Name: Ashwin Raghav Mohan Ganesh
UVA Id: am2qa

Assignment 2: High Throughput Computing

1.0 Problem Statement

The objective is so parallelize a set of blender rendering commands across a grid and collate them so that mencoder can be given the sequence of frames that can be rendered into the video.

2.0 Hardware and Software

Centurion001
CPU: AMD Opteron(tm) Processor 242
Main Memory 2GB RAM per node
OS :Ubuntu 10.04.3

3.0 General Info

The code for this assignment can be viewed in stages at *https://github.com/ashwinraghav/Assignments/commits/master*

3.1 Measurement of Wall time

The Wall time that I have measured and reported here is inclusive of:
1) Generation of the scripts from the master script
2) Queuing the job
3) Waiting time in Queue
4) Execution time
5) Time to transfer the results back onto the user's namespace.

This is a significantly longer than the wall time reported by PBS which is just a sum of 2 & 3. However, this is a fair measure of the wall time since it is the actual time elapsed from the beginning till the required output is obtained.

Note that the mencoder command execution is not part of the measurement since it takes only a constant amount of time once the input files are provided.

3.2 Comparing Graphs

It is important to note that the comparing 2 equivalent strategies on different graphs may not always be consistent since they were run at different times of the day at different traffic levels. However, I have made sure that all values in a given graph have been obtained in within an hour among themselves. Hence, values within the same graph are relative. But values between different graphs may not be relative.

4.0 Optimizations

Although most queing system suggest that specifying a fairly accurate estimate of the wall-time in the pbs script will help schedule the job efficiently, in most real-time cases, the running time of a job cannot be predicted accurately.

From that front, it makes sense to observe how different values factors relate to each other and identify the optimal greedy-strategy for a job.

4.1 Reducing the chatty nature by moving the binaries

Since the blender binaries are invoked 100s of times across jobs, each time it may be involve copying the binaries across the NFS. I was unsure whether there was a caching strategy in place across the network. Hence, as a first step, in each of the jobs, I copied the binary onto the local $TMPDIR and then invoked blender from the tmp dir.
Straight away There was a significant improvement in performance.

```
cd $PBS_O_WORKDIR
cp /usr/bin/blender $TMPDIR
$TMPDIR/blender -b Star-collapse-ntsc.blend -s 8 -e 8 -a
$TMPDIR/blender -b Star-collapse-ntsc.blend -s 16 -e 16 -a
```

4.11

No of Jobs	No of nodes	PPN	Dist Strategy	Time
8	2 per proc	1	Linear	181
8	2 per proc	1	Linear	146

4.20 Fixing the number of parallel tasks:

In line with intuition, this graph shows that simply increasing the number of parallel jobs will not always yield the optimal result since the waiting time in the queue will increase for the jobs that join last. In this case for eg, at around 100 parallel tasks, a fastest running time of 77 seconds was experienced

4.21

Factor	Value
Nodes	1
PPN	1
No of frames per invocation	1 frame i.e 250 invocations

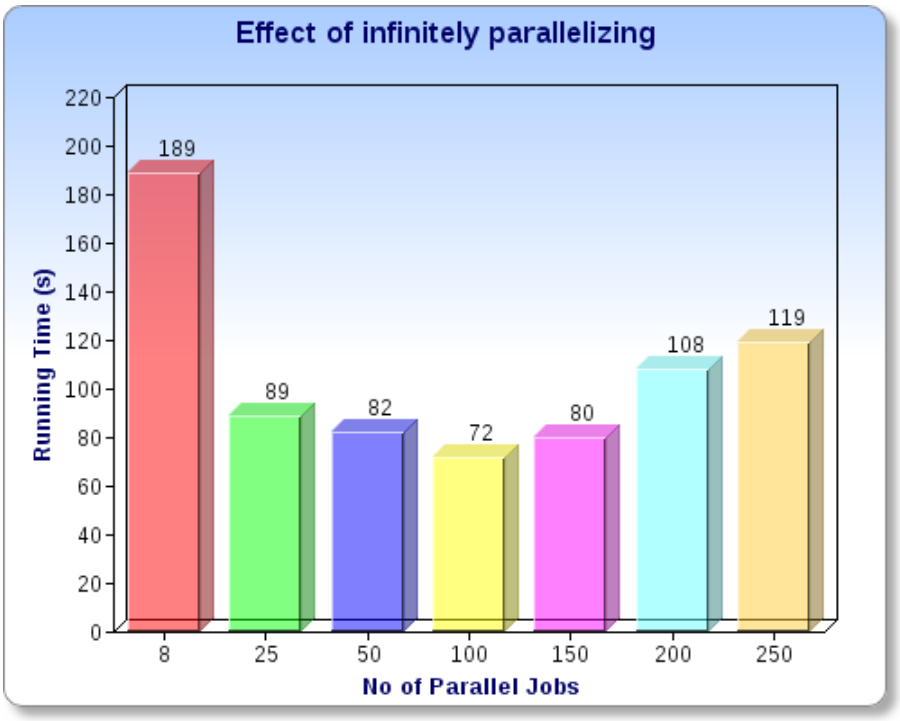


Fig 4.22 Parallel Jobs Vs Execution Time

4.23 Inference

For each type of job to be run, there is bound to be an greedy- yet optimal number of jobs. Blindly increasing the number of parallel tasks can slow down the system. This is because each time a job completes, the system evaluates the next best contender for the slot and infinite jobs simply means infinite contentions. One job means one contention but no parallelization. Hence a classical trade-off Engineering problem

---

4.30 Blender invocation Count

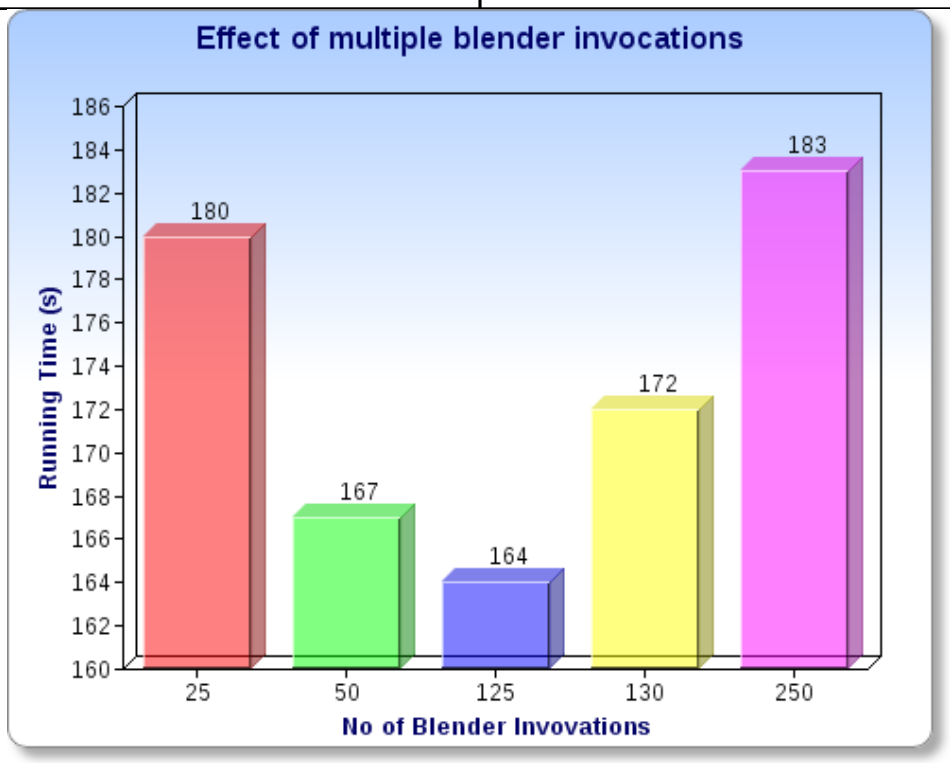
This was done by varying the number of frames generated per blender invocation. For eg: 50 invocations if 5 tasks have 10 invocations generating 25 frames each

```
INVOCATIONS=125 #must be a factor of 250
for (( i = 0; i < $INVOCATIONS; i++ )); do
  PBS_FILE="job-$(($i % $NODES + 1)).pbs"
  START_FRAME = $((($i* $(($FRAMES/$INVOCATIONS))) + 1))
  END_FRAME = $((($i+1)* $(($FRAMES/$INVOCATIONS)))))
  echo "\$TMPDIR/blender -b \$TMPDIR/Star-collapse-ntsc.blend -s
$START_FRAME -e $END_FRAME -a" >> $PBS_FILE
done
```

4.31

Factor	Value
Nodes	1

PPN	1
No of parallel jobs	8



4.32 Number of Blender Invocations vs Execution Time

4.33 Inference

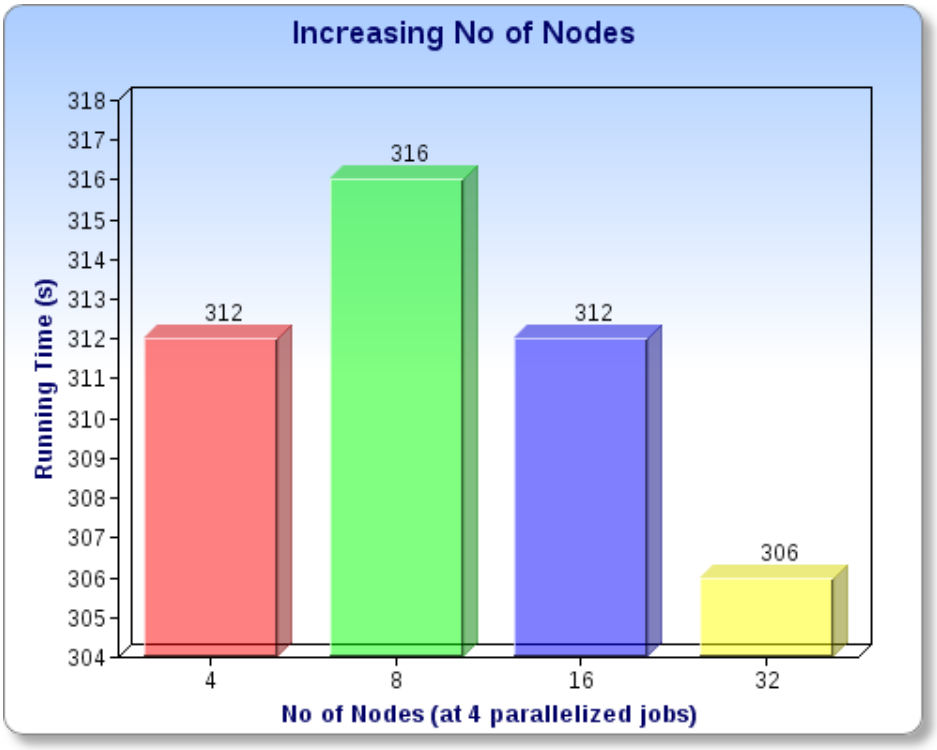
The performance hit resulting from invoking blender multiple times is more than compensated for by the better load balancing we have across the jobs. However this is clearly not a random factor manifesting. It is observed that at 125 blender invocations i.e reducing the number of invocations, we git the sweet-spot. Any lower/higher, the 2 factors of load-balancing and blender-invocations--one outweighs the other.

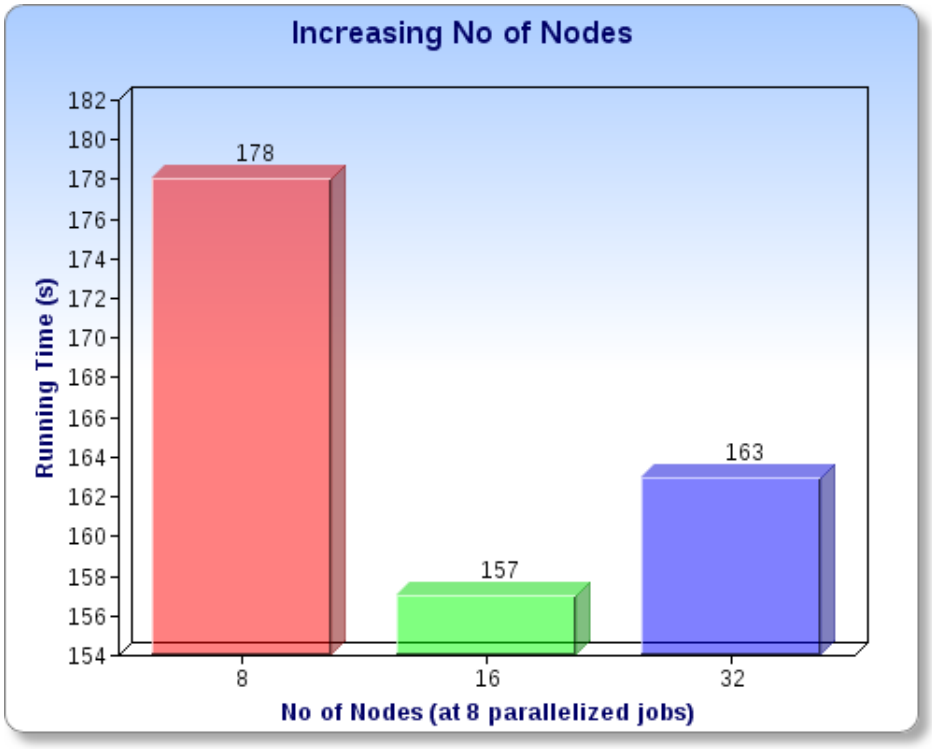
4.40 No of Nodes

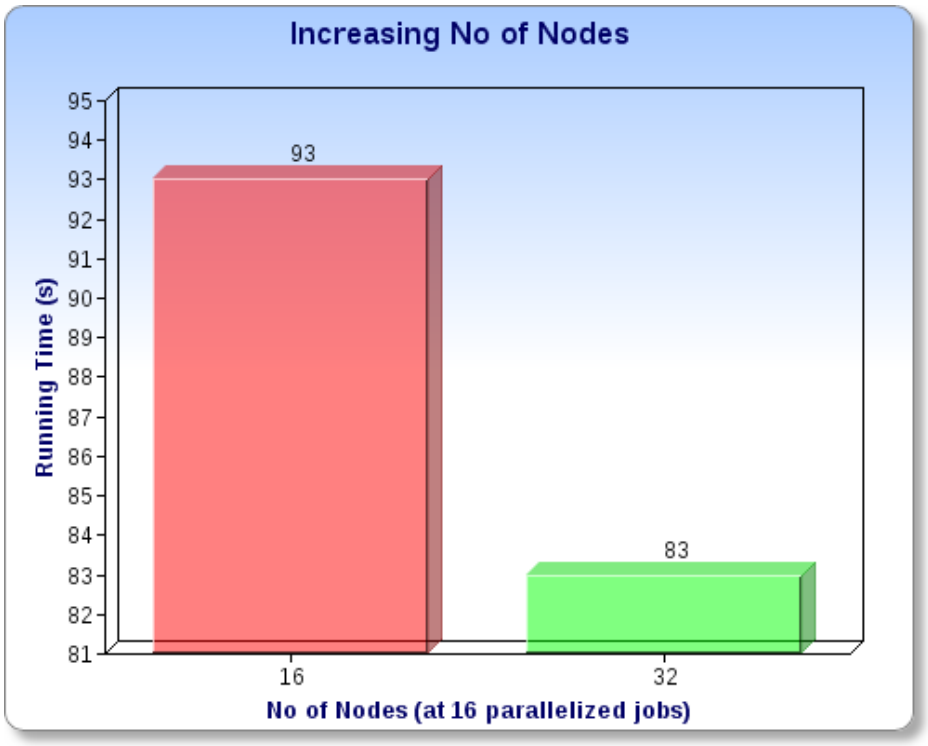
The number of nodes that are required by a job has an effect on the wall time in a non-linear way. Hence I have measured the effect had by the number of nodes on a job with varying number of jobs.

4.41

Jobs	No of Blender invocations	PPN	Figure
4	125	1	1
8	125	1	2
16	125	1	3







4.42 Inference

Once again, blindly increasing the number of nodes (independent of the number of parallel processes) does not linearly decrease running time.

This is especially application when a large number of nodes are spread across a small number of processes as in Fig1. The speedup obtained as a result of the increased nodes is lost as a result of the time that a process has to wait in the queue waiting till the required number of nodes can be allocated in one go. Hence at each job-count config, there is a sweet spot for the ideal number of nodes to be used.

5.0 Failed Optimizations

5.1 NP-Hard Partition Problem and Videos

A good work distribution strategy among the parallel jobs is essential to maximize resource utilization and reduce wall time. So one of the toughest (and most time consuming) strategies that I have tried is to first solve the partition problem.

5.2 Spatial similarity of Frames

One assumption I made in this stage was that since we are essentially talking about a video file of length 250 frames (at 25 frames per second), a sample run of every 10th frame would be indicative of how much time the frames around the 10th frame took.

Once this sample was collected, it would leave us with a set of 250 (approximated) numbers that would need to be partitioned into n equal sets (jobs). This is an NP-Hard problem that can be solved recursively by dynamic programming. In the test run, I used an online tool to give me the approximate partitions and scheduled the jobs appropriately.

However, I found that this barely improved the total running time of the jobs. I inferred that solving an NP hard problem before running the rendering jobs would not yield any better wall-time.

5.3 Proof of Rationality for failure

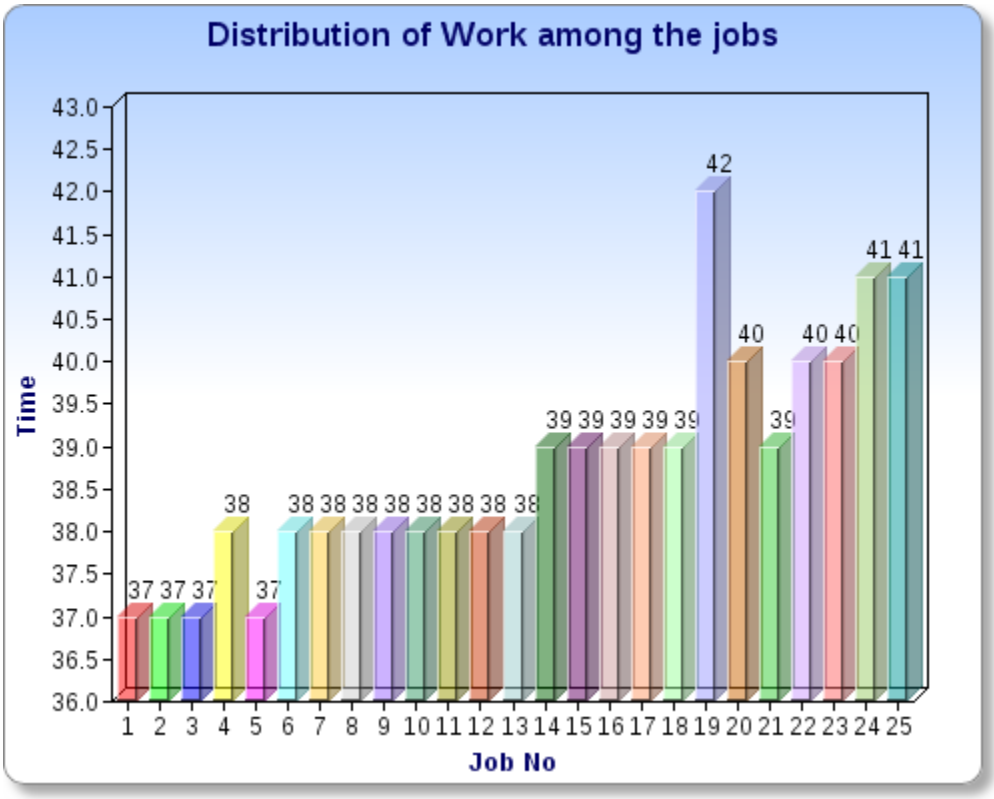
As mentioned above, the time to render spatially proximal frames is approximately the same. This is an inherent property of most parts of a video files (except when a scene changes). Hence even a simple strategy of linearly distributing the frame generations itself sufficiently helps appease the problem of resource hogging by a job or the converse of it being under-utilized at any point of time. It is pretty evident that the jobs are fairly evenly time consuming which is indicative of good resource utilization.

As a proof, I have shown below the individual times taken by jobs that adopt this simple strategy.

5.31

Factor	Value

No of parallel jobs	25
Job Dist Strategy	1,9,17... in job1 | 2,10,18.. in job 2 | 3,11.. in job 3 etc



5.32 Time Distribution across 250 frames

5.33 Inferences

As shown, all 25 jobs fairly consume the same amount of time.

The above chart was also generated for 60 parallel jobs at around 4 blender commands each. In this case however, it was observed that there was a larger variance in the time take by each job.The jobs ran between 4-8 seconds. Clearly this was because 250/60 yields an unequal number of commands to each job.

However, even at this 60 job config, the cumulative running time only improved.

The reason being that at any given point all resources were occupied and as each job completed, another job came in to take its place.

And in the last batch of jobs, the likelihood of there being an unequal number of jobs is low since we are distributing jobs in a linear way and these jobs are queued in a linear manner.

5.4 Alternating Corners distribution strategy

As an alternate to the straight linear way the frame ordering was distributed, There is an alternate that can be performed that will ensure an equal distribution.

This strategy will order the frames alternatively at the corners of the 'window' on either side and move inwards towards the centre and outwards in the opposite direction.

```
for (( i = 1; i <= 250; i++ )); do
  if ((($i%2) == 0));then
    PBS_FILE="job-$(($X+1)).pbs";
    X=$((($X+1)%$NODES));
  else
    PBS_FILE="job-$(($NODES-$Y)).pbs";
    Y=$((($Y+1)%$NODES));
  fi
 echo "\$TMPDIR/blender -b \$TMPDIR/Star-collapse-ntsc.blend -s $i -e
$i -a" >> $PBS_FILE
done
```

6.0 Conclusion -- Final best readings

I have provided the readings at 8 jobs (in the interest of practiality) and at 70 jobs where the best performance was experienced

No of Jobs	No of nodes	PPN	Dist Strategy	Time(s)
8	16	2	Alternate corners	103
70	16	2	Alternate Corners	68

I have also provided a printed copy of the code.

```
#created by ASHWIN RAGHAV MOHAN GANESH
#am2qa

if [ "$PBS_ENVIRONMENT" != "" ] ; then
  echo Ack!  Launched via a qsub -- not continuing
  exit
fi

#clean up previously created files
NODES=8

#Start the timer
START=$(date +%s)

FRAME_COUNT=250
rm -rf output-*
rm star-*
rm job-*

for (( i = 1; i <= $NODES; i++ )); do
   PBS_FILE="job-$i.pbs"
   `touch $PBS_FILE`
   echo "#!/bin/sh" >  $PBS_FILE
   echo "#PBS -l nodes=1:ppn=1" >>  $PBS_FILE

   #The walltime is inaccurately estimated
   echo "#PBS -l walltime=12:00:00" >>  $PBS_FILE

   echo "#PBS -o output.txt" >>  $PBS_FILE
   echo "#PBS -j oe" >>  $PBS_FILE
   echo "#PBS -m ea" >>  $PBS_FILE
   echo "#PBS -M am2qa@virginia.edu" >>  $PBS_FILE

   echo "cd \$PBS_O_WORKDIR" >>  $PBS_FILE

   #Copy the blender binaries and the .blend file into the node to
remove chat acros the network
   echo "cp /usr/bin/blender \$TMPDIR" >> $PBS_FILE
   echo "cp Star-collapse-ntsc.blend \$TMPDIR" >> $PBS_FILE
done

X=0 # X is the front index
Y=0 # Y is the rear index

for (( i = 1; i <= $FRAME_COUNT; i++ )); do
```

```
   #alternating put the frames at the beginning and end of the
shrinking window
   if (($i%2) == 0));then
     #put the frame in the beginning
     PBS_FILE="job-$(($X+1)).pbs";
     X=$((($X+1)%$NODES));
   else
     #put the frame to the end
     PBS_FILE="job-$(($NODES-$Y)).pbs";
     Y=$((($Y+1)%$NODES));
   fi

   #one frame generated per blender invocation
   #node that the invovation is not from /usr/bin but from the $TMPDIR
directory
   echo "\$TMPDIR/blender -b \$TMPDIR/Star-collapse-ntsc.blend -s $i -
e $i -a" >> $PBS_FILE
done

#queue up all files in order
for (( i = 1; i <= $NODES; i++ )); do
   PBS_FILE="job-$i.pbs"
   qsub $PBS_FILE
done


#wait till all files are on the local file-system
while true; do
   DONE_COUNT=`ls star-collapse-* 2> NUL |wc -l`
   if (($DONE_COUNT = $FRAME_COUNT)); then echo "DONE!!"; break; fi
done


#Measure the total time and print
END=$(date +%s)
echo "It took $(($END - $START)) seconds wall time"
```