Carrera de Ingeniería de Sistemas y Computación
Programación Paralela

Profiling CUDA programs

Mayo 2022

Tek

GPU Vector Addition using blocks and threads

- Typical problems are not friendly multiples of `blockDim.x`
- Data set is larger than the number of threads; use a *grid-stride loop* within the kernel:

```
__global__ void add ( int *a, int *b, int *y, int N )
{
  int indexWithinTheGrid = blockIdx.x * blockDim.x + threadIdx.x;
  int gridStride = gridDim.x * blockDim.x;

  for ( int index = indexWithinTheGrid; index < N; index += gridStride )
    y[index] = a[index] + b[index];
}
```

GPU Vector Addition using blocks and threads

- `nvcc -o single-thread-vector-add 00-vector-add.cu –run`
- `nsys profile --stats=true ./single-thread-vector-add`
- nsys profile will generate a qdrep report file which can be used in a variety of manners. We use the --stats=true flag here to indicate we would like summary statistics printed; you can provide the -f flag to nsys profile to allow overwriting an existing report file. There is quite a lot of information printed:
- Profile configuration details
- Report file(s) generation details
- CUDA API Statistics
- CUDA Kernel Statistics
- CUDA Memory Operation Statistics (time and size)
- OS Runtime API Statistics

GPU Vector Addition using blocks and threads

- Make a simple optimization to 00-vector-add.cu by updating its execution configuration so that it runs on many threads in a single thread block
- `nvcc -o multi-thread-vector-add 01-vector-add.cu –run`
- `nsys profile --stats=true ./multi-thread-vector-add`
- List 3 to 5 different ways you will update the execution configuration, being sure to cover a range of different grid and block size combinations.
- Compile and profile your updated code with the two code execution cells below.
- Record the runtime of the kernel execution, as given in the profiling output.
- Repeat the edit/profile/record cycle for each possible optimization you listed above
- Which of the execution configurations you attempted proved to be the fastest?

GPU Vector Addition using blocks and threads

- `nvcc -o vector-add-no-prefetch vector-add-no-prefetch.cu –run`
- `nsys profile --stats=true -o vector-add-no-prefetch-report ./vector-add-no-prefetch`
- Open NVIDIA Nsight Systems; When prompted, click "Yes" to enable usage reporting. When prompted, select *GPU Rows on Top* and then click *Okay*. Open the Report File
- Open this report file by visiting File -> Open from the Nsight Systems menu and select vector-add-no-prefetch-report.qdrep. You can close and ignore any warnings or errors you see, which are just a result of our particular remote desktop environment.
- Make More Room for the Timelines: To make your experience nicer, full-screen the profiler, close the Project Explorer and hide the Events View.
- Expand the CUDA Unified Memory Timelines: Next, expand the CUDA -> Unified memory and Context timelines, and close the Threads timelines:
- Observe Many Memory Transfers: From a glance you can see that your application is taking about 1 second to run, and that also, during the time when the addVectorsInto kernel is running, that there is a lot of UM memory activity:

GPU Vector Addition using blocks and threads

- Zoom into the memory timelines to see more clearly all the small memory transfers being caused by the on-demand memory page faults. A couple tips: You can zoom in and out at any point of the timeline by holding CTRL while scrolling your mouse/trackpad. You can zoom into any section by click + dragging a rectangle around it, and then selecting Zoom in

GPU Vector Addition using blocks and threads

- `nvcc -o vector-add-prefetch vector-add-prefetch.cu -run`
- `nsys profile --stats=true -o vector-add-prefetch-report ./vector-add-prefetch`
- Open the report in Nsight Systems, leaving the previous report open for comparison.
- How does the execution time compare to that of the addVectorsInto kernel prior to adding asynchronous prefetching?
- Locate cudaMemPrefetchAsync in the CUDA API section of the timeline.
- How have the memory transfers changed?

GPU Vector Addition using blocks and threads

- In the previous iteration of the vector addition application, the vector data is being initialized on the CPU, and therefore needs to be migrated to the GPU before the addVectorsInto kernel can operate on it.
- The next iteration of the application, add-vector-init-kernel.cu, the application has been refactored to initialize the data in parallel on the GPU.
- Since the initialization now takes place on the GPU, prefetching has been done prior to initialization, rather than prior to the vector addition work. Review the source code to identify where these changes have been made.
- After reviewing the changes, compile and run the refactored application using the code execution cell directly below. You should see its success message printed.
- `nvcc -o vector-add-init-kernel vector-add-init-kernel.cu -run`
- `nsys profile --stats=true -o vector-add-init-kernel -report ./ vector-add-init-kernel`

GPU Vector Addition using blocks and threads

- Open the new report file in Nsight Systems and do the following:
- Compare the application and addVectorsInto run times to the previous version of the application, how did they change?
- Look at the Kernels section of the timeline. Which of the two kernels (addVectorsInto and the initialization kernel) is taking up the majority of the time on the GPU?
- Which of the following does your application contain?
- Data Migration (HtoD)
- Data Migration (DtoH)

- Currently, the vector addition application verifies the work of the vector addition kernel on the host. The next refactor of the application, vector-add-prefetch-to-host.cu, asynchronously prefetches the data back to the host for verification.
- After reviewing the changes, compile and run the refactored application using the code execution cell directly below. You should see its success message printed.
- nvcc -o vector-add-prefetch-to-host vector-add-prefetch-to-host.cu -run
- Now create a report file for this version of the application:
- nsys profile --stats=true -o vector-add-prefetch-to-host.report ./ vector-add-prefetch-to-host
- Open this report file in Nsight Systems, and do the following
- Use the Unified Memory section of the timeline to compare and contrast the Data Migration (DtoH) events before and after adding prefetching back to the CPU.

Further Reading

- Kirk, D. B., Hwu W. W. Programming Massively Parallel Processors: A Hands-on Approach, Third Edition. Morgan Kaufmann, 2016
- Han, J.; Sharma, B. Learn CUDA Programming. Packt Publishing, 2019.

- NVIDIA Corporation. NVIDIA CUDA C Programming Guide, version 8.0. NVIDIA Corporation, 2017
- NVIDIA Corporation. NVIDIA CUDA C Programming Guide, version 11.4.2. NVIDIA Corporation, 2021

Acknowledgements

Some slides were adapted from NVIDIA material