Lab 9

Jarred Parr, Alexander Fountain

1. Working off of eos05.

- There is currently 16283404kb of physical memory on the system.
- There is currently 13183648kb of free memory on the system.

2. **a.** Estimated memory is about 16000000 bytes (16mb) this is calculated by the amount which would be allocated as a result of a malloc call. **b.** The approximate memory demand is: ~15mb when recorded with `vmstat 1 15` and once again, the before and after values were compared. Using free during the runtime of the program was what allowed me to have similar results for part a and b.

   c. The observed difference, however slight it may be, appears to be the result of the accuracy of vmstate being able to run continuously and check the memory at a reliable timestep. Because of this bit of error in manually reading calculations and watching the `free` command run.

3. **a.** The `COEFFICIENT` parameter of `60` was chosen by manually calculating the memory overhead of using the system given the amount of free memory available. This was calculated manually with the program. As seen in problem one, we have about 13 gigs of free space available of 16, after some trial and error and manual calculation, this value was the one that made the most sense to theoretically use the maximum amount of memory.

   b. Other fields that changed are the amount of virtual memory. Virtual memory is used in this case because the amount of RAM currently being used is not enough to maintain the system. As a result, it must got to the hard disk for more space. This also caused a significant slowdown on the system. This was expected based on the calculations

   c. The amount of memory used as cache went down by about 480mb and the `buf` field went down by about 10mb. `so` and `si` also went way up to show how much memory was swapping in and out of disk as a result of the use of virtual memory. Pretty much everything from blocks, to interrrupts, to even time stolen from the virtual machine all saw a sharp spike as a result of the increase in load on the system imposed by the program.
4. **a.** The page size is 4096 bytes. It takes 9.094 seconds to run. **b.** By swapping the i and j values we now access memory column by column instead of row by row. This makes cache locality hard to maintain **c.** The execution time is increased to 9.881 seconds, a 0.787 second difference. **d.** By swapping the i and j values we now have invalid TLB hits so it takes longer. (See Diagram Below)

5. **a.** After manually calculating, it is clear that the value of `65` for the `COEFFICIENT` parameter is the best option here. As calculated, this would theoretically take up the entirety of available memory upon malloc (> 16gb) **b.** Free memory is constantly decreased until a certain threshold as the `swapd` field increased a significant amount. Other fields tracking the amount of swaps to disk etc also went up dramatically as a symptom of the system using more and more of the available physical memory. **c.** Similar to what was observed in question 3, the system used physical hard drive storage to get more pages of memory to run the intensive program in memory. The command `ps -eo min_flt,maj_flt,cmd` is used to see how many page faults occured during the run of the process. This number showed that > 11000 faults occured which were minor and ~300 major page faults. This is pretty significant, but definitely not nearly as much as an x2go session might incur on a system, which is in the millions. This shows that even when artificially taxing a system, nothing beats a truly intensive program