COMP1001

Computer Systems

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk

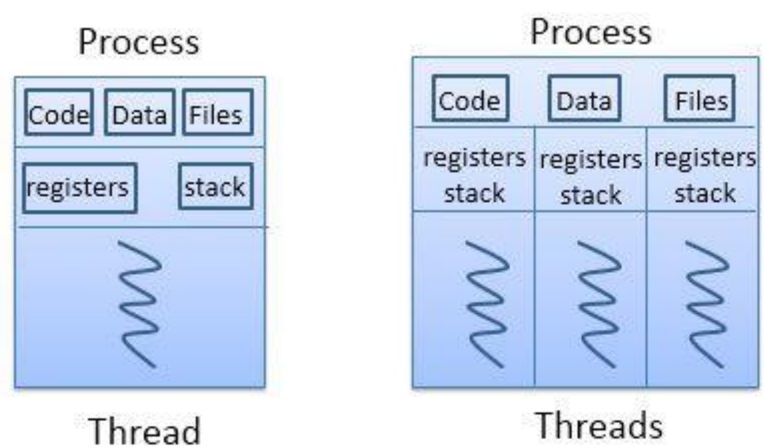
Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

Outline

- Introduction to Threads
- Two simple multi-threaded examples

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

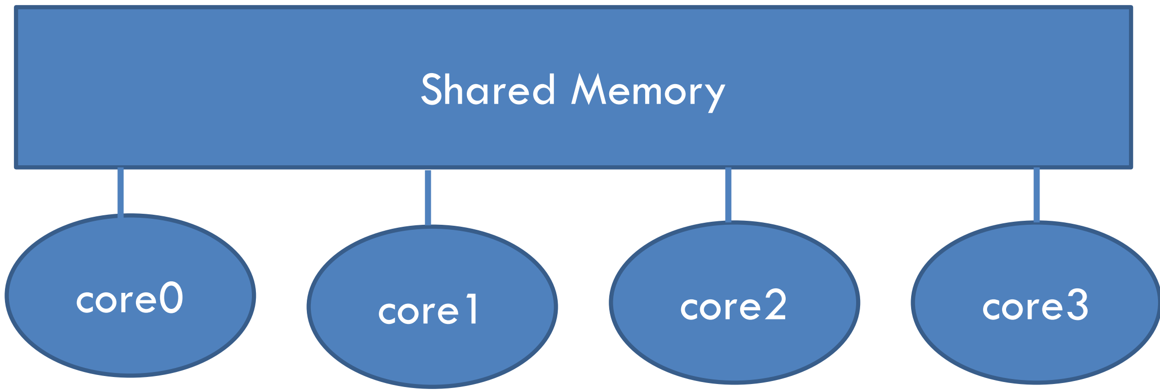
Threads – Advantages of using threads

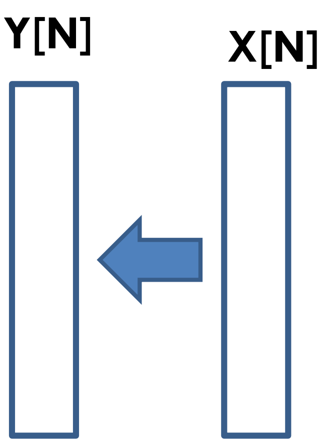
- Threads are light compared to processes
  - The OS does not have to create a new memory map (recall previous lecture) for a new thread, as it does for a process
  - The OS does not have to keep track of open files amongst different threads
  - Switching between threads of the same process has less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

How can we parallelize this program multiple threads? See sqrt.c program on github


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

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

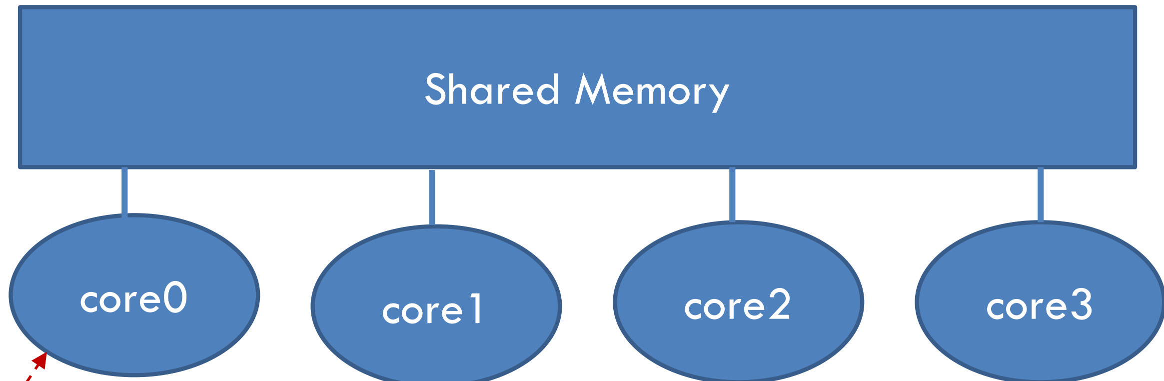




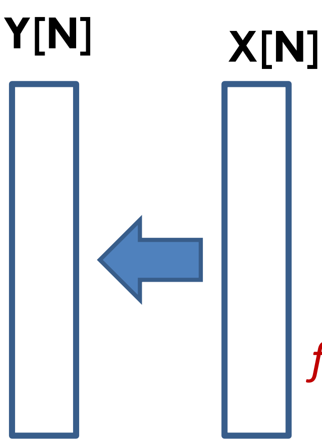
How can we parallelize this program multiple threads? See sqrt.c program on github


```
#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] ) ;
```

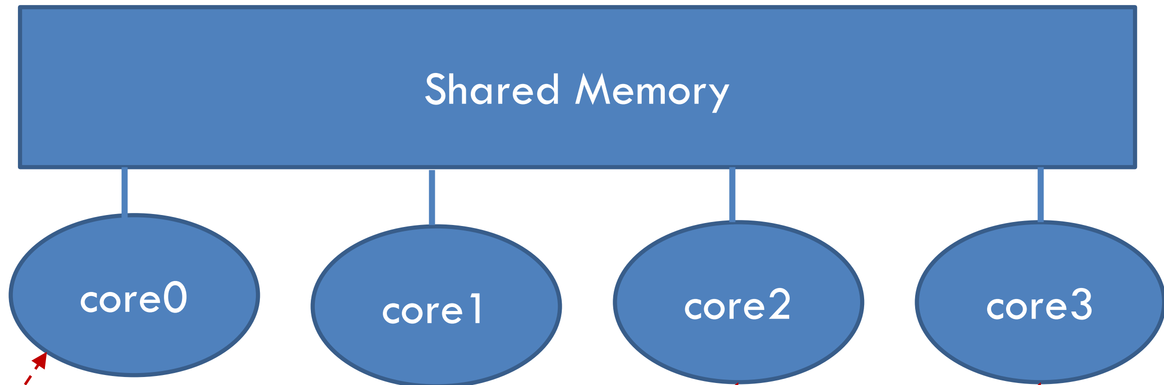
```
for (j=250; j<500; j++)
   Y[i] = sqrt( X[j] ) ;
```

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

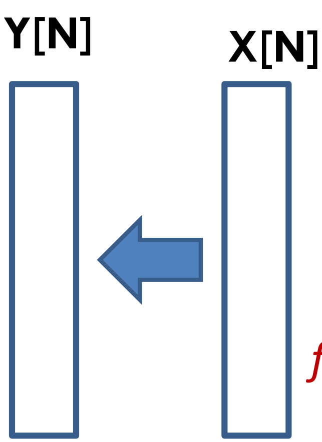
```
for (j=750; j<1000; j++)
   Y[i] = sqrt( X[j] ) ;
```

How can we parallelize this program multiple threads? See sqrt.c program on github

- 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] )
```




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

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

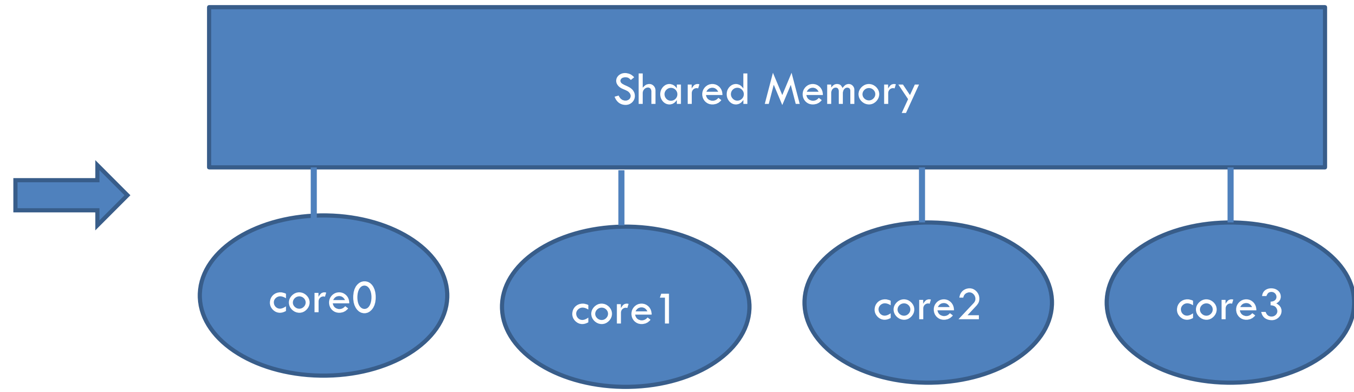
```
for (j=500; j<750; j++)
  Y[i] = sqrt( X[j] ) ;
```

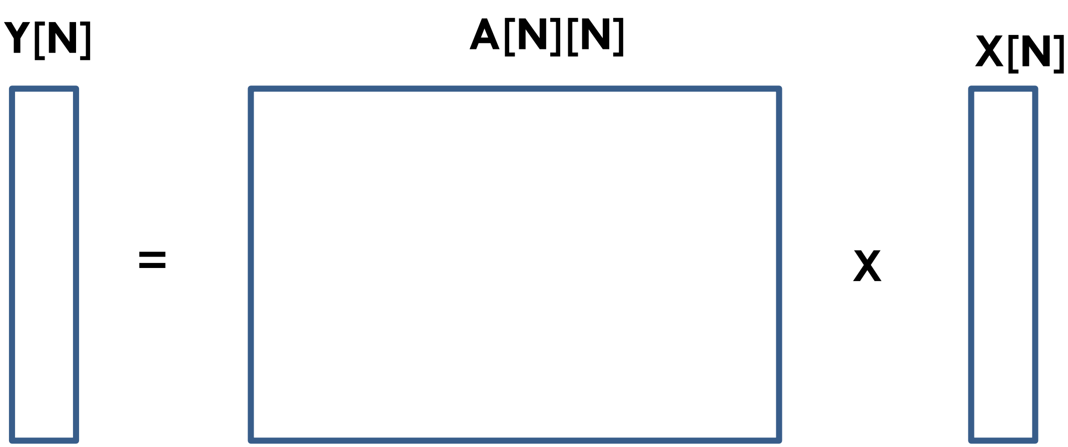
```
for (j=750; j<1000; j++)
  Y[i] = sqrt( X[j] ) ;
```

How can we parallelize this program?


```
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];
```



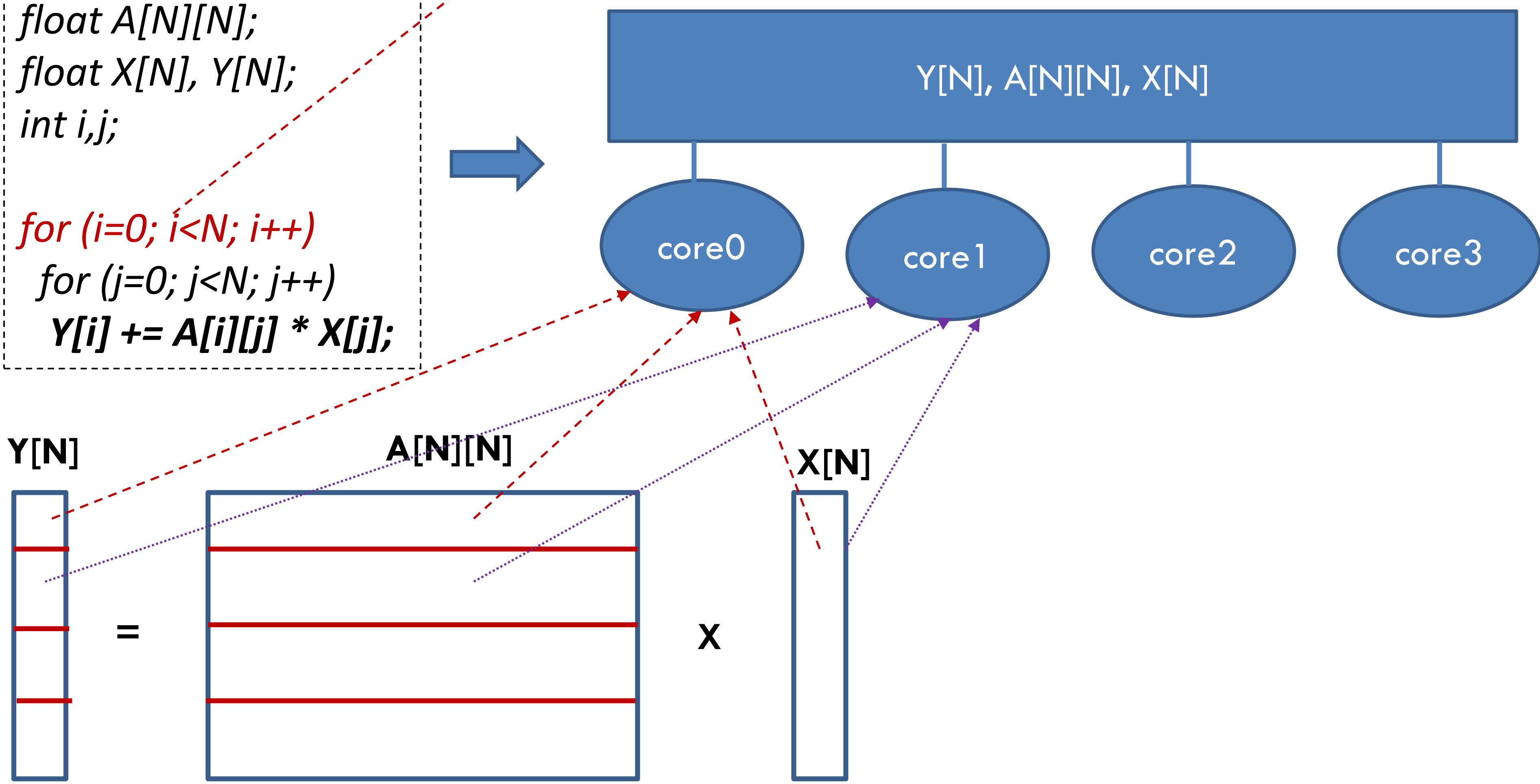


How can we parallelize this program?

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];
```

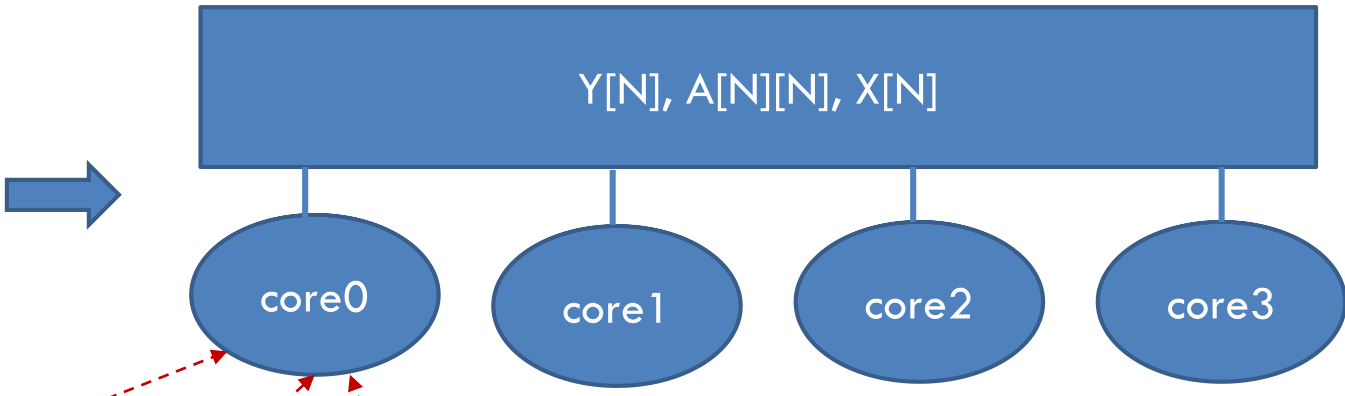


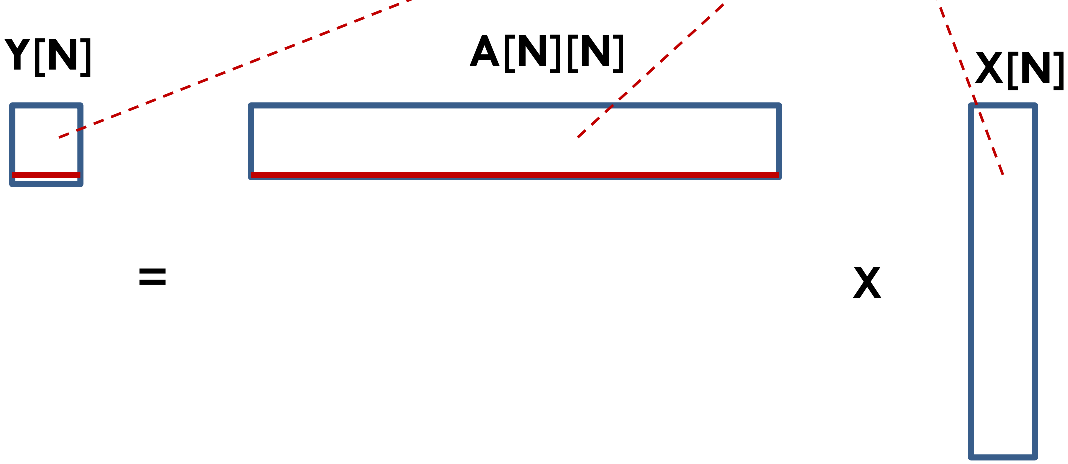
How can we parallelize this program?

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];
```



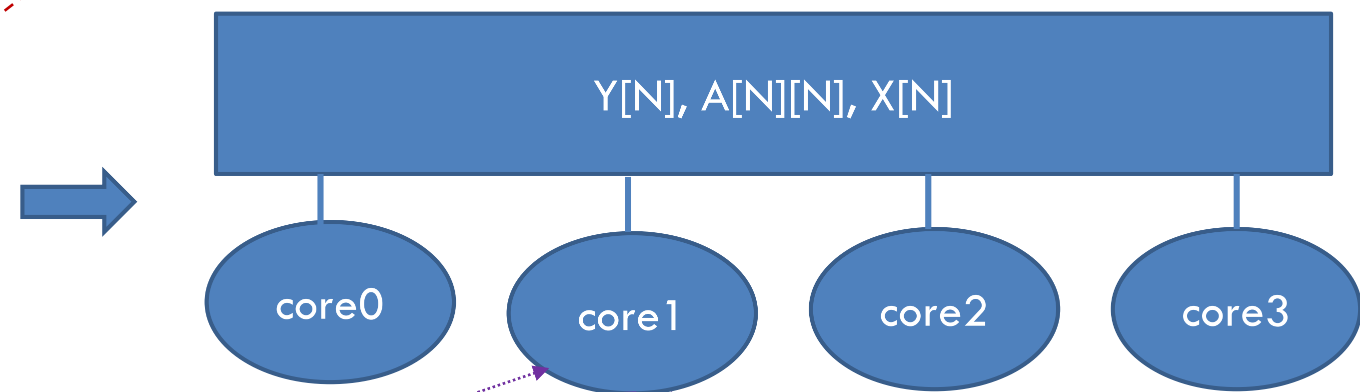


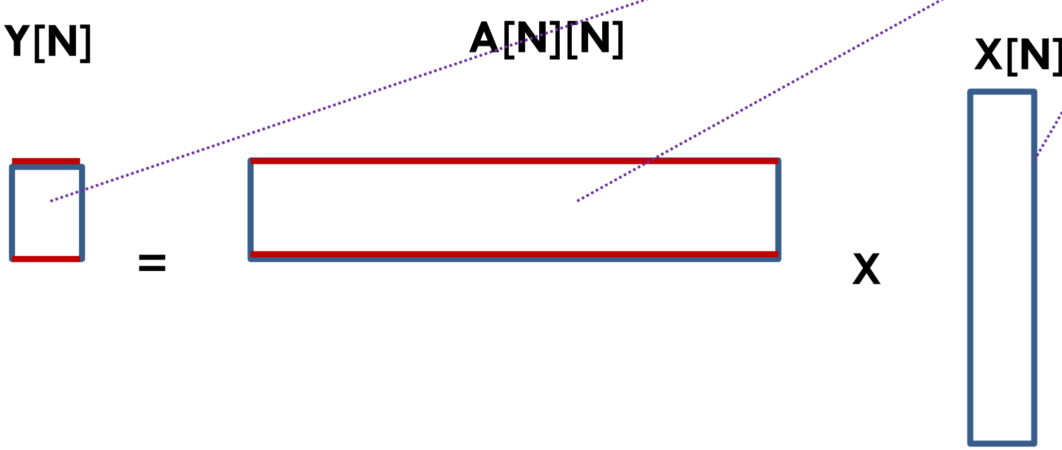
How can we parallelize this program?

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];
```



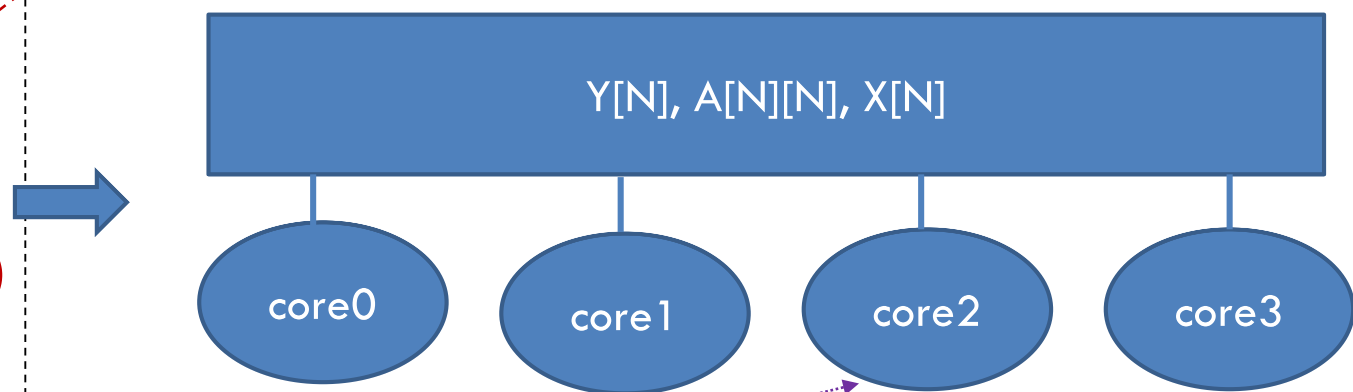


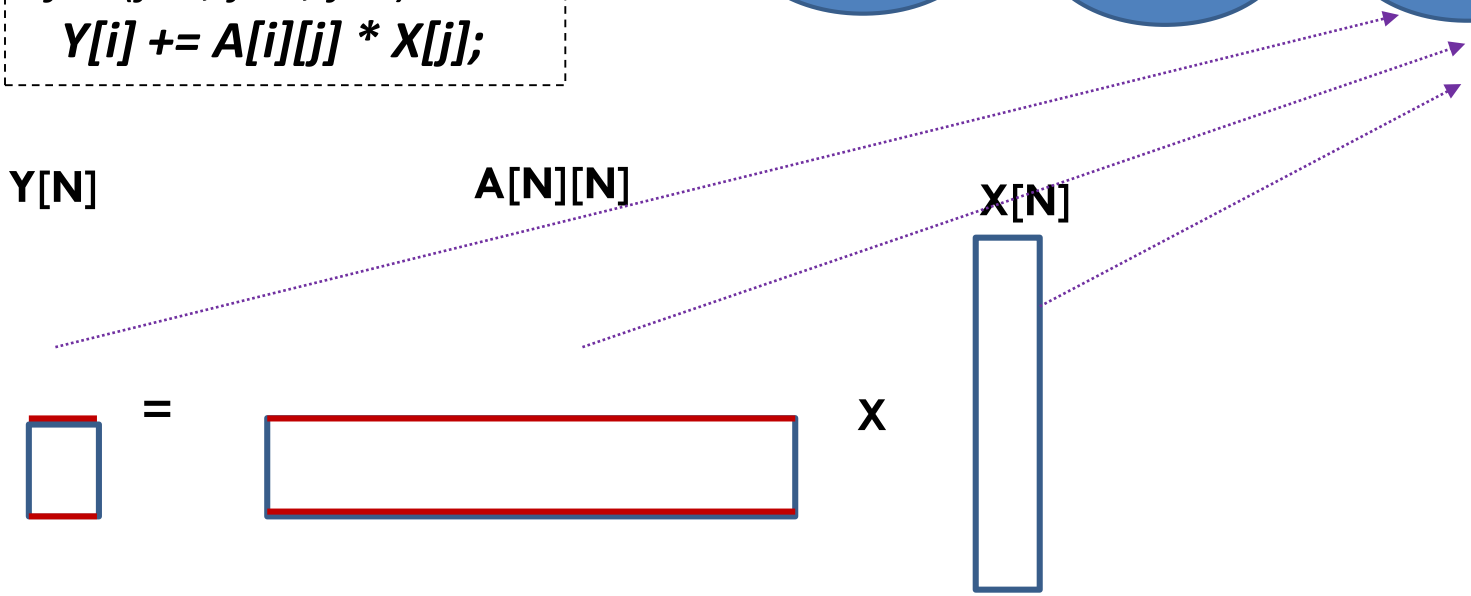
How can we parallelize this program?

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];
```



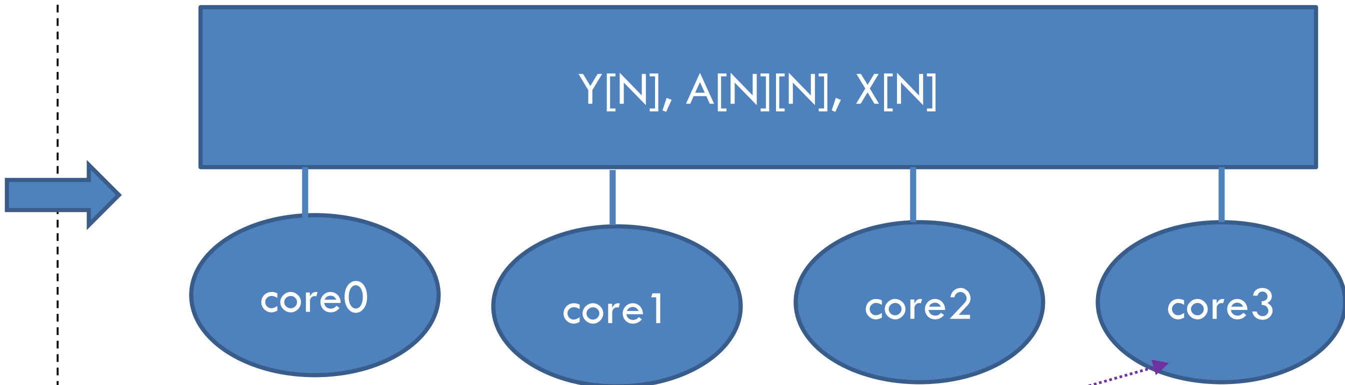


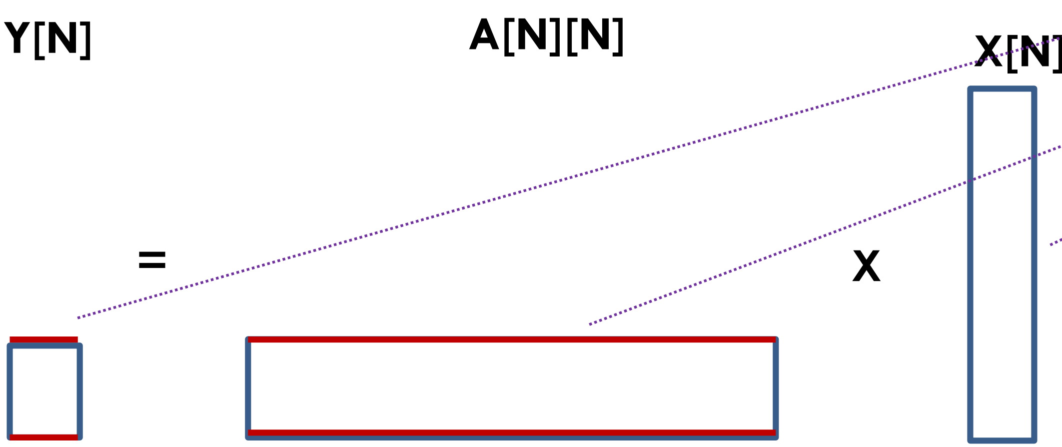
How can we parallelize this program?

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];
```

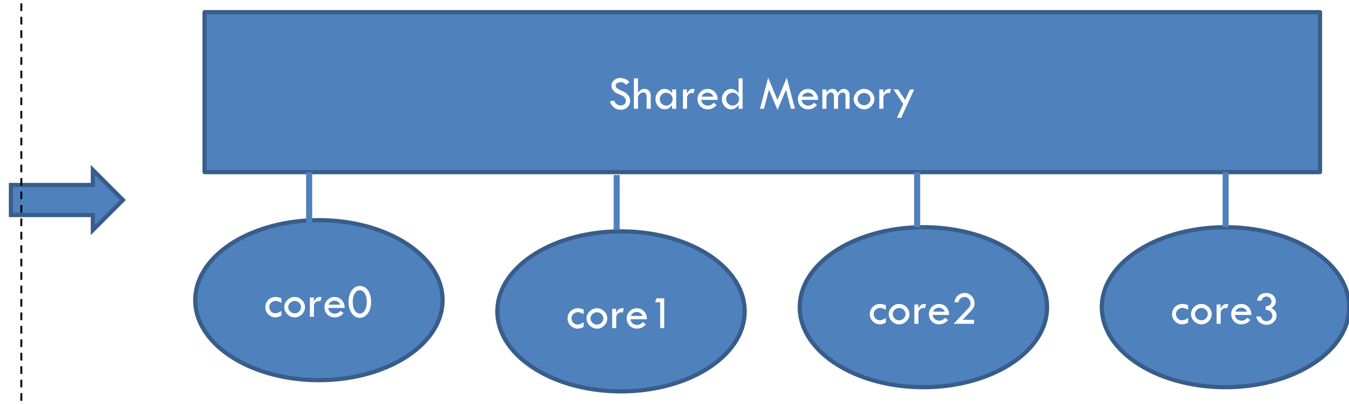




How can we parallelize this program?


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

#pragma omp parallel
for private (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?


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

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

- When using OpenMP, 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?

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