18.335 Problem Set 1 Solutions

Problem 1: Gaussian elimination

The inner loop of LU, the loop over rows, subtracts from each row a different multiple of the pivot row. But this is exactly a rank-1 update $U \to U - xy^T$, where x is the column-vector of multipliers and y^T is the pivot row. More explicitly, we can rewrite Gaussian elimination without row swaps (pivoting) as:

$U = A$
`for` $k = 1$ `to` $m - 1$
    $x = u_{k+1:m,k}/u_{kk}$
    $U_{k+1:m,k:m} = U_{k+1:m,k:m} - xu_{k,k:m}$

Note that I have used Matlab notation $n : n'$ to denote ranges (from n to n') of rows or columns. In particular, note that we only have to do a rank-1 update of a submatrix of U, and that $u_{k,k:m}$ is a row vector of the k-th row of U from column k to column m.

Problem 2: Asymptotic notation

(a) Θ means both O and Ω. Let's do one at a time. By the definition of O, $f(n) \le C_1 F(n)$ for $n > N_1$ and $g(n) \le C_2 G(n)$ for $n > N_2$ (no absolute values since the functions were given to be nonnegative), for some constants $C_{1,2}$ and $N_{1,2}$. If we let $C = \max(C_1, C_2)$ and $N = \max(N_1, N_2)$, then $f(n) + g(n) \le C_1 F(n) + C_2 G(n) \le C[F(n) + G(n)]$ for $n > N$, hence $f + g$ is $O(F + G)$. Similarly for Ω, replacing $\le$ with $\ge$ and max with min, so $f + g$ is $\Omega(F + G)$. Hence $f + g$ is $\Theta(F + G)$.

(b) $f(n) \in O[g(n)] \Leftrightarrow |f(n)| \le C|g(n)|$ for some $C > 0$ and $n > N \Leftrightarrow |g(n)| \ge C^{-1}|f(n)|$ for $C^{-1} > 0$ and $n > N \Leftrightarrow g(n) \in \Omega[g(n)]$. Q.E.D.

(c) $f(n) \in O[F(n)] \Leftrightarrow |f(n)| \le C|F(n)|$ for $n > N$. If $h(n) \in O[f(n) + c\,F(n)]$ then for $n > N'$ we have $|h(n)| \le C'|f(n) + c\,F(n)| \le C'|f(n)| + C'|c|\,|F(n)|$ for some $C' > 0$. For $n > \max(N', N)$, $|h(n)| \le (C'C + C'|c|)\,|F(n)|$ where $C'C + C'|c| > 0$, and thus $h(n) \in O[F(n)]$. However, the same inference is not true if we replace O with Θ; as a simple example, consider $f(n) = n^3$ and $F(n) = n^3 - n^2$ with $c = -1$: in this case $n^3 \in O(n^3 - n^2)$, but $f(n) + c\,F(n) = n^2$ and $\Theta(n^2)$ is not a subset of $\Theta(n^3 - n^2) = \Theta(n^3)$.

(d) If the running time is $O(n^2)$, that means that the time is $\le n^2$ multiplied by a constant, asymptotically. If it is "$O(n^2)$ or worse", that would mean that the time is bounded above by any function $\ge n^2$, which is true of *every* function! Usually, when you hear things like this, what people really mean is "$\Theta(n^2)$ or worse," or equivalently "$\Omega(n^2)$".

Problem 3: Caches and matrix multiplications

(a) See figure 1. For discussion of the results, see part (d).

(b) The only temporal locality in a matrix-vector multiply $y = Ax$ is that the vector x is re-used to multiply against each row of A; nothing in A is re-used, as each element of A is needed exactly once. Thus, there will always be m^2 misses to read in A. Furthermore, x is read in m times, but asymptotically (for large $m > Z$) x will not fit in cache and hence (in the straightforward row-column algorithm) every read of x will incur m cache misses (by the time you get back to the first element of x, it has left the cache), for m^2 misses in total. Or, technically, with an ideal cache it would store $Z - 1$ of the elements in cache and read in the remaining elements one by one, for $m + (m-1)(m-Z+1)$ misses, but for large m only the m^2

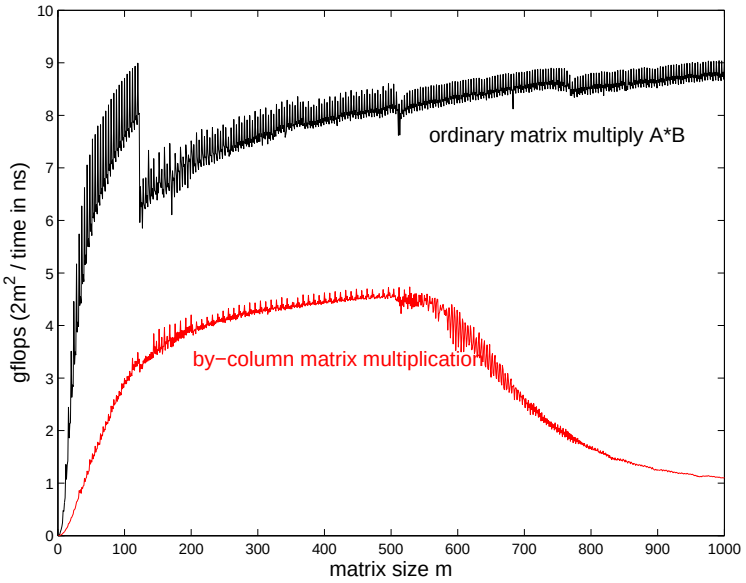

Figure 1: Matrix-multiply benchmarks in Matlab 7.6.0.324 (R2008a) on a 2.66GHz Intel Core 2 duo. I attempted to use only one CPU by setting `maxNumCompThreads(1)`, but `top` showed that it was still using 2 CPUs for some reason.

leading term matters. Thus, there are $2m^2$ misses asymptotically, independent of Z. (Note that, like our analyses in cache, I'm not including cache-line effects. If you include cache lines of length L then you get $2m^2/L$ misses if A is row-major, or $m^2 + m^2/L$ if A is column-major since it is being read in the "wrong" order.)

(c) Since x is re-used, we should read x in blocks of size αZ for some fraction α to be determined. Then, for each block, we will multiply it by the corresponding block of αZ columns of A before moving on to the next block of x.

let $y = A_{:,1:\alpha Z} x_{1:\alpha Z}$ (first block)
let $y = y + A_{:,(\alpha Z+1):2\alpha Z} x_{(\alpha Z+1):2\alpha Z}$ (next block)
et cetera, for $m/(\alpha Z)$ blocks

Now, what should α be? Since we can't get any re-use for A, then we should read A only one element at a time and then discard it; this means we only need to reserve one cache entry for A. Unfortunately, we have now introduced a new set of reads: y has to be read in each time in order to add it to the next block. In the above algorithm, once $m > Z$ we can't really get cache reuse, like in part (b), and incur $(\frac{m}{\alpha Z} - 1)m \approx m^2/\alpha Z$ misses. So, in that case we might as well reserve only one cache entry for reading in y (i.e. reading it one entry at a time and then discarding form the cache). So $\alpha Z = Z - 2$. By construction, x is read into cache only once, for m cache misses (asymptotically negligible compared to m^2). Thus, the total number of cache misses is $m^2 + m^2/(Z-2) \approx m^2(1 + 1/Z) \approx m^2$. For large Z, this is a savings of about a factor of 2 over the naive algorithm—nothing to sneeze at, but nothing like the factor of $\sqrt{Z}$ we can save for a matrix-matrix multiply!

You might wonder whether we can gain some additional savings by blocking y as well. That

is, by doing the above algorithm in blocks of $\approx Z/2$ in y and blocks of $\approx Z/2$ in x. However, in this case we would have to read in x multiple times, and the number of cache misses would end up being $m^2(1+2/Z)$, slightly worse (essentially because we are only using half the cache for x). There are other ways to re-arrange the algorithm as well, but none of them do better than $m^2(1+1/Z)$ as far as I can tell. In any case, for large m and Z we are dominated by the m^2 misses to read in A, and there's nothing that can be done about this.

(d) Yes. In the by-column matrix multiplication, we can see a huge drop in performance once the matrix size goes out of the L2 cache, whereas there is no such drop for the ordinary matrix multiplication because the $1/\sqrt{Z}$ factor makes the cache-miss cost negligible.

It ultimately achieves roughly the peak flop rate, as in class; note that if you have multiple CPUs, it may be impossible to prevent Matlab from using at least 2 processors. Even calling `maxNumCompThreads(1)` as documented in the Matlab manual, it still used two processors for me in Matlab 7.6! *Grrr.* Hence, I got a peak flop rate of almost 10 gflops on the 2.6GHz Intel Core 2 (two processors, ~5 gflops peak for each using SSE2 instructions).

Note that, for small matrix sizes, the performance is a lot lower in both cases, but especially in the by-column multiply. This is simply the overhead of the Matlab interpreter, which is much larger for the by-column case because that has a Matlab loop (versus a single Matlab call for the ordinary matrix-multiply). In both cases, the interpreter overhead (which is $O(m)$ in the by-column case) becomes negligible for large m, however.

(I'm not sure what the temporary drop in the ordinary matrix-multiply case is around $m = 100$.)

Problem 4: Caches and backsubstitution

(a) For each column we get a cache miss for each entry r_{ij} of the matrix R, an there are roughly $m^2/2$ of these. For large enough m, where $m^2 > Z$, these are no-longer in-cache and incur new misses on each column. Hence there are roughly $m^2 n/2$, or $\Theta(m^2 n)$ misses. (No asymptotic benefit from the cache.)

(b) We can solve this problem in a cache-oblivious or cache-aware fashion. I find the cache-oblivious algorithm to be more beautiful, so let's do that. We'll divide R, X, and B into $\frac{m}{2} \times \frac{m}{2}$ blocks for sufficiently large m:[1]

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

where R_{11} and R_{22} are upper-triangular. Now we solve this, analogous to backsubstitution, from bottom to top. First, for $k = 1, 2$, we solve

$$R_{22} X_{2k} = B_{2k}$$

recursively for X_{2k}. Then, for $k = 1, 2$ we solve

$$R_{11} X_{1k} = B_{1k} - R_{12} X_{2k}$$

recursively for X_{1k}. We use a cache-optimal algorithm (from class) for the dense matrix multiplies $R_{12} X_{2k}$, which requires $f(m) \in \Theta(m^3/\sqrt{Z})$ misses for each $\frac{m}{2} \times \frac{m}{2}$ multiply. The

[1] If m is not even, then we round as needed: R_{11} is $\lceil \frac{m}{2} \rceil \times \lceil \frac{m}{2} \rceil$, R_{12} is $\lceil \frac{m}{2} \rceil \times \lfloor \frac{m}{2} \rfloor$, R_{21} is $\lfloor \frac{m}{2} \rfloor \times \lceil \frac{m}{2} \rceil$, and $R_{22}$is $\lfloor \frac{m}{2} \rfloor \times \lfloor \frac{m}{2} \rfloor$; similarly for X and B.

number $Q(m)$ of cache misses then satisfies the recurrence:

$$Q(m) = 4Q(m/2) + 2f(m) + 4m^2,$$

where the $4Q(m/2)$ is for the four recursive backsubsitutions and the $4m^2$ is for the two matrix subtractions $B_{1k} - R_{12}X_{2k}$. This recurrence terminates when the problem fits in cache, i.e. when $2m^2 + m^2/2 \leq Z$, at which point only $\sim 3m^2/2$ misses are required. (Since we are only interested in the asymptotic Θ results, these little factors of 3 and 4 don't matter much, and I'll be dropping them soon.) Noting that $f(m/2) \approx f(m)/8$, we can solve this recurrence as in class by just plugging it in a few times and seeing the pattern:

$$\begin{aligned}
Q(m) &\approx 4[4Q(m/4) + 2f(m)/8 + 4m^2/4] + 2f(m) + 4m^2 \\
&= 4^2 Q(m/4) + 2f(m)\left[1 + \frac{1}{2}\right] + 4m^2\ [1+1] \\
&\approx 4^3 Q(m/8) + 2f(m)\left[1 + \frac{1}{2} + \frac{1}{2^2}\right] + 4m^2\ [1+1+1] \\
&\approx \cdots \\
&\approx 4^k \Theta[(m/2^k)^2] + 2f(m)\left[1 + \frac{1}{2} + \cdots + \frac{1}{2^{k-1}}\right] + 4m^2\ [k] \\
&\approx \Theta(m^2) + \Theta(m^3/\sqrt{Z}) + \Theta(m^2)k
\end{aligned}$$

where $\left[1 + \frac{1}{2} + \cdots + \frac{1}{2^{k-1}}\right] \leq 2$ and k is the number of times we have to divide the problem to fit in cache, i.e. $3(m/2^k)^2/2 \approx Z$ so k is $\Theta[\log(m^2/Z)]$. Hence, for large m where the m^3term dominates over the m^2 and $m^2 \log m$ terms, we obtain

$$Q(m; Z) \in \Theta(m^3/\sqrt{Z})$$

and hence we can, indeed, achieve the same asymptotic cache complexity as for matrix multiplication.

We could also get the same cache complexity in a cache-aware fashion by blocking the problem into m/b blocks of size $b \times b$, where b is some size in $\Theta(\sqrt{Z})$ chosen so that pairwise operations on the individual blocks fit in cache. Again, one would work on rows of blocks from bottom to top, and the algorithm would look much like the ordinary backsubstitution algorithm except that the numbers b_{ij} etcetera are replaced by blocks. This is a perfectly acceptable answer, too.

18.335J / 6.337J Introduction to Numerical Methods
Fall 2010