Parallelization of the Friends-of-Friends Halo Finder Algorithm for a Hybrid Computing with OpenACC

Ana Luisa Solórzano[a], Andrea Schwertner Charão[a],
Haroldo Fraga de Campos Velho[b] and Renata Sampaio da Rocha Ruiz[b]
[a]Universidade Federal de Santa Maria, RS, Brazil
[b]National Institute for Space Research, São José dos Campos, SP, Brazil

Abstract

The use of accelerators to process application sections that deals with large amounts of data is seen as a promissory field in hybrid computing. Cosmology area records giant and heterogeneous databases to be analyzed using classification procedures. A well known algorithm to classify astronomical data is Friends-of-Friends (FoF). A framework for cosmological studies is to apply N-body simulations, and the analysis of such data can be done with FoF. In this work, we present parallel versions of the FoF with $O(N^2)$ time complexity using the OpenACC standard in an environment with CPU and GPU. The results show that the algorithm can be well explored in this context, with an expressive performance improvement when rewriting the original program code to the hybrid environment and with a significant improvement when parallelized with minimal modifications using only compiling directives.

Keywords: hybrid computing, parallel computing, Friends-of-Friends algorithm, astronomy.

1 Introduction

Computational analysis of astronomical data and N-body cosmological simulations addresses large-scale problems. Development of efficient computer programs with a low execution time is primordial to process data used to predict the formation of large structures observed in the Universe, such as galaxies and clusters of galaxies [10, 7].

To pursue high performance executions, developers can explore the hybridity of parallel architectures to process data. Currently, there are multiple accelerator devices used as co-processors in many high performance computing sites. The GPUs (Graphic Processing Units) is a very used device, viewed as many-core parallel architectures for scientific computing.

The Friends-of-Friends (FoF) grouping algorithm is one of the most popular percolation methods to classify astronomical data based on physical proximity [4, 5]. In previous works, FoF was implemented with high computational cost of $O(N^2)$ time complexity to run sequentially in a processor [11, 13]. There were parallel versions of the code for a multi-core processor using CPU [12, 1]. However, their behavior in a hybrid computing environment was not explored yet.

In this work, we present parallel versions of the FoF with $O(N^2)$ time complexity using the OpenACC standard in an environment with CPU and GPU. We start using compiler directives and minimum code modification, and continue with a new approach rewriting the original program code, that brought an significant performance improvement comparing with the first approach.

2 Friends-of-Friends Algorithm

The Friends-of-Friends (FoF) is a percolation algorithm used in N-body simulations to identify structures in the Universe [5, 2]. For this, it starts reading a single free parameter that determines the linking length of percolation (l) and a database as input file.

The size of l is often given by b times the mean separation between the particles, with the values of b and l depending on the nature of the application [2]. The higher l is, the lower the density contrast and the greater the number of particles attached to the groups [10]. Commonly, the used value of b is 0.2 to identify halos of agglomerates [6, 8]. In this paper, as in Caretta and Ruiz, we will use $l = 0.1$ *Mpc* for haloes of galaxies.

The FoF basic idea is: considering a sphere of radius R around a particle, the algorithm checks if there are other particles inside it, if so, they are considered belonging to the same group and called "friends". After, it performs the same procedure for each of the other particles, until no new friend can be added to the group.

Algorithm 1 describes the main processing of FoF, composed of three nested loops, using arrays `igru`, to index the groups of particles and `x`, `y`, and `z` to store the position of each particle.

```
for (i = 0 ; i < N ; i++){
  k++;
  while (igru[i] != 0 )i++;
  igru[i] = k;
  for (j = i ; j < N ; j++){
```

```
      if(igru[j] == k) {
        for (l = (i + 1) ; l < N ; l++){
          if (igru[l] == 0){
            dist=sqrt((x[j]-x[l])*(x[j]-x[l])+(y[j]-y[l])*
            (y[j]-y[l])+(z[j]-z[l])*(z[j]-z[l]));
            if (dist <= rperc)
              igru[l] = k;
          }
        }
      }
    }
}
```

Algorithm 1: Sequential FoF of complexity $O(N^2)$

The outermost loop iterates through all particles creating a new group if the current particle is not yet classified. Otherwise, if it already belongs to one group, it continues with the next particle.

3 Parallelization with OpenACC

OpenACC [9] is a standard for parallel programming proposed in 2011, in a partnership between NVIDIA, Cray, Portland Group (PGI) and CAPS Enterprise. Designed to ease programming in hybrid systems consisting of CPU/GPU, this standard uses compiler directives to express parallelism. It also requires compiler support for recognizing the directives and generating optimized code for different architectures.

The OpenACC standard enables the generation of parallelized code written in C/C++ and Fortran languages to run on a system composed of a host (CPU) and an accelerator device (GPU), without needing to know further about its architecture, using only compilation directives. In addition, it has high portability and allows to generate a code with few modifications compared to the original one.

4 Parallel Implementations

FoF is an algorithm heavily dependent on input data values, specially the particles position. Because of this, each GPU core would perform a different amount of processing. Is the case if many particles were assigned to a group

during the first particles classification causing less processing to the last ones, leaving GPUs threads idle.

One potential solution to that could be use different GPU kernels to perform in certain parts of the processing or tackled with a dynamic work balancing strategy. However, setting less kernels to the main process could quickly exceed the available global GPU memory, and the use of load balancing should be adapted for each entry case.

In the next subsections, we present the two approaches used to parallelize the FoF using OpenACC. Our first version used a directive-based programming paradigm, which fosters incremental parallelization with minimum code modification, and in our second one we chose to rewrite the original program code focusing on the hybrid environment.

4.1 First approach using OpenACC

In the first approach we parallelized the innermost loop of the algorithm, since it has no data dependency. To this, we use the directive `#pragma acc parallel`, that describes a portion of code to run on the GPU kernels, and set the variable `dist` as private to avoid inconsistent values.

Since the parallel loop reads and can write in the `igru` vector, we found two similar possibilities to ensure data consistency: (i) use the `#pragma acc copy` directive, that copy the vector from the CPU to the GPU before entering the parallel region and from GPU to CPU before leaving, or (ii) allocate the vector in the accelerator and use the `#pragma acc update device` and the `#pragma acc update self` directives, that copy only the modified values in the vector from the local memory to the device (GPU) and from the device to the local memory. For the other vectors that are just read, we use the `#pragma acc data copyin` before the outermost loop, to copy the read values to the GPU once.

4.2 Second approach using OpenACC

Our second approach consists of dividing the domain into several GPU kernels, each one performing the grouping of its data using FoF. The number of kernels is set by command line by the user, determining the parallelism in the accelerator.

In CPU, the program reads the input file and divide the total of particles to the number of kernels chosen. After, the program uses a loop to distribute the executions of each kernel over its portion of data. For this, we use the `#pragma acc parallel loop`, allocating the needing vectors in GPU.

In GPU, we use OpenACC directives to guarantee that the data was present in the accelerator using `#pragma acc data present` and that the FoF would run sequentially by the kernels. After identifying the groups, each kernel mapped the identified groups in an array sent from the CPU to the GPU at the beginning of the parallel region. Thus, at the end of the executions this vector can be accessed in CPU with the new values set.

We observed that the original FoF of complexity $O(N^2)$ does not address a case where a particle is classified as belonging to one cluster and afterwards the algorithm classifies it as belonging to another cluster, that is, particles responsible for linking two distinct clusters and that were classified by distinct kernels. This fact caused a delay and confusion during the implementation, to compare correct results with the OpenACC versions. Also, in order to treat this case we performed a post-processing in CPU.

To treat the reported problem, we first ordered the particles by one of their position coordinates just after being read. After the executions and grouping classifications by each kernel, we do the post-processing in CPU. This was done calculating the distance between two points for doubles of particles, considering the percolation radius as the distance limit between them. When one particle was associated with another of a distinct group, it was necessary to change the group of particles associated with it. This reclassification generated a new particle scan using a `for` loop parallelized with the OpenMP [3] pattern.

5 Experiments and Results

We performed experiments to compare the performance of the two parallel approaches with OpenACC against the original, $O(N^2)$ algorithm. All experiments were executed on a server node with two Intel Xeon E5-2650 v3, each with 10 cores running at 2,3GHz, with Hyperthreading^{TM}, amounting to 20 physical cores, 128 GB DDR4 RAM, and two GPUs NVIDIA Tesla K80 with 2 x 2496 CUDA Threads, running Ubuntu 18.04.1 LTS. We use the Portland Group Community edition compiler[1] version 18.4-0.

The executions were performed with three samples of observational data from the Virgo Consortium[2], one with 65,536 particles (File 1), 174,761 (File 2) and 249,420 (File 3). We present below the mean execution times for 30 executions of each algorithm. As output, the FoF shows how many groups with more than one element were found using the percolation method and

[1] https://developer.nvidia.com/openacc-toolkit
[2] http://www.mpa-garching.mpg.de/Virgo/data_download.html

the execution time in microseconds. The source codes can be accessed in: https://github.com/anaveroneze/laquibrido/tree/master/ccis2019.

5.1 First approach

Table 1 presents the mean execution times and speedup of the two algorithms generated in the first approach. The first one, using the data transfer directive `copy` and the second using the `update device` and `update self` transfer directives.

Table 1: Performance of first approach algorithms with OpenACC

File	Time (s)	Speedup	Time (s)	Speedup
1	15.07	3.03	13.77	3.32
2	87.98	3.49	78.47	3.91
3	2478.90	2.17	1401.80	3.83

The results showed that our first approach is able to accelerate executions with the addition of a few lines of code to the sequential version. Even without a significant speedup, this approach is worth noting this was obtained at the cost of only a few compiler directives.

We also observed a time difference between the two versions, resulting in a speedup difference of 1.66 for the larger file. Since in the first version the OpenACC directive writes the entire vector from GPU to CPU at each iteration and in the second version it only updates the altered data in the vector between GPU and CPU, we assume that the first version has a greater cost processing compared to the second one, which had a higher performance in all cases.

Despite using the parallel processing, with a gain up to 3.91 to the second version, the program did not explore well the GPU usage to the FoF execution, as observed with the nvprof tool and described in section 5.3. We assume that this occurs due to the logic used in the FoF algorithm, which contains several conditional structures and data dependencies, which made it difficult to parallelize it in the device used.

5.2 Second approach

Table 2 presents the results without considering the reading time of the input, and considering the sum of FoF execution time in GPU with the post-processing execution time in CPU. We set the number of kernels as a parameter to the user choose depending on the number of particles in the

input file. For our measurements, we used 300 kernels for the File 1 and 2 and 900 for File 3.

Table 2: Performance of second approach algorithms with OpenACC

File	Time (s)	Speedup
1	1.32	34.62
2	6.28	48.90
3	286.63	18.76

From the results, we noticed a gain of performance of the new implementation compared to the serial and to the first approach. It stands out the speedup of almost 49 for File 2, where we found a case that properly exploited the GPU kernels defined to carry out the processing.

The particle sorting at the beginning of the program has generated a distinct classification of groups compared to that performed by the serial $O(N^2)$ FoF. This happens because the original algorithm does not address the cases treated in the post processing, and so the values may differ. Thus, for File 1, 46,382 groups were found in the serial version and 45,683 in the new version with OpenACC, 113,660 and 111,313 for File 2, and 157,682 and 154,384 for File 3.

We did not observe relevant performance differences using OpenMP, so the post-processing run sequentially in the tests. We supposed that this occurs because using a low percolation value (as 0.1), the association of particles in the post-processing tends to decrease, becoming more costly in some cases.

5.3 GPU performance

In order to better understand our results, we used the nvprof[3] profiling tool, which dynamically analyzes the execution of a program. The tool is executed by command line and presents an overview of the GPU kernels and memory copies used in an application. In the execution option used, it shows the total time and percentage of the total application time for each kernel.

Here we will present analysis of the percentage of time spent in monitored activities by the nvprof. The values are about the total time in each category: GPU activities, API calls and the use of OpenACC directives. To the first approach, we will refer as *first version* the algorithm with `copy`

[3]https://developer.nvidia.com/nvidia-visual-profiler

transfer directives and *second version* the algorithm that used `update self, device`.

- **GPU activities:** For the two versions of the first approach the GPU kernels were mostly used to perform the communication between host and device. In the first version, all inputs presented similar results in this question, using about 46% to transfer data from the host to the device and 45% from the device to the host, leaving only 6% to 8% to run the FoF.

  The second version spent most of the time transferring data from device to host, with 51% to the first file until 77% to the third file. To transfer from host to device it spent around 39% for the first file and 11% for the third. With this, the second version used more time to process the FoF in GPU: from 9.87% to 10.92%. To the second approach, almost all the time in GPU was used to process the FoF (from 97% to 99%) since it executed entirely in GPU, taking total advantage of the device.

- **API calls:** In here, to both approaches most of the time was dedicated to the management of the host and device data stream during parallelization. In the first version, all the files spent around 96% of CPU threads blocking time waiting until the stream finished all the tasks, while in the second version it was spent 72.87% for the first input file, 87.04% for the second and 96.44% to the third. In the second approach 48.46% for file 1, 87.93% for file two and 96.33% for file 3, being the retain of the primary context on the device the second most costly activity.

- **OpenACC directives:** To the first approach, most of the time spent here was waiting to enter and to leave the parallel region set in the innermost loop. The larger the input file the longer the time spent to exit the parallel regions, while the waiting time to enter was similar to all input files. However, using `update` directives in the second version, the waiting time to enter the parallel region decrease, with 17.64% for the first time compared with the 26.64% in the first version. To the second approach, the time was mostly spent in waiting to generate and after to enter the parallel region, with 80.99% of waiting to the file 1 and 98.38% to the file 3.

6 Conclusions

This paper presented a new proposal to optimize the serial $O(N^2)$ Friends-of-Friends algorithm using parallelization for hybrid computing with the OpenACC standard. Although OpenACC abstracts implementation details, such as interactions between host and GPU, it is a simple and portable tool, highly recommended for initial investigations.

We presented two approaches to take advantage of multicore CPUs and many-core GPUs. The first one used minimal code modification, using only OpenACC compiling directives. The second brought a new idea, running the algorithm sequentially but in many GPU kernels, acting in parallel to group chunks of particles post-processed in CPU.

The results presented showed that it is possible to make great use of the GPU for the processing of FoF with deeper changes in the code, demonstrating expressive performance improvement compared to the serial version and the first approach with OpenACC.

Acknowledgments. The authors gratefully acknowledge financial support from the CNPq, Brazilian agency for research support, in partnership with the National Institute for Space Research (INPE), who granted the Scientific Initiation scholarship to the first author.

References

[1] L. Berwian, E. T. Zancanaro, D. J. Cardoso, A. S. Charo, R. S. da Rocha Ruiz, and H. F. de Campos Velho. Comparao de estratgias de paralelizao de um algoritmo friends-of-friends com openmp. In *Anais da XVII Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul*, pages 279 – 252, 2017.

[2] C. A. Caretta, R. R. Rosa, H. F. de Campos Velho, F. M. Ramos, and M. Makler. Evidence of turbulence-like universality in the formation of galaxy-sized dark matter haloes. *Astronomy & Astrophysics*, 487(2):445–451, 2008.

[3] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[4] M. Duarte and G. Mamon. How well does the friends-of-friends algorithm recover group properties from distance- and luminosity-limited

galaxy catalogs? *Monthly Notices of the Royal Astronomical Society*, 440, 01 2014.

[5] J. P. Huchra and M. J. Geller. Groups of galaxies. I - Nearby groups. *Astrophysical Journal*, 257:423–437, June 1982.

[6] C. Lacey and S. Cole. Merger rates in hierarchical models of galaxy formation - part two - comparison with n-body simulations. *Monthly Notices of The Royal Astronomical Society - MON NOTIC ROY ASTRON SOC*, 271, 12 1994.

[7] M. S. Madsen. In *The Dynamic Cosmos - Exploring the Physical Evolution of the Universe*, New York, NY, USA, 1996. Chapman & Hall.

[8] S. More, A. V. Kravtsov, N. Dalal, and S. Gottlber. The overdensity and masses of the friends-of-friends halos and universality of halo mass function. *Astrophysical Journal Supplement Series - ASTROPHYS J SUPPL SER*, 195, 02 2011.

[9] OpenACC. The OpenACC application program interface, 2015. Available: http://www.openacc.org/sites/default/files/OpenACC_2pt5.pdf.

[10] R. S. d. R. Ruiz. *Turbulência em cosmologia: análise de dados simulados e observacionais usando computação de alto desempenho*. PhD thesis, Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011-05-26 2011.

[11] R. S. R. Ruiz, H. F. Campos Velho, A. Caretta, C., S. Charão A., and P. Souto R. Grid Environment for Turbulent Dynamics in Cosmology. *Journal of Computacional Interdisciplinary Sciences*, 2:87, 2011.

[12] R. S. R. Ruiz, H. F. Campos Velho, and C. A. Caretta. Parallel algorithm friends-of-friends to identify galaxies and cluster of galaxies for dark matter halos. In *Proceedings...* Workshop dos Cursos de Computação Aplicada do INPE, 9. (WORCAP)., Instituto Nacional de Pesquisas Espaciais (INPE), 2009.

[13] E. C. Vasconcellos, R. R. de Carvalho, R. R. Gal, F. L. LaBarbera, H. V. Capelato, H. F. Campos Velho, M. Trevisan, and R. S. R. Ruiz. Decision Tree Classifiers for Star/Galaxy Separation. *Astronomical Journal*, 141:189, June 2011.