Example Exam Questions from 2021 with Solutions

1) Mark needs to run a simulation for his thesis project and his advisor wants him to run it for a fixed problem size. Mark can make 90% of the program parallel, with 10% of it remaining sequential.

a) What speedup can Mark expect on a machine with 10 processors?

b) What would be the maximum speedup on an infinite number of processors?

Solution:
Using Amdahl's Law.
a) $Sp = 1 / (f + (1 - f)/p) = 1 / (.1 + (.9)/10) = 1 / .19 = 5.26$
b) $\max Sp = 1 / (.1 + .9/\infty) = 1 / .1 = 10$ maxSpeedUp = 1 / f

2) Mary has a computational problem of which the size can increase with an increasing number of processors. She executes the program and determines that in a parallel execution on 100 processors, 5% of the time is spent in the sequential part of the program. What is the scaled speedup of the program on 100 processors?

Scaled speedup = N + (1 - N) * f, f - seq. part

Solution:
Using Gustafson scaled speedup: $Sp = 100 + (1 - 100) * .05 = 100 - 4.95 = 95.05$

3) A research computing company is assigned a task, that is to run a large application on their hardware infrastructure. The application includes four programs. Each program depends on the result returned from the one before it. Hence, Program i cannot start before Program $i - 1$ has returned the result ($1 < i \leq 4$). The team analyzed and tested the **serial** execution time of each program on an x86 CPU. Their results were as follows:
• Program 1: 10 minutes (can be parallelized)
• Program 2: 10 minutes (can be parallelized)
• Program 3: 10 minutes (cannot be parallelized)
• Program 4: 10 minutes (cannot be parallelized)
a) Assume the company works **8 hours per day** and gets $10 each time they complete running the complete application (all four programs). How much do they earn per day?

Solution:
Completing the whole application takes 40 mins, which means $(8\times60)/40 = 12$ times per day. Then they can earn $12 \times 10 = \$120$ per day.

b) The company from the previous question is advised by a GPU sales person that running entirely on a GPU will help them a lot, and they could obtain the following speedups for each program:
• Program 1 - Speedup = 100
• Program 2 - Speedup = 10
• Program 3 - Speedup = 1
• Program 4 - Speedup = 0.2
By averaging the speedups = (100+10+1+0.2)/4 = 27.8, they think it is profitable to do so, and decide to buy the GPU product for this application. What do you think? Provide quantitative motivation for your answer.

Solution:

	Program 1	Program 2	Program 3	Program 4	Total execution time
Serial execution time	10	10	10	10	40
Speedup on GPU	100	10	1	0.2	
Execution time on GPU	10/100 = 0.1	10/10 = 1	10/1 = 10	10/0.2 = 50	61.1

The total execution time for all four programs when running on GPU = **61.1mins**. Running on GPU causes a slowdown compared to running on x86 entirely **(40 mins).** So it is not a good idea to run the whole application on the GPU.

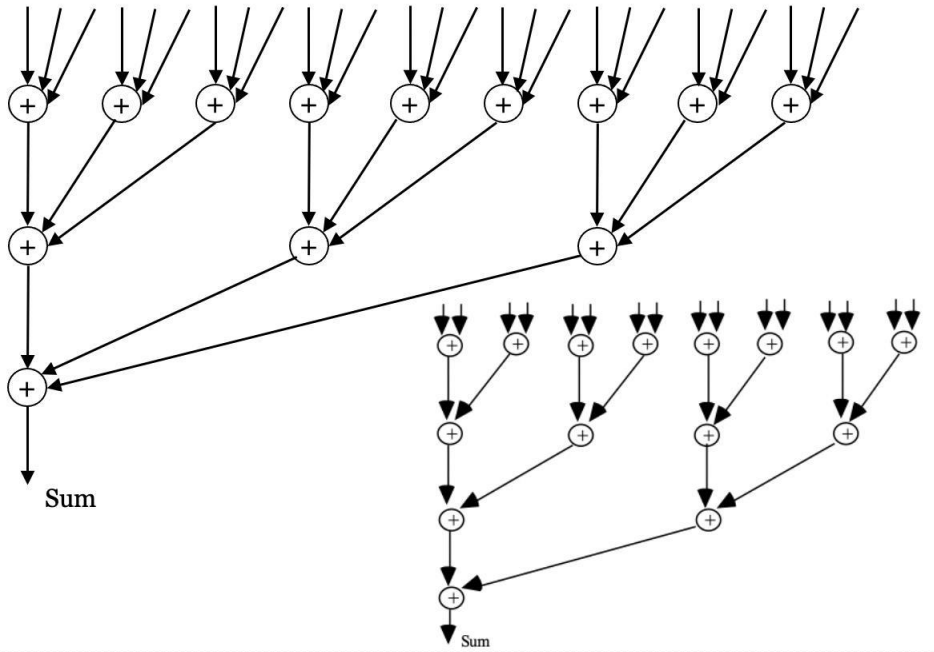
c) The engineers at the above company also think they could run Program 4 on the CPU and run the other three on the GPU. For simplicity, ignore the time needed for communication while sending data from the CPU to the GPU and vice versa. How about this combined method, when compared to running only on the CPU?

Solution:
The total execution time for all four programs using both GPU and CPU:
Execution time of P1 on GPU (T_GPU(P1)) + T_GPU(P2) + T_GPU(P3) + T_CPU(P4) = 10/100 + 10/10 + 10/1 + 10 = 21.1mins. Therefore, Speedup = 40 / 21.1 = 1.896.

d) Think about the Amdahl's law now. What is the upper limit speedup for this application? Give explanations about the speedups you get from parts b) and c).

Solution:
In the application, half of programs (program 1 and 2) can be parallelized, which means f = 0.5. The upper limit of speedup is 1/(1-f) = 2. The speed-up of 1.896 we get from problem (c) is getting close to the theoretical upper bound of 2. Note that if the "improvement" results in a slow-down for some portion (Program 4), then that can result in a very large performance degradation. It's tempting to just try to average the speedups. However, in reality, your really need to account for the slowest part.

4) SciPA Technologies has decided to become the leader in the large scientific computing server market, and has developed a cutting-edge computer system named Trifecta. The Trifecta computer is unique in that it supports reduce and broadcast trees of the third degree instead of the traditional degree-two trees supported by its competitors. In addition, Trifecta also supports a fused floating-point multiply-add operation (the operation $c + a * b$ is performed within one unit of time). Naturally, the floating-point 3-inputs wide add operation ($c + a + b$) is also completed in one unit of time. A reduction on this network is shown in the figure below along with the traditional approach used by all competitors.



Compare the time it takes to perform a sum reduction operation over 729 processors on the Trifecta computer to the time it takes to perform the same operation on a regular computer with the same number of processors that performs a conventional 2-inputs reduce. Assume the communication between processors also takes one unit of time on both machines (as well as the 2-input additions of the conventional system).

Solution:

The time to perform a sum reduction over P (= 729) processors is equal to: computational time + communication time	Trifecta computer	Regular computer
	$\log_3 P + \log_3 P = 2 \cdot \log_3 P = 12$	$\log_2 P + \log_2 P = 2 \cdot \log_2 P = 20$

5- Mark is trying to improve the performance of his program and comes across the following loop:

```
// find the sum of integers in the part of the c array that is on this processor
   int localSum = 0;
   for (i=0; i < myUB; i++) {
     localSum += c(i);
    }
// The first table below is the value of variables here (program Point A)
int recvBuf, rank=99;
MPI_Status stat;
if (myRank == 0) {
  (void) MPI_Send(&localSum, 1, MPI_INT, 1, rank, MPI_COMM_WORLD);
} else {
  (void) MPI_Recv(&recvBuf, 1, MPI_INT, myRank-1, rank, MPI_COMM_WORLD,
&MPI_Status);
  localSum += recvBuf;
  if (myRank < numProc-1) {
```

```
    (void) MPI_Send(&localSum, 1, MPI_INT, myRank+1, rank, MPI_COMM_WORLD);
  }
}
// The second table below is the value of variables here (program Point B)
```

a) Provide the answers below for each processor after the for i loop (program Point A)

	P=0	P=1	P=2
c	0, 1, 2, 3	4, 5, 6, 7	8, 9, 10, 11
localSum	[a]	[b]	[c]

Solution:
a = 6, b=22, c =38

b) Provide the answers below for each processor after the if (program Point B)

	P=0	P=1	P=2
c	0, 1, 2, 3	4, 5, 6, 7	8, 9 10, 11
localSum	[d]	[e]	[f]
recvBuf		[g]	[h]

Solution:
d=6, e=28, f= 66, g= 6, h=28.

6- Assume the following simple OpenMP code:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main()
{
   int c1 = 0;
   int c2 = 0;
   int i;
   omp_set_num_threads(4);
   #pragma omp parallel
   {
      #pragma omp atomic
      c1++;
      printf("par: %d\n", omp_get_thread_num( ));        // Line1
      #pragma omp master
      {
         #pragma omp critical
         {
         c2++;
         printf("master: %d\n", omp_get_thread_num( )); // Line2
         }
      }
      #pragma omp single
      {
         #pragma omp critical
```

```
            {
            printf("single: %d\n", omp_get_thread_num( )); // Line3
            c2++;
            }
        }
    }
    printf("c1: %d, c2: %d\n", c1, c2); // Line4
    return(0);
}
```

a) What value(s) is/are printed for the thread number(s) in Line 1?

Solution: 0 and 1 and 2 and 3

b) What value(s) is/are printed for the thread number(s) in Line 2?

Solution: 0

c) What value(s) is/are printed for the thread number(s) in Line3?

Solution: 0 or 1 or 2 or 3

d) What values are printed for the c1 and c2 in Line4, respectively?

Solution: 4 and 2