Parallelization Strategies for GPU Accelerated Data Sampling

CLEMSON UNIVERSITY

Cooper Sanders and Jon Calhoun (Advisor)
Holcombe Department of Electrical & Computer Engineering – Clemson University
cssande@clemson.edu, jonccal@clemson.edu

Introduction

Compute hardware is outpacing I/O. To run large scientific workflows, data reduction is necessary to reduce the load on I/O hardware.

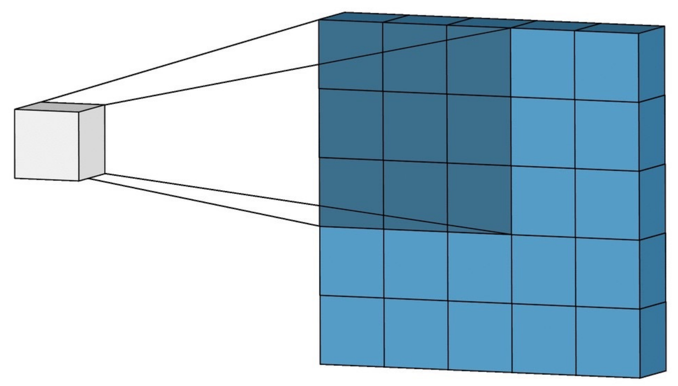
Sampling is a new approach to data reduction, but it doesn't have much availability on accelerating hardware like GPUs. To be practical, data reduction must be fast – accomplished with GPU implementations.

Gradient Sampling

Gradient based sampling is a new sampling approach that prioritizes data points with a higher gradient [1]. Samples near rapidly changing regions of interest are saved.

Gradient sampling has two main bottlenecks: histogramming and gradient computation. Both can be sped up using parallel processors. In each case, the algorithm can take advantage of shared memory when it runs on a GPU, at the cost of introducing more warp divergence. The optimal strategy depends on which factor has more weight on performance. This study aims to implement gradient sampling in CUDA by optimizing both steps.

Similar to convolution, each output element in the discrete gradient depends on surrounding input elements [3].



Contributions

- Analysis of parallel programming strategies for histogram computation
- Design and analysis of gradient computation in CUDA
- GPU implementation of full gradient sampling algorithm

Experimental Design

Several parallelization strategies at each stage of the algorithm were implemented in Python, C, & CUDA. They were benchmarked on synthetic data from a normal distribution using high-end hardware:

- 56 core Xeon Gold CPU
- Nvidia A100 GPU

Acknowledgements

Clemson University is acknowledged for generous allotment of compute time on the Palmetto cluster. This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197 and SHF-1943114.

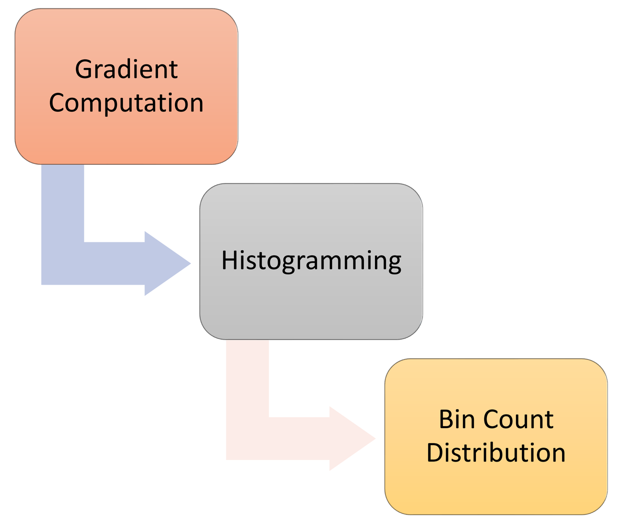
References

[1] Ayan Biswas, Soumya Dutta, Earl Lawrence, John Patchett, Jon C. Calhoun, and James Ahrens. 2021. Probabilistic Data-Driven Sampling via Multi-Criteria Importance Analysis. IEEE Transactions on Visualization And Computer Graphics 27, 12 (2021), 4439–4454.
[2] Megan Fulp, Ayan Biswas, and Jon Calhoun. 2020. Combining Spatial and Temporal Properties for Improvements in Data Reduction. (2020).
[3] https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Gradient Sampling Pseudocode [1]

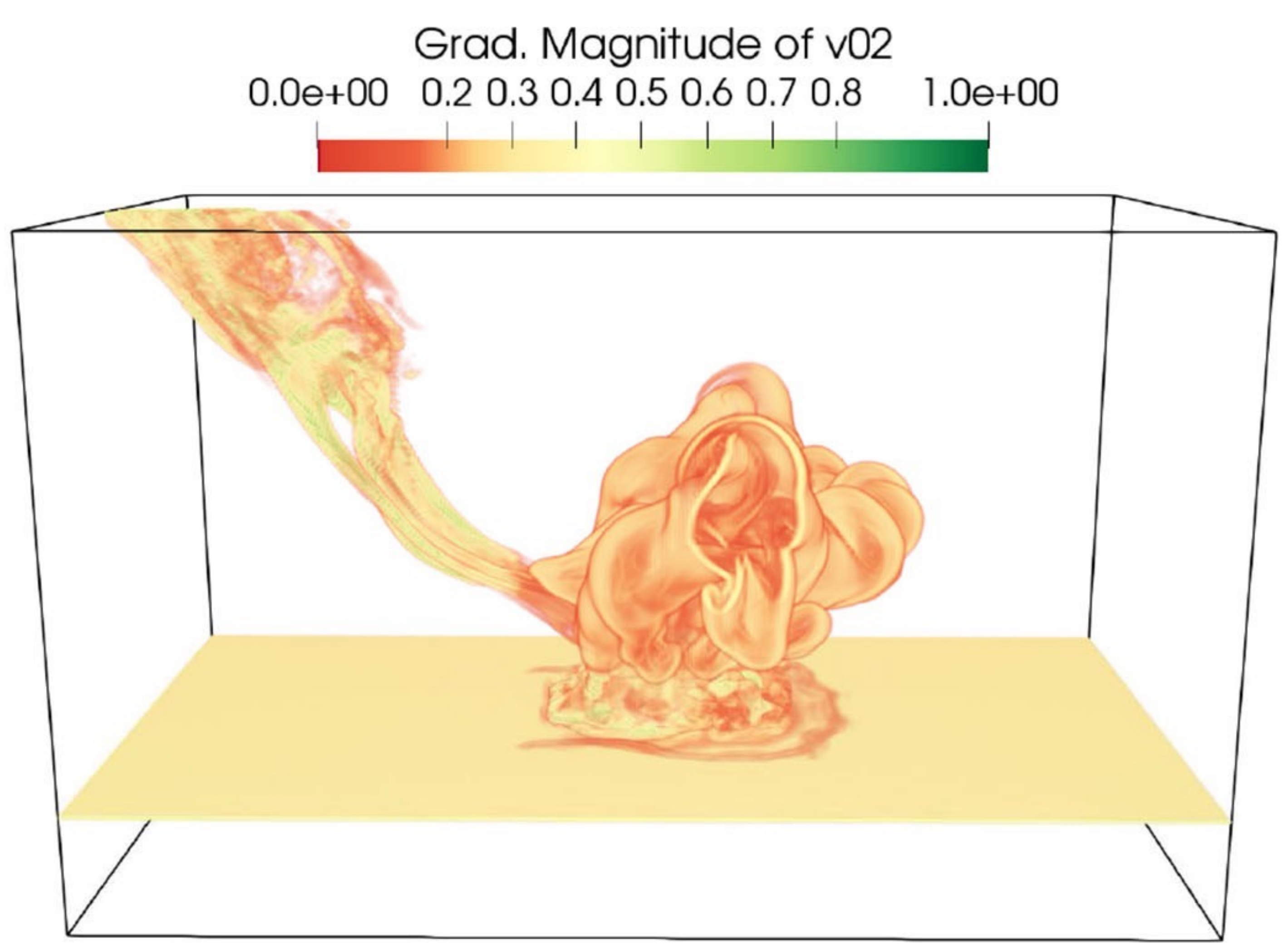
Input: D (data), N (number of data points), M (number of samples), B (number of bins)
Result: I_F (importance function/histogram for selecting M samples from N data points)

$G \leftarrow$ computeGradient(D);
$G_{mag} \leftarrow$ computeGradientMagnitude(G);
$H \leftarrow$ histogram(G_{mag}, N, B);
$I_F \leftarrow$ zeros(B);
$C \leftarrow M/B$; // Expected number of samples
$j = B - 1$;
while $j >= 0$ **and** $M > 0$ **do**
  $c_j \leftarrow H[j]$; // Count in bin j
  $I_F[j] = c_j$;
  $M = M - c_j$;
  $j = j - 1$;
end
/* Normalize by histogram count */
for $j \leftarrow 0$ **to** B **by** 1 **do**
  $I_F[j] \leftarrow I_F[j]/H[j]$;
end

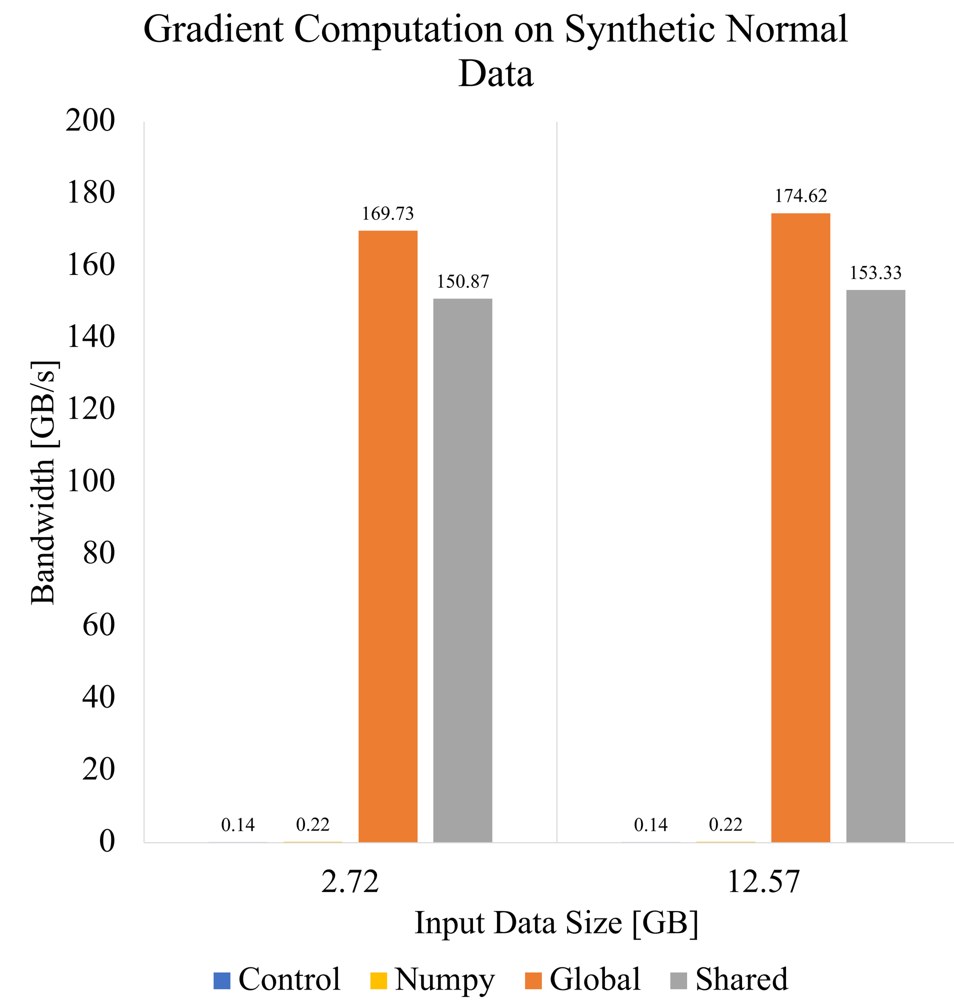


Applications

Below is a visualization of rapidly changing regions in an asteroid simulation [1]. Highlighted areas are prioritized in gradient-based data sampling.



Results and Discussion

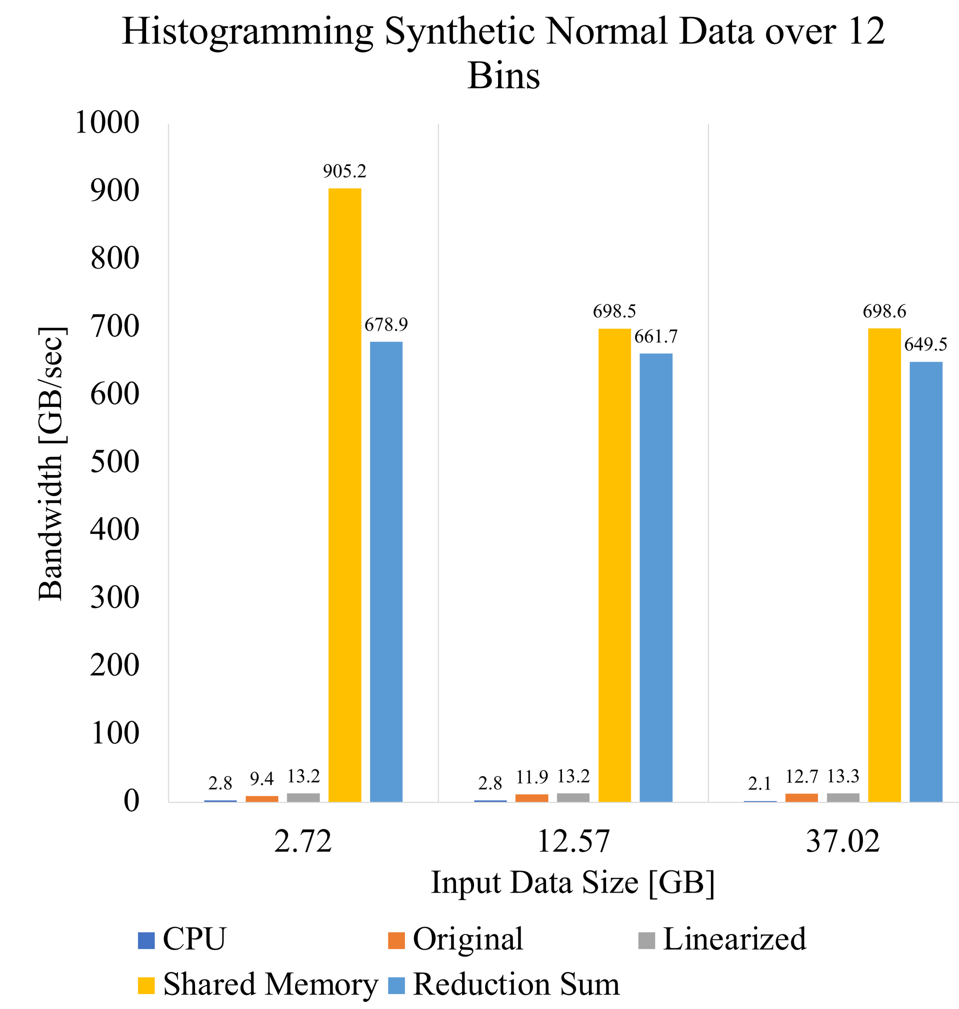


Further Discussion of Results:

Gradient Computation Performance

- Numpy: 1.6x speedup
- Global Memory Approach: 1200x speedup
- Shared Memory Approach: 1100x speedup

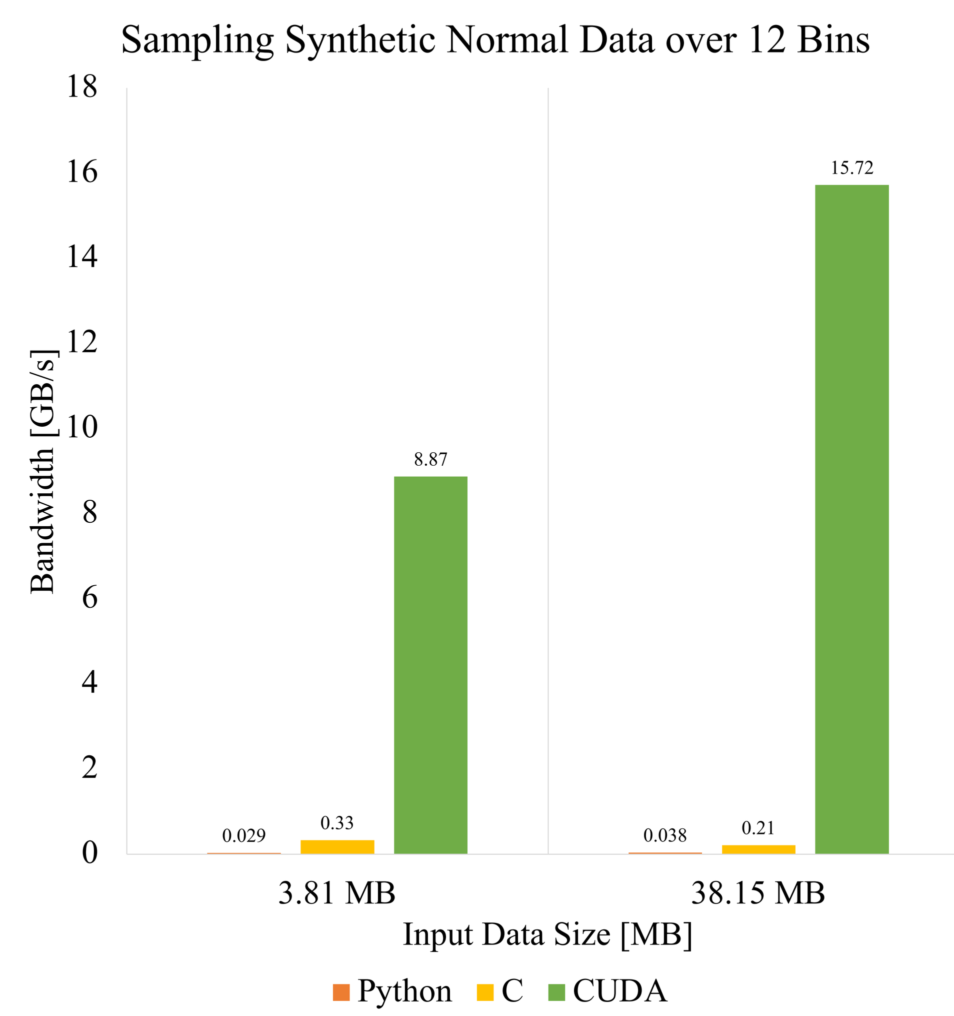
Comparing to a sequential CPU implementation in C



Histogram Computation Performance

- Original Approach [2]: 4.6x speedup
- Linearized Approach: 5.3x speedup
- Shared Memory Approach: 300x speedup
- Parallel Reduction Sum Approach: 260x speedup

Comparing to a sequential CPU implementation in C



Gradient Sampling Performance

- C: 8.5x speedup
- CUDA: 360x speedup

Comparing to the original Python implementation

Conclusions from Results

- Gradient Sampling is highly parallelizable and runs significantly faster on GPUs.
- Discrete Gradient Computation is most efficiently implemented on GPUs *without* using shared memory.
- In general, there is a tradeoff with shared memory between reduced global accesses and increased warp divergence. Complex algorithms with minimal data locality, like gradient computation, are slowed by shared memory.

Future Work

Gradient Sampling in CUDA is limited because not all of it is on the GPU yet – only the major parts of the algorithm covered in this poster.

Additionally, more sampling algorithms [1] use the stages covered in this poster:

- Joint Multi-Criteria Sampling
- Combined Independent Sampling