Open in Colab

Matrix x Matrix multiplication

As a step by step instruction has been presented in tutorial 2, here is a time for a stand-alone practice.

Accelerate the serial, element-wise square matrix addition code using cuda kernel.

In [1]:
```
%%file matrix_add.cu

// This program computes a simple version of matrix multiplication
// By: Nick from CoffeeBeforeArch

#include <algorithm>
#include <cassert>
#include <cstdlib>
#include <functional>
#include <iostream>
#include <vector>

using std::cout;
using std::generate;
using std::vector;

__global__ void matrixMul(const int *a, const int *b, int *c, int N) {
  // Compute each thread's global row and column index
  int row = blockIdx.y * blockDim.y + threadIdx.y;
  int col = blockIdx.x * blockDim.x + threadIdx.x;

  // Iterate over row, and down column
  c[row * N + col] = 0;
  for (int k = 0; k < N; k++) {
    // Accumulate results for a single element
    c[row * N + col] += a[row * N + k] * b[k * N + col];
  }
}

// Check result on the CPU
void verify_result(vector<int> &a, vector<int> &b, vector<int> &c, int N) {
  for (int row = 0; row < N; row++) {
    for (int col = 0; col < N; col++) {
      int tmp = 0; // For every element in the row-column pair
      for (int k = 0; k < N; k++) {
        // Accumulate the partial results
        tmp += a[row * N + k] * b[k * N + col];
      }
      // Check against the CPU result
      assert(tmp == c[row * N + col]);
    }
  }
}

int main() {
  int N = 1 << 10;  // Matrix size of 1024 x 1024;

  // Size (in bytes) of matrix
  size_t bytes = N * N * sizeof(int);

  // Host vectors
  vector<int> h_a(N * N);
  vector<int> h_b(N * N);
  vector<int> h_c(N * N);

  // Initialize matrices
  generate(h_a.begin(), h_a.end(), []() { return rand() % 100; });
  generate(h_b.begin(), h_b.end(), []() { return rand() % 100; });

  // Allocate device memory
  int *d_a, *d_b, *d_c;
  cudaMalloc(&d_a, bytes);
  cudaMalloc(&d_b, bytes);
  cudaMalloc(&d_c, bytes);

  // Copy data to the device
  cudaMemcpy(d_a, h_a.data(), bytes, cudaMemcpyHostToDevice);
  cudaMemcpy(d_b, h_b.data(), bytes, cudaMemcpyHostToDevice);

  // Threads per CTA dimension
  int THREADS = 32;

  // Blocks per grid dimension (assumes THREADS divides N evenly)
  int BLOCKS = N / THREADS;

  // Use dim3 structs for block  and grid dimensions
  dim3 threads(THREADS, THREADS);
  dim3 blocks(BLOCKS, BLOCKS);

  // Launch kernel
  matrixMul<<<blocks, threads>>>(d_a, d_b, d_c, N);

  // Copy back to the host
  cudaMemcpy(h_c.data(), d_c, bytes, cudaMemcpyDeviceToHost);

  // Check result
  verify_result(h_a, h_b, h_c, N);

  cout << "COMPLETED SUCCESSFULLY\n";

  // Free memory on device
  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);

  return 0;
}
```

```
Writing matrix_add.cu
```

In [2]:
```
!echo "Check your GPU version"
!nvidia-smi
```

```
Check your GPU version
Sun Apr 16 17:40:29 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.108.03   Driver Version: 510.108.03   CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:01:00.0  On |                  N/A |
|  0%   58C    P5    31W / 250W |   1452MiB /  8192MiB |     28%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      1703      G   /usr/lib/xorg/Xorg                 96MiB |
|    0   N/A  N/A      2624      G   /usr/lib/xorg/Xorg                684MiB |
|    0   N/A  N/A      2819      G   /usr/bin/gnome-shell               66MiB |
|    0   N/A  N/A      3712      G   ...RendererForSitePerProcess       13MiB |
|    0   N/A  N/A      8175      G   ...features=BackForwardCache       10MiB |
|    0   N/A  N/A      8543      G   ...9572488673405287 64,131072     216MiB |
|    0   N/A  N/A     13016      G   ...AAAAAAAAA= --shared-files       35MiB |
|    0   N/A  N/A     14339      G   ...b/thunderbird/thunderbird      121MiB |
|    0   N/A  N/A     15643      G   ...RendererForSitePerProcess      169MiB |
+-----------------------------------------------------------------------------+
```

In [3]:
```
%%bash

CUDA_SUFF=70 # or CUDA_SUFF=35
nvcc -gencode arch=compute_${CUDA_SUFF},code=sm_${CUDA_SUFF} ./matrix_add.cu -o matrix_add
./matrix_add
```

```
COMPLETED SUCCESSFULLY
```

In [4]:
```
%%bash
# ls
# nvprof  ./matrix_add
nvprof  ./matrix_add
```

```
==38153== NVPROF is profiling process 38153, command: ./matrix_add
COMPLETED SUCCESSFULLY
==38153== Profiling application: ./matrix_add
==38153== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   79.92%  5.1594ms         1  5.1594ms  5.1594ms  5.1594ms  matrixMul(int const *,
int const *, int*, int)
                  13.46%  868.93us         2  434.47us  423.68us  445.25us  [CUDA memcpy HtoD]
                   6.62%  427.07us         1  427.07us  427.07us  427.07us  [CUDA memcpy DtoH]
      API calls:   93.22%  99.806ms         3  33.269ms  44.159us  99.717ms  cudaMalloc
                   6.30%  6.7416ms         3  2.2472ms  477.74us  5.7542ms  cudaMemcpy
                   0.30%  320.03us         3  106.68us  94.070us  113.00us  cudaFree
                   0.13%  137.88us       101  1.3650us     209ns  55.238us  cuDeviceGetAttribute
                   0.03%  28.029us         1  28.029us  28.029us  28.029us  cuDeviceGetName
                   0.01%  15.537us         1  15.537us  15.537us  15.537us  cudaLaunchKernel
                   0.01%  8.0320us         1  8.0320us  8.0320us  8.0320us  cuDeviceGetPCIBusId
                   0.00%  2.4130us         3     804ns     347ns  1.2840us  cuDeviceGetCount
                   0.00%  1.4810us         2     740ns     299ns  1.1820us  cuDeviceGet
                   0.00%     747ns         1     747ns     747ns     747ns  cuDeviceTotalMem
                   0.00%     456ns         1     456ns     456ns     456ns  cuDeviceGetUuid
```

What is the difference between 'GPU activities' and 'API calls' in the results of 'nvprof'?

Answer from https://forums.developer.nvidia.com/t/what-is-the-difference-between-gpu-activities-and-api-calls-in-the-results-of-nvprof/71338/1

Section 'GPU activities' list activities which execute on the GPU like CUDA kernel, CUDA memcpy, CUDA memset. And timing information here represents the execution time on the GPU.

Section 'API Calls' list CUDA Runtime/Driver API calls. And timing information here represents the execution time on the host.

For example, CUDA kernel launches are asynchronous from the point of view of the CPU. It returns immediately, before the kernel has completed, and perhaps before the kernel has even started. This time is captured for the Launch API like cuLaunchKernel in the 'API Calls' section. Eventually kernel starts execution on the GPU and runs to the completion. This time is captured for kernel in the 'GPU activities'.

In [5]:
```
%%bash
nvprof --print-gpu-trace ./matrix_add --benchmark
```

```
==38225== NVPROF is profiling process 38225, command: ./matrix_add --benchmark
COMPLETED SUCCESSFULLY
```

```
==38225== Profiling application: ./matrix_add --benchmark
==38225== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Thr
oughput  SrcMemType  DstMemType           Device   Context    Stream  Name
308.40ms  445.57us                    -               -         -         -         -  4.0000MB  8.7
669GB/s     Pageable      Device  NVIDIA GeForce          1         7  [CUDA memcpy HtoD]
308.94ms  426.95us                    -               -         -         -         -  4.0000MB  9.1
493GB/s     Pageable      Device  NVIDIA GeForce          1         7  [CUDA memcpy HtoD]
309.37ms  5.2208ms            (32 32 1)       (32 32 1)        28        0B        0B         -
      -           -           -  NVIDIA GeForce          1         7  matrixMul(int const *, int con
st *, int*, int) [116]
314.60ms  428.26us                    -               -         -         -         -  4.0000MB  9.1
212GB/s      Device    Pageable  NVIDIA GeForce          1         7  [CUDA memcpy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by th
e CUDA driver and/or tools and can be more than what the compiler shows.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy
```

In [ ]: