Profiling CUDA kernels using Nsight developer tools

Shweta Pandey

PhD student,

Indian Institute of Science

Executive summary

- Overview of a CUDA program
- Debugging performance of a CUDA program
- Overview of Nvidia Nsight profiling tools
- Tutorial

Overview of a CUDA program

1. Allocate data on GPU
2. Copy input data from CPU to GPU
3. Launch the CUDA kernel
4. Copy output results from GPU to CPU
5. Free allocated memory

```
matmul_naive<<<grid, block>>>(dA, dB, dC, N);
```

```
__global__ void matmul_naive(const float* A, const float* B,
                             float* C, int N)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    float sum = 0.f;
    for (int k = 0; k < N; ++k)
        sum += A[row * N + k] * B[k * N + col];
    C[row * N + col] = sum;
}
```

```
cudaFree(dA); cudaFree(dB); cudaFree(dC);
```

Debugging perf. of CUDA programs

```
__global__ void matmul_naive(const float* A, const float* B,
                             float* C, int N)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    if (row < N && col < N) {
        float sum = 0.f;
        for (int k = 0; k < N; ++k)
            sum += A[row * N + k] * B[k * N + col];
        C[row * N + col] = sum;
    }
}
```

```
int main(int argc, char** argv)
{
    /* Host code and allocations commented out */

    float *dA, *dB, *dC;
    cudaMalloc(&dA, bytes); cudaMalloc(&dB, bytes); cudaMalloc(&dC, bytes);

    dim3 block(32, 32);
    dim3 grid((N + block.x - 1) / block.x, (N + block.y - 1) / block.y);

    cudaMemcpy(dA, hA, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(dB, hB, bytes, cudaMemcpyHostToDevice);

    matmul_naive<<<grid, block>>>(dA, dB, dC, N);

    cudaMemcpy(hC, dC, bytes, cudaMemcpyDeviceToHost);

    cudaFree(dA); cudaFree(dB); cudaFree(dC);
    return 0;
}
```

- Is the GPU under-utilized?
- Is the kernel bound by compute or memory?
- How is the L1/L2 cache utilization?

Is the program spending too much time in memory copying?

Are kernel launch overheads killing the performance?

Nvidia's performance debugging toolbox

Nsight systems (Nsys)

Provides comprehensive application-level performance

Deep dive into *top* CUDA kernels and understand their resource utilization

Nsight compute (Ncu)

Detailed CUDA kernel level performance

Nvidia Tool Extension (NVTX)

Annotating events, code ranges, and resources in your applications

Nsight systems: system level profiler

- **System-wide tracing**
  - One capture spans CPU, GPU, OS runtime, libraries, NVTX—see the *entire* application, not just kernels.
  - Works seamlessly across multiple GPUs, processes and containers.
- **Locate optimisation opportunities**
  - Pinpoint where execution time is spent.
  - Visualise every GPU kernel alongside copies and host work.
  - Spot idle gaps or serialisation to guide overlap and tuning.
- **Fast, low-overhead collection**
  - Generating an nsys report typically takes about the same time as the application run itself.

Collect a profile with Nsight systems

```
$ nvcc -lineinfo -lnvToolsExt
 my_application.cu -o my_application

$ nsys profile -o report --stats=true ./my_application
```

Generated file: report.nsys-rep

```
$ nsys stats report.nsys-rep # Open in CLI
```

Alternatively: we can open the file in nsys-ui

Nsight compute: kernel profiling tool

- **Microscopic kernel insight**
  - Quantify compute-vs-memory utilisation, SM occupancy and warp efficiency.
  - Inspect registers, shared memory, and L1/L2 traffic; expose scheduler stalls.

- **Roofline & heat-map visualisation**
  - Creates a roofline plot plus per-source-line heat-map for every kernel.

- **Deeper detail, higher overhead**
  - Multiple metric passes may be required—profiling takes longer than application execution.

Collect a profile with Nsight compute

```
$ sudo ncu --set full --target-processes application
./my_application
```

Generated: report.ncu-rep

```
$ ncu --import report.ncu-rep --page raw --csv >
raw_report.csv
```

Alternatively: one can open it using ncu-ui

Debugging perf. of a GPU program

```cpp
__global__ void matmul_naive(const float* A, const float* B,
                             float* C, int N)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    if (row < N && col < N) {
        float sum = 0.f;
        for (int k = 0; k < N; ++k)
            sum += A[row * N + k] * B[k * N + col];
        C[row * N + col] = sum;
    }
}
```

- Is the GPU under-utilized?
- Is the kernel compute or memory bound?
- How is the L1/L2 cache utilization?

Nsight Compute

```cpp
int main(int argc, char** argv)
{
    /* Host code and allocations commented out */

    float *dA, *dB, *dC;
    cudaMalloc(&dA, bytes); cudaMalloc(&dB, bytes); cudaMalloc(&dC, bytes);

    dim3 block(32, 32);
    dim3 grid((N + block.x - 1) / block.x, (N + block.y - 1) / block.y);

    cudaMemcpy(dA, hA, bytes, cudaMemcpyHostToDevice);
    cudaMemcpy(dB, hB, bytes, cudaMemcpyHostToDevice);

    matmul_naive<<<grid, block>>>(dA, dB, dC, N);

    cudaMemcpy(hC, dC, bytes, cudaMemcpyDeviceToHost);

    cudaFree(dA); cudaFree(dB); cudaFree(dC);
    return 0;
}
```

Is the program spending too much time in memory copying?

Are kernel launch overheads killing the performance?

Nsight System

Summary

- Nsys is a system wide profiling tool – use it *first* to locate opportunities for performance improvement
- Ncu provides a microscopic view of a kernel - use it to identify the performance improvement opportunities *within* a kernel

Resources:

- Nsys documentation: https://docs.nvidia.com/nsight-systems/UserGuide/index.html
- Ncu documentation: https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html
- Tutorial: https://github.com/csl-iisc/gpu-profiling-tutorial.git