GPGPU Integration and GPGPU User Application Support

Miguel Cárdenas Montes
on behalf of CIEMAT colleagues

Introduction

Motivation for this Presentation

To show some success cases in GPU computing, including drawbacks and benefits.

Introduction

Motivation for this Presentation

To show some success cases in GPU computing, including drawbacks and benefits.

Motivation for GPU Computing

To analyse or simulate the high volume of data in Cosmology in a reasonable processing time.

Cosmology-Astronomy Facing Data Challenge

Introduction

Why GPU Computing?

- GPU has a good relation FLOPS per watt.
- It is not expensive, at least not more than the budget for computing in cosmology.
- It can cover the computational need of a mid-sized group.
- Two entries (2nd Titan and 6th Piz Daint) in the top 10 of Top500, and 10 entries in Green500.
- It is a hot topic (publications).

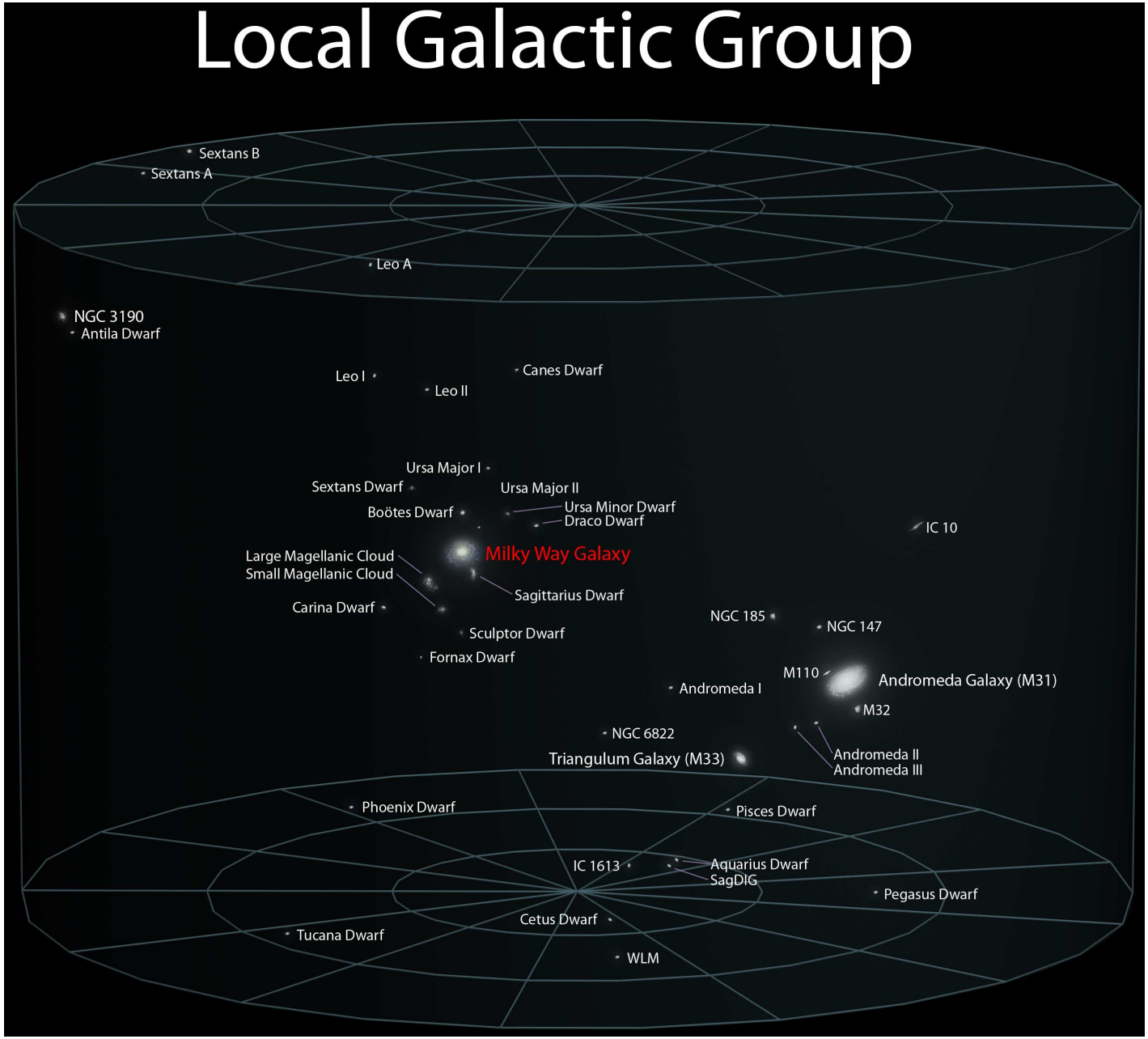
Introduction

This is not:

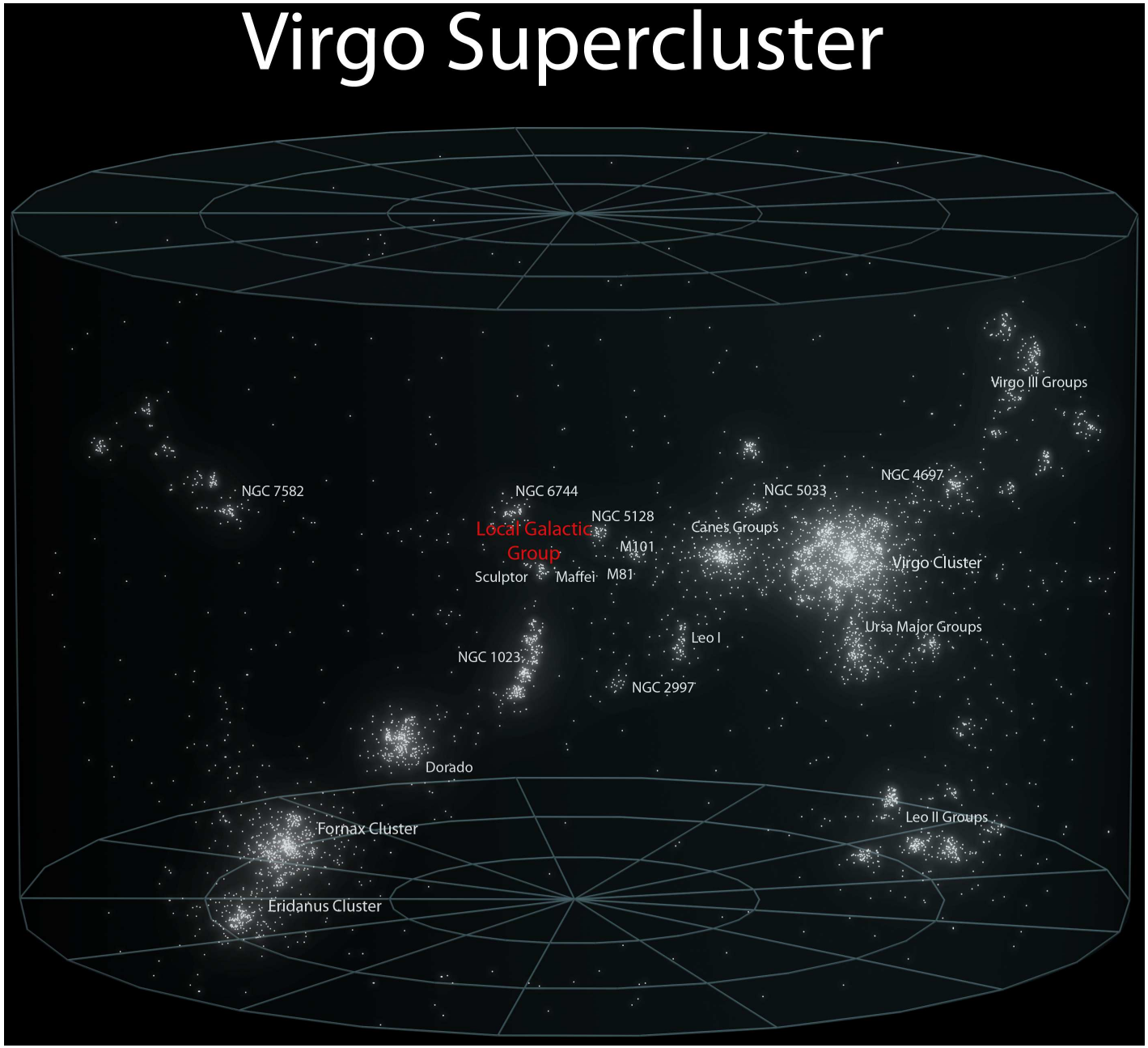
- A presentation about the NVIDIA hardware (I am not NVIDIA staff),
- nor a CUDA training,
- but our own experience when trying to solving scientific problems with GPU-CUDA.

First Project: The Two-Point Angular Correlation Function

Local Galactic Group



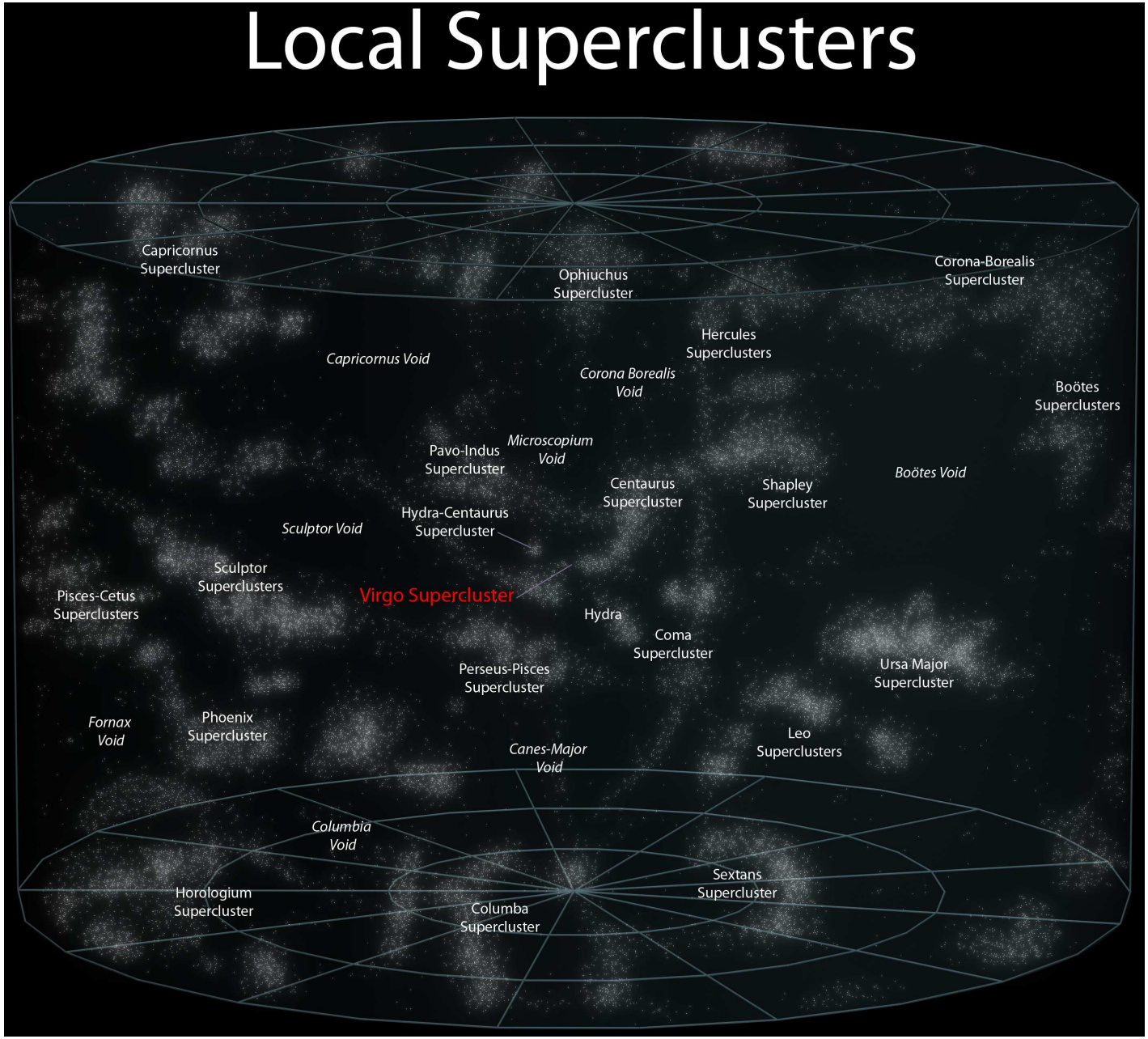
Virgo Supercluster

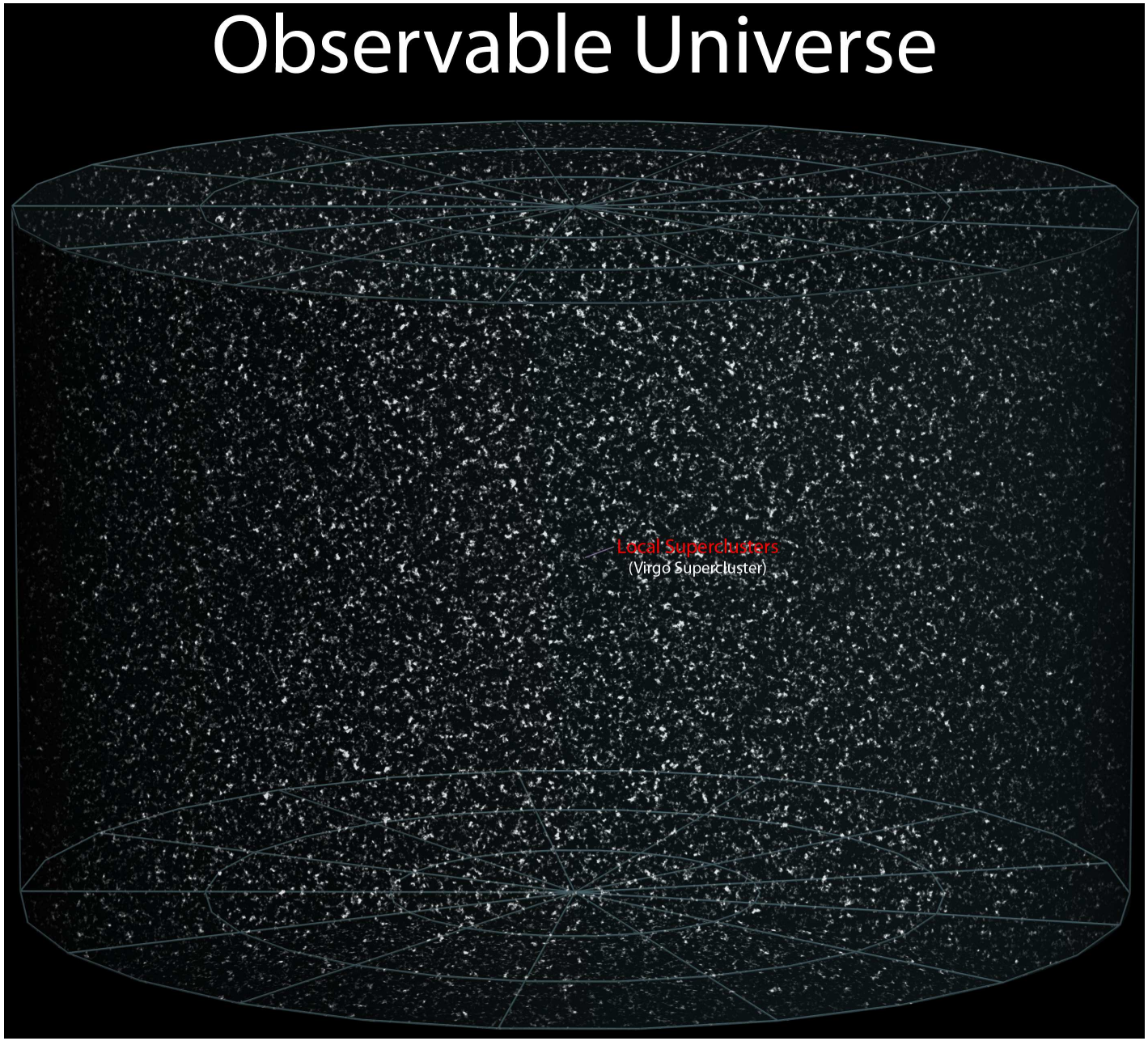


Large-scale structure of the Universe:

- Non-linear structures for small scales (less than 10 Mpc).
- Cluster and super-cluster of galaxies are consequences of this non-linearity.
- On large scales became nearly linear.

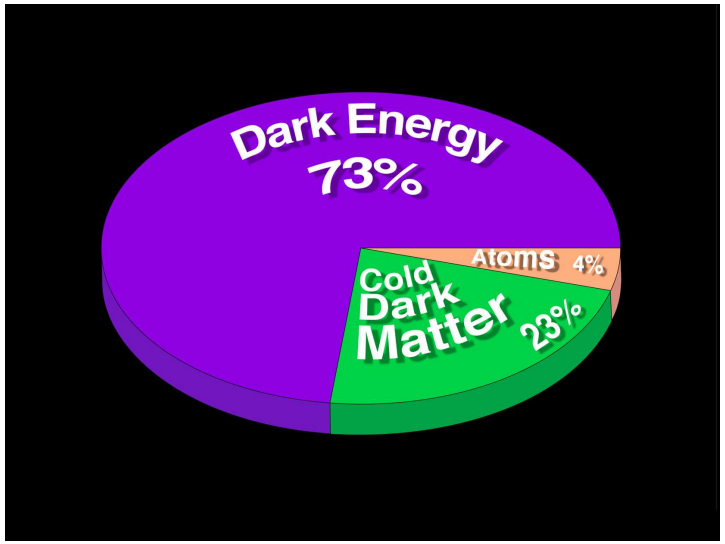
Local Superclusters




Observable Universe
Local Superclusters
(Virgo Supercluster)

2PACF

- The distribution of galaxies in the universe is one of the most important probes for cosmological models.
- The 2PACF is a test to measure this distribution.
- The calculation of the 2PACF is computationally demanding, $O(N^2)$. CPU implementation takes around 8 hours for a sample of 430K galaxies.



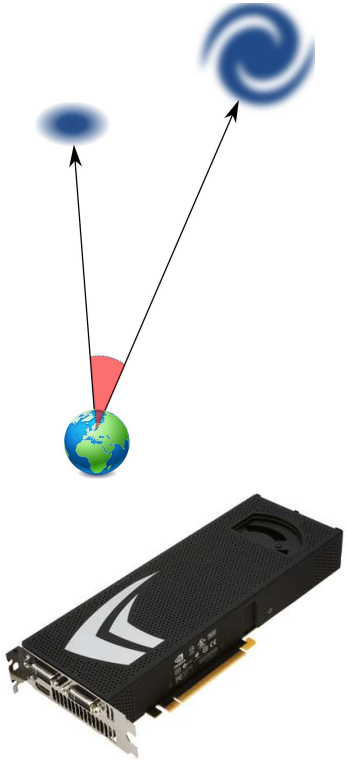
Data Volume

Statistical Analysis from 10^6 to 10^{10} galaxies in the near future.

2PACF

Our goals:

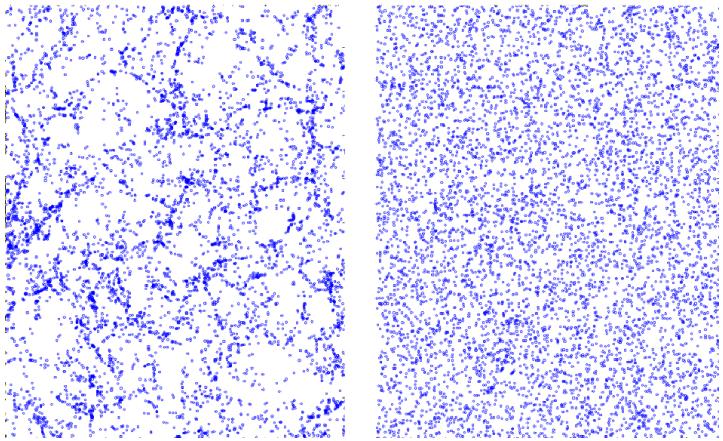
- Fast Implementation of Two-Point Angular Correlation Function. CPU implementation takes around 8 hours for a sample of 430K galaxies, and $O(N^2)$!
- An implementation able to deal with very large surveys, $> 10^6$ galaxies.
- Low budget.



2PACF

The 2PACF, $\omega(\theta)$, is a measure of the excess or lack of probability of finding a pair of galaxies under a certain angle with respect to a random distribution. In general, estimators 2PACF are built by combining the following quantities:

- DD(θ) is the number of pairs of galaxies for a given angle θ chosen from the data catalogue (D).
- RR(θ) equivalent on the random catalogue (R).
- DR(θ) one galaxies from each catalogue.



2PACF

$$\omega(\theta) = 1 + \left(\frac{N_{random}}{N_{real}}\right)^2 \cdot \frac{DD(\theta)}{RR(\theta)} - 2 \cdot \left(\frac{N_{random}}{N_{real}}\right) \cdot \frac{DR(\theta)}{RR(\theta)}$$

- If $\omega(\theta) > 0$ more frequently found at angular separation of θ than expected for a randomly distributed.
- If $\omega(\theta) < 0$ lack of galaxies in this particular θ.
- $\omega(\theta) = 0$ means purely random distribution.
- The sample 430K galaxies.

GPU Implementation

- Intense use of *shared memory* for intermediate calculations and histogram construction.
- Sub-histograms allocated on *shared memory*.
- Atomic operations on shared memory also required.
- Coalesced-access to global memory. Data ordered by coordinates, not by galaxies.

Execution Time

Table: Execution time and speedup for several implementations.

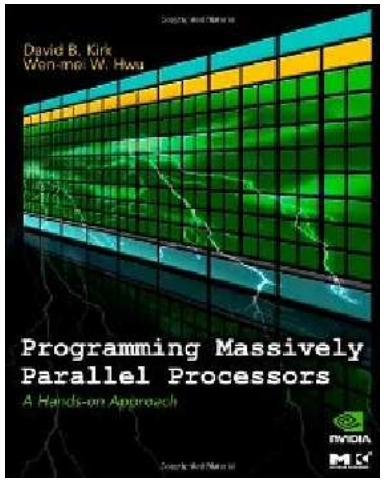
Implementation	Execution time (s)	Speedup
CPU	35,186.327	
OpenMP (8 cores+hyperthreading)	3,326.363	10
GPU (GTX295)	305.570	115
MPI 64 cores	403.937	87
MPI 128 cores	205.008	171

Cárdenas-Montes, Miguel, et al.: **New Computational Developments in Cosmology**, Ibergrid, 101-112, 2012

Ponce, Rafael, et al.: **Application of GPUs for the Calculation of Two Point Correlation Functions in Cosmology**, Astronomical Data Analysis Software and Systems XXI, 461, 73-76, 2012

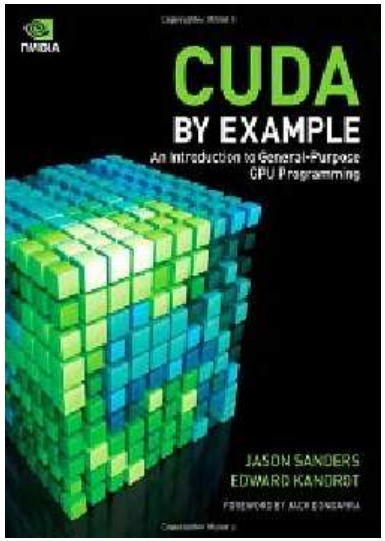
Drawbacks

- No expert in programming.
- No didactic books. No training events. Lack of information in web.
- No atomicAdd() for float, only integer. This is essential for some other analyses in Cosmology, specially in correlation functions.
- Single precision.



More Optimization - New Card

- New card, C2075. A bit extra budget!
  - AtomicAdd() in float.
  - Double precision.
- New books (very didactic).



More Optimization - New Card

- Initial implementation:
  - Coalesced-access to data in global memory.
  - Intense use of *shared memory* for intermediate calculations and histogram construction.
  - Atomic operations on shared memory also required: main bottleneck.
- Improvements:
  - Register for data frequently reused, incrementing the data locality.
  - Incrementation of the occupancy by reducing the number of variable and, therefore, forcing to recalculate.
  - When possible reducing branching by replacing if-conditionals by min-functions.

GPU Implementation

Implementation		
Single Precision, Compute Capability 1.2, GTX295	Original Code	299,566.4±15.3 ms
	Positive Strategies	275,303.8±17.6 ms
	Reduction	**24,262.6**
	Speedup	**1.09**
Single Precision, Compute Capability 2.0, C2075	Original Code	314,346.8±199.6 ms
	Positive Strategies	269,969.5±69.8 ms
	Reduction	**44,377.3**
	Speedup	**1.16**
Double Precision, Compute Capability 2.0, C2075	Original Code	452,097.3±287.2 ms
	Positive Strategies	438,934.0±132.2 ms
	Reduction	**13,163.4**
	Speedup	**1.03**

Outcome

- Satisfactory level of performance to deal with larger files $> 10^6$ galaxies even more.
- Code adopted in international collaborations: Physics of the Accelerating Universe and Dark Energy Survey.
- Other problem proposed.

Cárdenas-Montes, Miguel, et al.: **Calculation of Two-Point Angular Correlation Function: Implementations on Many-Core and Multicore Processors**, Ibergrid, Editorial Universitat Politecnica de Valencia, 203-214, 2013

A Bit of Hardware

TESLA C2075 Fermi 2.0

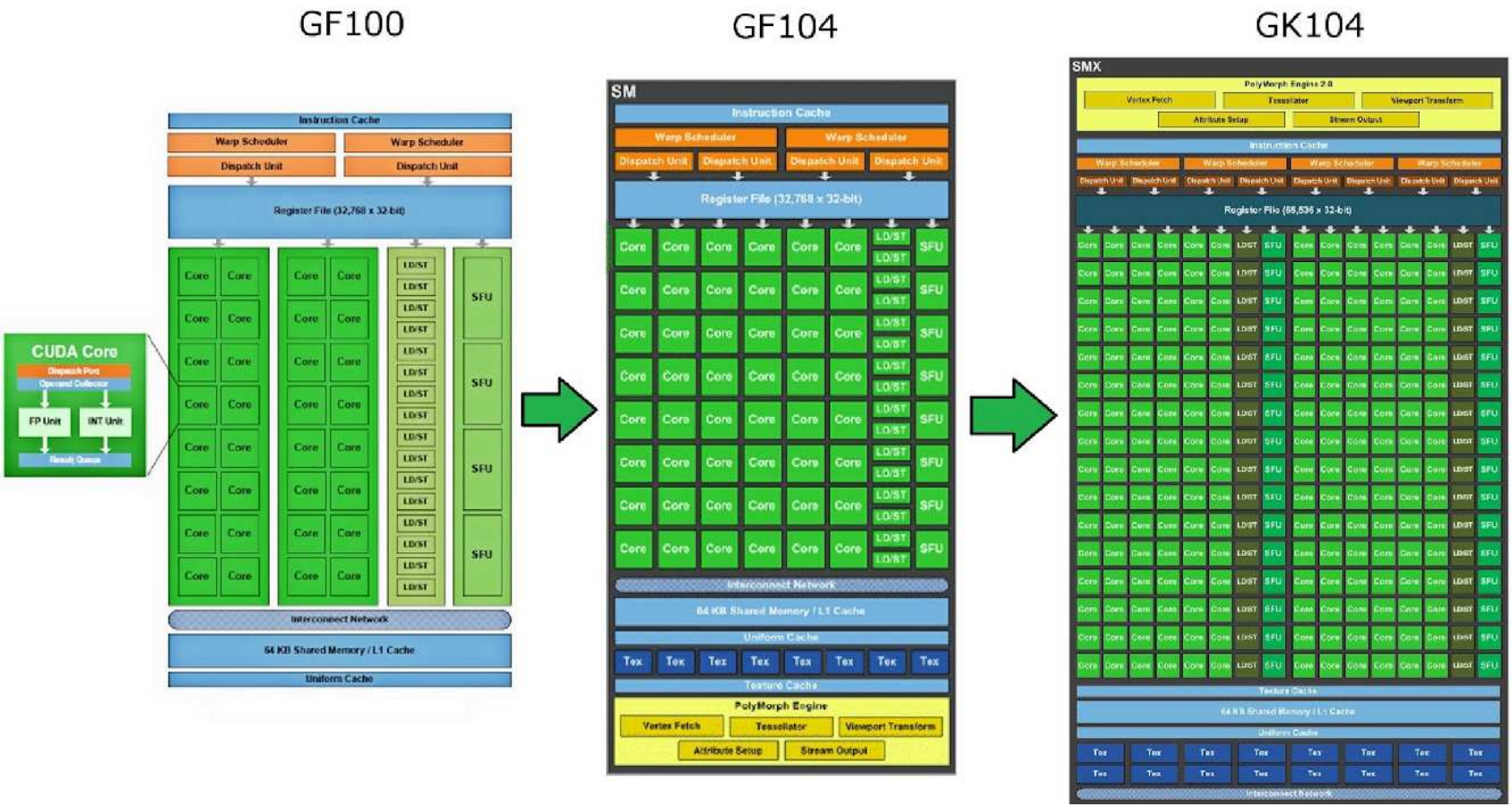
- 14 streaming multiprocessors (SM);
  32 cuda cores / SM = total of 448 cuda cores;
  32 threads / core = total of 14336 threads
- clock rate: 1.15GHz
- 6 GB memory (global memory)
- 64KB on-chip memory / SM and 768KB L2 cache (shared by all SMs)

XEON E7-8870

- 10 cores / 20 threads
- 2.40GHz
- 30MB cache

i7-3970X

- 6 cores / 12 threads
- 3.50GHz
- 15MB cache


GF100
GF104
GK104
CUDA Core
Dispatch Port
Operand Collector
FP Unit
INT Unit
Result Queue
Instruction Cache
Warp Scheduler
Dispatch Unit
Register File (32,768 x 32-bit)
Core
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
Uniform Cache
SM
Tex
Texture Cache
PolyMorph Engine
Vertex Fetch
Tessellator
Viewport Transform
Attribute Setup
Stream Output
SMX
PolyMorph Engine 2.0
Register File (65,536 x 32-bit)

Shear-Shear Correlation Function

Shear-Shear Correlation

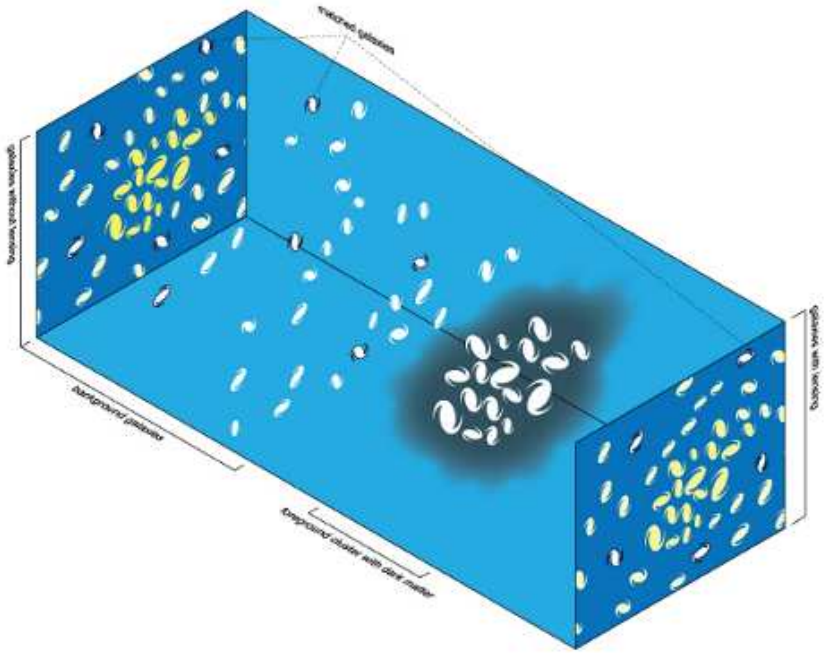
The physics of the problem:

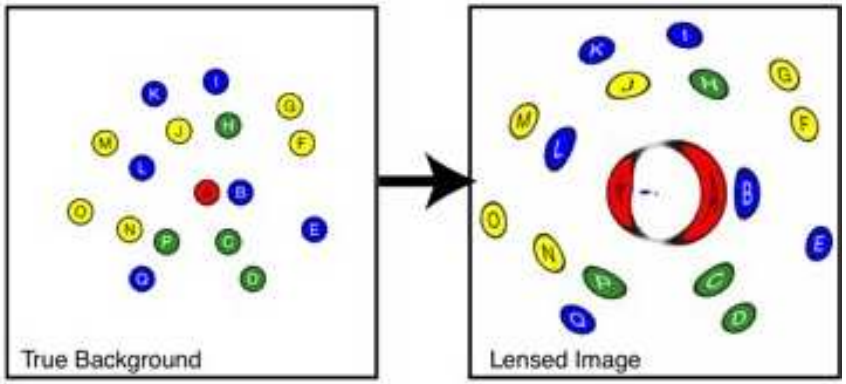
- Light rays are deflected when travelling through a gravitational potential, this phenomenon is known as gravitational lensing.
- This causes the observed shapes of distant galaxies to be very slightly distorted by the intervening matter in the Universe, as their light travels towards us. This distortion is called *cosmic shear*.
- By measuring this component it is possible to derive the properties of the mass distribution causing the distortion.

Shear-Shear Correlation

- In the past this analysis has been burden by instrumental errors, reduced volume of data and the available observational data span small regions of the sky.
- Data volume from tens of thousands to tens of millions.
- In this work an observational data set of 1 million of galaxies (Canada–France–Hawaii Lensing Survey, CFHTLenS).
- Shear-shear has a higher computational intensity than 2PACF (more calculation for each pair of galaxies).

Shear-Shear Correlation





Shear-Shear Correlation

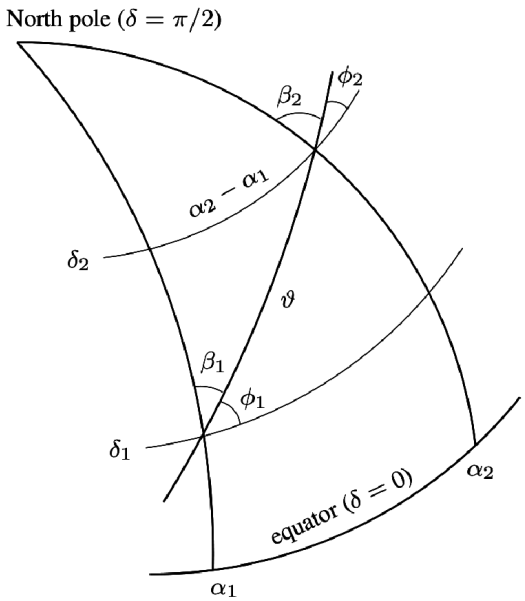
$$\cos\Phi_1 = \frac{\sin(\alpha_2-\alpha_1)\cos\delta_2}{\sin\theta}$$

$$\sin\Phi_1 = \frac{\cos\delta_2\sin\delta_1 - \sin\delta_2\cos\delta_1\cos(\alpha_2-\alpha_1)}{\sin\theta}$$

$$\xi_+(\theta) = \frac{\sum_{ij} w_i w_j (\gamma_t(\theta_i)\cdot\gamma_t(\theta_j) + \gamma_\times(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j}$$

$$\xi_-(\theta) = \frac{\sum_{ij} w_i w_j (\gamma_t(\theta_i)\cdot\gamma_t(\theta_j) - \gamma_\times(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j}$$

$$\xi_\times(\theta) = \frac{\sum_{ij} w_i w_j (\gamma_t(\theta_i)\cdot\gamma_\times(\theta_j))}{\sum_{ij} w_i w_j}$$



Global Memory (DRAM) Bandwidth

Ideal

Reality

©Wen-mei W. Hwu and David Kirk/NVIDIA
Barcelona, July 2-6, 2012

3

Global Memory Bandwidth

- Many-core processors have limited off-chip memory access bandwidth compared to peak compute throughput
- Fermi
  - 1 TFLOPS SPFP peak throughput
  - 0.5 TFLOPS DPFP peak throughput
  - 144 GB/s peak off-chip memory access bandwidth
    - 36 G SPFP operands per second
    - 18 G DPFP operands per second
  - To achieve peak throughput, a program must perform 1,000/36 = ~28 arithmetic operations for each operand value fetched from off-chip memory

New Weapons

Skilled Programmers

Scientists convinced

Base Implementation

The best practices are inherited:

- A coalesced pattern access to global memory.
- An intensive use of shared memory to store the results of intermediate operations is implemented.
- The use of registers to store the input data frequently accessed (such as galaxy coordinates and ellipticities).
- Sub-histogram construction on shared memory and final gathering on global memory.

Base Implementation

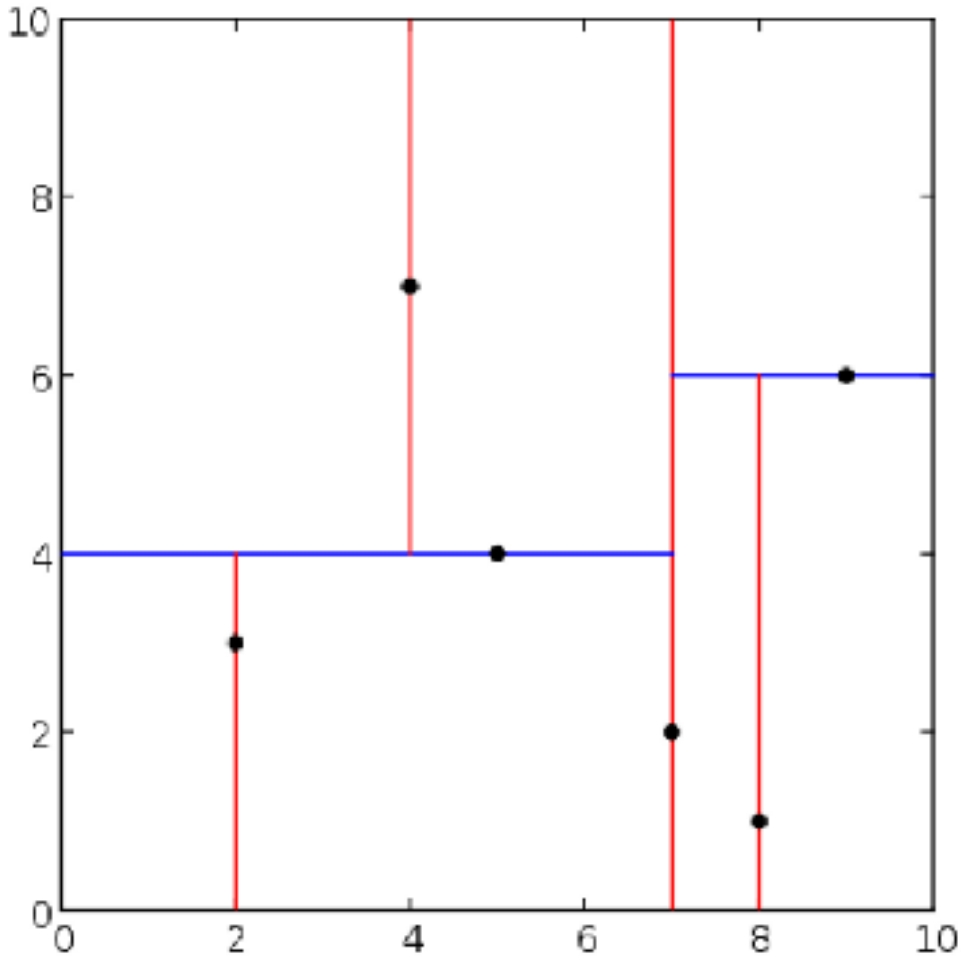
And other best practices are learned now:

- Use of double precision. Very tiny quantities are added in the histograms (in the 2PACF an unit added); and the galaxies are analysed at very small angles.
- AtomicAdd() in float required.
- Explicitly caching global memory into shared memory, L1 cache memory off.

Compute capability

- Compute capability 1.1: Atomic function only in global memory.
- Compute capability 1.2: Atomic function in shared memory.
- Compute capability 1.3: Some functions from single to double precision (IEEE 754-1985).

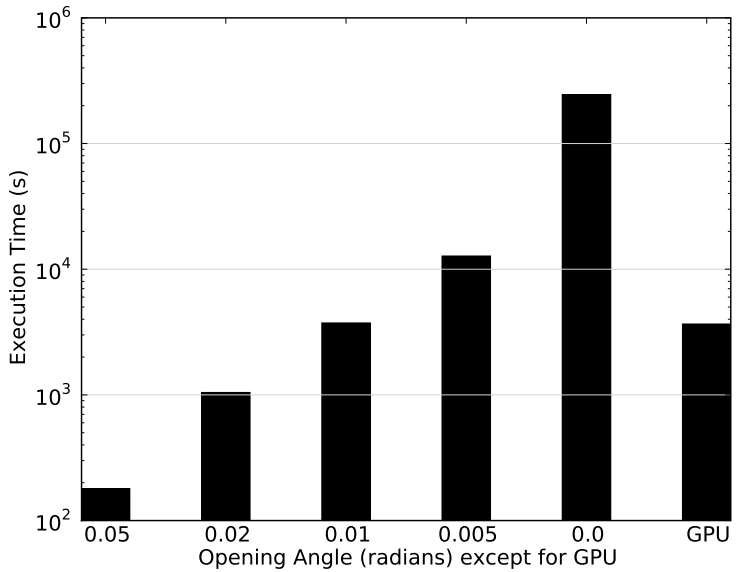
KD-tree



Shear-Shear Correlation

Comparison with the previous state-of-the-art code, ATHENA.

- ATHENA is a sequential code based on kd-trees to reduce the computational complexity of the calculations.
- Based on the parameter termed *Opening Angle, OA*. The smaller OA it is, the fewer approximations makes.



Outcome

Execution time: GPU 3,618.7s vs. ATHENA 247,681s at OA=0.

Cárdenas-Montes, Miguel, et al.: **GPU-Based Shear-Shear Correlation Calculation**, Computer Physics Communications, 185(1):11-18, ISSN: 0010-4655, 2014

Shear-Shear Correlation

Extra optimization applied:

Implementation	Execution Time (s)	Speedup Related to the Baseline	Speedup Related to *ATHENA* OA=0
Baseline	3,618.7		67
Reordered loops	3,243.3	1.12	75
Vectorized	3,184.2	1.14	77

MPI-CUDA Implementation

MPI-CUDA Implementation to deal with tens of million of galaxies (1M in this table):

Nodes	Execution Time	Speedup Related to MPI-CUDA Single-Node	Speedup Related to *ATHENA* OA=0
1	3,325.39		73.4
2	1,672.59	1.99	145.9
4	845.15	3.93	288.7
8	432.24	7.69	564.6
16	225.49	14.75	1082.2

MPI-CUDA Implementation

Outcome

- 15M galaxies in the GPU implementation, the execution time takes 169 hours,
- MPI-CUDA implementation with 16 nodes it takes 11 hours, achieving a speedup of 15.36.

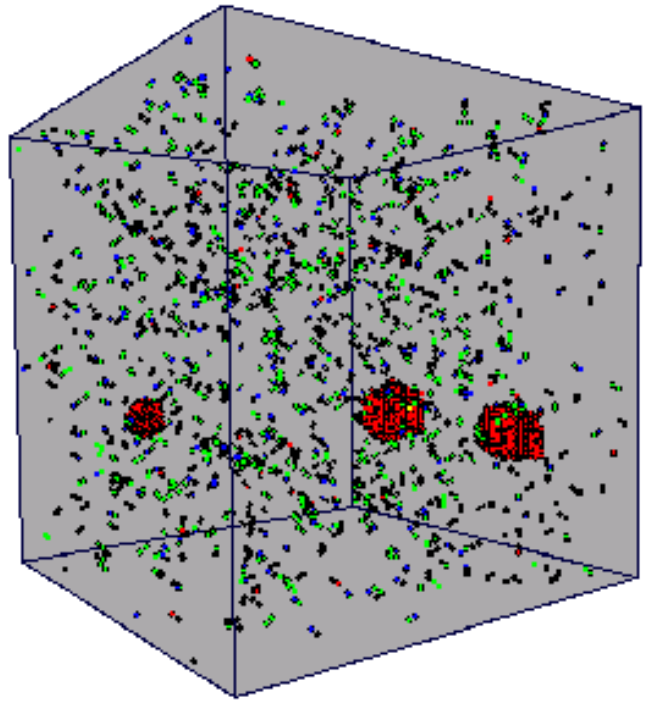
Cárdenas-Montes, Miguel, et al.: **High-Performance Implementations for Shear-Shear Correlation Calculation**. Cluster, IEEE Computer Society, 2014. Submitted

Object Kinetic Monte Carlo

On behalf of:
Christophe Ortiz and Fernando Jiménez

Object Kinetic Monte Carlo

- The kinetic Monte Carlo is a computer simulation method intended to simulate the evolution of a set of objects, given the type of event those objects can perform and the probability for each event to occur.
- Probabilities of events must be given. kMC cannot predict them.



- Case of radiation in solids:
  - Objects: interstitials, vacancies, He, H, clusters...
  - Events: Diffusion jumps, agglomeration, dissociation from clusters, recombination...

Drawback of Sequential Approach

- Only one particle moves during one step.
- If number of defects increases CPU time increases accordingly. Problem if you want to simulate high irradiation dose. Limited to only $\sim 10^5$ particles.
- Very CPU demanding: days - weeks/months.
- In practice limited to low doses and small volumes $\sim$100 nm x 100 nm x 100 nm.
- Too small compared to a grain in a polycrystal $\sim$50 μm.
- OkMC not suited to explore realistic piece of materials.

Object Kinetic Monte Carlo

Our Expectations of GPU approach

- Implement a parallel version of the residence-time algorithm.
- Move thousands of particles during one step.
- Handle millions of particles to simulate high irradiation dose.
- Significantly decrease runtime.
- Investigate evolution of defects in realistic piece of materials.

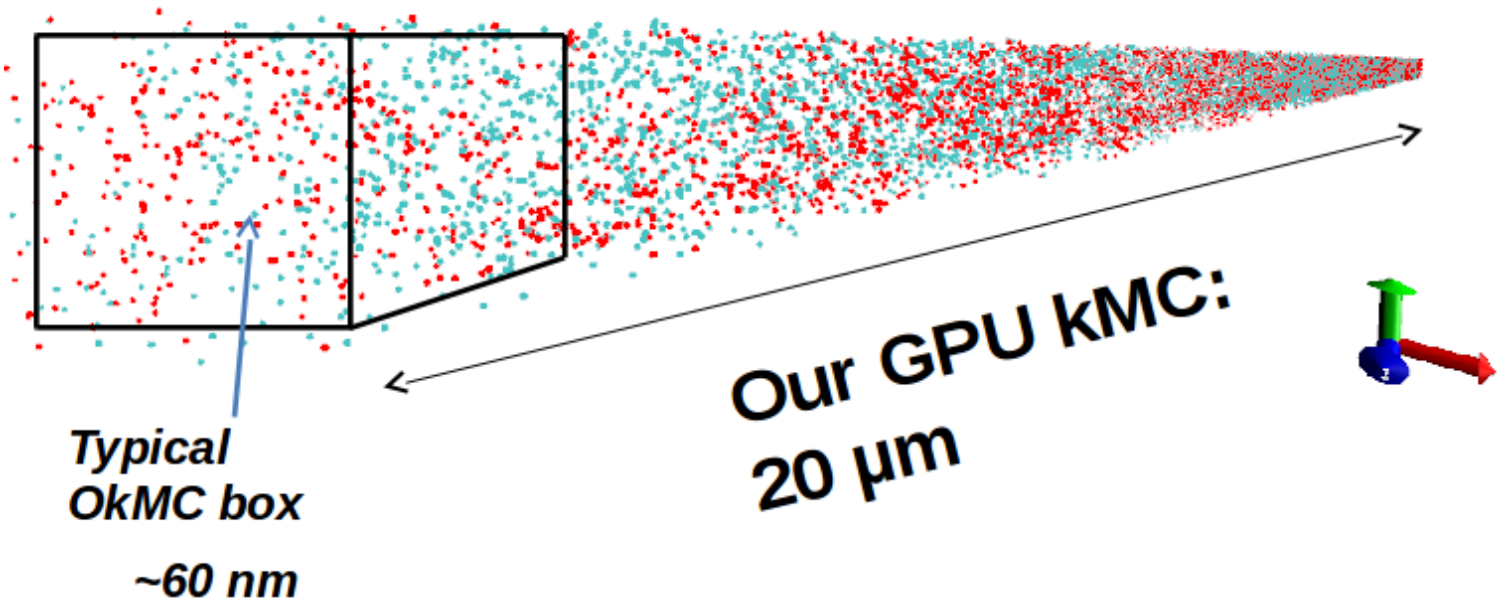
Object Kinetic Monte Carlo

Benefits

- Our GPU-OkMC is able to simulate evolution of $\sim 10^7$ particles, in contrast to $\sim 10^5$ with classical OkMC.
- Only few hours necessary to simulate evolution of millions of particle in realistic conditions, in contrast to days with classical OkMC.

Object Kinetic Monte Carlo

Benefits



- Allows to simulate evolution of defects in a realistic piece of materials ($\sim$20 μm), in contrast to classical OkMC that only allow small simulation boxes $\sim$60 nm.

Pressure Poisson Solver

On behalf of:
Pedro Valero, Alfredo Pinelli and Manuel Prieto

Pressure Poisson Solver

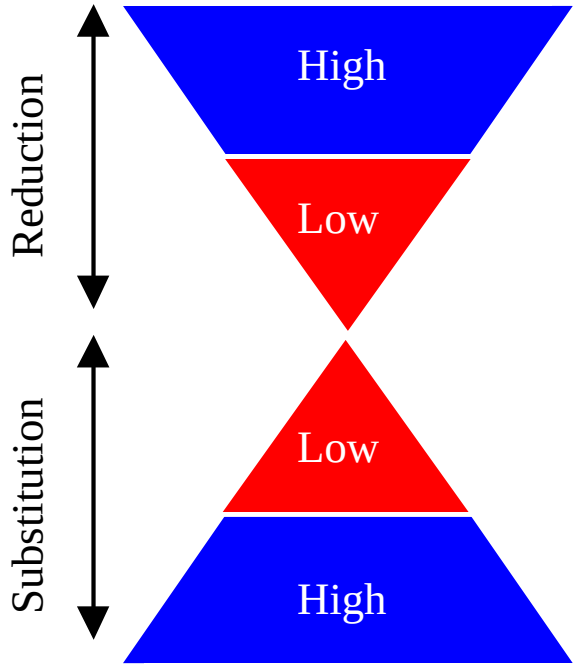
- Main bottleneck in most Navier-Stokes solver.
- Block Tridiagonal Problems for Elliptic problems: Steady, Subsonic, Inviscid, Incompressible flows

Direct Solver

Open Source Fortran Fishpack Library

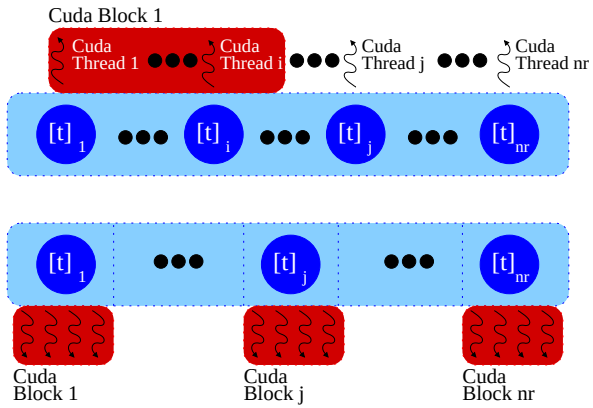
Parallel Characteristics

- Core of the algorithm –>solving of tridiagonal problems
- Dynamic parallelism:
  Reduction –>decreases
  Substitution –>increases

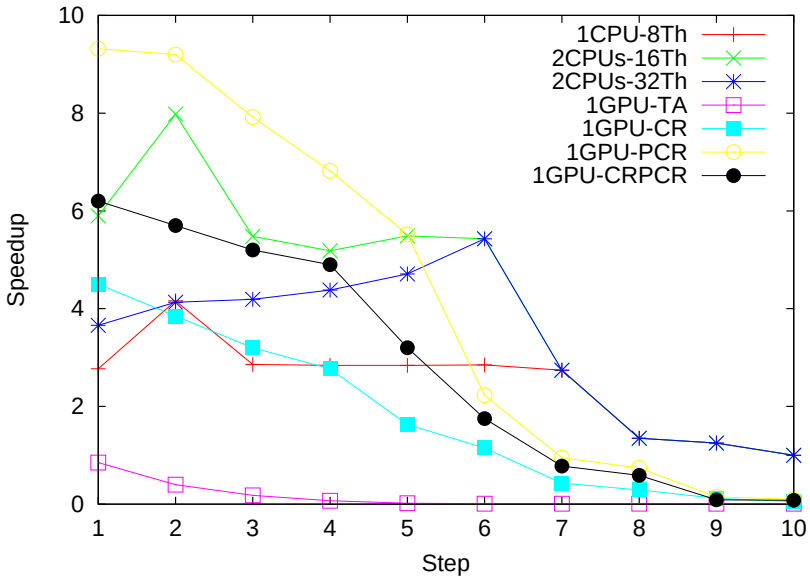


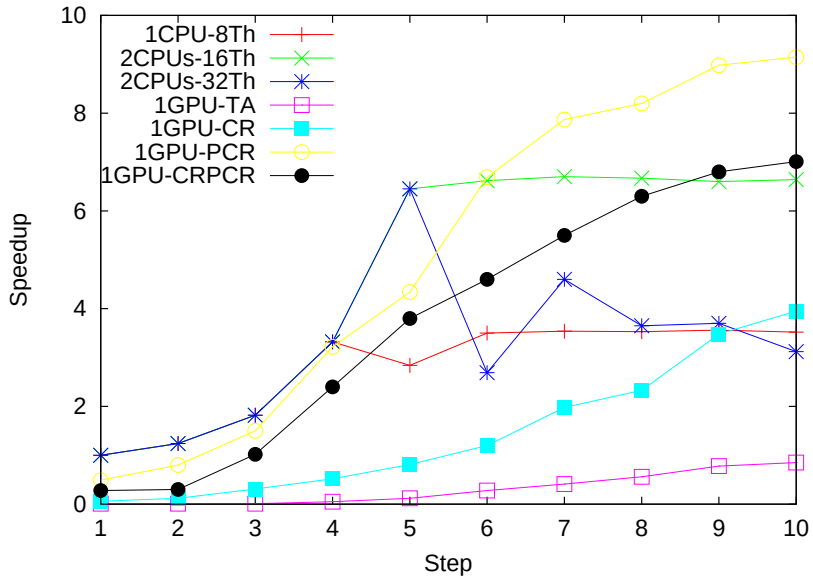
Parallel Characteristics

- Regions with a high parallelism on GPU (fine-grained)
  1 block of threads per term to be solved
  PCR for solving tridiagonal problems
- Regions with a low parallelism on multicore (coarse-grained)
  1 thread solves multiple independent elements
  Thomas algorithm for solving tridiagonal problems



Performance Evaluation

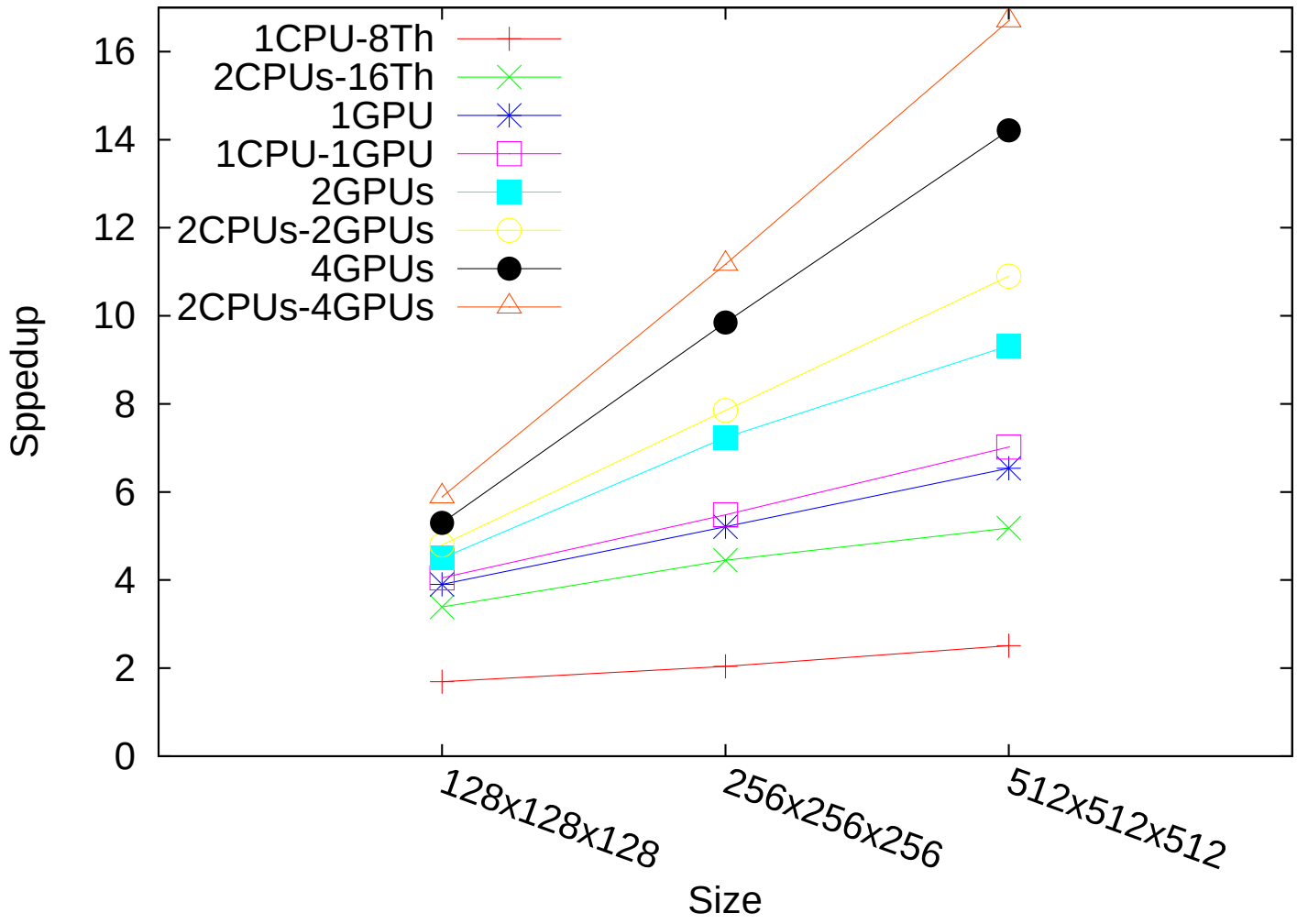




Outcome

Valero-Lara, Pedro, et al.: **Fast finite difference Poisson solvers on heterogeneous architectures**. Computer Physics Communications 185(4): 1265-1272 (2014)

Speedup for 3D

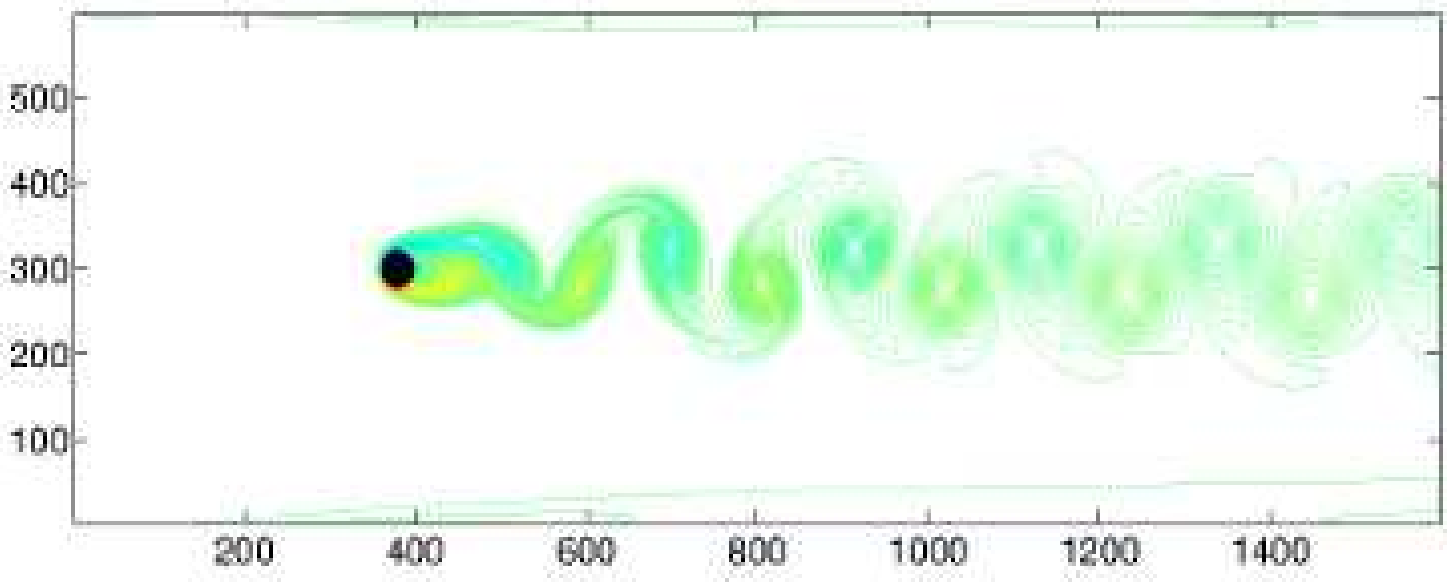


Solid-Fluid Iteration

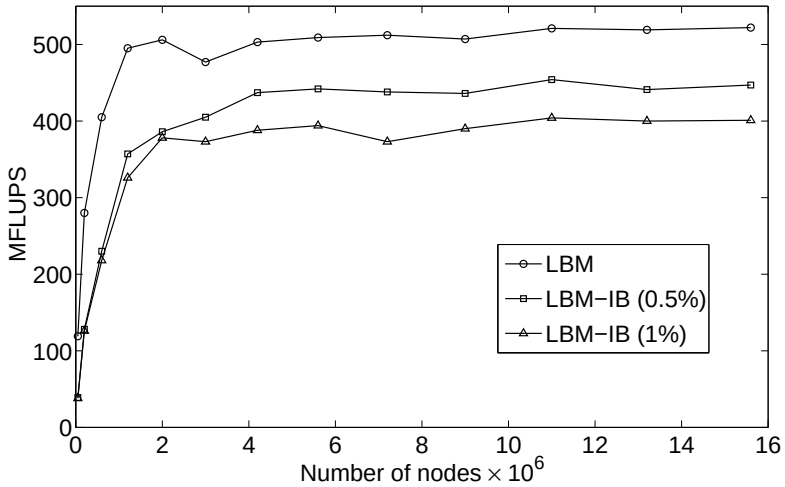
On behalf of:
Pedro Valero, Alfredo Pinelli and Manuel Prieto

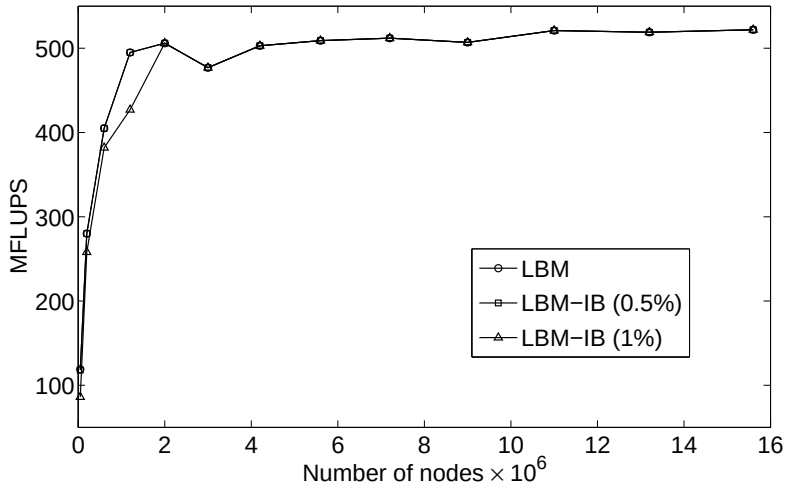
Solid-Fluid Iteration

- High fall in performance dealing with solids within fluid
- The objective: achieving the same performance that pure fluid solvers, minimizing/avoiding the overhead of the computing of the solids contribution.
- Strategy: overlapping GPU computing (fluid) with multicore computing (solids)



Solid-Fluid Iteration





Outcome

Valero-Lara, Pedro, et al.: **Solid-Fluid Iteration through the use of the coupling Lattice-Boltzmann and Immersed-Boundary**. International Conference on Computational Science (2014)

DiVoS:
Dynamic Vortex for Superconductivity
Simulation of a superconductive surface

On behalf of:
Manuel Aurelio Rodríguez

DiVoS

Very simple:

- Mangets located in the edges of rectangles
- Charged particles freely moving inside
- Try to find an equilibrium situation: minimum

but

- Every particle must consider all the rest plus the static magnets: exponential growth
- Full of local minimums (unwanted)

DiVoS

Previous

- Grid exploration of a small solution space to model its behaviour and serve as a reference.

Now

- (Now)Genetic algorithm to explore the solution space.
- Greedy algorithm to improve every solution (move towards minimum).

DiVoS

Implementation:

- PyCuda

Performance

- Over 99.5
- 300X speedup with a single GPU : great success!
- Future –>GPU + MPI

GPU Data Layout of Parallel Evolutionary Algorithms

GPU Data Layout for PEAs

- What data layout allows a faster evaluation of a non-separable function?
- Non-separable functions are frequently used as benchmark functions in PEAs.

Non-Separable Fitness Functions

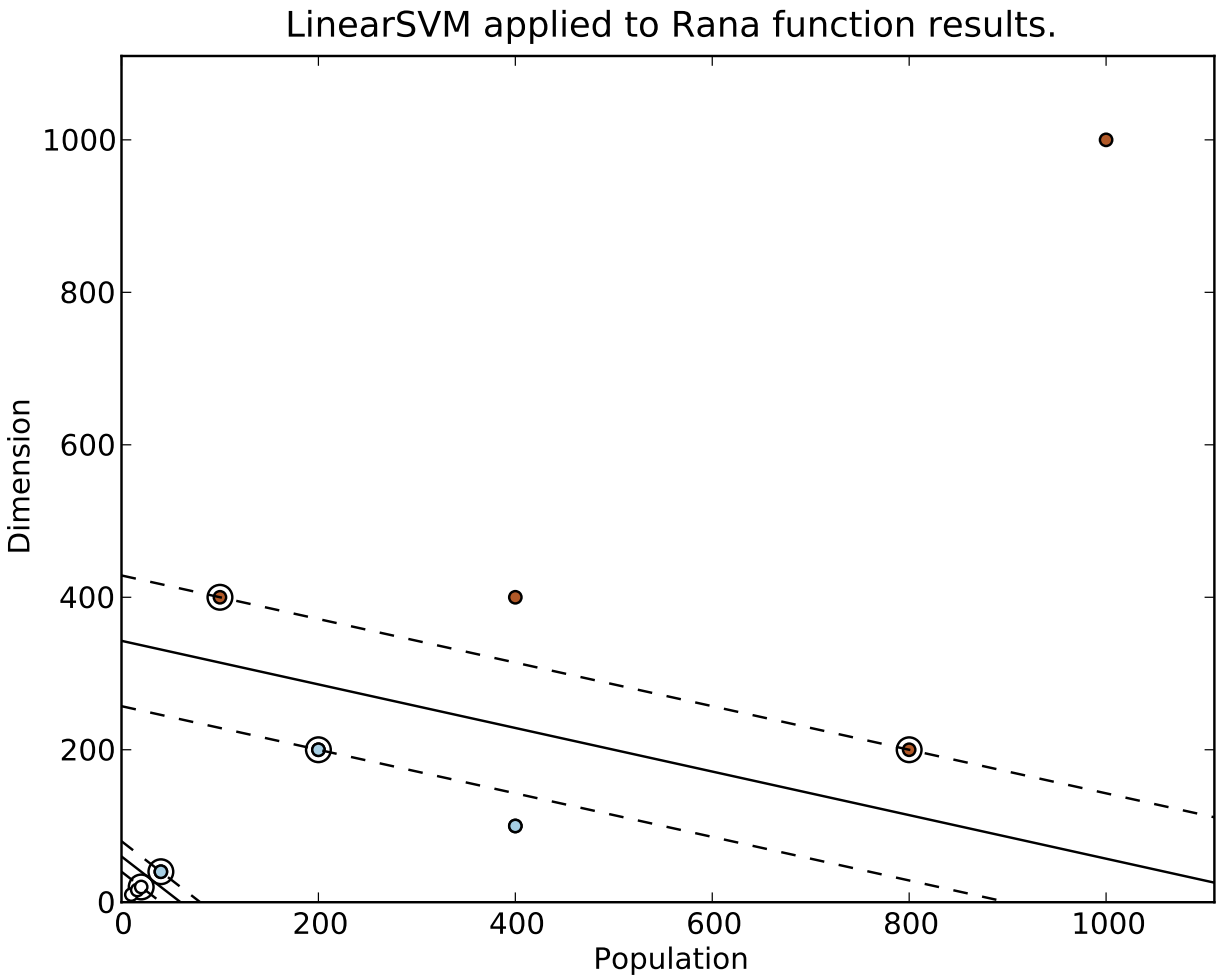
$$f_{Rosenbrock} = \sum_{i=1}^{D-1} 100 \cdot [(x_i^2 - x_{i+1})^2 + (x_i - 1)^2] \quad (1)$$

$$f_{Rana} = \sum_{i=1}^{D-1} (x_{i+1} + 1.0) \cdot cos(t_2) \cdot sin(t_1) \; + \; cos(t_1) \cdot sin(t_2) \cdot x_i$$
$$where\ t_1 = \sqrt{|x_{i+1} + x_i + 1.0|}\ , and\ t_2 = \sqrt{|x_{i+1} - x_i + 1.0|} \quad (2)$$

Data Layouts

- S1 Allocation of one individual per thread on registers.
- S2 Allocation of one individual per thread on shared memory.
- S3 Allocation of one individual per thread-block on share memory with coalesced access to global memory and atomic operations.
- S4 Allocation of one individual per thread on registers with coalesced access to global memory.
- SE And sequential evaluation.

Results for Rana Function



GPU Data Layout for PEAs

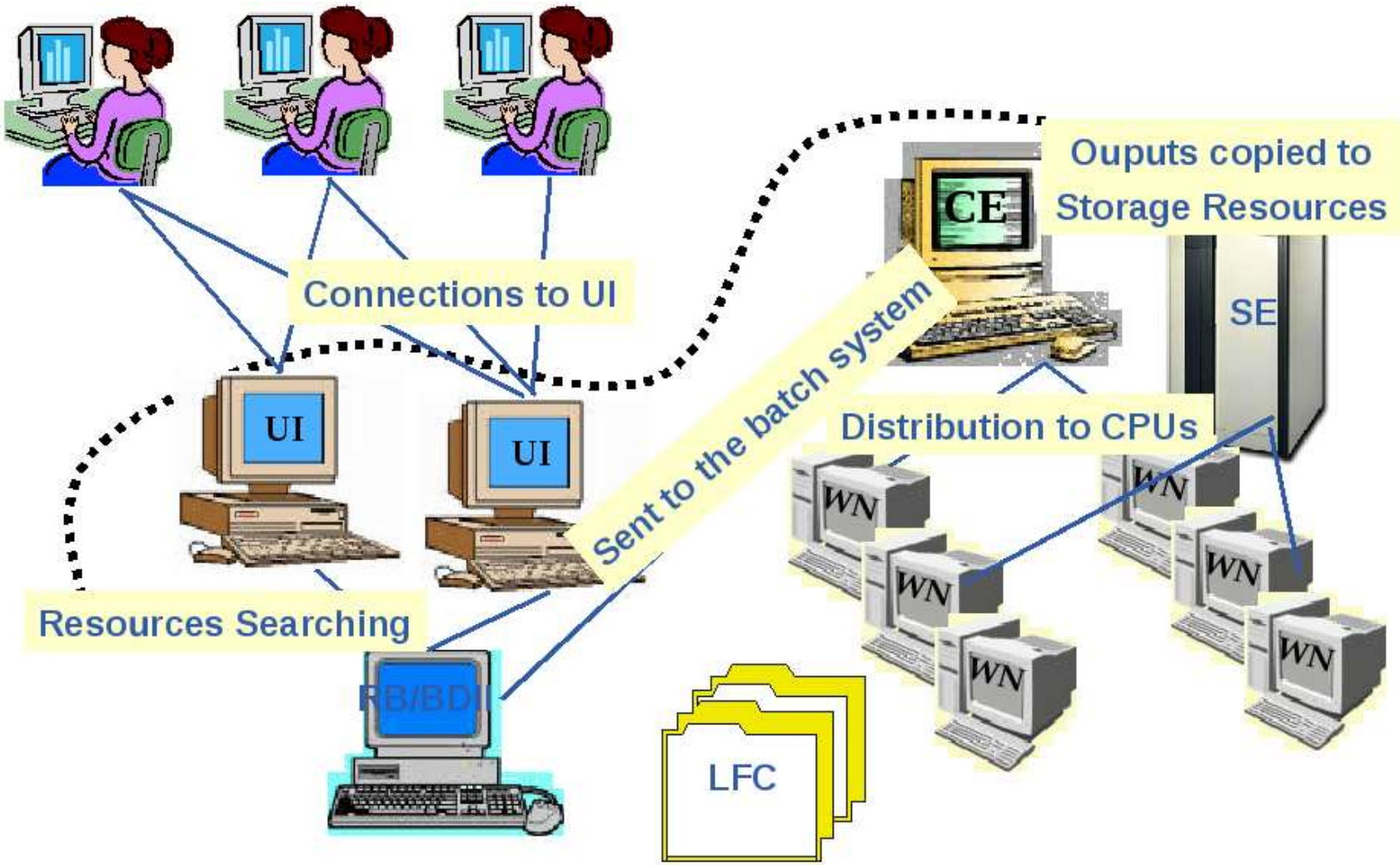
Outcome

An assessment over the most suitable data layout in function of the computational intensity of the function and the input size can be presented. Not restricted to PEAs!

Cárdenas-Montes, Miguel, et al.: **Effect of data layout in the evaluation time of non-separable functions on GPU**, Computing and Informatics, 1-21, ISSN: 1335-9150, 2015? Accepted

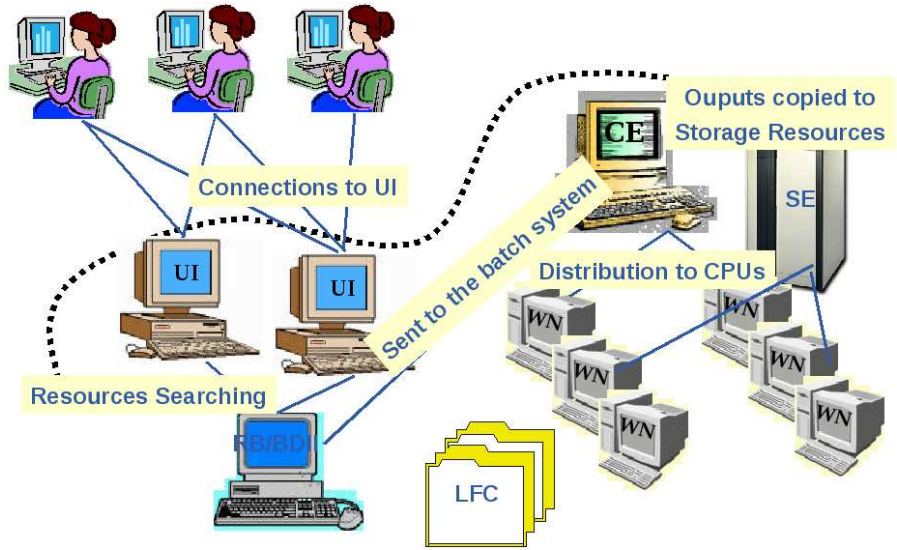
GPU+Grid

Grid Vintage Vision



Open Questions

- How I can discover GPU resources?
- How the information system has be modified? And the submission system?
- When a job is running in the GPU, the other cores can accept other jobs? Can pilot jobs help?
- More open questions in the audience...



And a lot of benefits!


Lessons
Learned
© JJAVA - Fotolia.com

Learned Lessons

- The learning curve is not negligible.
- Training and advisory activities are essential for success.
- Difficult to port existing codes. Even more difficult to efficiently port existing codes.
- High-intensity computational problems are necessary for exploiting the GPU capabilities.

Which is your Objective?

To get some acceleration: Almost sure due to hardware.
Objective when starting with GPUs. You walk along the learning curve.
Only some parts of the code are accelerated.

To port a code: The most difficult option.

To write a new code: IMHO the highest probability of success.
The features of the card are fully exploited.

To Swim in the Stream

- GPUs are incorporating stronger capacities. From Fermi (14 SM - 448 cores - 14336 threads, 0.52 Tflop/s double precision) to Kepler (84 SM - 2668 cores - 85376 threads, 1.31 Tflop/s double precision).
- CUDA includes more libraries: CUBLAS, CUFFT, CURAND, CUSPARSE, NPP, THRUST; plus debugging tools, profilers, etc.
- CUDA codes are forward compatible without modification. I don't tell any about the performance!
- More knowledge, expertise and success cases among our colleagues.

Conclusions

Conclusions

- GPU-CUDA a mature technology.
- Soundness of the science provided. More and more disciples on it.
- Jump forward other capabilities: analyses and simulations.
- Waiting for the GPU in grid.

Thanks

Acknowledgements

Antonio Delgado Peris, Christophe Ortiz, Fernando Jiménez, Juan José Rodríguez Vázquez, Ignacio Sevilla, Rafael Ponce, Eusebio Sánchez, Pedro Valero, Manuel Aurelio Rodríguez.

Thank you
Gracias

Questions?

More questions?