Uni.lu HPC School 2019

PS07: Scientific computing using MATLAB

Uni.lu High Performance Computing (HPC) Team

V. Plugaru

University of Luxembourg (UL), Luxembourg

http://hpc.uni.lu

Latest versions available on Github:

UL HPC tutorials: https://github.com/ULHPC/tutorials

UL HPC School: http://hpc.uni.lu/hpc-school/

PS07 tutorial sources:

https://ulhpc-tutorials.rtfd.io.rtfd.io/en/latest/maths/matlab/basics/

Summary

Session Objectives

Better understand the usage of MATLAB on the Uni.lu HPC Platform

- running in interactive mode
  - ↪ with either the full graphical or the text-mode interface
  - ↪ graphical web portal based on OnDemand coming soon

Session Objectives

Better understand the usage of MATLAB on the Uni.lu HPC Platform

- running in interactive mode
  - ↪ with either the full graphical or the text-mode interface
  - ↪ graphical web portal based on OnDemand coming soon

- running in passive mode
  - ↪ several ways of submitting MATLAB jobs
  - ↪ example launchers for SLURM

Session Objectives

Better understand the usage of MATLAB on the Uni.lu HPC Platform

- running in interactive mode
  - ↪ with either the full graphical or the text-mode interface
  - ↪ graphical web portal based on OnDemand coming soon

- running in passive mode
  - ↪ several ways of submitting MATLAB jobs
  - ↪ example launchers for SLURM

- checking available toolboxes & licenses status

Session Objectives

Better understand the usage of MATLAB on the Uni.lu HPC Platform

- running in interactive mode
  - ↪ with either the full graphical or the text-mode interface
  - ↪ graphical web portal based on OnDemand coming soon

- running in passive mode
  - ↪ several ways of submitting MATLAB jobs
  - ↪ example launchers for SLURM

- checking available toolboxes & licenses status

- using script (.m) files

Session Objectives

Better understand the usage of MATLAB on the Uni.lu HPC Platform

- running in interactive mode
  - ↪ with either the full graphical or the text-mode interface
  - ↪ graphical web portal based on OnDemand coming soon
- running in passive mode
  - ↪ several ways of submitting MATLAB jobs
  - ↪ example launchers for SLURM
- checking available toolboxes & licenses status
- using script (.m) files
- plotting data, saving the plots to file

Summary

Tutorial files

Sample MATLAB scripts used in the tutorial

- download only the scripts

```
mkdir $HOME/matlab-tutorial
cd $HOME/matlab-tutorial
wget https://raw.github.com/ULHPC/tutorials/devel/maths/\
     matlab/basics/code/
                          example1.m
                          example2.m
                          google_finance_data.m
                          file_data_source.m
                          AAPL.csv
```

- *or* download the full repository and link to the MATLAB tutorial

```
git clone https://github.com/ULHPC/tutorials.git
ln -s tutorials/maths/matlab/basics $HOME/matlab-tutorial
```

X Window System

In order to see locally the MATLAB graphical interface, a package providing the X Window System is required:

- on OS X: **XQuartz** http://xquartz.macosforge.org/landing/
- on Windows:
  - ↪ in combination with Putty: **VcXsrv** http://sourceforge.net/projects/vcxsrv/
  - ↪ with MobaXTerm: nothing additional needed

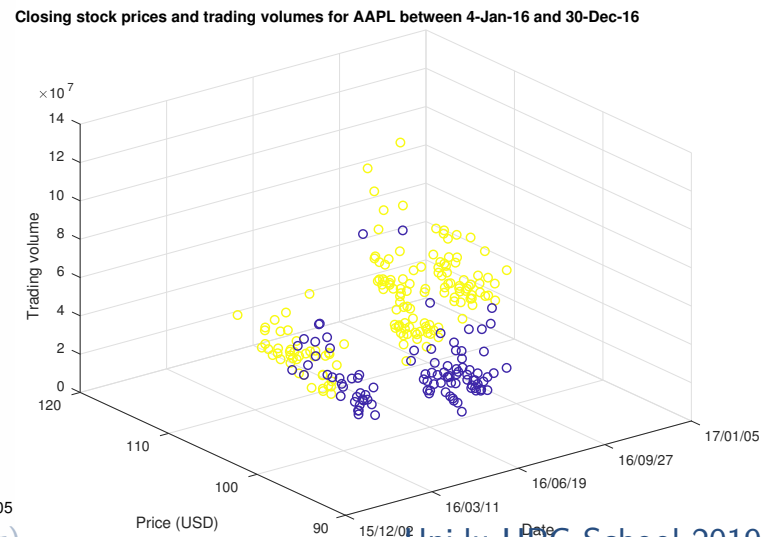
Now you will be able to connect with X11 forwarding enabled:

- on Linux & macOS: `ssh iris-cluster -X`
- on Windows
  - ↪ with Putty: Connection → SSH → X11 → **Enable X11 forwarding**
  - ↪ with MobaXTerm: remote GUI applications should work by default

Scripts and plots

example1.m: non-interactive script that shows

- the use of a stopwatch timer
- how to use an external function (financial data retrieval)
- how to use different plotting methods
- how to export the plots in different graphic formats



Parallelization

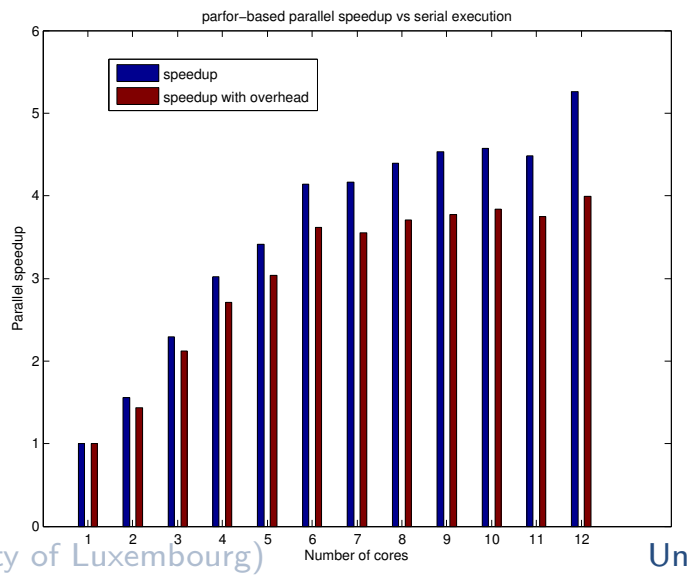
example2.m: non-interactive script that shows

- the serial execution of time consuming operations

Parallelization

example2.m: non-interactive script that shows

- the serial execution of time consuming operations
- and revisited in the second part of the tutorial:
  - ↪ the parallel execution and relative speedup vs serial execution
  - ↪ setting the # of parallel threads through environment variables
  - ↪ GPU-based parallel execution



Beyond simple executions

- application-level checkpointing
  - ↪ using in-built MATLAB functions

Beyond simple executions

- application-level checkpointing
  - ↪ using in-built MATLAB functions

- taking advantage of some parallelization capabilities
  - ↪ use of **parfor**
  - ↪ use of GPU-enabled functions

Beyond simple executions

- application-level checkpointing
  - ↪ using in-built MATLAB functions

- taking advantage of some parallelization capabilities
  - ↪ use of **parfor**
  - ↪ use of GPU-enabled functions

- adapting parallel code with checkpoint/restart features

Checkpointing

What is it?

Technique for adding fault tolerance to your application.

- You adapt your code to (regularly) save a snapshot of the environment (workspace)
- ... and restart execution from the snapshot in case of failure.

Checkpointing

What is it?

Technique for adding fault tolerance to your application.

- You adapt your code to (regularly) save a snapshot of the environment (workspace)
- ... and restart execution from the snapshot in case of failure.

Why make the effort to checkpoint?

- because your code may take longer to execute than the maximum walltime allowed
- because losing (precious) hours or days of computation **when** something fails may (should!) not be acceptable

Checkpointing pitfalls

- checkpointing (too) often can be counterproductive
  - ↪ saving state in each loop may take longer than its actual computing time
  - ↪ saving state incrementally can lead to fast exhaustion of your $HOME space
  - ↪ in extreme cases can lead to platform instability – especially if running parallel jobs!

Checkpointing pitfalls

- checkpointing (too) often can be counterproductive
  - ↪ saving state in each loop may take longer than its actual computing time
  - ↪ saving state incrementally can lead to fast exhaustion of your $HOME space
  - ↪ in extreme cases can lead to platform instability – especially if running parallel jobs!

- checkpointing (especially parallel) code can be tricky
- extra-care required if checkpointing simulations involving PRNG (e.g. Monte Carlo-based experiments)
- ensure results consistency after you add checkpointing

Checkpointing basics

1. Check that a checkpoint file exists:
```
exist('save.mat','file')
```
2. If it exists, restore workspace data from it:
```
load('save.mat')
```

Checkpointing basics

1. Check that a checkpoint file exists: `exist('save.mat','file')`
2. If it exists, restore workspace data from it: `load('save.mat')`

3. During computing steps, use control variables to direct (re)start of computation

Checkpointing basics

1. Check that a checkpoint file exists: `exist('save.mat','file')`
2. If it exists, restore workspace data from it: `load('save.mat')`

3. During computing steps, use control variables to direct (re)start of computation

4. Every *n* loops, or if execution time (in loop or since startup) is above threshold, checkpoint:
   - ↪ save full workspace state: `save('save.tmp')`
   - ↪ save partial state: `save('save.tmp', 'var1', 'var2')`

Checkpointing basics

1. Check that a checkpoint file exists: `exist('save.mat','file')`
2. If it exists, restore workspace data from it: `load('save.mat')`

3. During computing steps, use control variables to direct (re)start of computation

4. Every *n* loops, or if execution time (in loop or since startup) is above threshold, checkpoint:
   - ↪ save full workspace state: `save('save.tmp')`
   - ↪ save partial state: `save('save.tmp', 'var1', 'var2')`

5. Rename state file to final name: `system('mv save.tmp save.mat')`
   - ↪ this process ensures that in case of failure during checkpointing, next execution doesn't try to restart from incomplete state

When to trigger checkpointing?

- when (loop) execution time is above threshold (e.g. 1h):
  - ↪ use `tic` and `toc` stopwatch functions, remember they can be assigned to variables
  - ↪ use the `clock` function
  - ↪ add some **randomness** to the threshold if you run several instances in parallel!

When to trigger checkpointing?

- when (loop) execution time is above threshold (e.g. 1h):
  - ↪ use `tic` and `toc` stopwatch functions, remember they can be assigned to variables
  - ↪ use the `clock` function
  - ↪ add some **randomness** to the threshold if you run several instances in parallel!

- every *n* loop executions
  - ↪ remember that saving state takes time, depending on workspace size & shared filesystem usage, and
  - ↪ if loops finish fast your code may be slowed down considerably
  - ↪ add some **randomness** to *n* if you run several instances in parallel!

Adding checkpointing to seq. code

example1.m: non-interactive script that shows

- the use of a stopwatch timer
- how to use an external function (financial data retrieval)
- how to use different plotting methods
- how to export the plots in different graphic formats

Tasks to tackle with checkpointing

- modify the script to download data for Fortune100 companies
- add & test checkpointing to save state after each company's data is downloaded
- more granular downloads - modify download period from 1 year to 1 month, add & test checkpointing to save state after each download

Ref. documentation - parallelization

- **Parallel Computing Toolbox**

  http://www.mathworks.nl/help/distcomp/index.html
- Parallel for-Loops (parfor)

  http://www.mathworks.nl/help/distcomp/getting-started-with-parfor.html
- GPU Computing

  http://www.mathworks.nl/discovery/matlab-gpu.html
- Multi-GPU computing examples

  https://nl.mathworks.com/help/parallel-computing/examples/

  run-matlab-functions-on-multiple-gpus.html

Accelerating the time to result

- Option 1: Split input over several parallel, independent jobs
  - ↪ great **if** it's possible (embarrassingly parallel problem)

Accelerating the time to result

- Option 1: Split input over several parallel, independent jobs
  - ↪ great **if** it's possible (embarrassingly parallel problem)
- Option 2: Use **parfor** to execute loop iterations in parallel
  - ↪ single node **only**
  - ↪ Iris `bigmem` partition nodes with 112 cores for big problems

Accelerating the time to result

- Option 1: Split input over several parallel, independent jobs
  - ↪ great **if** it's possible (embarrassingly parallel problem)
- Option 2: Use **parfor** to execute loop iterations in parallel
  - ↪ single node **only**
  - ↪ Iris `bigmem` partition nodes with 112 cores for big problems
- Option 3: Use GPU-enabled functions that work on the **gpuArray** data type
  - ↪ **require** the code to be run on GPU nodes (subset of Iris)
  - ↪ **great speedup** for some workloads
  - ↪ **multiple hundreds** of in-built MATLAB functions work on gpuArray
    - ✓ including discrete Fourier transform, matrix multiplication, left matrix division

Accelerating the time to result

- Option 1: Split input over several parallel, independent jobs
  - ↪ great **if** it's possible (embarrassingly parallel problem)
- Option 2: Use **parfor** to execute loop iterations in parallel
  - ↪ single node **only**
  - ↪ Iris `bigmem` partition nodes with 112 cores for big problems
- Option 3: Use GPU-enabled functions that work on the **gpuArray** data type
  - ↪ **require** the code to be run on GPU nodes (subset of Iris)
  - ↪ **great speedup** for some workloads
  - ↪ **multiple hundreds** of in-built MATLAB functions work on gpuArray
    - ✓ including discrete Fourier transform, matrix multiplication, left matrix division
- Option 4: MATLAB Distributed Computing Server (MDCS)
  - ↪ allows multi-node parallel execution
  - ↪ **not yet** part of the MATLAB license

Speed up your seq. code

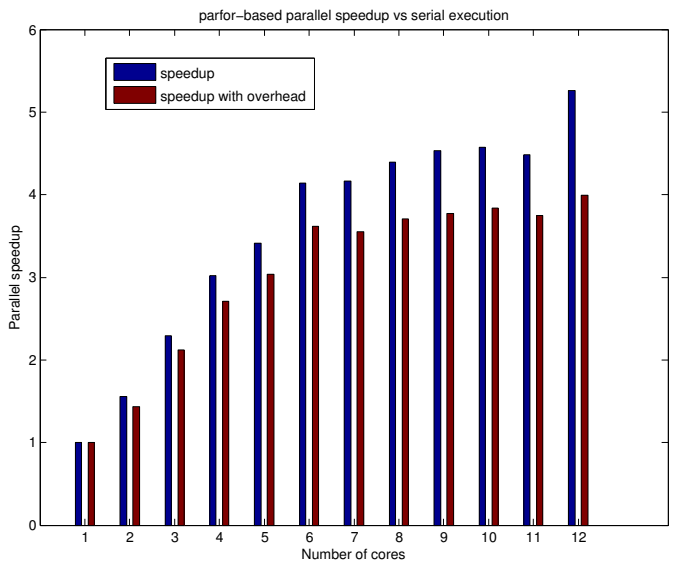
example2.m: non-interactive script that shows

- the serial execution of time consuming operations

Speed up your seq. code

example2.m: non-interactive script that shows

- the serial execution of time consuming operations
- and revisited in the second part of the tutorial:
  - ↪ the parallel execution and relative speedup vs serial execution
  - ↪ setting the # of parallel threads through environment variables
  - ↪ GPU-based parallel execution



Speed up your seq. code

example2.m: non-interactive script that shows

- the serial execution of time consuming operations
- and revisited in the second part of the tutorial:
  - ↪ the parallel execution and relative speedup vs serial execution
  - ↪ setting the # of parallel threads through environment variables
  - ↪ GPU-based parallel execution

Tasks to tackle

- execute the script on regular vs GPU nodes (with diff. GPUs)
- increase # of iterations, matrix size
- increase # of workers with/out changing # of req. cores
- modify the script with other GPU-enabled functions

Summary

Exercises - your mission today

- Read and understand the MATLAB tutorial

  https://github.com/ULHPC/tutorials/tree/devel/maths/matlab

  ↪ all provided scripts are fully commented
- Run all the examples
  - ↪ launching interactive/passive mode MATLAB
  - ↪ plotting script
  - ↪ parallel execution script

Useful links

- Getting Started with Parallel Computing Toolbox

  http://nl.mathworks.com/help/distcomp/getting-started-with-parallel-computing-toolbox.html

- Parallel for-Loops (parfor) documentation

  https://nl.mathworks.com/help/distcomp/parfor.html

- GPU Computing documentation

  https://nl.mathworks.com/discovery/matlab-gpu.html

- Multi-GPU computing examples

  https://nl.mathworks.com/help/parallel-computing/examples/run-matlab-functions-on-multiple-gpus.html

What we've seen so far (I)

- MATLAB execution modes on the Uni.lu HPC Platform
- Checking for available toolboxes and licenses
- Basics of plotting

Perspectives

- Personalize the UL HPC launchers with the MATLAB commands
- Check example #2 M-file for insight into basic parallel execution
- Parallelize your own tasks using parfor/GPU-enabled instructions

What we've seen so far (II)

- Checkpointing basics
- Specific MATLAB instructions for checkpointing
- MATLAB parallelization capabilities

Perspectives

- (incrementally) modify your own MATLAB code for fault tolerance
- parallelize your own tasks using parfor/GPU-enabled instructions

Questions?

http://hpc.uni.lu

High Performance Computing @ uni.lu

Prof. Pascal Bouvry
Dr. Sebastien Varrette
Valentin Plugaru
Sarah Peter
Hyacinthe Cartiaux
Clement Parisot
Dr. FrÃľderic Pinel
Dr. Emmanuel Kieffer

University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu