CS33 Homework Assignment 7 Solutions

Fall 2019

1. A lower bound on the time required for matrix multiplication can be obtained by considering the constraint that the entire multiplier and multiplicand matrices must be transferred from memory to the processor and the resulting product matrix must be transferred from the processor to memory.
   a. Given a maximum memory bandwidth of 10.6 GB/sec for SunLab computers (Intel Core I5-4690 processors with DDR3-1333 memory), what is the lower bound on the time required to multiply two 2048x2048 matrices of doubles? (Assume "mega" means 2^{20} and "giga" means 2^{30}).

      Answer: The total size of the multiplier, multiplicand, and result matrices is 96 MB. At 10.6 GB/sec, it would take 8.8 milliseconds to transfer that amount of memory.

   b. This bound is not achievable unless everything fits in cache (and, even then, it wouldn't be easy). Let's assume our processors have minimal caches, and thus everything must be loaded from or stored to memory on most accesses. Assuming the elements of two of the matrices (multiplicand, multiplier, and product) must be accessed n^3 times (better algorithms exist, but are, in general, not practical), how much time is required, just for memory access, to multiply two 2048x2048 matrices (i.e., n = 2048)?

      Answer: The total number of data accesses is 2^{34} (i.e., $(2^{11})^3 \cdot 2$) (plus a relatively negligible amount for the product matrix). Each access is to 8 bytes. Thus the total amount of data transferred is 2^{37} bytes (128 gigabytes). At 10.6 GB/sec, this would take 12.1 seconds.

   c. The kji algorithm of slide XVII-23 takes 146 seconds on SunLab machines for 2048x2048 matrices of doubles (this isn't the best algorithm we looked at). Explain why so much additional time is required than mentioned in your answer to 2b. (While it does take time to do the necessary multiplications and additions, you may assume this is a rather small part of the difference between 146 seconds and your answer to 2b.) We are not asking you to account for every second of the difference, but to give a qualitative answer explaining why this difference exists. Note that the size of a cache line on SunLab processors is 64 bytes (8 doubles).

      Answer: As the numbers indicate, the kji code can't get the processor to transfer memory at this sustained rate; the measured performance is a factor of 12 slower. The maximum transfer rate of 10.6 GB/sec assumes 64-byte transfers and that memory is being transferred continuously. In the inner loop, the multiplicand (A) and the product (C) are accessed in column (non-contiguous) order. Thus for every 64 bytes transferred, only 8 are used. This accounts for a factor of 8 slowdown from the predicted 12. The remaining factor of 1.5 is most likely due to data not being transferred continuously — there are periods when no data is being transferred.

2. Consider the following program:

```
int main() {
   for (int i=0; i<=4; i++)
      fork();

   return 0;
}
```

How many processes are created (including the process that calls *main*)? Do all processes terminate?

Answer: Note that when a new process is created, its execution begins as it returns from *fork*, and its stack is a copy of its creator's. Thus it has the same value of *i* as its creator did when *fork* was called. Just one process (the one that called main) calls *fork* in the first iteration of the loop ($i = 0$), with the result that two processes are executing at the end of the loop body. From one iteration to the next, since each process calls *fork* once, the number of processes doubles. Thus there is one process calling *fork* for i=0, two for i=1, four for i=2, eight for i=3, and sixteen for i=4. Thus we end up with 32 processes. Each process returns from *main* once it completes the loop, and thus all terminate.