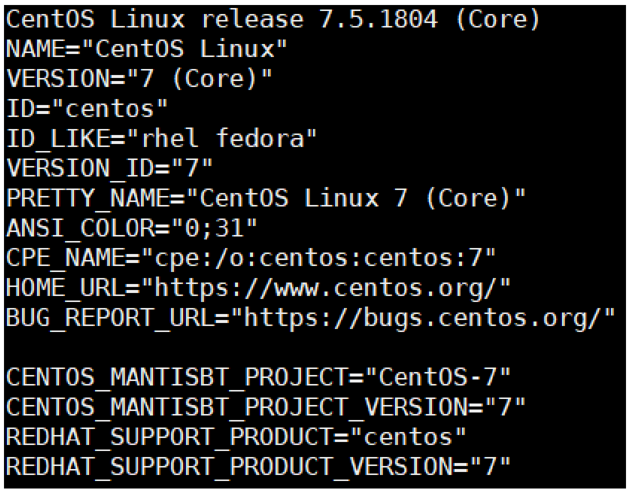
Alex Breault

CIS 625 Homework 3 Performance Analysis

To begin, we were given the task of increasing performance by running code in parallel using Open MPI. The code runs through four for loops and two do while loops. The code also reads in a dictionary of 50,000 words and stores them. Then it reads in 100,000 words from a word list in and stores that as well. The goal of the code is to loop through the two word lists and match words in the dictionary to words from the word list and output matches with their line number in the larger word list.

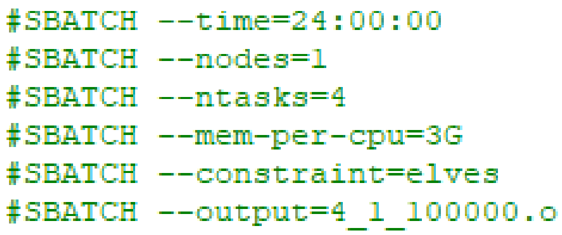
I ran everything on the elf nodes. The current bash version is: GNU bash, version 4.2.46(2)-release (x86_64-redhat-linux-gnu). The current Linux version is:



```
CentOS Linux release 7.5.1804 (Core)
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"
```

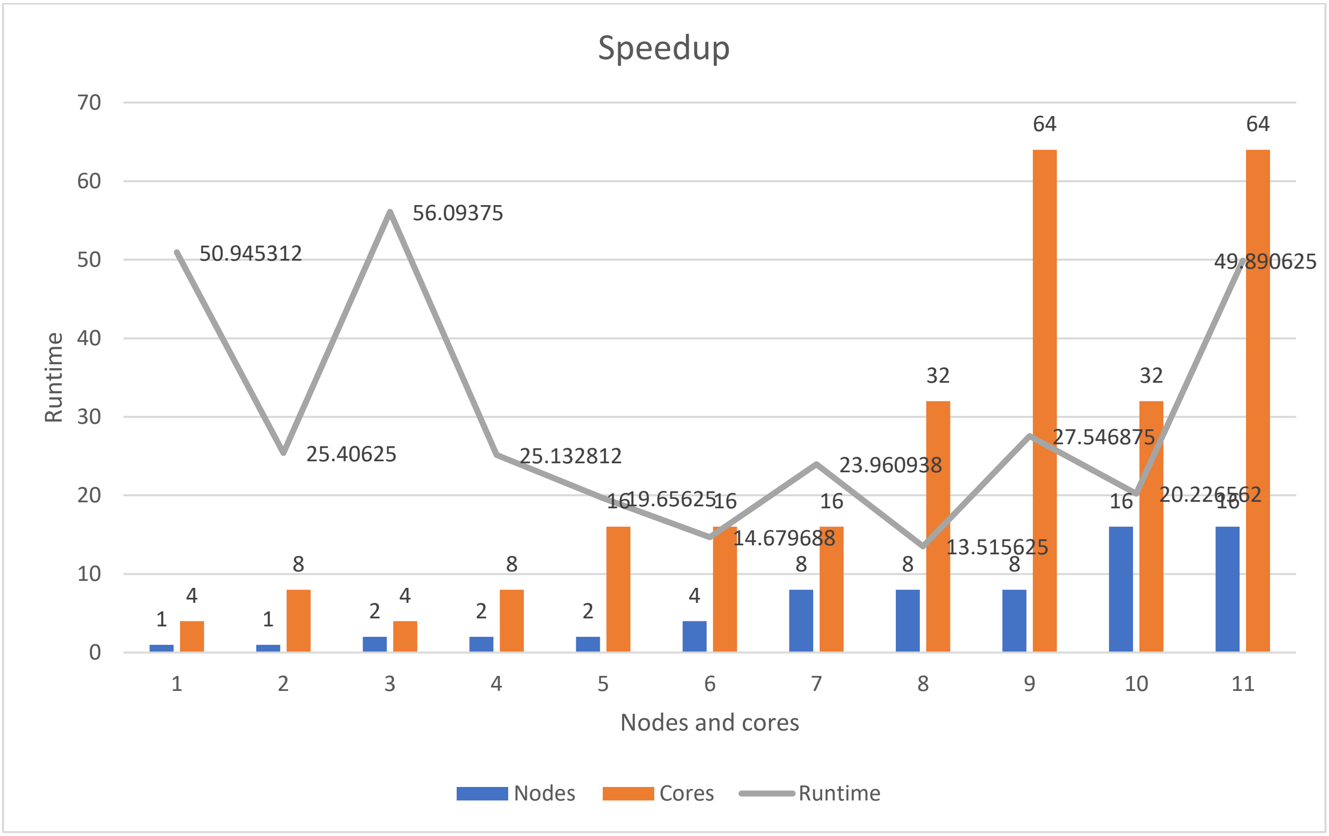
When running the program, I used a shell script to load OpenMPI and then used mpirun to run the code. I used the following sbatch parameters:



```
#SBATCH --time=24:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --mem-per-cpu=3G
#SBATCH --constraint=elves
#SBATCH --output=4_1_100000.o
```

I set the maximum time needed to 24 hours because I assumed this would take a long time. I ran multiple tests over a different combination of nodes and cores and set the output accordingly. I also set the memory per processor to 3 gigabytes as a massive overhead of "just in case."

After running 10 tests reading in 100,000 words in different node and core couples, the tests spoke for themselves. It showed that, in certain situation, throwing all the power you have at a process will not conclude in speedup. I timed the loop of when the program went through and compared the two files. I ran each combination of nodes and cores 5 times and averaged out the numbers. The graph is as follows:



The graph above shows that, the more we spread the work up between different nodes (for example, 2 nodes with 4 cores) the runtime was very long in comparison to the others. This was due to the constant attempt of communication across the two different machines because the two different machines did not have enough resources able to use. When running on multiple machines with more cores allocated (for example, 8 nodes and 32 cores) that is four cores for each machine. In a computational task as searching through two different lists, four cores is plenty for each machine.

When trying to push for more resources (for example, 16 nodes and 64 cores) this was an extreme overkill for the task at hand. Multiple cores were probably sitting there waiting for their next task at hand to be given to them. The communication between all 16 machines would have also been a reason for the slow time.

In conclusion, I saw good times for the 100,000 lines of the word list. In the future, I would have liked to run more tests with varying numbers of lines. I did not have time to do so because debugging and making the code work correctly took the majority of the time. This taught me how to properly use Open MPI and how there are many different ways to make code run in parallel.