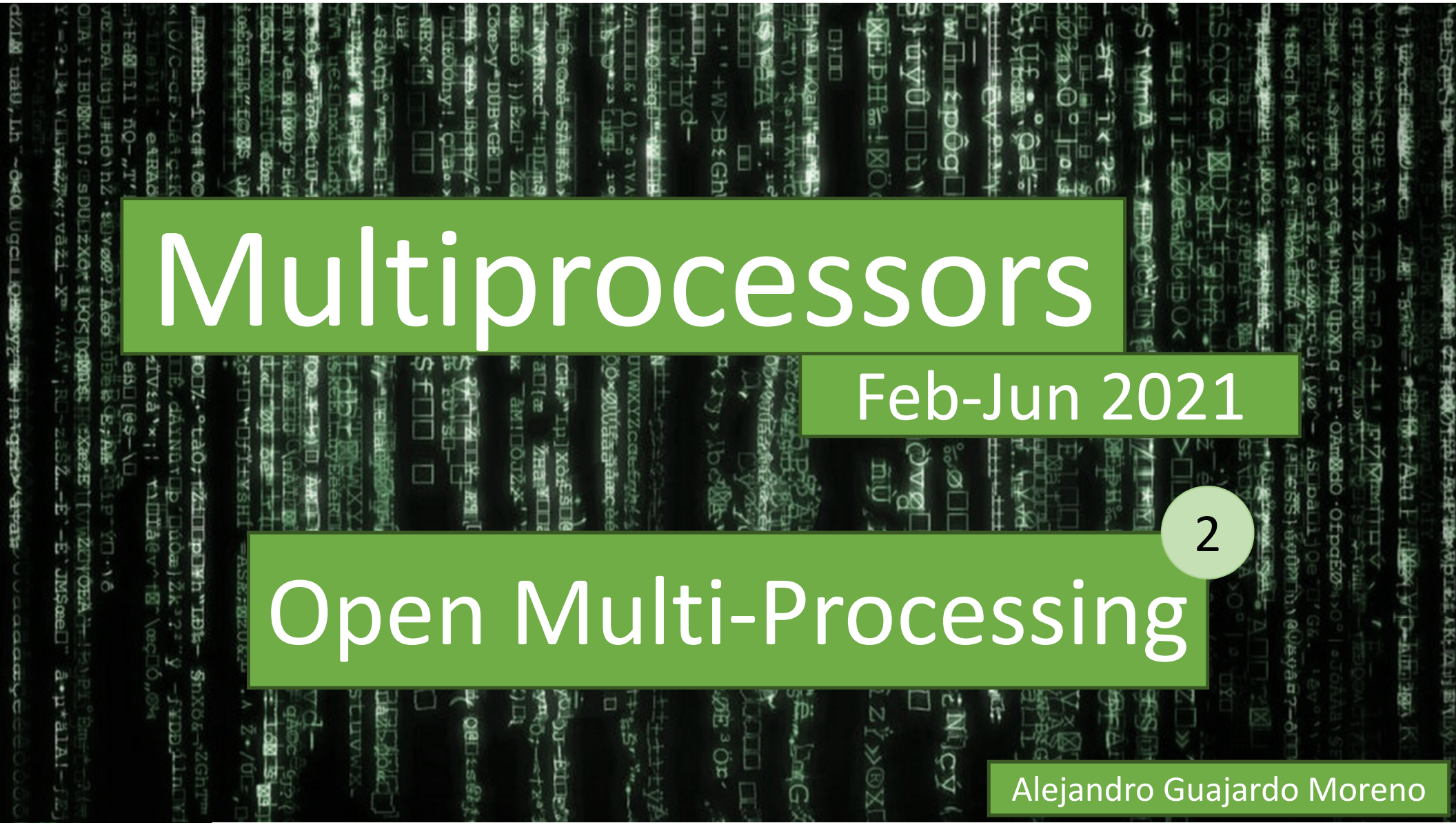


Multiprocessors

Feb-Jun 2021

2

Open Multi-Processing

Alejandro Guajardo Moreno

Recap… OMP

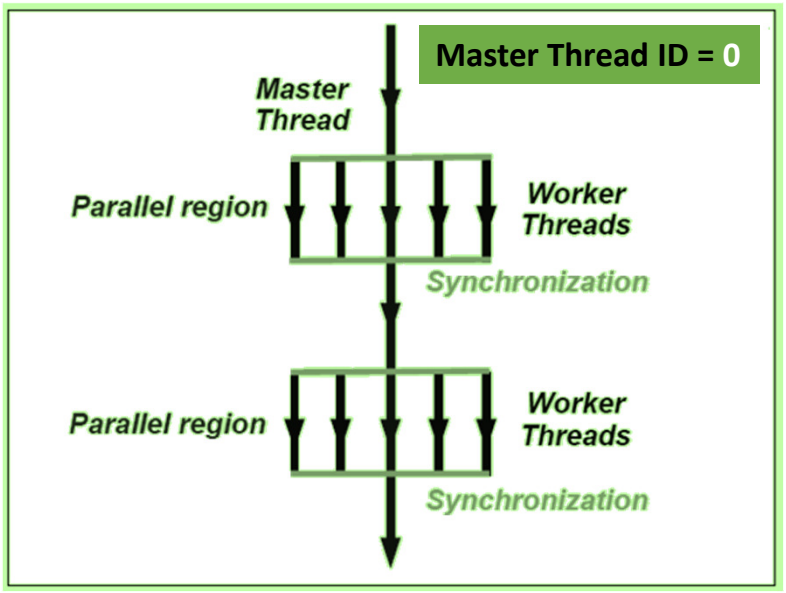
- Simplifies writing **multi-threaded** programs in C, C++ and Fortran.
- Consist of compiler directives, library functions and environmental variables.
- User Fork-Join model: one master thread and forks to multithread execution in parallel regions.
- THE Directive: `#pragma omp parallel`

Fork-join model

- The model uses one main thread and forks to multithread execution.
- The model may use several blocks of parallelization.

Fork = Parallel Region

Join = Synchronization



Note on the number of threads…

You can **ASK** for the numbers of threads **you want**…

The Kernel **WILL** give you **AT MOST** the amount of threads you requested.

Hey MCP, I am Master Thread ID = 0!

&%$#???? Hey MCP, `#pragma omp parallel num_threads(1,000,000)!`

Hello Master Thread ID = 0

Here are your 10 threads.

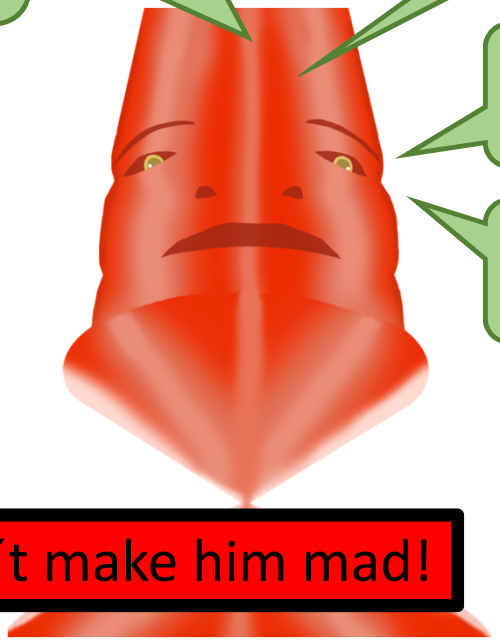
Hey MCP, `omp_set_num_threads (1,000,000);!`

I need tons of processing capacity!

Here are your 10 threads.

What the F*%$&! Please?

Have a NICE DAY!



Beware! Don´t make him mad!

A new function… omp_get_num_threads()

✓Included in <omp.h>

Function	Use
omp_set_num_threads(nthreads)	Sets the number of threads in upcoming parallel regions, unless overridden by a num_threads clause.
omp_get_thread_num()	Returns the thread number of the thread executing within its thread team.
omp_get_num_threads()	Returns the number of threads in the parallel region.

Exercise on the number of threads - Individual

1.- For how long will the kernel be pleasing our requests? Find how many threads you can ask the kernel without distubring him and making him mad.

What you will need?

```
Environment Variable
set OMP_NUM_THREADS = #

Functions, include <omp.h>
omp_set_num_threads(nthreads) - Sets the number of threads in upcoming parallel regions
omp_get_thread_num() - Returns the thread number of the thread executing within its thread team
omp_get_num_threads() - Returns the number of threads in the parallel region.


Directive
#pragma omp parallel num_threads(#)
```

2.- Answer:

❖ **For how long was the kernel nice to you?**

Note on the number of threads…

```
#include <stdio.h>
#include <omp.h>
int main() {
   int rT = 0;   //Requested Threads
   int aT = 0;   //Actual Threads

   do {
      rT+= 1;
      omp_set_num_threads(rT);
      #pragma omp parallel
      {
         if (omp_get_thread_num() == 0) {
         aT = omp_get_num_threads();
         printf("Master Control Program gave me %d threads. He's a nice program.\n", aT);
         }
         int a = 50 + 100; //Just some work...
      }
   } while (rT == aT);

   printf("\nI take it back! Master Control Program gave me %d thread. He's EVIL!\n", aT);
   return 0;
}
```

Note on the number of threads…

```
mp@multiprocesadores: ~/multiprocs
File  Edit  View  Search  Terminal  Help
Master Control Program gave me 4837 threads. He's a nice program.
Master Control Program gave me 4838 threads. He's a nice program.
Master Control Program gave me 4839 threads. He's a nice program.
Master Control Program gave me 4840 threads. He's a nice program.
Master Control Program gave me 4841 threads. He's a nice program.
Master Control Program gave me 4842 threads. He's a nice program.
Master Control Program gave me 4843 threads. He's a nice program.
Master Control Program gave me 4844 threads. He's a nice program.
Master Control Program gave me 4845 threads. He's a nice program.
Master Control Program gave me 4846 threads. He's a nice program.
Master Control Program gave me 4847 threads. He's a nice program.
Master Control Program gave me 4848 threads. He's a nice program.
Master Control Program gave me 4849 threads. He's a nice program.
Master Control Program gave me 4850 threads. He's a nice program.
Master Control Program gave me 4851 threads. He's a nice program.
Master Control Program gave me 4852 threads. He's a nice program.
Master Control Program gave me 4853 threads. He's a nice program.
Master Control Program gave me 4854 threads. He's a nice program.
Master Control Program gave me 4855 threads. He's a nice program.
Master Control Program gave me 4856 threads. He's a nice program.
Master Control Program gave me 4857 threads. He's a nice program.

libgomp: Thread creation failed: Resource temporarily unavailable
mp@multiprocesadores:~/multiprocs$
```

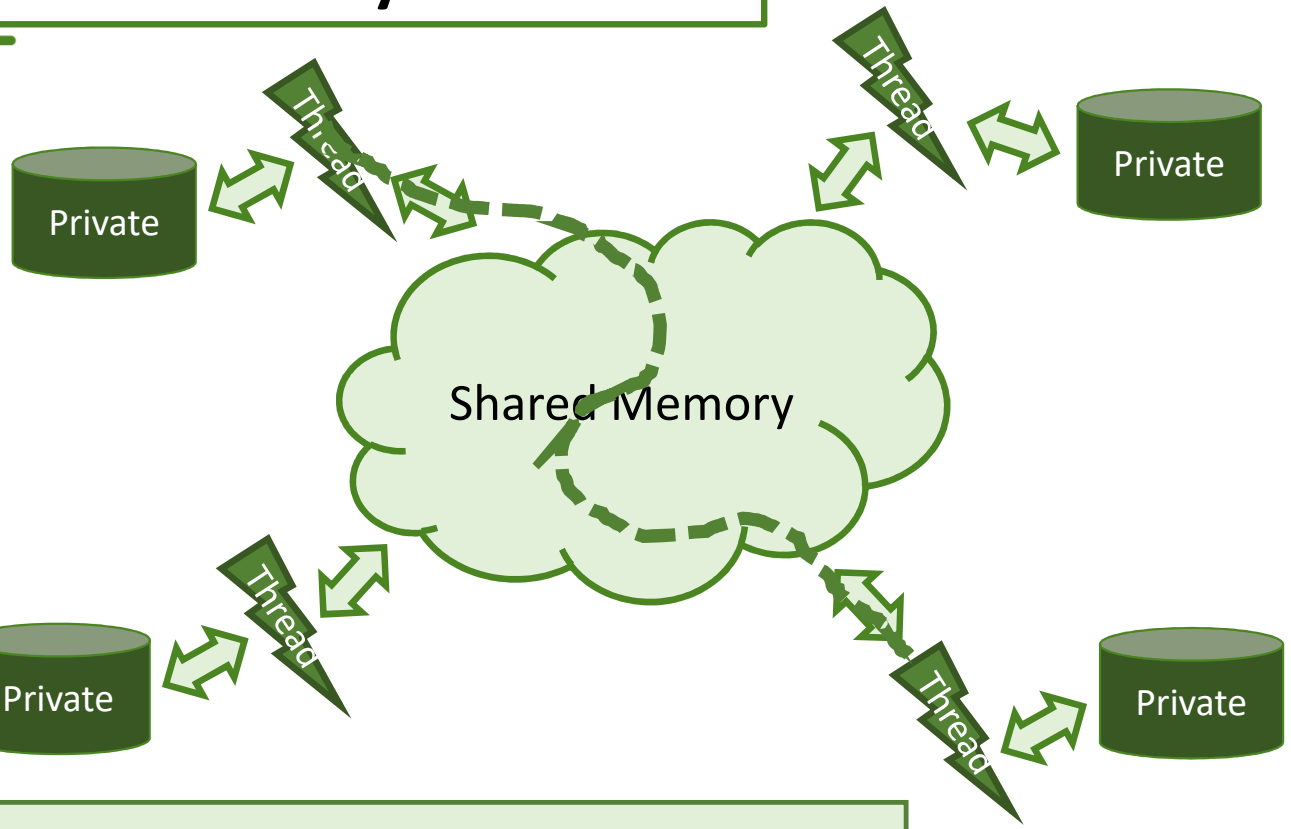
```
Command Prompt
Master Control Program gave me 8168 threads. He's a nice program.
Master Control Program gave me 8169 threads. He's a nice program.
Master Control Program gave me 8170 threads. He's a nice program.
Master Control Program gave me 8171 threads. He's a nice program.
Master Control Program gave me 8172 threads. He's a nice program.
Master Control Program gave me 8173 threads. He's a nice program.
Master Control Program gave me 8174 threads. He's a nice program.
Master Control Program gave me 8175 threads. He's a nice program.
Master Control Program gave me 8176 threads. He's a nice program.
Master Control Program gave me 8177 threads. He's a nice program.
Master Control Program gave me 8178 threads. He's a nice program.
Master Control Program gave me 8179 threads. He's a nice program.
Master Control Program gave me 8180 threads. He's a nice program.
Master Control Program gave me 8181 threads. He's a nice program.
Master Control Program gave me 8182 threads. He's a nice program.
Master Control Program gave me 8183 threads. He's a nice program.
Master Control Program gave me 8184 threads. He's a nice program.
Master Control Program gave me 8185 threads. He's a nice program.
Master Control Program gave me 8186 threads. He's a nice program.
Master Control Program gave me 8187 threads. He's a nice program.
Master Control Program gave me 8188 threads. He's a nice program.
Master Control Program gave me 8189 threads. He's a nice program.
Master Control Program gave me 8190 threads. He's a nice program.
Master Control Program gave me 8191 threads. He's a nice program.
Master Control Program gave me 8192 threads. He's a nice program.
Master Control Program gave me 8192 threads. He's a nice program.

I take it back! Master Control Program gave me 8192 thread. He's EVIL!

C:\Users\Alex\source\repos\OpenMPT\x64\Debug>
```

OpenMP Memory Model

Threads play by these rules:

- All threads have access to the process´s memory.
- The process's memory is globally shared.
- Data in the shared area is accesible by all threads.
- Threads can have their own private memory.
- No thread can access other thread's memory.
- Data transfer is transparent to the programmer.
- Synchronization is almost always implicit.

OpenMP Memory Model



2 types of memory

Heap – For all the program (OMP = Shared)

Stack – For just the thread (OMP = Private)

OpenMP Memory Model – SHARED Memory

In SHARED memory:

- There is only ONE instance of the data.
- All changes to memory are visible to all threads.
- Read and writes to memory are not protected; each thread can read and write when they please.

OpenMP Memory Model – PRIVATE Memory

In PRIVATE memory:

- Each thread has **a copy** of the original data.
- Only the owner thread can access this data.
- Changes are only visible to the owner thread.

Shared and Private Data

```
#include <stdio.h>

int main() {

   type global_variable = value;

   #pragma omp parallel
   { //Parallel region begins

      type private_variable = value;

      printf("Hello OMP World!...\n");
   } //Parallel region ends

   printf("This will print after sync...
And only once!...\n");

}
```

Shared

Private

Exercise

1.- Test how the data behaves if modified before and inside a parallel region.

2.- Run the code as is.

3.- Add a very short Sleep(), 1 millisecond should be enough. What happens?

```
#include <omp.h>
#include <stdio.h>
#include <windows.h>

int main() {
   int sharedVar = 6;
   #pragma omp parallel num_threads(10)
   {
      int privateVar = omp_get_thread_num(); //This data is mine!
      sharedVar = omp_get_thread_num(); //I´ll leave my mark!
      //Add here a pause.
      printf("Thread (%d), privateVar = %d, sharedVar= %d \n",
omp_get_thread_num(), privateVar, sharedVar);
   }
}
```

Important concept: Race Condition

- Caused by unintended sharing of data.
- Identified when the program's outcome changes with each run (threads are scheduled differently).
- Synchronization provides a way to control race conditions, by protecting data conflicts.
- Synchronization is very expensive. Use intelligent data access algorithms to minimize the need for synchronization.

For NEXT class… Calculating π using OMP (Teams of 2)

Create a single program that does the following 3 things:

1.- Using the original NON-Multithreaded code, run it 5 times, record and average them to get a baseline. Add the results to a table.

2.- Use Windows/Posix multithreaded code. Use the amount of threads equal to the number of virtual processors you have in your computer. Run it 5 times and also get the average. Record the results to the table.

3.- Use OpenMP to parallelize your code. Again, use the same amount of threads you used in #2. Run it 5 times and get the average. Record results in the table.

In a document, answer these:

a.- Which code ran faster?

b.- Was it easier of harder to use OMP over Windows Threads?

c.- Include your table with all your runs and averages.

Include your code. Visual Studio, a ZIP file with the full Solution. If Linux, a ZIP file with all the code files; in each code file you must include the compile line used as a comment at the beginning of the file.

Careful! Watch out for RACE CONDITIONS!

For NEXT class…

```
#include <stdio.h>
#include <time.h>

   long cantidadIntervalos = 1000000000;
   double baseIntervalo;
   double fdx;
   double acum = 0;
   clock_t start, end;

   void main() {
      double x;
      long i;
      baseIntervalo = 1.0 / cantidadIntervalos;
      start = clock();
      for (i = 0, x = 0.0; i < cantidadIntervalos; i++) {
         fdx = 4 / (1 + x * x);
         acum = acum + (fdx * baseIntervalo);
         x = x + baseIntervalo;
      }
   end = clock();
   printf("Resultado = %20.18lf (%ld)\n", acum, end - start);
}
```

You only need to use:

Directive
#pragma omp parallel

Functions
omp_set_num_threads
omp_get_num_threads
omp_get_thread_num

See you soon…