Workshop 8: Parallelising the Mandelbrot program with MPI

In the previous workshop we saw that the Mandelbrot calculation can suffer load balancing problems because the number of iterations at each point in the complex plane varies between loop iterations. With OpenMP we were able to improve the parallel performance by using dynamic loop scheduling. In this workshop we will test three different strategies for distributing the workload with MPI.

1 Getting set up

Isca is physically located in the university data centre so it is accessed remotely by connecting to a log in node using SSH. On the module ELE page there are instructions describing how to log in to a remote Linux computer under the "Workshop 2" section.

- Start by logging in to Isca as described in the Workshop 6 instructions.
- The example programs for this workshop are provided as a tar file. There are two versions of the tar file (one for ECM3446 and one for ECMM461) Take a copy of the tar file with the correct module code by running either

  ```
  cp /lustre/projects/Research_Project-IscaTraining/HPC/workshop8_ecm3446.tar .
  ```

  or

  ```
  cp /lustre/projects/Research_Project-IscaTraining/HPC/workshop8_ecmm461.tar .
  ```

  where the dot at the end of the command causes the file to be copied into the current working directory. When you first log in to Isca the current working directory is your home directory, which is mounted on the log in nodes and all the compute nodes. There should be sufficient space in your home directory for all the workshops on this module.
- Unpack the tar file by running

  ```
  tar xvf workshop8_ecm3446.tar
  ```

  or

  ```
  tar xvf workshop8_ecmm461.tar
  ```

  This will create a directory called `workshop8` which contains the example program for this workshop.

2 Running the single node scaling tests

In this section we will measure the single node parallel scaling of the three different versions of the program.

- Start by loading the Intel Cluster Toolkit module by running the command

  ```
  module load intel/2017b
  ```

- Compile the first MPI version of the program then submit the `run_mb.sh` job script which runs a scaling test on 2, 4, 8, 12 and 16 processors:

```
cd workshop8/mpi_v1
mpicc -o mandelbrot mandelbrot_mpi.c -lm
sbatch run_mb.sh
```

- The program will report the elapsed time in the SLURM output file. Look in the SLURM output file and check that the output looks similar to the following:

```
STATS (num procs, elapsed time): 2 63.205713
STATS (num procs, elapsed time): 4 54.623414
STATS (num procs, elapsed time): 8 31.188957
STATS (num procs, elapsed time): 12 26.462227
STATS (num procs, elapsed time): 16 20.342495
```

- We can extract the scaling data from the SLURM output using the `awk` command. Run the command

```
awk '/STATS/ {print $6, $7}' < slurm-123456.out
```

  where you should replace `slurm-123456.out` with the name of your SLURM output file. This command extracts lines from the file which contain the string `STATS` and prints the 6th and 7th items (whitespace separated) which are the number of MPI processes used and the elapsed time.

- We can also use `awk` to calculate the parallel speed-up. We measured a serial run time in Workshop 7 and we can use this as T_0 for our speed-up calculation. Using $T_0 = 90.985\,\mathrm{s}$ the following command prints three columns: number of MPI processes, elapsed time and speed-up:

```
awk '/STATS/ {print $6, $7, 90.985/$7}' < slurm-123456.out
```

  Remember that you can recall and edit the previous command using the up-arrow.

- To store the scaling data in a file run the following command

```
awk '/STATS/ {print $6, $7, 90.985/$7}' < slurm-123456.out > v1.dat
```

  which redirects the output to a file called `v1.dat` (overwriting the file if it already exists).

- Run the scaling test for version 2 of the program:

```
cd ../mpi_v2
mpicc -o mandelbrot mandelbrot_mpi_inter.c -lm
sbatch run_mb.sh
```

- When the job has completed process the output to extract the scaling data and store it in a file:

```
awk '/STATS/ {print $6, $7, 90.985/$7}' < slurm-123456.out > v2.dat
```

  where you should replace `slurm-123456.out` with the name of your SLURM output file for this job.

- Run the scaling test for version 3 of the program:

```
cd ../mpi_v3
mpicc -o mandelbrot mandelbrot_mpi_mw.c -lm
sbatch run_mb.sh
```

- When the job has completed process the output to extract the scaling data and store it in a file:

  ```
  awk '/STATS/ {print $6, $7, 90.985/$7}' < slurm-123456.out > v3.dat
  ```

  where you should replace `slurm-123456.out` with the name of your SLURM output file for this job.

3 Plotting the single node scaling results

Now that we have run the scaling tests we can plot the results to compare the parallel performance

- The `plots` directory contains `gnuplot` scripts for plotting the results from this workshop. Use the `plot_scaling` script to plot the single node scaling results

  ```
  cd ../plots
  gnuplot plot_scaling
  ```

This will generate a file called `scaling.png` which can be copied to your local computer for viewing using `scp` or `pscp` e.g.

```
scp abc123@login.isca.ex.ac.uk:/lustre/home/abc123/workshop8/plots/scaling.png .
```

where you should run the command on your local computer and replace `abc123` with your username on Isca.

Question 1: Which version of the program performs best and how does it distribute the workload?

Question 2: Which version of the program performs worst and how does it distribute the workload?

Question 3: Why is version 3 the slowest version with 2 processors?

4 2 node performance

In the last section we will measure the performance when 2 compute nodes and 32 processors are used.

- Run the 2 node job script for the version 1 program:

  ```
  cd ../mpi_v1
  sbatch run_mb_2nodes.sh
  ```

- When the program has run extract the scaling data and append it to the `v1.dat` file

  ```
  awk '/STATS/ {print $6, $7, 90.985/$7}' < slurm-506290.out >> v1.dat
  ```

  The `>>` redirection appends the output to the file instead of overwriting the file.

- Run the 2 node job script for version 2 and version 3 and append the output to `v2.dat` and `v3.dat`

- Now plot the scaling results with the 2 node results added using the script `plot_scaling_2node`

```
cd ../plots
gnuplot plot_scaling_2node
```

Question 4: Which version of the program performs best running on 2 nodes?

Question 5: How does the performance of the fastest MPI version compare with the fastest OpenMP version of program?