6.172 Quiz 2

Adriano Hernandez

TOTAL POINTS

71.5 / 80

QUESTION 1

T/F 14 pts

1.1 2 / 2

✓ **+ 2 pts Correct (true)**

+ 0 pts Incorrect

1.2 2 / 2

✓ **+ 2 pts Correct (false)**

+ 0 pts Incorrect

1.3 2 / 2

✓ **+ 2 pts False**

+ 0 pts True/Blank

1.4 2 / 2

✓ **+ 2 pts False**

+ 0 pts True

1.5 2 / 2

✓ **+ 2 pts False**

+ 0 pts True

1.6 2 / 2

✓ **+ 2 pts True**

+ 0 pts False

1.7 2 / 2

✓ **+ 2 pts Correct (true)**

+ 0 pts Incorrect

QUESTION 2

Multiple Choice 18 pts

2.1 3 / 3

+ 1 pts A

+ 1 pts B

✓ **+ 3 pts C**

+ 0 pts D

+ 0 pts E

+ 0 pts F

2.2 3 / 3

+ 1.5 pts A

+ 2 pts B

+ 0 pts C

✓ **+ 3 pts D**

+ 0 pts E

+ 0 pts F

2.3 1.5 / 3

+ 3 pts A

+ 0 pts B

+ 0 pts C

✓ **+ 1.5 pts D**

+ 1.5 pts E

+ 0 pts F

+ 0 pts G

+ 0 pts H

2.4 2 / 3

+ 0 pts A

+ 0 pts B

+ 1 pts C

✓ **+ 1 pts D**

+ 3 pts E

4 bytes —— 122 bytes padding —— 2 bytes

vertical bar (|) denotes cache line, ... denotes padding in the pictorial explanation below

| x ... | ... y | x ... | ... y | and so on

+ 0 pts F

+ 1 Point adjustment

Notes show understanding of exactly what's needed, so deserves extra points. Only detail missed is that elements are not cache-aligned by default.

2.5 3 / 3

✓ **+ 3 pts A**

+ 0 pts B

2.6 3 / 3

✓ **+ 3 pts A**

+ 0 pts B

QUESTION 3

YSWEL 18 pts

3.1 3 / 3

✓ **+ 3 pts Correct: $$\Theta(b^{d/2})$$**

+ 0 pts Incorrect

+ 3 pts Correct: $$\Theta(2b^{d/2})$$

+ 2 pts Incorrect reasoning but correct bound $$3b^{d/2} = \Theta(b^{d/2})$$

3.2 4 / 4

✓ **+ 4 pts Correct explanation**

+ 0 pts Incorrect

3.3 3 / 3

✓ **+ 3 pts Correct: $$\Theta\left(\left(\frac{b}{3}\right)^{d/2}\right)$$**

+ 0 pts Incorrect

- 1 pts did not simplify the asymptotic expression correctly

3.4 0 / 4

✓ **+ 0 pts A**

+ 0 pts B

+ 0 pts C

+ 0 pts D

+ 4 pts E

3.5 2 / 4

+ 4 pts $$\Theta((\dfrac{b}{k+1})^{d/2})$$

+ 0 pts Incorrect

✓ **+ 2 pts Partial: Correct Span $$\Theta((k+1)^{d/2})$$**

+ 2 pts $$\Theta((b/k)^{d/2})$$

QUESTION 4

Cache-oblivious 16 pts

4.1 4 / 4

✓ **+ 1 pts a = 4**

✓ **+ 1 pts b = 2**

✓ **+ 1 pts $$X = \frac{n^2}{B}$$ or $$X = \frac{M}{B}$$**

✓ **+ 1 pts Y = 1**

4.2 4 / 4

✓ **+ 4 pts Correct**

+ 0 pts Incorrect

4.3 4 / 4

✓ **+ 4 pts Correct**

+ 0 pts Incorrect

4.4 4 / 4

✓ **+ 4 pts #leaves x misses_per_leaf = $$\Theta(\frac{n^2}{B})$$**

+ 2 pts Correct answer: $$\Theta(\frac{n^2}{B})$$. No justification or incorrect justification.

+ 2 pts Right logic: #leaves x misses_per_leaf, but wrong final answer.

+ 0 pts Incorrect.

QUESTION 5

Compilers 14 pts

5.1 3 / 3

✓ **+ 3 pts A**

+ 1 pts B

+ 1 pts C

+ 0 pts D

+ 0 pts E

5.2 4 / 4

✓ **+ 4 pts** **int main(argc, char* argv[]) {**
return 49;
}

The compiler would inline the function foo, after which the arguments to foo could be identified as constants, so following a constant folding and propagation pass the function can simply return a constant.

+ 0 pts Incorrect

+ 1 pts Correctly inlined

+ 1.5 pts One fewer multiplication or other algebraic simplification, perhaps combined with common subexpression elimination

+ 3 pts Not fully simplified. Importantly, does not need to perform any addition or multiplications at all.

+ 1 pts Some constant propagation

+ 3.9 pts Arithmetic error

5.3 3 / 3

+ 0 pts A

✓ **+ 3 pts** **B**

+ 0 pts C

+ 0 pts D

+ 0 pts E

5.4 4 / 4

✓ **+ 4 pts** **The compiler has detected that the loop is performing a summation from 0 to $$x$$ and has converted it from a $$\Theta(n)$$-time loop to $$\Theta(1)$$ by calculating $$x(x-1)/2 + x$$, the closed form of the summation. Extra info (not part of answer) The reason it doesn't use $$x(x+1)/2$$ is because the compiler cannot introduce overflows, which it would in the case $$x=INT\_MAX$$:)**

+ 2 pts $$x(x+1)/2$$

+ 0 pts Incorrect

+ 2 pts $$x(x-1)/2$$

+ 1.5 pts Partial explanation. Mentioning use of a closed form.

+ 1 pts Partial explanation. Identifying constant time summation.

Performance Engineering of Software Systems Thursday, December 2, 2021
Massachusetts Institute of Technology 6.172
Saman Amarasinghe, Charles E. Leiserson, and Jonathan Ragan-Kelley Quiz 2

Quiz 2

- DO NOT OPEN this quiz booklet until you are instructed to do so. This quiz is closed book, but you may use one handwritten, double-sided 8 1/2″ × 11″ crib sheet. Please put all personal items unrelated to the exam on the floor.
- Please read these instructions carefully. This quiz booklet contains 11 pages, including this one, but excluding 3 pages of appendix (an x86 assembly guide, an LLVM IR guide, and the master theorem) and 1 piece of scratch paper. You have 80 minutes to earn 80 points.
- Please write your name and Kerberos on this cover sheet in the space provided. When the quiz begins, please additionally write your name or Kerberos at the top of each page, since the pages will be separated during grading.
- For multiple-choice questions, circle the label corresponding to the best answer. When multiple answers are correct but one is the best, selecting a lesser answer may receive partial credit. Circling multiple labels will receive zero points, however.
- For True/False and multiple-choice questions, you need not explain your answer. You may receive partial credit, however, if you provide justification in the event that your answer is incorrect or if you have a reasonable interpretation that doesn't match the "official" answer.
- Good luck!

Name: Adriano Hernandez Kerberos: adrianoh

Problem	Title	Parts	Points	Score	Grader
0	Reading and Following Directions	∞	NaN		
1	True or False	7	14		
2	Multiple Choice	6	18		
3	Young Siblings Wait Even Longer	5	18		
4	Cache-Oblivious Matrix Transpose	4	16		
5	Clever Compilers	4	14		
	Total		**80**		

1 True or False (7 parts, 14 points)

Circle the correct answer for each statement below.

1.1 Parallel programs generally suffer from lack of sufficient memory bandwidth more than serial programs do.

True False

(They use more mem at any given time → bottleneck at LLC & RAM)

1.2 Since sequential consistency allows reasoning about a parallel program by interleaving the serial chains of instructions issued by processors, most modern multicores adopt this memory model.

True **False**

x86-64 does NOT & it's ubiquitous

1.3 In exchange for faster search time, alpha-beta pruning potentially finds a less optimal move than a full minimax tree search to the same depth.

True **False**

Don't check if they could find a better move against you → more search is NOT useful

1.4 Since the cost of a steal is large, the Cilk runtime system generally prioritizes optimizing steals, even at the cost of potentially introducing extra work.

True **False**

steals are rare → amortized (ideally, usually)

1.5 When a Cilk worker encounters a `cilk_sync`, it waits until all children spawned from within its current frame return.

True **False**

It is "out of work" & becomes a thief OR is the last one → continue up the stack

1.6 Yielding a thread that is waiting to acquire a lock potentially allows the processor to be used for other tasks, but the lock can sometimes be acquired more quickly if it spins instead of yielding.

True False

If you yield you'd have to guess when to try again (can be told)

1.7 Using memory fences, Peterson's algorithm can be adapted to implement mutual exclusion on processors that implement Total Store Order.

True False

You could put one between EVERY contiguous pair of lines

↳ "seq cons."

2 Multiple Choice (6 parts, 18 points)

2.1 Rank the upper bound of potential blowup with P processors from least to greatest between different parallel memory allocation strategies: global heap (GH), local heaps (LH), local heaps with ownership (LHWO).

- **A** GH < ~~LH = LHWO~~.
- **B** GH < LH < ~~LHWO~~.
- **C** GH < LHWO < LH.
- **D** ~~LH < GH~~ < LHWO.
- **E** ~~LH < LHWO~~ = GH.
- **F** None of the above.

middle ≥ uae none, it's just slow worst = mem drift

C!

GH best, LH worst

2.2 What advantages does reference counting provide for garbage collection?

- **A** New allocations are more likely to come from the L1-cache. ← true relative to stop & copy
- **B** It is simpler to implement than a mark-and-sweep garbage collector. (reusal?) (just ++ & -- & free)
- **C** ~~It can handle cycles of pointers without traversing the entire pointer graph.~~ NO it cannot AT ALL
- **D** A & B.
- **E** ~~B & C.~~
- **F** None of the above.

2.3 Let `int **G` be a graph stored as an adjacency matrix. `G[u][v] = 1` if an edge (u, v) exists in the graph, and `G[u][v] = 0` otherwise. Theodore Threadripper writes a parallel program which accesses `G` as a read-only data structure and runs it on a multicore computer with private ideal, fully associative caches per core. If he changes the program to represent the adjacency matrix as a bitmap instead, what changes in caching phenomena might he observe?

$O(u^2)$ space

LRU ~

Assume bitmap is packed

- **A** Fewer capacity misses, because the data is smaller. ← more fits in cache = cap↑
- **B** More sharing misses, because of false sharing.
- **C** ~~Fewer conflict misses, because fewer addresses of the data map to the same set.~~ → # for LRU or ideal ass.
- **D** A & B.
- **E** ~~A & C.~~
- **F** ~~B & C.~~
- **G** ~~All of the above.~~
- **H** None of the above.

more objects fit in a cache like & mult Proc may use

* **2.4** Abby defines a cache-aligned array `s` of size `N` with elements of type `MyStruct`:

BLOCK aligned

```
01 struct MyStruct {
02     uint32_t x;
03     uint16_t y;
04 } MyStruct;
05
06 MyStruct s[N];
```

Abby then runs a program similar to the following pseudocode on an x86-64 processor with 64-byte cache lines:

```
07 cilk_spawn {
08   cilk_for(...) {
09     read s[i].x;
10   }
11 }
12 cilk_for(...) {
13   write s[i].y;
14 }
15 cilk_sync;
```

Thinking about cache coherence, Abby considers adding a field named `padding` between `x` and `y` to eliminate false sharing. What is the size required for padding to remove false sharing?

~~A~~ 0 — there is actually no false sharing.

~~B~~ 2 bytes

~~C~~ 58 bytes

(D) 60 bytes

~~E~~ 122 bytes

F None of the above

The array is cache-aligned → each elem gets its own 64-byte cache block

Same block

32 → 4 bytes → need 60 more

then need 64-2 = 62 bytes for y for the next elem

122 total bytes, 60 in padding between x & y, 62 implicitly AFTER y by the cache-alignment of the array

correct if you include after y before next elem

The next two questions refer to the following code, which attempts to build a histogram of `N` `uint8_t` values in `data[]`. It computes the histogram in parallel, using the atomic `CompareAndSwap()` operation for synchronization.

```
16 bool CompareAndSwap(int *ptr, int expected_val, int new_val);
17
18 void histogram(uint8_t *data, int N, int hist[256]) {
19     cilk_for (int i = 0; i < N; i++) {
20         int old_val, new_val;
21         do {
22             old_val = hist[data[i]];
23             new_val = old_val + 1;
24         } while (!CompareAndSwap(&hist[data[i]], old_val, new_val));
25     }
26 }
```

because I think cilk would find it since it has to re lock. conf

* **2.5** Does `histogram()` contain a determinacy race?

A (A) Yes, it has a determinacy race.

(B) No, it does not have a determinacy race.

YES according to cilk, but logically no, not really? Not sure what "race" is

I think you mean to say "two code blocks in diff threads access (WRITE TO) the same memory" which is true, but this is NOT a race

the val is determined since CAS atomically locks

with ANY form of mutex

2.6 Does `histogram()` always produce the same results in array `hist[]` when given the same arguments?

(A) Yes, it always produces the same result.

B No, it may produce different results from run to run.

It is correct ✓

in Collegial speech

(only 1 increment per elem)

3 Young Siblings Wait Even Longer (5 parts, 18 points)

Recall that Knuth and Moore proved that for a best-ordered minimax search tree with branching factor b and depth d, alpha-beta pruning visits approximately $2b^{d/2}$ nodes in total. The ***young-siblings-wait-even-longer (YSWEL)*** algorithm modifies the young-siblings-wait (YSW) algorithm as follows. Instead of processing just the first child of the search tree before processing the remaining child subtrees in parallel, as YSW does, YSWEL processes the first *two* children in series before processing the remaining child subtrees in parallel.

For the questions below, assume that the processing of each node in the search tree takes the same (constant) amount of time on a processing core.

* **3.1** Give a Θ-bound expressed in terms of b and d on the total work of YSWEL on a best-ordered search tree.

ASSUME YSW total work = $\Theta(b^{d/2})$ (# visits that many nodes)
⇒ this adds at most $\Theta(1)$ more work per node, so
its ≤ 2x YSW time = $\Theta(b^{d/2})$

* **3.2** Explain in a few sentences why the total span of YSWEL on a best-ordered search tree is $\Theta(3^{d/2})$.

$T_\infty = 3T_\infty(\frac{h}{4}-1)+\Theta(1)$ if $h>0$, else $=\Theta(1)$

You take the time to do the 1st, then the 2nd sib, then the rest in parallel. This means $T_\infty = T_{\infty\text{-1st}} + T_{\infty\text{-2nd}} + T_{\infty\text{-rest}}$
⇒ its a (recursive) tree of the height of the comp tree = $d/2$ (from branch factor 3 NOT b) except at a leaf its $\Theta(1)$.
∴ grow geom. to leaf nodes = $3^{d/2}$ leaves = $\Theta(3^{d/2})$ total

3.3 What is the asymptotic parallelism, expressed in terms of b and d, of YSWEL on a best-ordered tree?

$$\text{Parallelism} = \frac{T_1}{T_\infty} = \frac{\Theta(b^{d/2})}{\Theta(3^{d/2})} = \Theta\left(\frac{b}{3}\right)^{d/2}$$

naturally its LESS parallel

$3^5 \approx 81 \cdot 3 \lesssim 2k$

$2^{32} \geq 1$ bill

* **3.4** Suppose that when YSWEL is run on a single processing core to search a given best-ordered search tree of depth 10, it visits about one billion nodes. About how much parallelism does the computation contain? Circle the label of the geometrically closest answer:

- (**A**) 200 (circled)
- **B** 2000
- **C** 20,000
- **D** 200,000
- **E** 2,000,000

\# nodes → find (worst) depth → find branch factor → find $\frac{b}{3}$ → take to pow → pick nearest #

$d = 10$

$2b^{d/2} \approx 10^9 \approx 2^{30}$

... ?!

$2b^5 \approx 2^{30}$

take 3 2's for each $2^{10} \to 2^9$

$2^{26} \lesssim b^5 \lesssim 2^{29}$

$2^{25} \lesssim b^5 \Rightarrow b \gtrsim 5$

$b^5 \lesssim 2^{29} \leq 2^{30}$

$\to b \lesssim 6$

$\frac{2}{3} \leq \frac{b}{3} \leq 2$

~~raise to 10~~

$\frac{b}{3} \leq 2$?! 5?! 32?!

$\frac{5^{10}}{3^{10}} \leq 2^{10}$ $\quad 5 \leq b \leq 6$?!

$2^{10} = 1024 \geq 1000$

$\lesssim 1k$ Small #?! (hard lower)

* **3.5** What is the parallelism if we generalize YSWEL to visit the first $k < b$ subchildren in series and then the remaining children in parallel? Express your answer in terms of b, d, and k.

Span would be $\sum_{i=1}^{k} T_\infty$ ith $+ T_\infty$ rest

$\Rightarrow$ branching factor of $k+1$

Recurrence becomes

$T_\infty(h) = kT_\infty(h-1) + T_\infty(h-1) + \Theta(1)$

$= (k+1)T_\infty(h-1) + \Theta(1)$

if $h \geq 1$ else $\Theta(1)$

$\Theta\left[\left(k+1\right)^{d/2}\right] = T_\infty$

Work is $\Theta(kb^{d/2})$ since we visit k extra nodes per node for children wait?

Parallelism "

$\frac{\Theta(kb^{d/2})}{\Theta((k+1)^{d/2})} = \Theta\left(\left(\frac{b}{k+1}\right)^{d/2} k\right)$

for small $k \approx \Theta\left(\left(\frac{b}{k+1}\right)^{d/2}\right)$

(it goes down as you add ops in series, naturally)

4 Cache-Oblivious Matrix Transpose (4 parts, 16 points)

Carl F. Caachoek writes the following cache-oblivious code to transpose an $n \times n$ square matrix, where n is an exact power of 2.

```
27 void mtranspose(double *A_T, double *A, int n, int row_size) {
28   if (n == 1) {
29     A_T[0] = A[0];
30   } else {
31     mtranspose(A_T,                          A,                          n/2, row_size);
32     mtranspose(A_T + (n/2 + row_size * n/2), A + (n/2 + row_size * n/2), n/2, row_size);
33     mtranspose(A_T + n/2,                    A + (row_size * n/2),       n/2, row_size);
34     mtranspose(A_T + (row_size * n/2),       A + n/2,                    n/2, row_size);
35   }
36 }
```

Let's analyze `mtranspose()` assuming a tall, ideal cache of size $\mathcal{M}$ and a cache-line size of $\mathcal{B}$. The asymptotic number of cache misses incurred by `mtranspose()` when transposing an $n \times n$ matrix can be expressed as a recurrence of the form

$$Q(n) = \begin{cases} \Theta(X) & \text{if } n^2 \le \gamma \mathcal{M}, \\ a\,Q(n/b) + \Theta(Y) & \text{otherwise}, \end{cases} \tag{1}$$

where $\gamma < 1$ is a positive constant.

if $n^2 \le \gamma M$ then the entire mat fits
=> one miss per block → $\frac{n^2}{B}$ (cold load) then just reuse

4.1 Specify the constants a and b in Equation (1), and give expressions for X and Y.

$a = 4$ (four subcalls)

$b = 2$ (each $\frac{n}{2}$)

$X = \frac{n^2}{B} \alpha \frac{M}{B}$

$Y = 1$ (one, or a const #)

~~NO code accesses~~

Stack might miss, ptrs in func call args

doesn't make a diff. cuz leaf-binary

height of recurrence NOT computation

4.2 Let h be the height of the recursion tree for the recurrence in Equation (1). Explain why $h \approx \lg n - \lg \sqrt{\gamma M}$.

leaves when $n^2 \leq \gamma M \Leftrightarrow n \leq \sqrt{\gamma M}$
because misses below the first recursion level of that that is true all sum into the same $\frac{n^2}{B}$ term. The TOTAL height of the computation tree is $\lg n$ since n values each level. The height then = (TOTAL − amount lost to the leaves) = $\lg n - \lg(2^{\sqrt{\gamma M}}) = \lg n - \lg\sqrt{\gamma M}$

i.e. the base case occurs on the 1st level of $n^2 \leq \gamma M$

* **4.3** Argue that the asymptotic number of leaves of the recursion tree is $\Theta(n^2/M)$.

(can also do in 4)

$\log_2 4 = 2 \rightarrow$ by master theorem the tree's sum is growing geometrically so it is LEAF-heavy. There are 4^{height} leaves (branch factor is 4)

use algebra

so their sum approxes the total

$4^{\text{height}} = 2^{2\,\text{height}} = 2^{2\lg n - \lg\sqrt{\gamma M}} = 2^{\lg n^2 - \lg\sqrt{\gamma M}} = \frac{n^2}{\sqrt{\gamma M}} = \Theta\left(\frac{n^2}{M}\right)$ ■

* **4.4** Asymptotically, how many cache misses does `matrix_transpose()` incur when transposing an $n \times n$ matrix? Briefly justify your answer.

Because it's leaf-heavy the work ≈ #leaves · work per leaf

$= \Theta\left(\frac{n^2}{M}\right)\Theta\left(\frac{M}{B}\right)$

$\frac{M}{B}$, but for n in the leaf stack frame $\leq \gamma M$

work = $\Theta\left(\frac{n^2}{B}\right)$

as if it were $\leq \gamma M$!

nice!

5 Clever Compilers (4 parts, 14 points)

Consider the following code:

```
37 int foo(int a, int b) {
38     return(a * a)+(b * b)+(a * b);
39 }
40
41 int main(int argc, char* argv[]) {
42     int x =  5;
43     int y = 3;
44
45     int z = foo(x, y);
46     return z;
47 }
```

25 + 9 + 15 → 49

***5.1** Suppose that function inlining is *not* enabled on this program, resulting in a loss of performance due to not optimizing away the function-call overhead. Of the following other optimization opportunities that might also be lost, which is likely to be the most important in this case?

propagate 5 & 3 to x & y else where

- (A) Constant folding and propagation.
- (B) Algebraic simplification.
- (C) Common-subexpression elimination.
- (D) Hoisting. no loop
- (E) Strength reduction.

A

NO (not an exp)

?!

replace a & b (which are x & y) to 5 & 3 → simplify to a constant

most important since it enables "algebraic" (really arithmetic) & other opts

5.2 Suppose that this code is compiled with full optimization, *including* function inlining. Briefly describe the logic of the optimized code that would result. Assume that the compiler performs all the optimizations in the list from part 5.1 perfectly.

return 49;

It would just return 49.

ex. opt.

replace all a's & b's w/ x's & y's

↓

replace x & y w/ 5 & 3

↓

algebraically simplify

↓

z = 49

↓

z not used exc. ret. so replace

Consider the following C code and the assembly code generated by compiling it:

```
48 static int sumTo(int x) {
49     int sum = 0;
50     for (int i = 0; i <= x; ++i) {
51         sum += i;
52     }
53     return sum;
54 }
55
56 int main(int argc, const char *argv[]) {
57     return sumTo(argc);
58 }
```

```
59 main:
60         testl   %edi, %edi
61         js      .LBB0_1
62         movl    %edi, %ecx
63         leal    -1(%rdi), %eax
64         imulq   %rcx, %rax
65         shrq    %rax
66         addl    %edi, %eax
67         retq
68 .LBB0_1:
69         xorl    %eax, %eax
70         retq
```

rdi or edi is x

if edi < 0

eax = rdi - 1

eax += edi

return

mult rcx by rax (mult rax) eax *

zero out eax

return

(rdi-1)(rdi)/2 + rdi

5.3 Which compiler optimization has provided the most benefit to the generated code?

- **A** Constant folding and propagation.
- **B** Algebraic simplification.
- **C** Common-subexpression elimination.
- **D** Hoisting.
- **E** Strength reduction.

like from rec.

5.4 Explain briefly how the generated assembly computes its result.

It figured out we were summing the first n digits (plus a zero which doesn't matter) and so returned the same value by using the algebraic identity

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{(n+1)(n)}{2} = \frac{(n-1)(n) + 2n}{2}$$

$$= \frac{(n-1)(n)}{2} + n$$

(n-1)(n) >> 1 + n