Profiling CUDA Applications with Nvidia Nsight Systems

BY: KEVIN MAHESH KURIAKOSE

What is Nsight?

- From Nvidia: "*[…] a system-wide performance analysis tool designed to visualize an application's algorithms, identify the largest opportunities to optimize, and tune to scale efficiently across [various systems]*"
- Profiler/Tracer for GPU-based applications on Nvidia hardware
  - Graphics: OpenGL, OpenXR, Vulkan, DirectX
  - Video: NVDEC, NVENC
  - Compute: CUDA, OpenACC
  - Communication: MPI, OpenSHMEM, UCX, NCCL
  - CPU: OpenMP, Python, C/C++
- Successor to NVProf

Why Profiling/Tracing?

- Shows where your program is spending its time
  - Often, bottlenecks are in small sections of the program
  - Helps focus performance optimizations
- Tracing gives a timeline of all events
  - Very detailed, lots of data
- Profiling gives you a statistical report from sampled events
  - Useful for long-running programs

What can be profiled? (non-exhaustive)

- Kernel Executions
  - Time Taken
  - Grid Dimensions
  - Register/Shared Memory Usage
  - Occupancy
- Memory Transfers
  - Time Taken
  - Source/Dest. Type
  - Throughput
- Communication
  - MPI/OpenSHMEM/UCX/NCCL API Calls
  - InfiniBand Transfer Metrics
- GPU Hardware Metrics
  - GPU Context Switches
  - GPU I/O
  - Clock Speed
  - Kernels in Flight
  - Power Draw
- OS Metrics
  - CPU Context Switches
  - CPU Instruction Pointer Sampling
  - System Calls

How to profile

- Nsight Systems GUI
  - Can be used for profiling programs on the same machine
  - Can open profiler reports generated by the CLI
- `nsys` CLI utility
  - Useful for servers/clusters with separate GPU nodes
  - `nsys profile [options] <program> <args...>`
  - Generates an `.nsys-rep` file containing results
- More information in the User Guide

1	2	3	4	5	6	7	8	9	...

+

10	20	30	40	50	60	70	80	90	...

Example: Adding two vectors

- Out = A + B (Element-wise)
- A, B are 1GB FP32 vectors ($N = 2^{28}$)
- Starting from Naïve implementation, profile and optimize
- Tested on an Nvidia V100 GPU

Naïve Implementation

- Direct copy of CPU code
- No GPU-specific optimizations done
- Note the kernel launch arguments:
  - Grid size: 1
  - Block size: 1
- Launched as **`nsys profile ./main`**

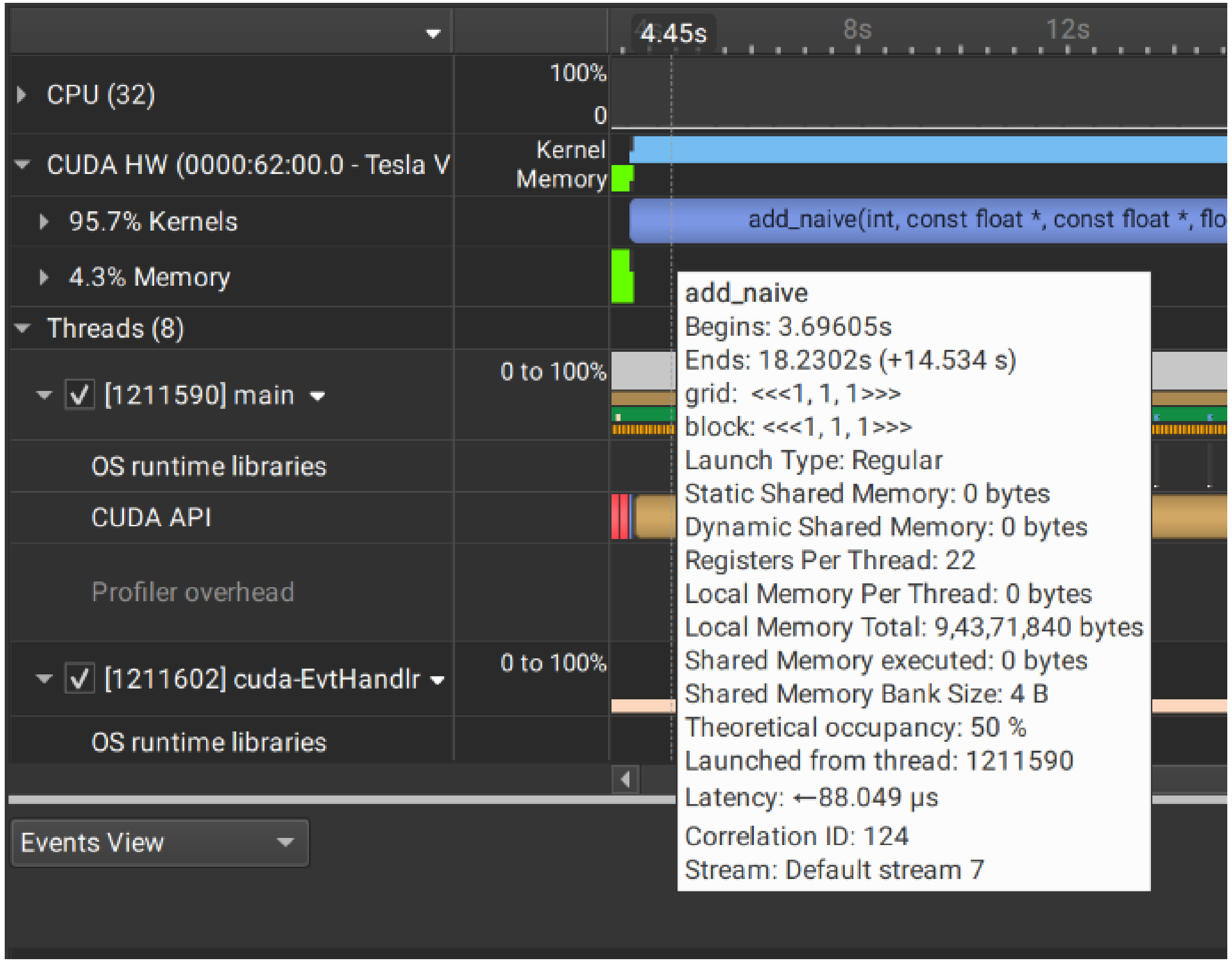
```
__global__ void add_naive(...) {
  for (int i = 0; i < N; i += 1) {
    out[i] = a[i] + b[i];
  }
}

// ...

int main() {
  add_naive<<<1, 1>>>(
    N, dev_a, dev_b, dev_out
  );
}
```

Naïve Implementation

- Overall time: 15.26 seconds
  - GPU Time: 14.53 seconds
- Practically no speedup
- Reason: Only one thread used



One Block Implementation

- Use one block of threads
  - Here, 256
- Per-thread loop now jumps ahead by block size

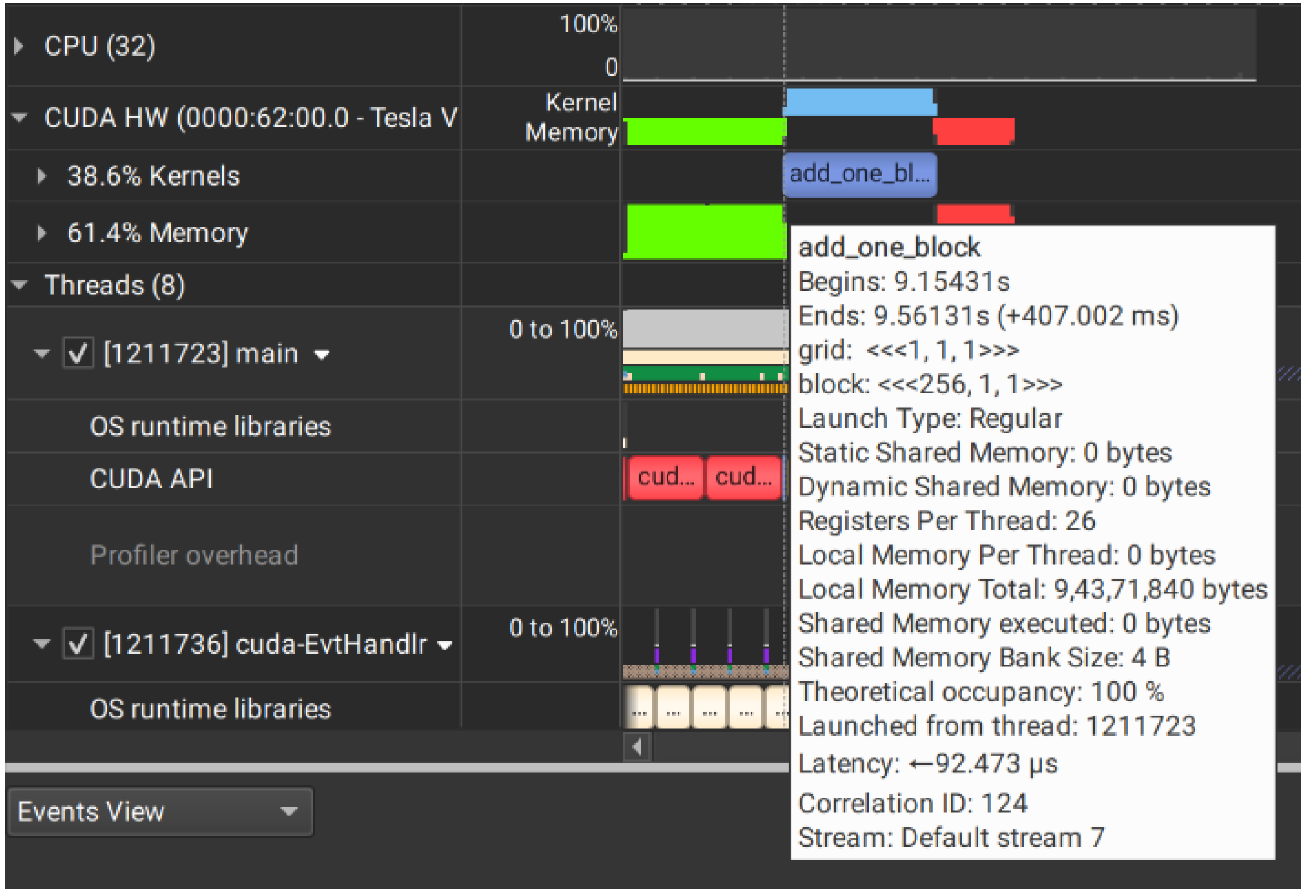
```
__global__ void add_one_block(...) {
  int start = threadIdx.x;
  int stride = blockDim.x;
  for (int i = start; i < N; i += stride) {
    out[i] = a[i] + b[i];
  }
}

// ...

int main() {
  const int BLOCK_SIZE = 256;
  add_one_block<<<1, BLOCK_SIZE>>>(...);
}
```

One Block Implementation

- Overall time: 1.06 seconds
    - GPU Time: 407 milliseconds
- Significant Speedup, but can be improved



Multi Block Grid Implementation

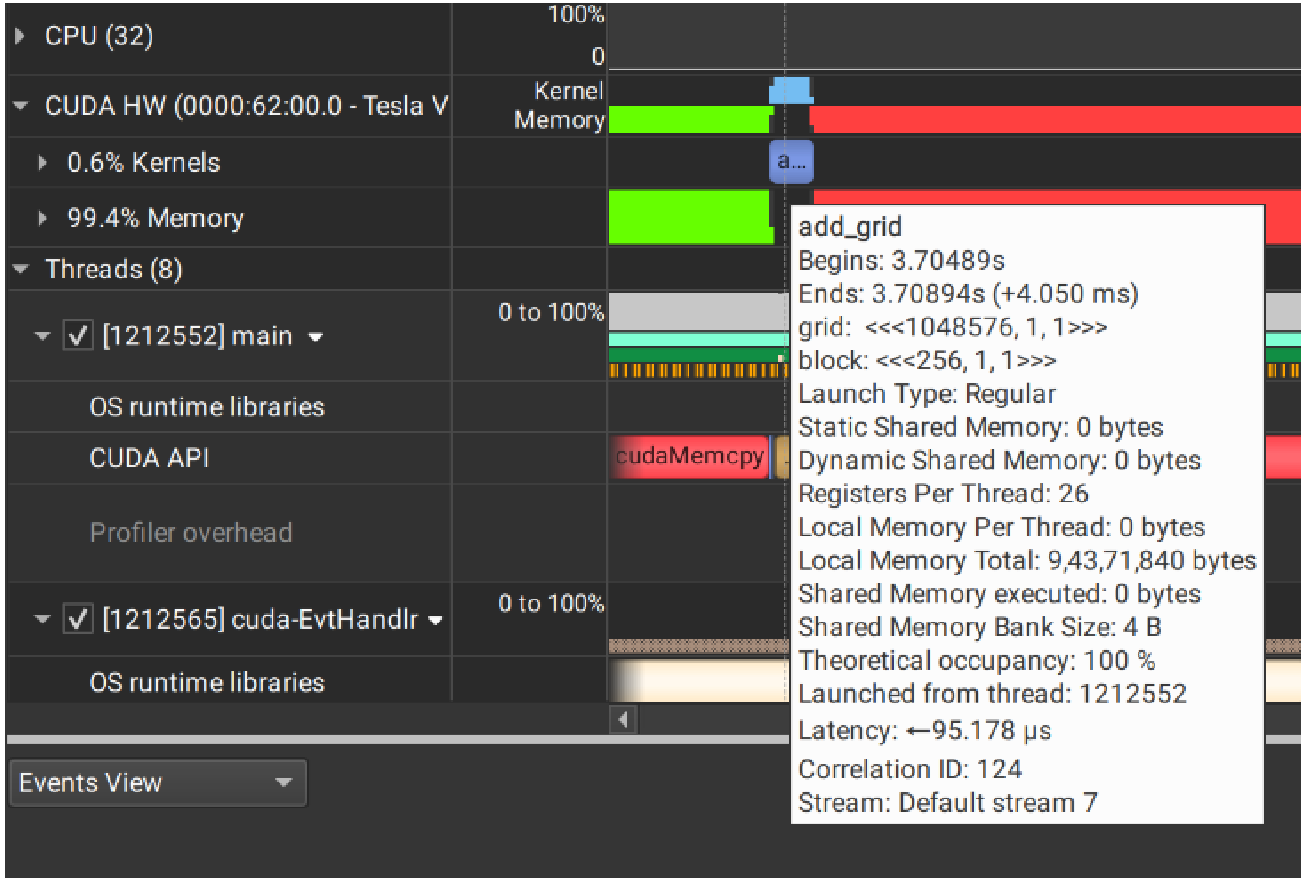
- Use a grid of multiple blocks
  - Launch as many blocks needed to cover vectors
  - Effectively 268,435,456 threads
- Per-thread loop now jumps ahead by grid size

```
__global__ void add_grid(...) {
  int start = (blockDim.x * blockIdx.x)
    + threadIdx.x;
  int stride = gridDim.x *  blockDim.x;
  for (int i = start; i < N; i += stride) {
    out[i] = a[i] + b[i];
  }
}
// ...

int main() {
  const int BLOCK_SIZE = 256;
  add_grid<<<N / BLOCK_SIZE, BLOCK_SIZE>>>(...);
}
```

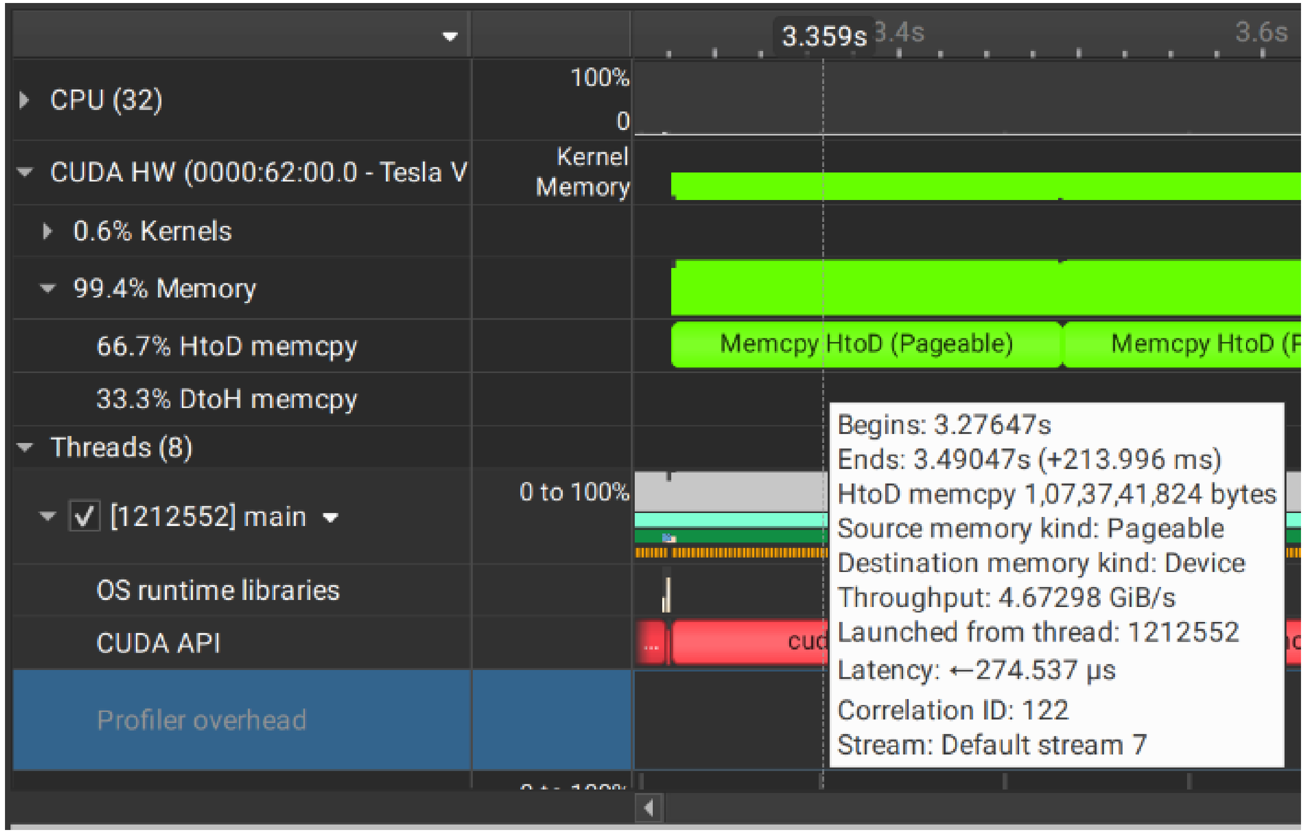
Multi Block Grid Implementation

- Overall time: 661 milliseconds
  - GPU Time: 4 milliseconds
- Compute is practically instant, but data transfers are slow



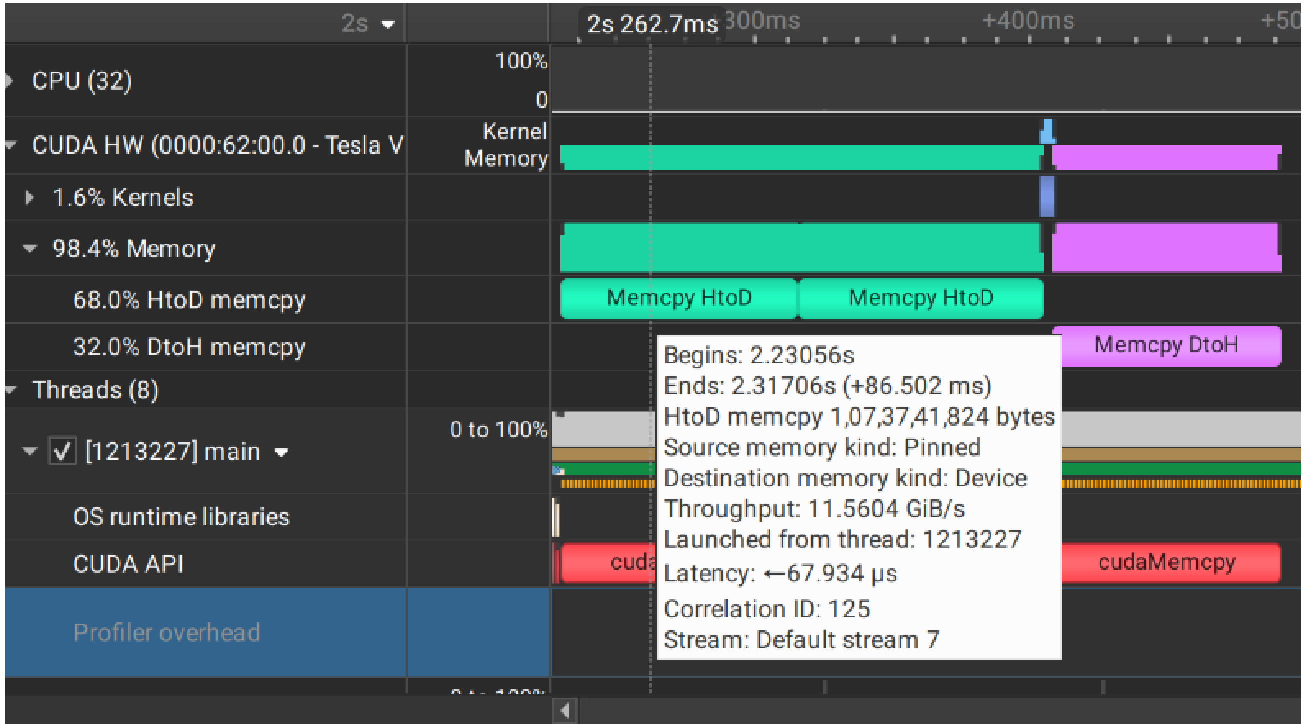
Memcpy Delay

- A, B, Out Copies: ~200 milliseconds each
  - Throughput: ~4 GB/s
  - Vector copies are to/from Pageable memory
- Driver makes an internal copy to ensure data is guaranteed to be in memory during transfer
- Solution: Create host vectors in pinned memory (`cudaMallocHost`)



Pinned Memcpy

- Overall time: 258 milliseconds
  - A, B, Out copies: ~80 milliseconds
  - Throughput: ~11 GB/s
- Many GPU applications are bottlenecked by transfers
- Further optimizations possible (ex. Multi-stream pipelining)



Extras

- `nsys stats <report file>`
    - Prints a summarized report of program statistics
- `nsys analyze <report file>`
    - Provides suggestions for improving performance based on the report

References

- Nvidia Nsight Systems Homepage: https://developer.nvidia.com/nsight-systems
- User Guide: https://docs.nvidia.com/nsight-systems/UserGuide
- Sample Program (Older Profiler): https://developer.nvidia.com/blog/even-easier-introduction-cuda/