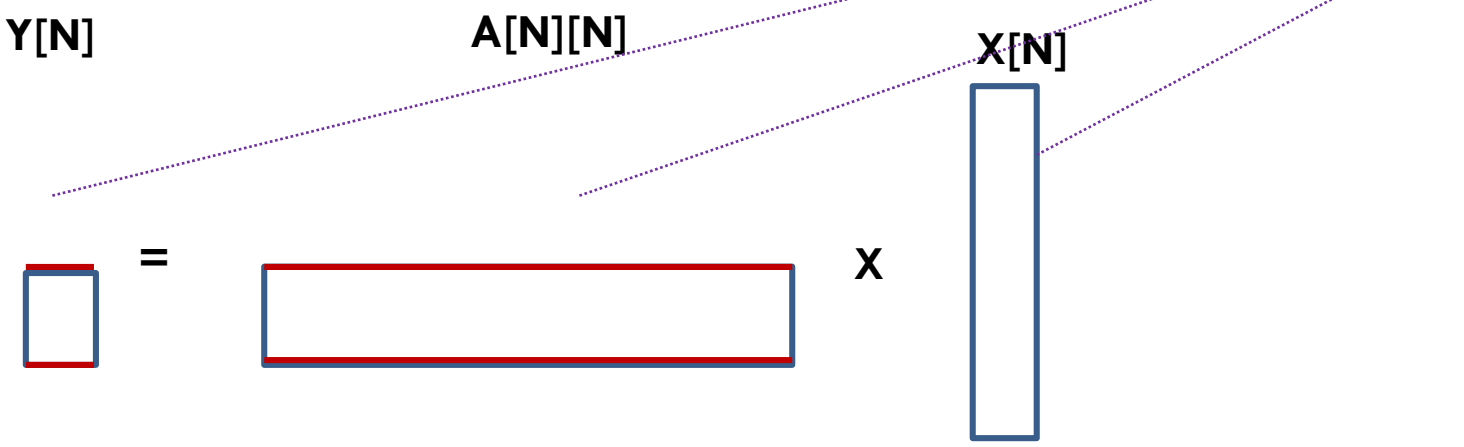
How can we parallelize this program using pthreads? MVM_pthreads.c program

We can split i loop into four equal parts, one for each core

```
float A[N][N];
float X[N], Y[N];
int i,j;

for (i=N/2; i<3*N/4; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```



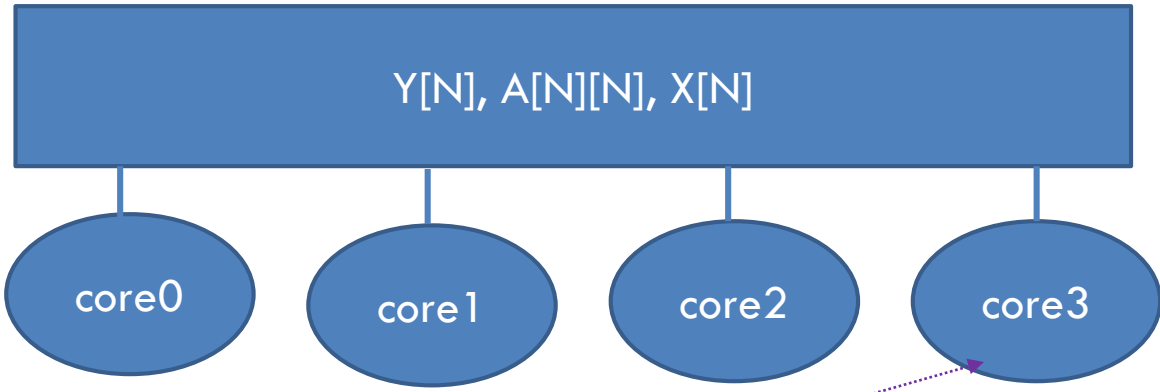


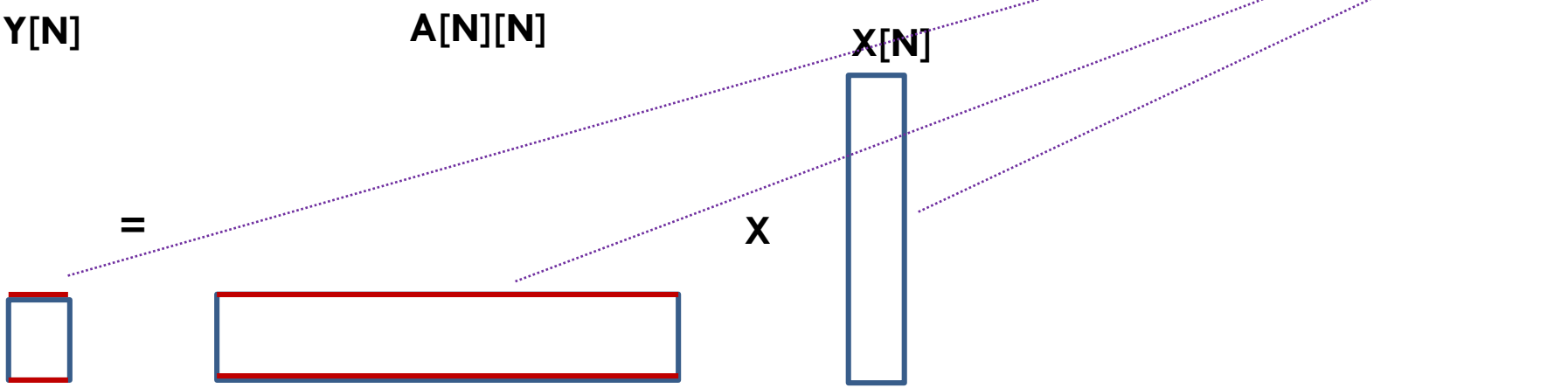
How can we parallelize this program using pthreads? MVM_pthreads.c program

We can split i loop into four equal parts, one for each core

```
float A[N][N];
float X[N], Y[N];
int i,j;

for (i=3*N/4; i<N; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```





How can we parallelize this program using pthreads? MVM_pthreads.c program


```
...
for (thread_num = 0; thread_num < NUM_THREADS; thread_num++)
    pthread_create(&thread_handles[thread_num], NULL, MVM, (void*) thread_num);
.....


void *MVM (void* rank) {

  long my_rank = (long) rank;
  int i, j;
  int local = N/NUM_THREADS;

  int starting_row = my_rank * local;
  int ending_row = starting_row + local - 1;

  for (i = starting_row; i <= ending_row; i++) {
    for (j = 0; j < N; j++)
      Y[i] += A[i][j] * X[j];     }

  return 0;
}
```

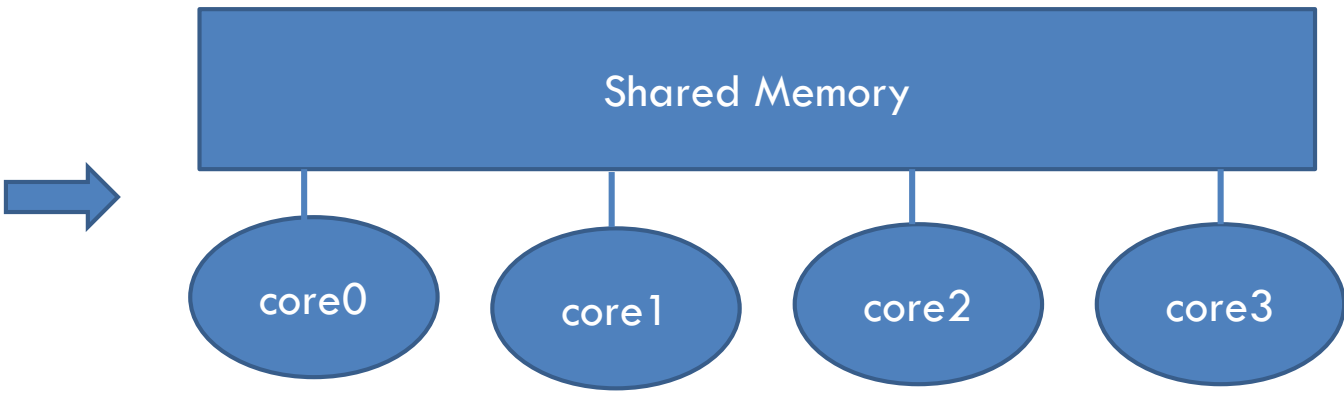
- ***Create 'NUM_THREADS' threads where they all execute MVM() function***
- *Let's assume N=1000 and NUM_THREADS=4*

Thread	Starting row	Ending row
0	0	249
1	250	499
2	500	749
3	750	999

How can we parallelize this program using pthreads? Results from the lab session


```
float A[N][N];
float X[N], Y[N];
int i,j;

for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```



Performance speedup

	N=100	N=200	N=500	N=1000	N=2000
2 threads	x1.47	x1.97	x1.98	x1.99	x1.99
4 threads	x1.16	x2.3	x3.47	x3.69	x3.7

Why our code does not scale well in these cases?

How can we parallelize this program using pthreads? Results from the lab session


```
float A[N][N];
float X[N], Y[N];
int i,j;

for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```

- When using pthreads, there is **an overhead in creating and synchronizing the threads**
- When this overhead becomes comparable to the thread's computation, the speedup is low
  - This is why the code does not scale well for small input sizes

Performance speedup

	N=100	N=200	N=500	N=1000	N=2000
2 threads	x1.47	x1.97	x1.98	x1.99	x1.99
4 threads	x1.16	x2.3	x3.47	x3.69	x3.700

Why our code does not scale well in these cases?

OpenMP as an easier and higher level solution

- Using Pthreads is not that easy
  - Low level API
- OpenMP provides a higher level API which is easier to use
- See below how our previous example looks like using openMP…
- Pthreads provide more flexibility

```
#pragma omp parallel for private (i,j)
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    Y[i] += A[i][j] * X[j];
```

Further Reading

- Chapter 3 and chapter 4 in Operating Systems, Internals and Design Principles, available at https://dinus.ac.id/repository/docs/ajar/Operating_System.pdf
- POSIX Threads Programming, available at https://computing.llnl.gov/tutorials/pthreads/
- POSIX thread (pthread) libraries available at https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html

Thank you