Sai Jayanth Kalisi
Van Tha Bik Lian

Autumn 2024
EE P 524: Kernel Tuning Contest
Final Project Report

INTRODUCTION

In this project, we explore three patterns in parallel computing: the 3D stencils, histograms, and reductions.

Stencil patterns are a combination of maps with a local gather over a fixed set of offsets. Every output of a stencil is a function of some neighborhood of elements in the input collection. Its local neighborhood structure exposes opportunities for data usage and optimization of data locality. Stencils are extremely common in image processing and scientific simulations. They are often 2D or 3D (can be extended to higher dimensions), and computationally, they feature high memory traffic, and low arithmetic intensity, and are often memory-bound when computation uses all global memory accesses.

Histograms display the number count/percentage of occurrences of data values in a dataset and can be used to extract notable features/patterns from large datasets. Histograms can be used to summarize key data distribution characteristics, extract features for object recognition, fraud detection, and so on. Computing each count or bin is an operation that can be done in parallel.

Reduction of an array is useful for summarizing all values of that array into a single value and for example, finding the mean, max, min, product, median, etc. It is beneficial for statistical analysis of large swaths of data, in machine and deep learning, physics and engineering simulations, rendering of images, and massively parallel algorithms. In particular, the reduction we intend to implement is that of an aggregate sum across all values in the array provided

In this project, we look at several kernels for each kernel and iteratively fine-tune each kernel to get the best performance. We present our best-performing kernels per pattern.

TECHNICAL OVERVIEW

Kernel tuning refers to the process of optimization of GPU kernel-level functions to gain better performance. For our hardware, we had a few limitations for example maximum block size of 1024, or the maximum device memory achievable. These limitations were important to try to work around.

Kernel tuning includes the modification of block and grid size optimizations. These are usually done to maximize the occupancy. Other ways include memory optimization, to reduce or increase dependency on shared memory, device memory, and thread-level memory. Doing this could lead to speedups. In this project, we looked at device/global memory and shared memory for the stencil category of kernel in particular.

Effective tuning often requires repeated iterations of modifications and testing. Because of this, there is a time commitment. However, it is a fruitful way of finding out application-specific modifications that could be done to the kernel, to get the most out of the hardware.

WORK DISTRIBUTION

Work was distributed quite evenly. A significant portion of the project's debugging and design/creativity occurred in group meetings.

In terms of general analysis, stencils were scrutinized by Van, while Sai Jayanth looked at Reduction. Both members analyzed histograms together. Both members double-checked the analysis of the other.

Both members contributed to the generation of data, and population of results in the tables provided.

Finally, this report was written simultaneously in a group working session where both members were present.

Methods

Independent Variables

Independent values modified for each kernel varied. These included:

1) Block and Tile Size
    a) Grid Size
2) Coarsening Factor
3) Shared Memory size
4) Shared Memory vs Device Memory

Block and Tile size was modified for each of the kernel types. The coarsening factor was modified for each kernel that implemented thread coarsening.

All Kernels had a test case for SMEM modification. For Stencil and Reduction, this was the modification of tile size. Since there are no “tiles” available in reduction, the SMEM changed to match the block size for the segmented_multiblock kernel. For Histograms, this meant changing the bin size.

Dependent Variables

The following variables were determined as important to track.

Timing:
Duration Nsight, Duration Total Host ms, Duration CPU Host ms, Duration Kernal Host ms

Compute Workload analysis:
avg SM act cycl, avg SMSP act cyc, avg L1 Active Cycles, avg L2 Active Cycles, DRAM Act Cyc, Elapsed cycles

Speed of Light:
Compute throughput, SOL Mem throughput, SOL L1/TEX throughput, SOL L2 throughput

Occupancy:
Theoretical Occupancy, Theoretical AW/SM, Achieved Occupancy, Achieved AW/SM

Memory Workload Analysis:
Kernel-Global Mem Transfer, L1->L2 Transfer, L2->L1 Transfer, Total SMEM Wavefronts, Total L1 Requests, L1 Loads (Instructions), L2 Sectors/Req, Hitrate, Bank conflicts

Setup

The Independent variables were modified, one at a time. Each time, a new exe was created. Each exe file was run through NSight Compute and NSight Systems, generating reports for each. Reports from the NSight Compute were scrutinized further and the dependent variables were recorded.

Results

Tables

The link below includes comprehensive results for each combination of independent variables. There are multiple tabs in the spreadsheet link. One details the GPU configurations, and another details CPU times. Another tab includes the best-performing metric for each Kernel type for each category. A final tab includes the best performing kernel for each category/pattern.
https://docs.google.com/spreadsheets/d/1nPIY7L3BHG6XiCasgFNKrFSTcV0i-WUeaGxuU5gesas/edit?usp=sharing

Note that some columns have been hidden to highlight the main dependent variables of interest. Though some cases exist when the hidden metrics are valuable, we aim to improve throughput and latency for the Kernel Tuning Challenge, so the event timing metrics matter more.

BEST PERFORMING KERNELS

3D-STENCIL

Best kernel: Kernel implementing with thread coarsening (CF is 14). Stencil coefficients are moved to shared memory. Launch config - 16x16x1

In our exploration of the 3D stencil, we looked at a total of 18 kernels and identified the best kernel based on the following metrics: NSIGHT duration, kernel host duration, average SM act cycles, compute & memory throughput, achieved occupancy, and memory transfer across different memory stages.

Throughout our iterative process of identifying the best kernel for the 3D-STENCIL pattern, we used a 3D grid of 512x512x512 filled with assorted floats as our dataset. Our kernels were implemented in various ways; global naive where the kernel utilizes global memory access for computation, tiled SMEM where data is loaded square tile by square tile into shared memory by assigned threads, non-squared tiled (similar to the previous, but tiles are allowed to be non-squares, and several implementations with thread-coarsening where a single thread can be used to load multiple data points.

In addition, we realized that in each kernel a convolution for the 3D stencil requires the reuse of the stencil coefficients. So instead of keeping the coefficients on global memory, we also implemented each kernel where the stencil kernel is moved to shared memory. We iteratively changed the block sizes in each kernel launch and recorded statistics for the mentioned metrics for each iteration. While determining which kernel is best, any two kernels that are fairly similar across these metrics, we put more weight on the NSight and Kernel host duration to narrow down.

In general, we found that the duration, and compute/memory throughput vary across each kernel. We also observed a noticeably higher occupancy in kernels with thread coarsening which makes sense because a single thread can handle more than one computational element. Between the kernels with thread coarsening, the configuration with block size 16x16x1, corresponding to a coarsening factor of 14, had the highest stats across our measured metrics. For this configuration, we also have implementations with the stencil coefficient in global memory and shared memory.

One further optimization for this includes thread-level memory, where we could directly hardcode the stencil coefficients in the kernel. Doing this would eliminate the need for data transfer between L1 and L2 caches. There were also 0 observed bank conflicts across all kernel types.

The metrics are relatively close between GMEM and SMEM implementations. We chose to submit the implementation with coefficients loaded into shared memory based on the NSIGHT duration.

HISTOGRAM

Best kernel: Coarsened-contiguous, a block size of 1024, a coarsening factor of 8.

Our exploration of the Histogram pattern is similar to that of the 3D stencils. We looked at 20 kernel configurations and used the same iterative process as the 3D stencils to identify our best kernel. For each kernel, we used the same dataset (1 million random alphabets). The kernel type varied between global-naive, Private SMEM, Coarsened-Contiguous and Coarsened-Interleaved. Global Naive represents the naive implementation of a histogram, where each thread calls an atomic add, adding to the histogram memory. Private SMEM includes privatization and a block-wise SMEM that is then used as a template for addition. Coarsened Contiguous and Coarsened Interleaved are different implementations of thread coarsening for Histograms

In this category, we modified thread-block dimensions, grid dimensions, bin size, and the corresponding number of bins. This afforded us better flexibility in SMEM modifications.

Across our 20 kernels, we observed fairly low compute and memory throughputs. Due to time constraints, we kept to our 20 kernels and focused on the achieved occupancy and NSIGHT duration metrics to identify our best. One observation in our data is the large number of bank conflicts. Having a series of cascading additions, as done in histograms, raises the proper conditions for bank conflicts - all threads are attempting to add a series of N bins. This causes bank conflicts via atomicAdd in kernel implementations with SMEM being a large part.

The kernel we are submitting is our best. However, we acknowledge that there are several clear avenues we see to improve on this. Based on NSIGHT-Compute, we can achieve an 86.89% speed up in the L1 global load access, 70.14 % for uncoalesced shared access, and 63.44% for uncoalesced global access. Future iterations of this project should use this as a guide on how to proceed.

One improvement could include looking at only a subset of bins per pass. For example, we could run this on bins corresponding to A through M, while the second run could correspond to bins N through Z. Though this would take multiple runs, there is a chance that different passes for a single block can run synchronously. This would further streamline the histogram process.

REDUCTION

Best kernel: multiblock-coarsened kernel with a block size of 256

Once again, our exploration for the Reduction Sum is much similar to the 3D stencil pattern. For this pattern/category, we looked at 12 kernels and accessed them using data containing 32 million floats of all ones. These were split into Poor reduction, Improved reduction, segmented multiblock, and coarsened multiblock implementations.

Our primary modifications in this category/pattern include grid size, block size, and coarsening factor modification. Unlike the other patterns mentioned above, this did not involve modification of SMEM or GMEM. There were also 0 observed bank conflicts across all kernel types.

Across our 12 kernels, we found that the multi-block kernels with thread coarsening had the highest achieved occupancy, significantly lower NSIGHT duration, and memory throughput. The kernel we have submitted is not the absolute best in both NSIGHT and host kernel duration, but it is very close to the best. We chose this because the compute throughput is almost twice that of the next-best coarsened implementation.

Better loop unrolling and warp-level manipulation could effectively reduce values within the warp. This could avoid divergence and reduce the time taken to run on the GPU.

Appendix

Code and Datasets:

For further viewing of our code and datasets, please take a look at https://drive.google.com/drive/folders/1jLAVyduYBRk3cvbX2uSvfy8aQQoVpCtg?usp=drive_link

This link includes NSight Compute Reports of all test and best case scenarios, NSight Systems Reports of all best case scenarios, code that was used to generate the datasets along with the datasets themselves, and finally the code for configurations and kernels, which was derived heavily from the Kernel Tuning Challenge starter code given to us.