Mapreduce True / False

1. MapReduce programs running on a single core are usually faster than a simple serial implementation.

2. MapReduce works well on clusters with hundreds or thousands of machines.

3. MapReduce is the only framework used for writing large distributed programs.

4. MapReduce can sometimes give the wrong answer if a worker crashes.

5. A single Map task will usually have its map() method called many times.
Mapreduce True / False

1. MapReduce programs running on a single core are usually faster than a simple serial implementation. (FALSE)
2. MapReduce works well on clusters with hundreds or thousands of machines. (TRUE)
3. MapReduce is the only framework used for writing large distributed programs. (FALSE)
4. MapReduce can sometimes give the wrong answer if a worker crashes. (FALSE)
5. A single Map task will usually have its map() method called many times. (TRUE)
Caches

AMAT is influenced by three things - hit time, miss rate, and miss penalty. For each of the following changes, indicate which component will likely be improved:

1. using a second-level cache
2. using larger blocks
3. using a smaller L1$
4. using a larger L1$
5. using a more associative cache
Caches

1. using a second-level cache
   miss penalty
2. using larger blocks
   miss rate
3. using a smaller L1$
   hit time
4. using a larger L1$
   miss rate
5. using a more associative cache
   miss rate
AMAT Question
Suppose a MIPS program executes on a machine with a single data cache and a single instruction cache, and
- 20% of the executed instructions are loads or stores;
- the data cache hit rate is 95%
- the instruction cache hit rate is 99.9%
- both caches have a miss penalty of 10 cycles
- the instruction and data cache hit time is 1 cycle

a. How many memory references are there per executed instruction on average?
b. How many data cache misses are there per instruction?
c. How many instruction cache misses are there per instruction?
d. If there were no misses the CPI would be 1. What is the CPI actually?
e. Calculate the AMAT of the program.
a. How many memory references are there per executed instruction on average?
\[ 1 + 0.2 = 1.2 \]
b. How many data cache misses are there per instruction?
\[ 0.2 \times (1 - 0.95) = 0.01 \]
c. How many instruction cache misses are there per instruction?
\[ 1 \times (1 - 0.999) = 0.001 \]
d. If there were no misses the CPI would be 1. What is the CPI actually?
\[ CPI = CPI_{\text{ideal}} + E(\frac{\text{stall\_time}}{\text{inst}}) = 1 + 10 \times (0.01 + 0.001) = 1.11 \]
e. Calculate the AMAT of the program.
\[
\begin{align*}
AMAT &= 1 + P(\text{data\_access}) \times (P(\text{data\_miss}) \times \text{penalty}) \\
&\quad + P(\text{inst\_access}) \times (P(\text{inst\_miss}) \times \text{penalty}) \\
&= 1 + \frac{1}{6} \times (0.05 \times 10) + \frac{5}{6} \times (0.001 \times 10) = 1.0916666... 
\end{align*}
\]
MIPS
Consider visit_in_order, then translate into MIPS.

typedef struct node{
    int value;
    struct node* left;
    struct node* right;
} node;

void visit_in_order(node *root, void (*visit)(node*)) {
    if (root) {
        visit_in_order(root->left, visit);
        visit(root);
        visit_in_order(root->right, visit);
    }
}
vio:
    addiu $sp $sp -12
    sw $s0 0($sp)
    sw $s1 4($sp)
    sw $ra 8($sp)

    #nullcheck
    beq $0 $a0 Exit
    move $s0 $a0
    move $s1 $a1

    #vist left
    lw $a0 4($s0)
    jal vio

    Exit:
    lw $s0 0($sp)
    lw $s1 4($sp)
    lw $ra 8($sp)
    addiu $sp $sp 12
    jr $ra

#visit myself
move $a0 $s0
jalr $s1

#visit right
lw $a0 8($s0)
move $a1 $s1
jal vio
Cache bits

How many bits are needed for tag, index and offset in a direct mapped cache with $2^m$ bytes and $2^n$ lines, with $p$-bit addressing?

Suppose the above cache has $x$ maintenance bits (valid, dirty, etc...). How many bits does it have total?

If a cache has dirty bits, then it is almost definitely write-_______, rather than write-_______. (choose between through and back).
Cache bits

How many bits are needed for tag, index and offset in a direct mapped cache with $2^m$ bytes and $2^n$ lines, with $p$-bit addressing?

Index = $n$, offset = $m-n$, tag = $p - (n + m - n) = p - m$

Suppose the above cache has $x$ maintenance bits (valid, dirty, etc...). How many bits does it have total?

$8 \times 2^m + (x + \text{tag}) \times 2^n = 2^{m+3} + (x + p - m) \times 2^n$

If a cache has dirty bits, then it is almost definitely write-back, rather than write-through. (choose between through and back).
Floating Point

Convert the following decimal fractions to IEEE 754 32-bit floating point numbers (i.e. give the bit patterns). Assume rounding is always to the nearest bit, with ties rounding up.

a. 126.375/1
b. 23.6/0
c. -5/16
d. 0/0
Floating Point

Convert the following decimal fractions to IEEE 754 32-bit floating point numbers (i.e. give the bit patterns). Assume rounding is always to the nearest bit, with ties rounding up.

a. $126.375/1$
   \[ \begin{array}{c}
   \text{0 10000101 111100110000000000000000} \\
   + 2^6 * (1 + 0.5 + 0.25 + 0.125 + ...) \\
   \end{array} \]

b. $23.6/0$
   \[ \begin{array}{c}
   \text{0 11111111 000000000000000000000000} (+\text{inf}) \\
   \end{array} \]

c. $-5/16$
   \[ \begin{array}{c}
   \text{1 01111101 010000000000000000000000} \\
   -0.3125 \\
   - 2^{-2} * (1 + 0.25) \\
   \end{array} \]

d. $0/0$
   \[ \begin{array}{c}
   \text{1 11111111 100000000000000000000000} (+\text{NaN}) \\
   \text{(not unique)} \\
   \end{array} \]
Fill in the missing entries in the following table. Each row describes a stage in the process of transforming source code into a running program.

<table>
<thead>
<tr>
<th>Input (type)</th>
<th>Program</th>
<th>Output (type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>assembly (text)</td>
<td></td>
<td>object file (binary)</td>
</tr>
<tr>
<td>assembly (text)</td>
<td></td>
<td>assembly (text)</td>
</tr>
<tr>
<td>executable (binary)</td>
<td></td>
<td>executable (binary)</td>
</tr>
<tr>
<td>executable (binary)</td>
<td></td>
<td>program output</td>
</tr>
<tr>
<td>interpreter</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Compilers, Assemblers, Linkers & Loaders

Fill in the missing entries in the following table. Each row describes a stage in the process of transforming source code into a running program.

<table>
<thead>
<tr>
<th>Input (type)</th>
<th>Program</th>
<th>Output (type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>assembly (text)</td>
<td>assembler</td>
<td>object file (binary)</td>
</tr>
<tr>
<td>source code (text)</td>
<td>compiler</td>
<td>assembly (text)</td>
</tr>
<tr>
<td>object file (binary)</td>
<td>linker</td>
<td>executable (binary)</td>
</tr>
<tr>
<td>executable (binary)</td>
<td>loader</td>
<td>program output</td>
</tr>
<tr>
<td>source code (text)</td>
<td>interpreter</td>
<td>program output</td>
</tr>
</tbody>
</table>
MIPS Assembly

Consider the following MIPS function foobar:

```mips
foobar:
    addu $v0 $0 $0
loop:
    andi $t0 $a0 1
    addu $v0 $v0 $t0
    srl $a0 $a0 1
    bne $a0 $0 loop
```

Give the output of foobar for the following calls:

<table>
<thead>
<tr>
<th>Call</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>foobar(0)</td>
<td>0</td>
</tr>
<tr>
<td>foobar(0xC1001021)</td>
<td>0xC1001021</td>
</tr>
<tr>
<td>foobar(0xFFFF)</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>foobar(0x8000)</td>
<td>0x8000</td>
</tr>
</tbody>
</table>

Briefly describe the behavior of foobar:
MIPS Assembly

Consider the following MIPS function foobar:

```
foobar:
    addu $v0 $0 $0
loop:
    andi $t0 $a0 1
    addu $v0 $v0 $t0
    srl $a0 $a0 1
    bne $a0
```

Give the output of foobar for the following calls:

<table>
<thead>
<tr>
<th>Call</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>foobar(0)</td>
<td>0</td>
</tr>
<tr>
<td>foobar(0xC1001021)</td>
<td>6</td>
</tr>
<tr>
<td>foobar(0xFFFF)</td>
<td>16</td>
</tr>
<tr>
<td>foobar(0x8000)</td>
<td>1</td>
</tr>
</tbody>
</table>

Briefly describe the behavior of foobar:

It counts how many bits are set in $a0
Cache Locality

You have an array defined as follows:

```c
#define N 1024
int matrix[N][N];
```

Your CPU has a byte addressed, 16KB direct-mapped cache with 64-byte cache lines/blots. To improve cache locality, you process the array `matrix` in blocks of size 5x5.

- Number of offset bits =
- Number of index bits =

Does a 5x5 block of `matrix` fit entirely in the cache? Explain.

Does the answer change if N = 1056?
You have an array defined as follows:

```c
#define N 1024
int matrix[N][N];
```

Your CPU has a byte addressed, 16KB direct-mapped cache with 64-byte cache lines/blocks. To improve cache locality, you process the array `matrix` in blocks of size 5x5.

- # offset bits = 6 \(2^6 = 64\) bytes/block
- # index bits = 8 \(2^{14}\) bytes/\(2^6\) bytes/block = \(2^8\) blocks

Does a 5x5 block of `matrix` fit entirely in the cache? Explain.

Does the answer change if \(N = 1056\)?
Cache Locality

You have an array defined as follows:

```c
#define N 1024
int matrix[N][N];
```

Your CPU has a byte addressed, 16KB direct-mapped cache with 64-byte cache lines/blocks. To improve cache locality, you process the array `matrix` in blocks of size 5x5.

- # offset bits = 6 \(2^6 = 64\) bytes/block
- # index bits = 8 \(2^{14}\) bytes/\(2^6\) bytes/block = \(2^8\) blocks

Does a 5x5 block of `matrix` fit entirely in the cache? Explain. No. Rows in the `matrix` are 4096 \(2^{12}\) bytes apart, so only the top two bits (13-14) of the index change moving across rows. Rows 1&5 have the same index bits, so they collide.

Does the answer change if \(N = 1056\)?
Cache Locality

You have an array defined as follows:

```c
#define N 1024
int matrix[N][N];
```

Your CPU has a byte addressed, 16KB direct-mapped cache with 64-byte cache lines/blocks. To improve cache locality, you process the array `matrix` in blocks of size 5x5.

- # offset bits = 6 \(2^6 = 64 \text{ bytes/block}\)
- # index bits = 8 \(2^{14} \text{ bytes/}2^6 \text{ bytes/block} = 2^8 \text{ blocks}\)

Does a 5x5 block of `matrix` fit entirely in the cache? Explain.

No. Rows in the `matrix` are 4096 \(2^{12}\) bytes apart, so only the top two bits (13-14) of the index change moving across rows. Rows 1&5 have the same index bits, so they collide.

Does the answer change if \(N = 1056\)?

Yes. There are 4224 \(2^{12} + 2^7\) bytes between rows, more index bits change so 5 adjacent rows map to 5 different cache blocks.
Consider the following C function dot_product, which computes the dot product of two vectors of integers, \( a \) and \( b \), of size \( n \):

\[
\text{int dot_product(int } *a, \text{ int } *b, \text{ unsigned } n) \\
\{ \\
\quad \text{int result } = 0; \\
\quad \text{while}(n \neq 0) \{ \\
\quad \quad \text{result } += (*a) \times (*b); \\
\quad \quad a++; \\
\quad \quad b++; \\
\quad \quad \text{n}--; \\
\quad \} \\
\quad \text{return result;} \\
\}
\]

Implement dot_product in MIPS.
MIPS, C, and Pointers

Consider the following C function `dot_product`, which computes the dot product of two vectors of integers, `a` and `b`, of size `n`:

```c
float dot_product(int *a, int *b, unsigned n)
{
    int result = 0;
    while(n != 0) {
        result += (*a) * (*b);
        a++;
        b++;
        n--;
    }
    return result;
}
```

Implement `dot_product` in MIPS:

```
dot_product:
    addu $v0 $0 $0 # result = 0
loop:
    beq $a2 $0 done # done looping?
    lw $t0 0($a0) # load a elem
    lw $t1 0($a1) # load b elem
    mul $t0 $t1 $t0 # assume this is 1 instr.
    addu $v0 $v0 $t0 # result += (*a) * (*b)
    addiu $a0 $a0 4
    addiu $a1 $a1 4
    addiu $a2 $a2 -1
    j loop
done:
    jr $ra
```
MIPS, C, and Pointers

Consider the following C function **dot_product**, which computes the dot product of two vectors of integers, a and b, of size n:

```c
float dot_product(int *a, int *b, unsigned n)
{
    int result = 0;
    while(n != 0) {
        result += (*a) * (*b);
        a++;
        b++;
        n--;
    }
    return result;
}
```

How many instructions are executed by dot_product (expressed as a function of argument n)?
MIPS, C, and Pointers

Consider the following C function `dot_product`, which computes the dot product of two vectors of integers, `a` and `b`, of size `n`:

```c
float dot_product(int *a, int *b, unsigned n)
{
    int result = 0;
    while (n != 0) {
        result += (*a) * (*b);
        a++;
        b++;
        n--;
    }
    return result;
}
```

```mips
addu $v0 $0 $0 # result = 0  
loop:     beq $a2 $0 done # done looping?  
        lw $t0 0($a0) # load a elem  
        lw $t1 0($a1) # load b elem  
        mul $t0 $t1 $t0 # assume this is 1 instr.  
        addu $v0 $v0 $t0 # result += (*a) * (*b)  
        addiu $a0 $a0 4  
        addiu $a1 $a1 4  
        addiu $a2 $a2 -1  
        j loop  
done:      jr $ra
```

How many instructions are executed by `dot_product` (expressed as a function of argument `n`)? $3 + n * 9$
la $a0 foobar
li $a1 5
jal baz
j ELSEWHERE

baz:
  addiu $sp $sp -4
  sw $ra 0($sp)
  jal hrm
  addiu $t1 $v0 48
  addu $t0 $0 $0
  addu $t3 $a0 $0

L1:
  beq $t0 $a1 L1done
  lw $t2 0($a0)
  sw $t2 0($t1)
  addiu $t0 $t0 1
  addiu $a0 $a0 4
  addiu $t1 $t1 4
  j L1

What value is returned by this call to baz?

L1done:
  lw $a0 0($t3)
  sll $0 $0 0
  sll $0 $0 0
  sll $0 $0 0
  sll $0 $0 0
  lw $ra 0($sp)
  addiu $sp $sp 4
  jr $ra

hrm:
  addiu $v0 $ra -4
  jr $ra

foobar:
  addu $v0 $0 $0

loop:
  andi $t0 $a0 1
  addu $v0 $v0 $t0
  srl $a0 $a0 1
  bne $a0 $0 loop
  jr $ra
la $a0 foobar
li $a1 5
jal baz
j ELSEWHERE

baz:
    addiu $sp $sp -4
    sw $ra 0($sp)
    jal hrm
    addiu $t1 $v0 48
    addu $t0 $0 $0
    addu $t3 $a0 $0

L1:
    beq $t0 $a1 L1done
    lw $t2 0($a0)
    sw $t2 0($t1)
    addiu $t0 $t0 1
    addiu $a0 $a0 4
    addiu $t1 $t1 4
    j L1

What value is returned by this call to baz?
3 = # of bits that are set in encoding of addu $v0 $0 $0

L1done:
    lw $a0 0($t3)
    sll $0 $0 0
    sll $0 $0 0
    sll $0 $0 0
    sll $0 $0 0
    lw $ra 0($sp)
    addiu $sp $sp 4
    jr $ra

hrm:
    addiu $v0 $ra -4
    jr $ra

foobar:
    addu $v0 $0 $0

loop:
    andi $t0 $a0 1
    addu $v0 $v0 $t0
    srl $a0 $a0 1
    bne $a0 $0 loop
    jr $ra
Recall the definition of the dynamic power dissipation of a CPU:

\[ P = CV^2F \]

where

- \( P = \) power
- \( C = \) capacitive load
- \( V = \) voltage
- \( F = \) frequency

Suppose we build a new, simpler processor that has 85% of the capacitive load of an older, more complex processor. The new processor can also reduce its voltage by 15% and its frequency by 15% relative to the older processor. What impact does this have on dynamic power dissipation?
Recall the definition of the dynamic power dissipation of a CPU:

\[ P = CV^2F \]

where

- \( P \) = power
- \( C \) = capacitive load
- \( V \) = voltage
- \( F \) = frequency

Suppose we build a new, simpler processor that has 85% of the capacitive load of an older, more complex processor. The new processor can also reduce its voltage by 15% and its frequency by 15% relative to the older processor. What impact does this have on dynamic power dissipation?

\[
\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = \frac{(C \times 0.85) \times (V \times 0.85)^2 \times (F \times 0.85)}{C \times V^2 \times F}
\]

\[ = 0.85^4 = 0.52 \]

The newer processor dissipates roughly half as much power.