

PI computation GPU acceleration using Monte Carlo simulation on CUDA

General description

The number π is a mathematical constant. Originally defined as the ratio of a circle's circumference to its diameter, it now has various equivalent definitions and appears in many formulas in all areas of mathematics and physics. It is approximately equal to 3.14159.

Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. One of the basic examples of getting started with the Monte Carlo algorithm is the estimation of π .

The idea is to simulate random (x, y) points in a 2-D plane with domain as a square of side 1 unit. Imagine a circle inside the same domain with same diameter and inscribed into the square. We then calculate the ratio of number points that lied inside the circle and total number of generated points.

We know that area of the square is 1 unit sq while that of circle is:

$$\pi * \left(\frac{1}{2}\right)^2 = \frac{\pi}{4}$$

Now for a very large number of generated points,

$$\frac{\text{area of the circle}}{\text{area of the square}} = \frac{\text{no. of points generated inside the circle}}{\text{total no. of points generated or no. of points generated inside the square}}$$

that is,

$$\pi = 4 * \frac{\text{no. of points generated inside the circle}}{\text{no. of points generated inside the square}}$$

Parallelization methods

Naive approach

In our first implementation, we used a naive approach to distribute the computation of the value of π on CUDA blocks.

On the host device, we generated two vectors with random (x, y) pairs.

Then we copied the vectors into the memory of the graphics device. After that, we launched the CUDA kernel, which had the task of counting how many points land inside the circle and write the value of the counter in the **sampleCountPerThreads** shared vector. The first thread from each block then sums up the values from the thread counter and stores the result in a **device_sampleCountPerBlock** vector. The host device then does the computation of π , after summing up the values from the **device_sampleCountPerBlock** vector.

Deficitary aspects and optimizations

The first and most important optimization would be generating the random samples directly on the GPU threads. This would eliminate the severe overhead on the CPU.

Hardware & software devices used

| CPU | GPU | OS | CUDA version |
|---------------------------------|----------------------------|----------------|--------------|
| Intel Core i7-4790 @ 3.6 GHz | GeForce GTX 660, 2048MB | Windows 10 Pro | 10.2 |

Results analysis

| No. of samples | CPU version (ms) | | GPU version (ms) | | | | Acceleration |
|----------------|--------------------|----------------|--------------------|--------------------------|--------------------|--------------------------|--------------|
| | Samples generation | Pi computation | Samples generation | Copy data host -> device | Kernel computation | Copy data device -> host | |
| 10^5 | 13 | 1 | 12 | 0.2975 | 159.7696 | 0.0945 | 0.08 |
| 10^6 | 125 | 7 | 127 | 1.7931 | 159.7395 | 0.0543 | 0.46 |
| 10^7 | 1238 | 62 | 1235 | 16.0672 | 160.0351 | 0.0619 | 0.92 |
| 10^8 | 12255 | 510 | 12274 | 162.6636 | 203.5319 | 0.2047 | 1.01 |

```
Generating host X samples...
Generating host Y samples...
Generated vectors in 12255.000000 ms
pi = 3.141531
Pi calculus took 512.000000 ms
Whole process took 12767.000000 ms
```

Output for the CPU version

```
Generating host X samples...
Generating host Y samples...
Generated vectors in: 12274.000000 ms
Device malloc randX...
Device malloc randY...
Copying randX from host to device...
Copying randY from host to device...
Malloc and copy from --proc-- to --graphics card-- time: 162.663620
Device malloc sampleCountPerBlock...
Launching kernel...
Kernel execution time: 203.531998
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.204768
pi = 3.141514
```

Output for the GPU version

Optimizations

1. Generate samples per threads

We refactored the code to generate the random samples on the device threads instead of the host. The setup parameters are:

NUM_BLOCKS=2048,
NUM_THREADS_PER_BLOCK=32 and
NUM_SAMPLES_PER_THREAD=2000

Each thread generates NUM_SAMPLES_PER_THREAD samples. The number of samples in circle per thread are then stored in a shared memory zone. We then select a master thread per block, which sums up all the counters from the threads. Those sums are stored in a vector on the device, which is then copied to the host. The host applies a simple algorithm to estimate pi, based on the values from the vector.

These are the results of the run:

```
Device malloc sampleCountPerBlock...
Launching kernel <0>...
Kernel execution time: 2480.241943
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.068288
Device memory used: 500Mb
Total processing time: 2572.000000 ms
Number of samples: 131072000
pi = 3.141555
```

We can notice by far how much faster this version is. We had a total processing time of about 16 seconds for 1e8 samples on the previous version, and now we managed to obtain 2.5 seconds for $1.3 \times 1e8$ samples.

2. Optimized memory using Unified Memory

We optimized the usage of the vector of sums stored on the device, by using *cudaMallocManaged*. Now both the CPU and GPU have direct access to the vector, so there's no need in copying it from the device to the host. This would mean losing some processing time.

The results showed us that we are wrong.

```
Launching kernel <0>...
Kernel execution time: 2483.098145
Device memory used: 500Mb
Total processing time: 3142.000000 ms
Number of samples: 131072000
pi = 3.141440
```

We noticed that the kernel execution time is approximately the same, we don't waste time on transferring data from the device to the host, but we have a higher total processing time.

This optimization would have been better on the previous implementation, where we transferred heavy data from the host to the device. Using Unified Memory, the data would have been brought to the device before calling the kernel (prefetch).

3. Local reduction on gpu with warp shuffle at the beginning

After adding warp shuffle reduction to count the number of samples in circle / block, we had a better result:

```
Launching kernel <0>...  
Kernel execution time: 2471.587402  
Device memory used: 500Mb  
Total processing time: 3055.000000 ms  
Number of samples: 131072000  
pi = 3.141530
```

We then reverted the Unified memory changes, and this is what we got:

```
Device malloc sampleCountPerBlock...  
Launching kernel <0>...  
Kernel execution time: 2471.448242  
Copying sampleCountPerBlock from device to host...  
Copy from --graphics card-- to --proc-- time: 0.082112  
Device memory used: 500Mb  
Total processing time: 2548.000000 ms  
Number of samples: 131072000  
pi = 3.141598
```

Other trials:

```
Device malloc sampleCountPerBlock...
Launching kernel (0)...
Kernel execution time: 1619.937256
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.089632
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (1)...
Kernel execution time: 1595.012085
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.068576
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (2)...
Kernel execution time: 1597.667236
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.068416
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (3)...
Kernel execution time: 1598.412598
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.064320
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (4)...
Kernel execution time: 1596.179565
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.111776
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (5)...
Kernel execution time: 1595.510864
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.127904
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (6)...
Kernel execution time: 1594.199707
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.083392
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (7)...
Kernel execution time: 1593.300293
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.102656
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (8)...
Kernel execution time: 1594.677856
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.101728
Device memory used: 248Mb

Device malloc sampleCountPerBlock...
Launching kernel (9)...
Kernel execution time: 1595.950439
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.134624
Device memory used: 248Mb
Total processing time: 16086.000000 ms
Number of samples: 655360000
pi = 3.141607
```

Trial 1 - with 248Mb of samples, we called the kernel several times, to increase the accuracy of the result, but we lost the advantage in the Total processing time side.

```

Device malloc sampleCountPerBlock...
Launching kernel (0)...
Kernel execution time: 2478.885742
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.099872
Device memory used: 500Mb

Device malloc sampleCountPerBlock...
Launching kernel (1)...
Kernel execution time: 2444.839355
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.087904
Device memory used: 500Mb

Device malloc sampleCountPerBlock...
Launching kernel (2)...
Kernel execution time: 2441.284424
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.082016
Device memory used: 500Mb

Device malloc sampleCountPerBlock...
Launching kernel (3)...
Kernel execution time: 2441.229492
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.087200
Device memory used: 500Mb

Device malloc sampleCountPerBlock...
Launching kernel (4)...
Kernel execution time: 2445.831543
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.120032
Device memory used: 500Mb

Device malloc sampleCountPerBlock...
Launching kernel (5)...
Kernel execution time: 2444.996826
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.089248
Device memory used: 500Mb

Device malloc sampleCountPerBlock...
Launching kernel (6)...
Kernel execution time: 2443.628174
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.079808
Device memory used: 500Mb

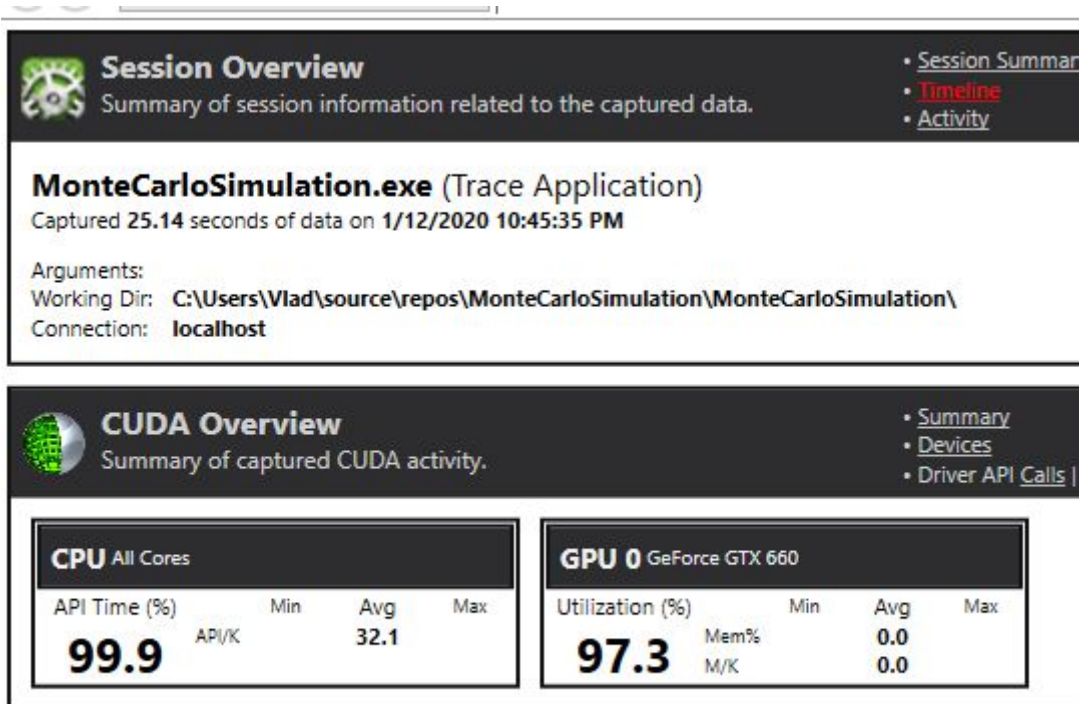
Device malloc sampleCountPerBlock...
Launching kernel (7)...
Kernel execution time: 2445.008301
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.102272
Device memory used: 500Mb

Device malloc sampleCountPerBlock...
Launching kernel (8)...
Kernel execution time: 2444.314697
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.168096
Device memory used: 500Mb

Device malloc sampleCountPerBlock...
Launching kernel (9)...
Kernel execution time: 2442.653076
Copying sampleCountPerBlock from device to host...
Copy from --graphics card-- to --proc-- time: 0.084544
Device memory used: 500Mb
Total processing time: 24585.000000 ms
Number of samples: 1310720000
pi = 3.141639

```

Trial 2 - with 500Mb of samples, we called the kernel several times, to increase even more the accuracy of the result, and we noticed that the Total processing time did not double.



Nsight output for Trial 2

```
Launching kernel (0)...  
Kernel execution time: 2467.950928  
Device memory used: 500Mb  
Launching kernel (1)...  
Kernel execution time: 2443.187500  
Device memory used: 500Mb  
Launching kernel (2)...  
Kernel execution time: 2444.209961  
Device memory used: 500Mb  
Launching kernel (3)...  
Kernel execution time: 2444.368896  
Device memory used: 500Mb  
Launching kernel (4)...  
Kernel execution time: 2443.156006  
Device memory used: 500Mb  
Launching kernel (5)...  
Kernel execution time: 2442.818848  
Device memory used: 500Mb  
Launching kernel (6)...  
Kernel execution time: 2442.350830  
Device memory used: 500Mb  
Launching kernel (7)...  
Kernel execution time: 2443.999268  
Device memory used: 500Mb  
Launching kernel (8)...  
Kernel execution time: 2443.906982  
Device memory used: 500Mb  
Launching kernel (9)...  
Kernel execution time: 2444.862549  
Device memory used: 500Mb  
Total processing time: 25528.000000 ms  
Number of samples: 1310720000  
pi = 3.141577
```

Trial 3 - After using the **Unified Memory** optimization strategy, with *cudaMallocManaged* instead of copying a vector from device to host, we noticed that the total processing time stays approximately the same.