ETH login ID:
(Please print in capital letters)

Full name: _____

263-2300: How to Write Fast Numerical Code
ETH/CS, Spring 2012
Midterm Exam
Friday, April 27, 2012

MASTER SOLUTION

Instructions

- Make sure that your exam is not missing any sheets, then write your full name and login ID on the front.
- The exam has a maximum score of 100 points.
- No books, notes, calculators, laptops, cell phones, or other electronic devices are allowed.

Problem 1 (12 = 4 each)	
Problem 2 (15 = 5 each)	
Problem 3 (18 = 14 + 4)	
Problem 4 (18)	
Problem 5 (16 = 6 + 3 + 4 + 3)	
Problem 6 (21 = 3 + 3 + 15)	
Total (100)	

Problem 1 (12 points)

1. Consider the following program:

```
double A[768*4096];
double B[768*4096];
double C[768];
int i, j;
for (i = 0; i < 768; i++)
  for (j = 0; j < 768; j++)
    C[i] += A[j * 4096] * B[i * 4096];
return;
```

On a particular Core 2 platform, you know that the working set (the part of A that is reused in every iteration of the i-loop) fits into cache. Even though you measure the runtime with warmed up the cache, the performance is bad.

(a) What is the most likely reason?

The stride we access with is a two power which might cause conflict misses.
Also the stride is bigger than a page -> meaning we get a big TLB penalty every iteration

(b) How would you resolve it?

For both problems copying the data would pay off

2. Illustrate with a simple pseudocode example a case where a write-back cache should be preferrable over a write-through cache.

The code above is a good example.
Every code that repeatedly updates the same variable will benefit from write back.

Problem 2 (15 points)

Consider the following loop performing a simple filter on an $n \times n$ array A of single precision floats.

```
float h = 1/2.;
for (i=1; i < n-1; i++)
  for (j=1; j < n-1; j++)
    B[i][j] = h*(A[i][j-1] + A[i][j+1] + A[i+1][j] + A[i-1][j]);
```

We assume a processor that can complete 1 floating point add and 1 floating point mult per cycle, and which has a memory bandwidth for loads of 2 bytes/cycle.

1. Determine a lower bound (in cycles) for the runtime based on the floating point operations performed.

Serialized execution due to dependencies

$r_{op} = 4 \cdot (n-2)^2$ cycles

2. Determine a lower bound (in cycles) based on loads (floating point only) and assuming a cold writeback cache with a cache block size of one double.

ASSUMPTION ∅: write-allocate cache

Bytes/float

$r_{load} = 4 \cdot \underbrace{\left[\overbrace{n^2-4}^{A} + \overbrace{(n-2)^2}^{B}\right]}_{\text{Bytes}} \cdot \underbrace{\frac{1}{2}}_{\text{cycles/Byte}} = 4n(n-2)$ cycles

ASSUMPTION 1: No-write-allocate cache

$r_{load} = 4(n^2-4) \cdot \frac{1}{2} = 2(n^2-4)$ cy

3. Based on the above, determine an upper bound (not asymptotic) on the operational intensity measured in flops/byte.

$I \leq \frac{4(n-2)^2}{4 \cdot \#\text{loads}}$

Ass. ∅ → $\frac{n-2}{2n}$

Ass. 1 → $\frac{n-2}{n+2}$

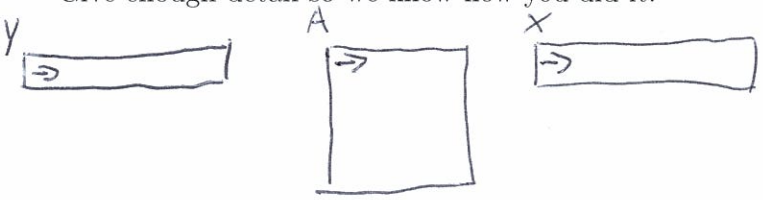
Problem 3 (18 points)

We consider double precision matrix-vector multiplication (MVM) in the form $y = Ax$, where x, y are of length n and A is $n \times n$. Assume a cache of size $C \geq 4n$ doubles, and a cache block size of 8 doubles. Further we assume a cold cache.

1. Using the above assumptions, estimate the number of cache misses (as function of n) of the below standard double loop MVM. Ignore possible conflicts between the placement of x, y, A in cache.

```
for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
      y[i] += A[i][j]*x[j];
```

Give enough detail so we know how you did it.



Because 4n fit into cache ~~x~~ all of y and x will fit and a row of A.
In the first iteration we get $\frac{n}{8}$ misses for x and y each.
(Compulsory misses for the first of the 8 elements of a cache block each
Same for $A_{p'}$
For every further iteration x and y will be in cache while A will have „n" times the $\frac{n}{8}$ misses
This totals to $\underbrace{\frac{n^2}{8}}_{A} + \underbrace{\frac{n}{8}}_{x} + \underbrace{\frac{n}{8}}_{y} = \frac{n^2}{8} + \frac{n}{4}$ misses

2. Using the result of the previous task give a bound on the operational intensity in O notation (including very short explanation).

We have n^2 adds and n^2 mults.
Our transfered memory would be #misses * cache line size * size of double

$$\frac{2n^2}{\left(\frac{n^2}{8} + \frac{n}{4}\right) \cdot 8 \cdot 8} = O(1)$$

Problem 4 (18 points)

Given is the following Matlab function implementing a divide-and-conquer algorithm. The input T is an $n \times n$ matrix, where $n = 2^d$. The output F is also an $n \times n$ matrix. The auxiliary function `aux(A, B, C)` assumes all matrices A, B, C are $k \times k$ for some k and requires $2k^3$ many floating point operations.

```
function F = func(T)

n = size(T);

% base case: n = 1
if n < 2
  F = T(1,1)^2;
else
  m = n/2;
  u = 1:m;
  v = m+1:n;

  % recurse
a.{ F1 = func(T(u,u));
  { F3 = func(T(v,v));

b. C  = F1*T(u,v); % matrix multiplication
c. F2 = aux(T(u,u), T(v,v), C);

  F = [F1 F2; zeros(n-m,m) F3]; % no ops here
end
```

Compute the exact floating point operations count $C(n)$ for an $n \times n$ input T. **Show the derivation.**

The formula $f_k = ca^k + \sum_{i=0}^{k-1} a^i s_{k-i}$ solving $f_0 = c,\ f_k = af_{k-1} + s_k,\ k \geq 1$ may be helpful.

$$C(1) = 1, \quad C(n) = \underbrace{2\,C\left(\frac{n}{2}\right)}_{a.} + \underbrace{2\frac{n^3}{8}}_{b.} + \underbrace{2\frac{n^3}{8}}_{c.}$$

$$n = 2^d, \quad C(2^d) = f_d$$

$$f_0 = 1, \quad f_d = 2f_{d-1} + 2^{3d-1}$$

Using the provided formula:

$$f_d = 2^d + \sum_{i=0}^{d-1} 2^i \cdot 2^{3d-3i-1}$$

$$= 2^d + 2^{3d-1} \sum_{i=0}^{d-1} 2^{-2i}$$

Using the formula for the sum of a geometric series:

$$f_d = 2^d + 2^{3d-1} \cdot \frac{4}{3}\left(\frac{4^d - 1}{4^d}\right)$$

$$= 2^d + \frac{2^{d+1}}{3}\left(2^{2d} - 1\right) = 2^d + \frac{2^{3d+1}}{3} - \frac{2^{d+1}}{3}$$

$$\downarrow \quad n = 2^d$$

$$C(n) = n + \frac{2}{3}n^3 - \frac{2}{3}n = \frac{2}{3}n^3 + \frac{n}{3}$$

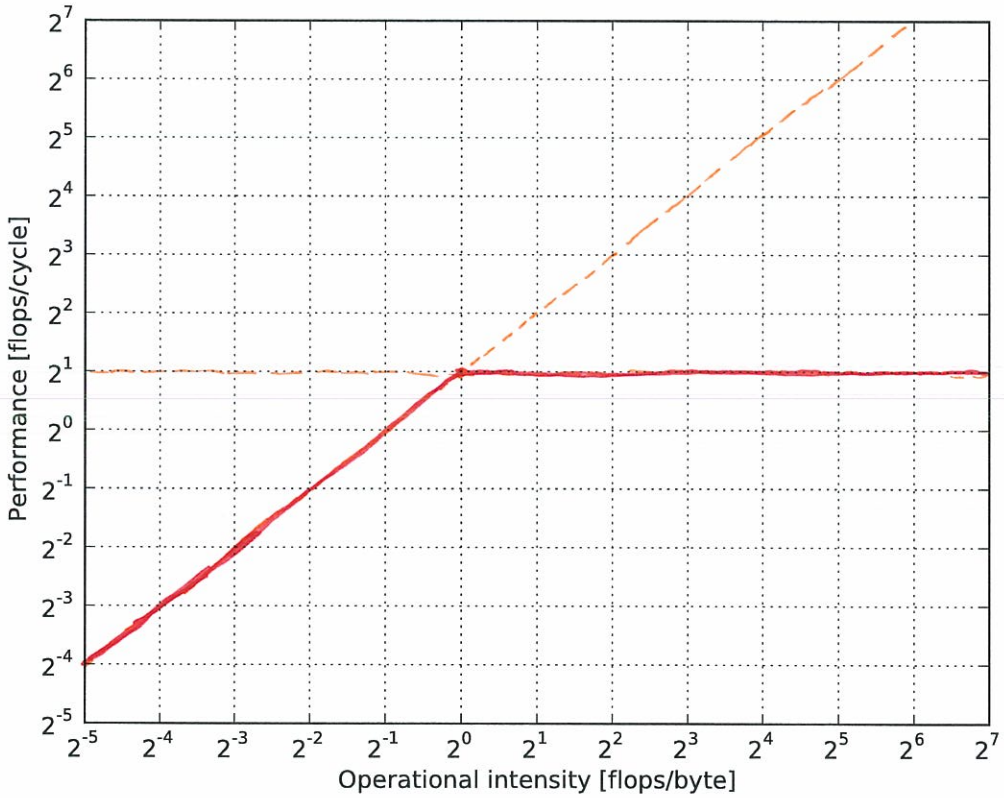
Problem 5 (16 points)

Assume you are using a system with the following features:

- A processor with a peak performance of 2 Gflop/s (double precision), and a CPU frequency of 1 GHz.
- The interconnection between CPU and main memory has a maximal bandwidth of 2 GB/s.

Answer the following questions:

1. Draw the roofline plot for this system. The units for x-axis and y-axis are performance in flops/byte and operational intensity in flops/cycle, both in log scale. The plot will contain two lines determining upper bounds on the achievable performance.



2. Assume the CPU frequency is increased to 2 GHz. How does this affect the two lines in 1.? (No need to redraw, just say.)

We would have a shorter cycle and therefor a smaller $\frac{\text{Bytes}}{\text{cycle}}$ ratio.

The diagonal line would cross the $2\frac{f}{c}$ horizontal line at $2\frac{f}{B}$

3. Consider the following code:

```
double v[64];
double sum = 0.;
double mul = 1.;

for(i = 0; i < 64; i++) {
  sum += 3*v[i];
  mul *= 2*(v[i]^2) + 5;
}
```

Assuming a cold cache:

(a) Compute the operational intensity assuming you performed scalar replacement on this code.

Assuming sum and mul are in registers

$$I = \frac{6 \cdot 64 \text{ flops}}{8 \cdot 64 \text{ Bytes}} = \frac{3}{4}\frac{f}{B}$$

(b) Based on the result: is the computation compute- or memory-bound (platform is still the one from the previous page) and why?

$$I < 1\frac{f}{B} \Rightarrow \text{memory-bound}$$

Problem 6 (21 points)

We consider the following program.

```
double A[4][4], B[4][4]

void transpose()
{
  int i, j;
  for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
      B[j][i] = A[i][j]
  return;
}
```

We assume a direct-mapped writeback cache of size 128 bytes and a cache block size of 16 bytes (2 doubles). We assume a cold cache. In the questions below only consider accesses to the arrays A and B. A is cache aligned, i.e., $A[0][0]$ would begin the first cache block. $B[0][0]$ is in memory directly after $A[3][3]$.
Answer the following

1. Where is spatial locality?
   There is spatial locality in A and also in B if after an iteration of the j loop B[j][i+1] is still in cache.
   There is of course spatial locality in the code.
2. Where is temporal locality?
   No element of the array is touched twice - so no temporal locality. Only i,j and the code are touched multiple times.

3. Determine the hit-miss sequence (something like HMMHHM...) for both A and B. It helps to draw the cache after each iteration of the i-loop and derive the sequence iteration by iteration. Show enough detail so we know how you did it.

i = 0

			Set	
1	*A[0][0]	A[0][1]	1	M
2	*B[0][0]	B[0][1]		M
3	A[0][0]	*A[0][1]		M
7	*A[0][2]	*A[0][3]	2	M,H
4	*B[1][0]	B[1][1]	3	M
			4	
5	*B[2][0]	B[2][1]	5	M
			6	
8	*B[3][0]	B[3][1]	7	M
			8	

↑ Access order

↑ Set

* Element accessed

---

i = 1

			Set	
	~~A[0][0]~~	A[0][1]	1	
2	B[0][0]	*B[0][1]		M
	A[0][2]	A[0][3]	2	
	~~B[1][0]~~	~~B[1][1]~~		
1,3	*A[1][0]	*A[1][1]	3	M,H
4	B[1][0]	*B[1][1]		M
5,7	*A[1][2]	*A[1][3]	4	M,H
6	B[2][0]	*B[2][1]	5	H
			6	
8	B[3][0]	*B[3][1]	7	H
			8	

↑ #Accessed

---

i = 2

	~~B[0][0]~~	~~B[0][1]~~	1	
	~~A[0][2]~~	~~A[0][3]~~	2	
2	*B[0][2]	B[0][3]		M
	~~B[1][0]~~	~~B[1][1]~~	3	
	~~A[1][2]~~	~~A[1][3]~~		
4	*B[1][2]	B[1][3]	4	M
	~~B[2][0]~~	~~B[2][1]~~	5	
1,3	*A[0][0]	*A[2][1]		M,H
	~~B[2][0]~~	~~B~~		
5	*A[2][2]	A[2][3]	6	M,M,M
6	*B[2][2]	B[2][3]		
7	A[2][2]	*A[2][3]		
	B[3][0]	B[3][1]	7	
8	*B[3][2]	B[3][3]	8	M

i = 3

			Set	
	B[0][0]	B[0][1]	1	
2	B[0][2]	*B[0][3]	2	H
	B[1][0]	B[1][1]	3	
4	B[1][2]	*B[1][3]	4	H
	A[2][0]	A[2][1]	5	
	~~A[2][2]~~	~~A[2][3]~~	6	
6	B[2][2]	*B[2][3]		M
	~~B[3][0]~~	~~B[3][1]~~	7	
1,3	*A[3][0]	*A[3][1]		H,H
	~~B[3][2]~~	~~B[3][3]~~	8	
5,7	*A[3][2]	*A[3][3]		M,H
8	B[3][2]	B[3][3]		M

Total sequence

A: MMMH MHMH MHMM MHMH

B: MMMM MMHH MMMM HHMM

10 Hits / 16 Accesses