University of Bristol BlueCrystal Matlab Job Manager

Overview

This is an application for creating, submitting and managing Matlab jobs on a University of Bristol BlueCrystal cluster. The cluster does not support conventional automatic Matlab parallelisation and requires Matlab tasks to be parallelised manually. This involves splitting Matlab task into subtasks and running subtasks in parallel on multiple machines and then aggregating results. Each subtask is represented by a Matlab function which is executed by 'Job Script'. Each subtask has to do a unique work and the easiest way to tell subtask which work it has to do is by passing parameters to the Matlab function.

This application automates creation and submission of BlueCrystal jobs where each job calls a Matlab function with different parameters.

The application has number of interfaces. You can select interface you want in menu when you start application or you can select interface by passing argument to the application. Note: the only available interface at the moment is command line interface (CLI).

Running a Matlab code on the cluster step by step

Installing the application

1. Copy unzipped application folder to the cluster
2. Connect to the server using SSH. (enable X11 forwarding if you are planning to use GUI).
3. Add python3 module loading to `.bashrc` file (add `module add languages/python-3.3.2` line to `.bashrc` file if it is not there yet). The jobs which are created require Python interpreter, and will fail to start unless you are loading Python >= 3.3 module in you `.bashrc` file.

Common steps

1. Connect to the server using SSH. (enable X11 forwarding if you are planning to use GUI).
2. Change directory to the application folder
3. Run the application: `python3 main.py`
4. Select interface you want to use and proceed with instruction in corresponding section

Command Line Interface (CLI)

1. You can get straight to CLI interface by running `python3 main.py cli` instead of `python3 main.py`
2. 1. Specify project folder (the folder with subfolders will be created after you press enter)
   2. If the project already exist you will have an option to continue with existing project
   3. If project directories where result is stored are not empty you will option to clear them
   4. Enter matlab function name
   5. Enter matlab function parameters (see detailed instruction below)
   6. Select matlab version you want to use
   7. Enter wall time in hours (e.g. 10 for 10 hours)
3. Jobs are now ready for submission. Before submitting jobs to the cluster you can check function parameters for each job in `Jobs.csv` file.
4. Submit jobs. The application will start submitting jobs for processing. You can stop submission at any time (using Ctrl-C) and continue it later by starting application, choosing same folder and selecting 'continue project'.
5. After all jobs have been submitted the application will periodically (every 5 seconds) check for job status and print status on screen. You can stop at any moment by pressing Ctl-C, this will not stop jobs, they will continue to run on the cluster. You can start monitoring status again by launching application, choosing same folder and selecting 'continue project'.
6. After all jobs completed the application will update `Jobs.csv` file, print summary and exit. You can now copy files produced by jobs from cluster and process them.

N.B. `Jobs.csv` file and log files are updated by the application. This means you need to have application running at same point after all jobs are finished. Otherwise `Jobs.csv` file will not contain all values, `Temp` folder will not be empty and log files will not be processed.

Project Structure

Each project has its own folder. if the folder does not exist it is created automatically by the application. The project consist of 5 folders and 2 files.

Folders

- `Code` You need to place a '.m' file with the function in this folder.
- `Data` It is recommended to put data which is used by Matlab code in this folder (e.g. antenna patterns, raytracing data).
- `Output` It is recommended to store all results which are produced by Matlab in this folder.
- `Log` In this folder log files which need attentions are stored. At the moment all non-empty log files are copied to this folder. Filenames have following pattern: <Job

internal id>_<Log file source>.txt
- `Temp` This folder is used for temporary files and is normally empty after all jobs are completed.

When new project is started there is an option to erase all files in `Output`, `Log` and `Temp` folders to avoid interference which previously generated results.

! N.B. ! The working directory for the function is the **top level project folder** and not **Code** folder. Take this into account when opening files in Matlab. E.g. When you want to store `1.m` file in `Output` you have to use `./Output/1.m` as a filename.

Files

- `project.conf` This file contains values which were used last time. The values are used as a suggestions when project is modified.
- `Jobs.csv` This file contains information about jobs in comma separated values format. The file is modified by the application and thus is updated only when application is running. You can open the file in any text editor or Spreadsheet application e.g. Google drive or Microsoft Excel.

Matlab functions

The application will submit jobs to cluster, each job will be executed on a separate computer. The job will do some housekeeping and will executed a Matlab function which you specified in the project settings. You need to provide the Matlab function code in m file. The function has to take as many parameters as you specified in the project settings and it has to return zero or one value. If function returns a value it has to be a string with parameters description. This string is used only for reference and will be shown in jobs summary table after jobs will be completed. Refer to Project Structure section for details of where to place the function file and how to organise I/O.

N.B. Filesystem is shared between the cluster nodes. For this reason each job should write results to files with names unique to the job. Otherwise jobs might access same files concurrently and destroying information in the files.

You can find examples of Matlab functions in 'MatlabCode' folder. Function do_work1.m takes one argument and does not return a description. Function do_work2.m takes two arguments and returns a description. The functions do nothing useful, but can be used as a starting point for writing your own functions.

Function parameters

You need to enter parameters that will specify how the inputs to the function will be generated. Each parameter is in general a vector that can be specified by its individual elements or as a range. In a special case, a vector can consist of one element only. Each parameter specification is separated by a semicolon. If you specify the vector using individual elements (values separated by commas) each element will be treated as literals and will be used as an input to function without any modifications. Use following syntax to

specify vector as a range: "<Start>..<Stop>[s<Step>[r]]". The application will generate values from start value to stop value inclusively. Step value equal to 1 if it is not set explicitly. The last option 'r' allows passing both start and stop points to the function (see example below). At the moment range notation can be used only with integer numbers.

Examples

f(1,2;5..7) will result in 6 jobs with following parameters:

- f(1, 5)
- f(1, 6)
- f(1, 7)
- f(2, 5)
- f(2, 6)
- f(2, 7)

g(b,g,n/ac;200..400s80r) will result in 9 jobs with following parameters:

- g(b, 200, 279)
- g(b, 280, 359)
- g(b, 360, 400)
- g(g, 200, 279)
- g(g, 280, 359)
- g(g, 360, 400)
- g(n/ac, 200, 279)
- g(n/ac, 280, 359)
- g(n/ac, 360, 400)

Parameters Notes

Empty parameters and values in lists are ignored. For example if you specify (1,,,3;;5) as a parameter generator for my_function there will be only 2 jobs created:

- my_function(1,5)
- my_function(3,5)

If you want to pass a literal parameter which contains '..' you can add comma before or after the value. The comma will result in parameters parser interpreting the parameter as list and will ignore '..' in the text. Example: if you want a job script to call my_function(1..2) in Matlab you need to specify (1..2,) as a parameter.