Point Distribution Tensor Computation on Heterogeneous Distributed Systems

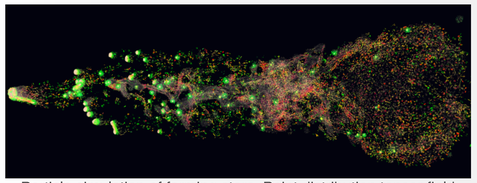
Ivan Grasso, Marcel Ritter, Biagio Cosenza, Werner Benger, Thomas Fahringer
University of Innsbruck, Austria

Overview

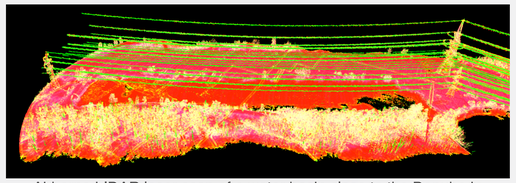
Point distribution tensor S:
- applied to point cloud data P_i
- describes local geometrical properties
- used to enhance visualization
  + enhance particle visualization
- used for other algorithms:
  + point classification of LIDAR laser scan data
  + eigenvector streamlines (power cable extraction)
- computationally expensive for big datasets (>500GB)

$$S(P_i) = \frac{1}{N} \sum_{K=1}^{N} \omega(|t_{ik}|, r)\left(t_{ik} \otimes t_{ik}^{\tau}\right)$$

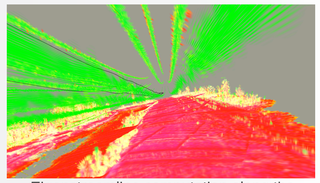
P_i point of a point cloud
N size of the neighborhood (number of points)
ω_{ik} weighting function
t_{ik} vector P_k - P_i
τ transpose of a vector
r neighborhood radius
$\otimes$ tensor product



Particle simulation of forming stars. Point distribution tensor field illustrates local geometrical properties, such as gas trails (green). Simulation data by Dominik Steinhauser, Institute of Astro- and Particle Physics, University of Innsbruck.

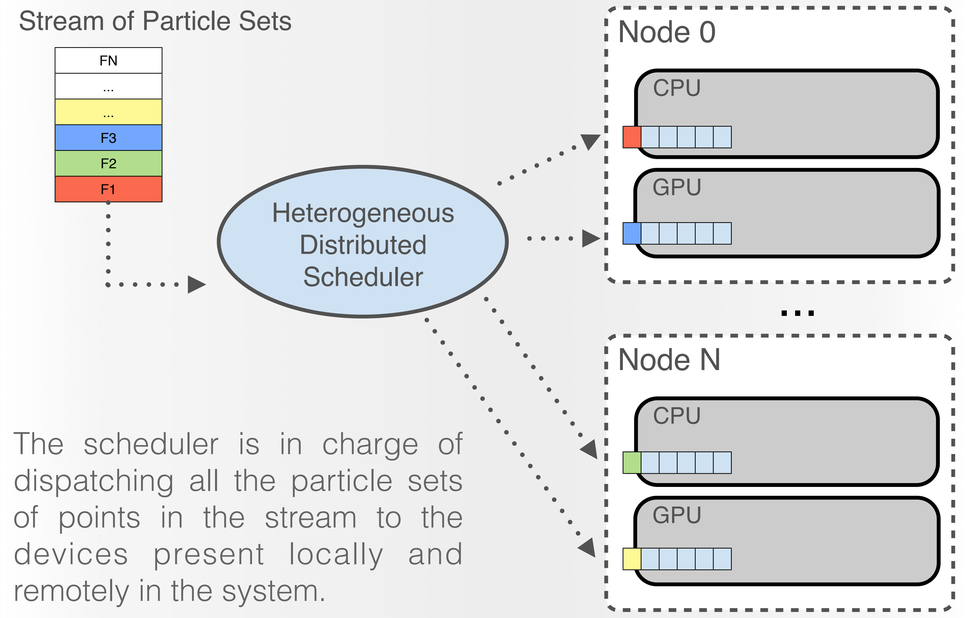


Airborne LIDAR laser scan of a water basin close to the Danube in Austria. The point distribution tensor field visually extracts power cables and the ground surface. Data acquisition by Frank Steinbacher, AHM, Innsbruck.



Eigenstreamline computation along the major eigenvectors of the point distribution tensor field reconstructs a power cable over a distance of 280m.
[Ritter M., Benger W., (2012). Reconstructing Power Cables from LIDAR Data Using Streamlines and the Point Distribution Tensor, Journal of WSCG.]

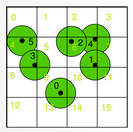
Heterogeneous Distributed Platform



The scheduler is in charge of dispatching all the particle sets of points in the stream to the devices present locally and remotely in the system.

GPU Implementation

The GPU implementation algorithm uses grid as a spatial data structure (spatial hashing). It comprises four steps:



1) for each particle a hash value is computed, i.e., the cell index where it is located

2) particles are sorted by hash; for this step an optimized bitonic sorting is utilized

3) the sorted list is used to compute the starting cell where the particle is located, running a thread for each particle, and performing scattered memory writes

4) tensor calculation: each particle searches the closest 27 grid cells from its location and it computes the tensor with each of the particles in these cells

Preliminary Results

We conducted our preliminary experiments with a small dataset of 1024 particles. Even for such a small amount of particles the GPU outforms the performance of a quadcore CPU by 25%, showing its suitability for this kind of computation.

In the future we will extend our tests to bigger datasets trying different heterogeneous scheduling approaches.
We will also explores different optimizations for different architectures including local memory for the GPU and vectorization for the CPU.

We will also tune our scheduling policy to support massive stream of particle sets in heterogeneous distributed systems.