October 9, 2013

CSCI 2830
Solutions to Application 1: Machine Issues in Linear Algebra Computational Performance

1 Matrix-vector multiplication (15 points)

1. **Building a Test Script**

   Here is my wrapper script (calling program):

```
function runtimes = matvec_runner

sizes = [10 50 100 200 500 1000 1500 2000 2500];
[vm,vn] = size(sizes);

for j = 1:vn

   n = sizes(j)
   a = rand(n,n);
   v = rand(n,1);
   x = v;
   y = v;
   z = v;

   t = cputime; matvec_by_row(a,v); rowtime = cputime-t;
   t = cputime; matvec_by_col(a,x); coltime = cputime-t;

   runtimes(j,1) = n;
   runtimes(j,2) = rowtime;
   runtimes(j,3) = coltime;

   [l,u] = lu(a);
   t = cputime; back(a,v); rowtime = cputime-t;
   t = cputime; backcol(a,v); rowtime = cputime-t;
```

```
   runtimes(j,4) = rowtime;
   runtimes(j,5) = coltime;

end

clf
hold on;
plot(runtimes(:,1),runtimes(:,2),'bo')
plot(runtimes(:,1),runtimes(:,3),'g+')
plot(runtimes(:,1),runtimes(:,4),'rv')
plot(runtimes(:,1),runtimes(:,5),'mx')

print -djpeg 'timings'
hold off;
```

2. **Running Some Experiments**

   (a) Once you are satisfied with your wrapper script, expand your test matrix orders to `[5 10 50 100 200 500 1000 1500 2000 2500]`. Repeat the tests for those orders. (This experiment runs in a reasonable amount of time on bfs in the CSEL.)

   (b) **Analyzing the Results**
   What is the ratio of times for the two routines for matrix order 2500? In Assignment 2, you established that the two routines performed the same number of flops. Thus, difference in runtime are due to differences in memory access. From your results, what can you conclude about how MATLAB stores matrices? That is, are they stored in row-major format or column-major?

Solution:

You should see that the column version is faster than the row version because MATLAB uses column major storage. The ratio will depend on your hardware. Looking through the papers, I see lots of ratios in the 2-4 range, but your number might be something different. (I'd expect the ratio to be something pretty different from one.)

2 Solving upper triangular systems (25 points)

3. What follows are two different MATLAB scripts for performing backsubstitution to solve an upper triangular system.

```
function x = backrow(U,b)
%
% Solve Ux = b by backsubstitution.
%
[m,n] = size(U);
for k = n:-1:1
  tmp = 0;
  for j = k+1:n
     tmp = tmp + U(k,j)*x(j);
  end
  x(k) = (b(k) -  tmp)/U(k,k);
end
```

```
function x = backcol(U,b)
%
% Solve Ux = b by backsubstitution.
%
[m,n]=size(U);
for j = n:-1:2
  x(j) = b(j)/U(j,j);
  b = b - x(j)*U(:,j);
end
x(1) = b(1)/U(1,1);
x = x';
```

Explain why both of these scripts get the same answer. You must show how (or why) each one actually solves the system $Ux = b$.

Solution:

These algorithms mimic what you saw for mvrow and mvcol. Backrow is the usual backsubstitution algorithm that works up through the

upper triangular matrix from the bottom, each row in turn. Backcol recognizes that the product $Ux = b$ is a linear combination of the columns of U with coefficients equal to the elements of x. As an element of x is formed, its column is removed from b which lets us identify the the next element of x and so on.

4. Modify your wrapper script to time the two solvers. Plot runtime versus matrix order for the 10 orders specified above for all four linear algebra routines. (Test first for a smaller number of matrix orders!!) Now the timing array can be 10×5.

5. Do the times for the solvers validate your conclusion about how MATLAB stores matrices? Why or why not?

   Solution: You should see that the column version is again faster than the row version.

6. For an $n \times n$ matrix, backsubstitution requires about $\frac{n^2}{2}$ flops while matrix-vector multiplication (when the matrix is full, not upper triangular) requires about $2n^2$ flops. Explain the difference in runtimes for the four routines.

   You can't give a fully qualified answer to this question without instrumenting your codes to follow memory traffic, but you can give a good, qualitative answer based on how the matrix elements are accessed from the linear array in the four scripts.

   Solution:

   If floating-point arithmetic were all that mattered, the matrix-vector multiplication routines would run at the same rate and as would the two backsubstitution routines. The mat-vec routines would be about four times slower than the solvers. Because of the memory access costs, however, the column oriented routines are faster than row oriented ones. You also may or may not see the factor of four, depending on what's going on with memory access.

   It is fine to have all plots on the same graph as long as it is possible to distinguish them. All of the curves should be concave up parabolas.

Turn in a copy of your final wrapper script (that calls all four routines), a copy of your matrix-vector multiplication timing plot, and a copy of your

timing plot for all four routines in addition to your answers to the above questions.

Also, please tell me as much as you know about the machine you used– processor type and speed, cache and memory sizes, name of CS machine, etc. I tried these experiments on two different UltraSPARCs (bfs is one) and on a Pentium III-based machine with similar results.