Proceedings of the 18th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2018
July 9–14, 2018.

Towards a Multi-device Version of the HYFMGPU Algorithm for Hyperspectral Scenes Registration

Jorge Fernández-Fabeiro[1], Álvaro Ordóñez[2], Arturo González-Escribano[1] and Dora B. Heras[2]

[1] *Departamento de Informática, University of Valladolid, Spain*

[2] *Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Spain*

emails: jorge@infor.uva.es, alvaro.ordonez@usc.es, arturo@infor.uva.es, dora.blanco@usc.es

Abstract

Hyperspectral image registration is a relevant task for real-time applications like environmental disasters management or search and rescue scenarios. Traditional algorithms were not devoted to real-time performance, the HYFMGPU algorithm having arisen as a solution to such a lack. Sensors are expected to evolve and thus generate images with finer resolutions and wider wavelength ranges, so a multi-GPU implementation seems to be necessary in a near future. This work presents a first approach to such a multi-device version, identifying some stages of the pipeline as the most suitable to run in parallel in several GPUs. An MPI+CUDA variation of the original HYFMGPU algorithm is implemented, achieving speedups of 1.83× in 2 GPUs and 3.08× in 4 GPUs for the stages of the pipeline distributed among several devices. Different issues related to communications-derived time overloads and to some CUDA-based libraries particularities, as long as some optimization possibilities out of the currently distributed stages, were also detected. We plan to tackle them in further development stages of this multi-GPU implementation.

Key words: Hyperspectral imaging, image registration, Fourier transforms, multi-GPU, CUDA, OpenMP, remote sensing.

 ISBN: 978-84-697-7861-6

1 Introduction

The task consisting on estimating the translation, rotation and scaling parameters of a given image with respect to a second take of the same scene obtained at different times, viewpoints and/or lightning conditions is known as *image registration*. During the last years, a whole family of FFT-based image registration techniques appeared, but most of them ignored time performance. However, many real-time applications such as the management of natural disasters or surveillance operations depend on hyperspectral images being processed in real-time. GPUs were used to boost tasks like classification, target detection or segmentation of this kind of images, but few efforts were made to achieve a real-time implementation of a hyperspectral registration algorithm. In [1], Ordóñez et al. introduced HYFM, a Fourier-Mellin algorithm for hyperspectral images registration, and implemented a sequential CPU version of it. That work was followed by HYFMGPU [2], a single-GPU version whose performance makes it suitable to be used in real-time environments. Nevertheless, as both the image resolution and the wavelength range of the sensors that capture hyperspectral images improve, both the size of images and the number of bands to be processed is expected to increase. In this work we present the first steps taken to achieve a multi-GPU implementation able to satisfy these needs.

The rest of this extended abstract is organized as follows: Section 2 summarises the HYFMGPU algorithm, describing then in Section 3 the first approach to a multi-GPU version of it. The preliminary results obtained by this approach are introduced in Section 4, and finally in Section 5 we present the conclusions and some brief future research lines.

2 The HYFMGPU algorithm

The HYFMGPU algorithm expects as input a pair of hyperspectral images (*reference* and *target*), the goal being to register the target image, this is, to compute how it is rotated, shifted and scaled with respect to the reference image. The algorithm comprises six main stages, which are depicted in Figure 1.

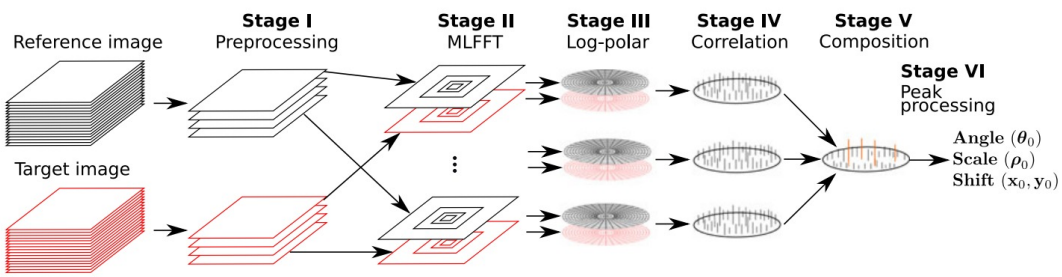

Figure 1: HYFM scheme for registration of two hyperspectral images

 ISBN: 978-84-697-7861-6

The HYFMGPU algorithm projects these stages to a single GPU by means of a CUDA implementation that relies on some specific libraries, namely cuBLAS, cuSOLVER, cuFFT, NVIDIA Performance Primitives (NPP), and Thrust. This implementation can be roughly decomposed in the following steps:

1. **Initialization**: Input images are loaded in global memory, and some arrays needed by further steps are also loaded and/or computed.

2. **Preprocessing**: In Stage I, a single-kernel Blackman filter is applied first to both inputs to remove higher frequencies, which might be detrimental for the precision of the registration. Those filtered images are centered, and then a principal component analysis (PCA) is applied to each of them in order to extract their most relevant features by retaining a reduced number of principal components (i.e., transformed bands of the original images). Both cuBLAS and cuSOLVER operations are used to implement this analysis. Finally, sizes of both reduced images are expanded to the nearest common upper power of 2, and data is transformed to complex values.

3. **Selected bands processing and composition**: In this step, pairs composed of the same PCA-extracted band from both inputs are processed by computing a high-pass filter (Stage II), a log-polar map (Stage III), and a phase correlation (Stage IV). As this three substeps are FFT-based, the underlying operations are implemented by means of the cuFFT library. Since this is a single-GPU implementation, the device is commanded to iterate over all the band pairs in order to perform this stage. Finally, a single-kernel reduction (Stage V) is performed in the device in order to average the log-polar correlated maps obtained for each pair of PCA-extracted bands.

4. **Peak processing**: The log-polar peaks contained in the average map computed in the previous step are descendently sorted in the device using the Thrust library, selecting a given number and processing them one-by-one. This process starts by rotating and scaling the first component of the target image several times using specific functions from the NPP library. Next, a phase correlation and a cuBLAS-based maximum search are performed on the cartesian grid to determine the correct angle and translation parameters. Finally, the highest peak of all the cartesian grids is selected, as its coordinates determine the *shift* parameters. In turn, their log-polar counterparts decide the *scale* factor and the rotation *angle*. All these values are the expected output of the Stage VI of the pipeline.

3 Proposal of multi-GPU parallelization

A quick glance at the HYFM pipeline depicted in Figure 1 reveals the workload comprising Stages II, III and IV as a very interesting candidate to a multi-device coarse-grain par-

 ISBN: 978-84-697-7861-6

allelization. Notice how this workload, described in the step 3 of the single-GPU CUDA implementation introduced in Section 2, is performed by a loop that iterates over each pair of bands extracted from reference and target images.

Initial OpenMP+CUDA approach

In a very initial approach we opted to implement a OpenMP-threaded version of the HYFMGPU host program, keeping a master thread to control the single-GPU parts of the multi-device implementation (Steps 1, 2 and 4), and defining a parallel section to command each GPU to run its part of the Step 3. So, once the master thread loads the images, it commands its GPU to apply the Blackman filter, run the PCA, and distribute the extracted components equally among the GPUs available. Once the program enters the parallel section, each thread commands its GPU to convert its subset of the PCA-extracted bands to complex type and perform over them in a loop Stages II, III and IV of the pipeline. After the program exits this parallel section, the master threads takes the control back, averaging the partial log-polar maps computed by each GPU (Stage V) and processing the peaks (Stage VI) in order to get the final *shift*, *scale* and *angle* outputs expected after this last stage.

As Step 3 of the CUDA implementation description from Section 2 shows, the cuFFT library is used to process a the PCA-extracted bands. Because of that, each GPUs needs its own set of FFT execution plans, whose initialization must be invoked from the corresponding OpenMP thread. The first tests showed that despite having distributed the workload among several GPUs, the execution time of the Step 3 was multiplied by the number of devices, instead of being divided as expected. So, the way how CUDA and OpenMP interact when threads are trying to prepare their corresponding cuFFT plans makes these initialization operations to be eventually sequentialized. As Lončar et al. explain in [3], combining FFT operations and OpenMP-threaded parallelizations is not a trivial task. They also recommend to use FFTW3 [4] in such cases, but there is no support for this FFT library in CUDA. To solve this cuFFT issue, the implementation we have just described was shifted from forking multiple OpenMP threads to run separated MPI processes. This way, the FFT plans needed by each GPU were prepared in different processes with their own memory spaces, which made the sequentialization problem disappear.

MPI+CUDA version

In this MPI-based evolution, a number of processes equal to the number of GPUs is launched. Since MPI follows a SPMD model, some noticeable variations were introduced in the workflow of the host program. Now each process loads its own full copy of the images, and commands its GPU to apply the Blackman filter and run the PCA on them. Despite having the all the PCA-extracted components available, each GPU is forced to perform the Stages II, III and IV of the pipeline just on its own bands subset. In this case, the interme-

 ISBN: 978-84-697-7861-6

diate log-polar maps processed by each GPU are gathered by means of MPI synchronous messages in the process controlling the GPU 0, which composes them in a final log-polar grid (Stage V) and explore the peaks (Stage VI) to compute the final result.

4 Experimental results

All tests were run in a node equipped with a dual-socket host CPU composed of 2 Intel Xeon E5-2609v3 (1.9 GHz, 6 cores each) with 64 GB of RAM, and 4 GPUs NVIDIA GeForce GTX TITAN Black (GK110B architecture, compute capability 3.5, 15 SMs with 192 CUDA cores each, up to 2880) controlled by the 384.59 driver. The CUDA code has been compiled under Linux using nvcc from CUDA Toolkit 9.0, as well as the libraries used. Multi-threaded preliminary tests were supported by linking libraries from OpenMP 3.1, whereas the MPI support was provided by mpich-3.2. The multi-GPU implementation was evaluated by registering rotated and scaled variations of the *Pavia University* test case, a common single reference hyperspectral image of 610×340 pixels and 103 bands already used in [2]. The values of the parameters that control the registration algorithm were also the same as those specified in [2], this is: 8 bands to be extracted in the PCA, vector $\alpha = \{1, 1/4, 1/16, 1/64\}$ for the FFTs performed to obtain the log-polar maps, and 50 highest peaks to be examined in the peak processing stage.

Version	Wall time in seconds (Speedup)				
	Step 1	Step 2	Step 3	Step 4	Whole algorithm
HYFMGPU	0.698 s	0.575 s	0.512 s	0.309 s	2.094 s
MPI+2GPU	0.687 s	0.569 s	0.279 s (1.83×)	0.323 s	1.852 s (1.13×)
MPI+4GPU	0.714 s	0.569 s	0.166 s (3.08×)	0.382 s	1.831 s (1.14×)

Table 1: Wall times in seconds (and selected speedups) for each step and the whole HYFM algorithm run for the *Pavia University* test case in *NVIDIA GeForce Titan Black* GPUs

Table 1 shows the wall time consumed by the original single-GPU CUDA implementation and the MPI+CUDA version when it is run in 2 and 4 GPUs to perform Steps 1 to 4 from Section 2 individually, and to run the whole algorithm. Moreover, the speedups achieved by the multi-GPU versions for both the Step 3 and the whole algorithm are shown in the corresponding entries of *Step 3* and *Whole algorithm* columns. In relation to the FFTs performed in Step 3, it must be noticed that the cuFFT API allows the programmer to disaggregate the creation of FFT execution plans [5] in three steps: (1) handle creation, (2) estimation of sizes for temporary buffers, and (3) execution plan definition. We experimentally checked that the time consumed by handles creation and plans definition was negligible with respect to the time needed to estimate the buffer sizes. Since this estimation step does not depend on the number of PCA-extracted components processed, we moved it to the initialization part of the program and, hence, the FFT plans creation

 ISBN: 978-84-697-7861-6

time is ascribed to Step 1. Along the computation of high-pass filters, log-polar grids and phase correlations, times in *Step 3* column also include the conversion to complex type of the extracted components and the calculation of some auxiliary buffers needed. It is also remarkable that the times shown in *Step 4* column increases as the number of GPUs does, since the communication of the partial results back to the process acting as master is being summed up here.

The speedups of 1.83× for 2 GPUs and 3.08× for 4 GPUs shown in *Step 3* column confirm that the loop performing the MLFFT+LogPolar+Correlation stages was a suitable candidate to be distributed among several GPUs, despite the time needed by additional auxiliary buffer computations and data communications making them to distance from the theoretical maximums of 2× and 4×, respectively. Furthermore, for the specific experiments run (*Pavia University* input images, GPUs with GK110B architecture and compute capability 3.5), the wall time consumed by this step represented about a 25% of the wall time of the whole algorithm run in a single GPU, so that Amdahl's Law is limiting the overall speedup achievable by distributing the workload of this step among 2 GPUs to 1.14×, and among 4 GPUs up to 1.23×. Therefore, by now the multi-device version is achieving a 99% and a 93% of the theoretical maximum performance when run in 2 and 4 GPUs respectively.

5 Conclusions and future work

In this work we have introduced an initial approach of a multi-device version of the original CUDA implementation of the single-GPU HYFMGPU algorithm. The speedups of 1.83× for 2 GPUs and 3.08× for 4 GPUs obtained for the Step 3 from Section 2 confirmed it as a suitable candidate to be distributed among several devices.

Furthermore, in the current experimentation conditions (*Pavia University* input images, GPUs with GK110B architecture and compute capability 3.5), the wall time of the tasks currently distributed among several GPUs only represented about a 25% of the total execution time of the HYFM pipeline, so that finding more steps of the algorithm that could be parallelized in several GPUs is another interesting research line. For instance, once the peaks are sorted at the beginning of Stage VI, they could be scattered among the GPUs. Then, each device would process a subset of the peaks locally and return its highest one to the host program, which will compute the final result. Moreover, currently the PCA is performed by a single-GPU non-iterative procedure based on that one presented in [6]. Namely, it would be quite interesting to implement a multi-GPU version of the PCA that was able not only to boost the component extraction part but also to make each subset of bands to be directly stored in the global memory of the corresponding GPU. Also more experiments with larger images (regarding both space and bands) and in different devices must be run in order to evaluate how the experimentation conditions affect the weight that each step has in the execution time of the algorithm.

 ISBN: 978-84-697-7861-6

Finally, although the usage of MPI instead of OpenMP helped to solve the cuFFT sequentialization issue, there is still a fixed time that must be consumed to estimate the sizes of the temporary buffers needed by the FFT execution plans. This inconvenience might be overcome by exploiting CUDA streamed execution, namely trying to hide the latency of these estimations among other CUDA and/or specific-purpose GPU libraries asynchronous calls [5, 7, 8].

Acknowledgements

This work has been partially supported by:

Universidad de Valladolid – Junta de Castilla y León, Ministry of Economy, Industry and Competitiveness of Spain, and European Regional Development Fund (ERDF) program: Project PCAS (TIN2017-88614-R), Project PROPHET (VA082P17) and CAPAP-H6 network (TIN2016-81840-REDT).

Universidade de Santiago de Compostela – Consellería de Cultura, Educación e Ordenación Universitaria of Xunta de Galicia (grant numbers GRC2014/008 and ED431G/08) and Ministry of Economy, Industry and Competitiveness of Spain (grant number TIN2016-76373-P), all co–funded by the European Regional Development Fund (ERDF) program.

References

[1] Á. Ordóñez and F. Argüello and D. B. Heras, *Fourier–Mellin registration of two hyperspectral images*, International Journal of Remote Sensing, Volume 38, Number 11 (2017), pp. 3253–3273.

[2] Á. Ordóñez and F. Argüello and D. B. Heras, *GPU Accelerated FFT-Based Registration of Hyperspectral Scenes*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, Volume 10, Number 11 (2017), pp. 4869–4878.

[3] V. Lončar and L. E. Young-S. and S. Škrbić and P. Muruganandam and S. K. Adhikari and A. Balaž, *OpenMP, OpenMP/MPI, and CUDA/MPI C programs for solving the time-dependent dipolar Gross–Pitaevskii equation*, Computer Physics Communications, Volume 209 (2016), pp. 190–196.

[4] M. Frigo and S. G. Johnson, *FFTW3 Library*, Massachusetts Institute of Technology (MIT), available in `http://www.fftw.org/`.

[5] NVIDIA Corporation, *cuFFT Library User's Guide*, available in `https://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf`

 ISBN: 978-84-697-7861-6

[6] A. S. Garea and D. B. Heras and F. Argüello, *GPU classification of remote-sensing images using kernel ELM and extended morphological profiles*, International Journal of Remote Sensing, Volume 37, Number 24 (2016), pp. 5918–5935.

[7] NVIDIA Corporation, *cuBLAS Library User's Guide*, available in `https://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf`

[8] NVIDIA Corporation, *cuSOLVER Library User's Guide*, available in `https://docs.nvidia.com/cuda/pdf/CUSOLVER_Library.pdf`

 ISBN: 978-84-697-7861-6