Parallel Image Analysis

Lab session 1:

Creating stand-alone executables using the MATLAB Compiler

In this lab you will be learning how to work with UPPMAX. You will enqueue image analysis jobs and collect their results. More often than not, you will write image analysis routines in MATLAB. However, running 20 MATLAB jobs at the same time would require 20 MATLAB licences. The way around this is to use the MATLAB compiler. Below I'll explain how to compile your MATLAB script to a stand-alone executable, how to install that executable, and how to run it. We'll use as an example the script and data you can get here:
`http://www.cb.uu.se/~cris/ParallelImageAnalysis/lab1.zip`

The ZIP file contains a script `msrall.m` that repeatedly calls `measurethickness.m`, using each of the names of the image files in the subdirectory. On a normal computer you'd call this script from within MATLAB, and the images would be processed sequentially. In this example, each image takes a minute or two to be processed. We will enqueue each of these independent tasks as a job on the UPPMAX cluster Kalkyl. They will then, depending on available nodes, be processed in parallel.

Note that the measurement function does not return the results as an output variable, like you would normally do. Instead, the result of the computation is saved in a MAT file. The `msrall.m` script reads in all the MAT files and averages the results.

1 Compiling

To use MATLAB and the MATLAB Compiler on Kalkyl, you need to load the MATLAB module:

```
module load matlab
```

To run the MATLAB Compiler, you first need to configure it to use the correct system compiler. As Kalkyl uses Linux, and on Linux we want to use the default GCC compiler, we can skip this step. But on other machines, you would need to run

```
mbuild -setup
```

from within MATLAB.

To compile the function into a standalone executable, use the following command from the bash shell (you can also run it from within MATLAB):

```
mcc -m -R '-nojvm,-nodisplay' measurethickness.m
```

Note! We have only three licences for the MATLAB Compiler at the IT department. When you use the compiler, it is locked to you for half an hour. Three licences is many more than we need under normal circumstances, but for a lab like this, where 12 groups need to use it at the same time, we are a little short. Therefore I have included the result of compilation in the ZIP file. Try running the compiler, if you get a license error, assume the compilation worked and you obtained the files `measurethickness` and `run_measurethickness.sh` that I've given you.

2 Testing

Finally, you can run the function, outside of MATLAB, through the `run_measurethickness.sh` script. The first parameter is the root of the MATLAB distribution, as returned by the `matlabroot` command from within MATLAB. On Kalkyl, this is `/bubo/sw/apps/matlab/x86_64/7.8`, and is stored in the environment variable `MATLAB_BASE`. Additional arguments are passed to the script as input arguments. These are always passed as strings. If your function takes other things as input, you can use `eval` to convert the strings to arrays, cell arrays, etc. The function `isdeployed` can be used to make your function behave differently when compiled, for example by `eval`'ing the input arguments. To test our program, do:

```
./run_measurethickness.sh $MATLAB_BASE images/01_test.tif
```

This script sets some paths and then calles the program `measurethickness`, which is actually a small program and an archive with the M-files and MEX-files that compose your program. The MATLAB interpreter is used to execute the program, so there is no speed gain compared to running the script from within MATLAB.

The result of the computation is stored in a little MATLAB MAT-file. You can `load` this file in MATLAB to see if the result is reasonable.

3 Installing

If you want to run the program on a computer that does not have MATLAB installed, you should provide the MCR (MATLAB Compiler Runtime) installer. This can be found here:
`$MATLAB_BASE/toolbox/compiler/deploy/glnxa64/MCRInstaller.bin`
(substituting the proper MATLAB root directory). If you are using a different architecture than 64-bit Linux, the path component `glnxa64` (which stands for

GNU Linux AMD 64) should be replaced with the name for the corresponding architecture, for example `glnx86` for 32-bit Linux, or `maci64` for 64-bit Mac OS X. When using your compiled program with the MCR, you would give the installation path of the MCR to the `run_measurethickness.sh` script, instead of the MATLAB path. Kalkyl has MATLAB installed, so we don't need to worry about this step now.

However, there is one thing that we can do that would speed up execution of our program: unzip the archive! The file `measurethickness`, as I said earlier, is a small executable with a large ZIP file attached (MATLAB refers to this as the CTF archive). It can be extracted with the command

```
$MATLAB_BASE/toolbox/compiler/deploy/glnxa64/extractCTF measurethickness
```

This produces the directory `measurethickness_mcr/`, feel free to explore it. Inside you will see, for example, all the functions in the Image Processing toolbox that we used (and the functions on which these depend).

The next time we execute our program, the files already extracted will be used instead of the files inside the archive inside the executable.

4 Runnig

You already know how to run the program. What you now need to do is enqueue the program 20 times, with a different input argument, to process all the images.

The script `msrall.m` applies the measurement function to each of the images, and then collectes the measurement results and generates a plot. After you have enqueued our analysis program 20 times, and obtained 20 MAT-files, you can run the last part of this script to collect and plot the measurement results.

When starting matlab on kalkyl or your desktop computer at CBA, use the license file that I gave you in the `lab1.zip` file:

```
matlab -c license-classroom.dat
```

This license is allowed *only* for classroom use, and will avoid our problem with limited available licenses.

Important information

We have several nodes on Kalkyl reserved during each of the lab sessions. To use the reserved nodes (and not have to wait in the queue with all the people doing actual work), add `--reservation=g2011040` to your `sbatch` and `interactive` commands.

The project number for this course is `g2011040`. This goes after the `-A` option of the `sbatch` and `interactive` commands.