Efficient GPU Techniques for Processing Temporally Correlated Satellite Image Data

Tahsin Reza
Electrical and Computer Engineering
University of British Columbia
Vancouver, BC
treza@ece.ubc.ca

Dipayan Mukherjee
Computer Science and Engineering
Indian Institute of Technology
Kharagpur, India
dipayan1992@gmail.com

Tanuj Kr Aasawat, Matei Ripeanu
Electrical and Computer Engineering
University of British Columbia
Vancouver, BC
{taasawat, matei}@ece.ubc.ca

ABSTRACT
Spatio-temporal processing has many usages in different scientific domains, e.g., geostatistical processing, video processing and signal processing. Spatio-temporal processing typically operates on massive volume multi-dimensional data that make cache-efficient processing challenging. In this paper, we present highlights of our ongoing work on efficient parallel processing of spatio-temporal data on massively parallel many-core platforms, GPUs. Our example application solves a unique problem within Interferometric Synthetic Aperture Radar (InSAR) processing pipeline. The goal is selecting objects that appear stable across a set of satellite images taken over time. We present three GPU approaches that differ in terms of thread mapping, parallel efficiency and memory access patterns. We conduct roofline analysis [4] to understand how the most time consuming GPU kernel can be improved. Through detailed benchmarking using hardware counters, we gain insights into runtime performance of the GPU techniques and discuss their tradeoffs.

Keywords
Remote sensing, InSAR, Image processing, Scientific computing, GPU acceleration.

1. SUMMARY
Spatio-temporal data analysis has various application in many scientific domains, such as, geostatistical, video and signal processing. One of the common objectives of spatio-temporal processing is pattern analysis –understanding how data is distributed and evolves over time. Spatio-temporal processing considers data along multiple dimensions, e.g., information about the same artifact at the same longitude-latitude at different points in time. From computational perspective, spatio-temporal processing typically operates on massive volumes of multi-dimensional data which makes efficient processing challenging: commonly used data structures, e.g., 3D arrays, incurs scattered data accesses. Also, difficult to prevent branch divergence and ensure memory coalescing on GPUs.

In this paper, we present highlights of our ongoing work on efficient processing of spatio-temporal data on massively parallel many-core platforms, GPUs. Our example application solves a unique problem within the InSAR processing pipeline. InSAR is an earth-orbiting satellite-based remote sensing technology used primarily for estimating displacement of the earth's surface [1]. InSAR has applications in monitoring earth subsidence and uplift due to urban infrastructure development, mining, oil and gas extraction, permafrost thawing etc.

The spatio-temporal processing problem considered in this paper is formally known as Persistent Scatterer (PS) Pixel Selection [2]. A branch of InSAR algorithms considers artifacts in radar images that are permanent over time. Pixels in a 2D radar image corresponding to permanent objects are called Persistent Scatterer. Efficiency and accuracy of several algorithms within the processing pipeline depend on successful identification of permanent objects. The objective of the PS selection algorithm is to assign each pixel a rank (a number in the range (0.0, 1.0]) which indicates how '*stable'* it is in a series of images taken over time.

In this work, we consider the *PtSel* PS selection algorithm discussed in [3]. Temporal coherence computation is the most curtail and time consuming operation in PtSel. We have identified two parallelization approaches for temporal coherence computation: *window-parallel* (WP) and *arc-parallel* (AP). In WP, each pixel in the *patch* (image segment) is mapped to a unique GPU thread. Parallelism is achieved across pixels as each thread computes temporal coherences of all the arcs within a search window. In the finer-grain AP, each *arc* (formed by a pixel and one of its neighbours) is independently mapped to a GPU thread and threads perform temporal coherence computations in parallel. When the GPU device memory is small or the depth of the interferogram network is large, the size of a patch ends up being relatively small (e.g., 100×100 pixels). In this case, the key advantage of AP is it can offer better parallel efficiency as it exploits more parallel threads and better utilizes the GPU. The AP approach discussed in [3] however, does not show advantage over WP when the patch size is large, e.g., 500x500 pixels. In this paper, we address this limitation of AP and focus on further improving the technique.

In CUDA, requests to the device memory from 32 threads in a warp can be grouped into a single memory transaction as long as the requested memory addresses perfectly align. Reducing number of memory transactions means lowering

the chance of stalls due to threads waiting for data. The first AP approach discussed in this work, leverages this feature of GPUs and exploits structure of arrays (SOA) to store the image data. However, it introduces data redundancy. Although achieves improved memory coalescing, unfortunately, due to data redundancy, AP substantially increases traffic to the slower device memory and does not offer speedup over WP.

Roofline analysis [4] revealed that one way to improve performance of the kernel in question is to improve data locality, i.e., on-die caching. To this end, we developed an improved version of the arc-parallel approach dubbed AP-B. AP-B maps arcs to threads but does not reorganize the image data to a SOA to improve memory coalescing. In order to improve locality, arcs are grouped so that they form a graph (K_9 complete graph minus four edges) consisting of 32 edges. These 32 arcs are mapped to 32 consecutive threads in a warp. The objective is to reduce stalls at the warp level due to data requests to the device memory through improving cache hit rate.

Evaluation using a Nvidia Tesla K40 GPU shows that AP-B improves kernel execution time over WP by 2.4x. Performance gain is attributed to 41% improved cache hit rate and 200GFLOP/s more FLOP-rate. Performance counter output analysis shows AP-B improves parallel efficiency as it is able to schedule more warps per cycle, and reduces memory requests related stalls and data traffic to device memory.

In the future, we want to utilize shared memory to further improve locality and eliminate effects of strided accesses within a warp. We also want to extend the AP-B thread mapping approach from the warp level to the block or SMX level on the GPU.

2. REFERENCES

[1] P. Rosen, S. Hensley, I. Joughin, F. K. Li, S. Madsen, E. Rodriguez, and R. M. Goldstein, “Synthetic aperture radar interferometry,” Proc. of the IEEE vol. 88, no. 3, pp. 333-382, March 2000.

[2] A. Hooper, “Persistent scatterer radar interferometry for crustal deformation studies and modeling of volcanic deformation,” Ph.D. Thesis, May 2006, Retrieved from http://web.stanford.edu/group/radar/people/Hooper_Thesis.pdf, on 16 December 2014.

[3] T. Reza, A. Zimmer, P. Ghuman, and M. Ripeanu, “Accelerating Persistent Scatterer Pixel Selection for InSAR Processing,” In Proc. of the 26th IEEE International Conference on Application-specific Systems, Architectures and Processors, Toronto, Ontario, 27 - 29 July 2015.

[4] S. Williams, A. Waterman, and D. Patterson. “Roofline: an insightful visual performance model for multicore architectures”. Commun. ACM, vol. 52, pp. 65–76, 2009.