CS 378 Spring 2015: Mid-term exam

March 11th, 2015

1. *Short questions (35 points)* Answer the following questions using 3-4 sentences for each one.

   (a) (4 points) What is meant by *direct* and *iterative* methods for solving linear systems?

   A linear system of equations can be written as Ax = b where x is the vector of unknowns. A direct method factorizes A into a product LU where L is lower triangular and U is upper triangular, and then solves two triangular systems to find x.

   An iterative method computes a series of approximations x_0, x_1, x_2,...,x_m to the solution. The vector x_0 is chosen to be either some arbitrary vector or by using some insight into the solution might be. Each of the remaining vectors x_i are obtained from the previous ones by computing the previous residuals. Jacobi and Gauss-Seidel use only the previous residual Ax_{i-1}-b to compute x_i from x_{i-1}.

   (b) (4 points) Name one iterative method for solving linear systems and describe briefly how it works.

   Jacobi iteration. The first equation is used to compute a new value for the first variable, the second equation is used to compute a new value for the second variable and so on. To compute a new value for a variable x_i, we substitute the old values for all other variables in the left-hand side of equation i, and solve it for the new value of x_i.

   (c) (4 points) What is the key computational kernel in iterative methods for solving linear systems?

   Matrix-vector multiplication.

   (d) (6 points) What is meant by BLAS1, BLAS2 and BLAS3 kernels? Give examples of each kind.

   BLAS stands for Basic Linear Algebra Subroutines, and it is a name for a set of dense linear algebra computations that are ubiquitous in applications. The number after BLAS refers to the number of loops required to implement a particular computation; for example, matrix-matrix multiplication is BLAS3 because it requires 3 nested loops.

BLAS1: inner product of vectors, SAXPY.

BLAS2: matrix-vector product

BLAS3: matrix-matrix product

(e) (3 points) What is data-parallelism? Give an example of a computation that exhibits data-parallelism.

Data-parallelism refers to the parallelism that arises when the same operation is performed on each element of a collection such as an array or a set. Squaring all the elements of an integer array is an example.

(f) (4 points) What are the sources of cache misses in sequential programs?

These are known as the 3 C's: cold, capacity and conflict misses.

(g) (10 points) Describe three algorithms for solving the SSSP problem, and explain how they differ, using terminology from TAO analysis. Which algorithm would you use on a sequential machine? Which one would you use on a parallel machine? Explain your answers briefly.

Chaotic relaxation: data-driven, unordered.

Dijkstra: data-driven, ordered. Work-efficient and good for sequential machines.

Bellman-Ford: topology-driven, unordered inner loop and ordered outer loop.

On a parallel machine, people use either Bellman-Ford or delta-stepping.

2. *Finite-differences (30 points)* Consider the ordinary differential equation
   $\frac{dy}{dx} = y * sin(x)$
   with initial condition $y(0) = 1$ in the interval $0 < x < 2$. Use finite-differences to find an approximate solution to this differential equation, using a step size $h = 0.5$.

   (a) (10 points) Recall that the forward-Euler approximation of the derivative is the following:
   $\frac{dy}{dx}|_{i*h} \approx \frac{y((i+1)*h)-y(i*h)}{h}$
   Write down a recurrence formula for computing $y(0), y(h), y(2h), y(3h)$.
   $y(0) = 1$
   $\frac{y((i+1)*h)-y(i*h)}{h} = y(i*h)sin(i*h)$
   Many of you did not understand how to convert the "y" and "x" on the right-hand of the differential equation into their discrete counterparts. The way to think of the discretization is that you are estimating the values of y at the points $x = 0, h, 2h, 3h$ etc. That is, you are computing the values of $y(0), y(h), y(2h), y(3h)$. In forward-Euler, you follow the derivative at the current point to estimate the next point.

   (b) (20 points) Recall that the centered-difference approximation of the derivative is the following:
   $\frac{dy}{dx}|_{i*h} \approx \frac{y((i+1)*h)-y((i-1)*h)}{2h}$.
   Write down a system of linear equations for computing the values of $y(0), y(h), y(2h), y(3h)$. You do not have to solve this system.
   $y(0) = 1$
   $\frac{y((i+1)*h)-y((i-1)*h)}{2h} = y(i*h)sin(i*h)$.
   For centered-differences, you can use the derivative at 0 to estimate the value of y(h). The derivative at 0 is y(0)*sin(0) (from the differential equation), which is 0. So y(h) is approximately y(0) + 0*h = 1. Now that you know y(0) and y(h), you can compute y(2h) and y(3h), but you were not asked to do this.

3. *Caches (35 points)* Consider the following kernel, where *old* and *new* are two arrays of size $N \times N$ that are initialized to some values. Assume that both arrays are stored in row-major order and that the processor does not pre-fetch cache lines.

```
for i = 2, N-1
   for j = 2, N-1
     new(i,j) = 0.25*(old(i-1,j)+old(i,j-1)+old(i,j+1)+old(i+1,j)
```

   The blocked/tiled code is the following:

```
for ib = 2,N-1,NB
  for jb = 2,N-1,NB
    for i = iB, iB+NB-1
      for j = jB, jB+NB-1
          new(i,j) = 0.25*(old(i-1,j)+old(i,j-1)+old(i,j+1)+old(i+1,j)
```

As we discussed in class, you can do this with 3 loops rather than 4.

(a) (5 points) Explain the opportunities for exploiting parallelism and locality in this kernel.

All points in the `new` array can be computed in parallel, and this is an example of data-parallelism.

Walking the arrays `old` and `new` in row-major order as this loop does is good for spatial locality since the matrices are stored in row-major order.

The accesses to `new` have good temporal locality as well since we access all the elements of a cache line of `new` (except at the boundaries), and then never revisit that cache line again. Two of the accesses to `old` will exhibit good temporal locality (the accesses to elements in row `i`) but the accesses to the rows `i-1` and `i+1` will not exhibit good temporal locality for large values of N.

(b) (5 points) What is the miss ratio when N is very large compared to the capacity of the cache? You may assume that the line size is one, so each miss brings in a line with one number in it.

The access to `new` will always be a cold miss since the line size is 1. The accesses to `old(i-1,j)` and `old(i+1,j)` will be misses, and the other two accesses will be hits. So the miss ratio will be 2/5 which is 0.4.

(c) (5 points) Is loop interchange legal in this kernel? If so, would you expect the interchanged version to perform better or worse than the original version?

Yes. Worse, because it would exhibit poor spatial locality.

(d) (10 points) Write down a blocked version of this code that operates on square blocks of size $NB \times NB$. You can assume that NB divides $(N-2)$ exactly so there is no clean-up code required to handle fractional blocks.

(e) (10 points) Explain briefly how you would determine a good value for NB.

You would pick NB to be as large as possible without suffering capacity misses within the block computation.

You did not have to calculate this, but the inequality is $2 * NB < C$ where C is the capacity of the cache, which gives an NB value of $C/2$. In practice, you would use a fraction of this value to correct for conflict misses.