Code and Screenshots

PA1_CPU_First

```
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#include <sys/time.h>
#include <math.h>



float threadFunc_int(long int noItr)
{
	int i,val1=1,val2=2,val3=4;
	struct timeval start_time;
	struct timeval end_time;

	gettimeofday(&start_time,NULL);

	for(i=0;i<noItr;i++)
	{
		val1=val2*val3;
		val3=val1+val2;
		val2=val3-val1;
		val1=val2/val3;

	}
	gettimeofday(&end_time,NULL);
	printf("%d %d",end_time.tv_sec,start_time.tv_sec);
	float timeTaken=(float)(end_time.tv_sec-start_time.tv_sec);
	return timeTaken;
```

```
}
float threadFunc_float(long int noItr)
{
                int i;
        float val1=1.2,val2=2.4,val3=4.6;

        struct timeval start_time;
        struct timeval end_time;

        gettimeofday(&start_time,NULL);

        for(i=0;i<noItr;i++)
        {
                val1=val2*val3;
                val3=val1+val2;
                val2=val3-val1;
                val1=val2/val3;
        }
        gettimeofday(&end_time,NULL);
        float timeTaken=(float)(end_time.tv_sec-start_time.tv_sec);

        return timeTaken;
}

int main(void)
{
        int i=0;
        int noThreads;
        struct timeval begin_time;
```

```
struct timeval last_time;
int op;
float timeTaken=0;
float avgTime=0.0;

printf("Enter Operation to Perform 1 int 2 float");
        fflush(stdout);
        scanf("%d",&op);
        printf("Enter threads");
        fflush(stdout);
        scanf("%d",&noThreads);
        pthread_t pth[noThreads];

for(i=0;i<noThreads;i++)
{
if(op==1)
pthread_create(&pth[i],NULL,threadFunc_int,100000000);
else
pthread_create(&pth[i],NULL,threadFunc_float,100000000);
}
for(i=0;i<noThreads;i++)
{
        pthread_join(pth[i],&timeTaken);
        avgTime=avgTime+timeTaken;
}

float ops=(float)(100000000*4*noThreads)/avgTime;
if(op==1)
{
```

```
        printf(" Iops in Hertz is %f",(double)ops);
        printf(" Iops in Giga Hertz is %f",ops/1000000000);
        }
        else
        {
                printf(" Flops in Hertz is %lf",ops);
                printf(" Flops in Giga Hertz is %f",ops/1000000000);
        }
        return 0;
}
```

PA1_CPU_Second.c

```
#include <stdio.h>
#include <pthread.h>
#include <time.h>
#include <sys/time.h>

#include <fcntl.h>
#include <sys/stat.h>
pthread_mutex_t lock;

int threadFunc_int(int s)
{

        int i=0,a=1,b=2,c=4;

        struct timeval t1;
```

```
        struct timeval t2;

                gettimeofday(&t1,NULL);
        for(i=0;i<s;i++)
        {

                a=b*c;
                c=a+b;
                b=c-a;
                a=(int)b/c;

                gettimeofday(&t2,NULL);
                if(t2.tv_sec-t1.tv_sec==1)
                                {
                        //printf("i is %d",i);
                        return i;
                                }
        }
}

int threadFunc_float(int s)
{

        int i;
        float a=1.2,b=2.3,c=4.5;

        struct timeval t1;
        struct timeval t2;
```

```
        int timer=0;

        while(timer==0)
        {
                gettimeofday(&t1,NULL);

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

                a=b*i;
                c=a+b;
                b=c-a;
                a=(float)b/c;

                gettimeofday(&t2,NULL);
                if(t2.tv_sec-t1.tv_sec==1)
                                {
                        timer=1;
                        //printf("%d",i);
                        return i;
                                }
        }
        }
}

int main(void)
{
        int i=0,j=0;
```

```
        int noThreads;
        int op;
        /* if (pthread_mutex_init(&lock, NULL) != 0)
            {
                printf("\n mutex init failed\n");
                return 1;
            }
         */

printf("Enter Operation 1 int 2 float");
fflush(stdout);
scanf("%d",&op);
        printf("Enter threads");
        fflush(stdout);
                scanf("%d",&noThreads);
                pthread_t pth[noThreads];
                char *fileName="values.txt";
                int fileDesc = open(fileName,O_CREAT|O_RDWR,S_IRWXU);

        for(j=0;j<600;j++)
        {
        int f=0;
        float avg=0;
        for(i=0;i<noThreads;i++)
        {
        if(op==1)
        {
        pthread_create(&pth[i],NULL,threadFunc_int,100000000);
        }
```

```
        else
        {

                pthread_create(&pth[i],NULL,threadFunc_float,100000000);
        }
        }




        for(i=0;i<noThreads;i++)
        {
                pthread_join(pth[i],&f);
                avg=(float)avg+(f*4);
                //printf("avg is %f",avg);
        }

        //printf("\n avg is %d",(int)avg/noThreads);
        float iops=(float)(avg/noThreads);
        printf("\n flops is %f ",iops);
        printf("\n flops in Giga Hertz is %f",iops/1000000000);

        char *c = (char *)malloc(sizeof(char)*10);

         sprintf (c,"%f",iops/1000000000);
         strcat(c,"\r\n");
   write(fileDesc,c,strlen(c));
                                }
        return 0;
}
```

Pa1_Memory.c

```
#include <stdio.h>

#include <pthread.h>

#include <time.h>

#include <sys/time.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <malloc.h>





pthread_mutex_t lock;

struct timeval start_time,end_time;

float seq_thread_func(long int size)

{

      // allocate a memory of blocksize 20 mb ,

      //read blocks of data using sequentital access from one memory area to
other using memcpy

      //and return the time taken for read and write

      int k=0;

      pthread_mutex_lock(&lock);

      long int blockSize=(long int)(20*1000000)/size;

      char *mem1;

    char *mem2;

    mem1=malloc(blockSize*size);

    mem2=malloc(blockSize*size);
```

```
      strncat(mem2,"hello",blockSize*size);

      gettimeofday(&start_time,NULL);

      for(k=0;k<(long int)(blockSize);k++)

          {

            memcpy(mem1+k,mem2+k,size);

            // copy required bytes from one memory to other sequentially

        }

      gettimeofday(&end_time,NULL);


    double
data1=(double)start_time.tv_sec+((double)start_time.tv_usec/1000000);

      double
data2=(double)end_time.tv_sec+((double)end_time.tv_usec/1000000);

      float dataTime=data2-data1;//calculate the time taken for read and
write in sec

      free(mem1);

      free(mem2);


      printf("%f",dataTime);

    pthread_mutex_unlock(&lock);


    return dataTime;

}




float random_thread_func(long int size)

{


      // allocate a memory of blocksize 20 mb ,

      //read blocks of data using random access from one memory area to other
using memcpy

      //and return the time taken for read and write


      int random_pos=0,k;
```

```
      pthread_mutex_lock(&lock);
      long int blockSize=(1000000*20)/size;
    char *mem1;
    char *mem2;

      mem1=malloc(blockSize*size);//allocate memory
      mem2=malloc(blockSize*size);//allocate memory

      strncat(mem2,"hello",blockSize*size);
      gettimeofday(&start_time,NULL);

      for(k=0;k<(long int)blockSize;k++)
    {

            random_pos = rand()%(blockSize);
            memcpy(mem1+random_pos,mem2+random_pos,size);
            // copy required bytes from one memory to other in random_pos
position
    }

      gettimeofday(&end_time,NULL);
    double
data1=(double)start_time.tv_sec+((double)start_time.tv_usec/1000000);
      double
data2=(double)end_time.tv_sec+((double)end_time.tv_usec/1000000);
      float dataTime=data2-data1;//calculate the time taken for read and
write in sec
      free(mem1);
      free(mem2);
printf("%f",dataTime);
    pthread_mutex_unlock(&lock);

    return dataTime;
```

```
}



void main()
{

  float throughput,latency;
int access;
int nothreads;
int operation;
long int size;
int i;

float timeTaken=0,data=0;
   printf("\n Block size:(Select 1 for 1B=1,2 for 1KB=1024,3 for
1MB=1048576):");
   fflush(stdout);
   scanf("%d",&size);
   printf("Access Method : 1-Sequential 2-Random");
   fflush(stdout);
   scanf("%d",&access);
   printf("Enter number of threads 1,2,4 :");
   fflush(stdout);
   scanf("%d",&nothreads);
   pthread_t  pth[nothreads];
   if(size==2)
   {
            // if 2 is selected assign 1024 bytes to read and write in memory

   size=(long int)1024;
   }
   if(size==3)
```

```
   {
            // if 3 is selected assign 1024*2014 bytes to read and write in
memory

      size=(long int)(1024*1024);
   }



   if (pthread_mutex_init(&lock, NULL) != 0)
      {
          printf("\n mutex init failed\n");
          return 1;
      }
   // create given no of threads  and perform operations (read/write)
    for(i=0;i<nothreads;i++)
   {
      if(access==1)
      //this block is for doing memcopy sequentially
   pthread_create(&pth[i],NULL,seq_thread_func,(long int)size);
      else
      //this block is for doing memcopy to random memory
   pthread_create(&pth[i],NULL,random_thread_func,(long int)size);
       }



    // joining the thread with other thread
   for(i=0;i<nothreads;i++)
   {
   pthread_join(pth[i],&data);
   timeTaken=timeTaken+data;        // add the time taken for all threads
   }
   printf("%f",timeTaken);
```

```
   timeTaken=timeTaken/nothreads;

   latency=(float)(timeTaken*size*1000)/(2*1000000*20);


   printf("Latency in milliseconds : %f \n",latency);

      throughput = (float)(1000000*20*2)/(timeTaken);//(no of
loops*size*2/time taken)

       printf("Throughput in MB/sec : %f\n",(throughput)/(1024*1024));


                     int filesc;

                     char *fs="values.txt";

                     filesc = open(fs,O_CREAT|O_RDWR,S_IRWXU);


                  char *c = (char *)malloc(sizeof(char)*10000);

              char *s = (char *)malloc(sizeof(char)*10);

                                   s[0]='\0';


                        c[0] = '\0';


                  strcat(c,"\r\n");

                        strcat(c,"size");

                        strcat(c," ");

                        strcat(c,"threads");

                        strcat(c," ");

                        strcat(c,"access");

                        strcat(c," ");

                        strcat(c,"throughput");

                        strcat(c,"   ");

                        strcat(c,"latency");

                        strcat(c,"\r\n");

                  sprintf (s,"%d",(int)size);

                        strcat(c,s);

                        strcat(c,"   ");
```

```
                s[0]='\0';
                sprintf (s,"%d",(int)nothreads);
                strcat(c,s);
                strcat(c,"     ");
                sprintf(s,"%d",(int)access);
                strcat(c,s);
                strcat(c,"     ");
                sprintf(s,"%f",(float)throughput/(1024*1024));
                strcat(c,s);
                strcat(c,"   ");
            s[0]='\0';
                sprintf(s,"%f",(float)latency);
                strcat(c,s);
                strcat(c,"\r\n");

                printf("\n c is \n %s",c);

              write(filesc,c,strlen(c));
                free(c);
                free(s);

}
```

Pa1_Disk.c#include <stdio.h>

#include <pthread.h>

#include <time.h>

#include <sys/time.h>

#include <fcntl.h>

#include <sys/stat.h>

```
int fileDesc ;
pthread_mutex_t lock;

struct timeval start_time,end_time;

float write_thread_seq(long int size)
 {
            // open the file , write blocks of data using sequential access and return the time taken for write

     pthread_mutex_lock(&lock);

     long int blockSize=(long int)((1000000*20)/size);

     char *fileName="trial.txt";
       fileDesc = open(fileName,O_RDWR,S_IRWXU);
            int itr;

                        char *writeData;
                        writeData=(char *)malloc(size);
          gettimeofday(&start_time,NULL);

                        for(itr=0;itr<blockSize;itr++)
                         {
                        int currentPageSize =write(fileDesc,writeData,size);//write sequentially from starting
position of file
                         }

            gettimeofday(&end_time,NULL);
```

```
        double data1=(double)start_time.tv_sec+((double)start_time.tv_usec/1000000);
                double data2=(double)end_time.tv_sec+((double)end_time.tv_usec/1000000);
               float dataTime=(double)data2-data1;//calculate the time taken for write in sec

                  pthread_mutex_unlock(&lock);
                  free(writeData);
        return dataTime;
}

float read_thread_seq(long int size)
 {

        // open the file , read blocks of data using sequential access and return the time taken for read

        long int blockSize=(long int)(1000000*20)/size;
    pthread_mutex_lock(&lock);
      char *fileName="trial.txt";
       fileDesc = open(fileName,O_RDWR,S_IRWXU);
       int itr;
                  char *readData;
         readData=(char *)malloc(size);

        gettimeofday(&start_time,NULL);
              for(itr=0;itr<blockSize;itr++)
              {
              int currentPageSize =read(fileDesc,readData,size);//read sequentially from starting position
of file
              }
```

```
        gettimeofday(&end_time,NULL);

        double data1=(double)start_time.tv_sec+((double)start_time.tv_usec/1000000);
            double data2=(double)end_time.tv_sec+((double)end_time.tv_usec/1000000);
            float dataTime=data2-data1;//calculate the time taken for read in sec

              pthread_mutex_unlock(&lock);
              free(readData);
              return dataTime;
}

float write_thread_random(long int size)
 {
        // open the file , write blocks of data using random access and return the time taken for write



              int itr;
        off_t random_pos;
        long int blockSize=(long int)(1000000*20)/size;
   char *fileName="trial.txt";
   char *writeData;

   fileDesc = open(fileName,O_RDWR,S_IRWXU);
   pthread_mutex_lock(&lock);

   writeData=(char *)malloc(size);
        gettimeofday(&start_time,NULL);

      for(itr=0;itr<blockSize;itr++)
```

```
{

random_pos = rand()%(int)blockSize;//assign random block to write
int currentPageSize =pwrite(fileDesc,writeData,size,random_pos);// writes from random block
position
}
    gettimeofday(&end_time,NULL);

    double data1=(double)start_time.tv_sec+((double)start_time.tv_usec/1000000);
        double data2=(double)end_time.tv_sec+((double)end_time.tv_usec/1000000);
      float dataTime=data2-data1;//calculate the time taken for write in sec
      free(writeData);

      pthread_mutex_unlock(&lock);
          return dataTime;
    }

float read_thread_random(long int size)
{
    // open the file , read blocks of data using random access and return the time taken for read
        int itr;
        off_t random_pos;
long int blockSize=(long int)(1000000*20)/size;
char *fileName="trial.txt";
char *readData;
fileDesc = open(fileName,O_RDWR,S_IRWXU);
pthread_mutex_lock(&lock);

        readData=(char *)malloc(size);
```

```
                    gettimeofday(&start_time,NULL);

        for(itr=0;itr<blockSize;itr++)
        {
        random_pos = rand()%(int)blockSize;//assign random block to read
        int currentPageSize =pread(fileDesc,readData,size,random_pos);// read from random block
position
        }
        gettimeofday(&end_time,NULL);
        double data1=(double)start_time.tv_sec+((double)start_time.tv_usec/1000000);
        double data2=(double)end_time.tv_sec+((double)end_time.tv_usec/1000000);
        float dataTime=data2-data1; //calculate the time taken for read in sec
        free(readData);

        pthread_mutex_unlock(&lock);
                    return dataTime;

            }

void main()
{

float throughput,latency;
int access;
int nothreads;
int operation;
long int size;
int i;
```

```
float timeTaken=0,dataTime;
  printf("Enter Operation to perform 1-Read 2-Write");
  fflush(stdout);
  scanf("%d",&operation);
  printf("\n Block size:(Select 1 for 1B=1,2 for 1KB=1024,3 for 1MB=1048576):");
  fflush(stdout);
  scanf("%d",&size);
  printf("Access Method : 1-Sequential 2-Random");
  fflush(stdout);
  scanf("%d",&access);
  printf("Enter number of threads 1,2,4 :");
  fflush(stdout);
  scanf("%d",&nothreads);
  pthread_t  pth[nothreads];
if(size==2)
{
          // if 2 is selected assign 1024 bytes to read or write from disk
size=(long int)(1024);
}
if(size==3)
{   // if 3 is selected assign 1024*2014 bytes to read or write from disk
          size=(long int)(1024*1024);


}


  if (pthread_mutex_init(&lock, NULL) != 0)
    {
       printf("\n mutex init failed\n");
       return 1;
```

```
    }
      char *fileName="trial.txt";
          char *f="hs";
          fileDesc = open(fileName,O_CREAT|O_RDWR,S_IRWXU);

      for(i=0;i<=(1024*1024*20);i++)
       {
         int currentPageSize =write(fileDesc,f,2);
       }

      // create given no of threads  and perform operations (read/write)
for(i=0;i<nothreads;i++)
{
         if(access==1 && operation==1)
         {
                  //this block is for read operation sequential access
                  pthread_create(&pth[i],NULL,read_thread_seq,(long int)size);
                  //creates a thread and calls read_thread function
         }
         if(access==2 && operation==1 )
         {                  //this block is for read operation random access
                            pthread_create(&pth[i],NULL,read_thread_random,(long int)size);
                            //creates a thread and calls read_thread_random function

         }
         if(access==1 && operation==2)
         {         //this block is for write operation using sequential access
                  pthread_create(&pth[i],NULL,write_thread_seq,(long int)size);
                  //creates a thread and calls write_thread_seq function
```

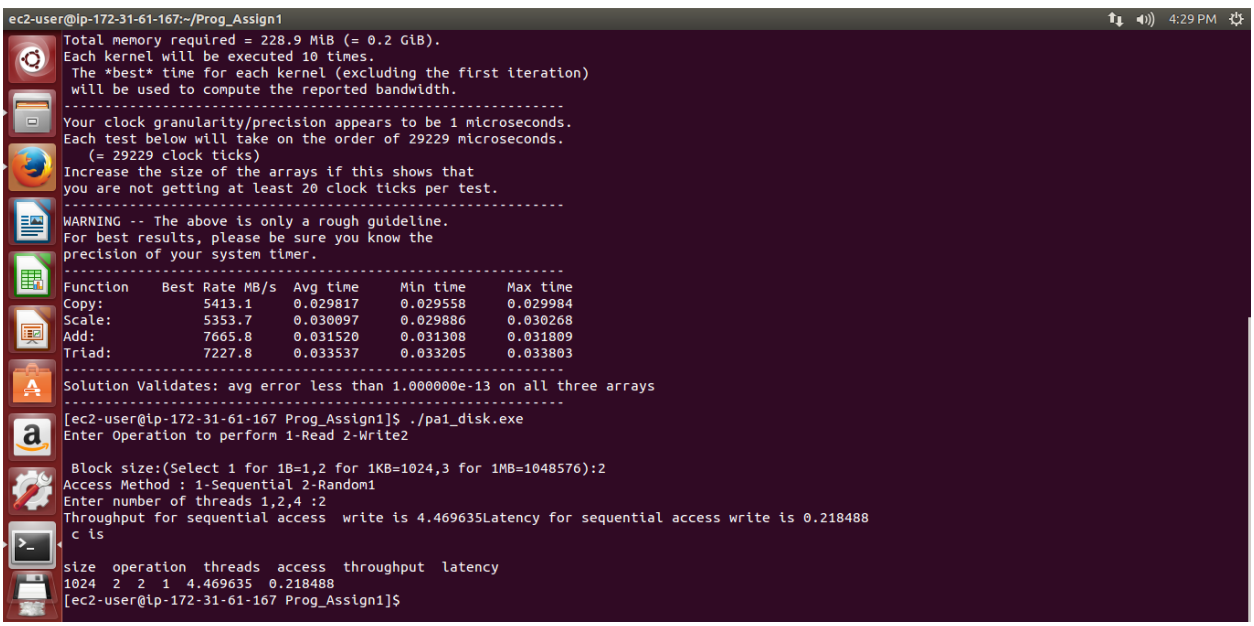
```
        }
        if(access==2 && operation==2)
        {       //this block is for write operation using random access
                        pthread_create(&pth[i],NULL,write_thread_random,(long int)size);
                        //creates a thread and calls write_thread_random function

        }

        }
// joining the thread with other thread
for(i=0;i<nothreads;i++)
{
         pthread_join(pth[i],&dataTime);
        timeTaken=(float)timeTaken+dataTime; // add the time taken for all threads
}
        timeTaken=(timeTaken)/nothreads;
        throughput=(float)(1000000*20)/(timeTaken);  //(no of loops*size*2/time taken)
   latency=(float)(timeTaken*1000*size)/(1000000*20);


        if(access==1 && operation==1)
         {
           printf("Throughput for sequential access read is %f ",(float)(throughput)/(1024*1024));
           printf("Latency for sequential access read is %f",latency);
         }

        if(access==2 && operation==1)
        {
```

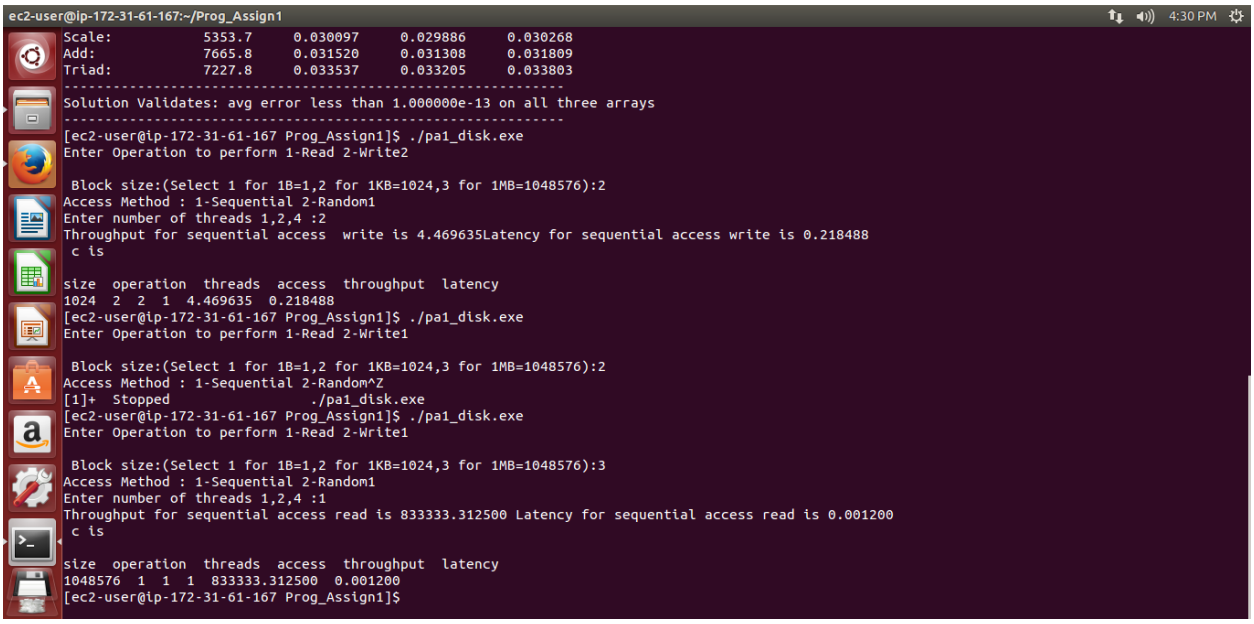
```
printf("Throughput for random access read is  %f",(float)(throughput)/(1024*1024));
printf("Latency for random access read is %f",latency);

        }           if(access==1 && operation==2)
         {
            printf("Throughput for sequential access  write is %f",throughput/(1024*1024));
            printf("Latency for sequential access write is %f",latency);

         }

        if(access==2 && operation==2)
        {
printf("Throughput for random access write is %f",throughput/(1024*1024));
            printf("Latency for random access write is %f",latency);

        }
        //store result in file
                    int filesc;
            char *fs="values.txt";
            filesc = open(fs,O_CREAT|O_RDWR,S_IRWXU);

                    char *c = (char *)malloc(sizeof(char)*10000);
                    char *s = (char *)malloc(sizeof(char)*100);
                            c[0] = '\0';
                            strcat(c,"\r\n");
                            strcat(c,"size");
                            strcat(c,"  ");
                            strcat(c,"operation");
                            strcat(c,"  ");
```

```
strcat(c,"threads");
strcat(c,"  ");
strcat(c,"access");
strcat(c,"  ");
strcat(c,"throughput");
strcat(c,"  ");
strcat(c,"latency");
strcat(c,"\r\n");

sprintf (s,"%d",(int)size);
strcat(c,s);
strcat(c,"  ");

sprintf (s,"%d",(int)operation);
strcat(c,s);
strcat(c,"  ");

sprintf (s,"%d",(int)nothreads);
strcat(c,s);
strcat(c,"  ");

sprintf (s,"%d",(int)access);
strcat(c,s);
strcat(c,"  ");

sprintf (s,"%f",(float)throughput/(1024*1024));
strcat(c,s);
strcat(c,"  ");
sprintf (s,"%f",(float)latency);
```

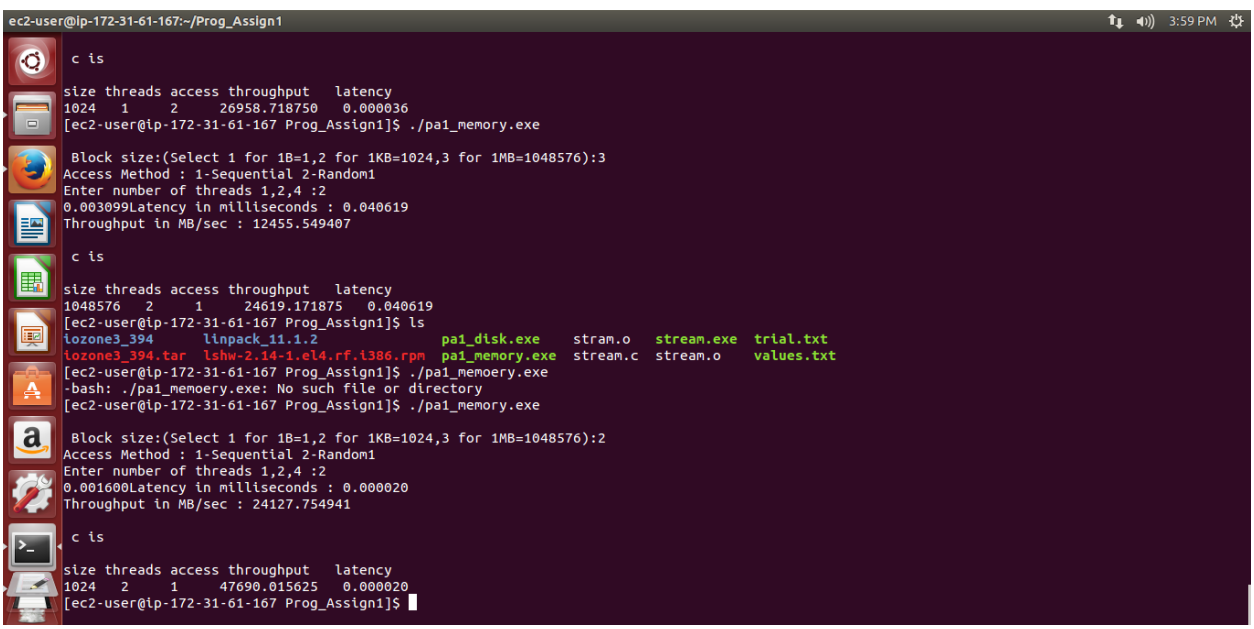
```
                    strcat(c,s);
                    strcat(c,"\r\n");


                    printf("\n c is \n %s",c);
                write(filesc,c,strlen(c));


}
```

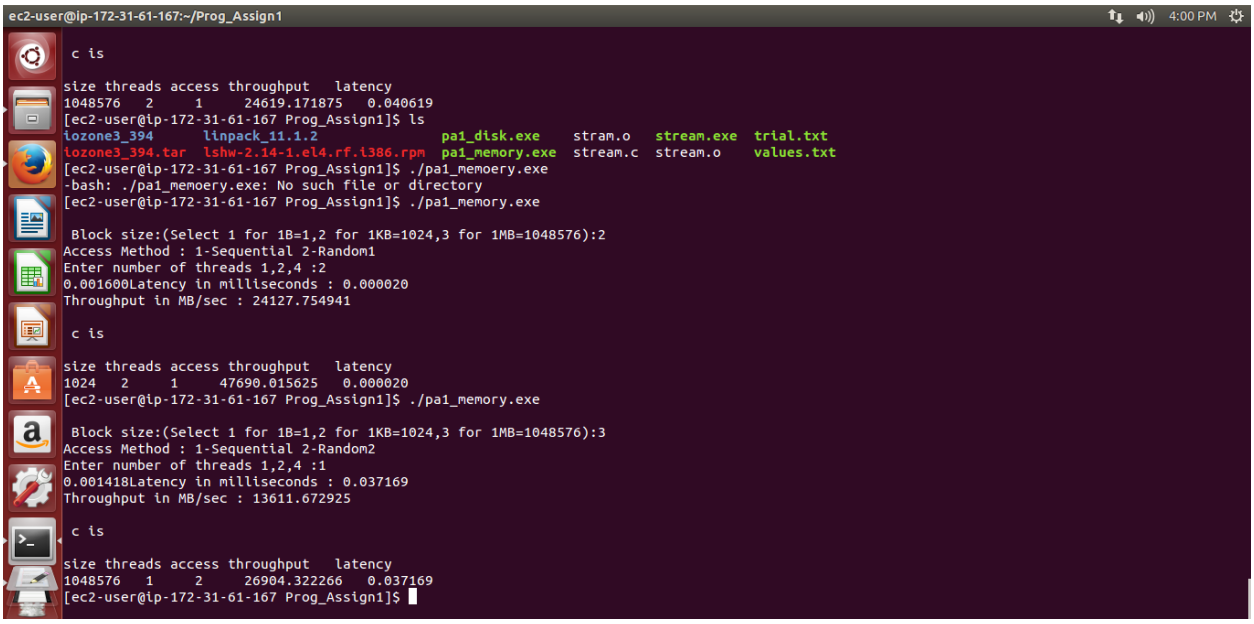
Disk Benchmark Screenshots





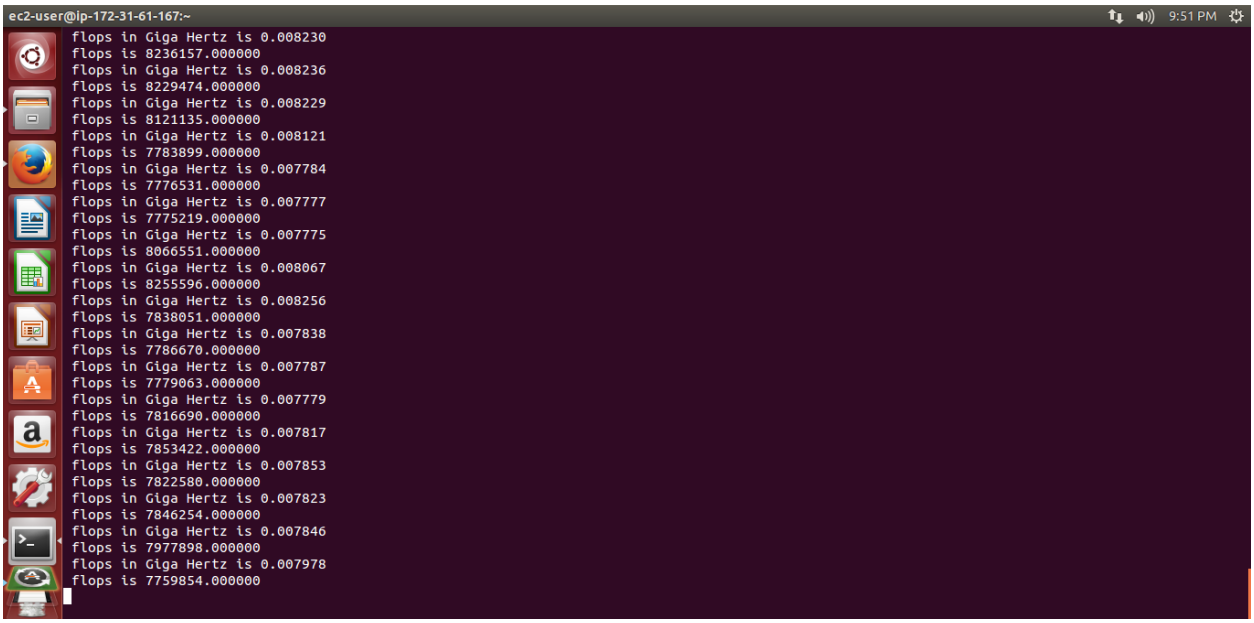

Memory Screenshot





```
ec2-user@ip-172-31-61-167:~/Prog_Assign1                                    4:00 PM
 c is

size threads access throughput   latency
1048576   2     1     24619.171875   0.040619
[ec2-user@ip-172-31-61-167 Prog_Assign1]$ ls
iozone3_394      linpack_11.1.2            pa1_disk.exe    stram.o   stream.exe  trial.txt
iozone3_394.tar  lshw-2.14-1.el4.rf.i386.rpm  pa1_memory.exe  stream.c  stream.o    values.txt
[ec2-user@ip-172-31-61-167 Prog_Assign1]$ ./pa1_memoery.exe
-bash: ./pa1_memoery.exe: No such file or directory
[ec2-user@ip-172-31-61-167 Prog_Assign1]$ ./pa1_memory.exe

 Block size:(Select 1 for 1B=1,2 for 1KB=1024,3 for 1MB=1048576):2
Access Method : 1-Sequential 2-Random1
Enter number of threads 1,2,4 :2
0.001600Latency in milliseconds : 0.000020
Throughput in MB/sec : 24127.754941

 c is

size threads access throughput   latency
1024   2     1     47690.015625   0.000020
[ec2-user@ip-172-31-61-167 Prog_Assign1]$ ./pa1_memory.exe

 Block size:(Select 1 for 1B=1,2 for 1KB=1024,3 for 1MB=1048576):3
Access Method : 1-Sequential 2-Random2
Enter number of threads 1,2,4 :1
0.001418Latency in milliseconds : 0.037169
Throughput in MB/sec : 13611.672925

 c is

size threads access throughput   latency
1048576   1     2     26904.322266   0.037169
[ec2-user@ip-172-31-61-167 Prog_Assign1]$
```

Cpu Screenshot



```
ec2-user@ip-172-31-61-167:~                                                 9:51 PM
 flops in Giga Hertz is 0.008230
 flops is 8236157.000000
 flops in Giga Hertz is 0.008236
 flops is 8229474.000000
 flops in Giga Hertz is 0.008229
 flops is 8121135.000000
 flops in Giga Hertz is 0.008121
 flops is 7783899.000000
 flops in Giga Hertz is 0.007784
 flops is 7776531.000000
 flops in Giga Hertz is 0.007777
 flops is 7775219.000000
 flops in Giga Hertz is 0.007775
 flops is 8066551.000000
 flops in Giga Hertz is 0.008067
 flops is 8255596.000000
 flops in Giga Hertz is 0.008256
 flops is 7838051.000000
 flops in Giga Hertz is 0.007838
 flops is 7786670.000000
 flops in Giga Hertz is 0.007787
 flops is 7779063.000000
 flops in Giga Hertz is 0.007779
 flops is 7816690.000000
 flops in Giga Hertz is 0.007817
 flops is 7853422.000000
 flops in Giga Hertz is 0.007853
 flops is 7822580.000000
 flops in Giga Hertz is 0.007823
 flops is 7846254.000000
 flops in Giga Hertz is 0.007846
 flops is 7977898.000000
 flops in Giga Hertz is 0.007978
 flops is 7759854.000000
```

```
ec2-user@ip-172-31-61-167:~                                                    9:44 PM
ws.comks
ssh: Could not resolve hostname ec2-54-85-29-163.compute-1.amazonaws.comks: Name
 or service not known
sushma@ubuntu:~$ ssh -i "sushma.pem" ec2-user@ec2-54-85-29-163.compute-1.amazona
ws.com:


^Z
[1]+  Stopped                 ssh -i "sushma.pem" ec2-user@ec2-54-85-29-163.comp
ute-1.amazonaws.com:
sushma@ubuntu:~$ scp -i sushma.pem pa1/pa1_disk.exe ec2-user@ec2-54-85-29-163.co
mpute-1.amazonaws.com:
pa1_disk.exe                                  100%   12KB  12.2KB/s   00:01
sushma@ubuntu:~$ scp -i sushma.pem pa1/pa1_disk.exe ec2-user@ec2-54-85-29-163.co
mpute-1.amazonaws.com:
pa1_disk.exe                                  100%   12KB  12.2KB/s   00:00
sushma@ubuntu:~$ ssh -i "sushma.pem" ec2-user@ec2-54-85-29-163.compute-1.amazona
ws.com
Last login: Sat Feb 13 00:50:24 2016 from 208-59-158-221.c3-0.mcm-ubr1.chi-mcm.i
l.cable.rcn.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
[ec2-user@ip-172-31-61-167 ~]$ ls
l_lpk_p_11.1.2.005.tgz  pa1_cpu_first.c  pa1_memory.exe  sushma@ubuntu
makefile                pa1_disk.exe     Prog_Assign1    trial.txt
pa1_cpu.exe             pa1_first.exe    sushma.pem      values.txt
[ec2-user@ip-172-31-61-167 ~]$ ./pa1_cpu.exe
Enter Operation to Perform 1 int 2 float1
Enter threads2
1455342199 14553421971455342199 1455342197 Iops in Hertz is 200000000.000000 Iop
[ec2-user@ip-172-31-61-167 ~]$
```

```
ec2-user@ip-172-31-61-167:~                                                    7:34 PM
Downloads          l_mklb_p_11.3.1.002          mysecond.c                    stream.c                                  Videos
examples.desktop   l_mklb_p_11.3.1.002.tgz      output                        streamfile
first.exe          lshw-2.17-1.el3.rf.i386.rpm  pa1                           stream.o
sushma@ubuntu:~$ ssh -i "sushma.pem" ec2-user@ec2-54-85-29-163.compute-1.amazonaws.com
Last login: Fri Feb 12 03:01:00 2016 from 208-59-158-221.c3-0.mcm-ubr1.chi-mcm.il.cable.rcn.com

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
[ec2-user@ip-172-31-61-167 ~]$ ls
l_lpk_p_11.1.2.005.tgz  makefile  pa1_cpu.exe  pa1_cpu_first.c  pa1_disk.exe  Prog_Assign1  trial.txt  values.txt
[ec2-user@ip-172-31-61-167 ~]$ ./pa1_cpu.exe
Enter Operation to Perform 1 int 2 float1
Enter threads1
1455247873 1455247872 Iops in Hertz is 400000000.000000 Iops in Giga Hertz is 0.400000[ec2-user@ip-172-31-61-167 ~]$
[ec2-user@ip-172-31-61-167 ~]$
[ec2-user@ip-172-31-61-167 ~]$ ./pa1_cpu.exe
Enter Operation to Perform 1 int 2 float2
Enter threads1
^C
[ec2-user@ip-172-31-61-167 ~]$ ^C
[ec2-user@ip-172-31-61-167 ~]$ ./pa1_cpu.exe
Enter Operation to Perform 1 int 2 float1
Enter threads2
1455247934 14552479321455247934 1455247932 Iops in Hertz is 200000000.000000 Iops in Giga Hertz is 0.200000[ec2-user@ip-172-31-61-167 ~]$ ./pa1_
cpu.exe
Enter Operation to Perform 1 int 2 float1
Enter threads4
1455247951 14552479481455247951 14552479481455247951 14552479481455247951 1455247948 Iops in Hertz is 133333336.000000 Iops in Giga Hertz is 0.1
33333[ec2-user@ip-172-31-61-167 ~]$ ./pa1_cpu.exe
Enter Operation to Perform 1 int 2 float2
Enter threads1
 Flops in Hertz is 21052632.000000 Flops in Giga Hertz is 0.021053[ec2-user@ip-172-31-61-167 ~]$
```