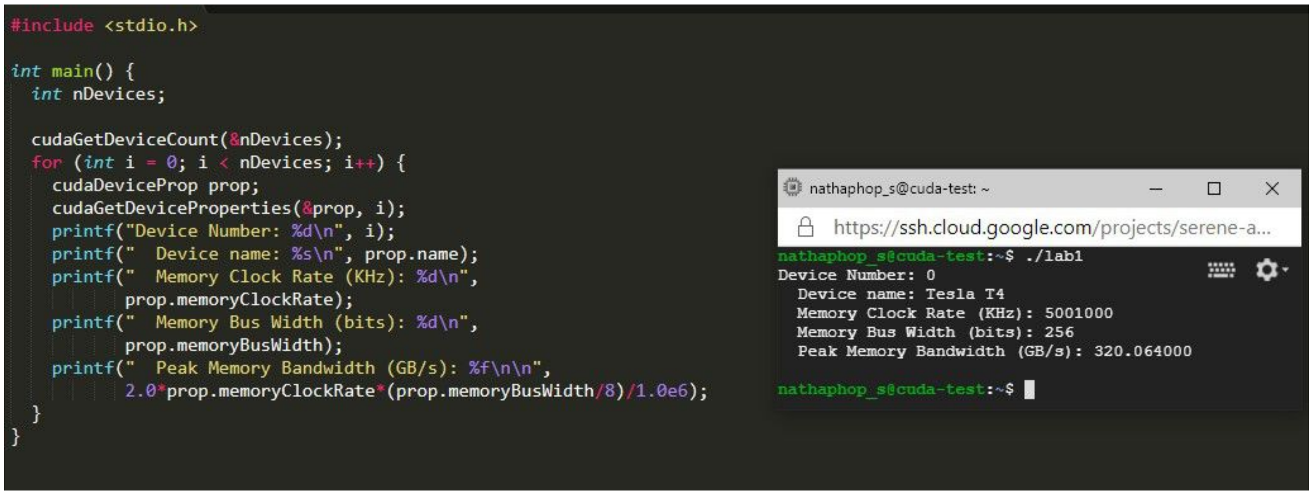
Nathaphop Sundarabhogin 60070503420
Supakit Artsamat 60070503461

LAB 7

LAB 7.1 First Run GPU Program

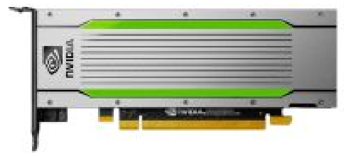


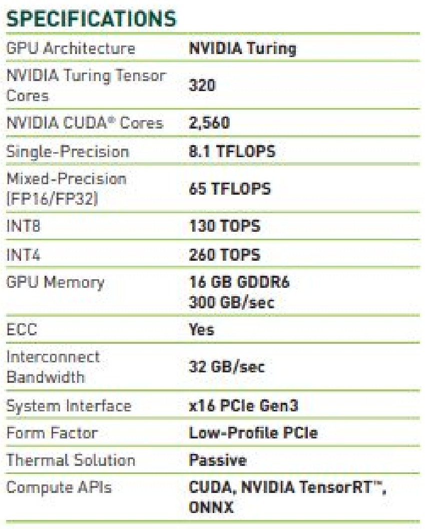
```
#include <stdio.h>

int main() {
  int nDevices;

  cudaGetDeviceCount(&nDevices);
  for (int i = 0; i < nDevices; i++) {
    cudaDeviceProp prop;
    cudaGetDeviceProperties(&prop, i);
    printf("Device Number: %d\n", i);
    printf("  Device name: %s\n", prop.name);
    printf("  Memory Clock Rate (KHz): %d\n",
           prop.memoryClockRate);
    printf("  Memory Bus Width (bits): %d\n",
           prop.memoryBusWidth);
    printf("  Peak Memory Bandwidth (GB/s): %f\n\n",
           2.0*prop.memoryClockRate*(prop.memoryBusWidth/8)/1.0e6);
  }
}
```

```
nathaphop_s@cuda-test:~$ ./lab1
Device Number: 0
  Device name: Tesla T4
  Memory Clock Rate (KHz): 5001000
  Memory Bus Width (bits): 256
  Peak Memory Bandwidth (GB/s): 320.064000

nathaphop_s@cuda-test:~$
```

The GPU that is used in the calculation is NVIDIA Tesla T4. Nvidia Tesla T4 is an enterprise graphic card, which contains 2,560 CUDA cores manufactured in Turing architecture together with 320 tensor cores for AI capabilities. For more comparison to show the difference between enterprise graphic card or GPGPU with customer graphic card or GPU, we would like to select a graphic card with the same architecture and we select Nvidia Geforce RTX2080 to compare with it.

Overall specification of Nvidia Tesla T4



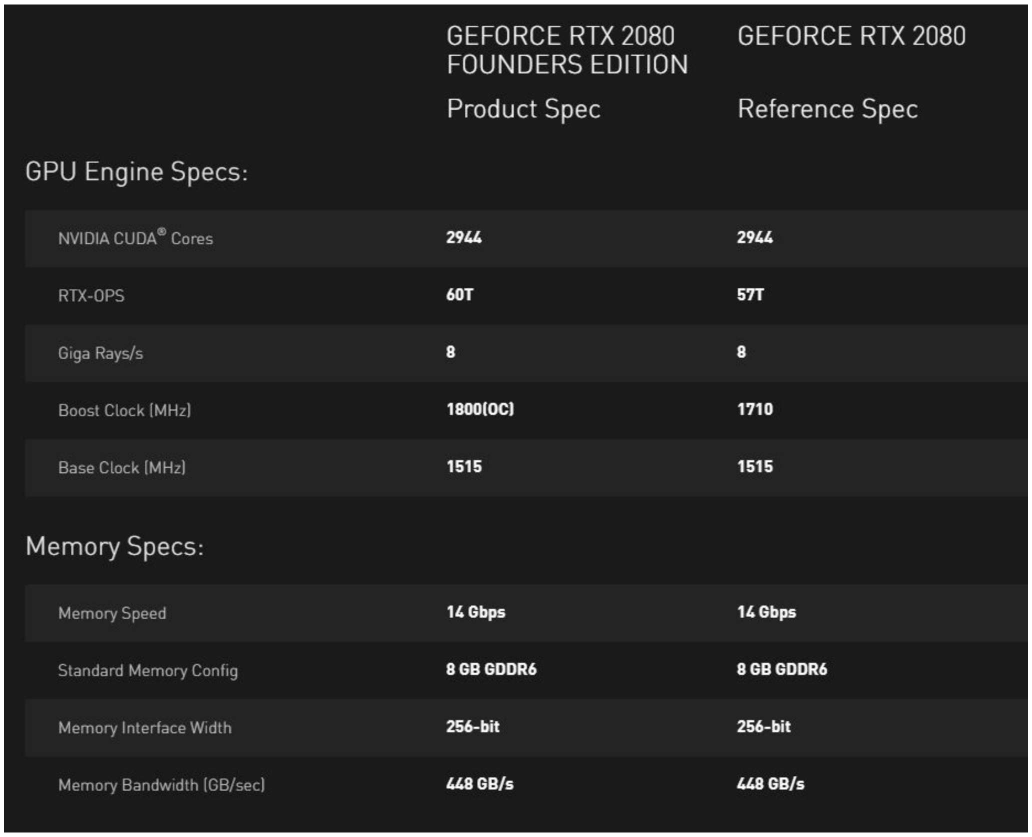


SPECIFICATIONS

GPU Architecture	NVIDIA Turing
NVIDIA Turing Tensor Cores	320
NVIDIA CUDA® Cores	2,560
Single-Precision	8.1 TFLOPS
Mixed-Precision (FP16/FP32)	65 TFLOPS
INT8	130 TOPS
INT4	260 TOPS
GPU Memory	16 GB GDDR6 300 GB/sec
ECC	Yes
Interconnect Bandwidth	32 GB/sec
System Interface	x16 PCIe Gen3
Form Factor	Low-Profile PCIe
Thermal Solution	Passive
Compute APIs	CUDA, NVIDIA TensorRT™, ONNX

Nathaphop Sundarabhogin 60070503420
Supakit Artsamat 60070503461

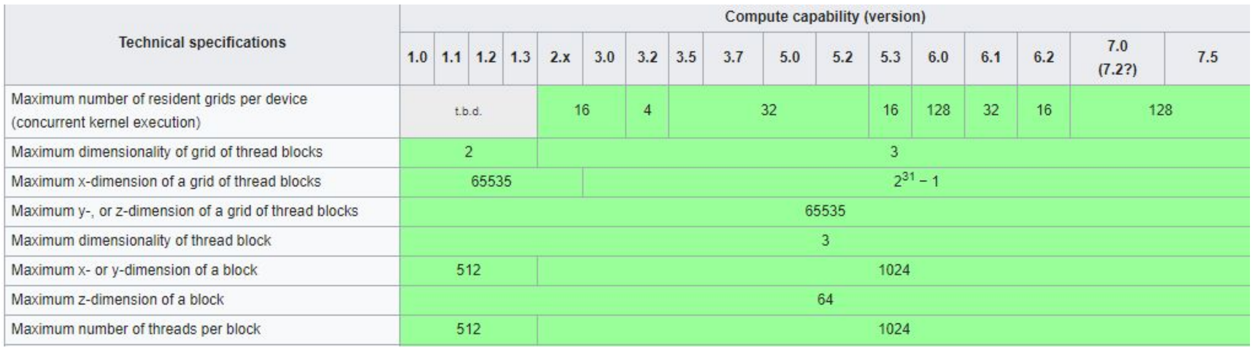
Overall specification of Nvidia Geforce RTX2080



	GEFORCE RTX 2080 FOUNDERS EDITION Product Spec	GEFORCE RTX 2080 Reference Spec
GPU Engine Specs:		
NVIDIA CUDA® Cores	2944	2944
RTX-OPS	60T	57T
Giga Rays/s	8	8
Boost Clock (MHz)	1800(OC)	1710
Base Clock (MHz)	1515	1515
Memory Specs:		
Memory Speed	14 Gbps	14 Gbps
Standard Memory Config	8 GB GDDR6	8 GB GDDR6
Memory Interface Width	256-bit	256-bit
Memory Bandwidth (GB/sec)	448 GB/s	448 GB/s

Nvidia Geforce RTX2080 and Tesla T4 are using the same Turing architecture and it has the same Compute Capability version 7.5, it means that if both graphic cards have the same amount of processing cores with the same operation frequency, it's likely to have the same performance.

The compute capability of version 7.5, the dimension of the grid can be 3 dimensions, x dimension can be $2^{31} - 1$ grids and y/z dimension can be 65535 grids. Dimension of the block can be 3 dimensions as the same as the grid dimension, x and y dimension can have 1024 blocks and z dimension can have 64 blocks. And each block can have 1024 threads.



Technical specifications	Compute capability (version)																
	1.0	1.1	1.2	1.3	2.x	3.0	3.2	3.5	3.7	5.0	5.2	5.3	6.0	6.1	6.2	7.0 (7.2?)	7.5
Maximum number of resident grids per device (concurrent kernel execution)	t.b.d.	t.b.d.	t.b.d.	t.b.d.	16	16	4	32	32	32	32	16	128	32	16	128	128
Maximum dimensionality of grid of thread blocks	2	2	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3
Maximum x-dimension of a grid of thread blocks	65535	65535	65535	65535	65535	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$	$2^{31} - 1$
Maximum y-, or z-dimension of a grid of thread blocks	65535	65535	65535	65535	65535	65535	65535	65535	65535	65535	65535	65535	65535	65535	65535	65535	65535
Maximum dimensionality of thread block	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Maximum x- or y-dimension of a block	512	512	512	512	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024
Maximum z-dimension of a block	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64	64
Maximum number of threads per block	512	512	512	512	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024

But Geforce RTX2080 has a bit higher core count by 384 cores with a higher core frequency, so the result is RTX2080 capable of processing faster without considering graphic card memory size and memory speed.

However, both graphic cards are built for different purposes. Tesla T4 is built specifically for install in the workstation or server, which is capable of installing many graphic cards on the same computer. That is why Tesla T4 has less core count and lower core clock frequency, it needs to be a compact size to make it able to fit inside the server. To make it more clear generally, Geforce RTX2080 will consume space at least 2 PCI-e slots (connect only 1 slot and useless 1 slot) but Tesla T4 uses only 1 slot or low-profile to make it can pack more tightly inside the computer.

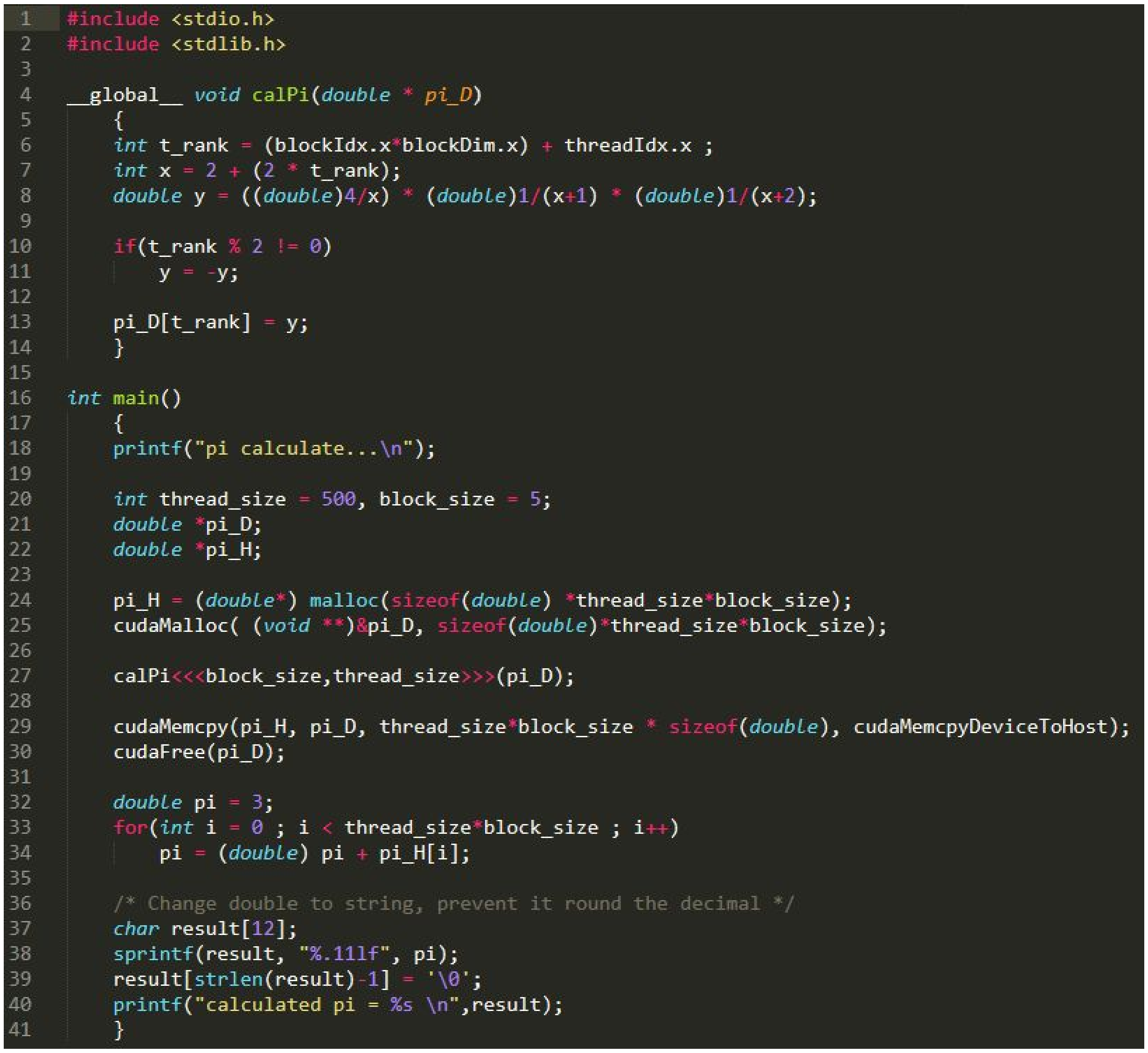
Moreover, space is not the only advantage. The result of less core count and lower core clock frequency will make Tesla T4 more power-efficient than Geforce RTX2080, Tesla T4 consumes only 70 watts but Geforce RTX2080 can consume up to 215 watts, so in the long term Tesla T4 and Geforce RTX2080 will have a big difference in operating cost.

Let's focus more on the workload of both graphic cards, it is a difference in workload type, for customer graphic cards mostly deal with an interactive task like a game, show user interface and decode video file. Unlike in the workstation and server are mostly dealing with the batch task. For this reason, Tesla T4 will have more graphic memory to hold more data near the processor for faster computation. And to make the memory more stable away from data faults in a memory module Tesla T4 supports ECC memory (Error-correcting code memory).

Nathaphop Sundarabhogin 60070503420
Supakit Artsamat 60070503461

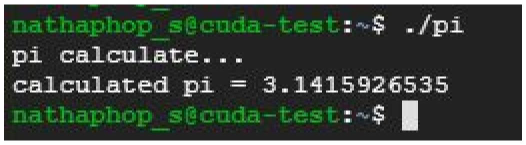
LAB 7.2 Estimate PI

Code:



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

__global__ void calPi(double * pi_D)
    {
    int t_rank = (blockIdx.x*blockDim.x) + threadIdx.x ;
    int x = 2 + (2 * t_rank);
    double y = ((double)4/x) * (double)1/(x+1) * (double)1/(x+2);

    if(t_rank % 2 != 0)
        y = -y;

    pi_D[t_rank] = y;
    }

int main()
    {
    printf("pi calculate...\n");

    int thread_size = 500, block_size = 5;
    double *pi_D;
    double *pi_H;

    pi_H = (double*) malloc(sizeof(double) *thread_size*block_size);
    cudaMalloc( (void **)&pi_D, sizeof(double)*thread_size*block_size);

    calPi<<<block_size,thread_size>>>(pi_D);

    cudaMemcpy(pi_H, pi_D, thread_size*block_size * sizeof(double), cudaMemcpyDeviceToHost);
    cudaFree(pi_D);

    double pi = 3;
    for(int i = 0 ; i < thread_size*block_size ; i++)
        pi = (double) pi + pi_H[i];

    /* Change double to string, prevent it round the decimal */
    char result[12];
    sprintf(result, "%.11lf", pi);
    result[strlen(result)-1] = '\0';
    printf("calculated pi = %s \n",result);
    }
```

Result:



```
nathaphop_s@cuda-test:~$ ./pi
pi calculate...
calculated pi = 3.1415926535
nathaphop_s@cuda-test:~$
```

The estimated pi is 3.1415926535 (10 decimals) which is the same as the pi value, 3.1415926535 ref: https://en.wikipedia.org/wiki/Pi.

Nathaphop Sundarabhogin 60070503420
Supakit Artsamat 60070503461

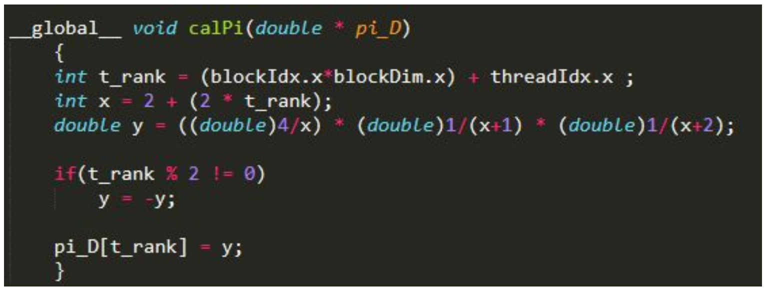
Description:

We estimate the Pi value by using Nilakantha Series, which the formula is

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \cdots$$

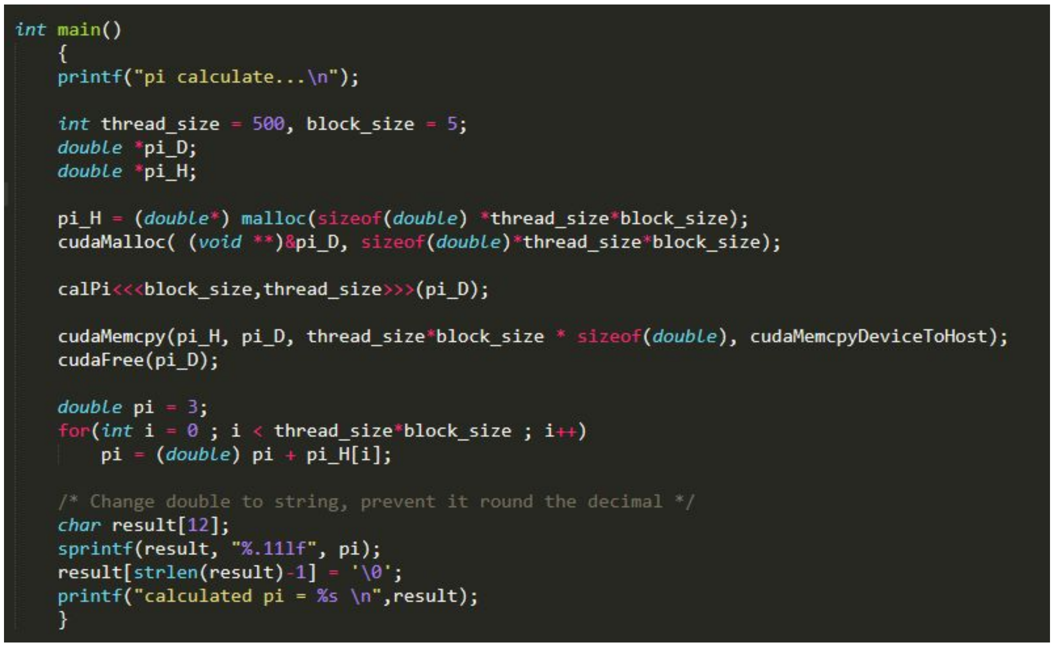
Reference: https://en.wikipedia.org/wiki/Pi

From the formula, after the first term, it is a series of $\pm \frac{4}{i \times i+1 \times i+2}$ which i is positive even integer. We find each series term in the formula by calculating in the CUDA and this is the function that runs in each thread.



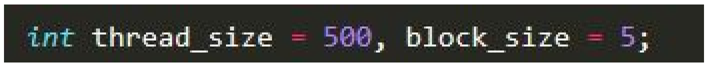
```
__global__ void calPi(double * pi_D)
    {
    int t_rank = (blockIdx.x*blockDim.x) + threadIdx.x ;
    int x = 2 + (2 * t_rank);
    double y = ((double)4/x) * (double)1/(x+1) * (double)1/(x+2);

    if(t_rank % 2 != 0)
        y = -y;

    pi_D[t_rank] = y;
    }
```

In the main function allocate memory space in both CPU and GPU, call the calPi function, get the data from GPU after calculation, sum the pi value together, and then print the estimated pi.



```
int main()
    {
    printf("pi calculate...\n");

    int thread_size = 500, block_size = 5;
    double *pi_D;
    double *pi_H;

    pi_H = (double*) malloc(sizeof(double) *thread_size*block_size);
    cudaMalloc( (void **)&pi_D, sizeof(double)*thread_size*block_size);

    calPi<<<block_size,thread_size>>>(pi_D);

    cudaMemcpy(pi_H, pi_D, thread_size*block_size * sizeof(double), cudaMemcpyDeviceToHost);
    cudaFree(pi_D);

    double pi = 3;
    for(int i = 0 ; i < thread_size*block_size ; i++)
        pi = (double) pi + pi_H[i];

    /* Change double to string, prevent it round the decimal */
    char result[12];
    sprintf(result, "%.11lf", pi);
    result[strlen(result)-1] = '\0';
    printf("calculated pi = %s \n",result);
    }
```

Also, we know that the GPU that is used in calculation is NVIDIA Tesla T4 and it has 2560 cores inside. we tried to use only the minimum core in the GPU to decrease the context switch inside. So, we used only 5 blocks and 500 threads per block which is 2500 threads. The number of threads doesn't exceed the number of cores in the GPU to make it more efficient.



```
int thread_size = 500, block_size = 5;
```