Foundations of Programming for High Performance Computing: CS2101. UWO, November, 5, 2013.

Student name:	
Student ID number:	

Guidelines. The exam is closed book and all notes are forbidden. The duration is 1 hour 40 minutes. There are 20 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 4 exercises located from Page 2 to Page 16. 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	25	25 min .				
2	25	25 min.				
3	25	25 min.				
4	25	25 min.				
TOTAL	100	1h40				

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 `f(u,v)` prints “Hello World”
(b) the function `f(u,v)` does not return anything
(c) the function `f(u,v)` returns the sum of the vectors `u` and `v`
(d) the function `f(u,v)` returns the sum 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 `f(4,0.5)` returns `2.5`
(b) the function `f(4,0.5)` returns `2`
(c) the function `f(4,0.5)` returns `4`
(d) the function `f(u,v)` prints `x`

(3) A cache memory systematically stores the results of all recursive functions, such as `Fibonacci`, `Mergesort`, `Quicksort`, etc.

(a) True
(b) False

(4) In a computer desktop or laptop, each memory location at each memory level (main memory, cache memories) belongs to a cache line which size ranges between 8 and 512 bytes in size, while the size of a datum requested by a CPU instruction ranges between 1 and 16 bytes.

(a) True

(b) False

(5) In a computer desktop or laptop, when the CPU needs to read or write a memory location, it first checks the cache memories; if it finds that memory location there, then we have a:

(a) cache miss

(b) cache hit

(c) computer reboot

(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)
```

After executing them, one sees the following output value

(a) ”Task”

(b)
"start"
2
4
"stop"

(c) "start"

(d) "producer"

(e) "consumer"

(7) Making room for a new entry in a cache memory requires a replacement policy: the *Least Recently Used* (LRU) discards

(a) the least recently used items first

(b) the most recently used items first

(8) Distributed memory systems require a communication network to connect inter-processor memory.

(a) True

(b) False

(9) Pipelining is a common way to organize work with the objective of resolving memory contention in computer hardware.

(a) True

(b) False

(10) In data parallelism, tasks perform the same operation on their region of data, for example, multiply every array element by some value.

(a) True

(b) False

(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 `0:100`

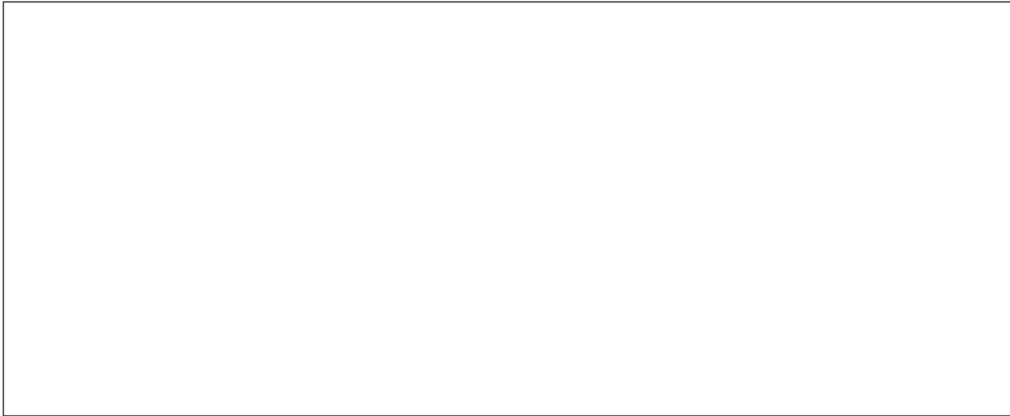
(b) all random matrices of format `100x100`

(c) all tasks (aka coroutines)

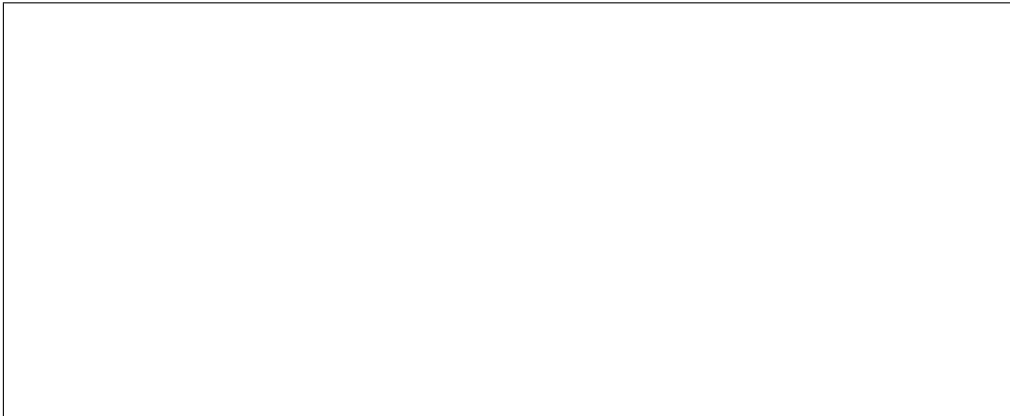
(d) all remote references

Exercise 2: Julia questions with short answers

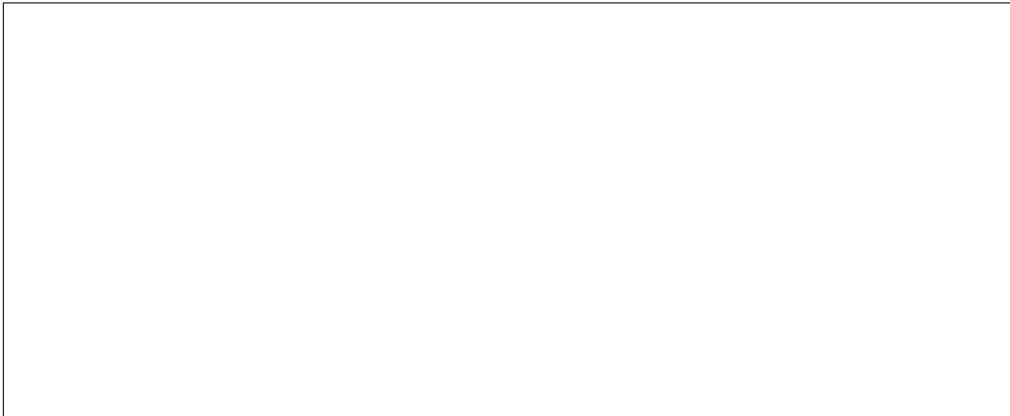
(1) Write a `Julia` function that takes as input two vectors `u` and `v` (whose coefficients could be integers, floats, etc.) of the same length and computes the square matrix `A` such that the element `A[i, j]` is the product `u[i] * v[j]`.



(2) Given a square matrix `A`, write a serial `Julia` function `Trace` computing the *trace* of `A`, that is, the sum of the diagonal elements of `A` (i. e. the elements `A[i, i]`).



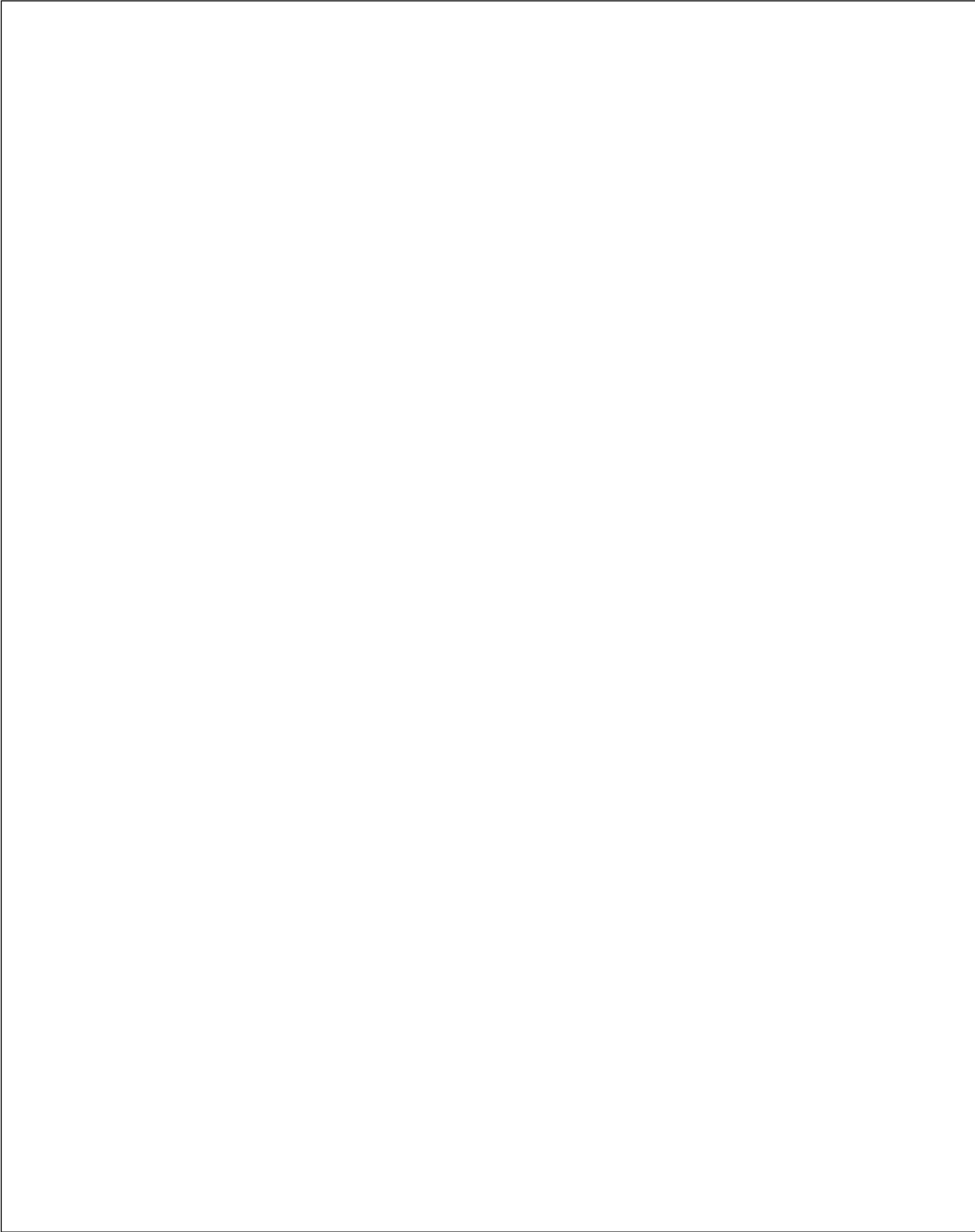
(3) Rewrite the above function `Trace` such that it takes advantage of a parallel construct.



(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
```



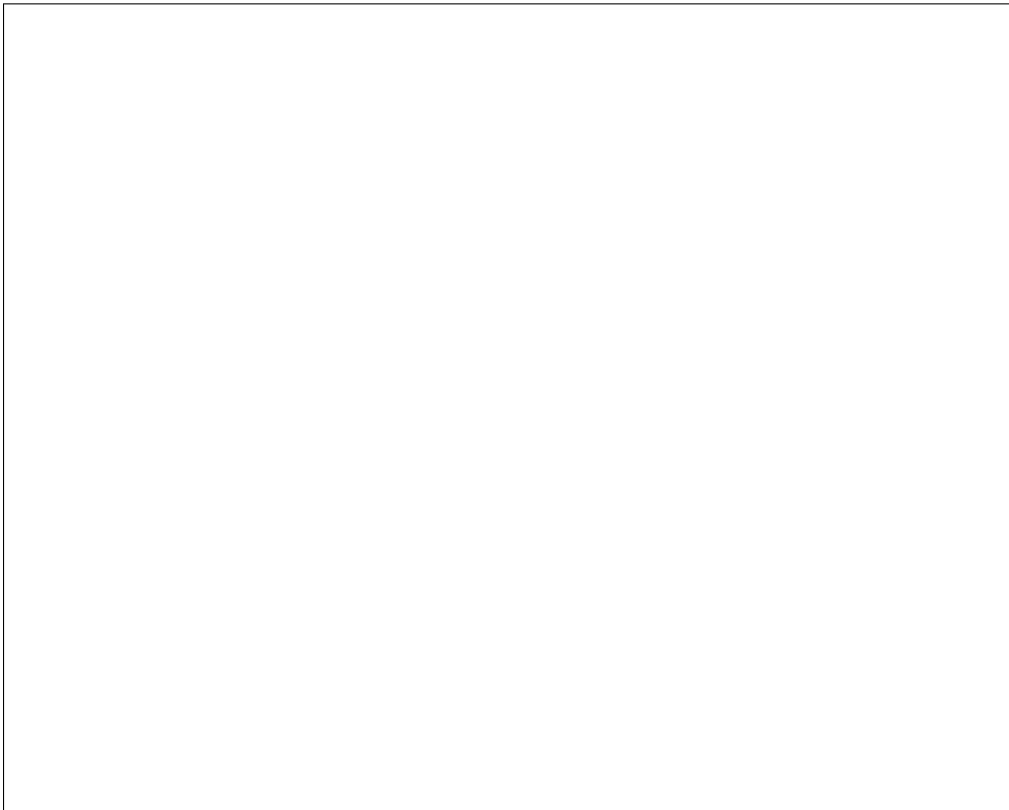
Exercise 3: writing a parallel `Julia` function

The goal of this exercise is, given a square matrix A of order n and given a positive integer k, to compute the sum

$$A + \frac{A^2}{2} + \frac{A^3}{3!} + \cdots + \frac{A^k}{k!} \tag{1}$$

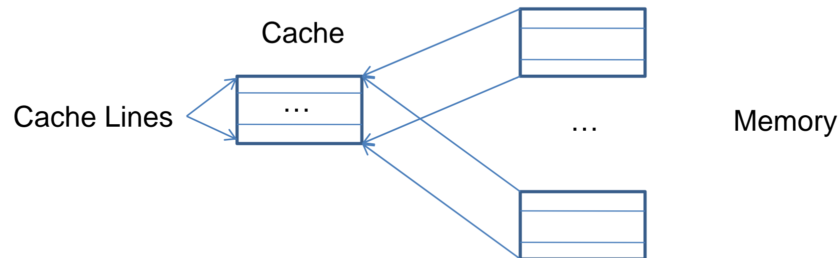
Write a parallel `Julia` function `matrixExponential` computing the above sum given A and k, using two processors and proceeding as follows:

1. the local processor, say Processor 1, accumulates the sum, thus computes successively $A, A + \frac{A^2}{2}, A + \frac{A^2}{2} + \frac{A^3}{3!}, \ldots, A + \frac{A^2}{2} + \frac{A^3}{3!} + \cdots + \frac{A^k}{k!}$
2. the remote processor, say Processor 2, computes the successive powers of A, that is, $A^2, A^3, \ldots, A^k$.



Exercise 4: analyzing cache misses

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

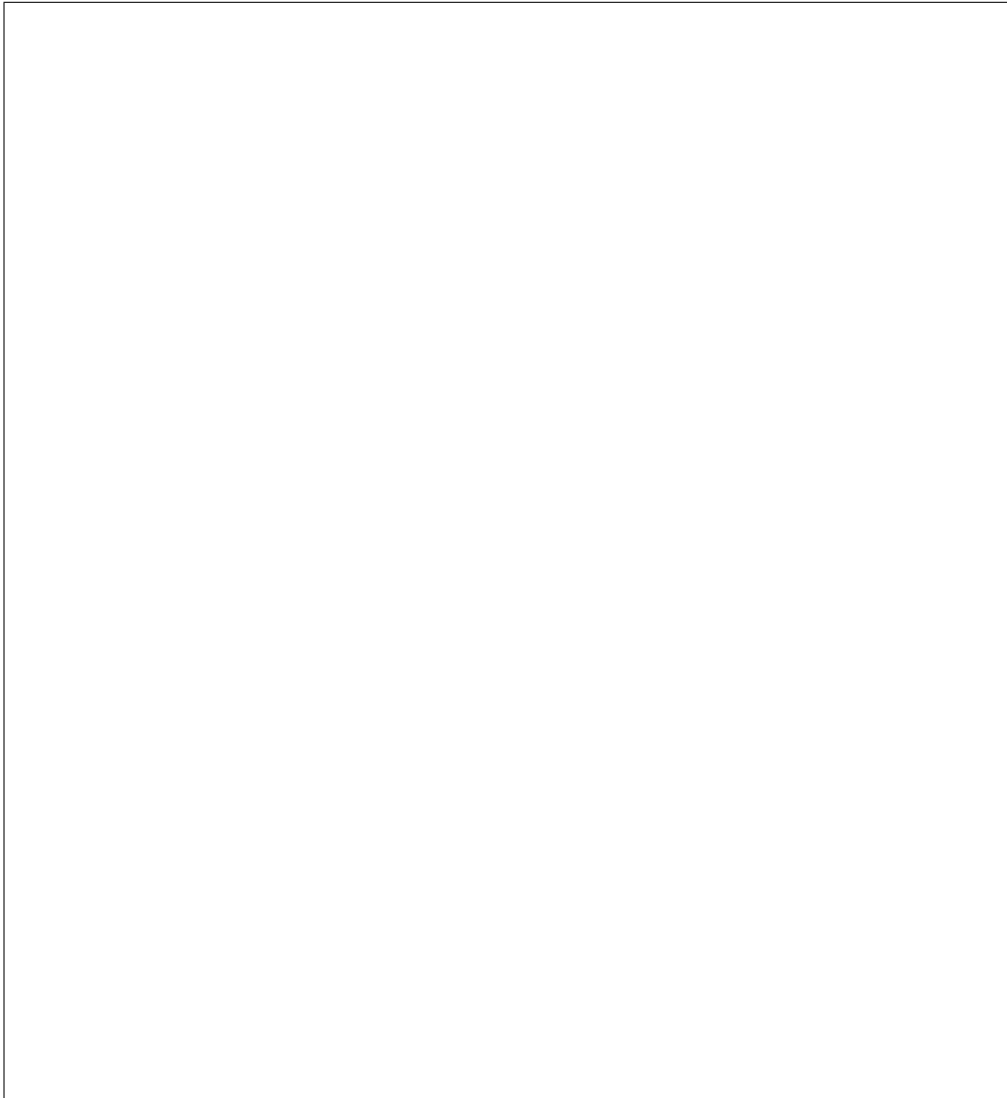


- Byte addressable memory
- The Cache has size 32Kbyte with direct mapping and 64 byte lines (512 lines); so the cache can fit $2^9 \times 2^4 = 2^{13}$ `int`.
- **Therefore**, successive 32Kbyte memory blocks can line up in cache.
- A cache access costs **1 cycle while.** a memory access costs **100 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[2];
}
```

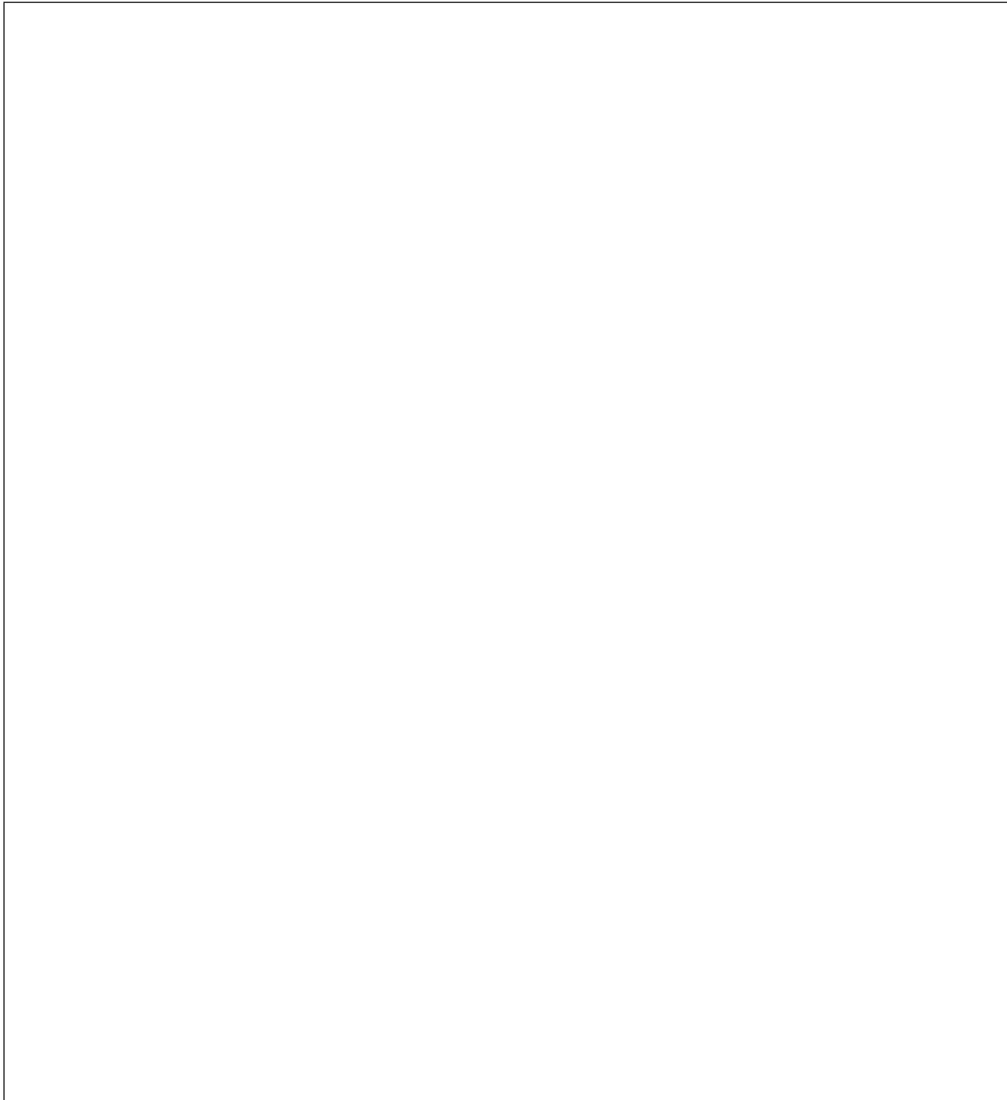
What is the total access time of this program? What kind of locality does it have, if any? What kind of misses?



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];
}
```

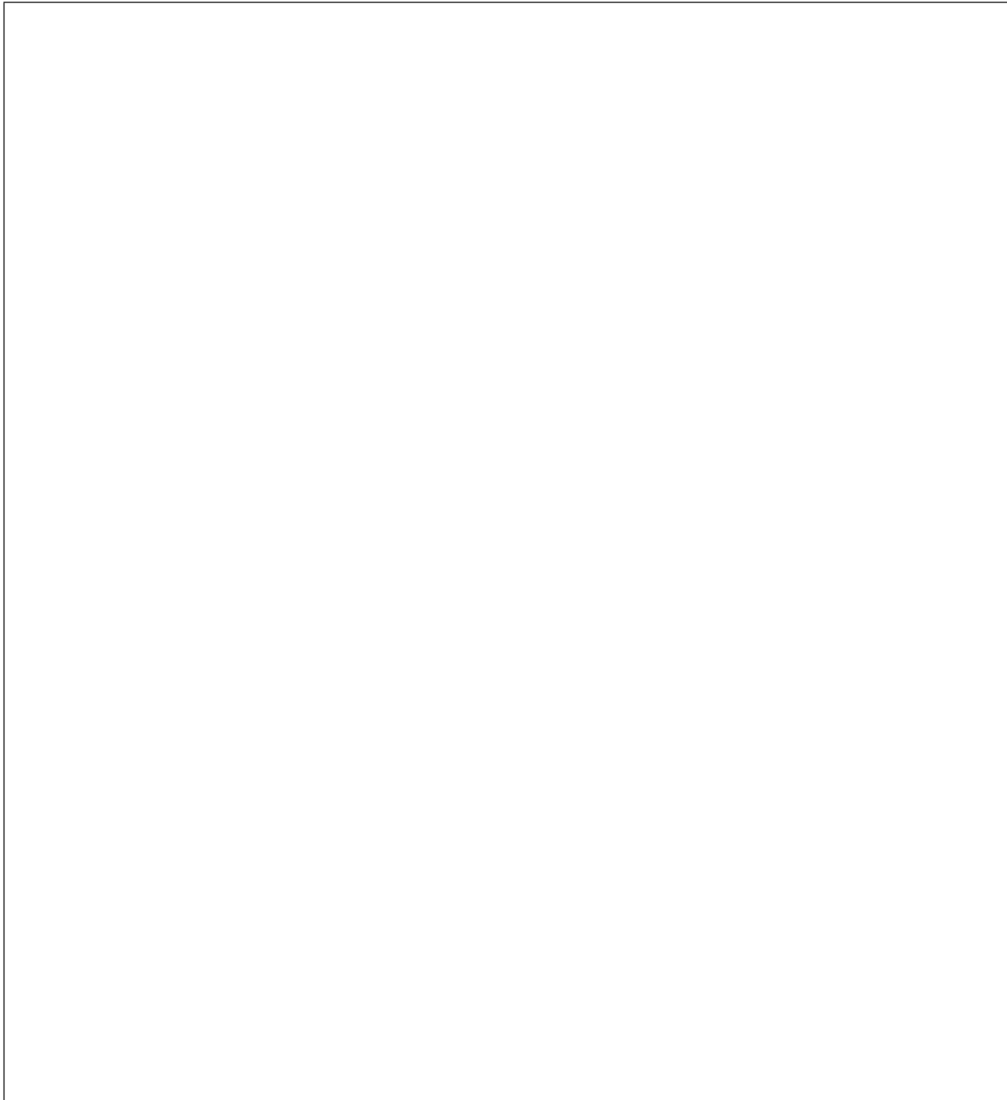
What is the total access time of this program? What kind of locality does it have, if any? What kind of misses?



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[(16 * 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?