161L Runtime Case Study

Marco Rubio

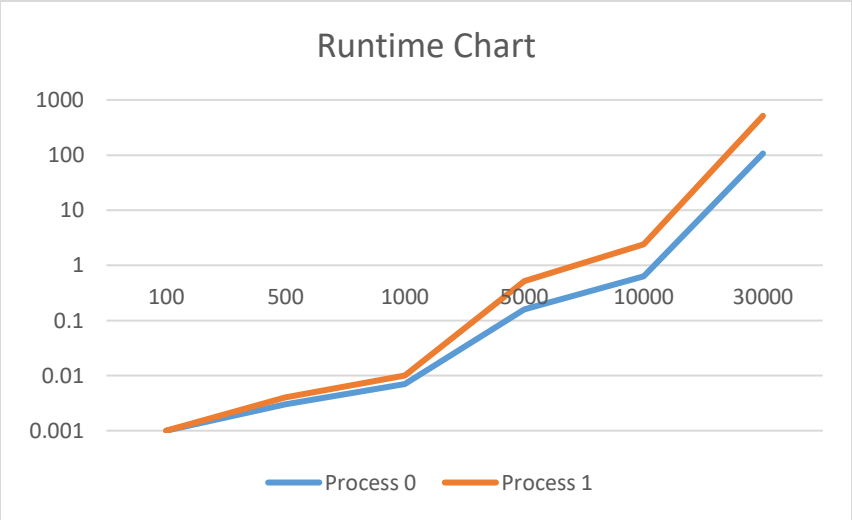
861 181 898

161L

Abstract

This is a case study designed to test two processes provided by the professor of 161L. The processes are individually ran and tested using input sizes between 100 and 30,000 to evaluate the runtime. An in-depth overview is covered in order to explain the difference in runtimes. The processes are ran under the Linux operating system and only the real time is taken from the time command.

Results



size	p0	p1
100	0.001	0.001
500	0.003	0.004
1000	0.007	0.01
5000	0.158	0.518
10000	0.624	2.398
30000	106.704	514.926

The data is taken in seconds using the real-time runtime under the Linux OS.

Discussion

The difference between the codes is very small. Below you can see the section of code where the two processes differ. At a glance, not much difference is observed. If you look closely the only difference is that in the main line of code for process 0 the "J" iterator is used to index thru the operation; in process 1 the "I" iterator is used to index thru the operation.

Process 0

```
for (i = 0; i < size; ++i) {
   for (j = 0; j < size; ++j) {
      y[i] += A[i*size + j] * x[j];
   }
}
```

Process 1

```
for (i = 0; i < size; ++i) {
   for (j = 0; j < size; ++j) {
      y[j] += A[j*size + i] * x[i];
   }
}
```

This difference might not look like a big deal but because of how computers work this actually results in some notable performance decline. This naturally lead to the question why does the performance degrade so much?

Why Performance Degraded

Computers take advantage of the concepts of ***temporal*** and ***spatial*** locality. *Temporal* locality is the property that if a piece of code was accessed once it's likely to be accessed again in the near future. *Spatial* locality is the property that if a piece of code is accessed the code preceding it is likely to also be accessed in the near future.

These two concepts are important in the results of the test shown here. If you notice how process 0 accesses the array it uses the iterator "J" which is incremented once at every iteration. In other words, the items of the array "A" are accessed sequentially in space and time. In contrast process 1 accesses the items in "J*size" steps. This means that the items are accessed in a non-sequential manner. This lowers the processors ability to use *temporal* and spatial *locality*.

Conclusion

As a beginner programmer, it's easy to overlook little mistakes like this. It's important to pay attention to these details because when working for large companies that use big data the amount of time that one process can take is very noticeable. Big data sets can have millions of items and as you can see from the results the time taken is not linear but exponential. You would not have the speedy internet you have today if it was not for simple optimizations like this.