EC527 Lab0
Group member: Seyed Reza Sajjadinasab, Bin Xu

Part1

1a.

1. Intel(R) Core(TM) i7-9700 CPU
2. 3.00GHz
3. 8 cores

```
xu842251@signals49$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    1
Core(s) per socket:    8
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 158
Model name:            Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz
Stepping:              13
CPU MHz:               800.170
CPU max MHz:           4700.0000
CPU min MHz:           800.0000
BogoMIPS:              6000.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              12288K
NUMA node0 CPU(s):     0-7
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic
```

1b.

1. 3 level,
    - L1d: 8x32kb 8-way set associative instruction caches
    - L1i: 8x32kb 8-way set associative instruction caches
    - L2: 8x256kb 4-way set associative instruction caches
    - L3: 12mb 12-way set associative instruction caches
2. Cascade lake

3. The cores in Intel(R) Core(TM) i7-9700 CPU are real. They are physical components in the hardware of the CPU.
4. Max Memory Bandwidth: 41.6 GB/s
5. Base Frequency: 3.0 GHZ
   Turbo Frequency: 4.7 GHZ

Part 2

2a.
Using precision measurement, accuracy measurement and resolution to calculate the accuracy. The timer depends on the constants. The constants are different for different systems. The accuracy is different for different systems.
Scientists determine the accuracy of a measurement by comparing it with known data.
It is important to note that the accuracy of a timer can be influenced by a variety of factors, including the quality of the hardware, the software implementation, and the operating environment. In general, it is a good idea to perform multiple tests using multiple methods to get a comprehensive assessment of the accuracy of a timer.

2b.
RDTSC is a cycle counter which is used as a timer.
The RDTSC value may not increase linearly, then it leads to incorrect time measurement results. Changes in CPU frequency or power state may affect the value of the RDTSC register. CPU operating frequency changes with different processes which impacts the timer value.

2c.
We change the CLK_RATE for RDTSC. We do not touch GET_TOD_TICKS and GET_SECOND_TICS.
Old CLK_RATE: 2.0e9
New CLK_RATE: 3.0e9

2e.
We made these two modifications to get the result of diff() function.

struct timespec temp = diff(time_start,time_stop);
time_sec = temp.tv_sec; time_ns = temp.tv_nsec

2f.
We added the below dummy function to delay the program for about 1 sec.

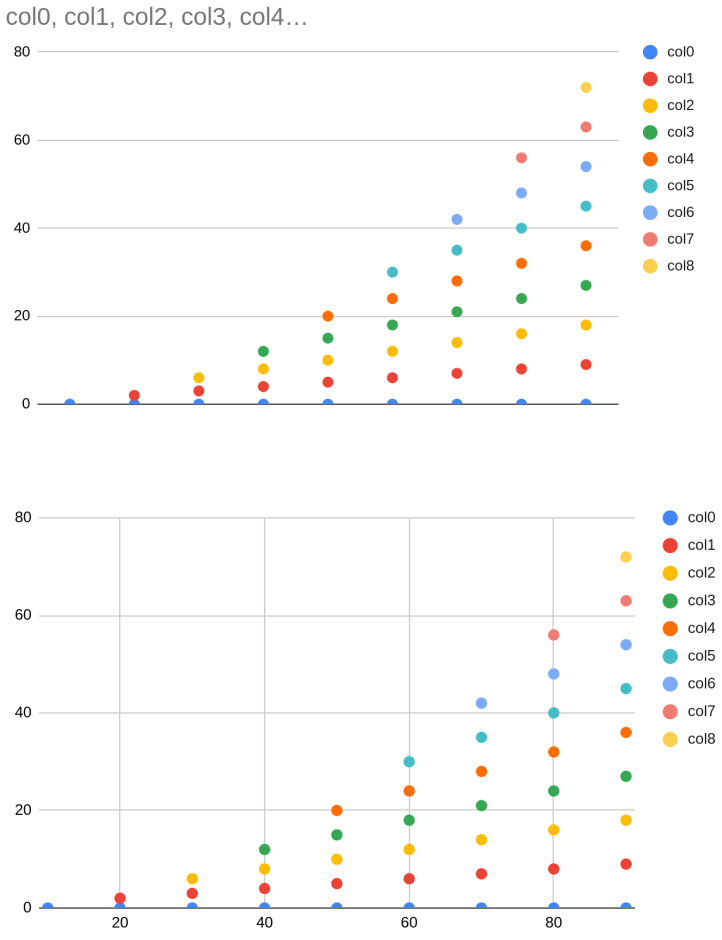
long long int s;
for(s=0; s<0.7e9; s++);

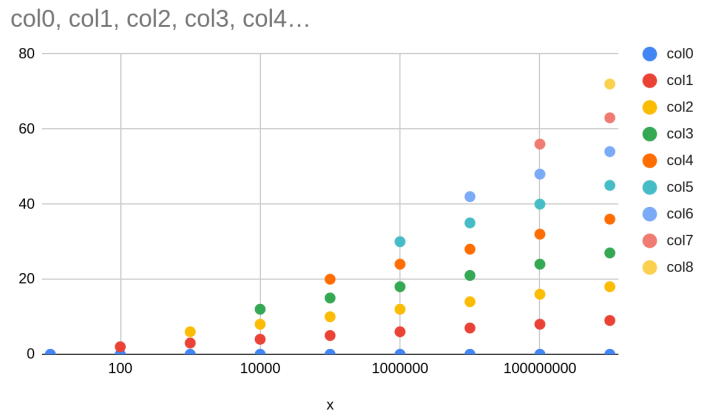
Then we run it for 5 times, and the result was this:

that took 1 sec and 77284685 nsec (1.077284685 sec)
that took 1 sec and 59192689 nsec (1.059192689 sec)
that took 1 sec and 62214956 nsec (1.062214956 sec)
that took 1 sec and 56543045 nsec (1.056543045 sec)
that took 1 sec and 52042067 nsec (1.052042067 sec)

The std dev is 0.0096047529871117

Part 3





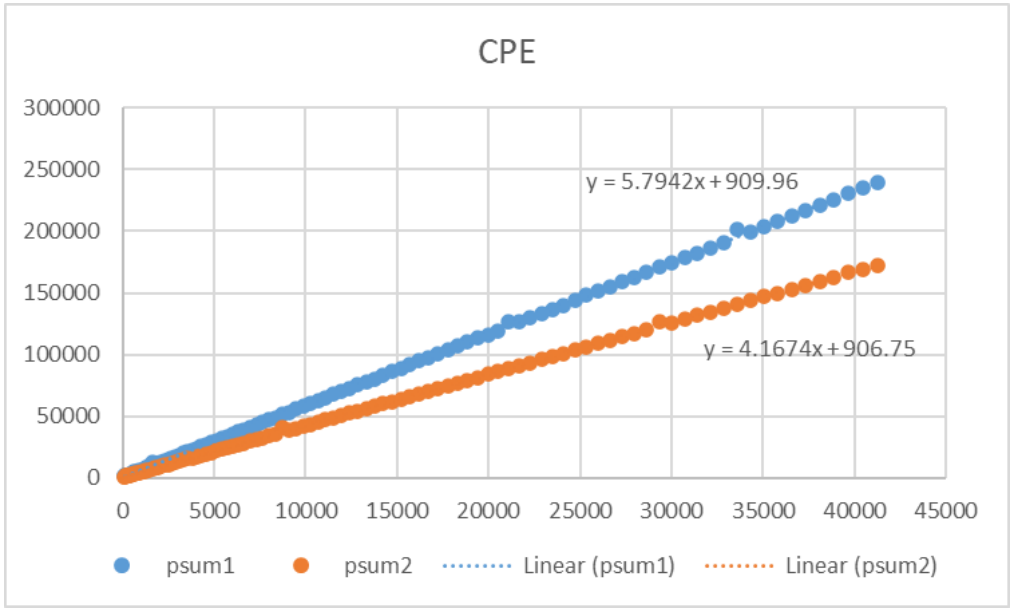
Part 4

4b.

One way to get rid of anomalies is to deduct the value of (Slope * n) from each data. Then, calculate the average and standard deviation of the data. Then, we can omit those values that are bigger than the standard deviation.

4c.

The slope of each line indicates the CPE for each psum algorithm. This value for psum1 is 5.79 cycles per element and for psum2 is 4.16. This indicates that psum2 works faster than psum1 which is what we expected and what the book said. However, the values are not the same. It is because of the different processors in which these programs ran.



Part 5

5a.
User: 0.558 sec

```
xu842251@vlsi50$ time ./test_O_leve

 Starting a loop

 done

real    0m0.573s
user    0m0.558s
sys     0m0.005s
```

5b.

```
xu842251@vlsi50$ gcc -O1 test_O_lev
xu842251@vlsi50$ time ./test_O_leve

 Starting a loop

 done

real    0m0.053s
user    0m0.048s
sys     0m0.002s
```

After running O1, the time is close to 0 second.

5c. 5d.
After using O1, operation in the for loop is optimized. Quasi_random is never used outside of the loop. It is discarded by the compiler.
In O0, there is no optimization, so the loop is kept.

5e.

```
xu842251@vlsi50$ gcc -O0 test_O_level.c -o test_O_level
xu842251@vlsi50$ time ./test_O_level

 Starting a loop
-1.559369
 done

real    0m0.564s
user    0m0.559s
sys     0m0.003s
xu842251@vlsi50$ gcc -O1 test_O_level.c -o test_O_level
xu842251@vlsi50$ time ./test_O_level

 Starting a loop
-1.559369
 done

real    0m0.352s
user    0m0.347s
sys     0m0.001s
```

After O1 optimization, the user time is 0.347s. Program is not optimized so much, because in this question we need to print out quasi_random.
The loop is still in the assembly code. But there is optimization. In O0, we get the quasi_random from memory. In O1, we get the quasi_random from the register. So after optimization, it takes less time.

Part 6

6a.
For the benchmark, it shows the memory bandwidth limitation and computational limitation. For kernel1, arithmetic intensity is 0.5 FLOPs/byte. It is limited by the memory bandwidth to no more than 8.0 GFLOPS/sec. For kernel2, arithmetic intensity is 4.0 FLOPs/byte. It is limited computationally to 16 GFLOPS/s.

6b.

According to P&H we expected that there should be some simple floating point operations as a benchmark code. However, the stream.c uses 4 different approaches to address this problem. stream.c uses four kernels for analysis:

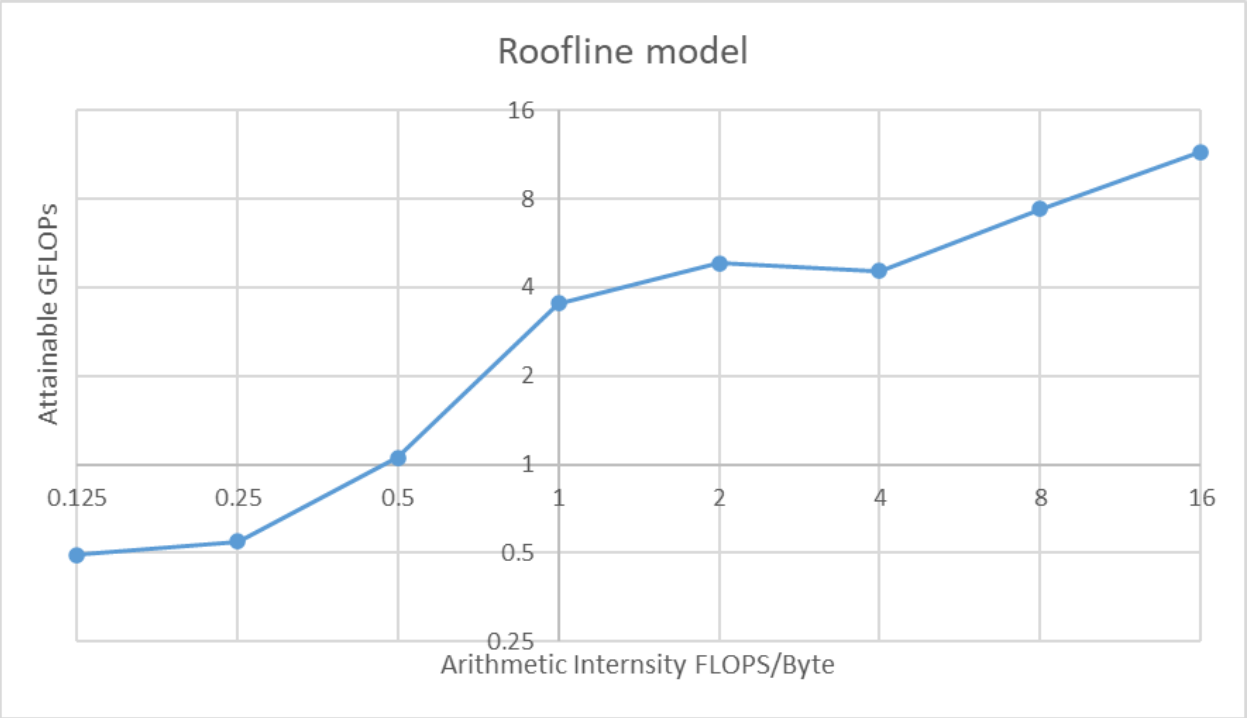
1. ”Copy’ ‘ measures transfer rates in the absence of arithmetic.
2. “Scale’ ‘ adds a simple arithmetic operation.
3. “Sum’ ‘ adds a third operand to allow multiple load/store ports on vector machines to be tested.
4. “Triad’ ‘ allows chained/overlapped/fused multiply/add operations.

Then, it calculates the average of them. Anyhow, as we can see in the following section the result is not exactly the same as the ideal maximum bandwidth that we expected for Intel(R) Core(TM) i7-9700 CPU.

6c.
memory bandwidth: 23966.6561 MB/s
It is less than the max bandwidth(41.6 GB/s)

6d.
AI = 0.125000 GFLOPs = 0.492557
AI = 0.250000 GFLOPs = 0.545595
AI = 0.500000 GFLOPs = 1.053775
AI = 1.000000 GFLOPs = 3.538577
AI = 2.000000 GFLOPs = 4.822688
AI = 4.000000 GFLOPs = 4.548970
AI = 8.000000 GFLOPs = 7.384210
AI = 16.00000 GFLOPs = 11.55

6e.



6f.
As we expected the attainable GFLOPs is increasing while it is limited by memory bandwidth. However, by increasing the AI we reach to another area where the task is limited by the processor capacity. Here we may not still reach that point since there is still a minor increase. Also, it can be a result of different optimizations for different codes. In other words, after an area of flatted graph with a new optimization done by the compiler, the resultant code is still bounded by memory bandwidth.

7a. No
7b. 10 hours
7c. No
7d. Yes