Getting Started with Serial and Parallel MATLAB Local

**CONFIGURATION**

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.

**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

>> 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{:}

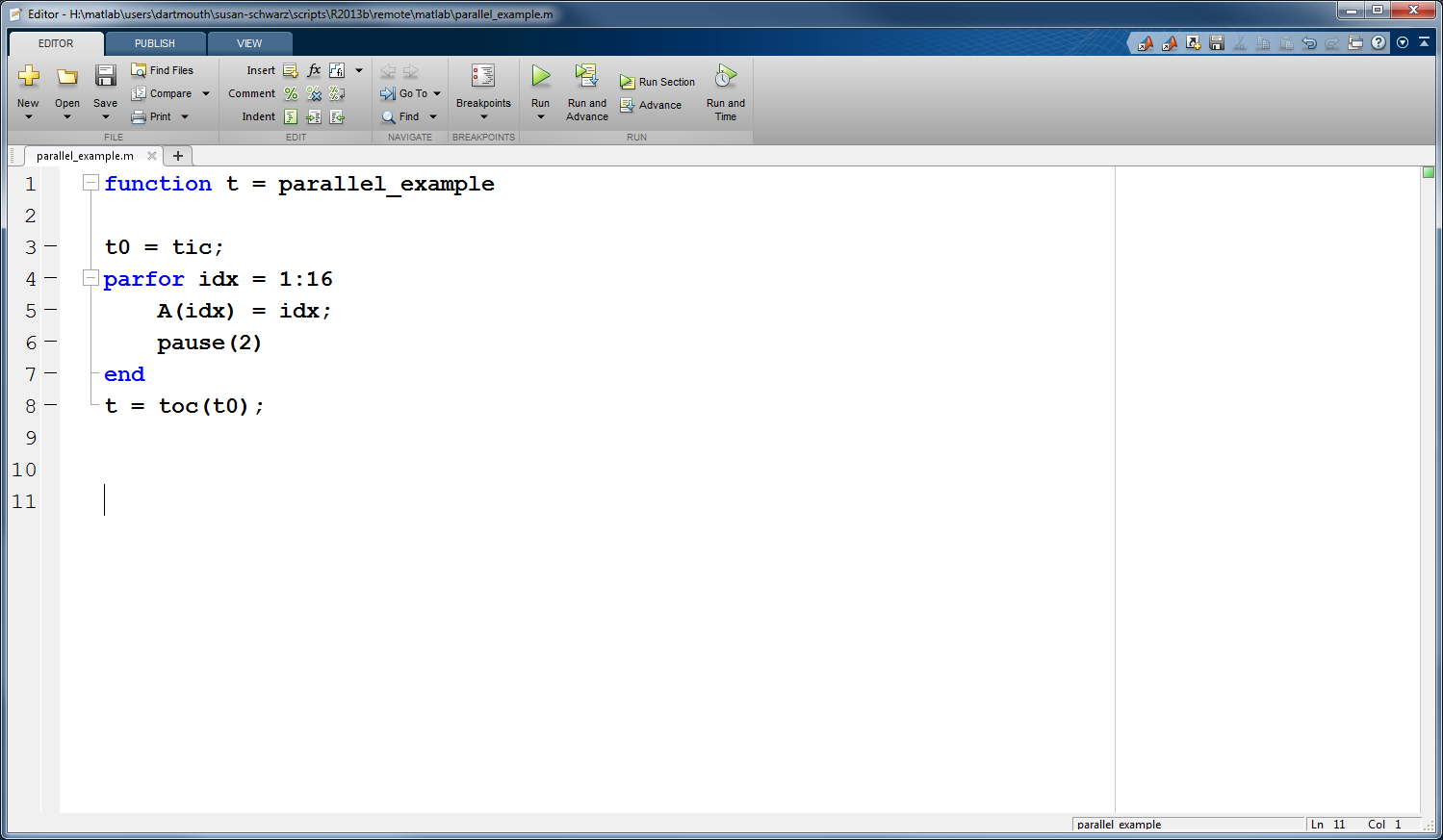
>> % If the job produces an error view the error log file

>> 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. Let’s use the following example for a parallel job.



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.m, 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 8.8872 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 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.m, 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 =

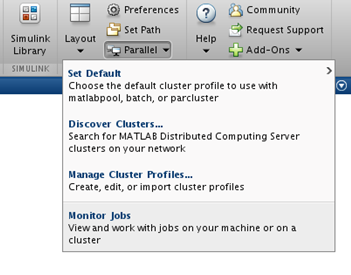
4.7270

>> % If necessary, retrieve output/error log file

>> c.getDebugLog(j)

The job now runs 4.7270 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).



**CONFIGURING JOBS**

Prior to submitting the job, we can specify various parameters to pass to our jobs, such as queue, e-mail, etc.

Specification is done with ClusterInfo. The ClusterInfo class supports tab completion to ease recollection of method names.

NOTE: Any parameters set with ClusterInfo will be persistent between MATLAB sessions.

>> % Specify a particular queue to use for MATLAB jobs

>> ClusterInfo.setQueueName(‘queue’)

>> % Specify e-mail address to receive notifications about your job

>> ClusterInfo.setEmailAddress(‘test@foo.com’)

To see the values of the current configuration options, call the state method. To clear a value, assign the property an empty value (‘’, [], or false), or call the clear method to clear all values.

>> % To view current configurations

>> ClusterInfo.state

>> % To clear a configuration that takes a string as input   
>> ClusterInfo.setEmailAddress(‘ ’)

>> % To clear all configurations

>> ClusterInfo.clear

TO LEARN MORE

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

* [Parallel Computing Coding Examples](http://www.mathworks.com/products/parallel-computing/code-examples.html)
* [Parallel Computing Documentation](http://www.mathworks.com/help/distcomp/index.html)
* [Parallel Computing Overview](http://www.mathworks.com/products/parallel-computing/index.htmlhttp:/www.mathworks.com/products/parallel-computing/index.html)
* [Parallel Computing Tutorials](http://www.mathworks.com/products/parallel-computing/tutorials.html)
* [Parallel Computing Videos](http://www.mathworks.com/products/parallel-computing/videos.html)
* [Parallel Computing Webinars](http://www.mathworks.com/products/parallel-computing/webinars.html)