Name: _____

18-645 How To Write Fast Code

March 5 2008

Total: 6 Questions. Maximum points: 100

1. *(20 pts, 2 each)* Which of the following statements are true? $f(n)$ and $g(n)$ are arbitrary positive real-valued function of n. No justification is necessary.

   Answer with true or false or just leave blank. Wrong answers will be assigned 2 negative points each (if you answer half the questions correctly and half incorrectly, your total grade will be a zero).

   (a) $100n = O(n)$

   (b) $\Theta(2f(n)) \subset O(f(n))$

   (c) $\cos(f(n)) = O(0.1)$

   (d) $\log_2(n) = O(n^{\frac{1}{20}})$

   (e) $2^{(n^2+n+1)} = O(2^{(n^2)})$

   (f) $O(n^2m + nm^2) = O(n^2)$

   (g) $\sum_{i=0}^{n} i = n^2 + O(n)$

   (h) $\log_{10}(n) = \Theta(\log_2(n))$

   (i) $3^{2n} = O(2^{3n})$

   (j) If $f(n) = O(n)$ and $g(n) = O(n)$ then $f(n) + g(n) = O(n)$

   (a) True (return to definition)

   (b) True because $\Theta(f(n)) \subset O(f(n))$

   (c) True because cos is bounded

   (d) True because log is smaller than any polynomial.

   (e) False because $2^{(n^2+n+1)} = O(2^n 2^{n^2})$

   (f) False (not comparable)

   (g) False because $\sum_{i=0}^{n} i = n(n+1)/2 = n^2/2 + O(n)$

   (h) True (class)

   (i) False because $3^{2n} = 9^n$ and $2^{3n} = 8^n$

   (j) True (return to definition)

Name: _____

2. *(30 pts)* The following code is a divide-and-conquer implementation of what is called the Walsh-Hadamard transform. It takes as input a vector `v_in` of size `size` which must be a power-of-two 2^k and produces a vector `v_out` of the same size as output.

```
void wht(int size, double *v_out, double *v_in){
  if (size==1){
    v_out[0] = v_in[0];
    return;
  }
  wht(size/2, v_out, v_in);
  wht(size/2, v_out+size/2, v_in+size/2);
  for(int i=0; i<size/2; i++){
    double a = v_out[i];
    double b = v_out[size/2 + i];
    v_out[i] = a + b;
    v_out[size/2 + i] = a - b;
  }
}
```

(a) Find an appropriate floating point cost measure for this algorithm (do not consider index computations).

There are only floating point additions in this algorithm so the count of the number of floating point additions is a good cost measure.

(b) Assuming a processor running at 3 GHz can steadily execute 1 vector addition and 1 vector multiplication per cycle with 2 doubles per vector, what is the maximum possible performance that can be achieved *with this algorithm* and why?

There are no multiplications in the algorithm so:

$$3 * 10^9 \frac{cycles}{s} * 1 \frac{vectoradd}{cycle} * 2 \frac{doubles}{vectoradd} = 6GFlop/s$$

(c) Using your cost measure, compute the exact cost of this algorithm.
Hints: You may use the following formula. The recurrence

$$\begin{aligned} f_0 &= c \\ f_k &= a f_{k-1} + s_k, \quad k \geq 1, \end{aligned}$$

has the solution

$$f_k = a^k c + \sum_{i=0}^{k-1} a^i s_{k-i}.$$

(If you use this formula, you'll have to simplify this expression of course.)

$$T(n) = 2T(n/2) + n$$

When introducing $2^k = n$ the formula above can be used ($T(1) = 0$). We finally find $n \log(n)$ additions.

Name: _____

3. *(30 pts)* Consider a direct-mapped cache of size $c = 2^{10}$ double precision floating point elements, with a line size of 4 doubles per line. Now consider a vector `X` of size $n = 2^{11}$, that is accessed in a pattern expressed by the following code:

```
for (int i=0; i < stride; i++)
  for (int j=0; j < n; j += stride) {
    int k = i+j;
    printf("Accessing x[%d]\n", X[k]);
  }
```

The code is parameterized by **stride**, which can take any of the values $1 = 2^0, 2^1, 2^2, \ldots, 2^{10}$.

We assume a cold cache, i.e., no part of `x` is in cache before the code is executed.

For the following questions, you must show your work. This is best done by drawing cache figures in addition to your answers where appropriate.

(a) Count the number of expected cache hits and cache misses that the code produces depending on the stride. Note that we are asking for the *number* of cache hits and misses, and *not* the hit/miss rate.

For stride $s = 1$, there would be $N/4$ cache misses, because of the neighbor use due to the line size of 4. For $s = 2$, there would be $N/2$ cache misses, since only 2 elements in each cache line are used before it gets evicted. For $s \geq 4$, there are N cache misses because there is no reuse. Cache hits and cache misses, obviously, add up to N in each case, using which the number of cache hits can be calculated.

(b) Does the cache usage involve data reuse (temporal locality)? Does it involve neighbor use (spatial locality)? Justify your answer.

There is no temporal locality or data reuse, since each element in the array is accessed exactly once. Neighbor use (due to spatial locality) does occur, but only for strides ≤ 2 (which is why these strides have a lower number of misses).

(c) Now consider a cache with the same parameters (including size) as given at the very beginning of this question, except that it is now a 2-way set associative cache. How does the number of cache hits and misses change (depending on the stride)?

With a 2-way set associative cache, the solution in part (a) still holds, since the fact that that we now have 2 sets is compensated for by the fact that the number of cache lines has been halved. (Note that this reasoning does not necessarily apply generally, but applies to this particular codelet).

However, there is one exception that might not be initially apparent: for $s = 2^9$ case, there would only be $N/4$ cache misses. You will need to work this out to see how and why this case exists. If you included this case in your answer, you received extra credit for it.

(d) Now consider a cache with the same parameters (including size) as given at the very beginning of this question, except that it now has a cache line size of 8 doubles per line (it is still direct-mapped). How does the number of cache hits and misses change (depending on the stride)?

Doubling the cache line size does have an effect on this codelet. Using the same reasoning as in part (a), we now have $N/8$ misses for stride $s = 1$, $N/4$ misses for $s = 2$, $N/2$ misses for $s = 4$, and N misses for $s \geq 8$.

Name: _____

4. *(10 pts)* Assume you have two algorithms A_1 and A_2 for the same numerical problem with arithmetic cost $\Theta(n^2)$ and arithmetic cost $\Theta(n^3)$, respectively (arithmetic cost = number of floating point adds and mults). For which algorithm to you expect the higher performance (in Mflop/s) and why?

   First, note that you are asked to consider performance in Mflop/s. This is the number of floating point operations *per unit time*.

   A practical assumption that needs to be made to answer this question is that the number of loads and stores remains the same among the implementations of A_1 and A_2. We also assume that both implementations are "reasonable" - i.e., they aren't deliberately crippled.

   With this assumption, algorithm A_2's implementation would have higher performance, since there is presumably a greater degree of reuse. Note that A_2's implementation would likely run slower than A_1, and would not be practical, but would nevertheless have a higher Mflop/s performance number. The lesson to be learned here is that higher performance (in Mflop/s) does not necessarily mean faster run time.

5. *(10 pts)* Consider BLAS 2 matrix vector multiplication, computed as $y = y + Ax$ by definition, $x, y \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ using a routine `mvm(double *x, double *y, double *A)`.

   Which of the elements of the input arrays are reused (temporal locality) and how often, i.e., in how many operations?

   Matrix A has zero reuse, as does array y. Array x is reused n times - once for each row of A.