CS 4444/6444 Spring 2016 – Due in class February 21

Assignment 2: High Throughput Computing

Introduction

In this assignment, you will write a shell script that will take a sequential program and have it run on many computers, but with different data on each node. You will explore the performance ramifications of converting a sequential program into a high-throughput implementation using the queuing system.

An animated film requires a huge amount of processing power to render the final movie. Consider the movie Final Fantasy (a box office flop, but amazing computer graphics) - if rendered on a single (modern) computer, it would take over 2,500 years to complete. Instead, they used a parallel cluster of 1,200 computers, and were able to complete the rendering in about 2.5 years (this was back in 2000). While this movie is a bit dated (it came out a year later in 2001), modern movies need similar amounts of computation - although our computers have gotten faster, movie makers want to put more graphics into their movies, effectively offsetting the gain in processor speed.

To render an animated movie (or a movie heavy with computer graphics), each separate server in the cluster renders a single frame at a time. Movies run at 30 frames per second - so a 2 hour movie contains over 200,000 frames. And each frame can take hours - or even days - to render.

For this assignment, we are going to render a (much smaller) movie clip, and without sound. Our movie is about 250 frames (at 25 frames per second, it's a 10 second clip). It's a not-very-authentic clip of a star (or planet?) collapsing, but it looks neat, which is why it was chosen. The low-resolution version takes an average of 6 seconds per frame to render - which means it would take about 25 minutes to render it on a single node. The multiple nodes that we are using will render separate individual frames, which will be combined when they are all completed.

The movie used in this assignment was adapted from http://www.centralsource.com/blender/explosions/ and was created by Dave Parsons. The only modifications were made for the movie size, output type, etc - no modifications were made to the animation.

This assignment starts with a bit of background as to how to generate the movie frames, and then how to combine them into a single movie file.

Files needed

To start, download from the Collab site the following files.

- Star-collapse-ntsc.blend: This is the file that contains the movie. It's a Blender file. This is the low-resolution version, set up for a standard (low-definition) NTSC television set (720x480).
- Star-collapse.avi: this is the rendered movie file that you will be creating (it's about 9 Mb in size). Your movie will look slightly different - see below under 'Making the movie'.
- star-collapse-0001.jpg: This is a render of the first frame of the image, which we'll need shortly. It's also shown below.
- slurm.sh: A sample SLURM shell script that you may use as a starter. As it is it can compute and build the whole movie. It should help you understand the problem.

Blender

First, we'll go over how to render frames of the movie. We will be using Blender, an animation program similar to 3D Studio Max and Maya. Blender has a very complicated GUI interface but we'll only be using it through the command line. To render a single frame, enter the following commands in the directory where you downloaded the Blender program. This tells blender to render frame 1 of the movie.

```
./blender -b <file> -s 1 -e 1 -a
```

If you specify the Star-collapse-ntsc.blend file for <file> in the above command, the image rendered should look like star-collapse-0001.jpg.

The -b option tells blender to use that particular file (specify the path/filename of the file instead of "<file>" - you should be using Star-collapse-ntsc.blend for now). The -s option tells blender which frame to start at (frame 1), and the -e option tells blender what frame to end at (again, frame 1) - thus, we are only generating a single frame (frames start from frame 1, not frame 0). The -a is needed also to tell blender that you want to render one or more files via the command line. Note that the order of the parameters matters! If you put the -a before the '-s 1 -e 1', it will render the entire movie instead of one frame.

An important note is that the only difference needed in the command line to generate different frames is just the integer parameters given after both the -s and -e flags. This will be needed later.

The result of running that command is single image, which Blender took a few seconds to create. The file is named star-collapse-0001. It's a jpeg image, even though there is no .jpg extension. One way to view the image is to copy it to a Windows/Mac machine. Don't forget to add the .jpg extension.

To create the entire movie, you could create it on a single node:

```
blender -b <file> -a
```

This would take about 25 minutes or so to complete. The other option, which is what we are doing in this lab, is to create the frames on multiple nodes. Either way, once completed, you will have 250 files in your directory, named star-collapse-0001 to star-collapse-0250 (if you renamed

star-collapse-0001 to star-collapse-0001.jpg, you should rename it back). You can enter 'ls star-collapse-* | wc -l' to find out how many frames are currently rendered - this will be useful as the frames are being rendered by your parallel job.

Making the movie

Once all the individual frames are rendered, the final task to make the movie is to combine the 250 image frames into a single animation. You may want to come back to this section once you have generated the files.

ffmpeg4 is the program we are using to "stitch" together the individual frames into a single movie file. To make the final movie, make sure you are in the same directory as your 250 frame files.

```
ffmpeg -framerate 25 -start_number 1 -i star-collapse-%04d.jpg -vcodec
mpeg4 output.avi
```

The command should all be on one line, even though the version above was wrapped to a second line. There are a number of options on this command line, if you are interested:

This command will take 5-10 seconds or so to run. When it's done, you should have a 9 Mb movie file. Again once you copy the file to a Windows/Mac machine, you can double click to play it. Note that the Star-collapse.avi move on Collab is the same as the NTSC version of the movie you are generating.

The particular blender file provided was created with an older version of Blender. Newer versions have changed the particle effect system, which is what produces the smoke effects, and some of the things flying away from the decaying star. Thus, your movie might look slightly different than the one we provide.

The task

The task for this assignment is to write a shell script or C program that will start off a series of separate blender jobs each of which will render a number of the frames. Each process is a sequential execution of blender, but the fact that there are multiple running at the same time (and each one is rendering different frames) yields the parallel performance for this assignment. The frames will be combined once they are all completed.

One way to create the parallel task would be to launch 250 separate blender jobs, each one computing a different frame. While this will certainly work for this assignment, it does not scale to larger applications. A full length movie may well have 200,000 frames - creating that many jobs would probably just crash the system. You are to explore the performance effects of using different numbers of nodes to generate the frames. Explore the performance differences of using 1, 10, 20, 40, and 80 processors. For you performance analysis, present the walltime from start to finish of your program. You may want to also compare walltime with CPU time. Discuss the type of speedup you get.

To begin, start on a single job shell script that generates a few of the frames (say, the first 5), and make sure you have that syntax correct. Then work on the other, "master", shell script that generates all of the job shell scripts. For help on that shell script, see slurm.sh. This is only one way to write your script. You do not have to follow this example.

Remember that you will have to add execute permission to your "master" shell script using chmod as the file will not be executable by default.

Optimizations

What parallel programming project would be complete without listing various optimizations?

There are many ways one can optimize the movie file created in Blender. We won't be dealing with much of that, though - and we are assuming that the blender executable and the movie file can't be optimized any further. Actually, the blender program is a parallel program - it can use many CPUs (and cores!) at once - but not for the movie clip we are using in this assignment.

Different frames in the movie take different amount of time to compute. As more things are going on in the graphical scene, it takes more time to render the image. Particle effects (the smoke trail behind the meteors), the shockwave, the flare effects - all of these take a lot more time to compute. Indeed, the first frame (of the NTSC version) takes 2.6 seconds to render on dogwood, whereas the last frame (frame 250) takes over 8 seconds to render.

The other thing to notice is that all the frames at the beginning of the movie are much quicker to render than the frames at the end of the movie (not much is going on at the beginning of the movie). Thus, if each of your blender processes does a sequential set of 32 frames (which is about 1/8th of the 250 frames), then the process that is doing frames 1-32 will finish long before the process that renders the last 32 frames. In our testing for this homework, the process that rendered the first 32 frames took 1 minute 17 seconds to complete, whereas the process that rendered the last 32 frames took 4 minutes 20 seconds to complete.

One way to optimize this is to have a single job shell script render every 8th frame (the first one does frames 1, 9, 17, 25, etc.) - this would balance the load a bit better. There will probably be a performance hit from invoking blender multiple times (as opposed to the previous version, which only invoked it once), but that would be more than compensated for by the better load balancing. You could play around with how you distributed the frame load to get better performance. Be sure to describe whatever partitioning method you choose in your write-up.

Other Notes

Blender may display warnings ('using old particle system', 'cannot find library', etc.) - generally, these are warnings, and not error messages. However, a 'camera not found' message means that the image was not created, perhaps due to not finding the file, or not having enough disk quota space left.

Slurm.sh

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=1096
#SBATCH --time=00:15:00
#SBATCH --part=training
#SBATCH --account=parallelcomputing
#SBATCH --output=script-output

# the number of frames to be used to generate the video
frame_count=$1

# load the renderer engine that will generate frames for the video
module load blender

# -b option is for command line access, i.e., without an output
console
# -s option is the starting frame
# -e option is the ending frame
# -a option indicates that all frames should be rendered
blender -b Star-collapse-ntsc.blend -s 1 -e $frame_count -a

# need to give the generated frames some extension; otherwise the
video encoder will not work
ls star-collapse-* | xargs -I % mv % %.jpg

# load the video encoder engine
module load ffmpeg

# start number should be 1 as by default the encoder starts looking
from file ending with 0
# frame rate and start number options are set before the input files
are specified so that the
# configuration is applied for all files going to the output
ffmpeg -framerate 25 -start_number 1 -i star-collapse-%04d.jpg -vcodec
mpeg4 output.avi
```