Q2 - College ID Generator

Dean Yockey

My computer

My personal computer is a Lenovo ThinkCentre I bought for $75 on eBay. It's running the Linux Mint operating system. When running `lscpu`, the first few lines read:

```
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          39 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   4
  On-line CPU(s) list:    0-3
```

My computer has 4 CPUs.

q2.c

Because the problem of checking if a student ID is valid is very similar to the problem of checking if a circuit is satisfiable, my code for this question is very similar to the code in `SATI.c`. I admit I did copy much of the code.

The main task is performed in the `checkID` function. This function calls two other functions, `digitsum` and `consecutivedigits`, which check the sum of the digits and if two consecutive digits are the same, respectively.

`checkID` uses `sprintf` to convert the int id into an array of characters. This is because iterating through an array is a simpler way to iterate over digits than using powers of ten and division and all that.

The `consecutivedigits` function takes a variable amount of time, since a number like 110000 will abort very early, while a number like 123456 will iterate through the whole array. Similarly, `checkID` will abort early if `consecutivedigits` fails.

`digitsum` always iterates through the entire 6-digit array, so it takes constant time. However, `checkID` will abort earlier if the sum is 7, before it checks 11 and 13, due to short-circuit evaluation.

Thus, `checkID` execution time is quite varying, just like `SATI.c`'s task execution time, so the method is the same, with simple cyclic mapping.

Program result

The program found that 527787 IDs met the criteria.

Benchmarking

I created a bash file for benchmarking. It reads:

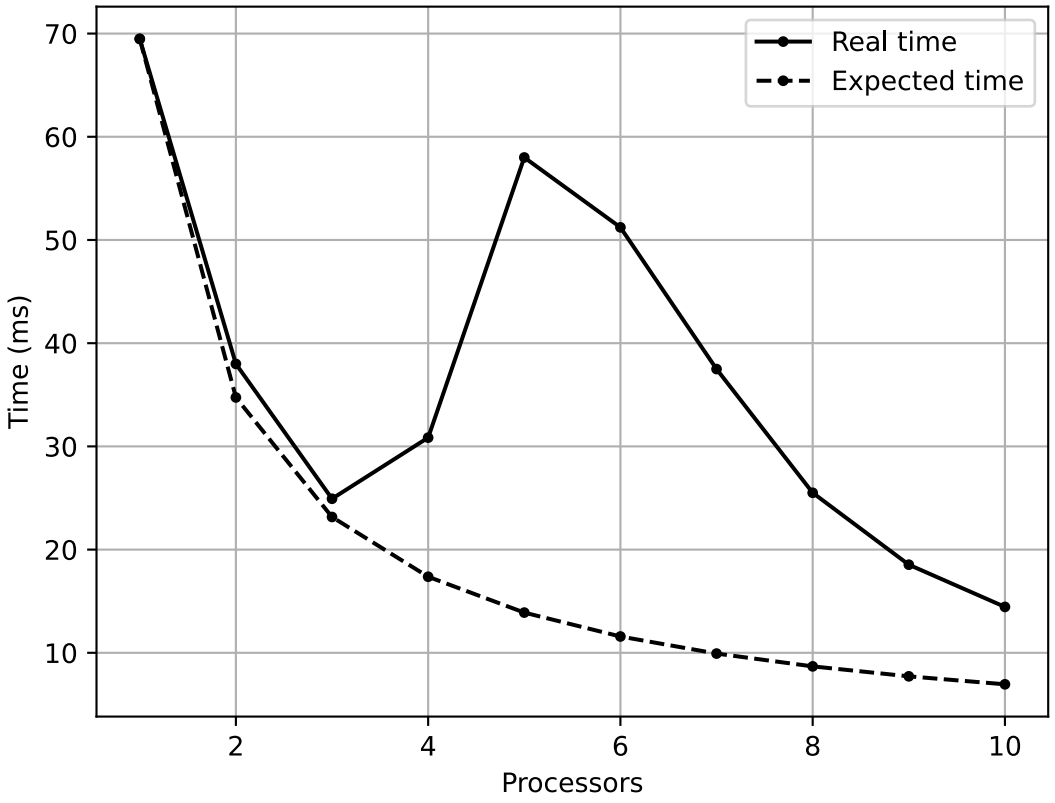
```
#! /bin/bash
echo "Q2" > "q2log.txt"
for p in {1..10}
do
     echo "P: $p" >> q2log.txt
     for i in {1..5}
     do
          mpirun -n $p --oversubscribe q2 >> q2log.txt
     done
done
```

I ran the program 5 times for each process count, as required by question 2. The output to `q2log.txt` looked like:

```
Q2
P: 1
T: 0.065038 s
T: 0.063883 s
T: 0.073802 s
T: 0.072916 s
T: 0.071771 s
P: 2
T: 0.033112 s
T: 0.039299 s
T: 0.033180 s
T: 0.037568 s
T: 0.046802 s
P: 3
T: 0.022140 s
T: 0.027808 s
...
```

My results

To extract the results from the log file, and visualize them in a line graph, I wrote a python script. I used regex and matplotlib. This is my graph.



Takeaways

I expected the program to have peak performance at 4 processors, since that's the number of processors my computer actually has, but it actually has the best performance at 10.

Additionally, 4, 5, 6, and 7 processors perform worse than either 3 or 7 processors (which are roughly the same). It's hard to guess why these middle numbers have worse performance than either side of them.

It's certainly possible, since the task of testing valid IDs takes a variable amount of time (as explained earlier), that some process counts just had very lopsided distribution by chance (perhaps process 3 got all the long ones, for example). That's my best guess.

However, it feels improper to make any conclusions off a benchmark with just 5 tests per process count. I would have certainly done more tests if I could, but the requirements of this question have my hands tied.