

# Matlab @ CINECA: Getting started with serial and parallel

In this page:

- [Pre-requisites](#)
- [Configuration](#)
  - [Remote Jobs submission](#)
  - [On Cluster submission](#)
  - [Configuring Jobs](#)
- [Serial Jobs](#)
- [Parallel Jobs](#)
  - [Interactive Jobs](#)
  - [Batch Jobs](#)
    - [Example 1](#)
    - [Example 2](#)
- [Debugging](#)
- [To learn More](#)
- [Parallel Computing Benchmark and Performance](#)
- [Troubleshooting](#)

---

The following guide describes how to load, configure and use MATLAB @ CINECA's cluster.  
At the moment MATLAB is available on [Marconi](#) and [Galileo100](#) clusters.

## Pre-requisites

To use MATLAB on CINECA HPC's environment, please check the following pre-requisites:

1. You and your collaborators have a **valid account** defined on HPC cluster, see "[Become a User](#)".
2. You have access to a **valid MATLAB license** to be used on CINECA HPC clusters.

Thanks to an agreement with MathWorks, **CINECA provides several MATLAB licenses** through its own license server that can be used on CINECA clusters.

Usage of the CINECA MATLAB licenses is allowed **exclusively for Open Science** (non-commercial) activities.

In case you are interested in using those licenses and you declare that your activity is devoted to Open Science, please write to [superc@cinca.it](mailto:superc@cinca.it) to be enabled to use CINECA licenses.

For all the other cases, or in case you would like to use your personal/department/university licenses we need to connect your license server, where a Flex-LM license is installed, with the CINECA compute nodes of the cluster. Please write to CINECA's staff at [superc@cinca.it](mailto:superc@cinca.it) to:

- provide us the port and host (IP or alias) of the license server where the MATLAB license is installed.
- the license holder needs to sign a document in which the holder declares to have a valid license and to relieve CINECA of future responsibilities for the usage of that license on CINECA's cluster.
- the license server administrators have to open their firewall to the IPs of CINECA's cluster.

When the aforementioned steps have been completed, your usernames and account(s) will be authorized to use your academic license of MATLAB to run your jobs on CINECA infrastructure.

You will also have to indicate us a representative (not necessarily the license holder but with his/her approval) to be contacted by CINECA HPC User Support to allow future requests to use your license server.

## Configuration

This section provides the steps to configure MATLAB to submit jobs to a cluster, retrieve results, and debug errors.

It is possible to configure MATLAB in order to submit jobs on CINECA clusters directly from your local MATLAB installation (Remote Jobs submission) or by login nodes of CINECA clusters (on Cluster submission).

### Remote Jobs submission

It is possible to submit MATLAB jobs to the compute nodes of a CINECA cluster directly from your local MATLAB installation.

Please check that the **"Parallel Computing Toolbox" is installed** on your computer.

To check for it, you can run the following commands on MATLAB:

```
>> license('test','distrib_computing_toolbox')
>> ~isempty(ver('parallel'))
```

if answer is 1 for both, it means that the Parallel toolbox is correctly installed. Otherwise you need to install it.

**Download** the following ZIP file ([cineca.Desktop.zip](#)). It contains a set of scripts needed to configure MATLAB to launch jobs remotely. Unzip the file in the location returned by the MATLAB command:

```
>> userpath
```

For different solutions you can refer to this [MathWorks dedicated User Guide page](#).

Open MATLAB and launch the command

```
>> configCluster
```

The command will ask you to insert the username you would like to use to access the CINECA cluster.

**Important:** you can access only clusters where you have an active budget account.

To manage the local cluster configuration in the top menu select "Parallel", then "Create and Manage Clusters..."

A window will be opened where you can modify the Additional Properties of your configuration based on your needs (See next Sections about a description of the Available Properties).

## On Cluster submission

Alternatively to Remote submission, you can launch MATLAB jobs directly from login nodes of CINECA clusters.

Log-in to Marconi or Galileo100 cluster and load the MATLAB module:

```
$ module load profile/eng
$ module load autoload matlab/<version>
```

There may be available more than one MATLAB version. You can check for it through the command [modmap](#).

Take care to select the last valid release for your license.

Configure MATLAB to run parallel jobs. This only needs to be called once per version of MATLAB and once per user.

```
$ configCluster.sh <account-name>
```

in alternative you can start MATLAB without desktop

```
$ matlab -nodisplay
```

then launch the command

```
>> configCluster
```

A new profile will be created ('cineca local R2018a' on Marconi or 'galileo100 R2020b' on Galileo100).

You can check the list of available profiles:

```
>> [ ALLPROFILES,DEFAULTPROFILE] = parallel.clusterProfiles
ALLPROFILES =
    1x2 cell array

    {'galileo100 R2020b'} {'local'}

DEFAULTPROFILE=

    'galileo100 R2020b'
```

Please check that the DEFAULTPROFILE is not set to 'local'.

The 'local' profile is not allowed on our cluster, so **don't use it**.

If it is set to 'local' you have to set

```
>> DEFAULTPROFILE='galileo100 R2020b'
```

on Galileo100 or 'cineca local R2018a' on Marconi.

## Configuring Jobs

Prior to submitting the job, various parameters have to be specified in order to be passed to jobs, such as queue, username, e-mail, etc.

NOTE: Any parameters specified using the below workflow will be persistent between MATLAB sessions if saved at the end of the configuration.

Before specifying any parameters, you will need to obtain a handle to the cluster object.

```
>> % Get a handle to the cluster
>> c = parcluster;
```

You are **required** to specify an Account Name, a Queue Name and the Wall Time prior to submitting a job. You can retrieve your Account Name / Budget info by using the [saldo](#) command

```
>> % Specify an Account to use for MATLAB jobs
>> c.AdditionalProperties.AccountName = 'account_name';

>> % Specify a queue to use for MATLAB jobs
>> c.AdditionalProperties.QueueName = 'queue-name';

>> % Specify the walltime (e.g. 5 hours)
>> c.AdditionalProperties.WallTime = '05:00:00';
```

For Galileo100 cluster queue name is generally 'g100\_usr\_prod'. In [Galileo100](#) dedicated page you can find other possible Queue and QOS available allowing for different combinations of nodes, walltime and priority.  
For Marconi cluster queue name can be 'skl\_usr\_dbg' or 'skl\_usr\_prod' with wall time, respectively, 30 minutes or 24 hours. You can find more details on [Marconi](#) dedicated page.

You can specify other **additional** parameters along with your job.

```
>> % Specify QoS
>> c.AdditionalProperties.QoS = 'name-of-qos';

>> % Specify the number of nodes you want to use.
>> c.AdditionalProperties.NumberOfNodes = 1;

>> % Specify processor cores per node. Default is 18 for Marconi and 48 for Galileo100.
>> c.AdditionalProperties.ProcsPerNode = 18;

>> % Specify memory to use for MATLAB jobs, per core (default: 4gb)
>> c.AdditionalProperties.MemUsage = '6gb';

>> % Require node exclusivity
>> c.AdditionalProperties.RequireExclusiveNode = true;

>> % Request to use a reservation
>> c.AdditionalProperties.Reservation = 'name-of-reservation';

>> % Specify e-mail address to receive notifications about your job
>> c.AdditionalProperties.EmailAddress = 'test@foo.com';

>> % Turn on the Debug Message. Default is off (logical boolean true/false).
>> c.AdditionalProperties.DebugMessagesTurnedOn = true;
```

On Galileo100 it is also possible to make use of GPUs for visualization and interactive jobs

```
>> % Request a number of GPU cards
>> c.AdditionalProperties.GpuCard = 'name-of-GPU-card';
>> c.AdditionalProperties.GpusPerNode = 1;
```

In this case you can select 'g100\_usr\_interactive' as Queue Name.

To see the values of the current configuration options, call the specific AdditionalProperties name.

```
>> % To view current configurations
>> c.AdditionalProperties.QueueName
```

Or to see the entire configuration

```
>> c.AdditionalProperties
```

To clear a value, assign the property an empty value ('', [], or false).

```
>> % To clear a configuration that takes a string as input
>> c.AdditionalProperties.EmailAddress = '';
```

To save a profile, with your configuration so you will find it in future sessions

```
>> c.saveProfile;
```

## Serial Jobs

Use the batch command to submit asynchronous jobs to the cluster. The batch command will return a job object which is used to access the output of the submitted job. See the [MATLAB documentation for more help on batch](#).

```
>> % Get a handle to the cluster
>> c = parcluster;
```

Submit job to query where MATLAB is running on the cluster

```
>> j = c.batch(@pwd, 1, {});
```

Query job for state: queued | running | finished

```
>> j.State
```

If state is finished, fetch results

```
>> j.fetchOutputs{:}
```

or

```
>> fetchOutputs(j)
```

Display the diary

```
>> diary(j)
```

Delete the job after results are no longer needed

```
>> j.delete;
```

To retrieve a list of currently running or completed jobs, call `parcluster` to retrieve the cluster object. The cluster object stores an array of jobs that were run, are running, or are queued to run. This allows us to fetch the results of completed jobs. Retrieve and view the list of jobs as shown below.

```
>> c = parcluster;
```

```
>> jobs = c.Jobs;
```

Once we've identified the job we want, we can retrieve the results as we've done previously.

`fetchOutputs` is used to retrieve function output arguments; if using batch with a script, use `load` instead. Data that has been written to files on the cluster needs be retrieved directly from the file system.

To view results of a previously completed job:

```
>> % Get a handle on job with job array index 2 (2x1)j
```

```
>> j2 = c.Jobs(2);
```

NOTE: You can view a list of your jobs, as well as their IDs, using the above `c.Jobs` command.

```
>> % Fetch results for job with ID 2
```

```
>> j2.fetchOutputs{:}
```

If the job produces an error view the error log file

```
>> c.getDebugLog(j.Tasks(1))
```

NOTE: When submitting independent jobs, with multiple tasks, you will have to specify the task number.

## Parallel Jobs

### Interactive Jobs

To run an interactive pool job on the cluster, you can use `parpool`.

```
>> % Get a handle to the cluster
```

```
>> c = parcluster;
```

Open a pool of 64 workers on the cluster

```
>> pool = c.parpool(64);
```

Rather than running local on the local machine, the pool can now run across multiple nodes on the cluster.

```
>> % Run a parfor over 1000 iterations
```

```
>> parfor idx = 1:1000
```

```
    a(idx) = rand();
```

```
end
```

Once we're done with the pool, delete it.

```
>> % Delete the pool
```

```
>> pool.delete;
```

## Batch Jobs

Users can also submit parallel workflows with batch.

The following example are available at the following directory available after loaded the module

CIN\_EXAMPLE=/cineca/prod/opt/tools/matlab/CINECA\_example

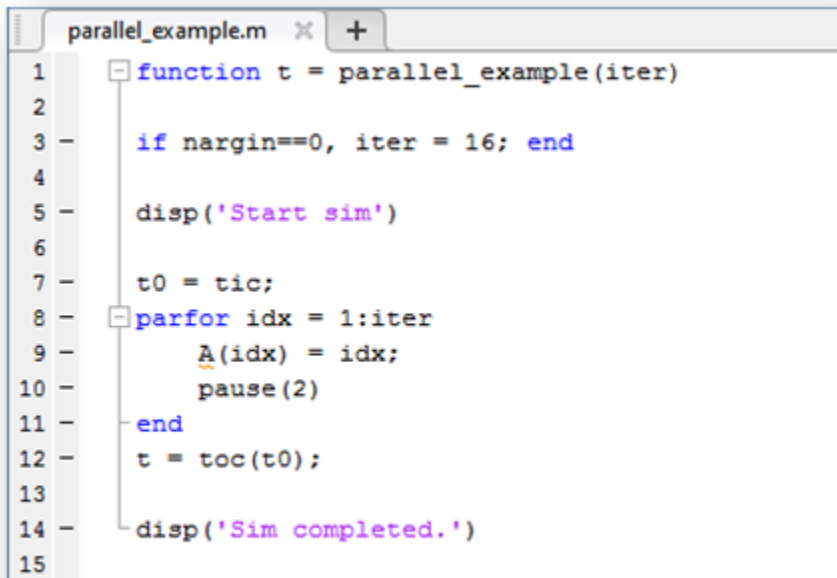
Example 1: [parallel\\_example.m](#) (click to download it)

Example 2: [hpcclLinpack.m](#) (click to download it)

### Example 1

parallel\_example.m

Let's use the following example for a parallel job.



```
1 function t = parallel_example(iter)
2
3     if nargin==0, iter = 16; end
4
5     disp('Start sim')
6
7     t0 = tic;
8     parfor idx = 1:iter
9         A(idx) = idx;
10        pause(2)
11    end
12    t = toc(t0);
13
14    disp('Sim completed.')
15
```

We will use the batch command again, but since we're running a parallel job, we'll also specify a MATLAB Pool.

```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Submit a batch pool job using 4 workers for 16 iterations
>> j = c.batch(@parallel_example, 1, {}, 'Pool', 4);
```

For more info on the batch commands, please see the [MATLAB on-line guide](#).

```
>> % View current job status

>> j.State
```

```
>> % Fetch the results after a finished state is retrieved

>> j.fetchOutputs{:}

>> ans = 15.5328
```

```
>> % Display the diary

>> diary(j)
```

The job ran in 15.53 sec. using 4 workers.

**Note that these jobs will always request N+1 cores for your job**, since one worker is required to manage the batch job and pool of workers.

For example, a job that needs eight workers will consume nine CPU cores.

We'll run the same simulation, but increase the Pool size. This time, to retrieve the results at a later time, we'll keep track of the job ID.

NOTE: For some applications, there will be a diminishing return when allocating too many workers, as the overhead may exceed computation time.

```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Submit a batch pool job using 8 workers for 16 simulations
>> j = c.batch(@parallel_example, 1, {}, 'Pool', 8);

>> % Get the job ID
>> id = j.ID

Id = 4

>> % Clear workspace, as though we quit MATLAB
>> clear j
```

Once we have a handle to the cluster, we'll call the findJob method to search for the job with the specified job ID.

```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Find the old job
>> j = c.findJob('ID', 4);

>> % Retrieve the state of the job
>> j.State

ans

finished

>> % Fetch the results
>> j.fetchOutputs{:}

ans =

6.4488

>> % If necessary, retrieve output/error log file
>> c.getDebugLog(j)
```

The job now runs 6.4488 seconds using 8 workers. Run code with different number of workers to determine the ideal number to use.

## Example 2

hpccLinpack.m

This example is taken from \$MATLAB\_HOME/toolbox/distcomp/examples/benchmark/hpcchallenge/

It is an implementation of the HPCC Global HPL benchmark

```
>> function perf = hpccLinpack( m )
```

The function input is the size of the real matrix m-by-m to be inverted. The outputs is perf, performance in gigaflops

Start to submit on 1 core, with m=1024:

```
>> j = c.batch(@hpccLinpack, 1, {1024}, 'Pool', 1)

Data size: 0.007812 GB
Performance: 1.576476 GFlops
```

Repeat on one full node on Marconi

```
>> j = c.batch(@hpccLinpack, 1, {1024}, 'Pool', 35)

Data size: 0.007812 GB
Performance: 0.311111 GFlops
```

Increase the size of the matrix,

```
>> j = c.batch(@hpccLinpack, 1, {2048}, 'Pool', 35)
```

```
Data size: 0.031250 GB  
Performance: 2.466961 GFlops
```

```
>> j = c.batch(@hpccLinpack, 1, {4096}, 'Pool', 35)
```

```
Data size: 0.125000 GB  
Performance: 47.951919 GFlops
```

Use two full nodes on Marconi

```
Data size: 0.125000 GB  
Performance: 14.709730 GFlops
```

and double matrix size

```
>> j = c.batch(@hpccLinpack, 1, {8192}, 'Pool', 71)
```

```
Data size: 0.500000 GB  
Performance: 86.003520 GFlops
```

```
...
```

```
..
```

```
>> j = c.batch(@hpccLinpack, 1, {16384}, 'Pool', 35)
```

```
Data size: 2.000000 GB  
Performance: 356.687648 GFlops
```

## Debugging

If a serial job produces an error, we can call the `getDebugLog` method to view the error log file.

```
>> j.Parent.getDebugLog(j.Tasks(1))
```

When submitting independent jobs, with multiple tasks, you will have to specify the task number. For Pool jobs, do not deference into the job object.

```
>> j.Parent.getDebugLog(j)
```

The scheduler job ID can be derived by calling `schedID`

```
>> schedID(j)
```

```
ans
```

```
25539
```

## To learn More

To learn more about the MATLAB Parallel Computing Toolbox, check out these resources:

- [Hands-On Workshop@CINECA](#)
- [Exercises Workshop Day 1](#)
- [Exercises Workshop Day 2](#)
- [Parallel Computing Coding Examples](#)
- [Parallel Computing Documentation](#)
- [Parallel Computing Overview](#)
- [Parallel Computing Tutorials](#)
- [Parallel Computing Videos](#)
- [Parallel Computing Webinars](#)

## Parallel Computing Benchmark and Performance

- [Benchmarking Parfor Performance Using a Simple Example: Game of Blackjack](#)
- [Resource Contention in Task Parallel Problems](#)
- [Benchmarking Distributed Jobs \(Task Parallel Applications on the Cluster\)](#)
- [Benchmarking Parallel "\ Operator \(A\b\)](#)
- [Profiling Load Unbalanced Distributed Arrays in Data Parallel Applications](#)
- [Profiling Explicit Parallel Communication while using Message Passing Functions in MATLAB](#)

## Troubleshooting

1) If you have the following issue when launching a parallel job:

```
About to evaluate task with DistcompEvaluateFileTask
About to evaluate task with DistcompEvaluateFileTask
About to evaluate task with DistcompEvaluateFileTask
Enter distcomp_evaluate_filetask_core
Enter distcomp_evaluate_filetask_core
```

This process will exit on any fault.

....

..

Error initializing MPI: Undefined function or variable 'mpiInitSigFix'.

Please type on matlab command line:

```
>> rehash toolboxcache
```

To updates the cache file, and make permanent the patch for InfiniBand QDR applied on top of the standard matlab installation.

2) Using MATLAB versions r2023a and following when running MATLAB on the login node and opening a parpool session you may end in the error

Parallel pool failed to start with the following error. For more detailed information, validate the profile 'galileo100 R2022b' in the Cluster Profile Manager.

Caused by:

```
Error using parallel.internal.pool.AbstractInteractiveClient>iThrowWithCause
Failed to initialize the interactive session.
Error using parallel.internal.pool.SpfcClientSession
An unexpected error occurred accessing a parallel pool. The underlying error was: Failed to connect to
endpoint after 5 attempts.
Cause: Attempt 2
Unable to connect to tcp://tcpnodelay=r500n002:27771/protocol/catapult
because: Unable to connect to: tcp://tcpnodelay=r500n002:27771/protocol/catapult
Error: Software caused connection abort [system:103].
```

This is due to a recent change done on version r2023a and following. You can solve the error by adding setting a specific additional property

c.AdditionalProperties.ClientConnectsToWorkers = false;

---

dependences:

- [MATLAB: local installation](#)