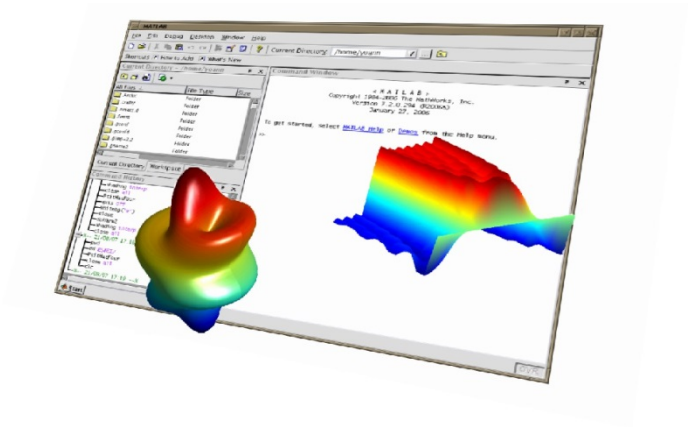
SEAMLESS MULICORE PARALLELISM IN MATLAB

Claude TADONKI and **Pierre-Louis CARUANA**
Mines ParisTech – CRI (Fontainebleau/France)
University of Paris-Sud (Orsay/France)



Parallel and Distributed Computing and Networks (*PDCN 2014)*

February 17 – 19, 2014

Innsbruck (AUSTRIA)

Motivations

- MATLAB is widely used for several kinds of application (*scientific computing*, *image processing*, …)
- MATLAB provides a programming language suitable for ordinary scientist (not programmers!)
- MATLAB is commonly used for heavy computations (*simulations, image&signal processing*)
- Multicore architecture is now a standard, with an increasing number of cores.
- MATLAB offers a built-in solution for parallel computing through additional packages.

Providing a flexible way to consider parallelism in MATLAB is really useful to easilly take advantage of this possibility.

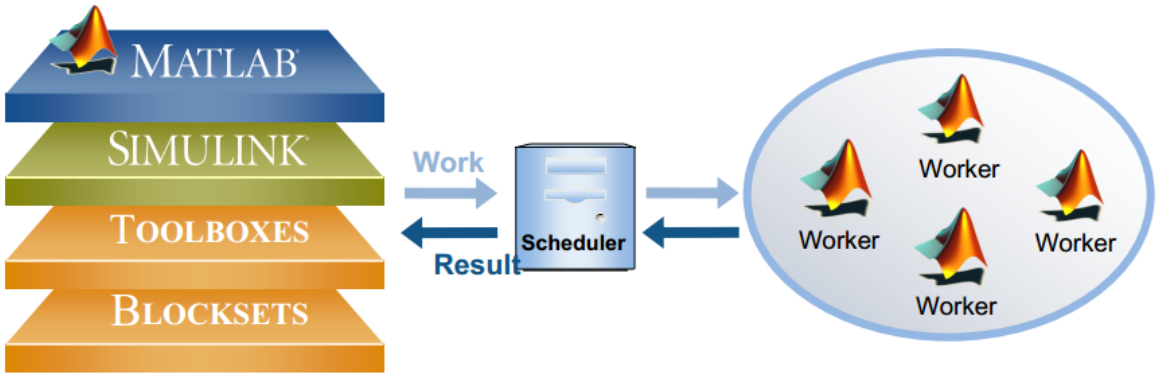
MATLAB Solutions for Parallel Computing (Tasks Feature)

```
% Create one job object
  j = createJob();
% Create tasks for job j
  createTask(j, @sum, 1, {U(1:4)});
  createTask(j, @sum, 1, {U(5:8)});
% Submit job j to the scheduler
  submit(j);
% Wait for job completion
  wait(j);
% Get the outputs of job j
  v = fetchOutputs(j);
% Aggregate the partial sums
  s = v{1} + v{2};
% Delete job j
  delete(j);
```

We now state some important facts:

- each task within a job is assigned to a unique MATLAB worker, and is executed independently of the other tasks
- the maximum number of workers is specified in the local scheduler profile, and can be modified as desired, up to a limit of twelve
- if a job has more tasks than allowed workers, the scheduler waits for one of the active tasks to complete before starting another MATLAB worker for the next task. In some cases, such an overloading will prevent the entire job from being executed.

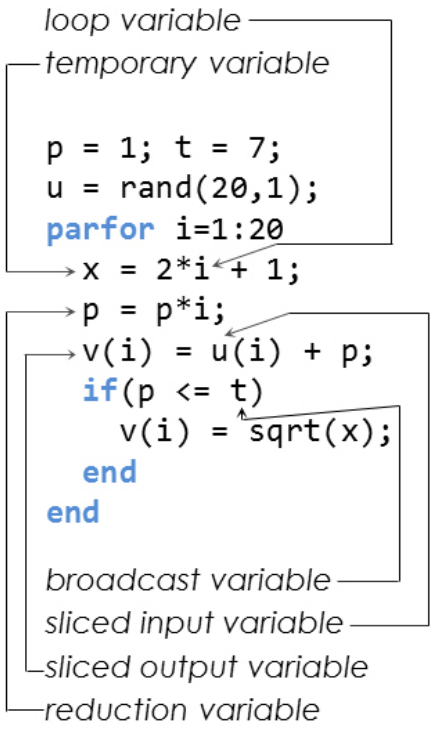
- **jobs/tasks**
  - Series of independent tasks; not necessarily iterations
  - Workflow : Always scheduled



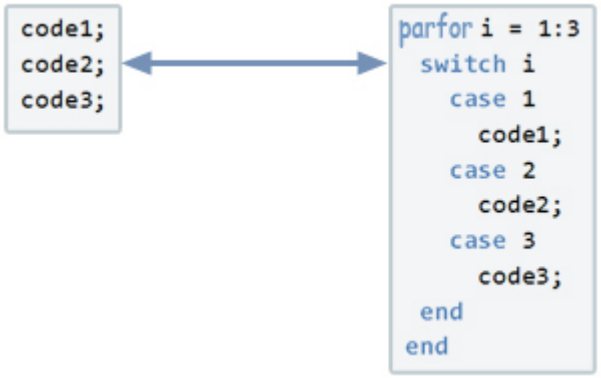
MATLAB Solutions for Parallel Computing (Parfor)

- **parfor**
  - Multiple independent iterations
  - Easy to combine serial and parallel code

★ The **parfor** construct is used, in place of a standard for statement, to specify that the Corresponding loop should be executed in parallel.

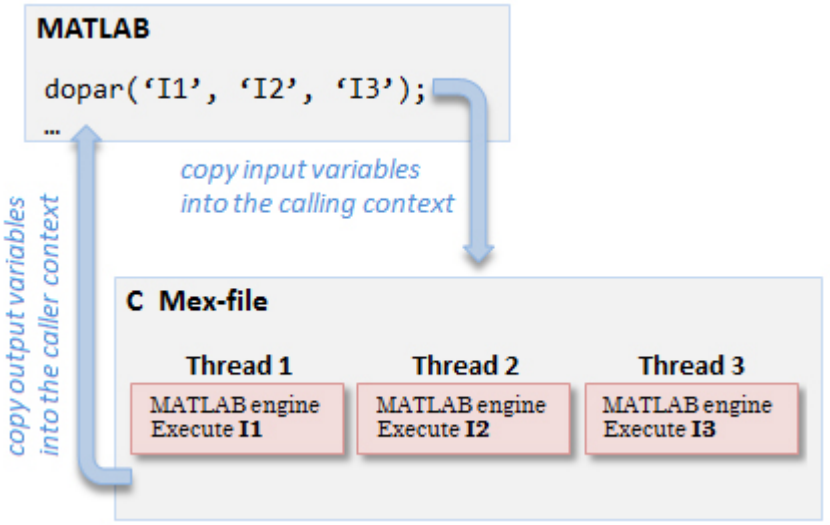


*Variable kinds within a **parfor** loop*



*Mapping a flow of instructions to a **parfor***

Our solution (MATLAN engine & Pthread)



1. *Parse the associated string* of each input instruction in order to get the list of all involved variables.

2. *Load the data of each right-hand-side variable* from the caller context into the current one.

3. *Launch as many POSIX threads as input instructions*, each thread executes its associated MATLAB instruction using a call to the MATLAB engine [11].

4. *Copy back the data* corresponding to each output variable into the context of the caller.

- ★ We run a C mex-file which creates and launch the threads and manages all data transfers.
- ★ Each thread call a MATLAB engine which excutes the associated MATLAB instruction .
- ★ On WINDOWS, the /Automation mode allow to avoid opening a new MATLAB each time we call the engine.

Function	Purpose
engOpen	Start up MATLAB engine
engClose	Shut down MATLAB engine
engGetVariable	Get a MATLAB array from the engine
engPutVariable	Send a MATLAB array to the engine
engEvalString	Execute a MATLAB command
engOutputBuffer	Create a buffer to store MATLAB text output
engOpenSingleUse	Start a MATLAB engine, nonshared use
engGetVisible	Determine visibility of MATLAB engine session
engSetVisible	Show or hide MATLAB engine session

run	pthread(s)	task(s)	parfor(s)
1	6.8228	5.9950	9.4820
2	4.9977	5.9581	9.4874
3	5.9762	5.9286	9.0390
4	4.9950	5.9685	8.9879
5	4.9103	5.9410	9.0397

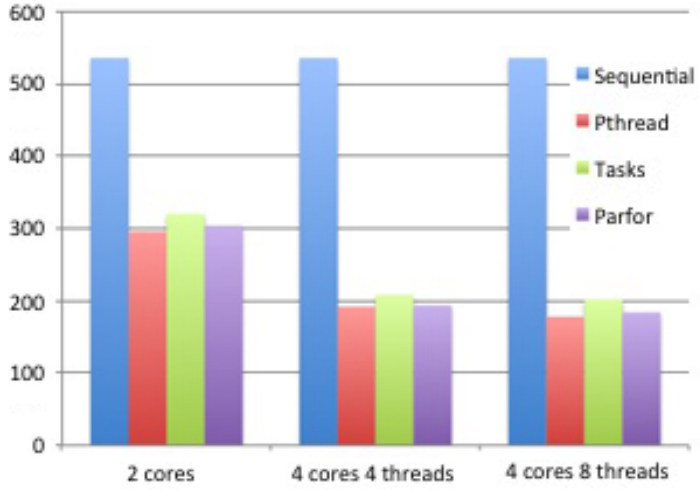
Table 1. Pure overhead of our mechanism

l_{vector}	pthread(s)	task(s)	parfor(s)
10^6	0.144	0.687	0.122
2×10^6	0.640	1.407	0.898
3×10^6	0.946	2.114	1.607
4×10^6	1.332	3.777	2.205
5×10^6	1.713	6.604	2.413

Table 2. Time costs for data import&export

- The time overheads provided do not depend on how heavy is the associated task.
- Pthreads based solution has the lowest overhead and is more stable.
- The cost for data import&export suggests that we better use them intensively

Matrix-Product

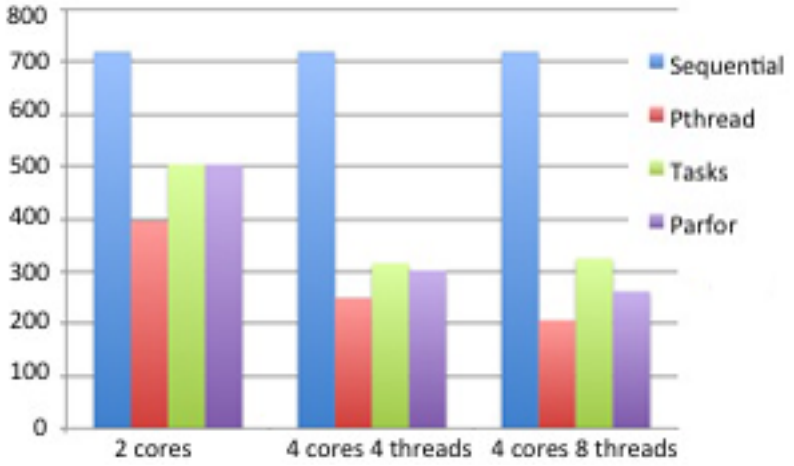


		Pthread	Tasks	Parfor
2 threads	400*400	0,12	0,16	0,12
	800*800	0,74	0,82	0,69
	1200*1200	1,34	1,22	1,25
	1600*1600	1,64	1,52	1,6
	2000*2000	1,81	1,68	1,76

		Pthread	Tasks	Parfor
4 threads	400*400	0,11	0,3	0,19
	800*800	0,89	1,27	1,06
	1200*1200	1,67	1,95	1,8
	1600*1600	2,42	2,42	2,56
	2000*2000	2,79	2,58	2,76

		Pthread	Tasks	Parfor
8 threads	400*400	0,1	0,3	0,36
	800*800	0,99	1,27	1,48
	1200*1200	2,13	1,91	2,18
	1600*1600	3,01	2,42	2,84
	2000*2000	3,01	2,65	2,91

Sorting

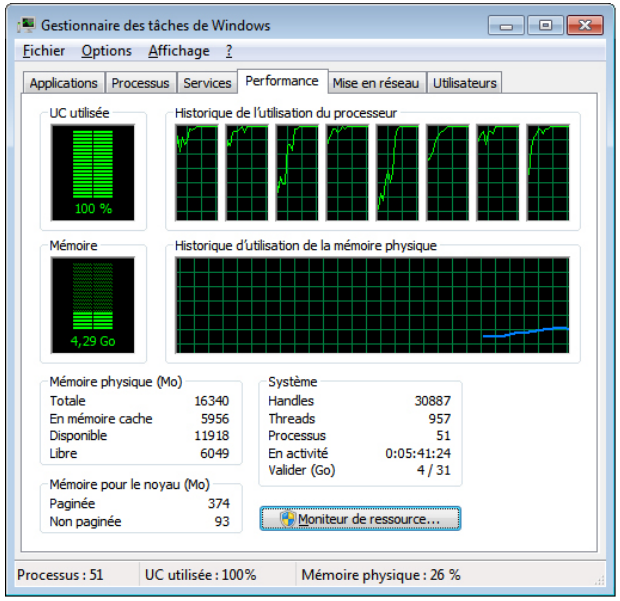


		Pthread	Tasks	Parfor
2 threads	1000000	1,42	1,21	1,13
	2000000	1,63	1,32	1,32
	3000000	1,65	1,33	1,32
	4000000	1,61	1,28	1,28
	5000000	1,82	1,43	1,43

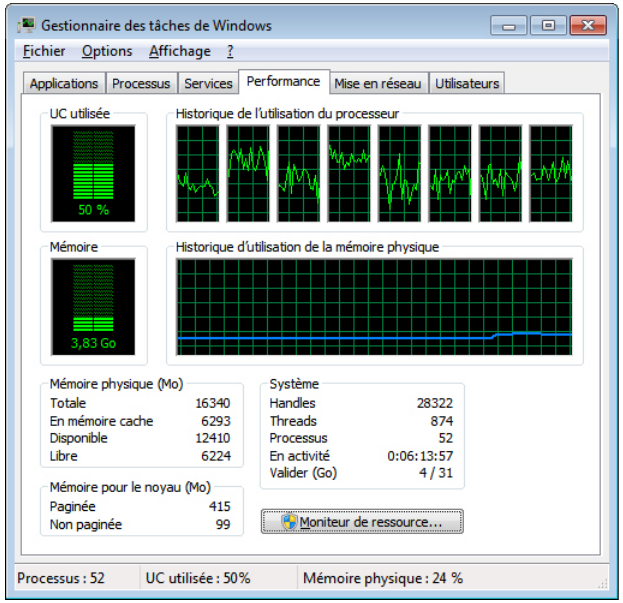
		Pthread	Tasks	Parfor
4 threads	1000000	1,94	1,94	1,79
	2000000	2,37	2,22	2,09
	3000000	2,57	2,14	2,06
	4000000	2,55	2,03	2,14
	5000000	2,88	2,28	2,38

		Pthread	Tasks	Parfor
8 threads	1000000	2,62	1,94	2,27
	2000000	3,11	2,14	2,48
	3000000	3,25	2,17	2,52
	4000000	3,17	2,13	2,41
	5000000	3,48	2,22	2,75

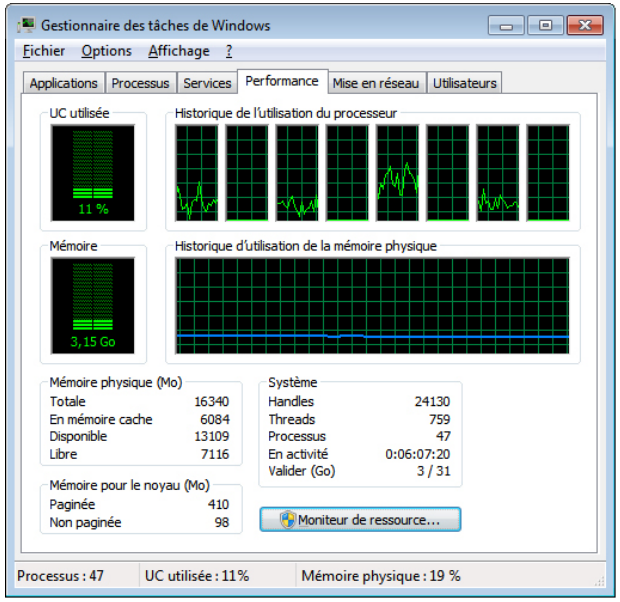
Introduction



*CPU-cores load with **Pthread***



*CPU-cores load with **Tasks***



*CPU-cores load with **Parfor***

THANKS FOR YOUR ATTENTION!
QUESTIONS ?