

Foundations of Programming for High Performance Computing: CS2101. UWO, December, 14, 2013.

Student name:	
Student ID number:	

Guidelines. The exam is closed book and all notes are forbidden. The duration is 3 hours. There are 24 pages in the exam. The last four pages are blank: they can be used as scratch paper and will not be marked. The exam consists of 5 exercises located from Page 2 to Page 20. The mark allotment and a suggested time allotment are provided in the table below. All answers should be written in the *answer boxes*. No justifications for the answers are needed. You are expected to do this exam on your own without assistance from anyone else in the class. If possible, please avoid pencils and use pens with dark ink. Thank you.

Marks. Please, **do not write** anything in the table below.

Exercise	Maximum Mark	Expected Time				
1	20	30 min .				
2	20	30 min.				
3	20	30 min.				
4	20	30 min.				
4	20	30 min.				
TOTAL	100	2h30				

Exercise 1: multiple choice questions

In each case, **zero, one or more answers** may be correct; indicate **all correct answers**.

(1) Consider the following Julia function `f`:

```
function f(u,v)
    n=length(u)
    [u[i] - v[i] for i=1:n]
end
```

which assumes that u and v are vectors of equal length.

- (a) the function call `f(u, v)` prints “Hello World”
- (b) the function call `f(u, v)` does not return anything
- (c) the function call `f(u, v)` returns the difference $u - v$ of the vectors u and v
- (d) the function call `f(u, v)` returns the difference $u - v$ of the vectors u and v provided that their coefficients are integer numbers.

(2) Consider the following Julia function `B`:

```
function B(n,precision)
    x = n/2
    while abs(x^2 - n) > precision
        x = 0.5 * (x + n / x)
    end
    return x
end
```

- (a) the function call `B(9, 0.5)` returns 2.5
- (b) the function call `B(9, 0.5)` returns 3
- (c) the function call `B(9, 0.5)` returns -3
- (d) the function call `B(9, 0.25)` prints 3

(3) Assume that in a Julia session, one enters the following command line:

```
x=@async println("Hello")
```

Which of the following words will be ultimately displayed:

- (a) Task but not Hello
- (b) Hello but not Task
- (c) both Task and Hello

(4) Assume that in a Julia session, one enters successively the following two command lines:

```
a = @async 1+2  
fetch(a)
```

What is the value returned after entering the second command like?

- (a) Task
- (b) 3

(5) Assume that in a Julia session, one enters the following command line:

```
da = @parallel [2 * i for i = 1:10]
```

Which of the following statements are true?

- (a) da is an integer variable whose value is the sum $(2 \cdot 1) + \dots + (2 \cdot 10) = 110$
- (b) da is the array of 10 values [2 4 6 8 10 12 14 16 18 20]
- (c) da is the distributed array of 10 values [2 4 6 8 10 12 14 16 18 20]

(6) Consider the following Julia function and commands

```
function producer()  
    produce("start")  
    for n=1:2  
        produce(2n)  
    end  
    produce("stop")  
end
```

```
p = Task(producer);
```

```
[consume(p) for i=1:4]
```

After executing them successively, one sees the following output value

(a) "Task"

(b) "start"
2
4
"stop"

(c) "start"

(d) "producer"

(e) "consumer"

(7) Consider the distributed array `da` defined as follows

```
da = @parallel [2 * i for i = 1:10]
```

and assume that 2 workers, with numbers 2 and 3, own `da`: Worker 2 owns the first 5 elements and Worker 3 owns the last ones. What does the following command return?

```
fetch(@spawnat 2 da[3])
```

- (a) 6
- (b) 2
- (c) 3
- (d) Bound error

(8) Consider again the distributed array `da` defined as follows

```
da = @parallel [2 * i for i = 1:10]
```

and assume again that 2 workers, with numbers 2 and 3, own `da`: Worker 2 owns the first 5 elements and Worker 3 owns the last ones. What does the following command return?

```
[(@spawnat p sum(localpart(da))) for p=procs(da)]
```

- (a) An array of two remote references
- (b) [2 3]

(9) Consider again the distributed array `da` defined as follows

```
da = @parallel [2 * i for i = 1:10]
```

and assume again that 2 workers, with numbers 2 and 3, own `da`: Worker 2 owns the first 5 elements and Worker 3 owns the last ones. What does the following command return?

```
map(fetch, { (@spawnat p sum(localpart(da))) for p=procs(da) })
```

- (a) A 2-element array with entries 30 and 80
- (b) A 2-element array with remote references as entries

(10) Consider again the distributed array `da` defined as follows

```
da = @parallel [2 * i for i = 1:10]
```

and assume again that 2 workers, with numbers 2 and 3, own `da`: Worker 2 owns the first 5 elements and Worker 3 owns the last ones. What does the following command return?

```
reduce(+, map(fetch, { (@spawnat p sum(localpart(da))) for p=procs(da) } ))
```

- (a) 2
- (b) 110

(11) The Julia language provides a multiprocessing environment based on message passing to allow programs to run on multiple processors in shared or distributed memory.

- (a) True
- (b) False

(12) Consider the following Julia session where two methods are proposed for computing the square of a random matrix.

```
#method 1
A = rand(1000,1000)
Bref = @spawn A^2
fetch(Bref)

# method 2
Bref = @spawn rand(1000,1000)^2
fetch(Bref)
```

- (a) In the first method, a random matrix is constructed locally, then sent to another processor where it is squared.

- (b) In the first method, a random matrix is both constructed and squared on another processor.
- (c) In the second method, a random matrix is constructed locally, then sent to another processor where it is squared.
- (d) In the second method, a random matrix is both constructed and squared on another processor.

(13) Consider the following Julia session:

```
n = @parallel (+) for i=1:10
      i
end
```

After executing the above, the value of n is:

- (a) 10
- (b) 55
- (c) the number of processors involved in this Julia session
- (d) a remote reference

(14) Consider the following Julia session:

```
a = zeros(4);
@parallel for i=1:4
    a[i] = i
end
```

After executing the above, the coefficients of the array a are:

- (a) respectively equal to 1, 2, 3, 4.
- (b) all equal to 4
- (c) all equal to 0
- (d) all remote references

(15) Consider the following Julia session:

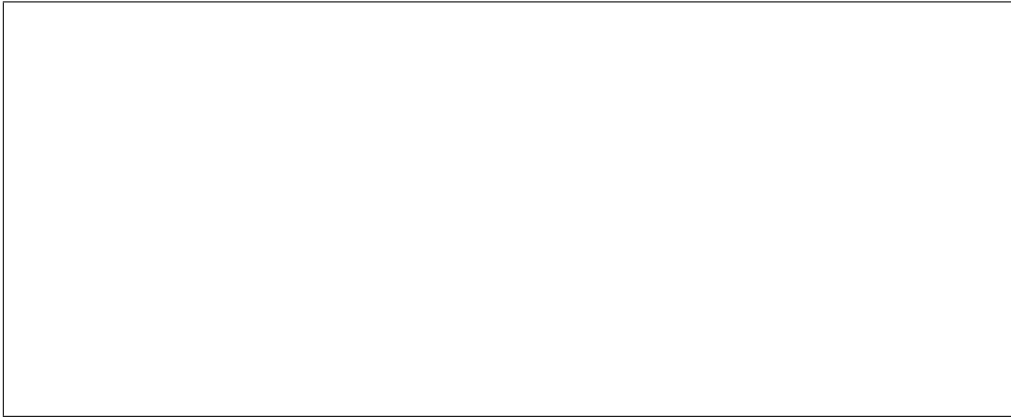
```
M = [rand(1000,1000) for i=1:4];  
R = [@spawnat i rank(M[i]) for i=1:4]
```

After executing the above, the coefficients of the array R are:

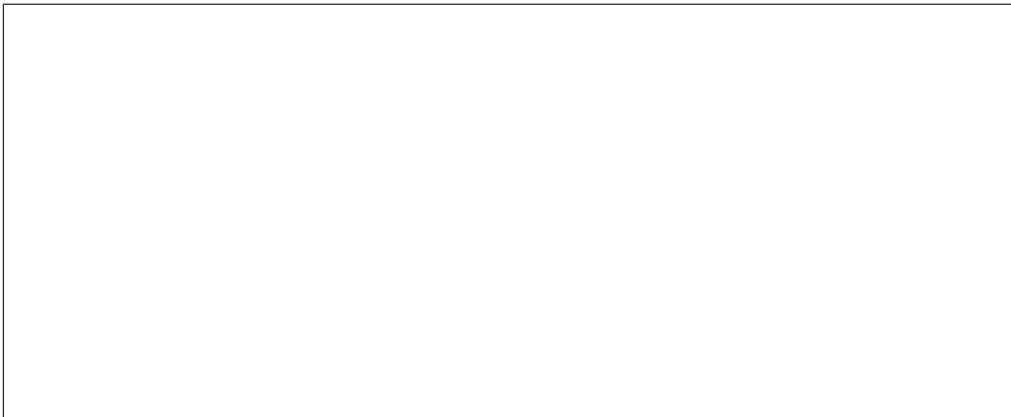
- (a) all integer numbers in the range 0:1000
- (b) all random matrices of format 100×100
- (c) all tasks (aka coroutines)
- (d) all remote references

Exercise 2: Julia questions with short answers

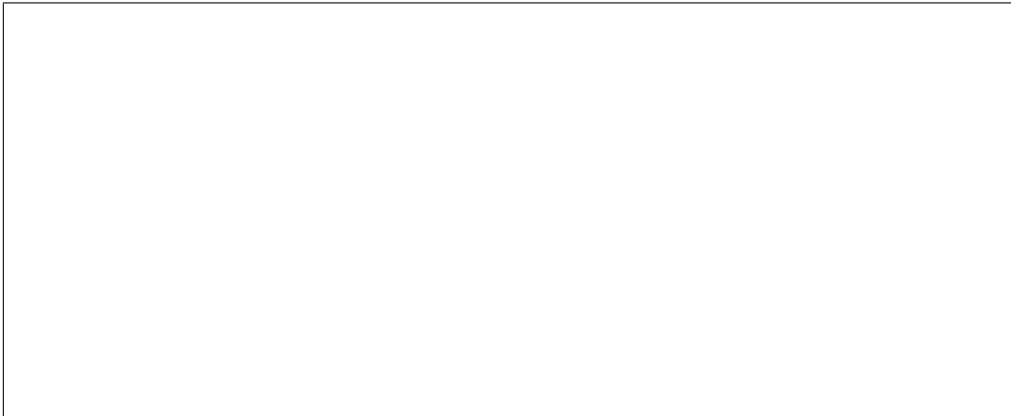
- (1) Write a Julia function called `KroneckerProduct` that takes as input two vectors `u` and `v` (whose coefficients could be integers, floats, etc.) of the same length and computes the square matrix `A` such that the element `A[i, j]` is the product `u[i] * v[j]`.



- (2) What is the output of the Julia function call `KroneckerProduct([1, 2, 3], [2, 3, 4])`



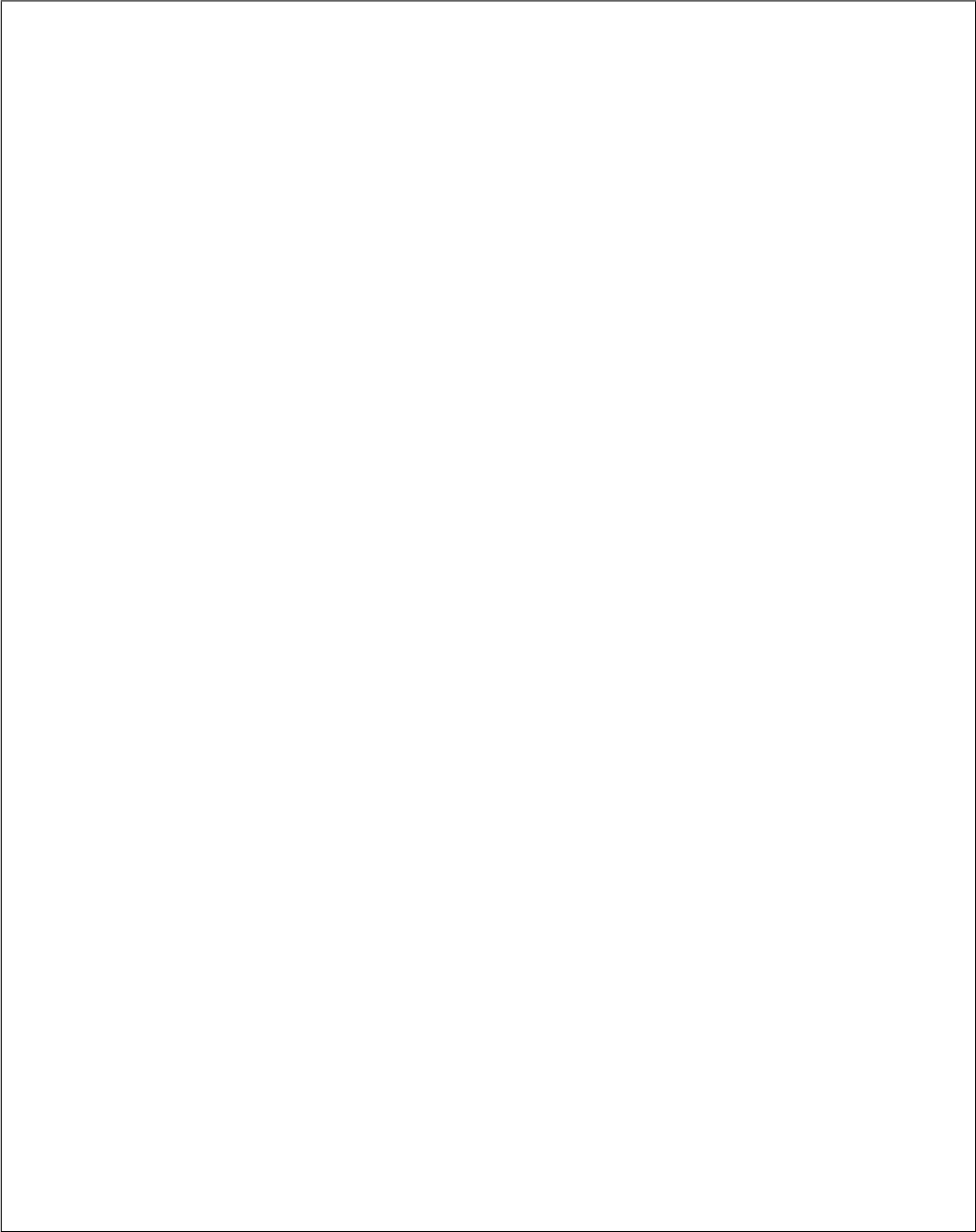
- (3) Write a Julia function called `DistributedKroneckerProduct` that takes as input two vectors `u` and `v` (whose coefficients could be integers, floats, etc.) of the same length and returns a distributed array `dA` such that the element `dA[i, j]` is the product `u[i] * v[j]`.



- (4) Consider the Julia's function below for multiplying two square matrices A and B of order n. Using Julia's construct `@spawnat` and `fetch` make a parallel version of that function that uses 4 processors.

```
function four_quadrant_mat_mul_serial(A, B, n)

    C = zeros(n, n)
    d = div(n, 2)
    e = d+1
    C[1:d, 1:d] = A[1:d, 1:d] * B[1:d, 1:d] +
                 A[1:d, e:n] * B[e:n, 1:d]
    C[1:d, e:n] = A[1:d, 1:d] * B[1:d, e:n] +
                 A[1:d, e:n] * B[e:n, e:n]
    C[e:n, 1:d] = A[e:n, 1:d] * B[1:d, 1:d] +
                 A[e:n, e:n] * B[e:n, 1:d]
    C[e:n, e:n] = A[e:n, 1:d] * B[1:d, e:n] +
                 A[e:n, e:n] * B[e:n, e:n]
    C
end
```



Exercise 3: writing a parallel Julia function

Consider a Julia session where one has defined a 1-D distributed array `da` of length `n`. The entries of `da` are all integer numbers. Recall that the function call `procs(da)` returns the processor numbers which own a part of `da`.

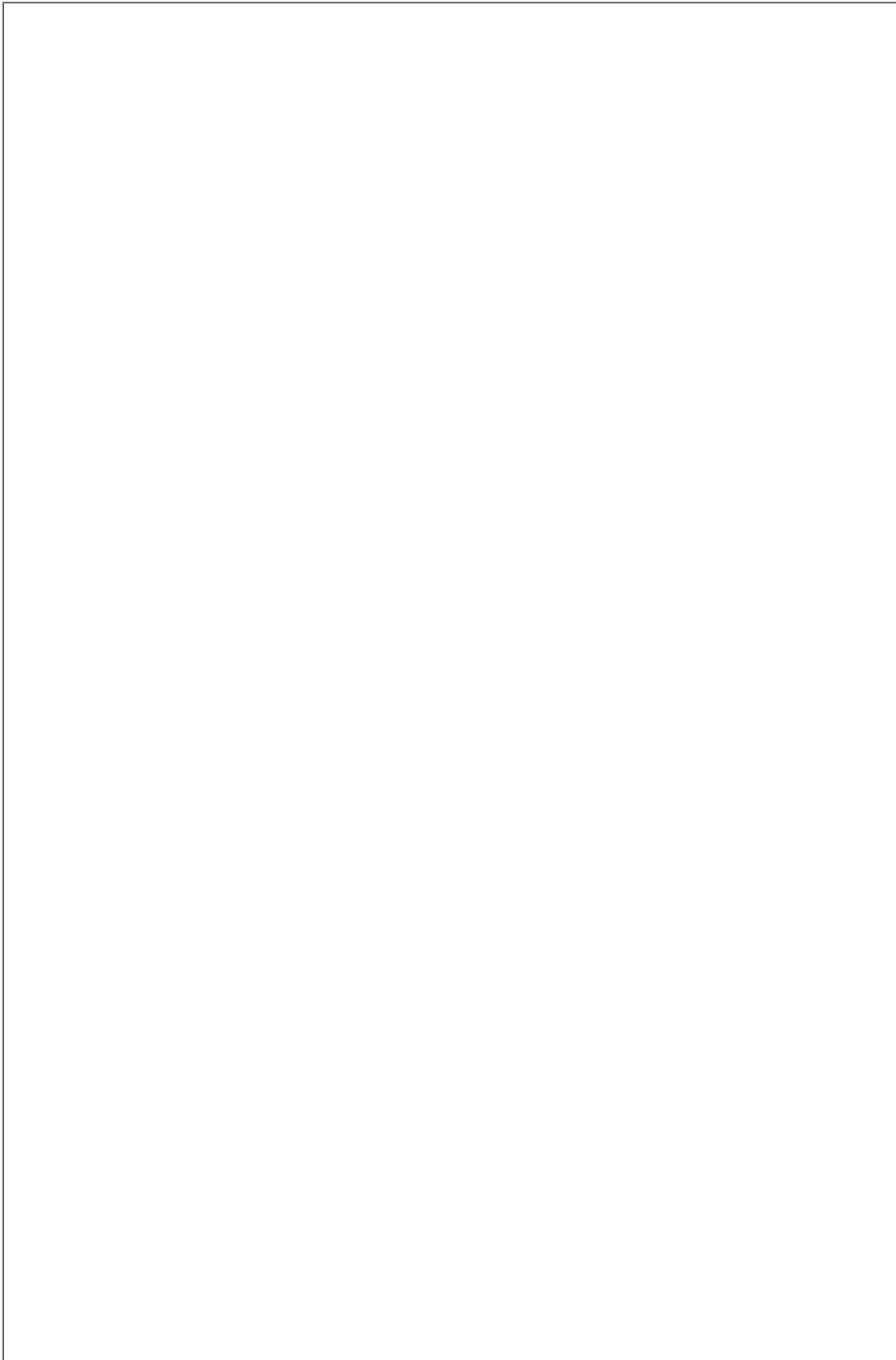
Let `f` be a Julia function which takes two integer numbers as input and returns another integer number. The function `f` is assumed to be associative, that is, $f(f(a, b), c) = f(a, f(b, c))$ for all integers `a, b, c`. Examples of such function `f` is *addition* and *multiplication*.

The objective of this exercise is to write a Julia function computing

$$f(a_1, f(a_2, f(a_3, \dots f(a_{n-1}, a_n) \dots))) \quad (1)$$

where a_i denotes the i -th element of `da`. In other words, you are required to write a Julia function called `produce` which *reduces* the function the function `f` on the distributed array `da` in a parallel fashion. Thus, this function must use parallel constructs like `@spawnat`, `fetch` and operation on distributed arrays like `procs` and `localpart`.





Exercise 4: writing a parallel Julia function

Consider a Julia session where one has defined a 1-D distributed array `da` of length `p`. Each entry of `da` is a 2-D array, each with `m` rows and `n` columns and with integer coefficients. Here's an example of construction of such array `da`

```
da = @parallel [ rand(Int32,2,2),10) for i=1:4]
4-element DArray{Array{Int32,2},1,Array{Array{Int32,2},1}}:
 2x2 Array{Int32,2}:
  7  3
 -6 -4
 2x2 Array{Int32,2}:
  7  0
 -5 -1
 2x2 Array{Int32,2}:
  6 -2
 -4 -6
 2x2 Array{Int32,2}:
 -2 -4
 -8  2
```

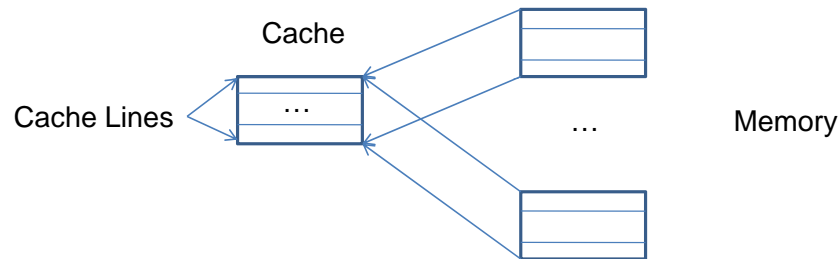
Each of these 2-D arrays encodes an *image* and we are interested in counting how many of these images have at least one coefficient equal to zero. Write a Julia function `ParallelCount` taking `da`, `p`, `m` and `n` as input and returning the total number of images which have at least one coefficient equal to zero:

- This function `ParallelCount` must use parallel constructs like `@spawnat`, `fetch` and operation on distributed arrays like `procs` and `localpart`.
- You will need also to write a Julia function `HasZeroCoefficients` which
 - takes as input a 2-D array `A` with `m` rows and `n` columns and
 - returns 1 if at least one coefficient of `A` is null and 0 otherwise.



Exercise 5: analyzing cache misses

The following four questions are using this simple cache memory; the same as in class.

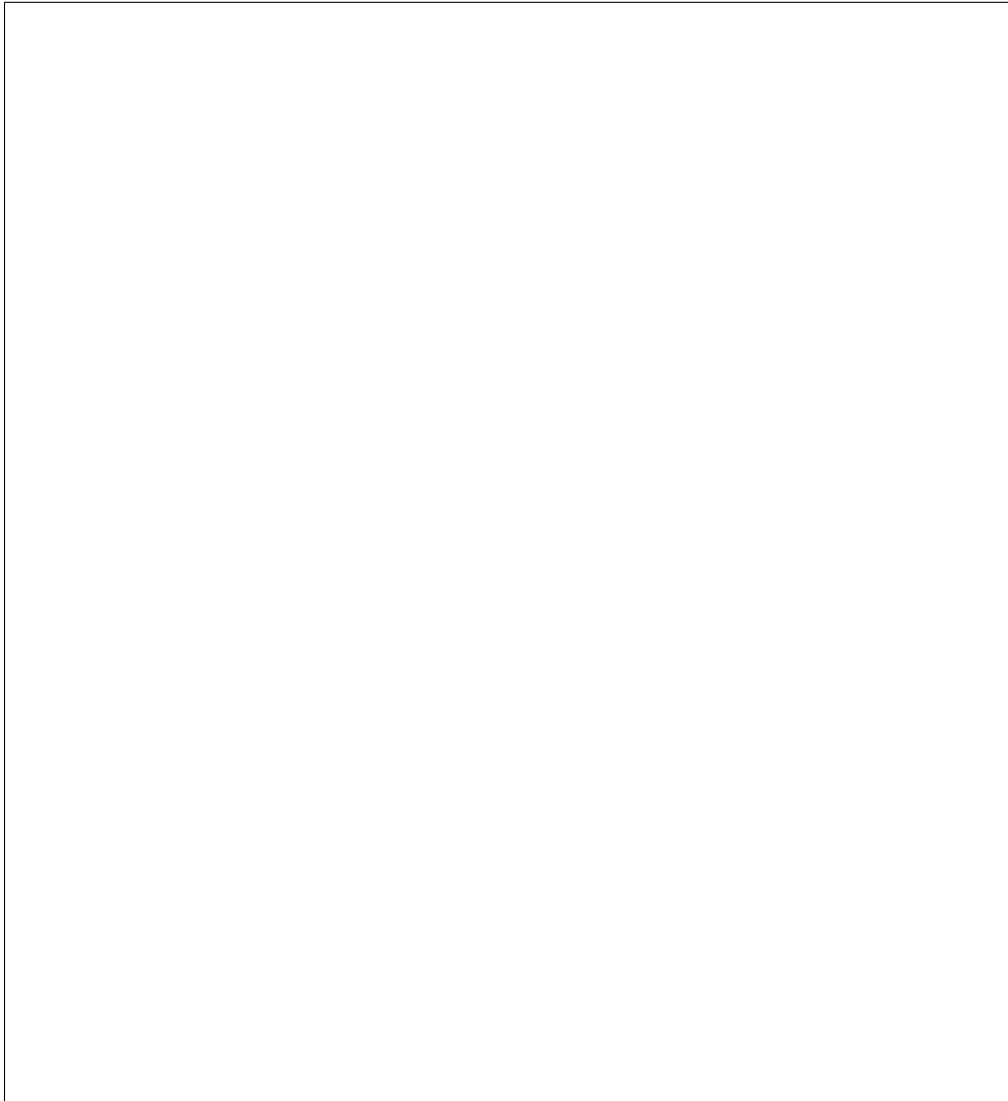


- Byte addressable memory
- The Cache has size 32Kbyte with direct mapping and 64 byte lines (512 lines); so the cache can fit $2^9 \times 2^4 = 2^{13}$ int.
- **Therefore**, successive 32Kbyte memory blocks can line up in cache.
- A cache access costs **1 cycle while**. a memory access costs **100 cycles**.
- How addresses map into cache
 - Bottom 6 bits are used as offset in a cache line,
 - Next 9 bits determine the cache line

Question 1.

```
// sizeof(int) = 4 and Array laid out sequentially in memory
#define S ((1<<20)*sizeof(int))
int A[S];
// Thus size of A is 2^(20) x 16 bytes
for (i = 0; i < S; i++) {
    read A[0];
}
```

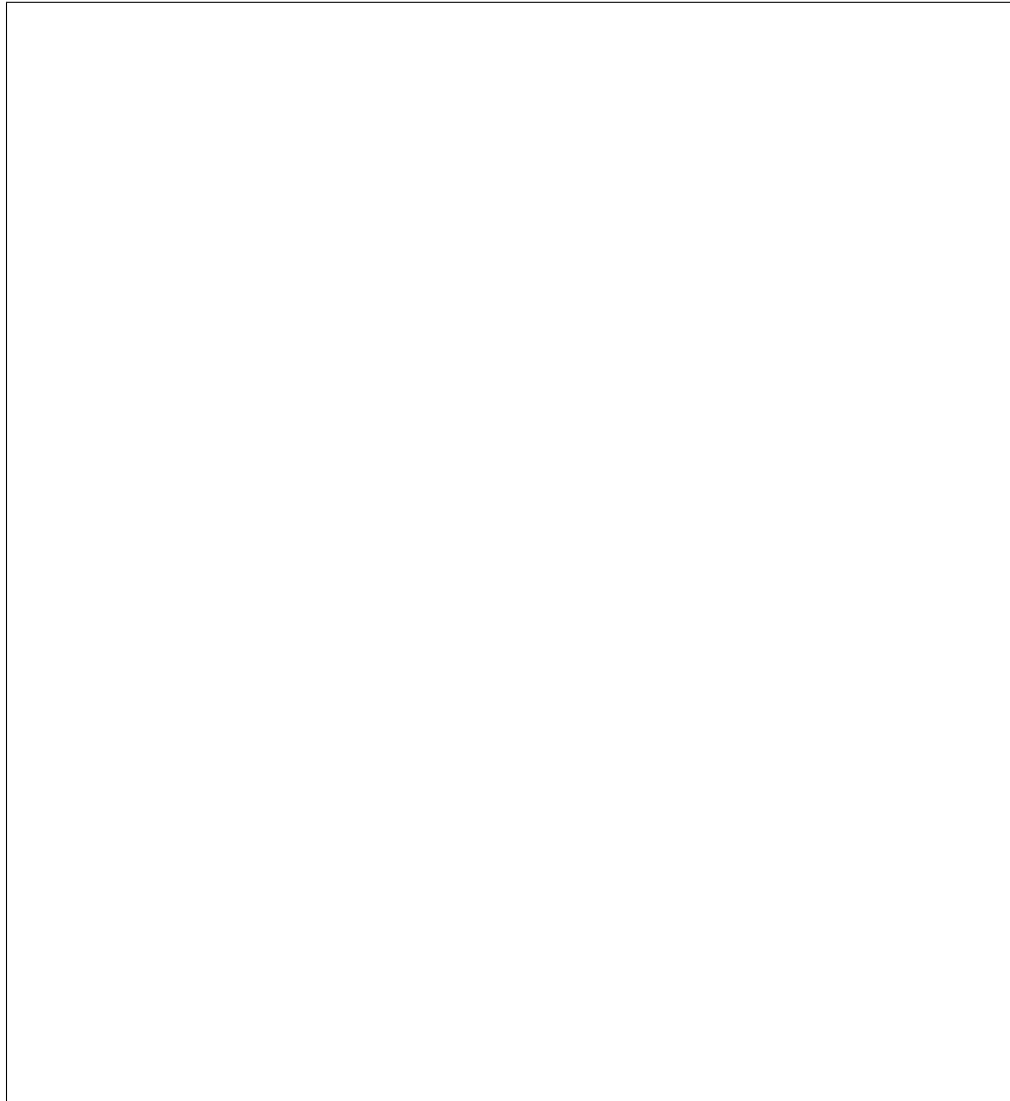
What is the total access time of this program? What kind of locality does it have, if any? What kind of misses?



Question 2.

```
// sizeof(int) = 4 and Array laid out sequentially in memory
#define S ((1<<20)*sizeof(int))
int A[S];
// Thus size of A is 2^(20) x 16 bytes
for (i = 0; i < S; i++) {
    read A[i];
}
```

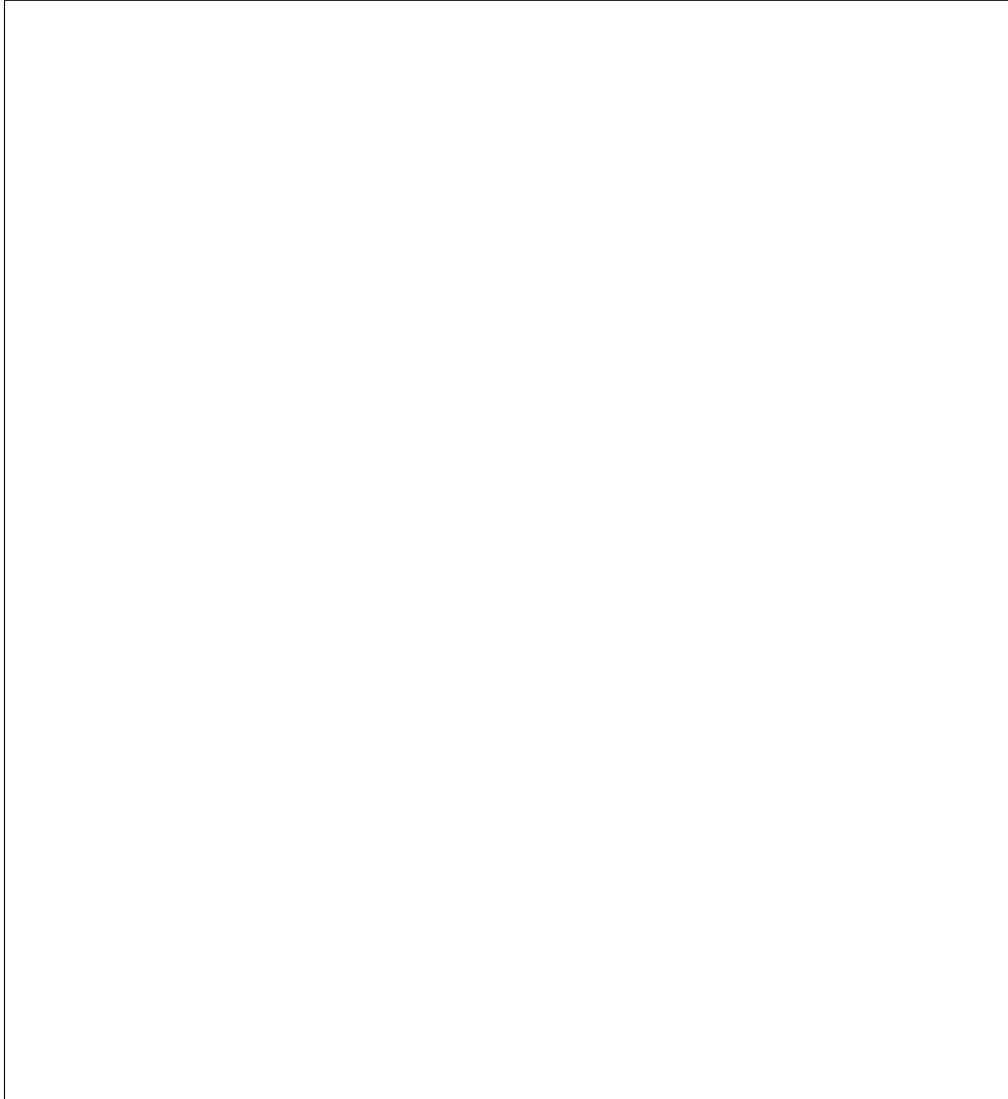
What is the total access time of this program? What kind of locality does it have, if any? What kind of misses?



Question 3.

```
// sizeof(int) = 4 and Array laid out sequentially in memory
#define S ((1<<20)*sizeof(int))
int A[S];
// Thus size of A is 2^(20) x 16 bytes
for (i = 0; i < S; i++) {
    read A[(8 * i) % S];
}
```

What is the total access time of this program? What kind of locality does it have, if any? What kind of misses?



Question 4.

```
#define S ((1<<19)*sizeof(int))
int A[S];
int B[S];
// Thus, in the main memory, the cache lines of
// B are just after all the cache lines of A
for (i = 0; i < S; i++) {
    read A[i], B[i];
}
```

What is the total access time of this program? What kind of locality does it have, if any? What kind of misses?

