Application of GPUs for the calculation of two point correlation functions in cosmology

Rafael Ponce, Miguel Cárdenas-Montes, Juan José Rodriguez-Vázquez, Eusebio Sánchez, Ignacio Sevilla
Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT), Madrid, Spain

In this work, we have explored the advantages and drawbacks of using GPUs instead of CPUs in the calculation of a standard 2-point correlation function (2pcf) algorithm, which is useful for the analysis of Large Scale Structure of galaxies (LSS). Taking into account the huge volume of data foreseen in our experiment, our main goal has been to accelerate significantly the analysis codes. We find that GPUs offer a 100-fold increase in speed with respect to a single CPU without a significant deviation in the results. For comparison's sake, a MPI version was developed as well. Some issues, like code implementation, which arise from using this option are discussed.

The two-point correlation function in cosmology

The 2pcf is a simple statistic that quantifies the clustering of a given distribution of objects. In studies of the large scale structure of the Universe, it is an important tool containing information about the matter clustering and the Universe evolution at different cosmological epochs. A classical application of this statistic is the galaxy-galaxy correlation function to find constraints on the matter density parameter Ω_m or the location of the baryonic acoustic oscillation peak.

Definition and estimation

The 2pcf measures the excess probability of finding a couple of galaxies separated by spatial distance r or angular distance θ with respect to the probability of finding a couple of galaxies separated by the same distance or angle in a random and uniform distribution. In this work we have used the angular version of the correlation function w(θ) though results are extendible to the 3-dimensional variant as well.

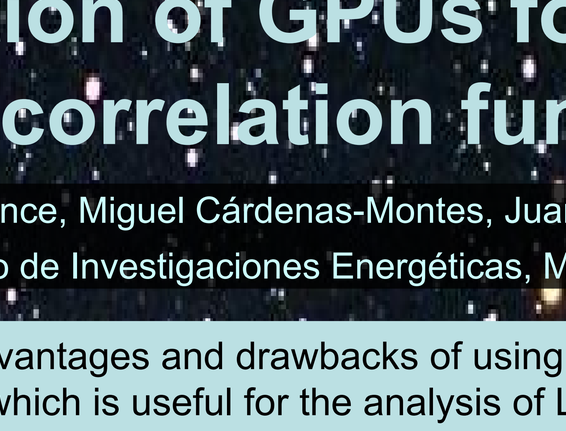

Fig. 1: Comparison between two angular correlation functions, the red one was calculated with CPU and the green one with GPU.

Landy & Szalay have found an estimator with minimum variance which is the standard one used in cosmological analyses:

$$\omega(\theta) = 1 + \left(\frac{N_{rd}}{N_{gal}}\right)^2 \frac{DD(\theta)}{RR(\theta)} - 2\left(\frac{N_{rd}}{N_{gal}}\right) \frac{DR(\theta)}{RR(\theta)} \qquad (1)$$

where Ngal is the number of galaxies in a real catalog, Nrd is de number of galaxies in a random catalog, DD(θ) is the number of pairs separated by an angular distance θ in the real catalog, RR(θ) is the number of pairs separated by an angular distance θ in the random catalog and DR(θ) is the number of pairs separated by an angular distance θ in the real catalog with respect to the random catalog.

Key Points in our work

- We try to improve previous work by creating a versatile and clear code able to do the angular correlation function in less time and with no data input limitations
- Obtaining the angular correlation function implies calculating the arc-cosine. It is the most important bottleneck in the 2pcf calculation
- Intensive use of shared memory: dot product and arc-cosine were fully implemented in this type of memory
- Due to the multithreaded nature of the kernel, we have used atomic functions to avoid that two or more threads modify the same angle bin simultaneously when creating the histogram of Eq. 1
- To accelerate the calculation we do an intelligent bin filling avoiding the use of "if" clause

Difference between CPU code and GPU code

Note that both codes are very similar, but in the GPU code we replaced the second "for" with a while strutcture and the speed increase comes from doing many operations simultaneously. See the marked lines in GPU code.

CPU Code

```
void binning(float *xd, float *yd, float *zd,
     int *ZZ, int numberoflines){
float angle;
int dibin[angleposition];

for (int i=0; i< numberoflines; i++){
  for (int j=0; j< numberoflines; j++){
    angle = xd[i]*xd[j] + yd[i]*yd[j] + zd[i]*zd[j];
    if(angle>1.) angle=1.;
    angle = acosf(angle) * 180./pi;
    position = int((angle)*binsperdegree);
    if(dibin[position]<binsperdegree*totaldegrees);
       dibin[position] = dibin[position]+1;
    }
  }
}
```

GPU Code

```
__global__ void binning(float *xd, float *yd, float *zd,
int *ZZ, int numberoflines){
__shared__ float angle[threadsperblock];
__shared__ int dibin[threadsperblock];
__shared__ int temp[threadsperblock];
int cacheIndex = threadIdx.x;
temp[cacheIndex]=0;
for (int i=0; i< numberoflines; i++){
 int dim_idx = blockIdx.x * blockDim.x + threadIdx.x;
 while (dim_idx < numberoflines){
   angle[cacheIndex] = xd[i]*xd[dim_idx] + yd[i]*yd[dim_idx] +
                       zd[i]*zd[dim_idx];
   __syncthreads();
   if(angle[cacheIndex]>1.) angle[cacheIndex]=1.;
   .
   .
   .
 dim_idx += blockDim.x * gridDim.x; } }
atomicAdd( &ZZ[threadIdx.x] , temp[threadIdx.x]); }
```

Our GPU code has been tested in three different GPU models and we have got impressive execution time results. (See table below)

Input file lines	CPU time	GPU time (GTX 295)	GPU time (C 1060)	GPU time (C 2050)
$0.43 \cdot 10^6$	10 hours	5.01 min.	4.85 min.	3.65 min.
$0.86 \cdot 10^6$	40 hours	19.97 min.	19.33 min.	14.60 min.
$1.00 \cdot 10^6$	55 hours	26.83 min.	25.97 min.	19.55 min.
$1.29 \cdot 10^6$	90 hours	44.66 min.	43.23 min.	32.85 min.
$1.72 \cdot 10^6$	160 hours	———	1.29 hours	58.43 min.
$3.45 \cdot 10^6$	644 hours	———	5.21 hours	3.92 hours
$6.89 \cdot 10^6$	2560 hours	———	20.70 hours	15.58 hours

With the newest GPU we have got a sensitive speedup of 164 fold.

GPU MPI Comparison

We also did an MPI version, in the next table we have the MPI speedups we have obtained. Note that the GTX295 GPU card has a perfomance similar to a cluster with a number of cores in the range from 64 and 128.

Cores	Speedup
2	2.035 ± 0.014
4	3.748 ± 0.047
8	7.499 ± 0.084
16	15.046 ± 0.562
32	29.667 ± 0.455
64	58.716 ± 0.965
128	115.713 ± 2.423
256	228.796 ± 14.210
512	333.090 ± 12.301

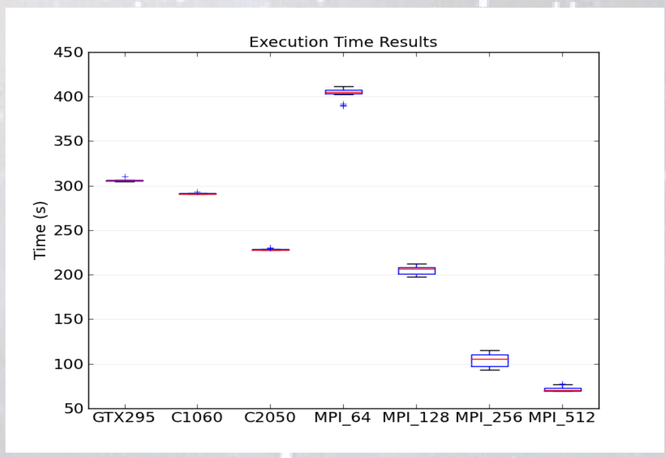

Fig. 2: Comparison between MPI time and different GPUs time.

In the Fig. 2 we have a comparison between our time in MPI with 128, 256 and 512 nodes and different GPUs.

Conclusions

We have developed an implementation of the Landy-Szalay two-point correlation function in CUDA to make use of the power GPUs have to offer in terms of parallelization. The speed-up with respect to an 8-core CPU is 120-fold using the same algorithm. (With respect to an implementation of k-trees in CPUs we obtain an increase of 3.7-fold). Several MPI options have been explored and single GPUs are only surpassed by the usage of more than 100 nodes (the MPI implementation being at a much higher cost, however). Some options to be explored remain, such as full-blown multi-GPU implementation, coding the k-trees or extending the work to higher order correlation functions, for other types of cosmological analyses such as understanding non-Gaussianities in the primordial perturbations.

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN
Ciemat
Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas
THE DARK ENERGY SURVEY