Evaluating GPU Acceleration of SoFiA-2

Carlos Carreras
Integrated Systems Laboratory (LSI)
Universidad Politécnica de Madrid (UPM)

Report - Project AMIGA-7
December 15, 2022

Abstract

SoFiA-2 (Source Finding Application) is a scientific software aimed at detecting and characterizing galaxies in extragalactic HI data cubes. As the SoFiA-2 processing times can be long, it could benefit from the hardware acceleration provided by GPUs. This document briefly summarizes the modifications introduced in SoFiA-2-2.5.0 to evaluate GPU acceleration using both, CUDA and OpenCL, as well as the performance results that have been obtained.

1 Introduction

SoFiA-2 [1, 2, 3] is a pipeline for the automated detection of radiation sources from space images stored as data cubes. Central in the pipeline is the Smooth + Clip Finder (S+C) stage, where the data cube is iteratively filtered for smoothing the image and eliminating noise. Since this is the most time consuming stage of the pipeline, it is the one considered here for hardware acceleration. Future implementations may want to consider the acceleration of other stages that also contribute significantly to execution times.

The S+C Finder performs two main tasks in each iteration: filtering the data cube in the spatial and spectral domains, and building a mask according to the registered noise levels and a noise threshold. Optionally, the noise in the data cube can be scaled before the masking operation. In this evaluation, only the previous two main tasks (including their accompanying auxiliary functions) have been approached for GPU acceleration. The optional noise scaling is still available if required, but it is performed in the host CPU.

Although two different languages have been used in the evaluation, CUDA [4] has been the preferred choice due to its simplicity, despite longer compilation times. On the other hand, OpenCL [5] has also been used to accelerate the filtering and some auxiliar operations on the data cube in order to obtain some initial results.

2 Code Modifications

The original code of SoFiA-2-2.5.0 has been modified to build an accelerated version called SoFiA-2-2.5.0_gpu. This version includes support for CUDA-based and OpenCL-based acceleration, while maintaining all the features of the original code.

The SoFiA-2 original files that have been slightly modified in version SoFiA-2-2.5.0_gpu are: `compile.sh`, `src/Parameter.c`, `template_par_file.par`, `sofia.c`, `src/DataCube.h` and, optionally, `src/common.c`.

The version SoFiA-2-2.5.0_gpu includes the following new files: `src/DataCubeCUDA.h` and `src/DataCubeCUDA.cu` for CUDA code, and `src/DataCubeOCL.h`, `src/DataCubeOCL.c` and `src/DataCubeKernel.cl` for OpenCL code. In these files, methods and kernels have been documented using the same conventions used in the rest of the SoFiA-2 package.

Compilation for GPU acceleration is performed by running the modified script `compile.sh` (Makefile has not been modified), and requires that the CUDA toolkit with the `nvcc` compiler has been previously installed. The script, before the final compilation of `sofia.c`, checks if `nvcc` is in the system and if so, it compiles the CUDA and OpenCL codes which are then linked in the compilation of `sofia.c`. Otherwise, `sofia.c` is compiled as in the original code. The script assumes that the CUDA toolkit libraries are in the the default location `/usr/local/cuda/lib64`.

The file `src/Parameter.c` has been modified to include a new S+C Finder parameter called `scfind.accelerator`. The file `template_par_file.par` has also been modified accordingly. By default, its value is empty, meaning that no accelerator is used and all computations are CPU-based. This parameter is used to select GPU-based computations by using one of the values `gpu_cuda` or `gpu_ocl`.

The `sofia.c` file has been modified to check the previous parameter value and, if GPU-based computations have been selected and `nvcc` was be used during compilation, call the appropriate S+C Finder method. The new S+C Finder methods that perform computations in the GPU, namely *DataCube_run_scfind_CUDA()* and *DataCube_run_scfind_OCL()*, are declared in `src/DataCube.h`. They, and the methods and kernels called from them, are defined and implemented in the new files added to the package that have been listed above.

Finally, the `src/common.c` has been modified so that method *timestamp()* prints CPU times including also milliseconds. Althought this change has been useful during development to measure execution times for a small test cube, it is in no way required for the accelerator code to work.

2.1 Acceleration using CUDA

Before the S+C Finder loop, method *DataCube_run_scfind_CUDA()* initializes the GPU, copies the original data and mask cubes to the GPU, and measures the noise in the original cube also in the GPU. Then, assuming that noise is not to be scaled, the complete S+C Finder loop is run in the GPU. This includes making a copy of the original cube, setting the flux to masked pixels, performing spatial and spectral filtering, copying again blanks from the original cube, measuring noise of the smoothed cube and adding pixels above the threshold to the mask.

In the case that noise has to be scaled, this task is performed in the host in all cases. This means that in each iteration the smoothed cube is copied back and forth between GPU and CPU memory, thus heavily penalizing performance. Considering possible future developments and given that kernels for measuring noise have already been developed, spectral noise scaling could possibly be implemented in CUDA without much effort. However, it is foreseen that a CUDA implementation of local noise scaling will probably require more effort.

2.2 Acceleration using OpenCL

In the case of *DataCube_run_scfind_OCL()*, it also includes all the OpenCL initialization and definitions before the S+C Finder loop. Within the loop, OpenCL code has been developed to set the flux to masked pixels in a copy of the original cube, performing spatial and spectral filtering, and copying again blanks from the original cube. Then, the smoothed cube is moved back to the host for possible noise scaling and to measure the cube noise. At the end of the iteration, the cube is moved back again to the GPU memory to add pixels above the threshold to the mask.

Therefore, in each iteration, the cube is always processed in both, the GPU and the CPU, and transfer times significantly increase execution times. So the OpenCL version of the code is in earlier stages than the CUDA code.

3 Performance Evaluation

All the experiments aimed at performance evaluation of hardware acceleration of SoFiA-2 have been run in the Magerit-3 cluster [6] at CeSViMa (Madrid Supercomputing and Visualization Center), UPM. In particular, the jobs have been executed in nodes with 2 × Intel Xeon Gold 6240R (24 cores @ 2.4 GHz and 192 GB of RAM) and 4 NVIDIA A100 (40 GB of RAM).

Three data cubes in `.fits` format have been used in the experiments. First, the cube provided for development in the SoFiA-2 repository, which we call here *test* [7], with a size of 16 MB. Second, one of the cubes that at some point were available at a website containing ASKAP simulations, which we call here *askap* [8], with a size of 1.07 GB. And third, one of the simulated data cubes provided at the SDC2 website of SKA 2020, which we call here *sdc2* [9], with size of 11.03 GB. As explained below, this last cube could only be used in the OpenCL experiments.

All following results are included in folder SoFiA-2-2.5.0_results which is provided together with the SoFiA-2-2.5.0_gpu code. The `/out*` subfolders contain the standard output provided by the simulations, where A files correspond to *test*, B files correspond to *askap*, and C files correspond to *sdc2* simulations, respectively. The `/ref*` subfolders contain the files generated by the simulation of *test* with default parameters. They can be used to verify the correctness of the code by comparing them with those in folder `/refcpu` which contains the reference files generated by the original code.

3.1 CUDA Results

The standard output files from GPU-CUDA (and corresponding CPU) simulations are in folder `/outcuda`. Reference results when running *test* are in folder `/refcuda`. Only the *test* and *askap* data cubes are used in the CUDA experiments, since processing the *sdc2* cube would trigger and out-of-memory call in the GPU. At this point, it is not clear why the GPU would run out of memory for that cube and no more time is available for debugging the problem. However, it seems it may occur when a function from the CUDA *thrust* library tries to allocate memory in the GPU.

Tables 1 and 2 collect the CPU and GPU-CUDA execution times obtained when processing *test* and *askap* with a number of different SoFiA-2 parameter combinations. Each parameter combination (excluding accelerator selection) is given a number which is used to identify it in all simulation runs. Three major sets of results are included, corresponding to:

- No noise scaling (`scaleNoise:enable=false,(mode=local),scfind=false`)
- Spectral noise scaling (`scaleNoise:enable=true,model=spectral,scfind=true`)
- Local noise scaling (`scaleNoise:enable=true,model=local,scfind=true`)

Within each set, the three statistics used to measure noise in the S+C Finder (`mad`, `std`, `gaussian`) are evaluated as they all have been implemented in CUDA. Finally, `negative` and `positive` values of `scfind.fluxRange` are considered for each statistic in order to verify the correctness of the implementation and to obtain a pair of values per statistic. GPU-CUDA execution is selected with value `gpu_cuda` for parameter `scfind.accelerator`. Default values are used for all remaining parameters.

The precision of the results is noted in the last column of the table, `Err`, which indicates by means of symbol $\simeq$ that differences are found between the standard output of the CPU and GPU executions. These differences are usually quite small. The can occur in the measured noise levels, the number of pixels detected and/or the number of sources found. We can't evaluate the impact of these differences as we are not experts in the application. However, they can be easily extracted for analysis by applying the `diff` Unix command to the provided standard output files, as in '`diff cpuB07.txt gpuB07.txt | less -r`'.

Cube	scaleNoise	scfind				CPU Time (sec)	Speedup	Err
		statistic	fluxRange	#	accelerator			
test	enable=false (mode=local) scfind=false	mad	negative	01	(cpu) gpu_cuda	1.545 0.298	5.18	≃
			positive	02	(cpu) gpu_cuda	1.542 0.298	5.17	
		(std)	negative	03	(cpu) gpu_cuda	1.472 0.279	5.28	
			positive	04	(cpu) gpu_cuda	1.413 0.228	6.20	X ≃ X
		gauss	negative	05	(cpu) gpu_cuda	1.574 0.306	5.14	
			positive	06	(cpu) gpu_cuda	1.563 0.297	5.26	
	enable=true mode=spectral scfind=true	mad	negative	07	(cpu) gpu_cuda	1.823 0.660	2.76	≃
			positive	08	(cpu) gpu_cuda	1.823 0.658	2.77	≃
		(std)	negative	09	(cpu) gpu_cuda	1.759 0.650	2.71	
			positive	10	(cpu) gpu_cuda	1.702 0.596	2.86	≃
		gauss	negative	11	(cpu) gpu_cuda	1.864 0.677	2.75	
			positive	12	(cpu) gpu_cuda	1.793 0.297	2.94	≃
	enable=true mode=local scfind=true	mad	negative	13	(cpu) gpu_cuda	4.635 3.485	1.33	≃
			positive	14	(cpu) gpu_cuda	4.417 3.225	1.37	X ≃ X
		(std)	negative	15	(cpu) gpu_cuda	4.595 3.513	1.31	≃
			positive	16	(cpu) gpu_cuda	4.323 3.203	1.35	≃
		gauss	negative	17	(cpu) gpu_cuda	4.702 3.570	1.32	≃
			positive	18	(cpu) gpu_cuda	4.413 3.225	1.37	

Table 1: Performance of the CUDA implementation with *test*

In some cases, differences can be due to changes in the ordering of the computations, since the floating-point format does not verify the associative property. However, most probably these differences occur because the original algorithms have been substituted by parallel versions that may have a slightly different behavior. For example, as documented in the code, when computing the MAD statistic, the median of an array is computed using a parallel cutting plane algorithm, which is rather different from the original sequential algorithm.

When a symbol X is found in the `Err` column, it means that the execution terminated prematurely. This has occurred when executing parameter combinations #04 and #14 with the *test* data cube, for both, the original code and the GPU codes.

Perfomance results show acceleration for all cubes and all parameter combinations. In the case of *test*, it is a data cube mainly used to verify correctness. It is a small cube and CPU times are also very small so parallelization has little impact. However speeups show the same tendencies that in the larger *askap* data cube. With this cube, very interesting speedups of around 35 are obtained when the execution of S+C Finder is completely performed in the GPU in parallel (i.e. without noise scaling). When noise scaling is applied, two effects occur. First, the transfer of the cube to and from the host memory in each loop iteration increases the overall execution times. Second, the percentage of execution time spent in non-parallel sections of the code (i.e. noise scaling) increases so, according to Amdahl's law, speedups from parallelization are reduced. In particular, speedups of around 10 when selecting `spectral` scaling, and of around 2 when selecting the costly `local` scaling have been obtained.

Cube	scaleNoise	scfind				CPU Time (sec)	Speedup	Err
		statistic	fluxRange	#	accelerator			
askap	enable=false (mode=local) scfind=false	mad	negative	01	(cpu) gpu_cuda	351.079 10.144	34.61	≃
			positive	02	(cpu) gpu_cuda	354.371 10.155	34.90	≃
		(std)	negative	03	(cpu) gpu_cuda	349.949 9.964	35.12	≃
			positive	04	(cpu) gpu_cuda	347.095 9.845	35.26	≃
		gauss	negative	05	(cpu) gpu_cuda	351.900 9.978	35.27	≃
			positive	06	(cpu) gpu_cuda	351.536 9.991	35.19	≃
	enable=true mode=spectral scfind=true	mad	negative	07	(cpu) gpu_cuda	373.507 41.211	9.06	≃
			positive	08	(cpu) gpu_cuda	391.265 40.304	9.71	≃
		(std)	negative	09	(cpu) gpu_cuda	371.896 37.032	10.04	≃
			positive	10	(cpu) gpu_cuda	384.892 41.346	9.31	≃
		gauss	negative	11	(cpu) gpu_cuda	374.990 37.121	10.10	≃
			positive	12	(cpu) gpu_cuda	375.962 37.196	10.11	≃ ≃ ≃
	enable=true mode=local scfind=true	mad	negative	13	(cpu) gpu_cuda	583.104 247.473	2.36	≃
			positive	14	(cpu) gpu_cuda	623.835 267.814	2.33	≃
		(std)	negative	15	(cpu) gpu_cuda	583.259 249.029	2.34	≃
			positive	16	(cpu) gpu_cuda	610.915 243.123	2.51	≃
		gauss	negative	17	(cpu) gpu_cuda	587.667 274.175	2.14	≃
			positive	18	(cpu) gpu_cuda	625.064 242.310	2.60	≃

Table 2: Performance of the CUDA implementation with *askap*

3.2 OpenCL Results

The standard output files from GPU-OpenCL (and corresponding CPU) simulations are in folder `outocl`. Although some of the CPU output files were already available from the experiments in the previous section, they have been obtained again here to show that, in fact, there is some variability in the execution times when running simulations with the same parameters. Reference results when running *test* with OpenCL are in folder `refocl`.

Table 3 collects the CPU and GPU-OpenCL execution times obtained when considering a number of different SoFiA-2 parameter values. The same sets ot results considered for CUDA are included here (no noise scaling, spectral noise scaling and local noise scaling). However, only the MAD statistic for measuring noise is used as noise measurements are not parallelized and always take place in the host.

In this case, the three available data cubes, *test*, *askap*, and *sdc2* are used since no out-of-memory call is triggered in the GPU when using OpenCL. Again, *test* runs are aimed at validation and their results are not very significant due to the limited data size.

In the OpenCL implementation, it is always required to move the data cube to and from the host in each iteration of S+C Finder, since noise levels are always determined in the host. Therefore, speedups are reduced with respect to CUDA. With *askap*, speedups of around 19 are obtained without noise scaling. They drop to 10 and 2.5 with `spectral` and `local` noise scaling, respectively, as these tasks become more relevant when only filtering is performed in parallel. This effect is clearly seen with the larger cube *sdc2*, as the time devoted to transfer the cube and to perform the sequential host computations mostly compensates the gains of parallel filtering, with speedups always below 3.

Cube	scaleNoise	scfind				CPU Time (sec)	Speedup	Err
		statistic	fluxRange	#	accelerator			
test	enable=false (mode=local) scfind=false	mad	negative	01	(cpu) gpu_ocl	1.459 0.598	2.44	
			positive	02	(cpu) gpu_ocl	1.477 0.580	2.55	
	enable=true mode=spectral scfind=true	mad	negative	07	(cpu) gpu_ocl	1.742 0.862	2.02	$\simeq$
			positive	08	(cpu) gpu_ocl	1.831 0.853	2.15	
	enable=true mode=local scfind=true	mad	negative	13	(cpu) gpu_ocl	4.417 3.501	1.26	$\simeq$
			positive	14	(cpu) gpu_ocl	4.162 3.249	1.28	X $\simeq$ X
askap	enable=false (mode=local) scfind=false	mad	negative	01	(cpu) gpu_ocl	367.755 18.947	19.41	$\simeq$
			positive	02	(cpu) gpu_ocl	371.418 18.667	19.90	$\simeq$
	enable=true mode=spectral scfind=true	mad	negative	07	(cpu) gpu_ocl	397.711 39.845	9.98	$\simeq$
			positive	08	(cpu) gpu_ocl	392.222 39.694	9.88	
	enable=true mode=local scfind=true	mad	negative	13	(cpu) gpu_ocl	633.642 254.358	2.49	$\simeq$
			positive	14	(cpu) gpu_ocl	632.242 251.681	2.51	$\simeq$
sdc2	enable=false (mode=local) scfind=false	mad	negative	01	(cpu) gpu_ocl	1499.803 548.057	2.74	$\simeq$
			positive	02	(cpu) gpu_ocl	1492.838 560.089	2.67	$\simeq$
	enable=true mode=spectral scfind=true	mad	negative	07	(cpu) gpu_ocl	1642.443 661.175	2.48	$\simeq$
			positive	08	(cpu) gpu_ocl	1631.416 653.945	2.49	$\simeq$
	enable=true mode=local scfind=true	mad	negative	13	(cpu) gpu_ocl	3766.341 2820.922	1.34	$\simeq$
			positive	14	(cpu) gpu_ocl	3750.708 2902.973	1.29	$\simeq$

Table 3: Performance of the OpenCL implementation

3.3 Comparison between CUDA and OpenCL

Since the original code is only commented away in the CUDA and OpenCL implementations of the S+C Finder, it is quite simple to modify the CUDA code to have the same behaviour that the OpenCL code, that is, to perform noise measurements in the host (this modified code is not provided). This allows to compare the performance of both languages.

The corresponding standard output files from the modified GPU-CUDA and the GPU-OpenCL (and corresponding CPU) simulations are in folder `outcudaocl`. Reference results when running *test* with the modified CUDA code are in folder `refcudaocl`.

In this case, only parameter combinations #01 and #02 (no noise scaling with MAD statistic for S+C Finder) are considered for the *test* and *askap* data cubes. The performace results are collected in Table 4.

Cube	scaleNoise	scfind				CPU Time (sec)	Speedup	Err
		statistic	fluxRange	#	accelerator			
test	enable=false (mode=local) scfind=false	mad	negative	01	(cpu) gpu_cuda_mod gpu_ocl	1.450 0.526 0.576	– 2.76 2.52	
			positive	02	(cpu) gpu_cuda_mod gpu_ocl	1.451 0.524 0.575	– 2.77 2.52	
askap	enable=false (mode=local) scfind=false	mad	negative	01	(cpu) gpu_cuda_mod gpu_ocl	348.941 17.043 18.757	– 20.47 18.60	 $\simeq$ $\simeq$
			positive	02	(cpu) gpu_cuda_mod gpu_ocl	351.653 17.049 18.796	– 20.63 18.71	 $\simeq$ $\simeq$

Table 4: Performance comparison between modified CUDA and OpenCL

While the performance of both implementations is quite similar, the CUDA implementation seems a little faster. In any case, the differences are small and of the same order than the variability of the measured times (as observed when comparing CPU times with those in Tables 1, 2 and 3 or OpenCL times with those in Table 3 for the same simulations).

3.4 Parallel execution with OpenMP

In order to compare the impact of GPU acceleration with parallel execution based on shared memory using OpenMP [10], a few experiments with data cube *askap* have been run in the Magerit-3 cluster compiling SoFiA-2 with the `-fopenmp` option. In particular, a Magerit-3 node allows up to 40 OpenMP threads, so this is the number of threads used in the experiments.

In fact, two versions of the OpenMP code are considered. First, the full version included in the original distribution that parallelizes not only the S+C Finder but also other parts of SoFiA-2. And second, a modified version that only uses OpenMP in the S+C Finder, so it resembles the GPU implementations presented in this report.

Table 5 collects the results of the modified and full OpenMP (and corresponding single CPU) simulations. No noise scaling is considered. The three statistics used to measure noise levels are evaluated as in the CUDA results. In this case, elapsed times provided in the standard output are used to compute the speedups, since CPU times are the sum of all execution times in the cores and, therefore, don't reflect parallel execution times.

Cube	**scaleNoise**	**scfind**				**Elapsed Time (sec)**	**Speedup**	**Err**
		statistic	**fluxRange**	**#**	**accelerator**			
askap	enable=false (mode=local) scfind=false	mad	negative	01	(cpu) (cpu-openmp)_mod (cpu-openmp)	353 29 22	– 12.17 16.05	
			positive	02	(cpu) (cpu-openmp)_mod (cpu-openmp)	356 29 22	– 12.28 16,18	
		(std)	negative	03	(cpu) (cpu-openmp)_mod (cpu-openmp)	352 29 22	– 12.14 16.00	
			positive	04	(cpu) (cpu-openmp)_mod (cpu-openmp)	348 28 21	– 12.43 16.57	
		gauss	negative	05	(cpu) (cpu-openmp)_mod (cpu-openmp)	353 29 25	– 12.17 14.12	
			positive	06	(cpu) (cpu-openmp)_mod (cpu-openmp)	353 28 25	– 12.61 14.12	

Table 5: Performance of modified OpenMP and full OpenMP with *askap*

Results show that when using 40 threads and all the OpenMP parallelization available in SoFiA-2 speedups are between 14 and 16. When only the S+C Finder stage uses OpenMP parallelization, speedups fall to around 12. So under the same conditions, the GPU runnig CUDA achieves a speedup of around 3 over a multicore processor running 40 OpenMP threads.

4 Conclusions

GPU hardware acceleration appears as a very promising alternative to run the huge data cubes that will be obtained from SKA. The speedups around 35 obtained when the complete S+C Finder loop is executed in the GPU (i.e. without noise scaling) are quite impressive and well above what is currently obtained with 40 OpenMP threads.

However, the cost of transforming the actual SoFiA-2 code to CUDA (or OpenCL) is relatively high. It would probably be even better to change the current sequential algorithms having parallelism in mind, before generating CUDA or OpenCL code. Obviously, this is not an easy task. In the meantime, it maybe worthy to continue the effort started here and to port

all the noise scaling methods to CUDA so the S+C Finder is always run in the GPU, obtaining even higher speedups as the time percentage of execution in parallel increases. Of course, the same applies if other time consuming sections of SoFiA-2 are also ported to CUDA.

SKA data cubes are expected to be huge so they will probably have to be partitioned in order to fit the memory available in the computing nodes. Additionally, reducing the amount of memory required for processsing by SoFiA-2 algorithms (i.e. memory optimization) would much desirable, not only for GPU-based implementations, but also for standard CPU executions. Nevertheless, GPUs would probably benefit more from such optimization because high-end GPUs usually support less RAM than high-end CPUs.

5 Acknowledgements

The authors gratefully acknowledge the Universidad Politécnica de Madrid (www.upm.es) for providing computing resources on Magerit Supercomputer.

References

[1] Serra, P., Westmeier, T., Giese, N., et al., *SoFiA: a flexible source finder for 3D spectral line data*, 2015, MNRAS, 448, 1922, "http://adsabs.harvard.edu/abs/2015MNRAS.448.1922S"

[2] Westmeier, T., Kitaeff, S., Pallot, D., et al., *SoFiA 2 – An automated, parallel HI source finding pipeline for the WALLABY survey*, 2021, MNRAS, 506, 3962, "http://adsabs.harvard.edu/abs/2021MNRAS.506.3962W"

[3] Westmeier, T., et al., *SoFiA-2*, "https://github.com/SoFiA-Admin/SoFiA-2"

[4] NVIDIA Corporation, *CUDA Toolkit Documentation*, "https://docs.nvidia.com/cuda"

[5] Khronos Group, *Open-CL, Open Standard for Parallel Programming of Heterogeneous Systems*, "https://www.khronos.org/opencl"

[6] CeSViMa, Centro de Supercomputación y Visualización de Madrid, Universidad Politécnica de Madrid, *Magerit-3 cluster* "https://www.cesvima.upm.es/infrastructure/magerit"

[7] Westmeier, T., et al., *SoFiA test data cube*, "https://github.com/SoFiA-Admin/SoFiA-2/wiki"

[8] Matthew Whiting, *ASKAP Simulations, ASKAPsim_Spectralline_Apr2010_fullspectrumsubset*, "https://www.atnf.csiro.au/people/Matthew.Whiting/ASKAPsimulations.php"

[9] SKA 2020, *SDC2 Challenge, sky_dev_v2*, "https://sdc2.astronomers.skatelescope.org/sdc2-challenge/data"

[10] OpenMP, *OpenMP API 5.2 Specification*, "https://www.openmp.org/specifications"