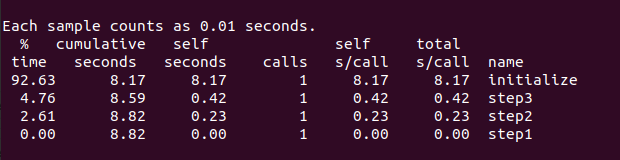
Coursework

# 1A)



Most computationally expensive

^

|

| Initialize -> 92.63%

| Step3 -> 4.76%

| Step2 -> 2.61%

| Step1 -> 0.00%

|

Least computationally expensive

# 1B)

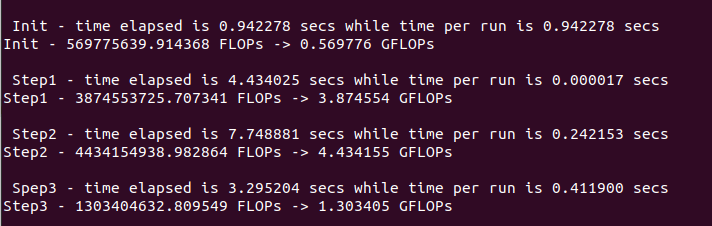
### a)

Below are screenshots of the FLOPs values given. The average FLOPs value will be calculated based on these.

Fig.1



Fig.2

Fig.3

Most computationally expensive routines are Initialize, Step3 & Step2. This means that we will ignore Step1.

**Finding the average FLOPs for step 3 (to 2 decimal places):**

**Finding the average FLOPs for Step2 (to 2 decimal places):**

**Finding the average FLOPs for Initialization (to 2 decimal places):**

### b)

#define BILLION 1000000000

**For Step3:**

start=omp\_get\_wtime();

for (i=0;i<8;i++){

reduction=step3();

}

end=omp\_get\_wtime();

printf("\n Spep3 - time elapsed is %f secs while time per run is %f secs\n",end-start, (end-start)/8);

flop = 2 \* N \* N + 1 \* N / 8;

timeper = (end-start)/8;

flops = flop/timeper;

printf("Step3 - %f FLOPs -> %f GFLOPs\n",flops, flops/BILLION);

**For Step2:**

start=omp\_get\_wtime();

for (i=0;i<32;i++){

step2();

}

end=omp\_get\_wtime();

printf("\n Step2 - time elapsed is %f secs while time per run is %f secs\n",end-start, (end-start)/32);

flop = ((8+8) \* N / 4) \* N;

timeper = (end-start)/32;

flops = flop/timeper;

printf("Step2 - %f FLOPs -> %f GFLOPs\n",flops, flops/BILLION);

**For Initialization:**

start=omp\_get\_wtime();

for (i=0;i<1;i++){

initialize();

}

end=omp\_get\_wtime();

printf("\n Init - time elapsed is %f secs while time per run is %f secs\n",end-start, (end-start)/1);

flop = N + 2 \* N \* N;

timeper = (end-start)/1;

flops = flop/timeper;

printf("Init - %f FLOPs -> %f GFLOPs\n",flops, flops/BILLION);

### c)

CPU: Intel Core i7 8750H 2.20GHz (Turbo 4.10 GHz)

DDR: DDR4 16GB 1200GHz

OS: Ubuntu 22.04 LTS (Jammy Jellyfish) (64-bit)

## 1C)

**Theoretical peak FLOPs:**

Explain the results obtained in Step B.

## 1D)

