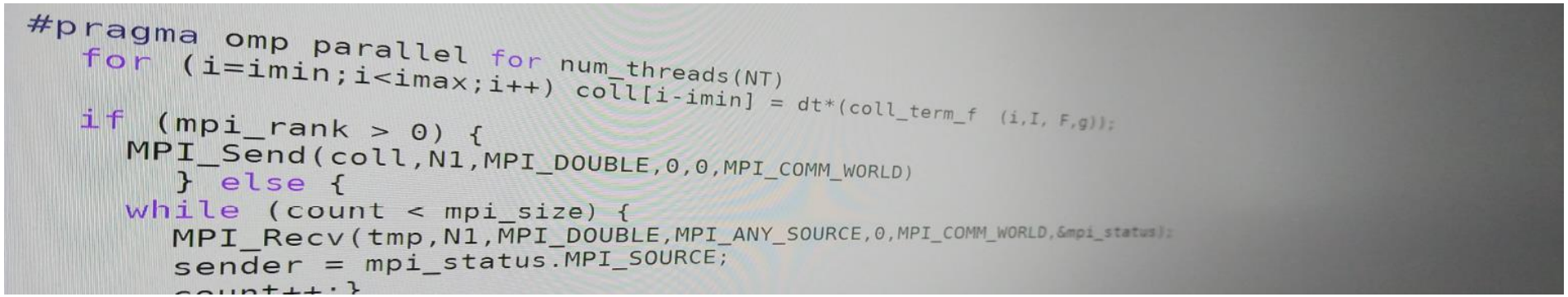


Introduction to parallel programming (for physicists)

FRANÇOIS GÉLIS & GRÉGOIRE MISGUICH, IPhT courses, June 2019.

IPhT, CEA-Saclay
Itzykson room
courses.ipht.fr

1. Introduction & hardware aspects (FG)
2. A few words about Maple & Mathematica
3. Linear algebra libraries
4. Fast Fourier transform
5. Python Multiprocessing
6. OpenMP
7. MPI (FG)
8. MPI+OpenMP (FG)

These slides (GM)

Increasing coding effort

Parallelism with Maplesoft

1. Parallel programming

   [From Maple's documentation] « Maple provides tools for two different types of parallel programming. The **Task Programming Model** enables parallelism by executing multiple tasks within a single process. The second type of parallelism comes from **the Grid package**, which enables parallelism by starting multiple processes."

   Remark: the task model is somewhat analogous Threads/OpenMP, and Grid is analogous to MPI. Both OpenMP and MPI will be presented in these lectures

2. **Automatic parallelization**

Automatic Parallelism with Maplesoft

Example of automatic parallelization: simple numerical sum

- Sequential version

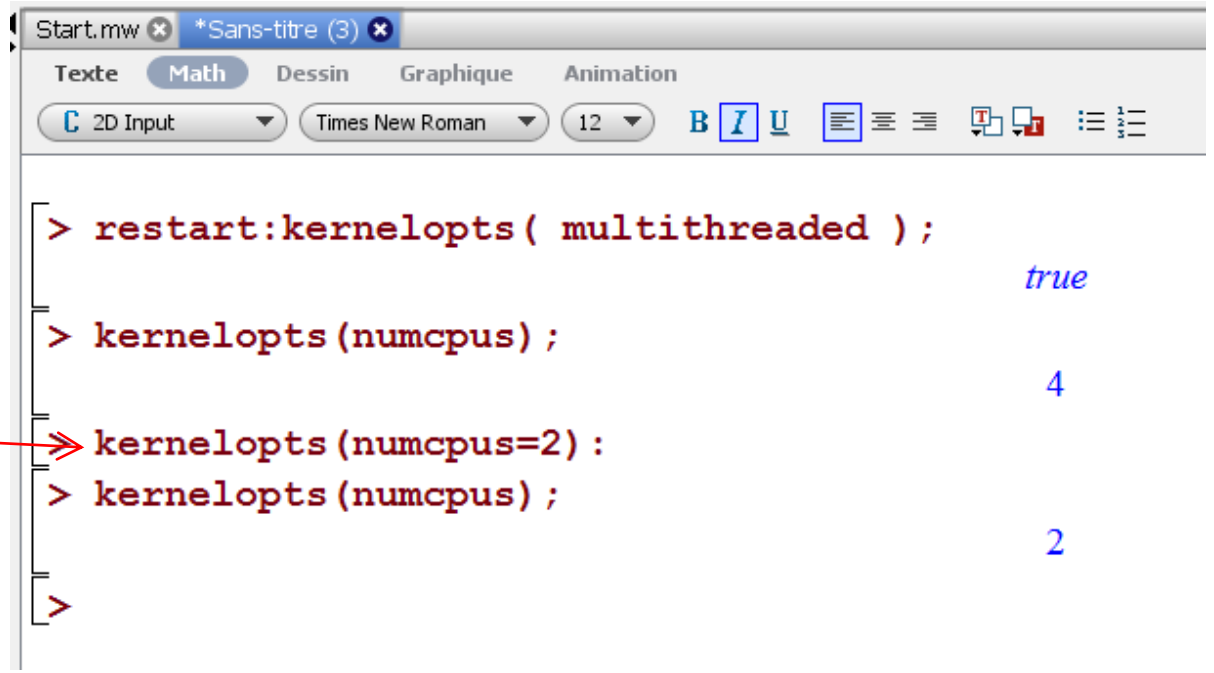
```
add(evalf(sin(i)),i=1..100000);
```

- Parallel version:

```
with(Threads);
Add(evalf(sin(i)),i=1..100000);
```

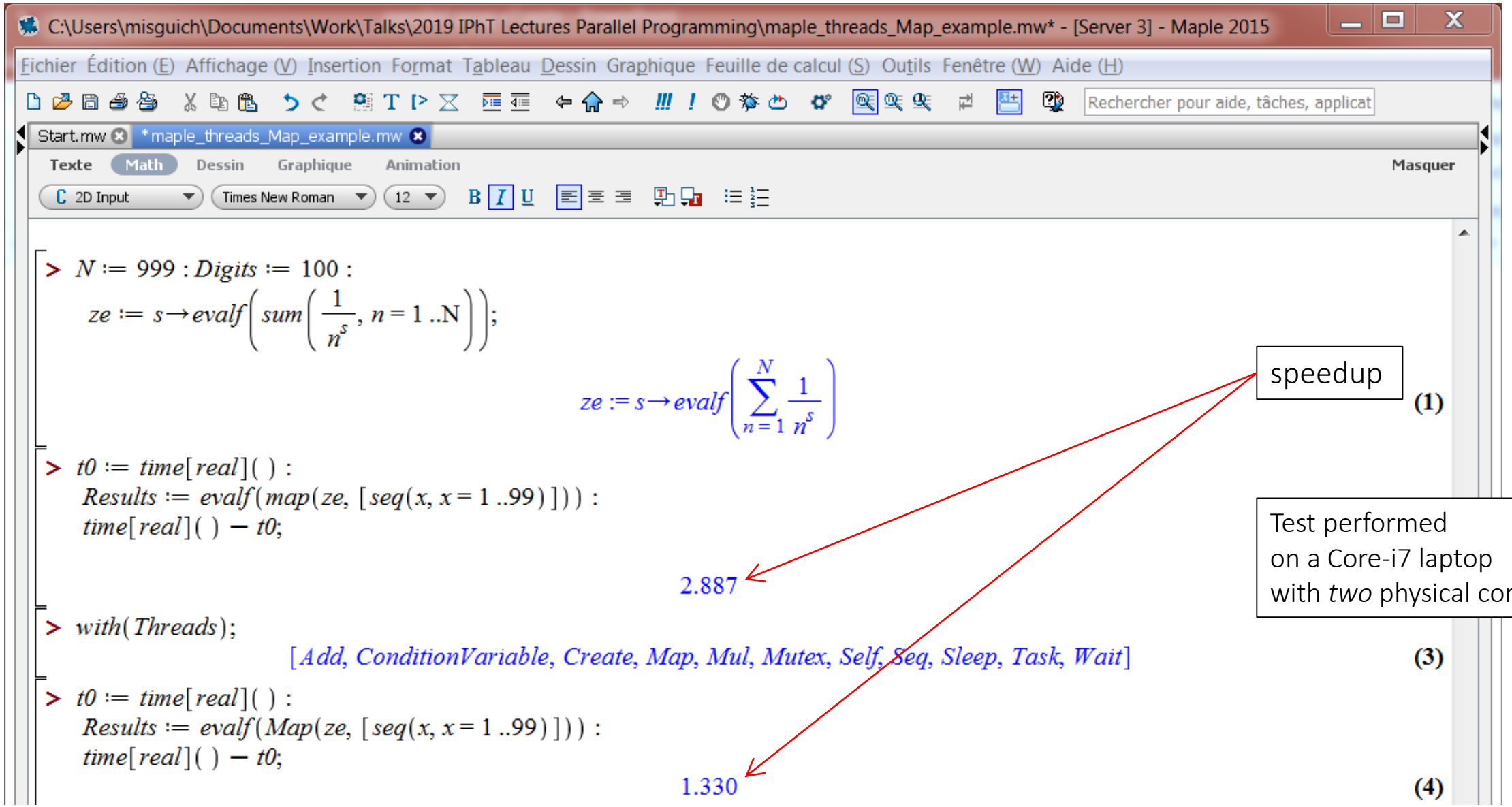
- by default Maple will create as many threads as available CPU cores
- Similar functions: **Mul**, **Seq**, and **Map**.

To select the number of CPU to be used by the multi-theaded engine:



Automatic Parallelism with Maplesoft

Example with `Map`



Automatic parallelism with Maplesoft

Example with `Add`

```
> t0 := time[real]( ) :
  add(evalf(sin(i)), i = 1 ..100000);
  time[real]( ) - t0;
```

1.847777103630379156630554301110741745559264562965641397408536842921550126002349383388906799091582245

15.803 (5)

```
> t0 := time[real]( ) :
  Add(evalf(sin(i)), i = 1 ..100000);
  time[real]( ) - t0;
```

1.847777103630379156630554301110741745559264562965641397408536842921550126002349383388906799091582245

8.406 (6)

```
>
```

speedup

Parallelism with Mathematica

Many possibilities:

Parallel Computing

The Wolfram Language provides a uniquely integrated and automated environment for parallel computing. With zero configuration, full interactivity, and seamless local and network operation, the symbolic character of the Wolfram Language allows immediate support of a variety of existing and new parallel programming paradigms and data-sharing models.

Automatic Parallelization

Parallelize — evaluate an expression using automatic parallelization
ParallelTry — try different computations in parallel, giving the first result obtained

Computation Setup & Broadcasting »

ParallelEvaluate — evaluate an expression on all parallel subkernels
DistributeDefinitions — distribute definitions to all parallel subkernels
ParallelNeeds — load the same package into all parallel subkernels

Data Parallelism »

ParallelMap ▪ **ParallelTable** ▪ **ParallelSum** ▪ **...**
ParallelCombine — evaluate expressions in parallel and combine their results

Concurrency Control »

ParallelSubmit — submit expressions to be evaluated concurrently
WaitAll — wait for all concurrent evaluations to finish
WaitNext — wait for the next of a list of concurrent evaluations to finish

Shared Memory & Synchronization »

SetSharedVariable — specify symbols with values to synchronize across subkernels
SetSharedFunction — specify functions whose evaluations are to be synchronized
$SharedVariables ▪ **$SharedFunctions** ▪ **UnsetShared** ▪ **CriticalSection**

Setup and Configuration »

LaunchKernels — launch a specified number of subkernels
$KernelCount — number of running subkernels
$KernelID ▪ **Kernels** ▪ **AbortKernels** ▪ **CloseKernels** ▪ **...**
$ProcessorCount — number of processor cores on the current computer

Multi-Processor and Multicore Computation

Compile — create compiled functions that run in parallel
Parallelization — execute compiled functions in parallel
CompilationTarget — create machine-level parallel compiled functions

GPU Computing »

CUDAFunctionLoad — load a function to run on a GPU using CUDA
OpenCLFunctionLoad — load a function to run on a GPU using OpenCL

File-Based Parallelism

FileSystemScan ▪ **FileSystemMap**

(automatic) Parallelism with Mathematica - **Parallelize**

```
Mathematica 10.4.1 for Linux x86 (64-bit)
Copyright 1988-2016 Wolfram Research, Inc.

In[1]:=  $KernelCount
Out[1]= 0

In[2]:=  AbsoluteTiming[Table[Length[FactorInteger[10^50 + n]], {n, 20}]]
Out[2]= {2.59524, {7, 4, 6, 4, 6, 4, 3, 7, 3, 6, 6, 7, 2, 5, 3, 3, 8, 3, 7, 6}}

In[3]:=  $KernelCount
Out[3]= 0

In[4]:= AbsoluteTiming[Parallelize[Table[Length[FactorInteger[10^50 + n]], {n,
20}]]]
Launching kernels...
Out[4]= {3.23171, {7, 4, 6, 4, 6, 4, 3, 7, 3, 6, 6, 7, 2, 5, 3, 3, 8, 3, 7, 6}}

In[5]:=  $KernelCount
Out[5]= 2

In[6]:=  AbsoluteTiming[Parallelize[Table[Length[FactorInteger[10^50 + n]], {n,
20}]]]
Out[6]= {1.55382, {7, 4, 6, 4, 6, 4, 3, 7, 3, 6, 6, 7, 2, 5, 3, 3, 8, 3, 7, 6}}
```

gives the number of subkernels available for parallel computations

Test done on a 2-(physical) core processor

speedup

(automatic) Parallelism with Mathematica - **ParallelMap**

```
Mathematica 10.3.0 for Linux x86 (64-bit)
Copyright 1988-2015 Wolfram Research, Inc.

In[1]:=  LaunchKernels[4]
Out[1]= {KernelObject[1, local], KernelObject[2, local], KernelObject[3, local],
KernelObject[4, local]}

In[2]:= Map[(Pause[1]; f[#]) &, {a, b, c, d}] // AbsoluteTiming
Out[2]= {4.00321, {f[a], f[b], f[c], f[d]}}

In[3]:= ParallelMap[(Pause[1]; f[#]) &, {a, b, c, d}] // AbsoluteTiming
Out[3]= {1.02335, {f[a], f[b], f[c], f[d]}}

In[4]:= CloseKernels[]

Out[4]= {KernelObject[1, local, <defunct>], KernelObject[2, local, <defunct>],
KernelObject[3, local, <defunct>], KernelObject[4, local, <defunct>]}
In[6]:= $KernelCount

Out[6]= 0

In[7]:=  LaunchKernels[2]
Out[7]= {KernelObject[5, local], KernelObject[6, local]}

In[8]:= ParallelMap[(Pause[1]; f[#]) &, {a, b, c, d}] // AbsoluteTiming
Out[8]= {2.00858, {f[a], f[b], f[c], f[d]}}
```

speedup

(automatic) Parallelism with Mathematica

Licence restrictions…

At IPhT (Mathematica network licence):

```
Mathematica 10.4.1 for Linux x86 (64-bit)
Copyright 1988-2016 Wolfram Research, Inc.

In[1]:=  $MaxLicenseProcesses

Out[1]= 10

In[2]:= $MaxLicenseSubprocesses

Out[2]= 80
```

8 subkernels per Process