

Getting Started with Serial and Parallel MATLAB on CoolMUC

CONFIGURATION – If running MATLAB client on the cluster

Start MATLAB. Configure MATLAB to run parallel jobs on your cluster by calling `configCluster`. For each cluster, `configCluster` only needs to be called once per version of MATLAB.

```
>> configCluster
```

Jobs will now default to the cluster rather than submit to the local machine.

CONFIGURATION – If running MATLAB client on your personal laptop or desktop

Add the MATLAB integration scripts to your MATLAB Path by placing the integration scripts into `$MATLAB/toolbox/local`.

Start MATLAB. Configure MATLAB to run parallel jobs on your cluster by calling `configCluster`. For each cluster, `configCluster` only needs to be called once per version of MATLAB.

Submission to the remote cluster requires SSH credentials. You will be prompted for your ssh username and password or identity file (private key). The username and location of the private key will be cached in MATLAB for future sessions.

CONFIGURING JOBS

Prior to submitting the job, we can specify various parameters to pass to our jobs, such as queue, username, walltime, e-mail, etc. None of these are required to be set.

NOTE: Any parameters specified using the below workflow will be persistent between MATLAB sessions.

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

>> % Specify a particular Account to use for MATLAB jobs
>> c.AdditionalProperties.AccountName = 'account-name';

>> % Specify a particular queue to use for MATLAB jobs
>> % Parallel jobs: mpp2_batch | Serial jobs: serial
>> c.AdditionalProperties.QueueName = 'queue-name';

>> % Specify a particular cluster to use for MATLAB jobs
>> % Parallel jobs: mpp2 | Serial jobs: serial
>> c.AdditionalProperties.Cluster = 'cluster';

>> % Specify e-mail address to receive notifications about your job
>> c.AdditionalProperties.EmailAddress = 'user-id@company.com';
```

```
>> % Specify the walltime (e.g. 1 hour)
>> c.AdditionalProperties.WallTime = '01:00:00';

>> % Specify the memory requirement for your job (per CPU core)
>> c.AdditionalProperties.MemUsage = '3G';
```

Save changes after modifying AdditionalProperties fields.

```
>> c.saveProfile
```

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

```
>> % To view current properties
>> c.AdditionalProperties
```

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

```
>> % Turn off email notifications
>> c.AdditionalProperties.EmailAddress = ' ';
```

INTERACTIVE JOBS – If running MATLAB client on the cluster

To run an interactive pool job on the cluster, continue to use parpool as before.

```
>> % Open a pool of 32 workers on the cluster
>> p = parpool(32);
```

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

```
>> % Run a parfor over a 1000 iterations
>> parfor idx = 1:1000
    a(idx) = ...
end
```

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

```
>> % Delete the pool
>> p.delete
```

SERIAL JOBS

Rather than running interactively, 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
>> j.State

>> % If state is finished, fetch results
>> j.fetchOutputs{:}

>> % 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 ID 2
>> 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))

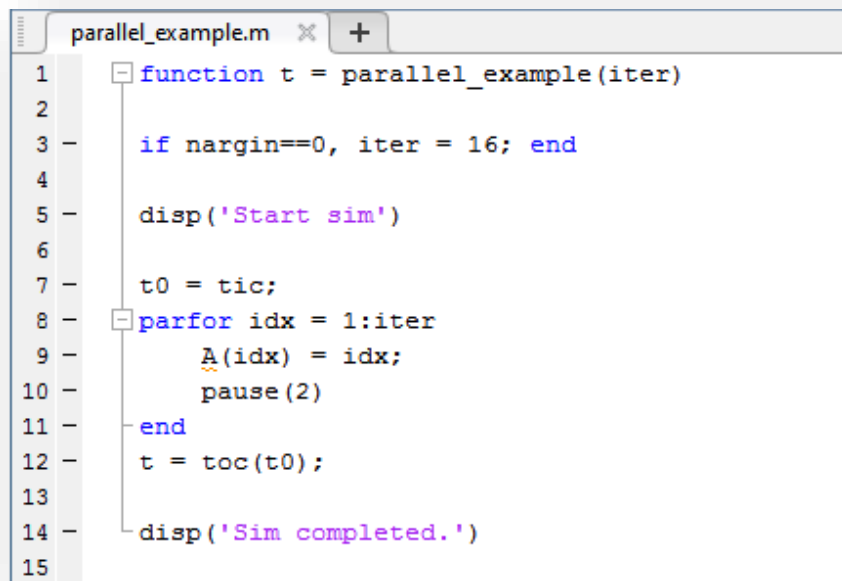
```

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

PARALLEL JOBS

Users can also submit parallel workflows with `batch`. **Coolmuc 2 has 28 core nodes, so when requesting resources make sure to stay within 1 node (28 cores) or fully utilize the additional node(s). Keep in mind that the number of workers used for a batch job will always be the number of workers you specify on the batch command line +1.**

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

A screenshot of a MATLAB script editor window titled 'parallel_example.m'. The script contains the following code:

```
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'll 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 simulations
>> j = c.batch(@parallel_example, 1, {}, 'Pool', 4);

>> % View current job status
>> j.State

>> % Fetch the results after a finished state is retrieved
>> j.fetchOutputs{:}

ans =

    8.8872
```

The job ran in 8.89 seconds using 4 workers. Note that these jobs will always request N+1 CPU cores, 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 later, 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{:};

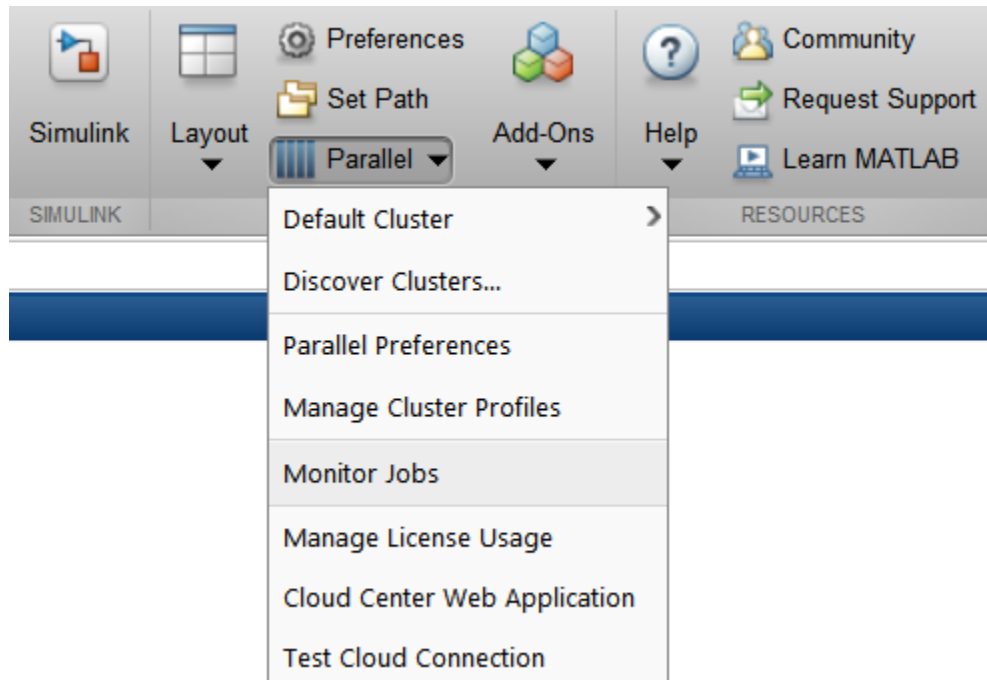
ans =

    4.7270

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

The job now runs 4.73 seconds using 8 workers. Run code with different number of workers to determine the ideal number to use.

Alternatively, to retrieve job results via a graphical user interface, use the Job Monitor (Parallel > Monitor Jobs).



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 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:

- [Parallel Computing Coding Examples](#)
- [Parallel Computing Documentation](#)
- [Parallel Computing Overview](#)
- [Parallel Computing Tutorials](#)
- [Parallel Computing Videos](#)
- [Parallel Computing Webinars](#)