An MPI Exercise on Blue Waters

Aaron Weeden, Shodor
2015
http://tinyurl.com/acca-cs-mpi

If you have not already done An OpenMP Exercise on Blue Waters, it is recommended you do so first.

Also, if you have not already read the slides for MPI: Terminology and Examples, it is recommend you do so first.

The example used in this exercise uses many files. You may find it helpful to reference this flowchart to see how they all interact.

For this exercise, you will need two terminal windows connected to Blue Waters. To connect to Blue Waters, follow the instructions in the *Logging In* section of A Blue Waters Usage Guide. Then, open a second window and repeat the process. On Windows, you can open a second PuTTY window by simply launching the program a second time. On Mac, you can open a second window in Terminal by typing **Command-N**.

Request interactive job

In the first window, request an interactive job for 1 hour using 2 nodes with 32 cores each. This can be accomplished using the following command:

```
qsub -I -l nodes=2:ppn=32:xe -l walltime=01:00:00<ENTER>
```

This is slightly different from what we did in the OpenMP exercise. Here, we are asking for 2 nodes instead of 1, because MPI allows us to use more than 1 node in a parallel job.

Request batch job

In the second window, request a batch job that will run the MPI version of the forest fire model five times each for total process counts between 1 and 64 on two nodes that each have 32 cores. The example script will also generate an output file with each process count and the average wall time it takes to run with that many processes:

With tabs:	`qsub ~awee<TAB>f<TAB>/m<TAB>/s<TAB>p<TAB><ENTER>`
Without tabs:	`qsub ~aweeden/fire/mpi/scale-mpi.pbs<ENTER>`

This will give back a job ID. Check the status of the job using this command:

```
qstat <job id><ENTER>
```

You should notice that the status of the job (second-to-last column) is **Q**, i.e. it is waiting in the queue. You can run the same **qstat** command from time to time and notice the status change from **Q** to **R** (running) to **C** (complete).

Update example code

While you wait for the job to finish, in the second window, update the example fire model code in your home directory (you should already have a directory called **fire** from doing the OpenMP exercise). **NOTE**: doing this update will replace the contents of the **mpi** folder. If you have worked with this directory on your own since the last exercise, make sure to make a backup of your files. After you have made any needed backups and are ready to replace the contents, the code can be updated using the following command:

With tabs:	**cp ~awee<TAB>f<TAB>/m<TAB>/* ~/f<TAB>m<TAB><ENTER>**
Without tabs:	**cp ~aweeden/fire/mpi/* ~/fire/mpi<ENTER>**

Change into the **fire** directory and into the **mpi** directory it contains:

```
cd ~/fire/mpi<ENTER>
```

Confirm you are now in the **mpi** directory:

```
pwd<ENTER>
```

You should get back **/u/training/<your username>/fire/mpi**

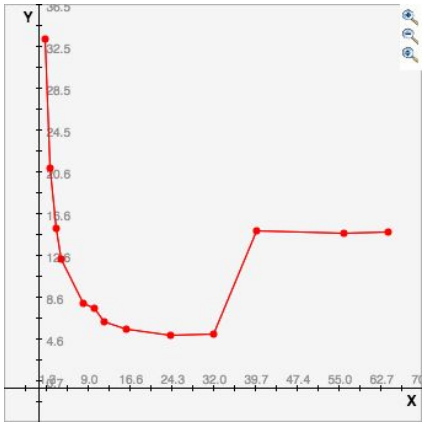
Create strong scaling plots

Inside the **mpi** directory is a sample file that is the result of running a batch job like the one you submitted earlier. Open this file in the vi text editor:

```
vi scale-mpi.out<ENTER>
```

The contents of the file are pairs of data: the number of MPI processes used and the average wall time (in seconds) to run the program with that many processes. Highlight the contents of the file using your mouse/trackpad. On Windows, this causes PuTTY to copy what you have selected. On Mac, press **Command-C** to copy.

Go to this website in a web browser: http://shodor.org/interactivate/activities/SimplePlot/. In the text box below the word **Data**, type the word **redgraph** followed by a new line, and paste. Click the button that says **Plot/Update**. You should get a graph that looks like the following, with the the x-axis measuring process counts and the y-axis measuring average wall time in seconds:



In the **Data** box, enter a new line and type the word **bluegraph** followed by a new line. Keep this browser page open; you will add more data in the instructions below.

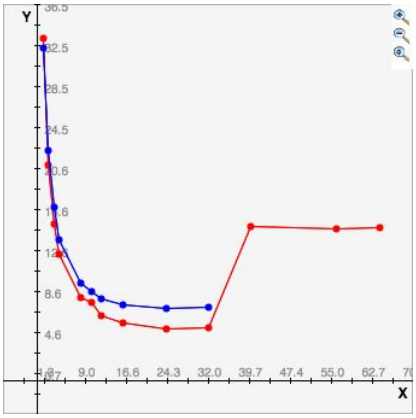
Close the file in vi by typing the following:

```
:q!<ENTER>
```

Open a file we used in a previous exercise with data from scaling the OpenMP version:

With tabs:	`vi ~awee<TAB>f<TAB>/o<TAB>s<TAB>o<TAB><ENTER>`
Without tabs:	`vi ~aweeden/fire/omp/scale-omp.out<ENTER>`

Copy/paste the data from this file into the **Data** box on the Simple Plot browser page. Click the **Plot/Update** button. You should now get a graph that looks like the following, with the MPI data in red and the OpenMP data in blue:



Notice how the MPI version (red) is slightly faster than the OpenMP version for all core counts, but the graph still levels out at the same core counts. Once we get beyond 32 processes, we have gone beyond 1 node, and the MPI version actually slows down because of the amount of time it takes for the 2 nodes to communicate with each other over the network (network speeds between nodes are much slower than bus speeds between cores).

Close the file in vi by typing the following:

```
:q!<ENTER>
```

Run interactively

These steps should be followed in the first window, the window in which you requested an interactive job.

Change directories to the example MPI code directory:

```
cd ~/fire/mpi<ENTER>
```

Decide how many MPI processes you want to use when running the fire model. There are 32 cores on each of the 2 nodes you requested, so the number of processes should be between 1 and 64, inclusive. You can use higher numbers than 64, but then there will be more than 1 process per core, which will probably hurt performance. Run the program using the command below:

```
time aprun -n <number of processes> ./fire-mpi -r 1300 -c 1300 -t 1300<ENTER>
```

This will run for a few seconds (up to around half a minute) and print the final total percentage of trees that burned (100%) in a forest with 1300 rows, columns, and time steps. It will also print the **`real`**, **`user`**, and **`sys`** time. The **`real`** time is the wall clock time and is more reliable for measuring performance than the **`user`** and **`sys`** times.

Run again using **`time aprun`** with a new number of processes after the **`-n`** (remember you can use the up-arrow and down-arrow keys to scroll through your command history, change commands, and enter them again). What is the new run time?

If you keep changing the number of processes, do you get data similar to the data you saw in the **`scale-mpi.out`** file?

The script that was used to generate the **`scale-mpi.out`** file can be run from an interactive job. Begin running the script by entering the command below:

```
./scale-mpi.sh<ENTER>
```

This will first ask you whether you want to erase the **scale-mpi.out** file. Enter **y** to say yes. The script will continue, printing out information about each run of the program: how many processes were used, and how much wall clock time elapsed.

After the script has run long enough for you to get an idea of what it is doing, terminate it early by pressing **Control-C**. You should get back your command prompt.

Open the script in vi:

```
vi scale-mpi.sh<ENTER>
```

The third line of this file sets the **RANGE** of process counts to use when running the script (starting at 1, going up to 64). Enter vi's insert mode by pressing the **i** key. Change the value of the range to be **(32 40 56 64)**. That is, delete all the numbers from 1 through 24.

The fifth line of the file sets the **PROBLEM_SIZE** of the model; this is the number of rows, columns, and time steps in the model. Change the problem size from **1300** to **4000**.

The seventh line of the file sets **NUM_TRIALS**, which is the number of times the script will run the **time aprun** command. Change this number from **5** to **1**.

Once you have made these changes, press **Esc** to exit vi's insert mode and enter command mode. Then, save your changes and quit the file by typing the following:

```
:wq<ENTER>
```

Run the script by entering the command below:

```
./scale-mpi.sh<ENTER>
```

If it asks you to, enter **y** to erase the **scale-mpi.out** file.
The script will print out a run time for 32, 40, 48, and 56 processes. The script will take a few minutes to complete. While you are waiting, move ahead to the next section.

Check the batch job's output

Check on the status of your jobs:

```
qstat -u <your username><ENTER>
```

The batch job will be named **scale-mpi.pbs**. If that job has a status of **C** (complete) or does not appear in the list, then follow the steps below. Otherwise, if the job is still in the list with a status of **Q** (in the queue) or **R** (running) come back to these steps later once the job is complete.

Change back to your home directory:

```
cd<ENTER>
```

Confirm the output and error files from the job now appear in the list of files:

```
ls<ENTER>
```

You should see files in the list with **scale-mpi.pbs.o<job ID>** (this is the output file) and **scale-mpi.pbs.e<job ID>** (this is the error file).

You should also see a file called **scale-mpi.out**. Show the contents of the file:

```
cat scale-mpi.out<ENTER>
```

Confirm that the contents of the file look very similar to the **scale-mpi.out** file you opened earlier.

Run another batch job

Change directories to the example code directory:

```
cd ~/fire/mpi<ENTER>
```

Open the file **fire-mpi.pbs** in vi:

```
vi fire-mpi.pbs<ENTER>
```

This file is a script that requests a batch job that runs the executable file. Initially it is set to run with 32 processes. You can change this number to a different number of processes by replacing the number 32 after **-n** with the number you choose. You should also update the **nodes=** and **ppn=** values to have enough nodes and cores to run the process count you picked (node count times core count should be larger than process count). You will also want to change the expected wall time to be larger than the average wall time for the number of processes you chose that you saw when you created the strong scaling plot. The format for the walltime is **HH:MM:SS**.

Once you have made the changes, saved and quit the file, you can request a new batch job by running the following command:

```
qsub fire-mpi.pbs<ENTER>
```

This will give back the job ID.

You can continue to monitor the status of the job using the following command:

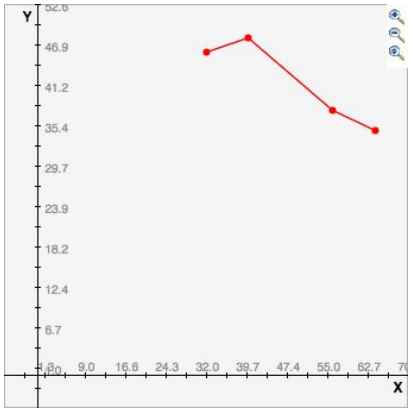
```
qstat <job ID><ENTER>
```

Check back in on the interactive job

In the interactive job (the first window), when the **`scale-mpi.sh`** script finishes, it creates a new version of the **`scale-mpi.out`** file. Display the contents of this file:

```
cat scale-mpi.out<ENTER>
```

Graph this data in Simple Plot. You should get something similar to the following:



In this graph, when the number of cores increases beyond 32, there is an initial slowdown (which we saw in previous graphs), but then a further speedup (which we did not see before). What this shows us is that even if *no* further speedup occurs beyond 1 node for a certain problem size (e.g. **`1300`**), there may be further speedup beyond 1 node for a larger problem size (e.g. **`4000`**).

Once you are finished using the interactive job, you can end it by typing **Control-D** or the following command:

```
exit<ENTER>
```

Compile the program

Change directories to the example code directory:

```
cd ~/fire/mpi<ENTER>
```

Remove the MPI executable file using the Makefile in that directory:

```
make clean<ENTER>
```

Confirm there is no longer a file called **`fire-mpi`** in the directory:

```
ls<ENTER>
```

Create the executable file again using the Makefile:

```
make<ENTER>
```

This may take a minute or so.

Confirm the new **fire-mpi** file is now in the directory:

```
ls<ENTER>
```

Remove the MPI executable file without using the Makefile:

```
rm fire-mpi<ENTER>
```

Confirm the **fire-mpi** file is no longer there:

```
ls<ENTER>
```

Create the executable file without using the Makefile:

```
cc -o fire-mpi fire-mpi.c<ENTER>
```

This may take a minute or so.

Confirm the new **fire-mpi** file is in the directory:

```
ls<ENTER>
```

Finish up with the batch job

Once the batch job is complete, open the output file it produced:

```
vi fire-mpi.pbs.o<job ID><ENTER>
```

After the “prologue” it will tell you what percentage of the trees were burned. This should be 100%.

Close the output file without saving changes:

```
:q!<ENTER>
```

Open the error file for the job:

```
vi fire-mpi.pbs.e<job ID><ENTER>
```

The **time** command writes to the error stream by default, so the **real**, **user**, and **sys** time will be shown in this error file. Compare the **real** time to the time you saw earlier in the **scale-mpi.out** file for the corresponding number of processes.

This gives you one data point (or ⅕ of one if you are going to take an average of 5 runs for each data point). See if you can do multiple runs with different numbers of processes and generate data points like those found in the **scale-mpi.out** file without using the automated **scale-mpi.pbs** or **scale-mpi.sh** scripts to create this file.

Additional work

As in the OpenMP version, an ASCII visualization can be generated using the **-o** option. This will not be covered in this exercise. If you would like to do this, see An OpenMP Exercise on Blue Waters for instructions, replacing the executable name **fire-omp** with **fire-mpi**.

Either in an interactive or batch job, you can explore running the **fire-mpi** executable with different parameters as explained in A Blue Waters Usage Guide**.**

If you wish to view or change the source code of the model, it is available in the file **~/fire/mpi/fire-mpi.c**. If you wish to create a new executable based on the changes you made, enter the **make** command again from within the **mpi** directory.

To see how the MPI version of the code compares to the serial version, download this file and open it in a web browser. The serial version is shown on the left, and the MPI version is shown on the right, with the differences highlighted.