Enhancing a Cross-Platform Hyperspectral Image Analysis Library for Heterogeneous CUDA Support

Brian A. Landrón-Rivera, Jonathan Torres-Conty, Nayda G. Santiago
Electrical and Computer Engineering Department, University of Puerto Rico, Mayagüez, PR 00681

Abstract—Modern hyperspectral image (HSI) analysis makes use of the high performance computing power provided by General Purpose Graphical Processing Units (GPGPUs) due to the large volume of hyperspectral sensor data involved. To aid rapid prototyping of HSI analysis platforms that make use of GPGPUs, an open source software library, *libdect*, supported by the NVIDIA Compute Unified Device Architecture (CUDA) is being improved. The *libdect* library includes an implementation of the Reed-Xiaoli (RX) and Matched Filter (MF) target detection algorithms and its infrastructure is supported by the CMake build system, incorporating cross-platform compatibility into *libdect*. Coding guidelines and a build log was integrated into CMake using KWStyle, CTest, and CDash. This paper addresses the partition of large HSIs for detectors to analyze using multiple kernel launches. The presented solution discusses how to select smaller workloads that a CUDA device can handle while ensuring coalesced memory transactions using a specific data format.

Keywords—GPU, high performance, software library, hyperspectral, build system, software engineering

1 Introduction

HSI analysis applications range in purpose from military reconnaissance devices to devices that aid medical diagnostics [1]. These applications require a substantial amount of learning and development to be conducted in order to correctly implement GPGPU based algorithms. In an attempt to significantly reduce prototyping time, the *libdect* library was developed [2]. This library is intended to be open source software to avoid complications associated with license restrictions that limit software use and availability [3]. The *libdect* library provides an encapsulation of detection algorithm implementations supported by the NVIDIA CUDA framework that future developers need not be concerned about while balancing performance.

Further author information:
Brian A. Landrón-Rivera: E-mail: brian.landron@upr.edu
Jonathan Torres-Conty: E-mail: jonathan.torres@upr.edu
Nayda G. Santiago: E-mail: nayda.santiago@ece.uprm.edu

2 Libdect Design

The *libdect* library supports a GPU and CPU version of the RX and MF target detectors, which were originally implemented by Blas Trigueros [1]. A set of coding style rules has been developed and can be tested for automatically with KWStyle, which has been incorporated into *libdect's* CMake infrastructure. Support for CTest (unit test driver) and CDash (distributed testing result website) has been added to CMake as well.

3 Workload Partition

The problems solved and described in this work are centered on enhancing *libdect* to include support for processing large HSIs on CUDA GPUs. The implemented solution allows *libdect* to process an entire HSI even if it cannot be stored on a CUDA capable device's global memory as a single workload. In addition the previously used HSI data format, band sequential (BSQ), ensured coalesced memory transactions by each warp [1], but partitioning HSIs in this format results in having CUDA kernels compute a detector's output with an incomplete dataset. A potential solution to this issue was developed where large datasets are partitioned into smaller workloads that a CUDA capable device can handle. This includes changing the supported HSI format from BSQ to band interleaved per line (BIL), which supports coalesced memory transactions [1].

For each kernel to process a HSI partition, *libdect* determines the largest amount of lines in a HSI that can be processed by the selected device, the amount of kernels needed to process the complete dataset, and the remaining amount of lines that occupy less than the selected device's available global memory. The amount of kernels needed is determined by dividing the size of the HSI by the available global memory size. The amount of lines that can be processed by the selected device is calculated by dividing available global memory size by the size of each line. These parameters (kernel count and lines per kernel) are used in the kernel configuration for any amount of lines that fit the GPU global memory in a loop until the complete dataset has been processed. In addition *libdect* uses the lines per kernel parameter as an offset when copying a HSI partition from host to device and each detector's partial output from device to host. A CUDA block size of 512 threads that ensures maximum occupancy on devices with compute capability 1.2 and above was determined using the CUDA Occupancy Calculator [4]. Block x and y dimensions, 32 and 16 respectively, have remained as in [1].

4 Results

The *libdect* library is emerging from its prototype phase. The partition of HSIs required to compute the MF detector output for HSIs that occupy more than the available global memory in a CUDA capable device has been completed. This will ensure the complete analysis of HSIs stored in BIL format of any dimension, with coalesced memory transactions. The partition approach has only been tested with the MF detector and we are currently working on testing the RX detector. This solution has been tested with synthetic HSIs of 0.5GB, 2GB, 4GB, and 8GB using a GeForce GTX 480, Tesla C1060, and a Tesla C2075. Spectral signatures were obtained from the Laboratory for Applied Remote Sensing and Image Processing (LARSIP) at the University of Puerto Rico, Mayagez Campus and were mixed using the linear mixing model [5] to generate HSI backgrounds and insert target signatures. This approach resulted in performance gains in all mentioned CUDA devices for datasets larger than 2GB. Average speedus for the 2GB, 4GB, and 8GB datasets were calculated using measurements from all three CUDA devices. In comparison to the non-partitioning CPU version the results reveal an average speedup of 0.089s, 1.03s, and 1.8s in the GPGPU analysis of the 2GB, 4GB, and 8GB datasets respectively.

5 Conclusions

Migration from BSQ to BIL HSI format is complete and ensures coalesced memory transactions. An algorithm that partitions the input and determines the largest amount of lines a selected device can handle is in effect. It also has the capability to determine the amount of kernels needed to analyze a complete dataset. Each kernel is launched with 512 threads per block to ensure maximum occupancy on devices of compute capability 1.2 and above. The MF algorithm has been tested with synthetic HSIs of various sizes.

References

[1] B. Trigueros, "GPU based implementation of target detection algorithms for hyperspectral images using NVIDIA CUDA," Master's thesis, University of Puerto Rico, Mayagüez, 2011.

[2] G. J. Pérez-Irizarry, F. De La Cruz-Sánchez, B. A. Landrón-Rivera, N. G. Santiago, and M. Vélez-Reyes, "Developing a portable gpu library for hyperspectral image processing," *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery*, vol. Proc. SPIE 8390, no. XVIII, pp. 839 017–839 017–11, 2012.

[3] I. H. Donner, "Don't judge a software license by its cover," *Computer*, vol. 29, no. 10, pp. 114–115, Oct. 1996.

[4] NVIDIA Corporation. (2013, Jul.) CUDA Occupancy Calculator. Available: http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls

[5] R. Schowengerdt, *Remote Sensing: Models and Methods for Image Processing*. Elsevier Science, 2006.