cerit
scientific cloud

Acceleration of 3D reconstruction in cryo-EM

David Střelák, Carlos Óscar Sorzano, José María Carazo, Jiří Filipovič

Fall 2023

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj
pro inovace

Visualization of Molecules

Several methods are able to visualize molecules on atomic level

- X-ray diffraction
- nuclear magnetic resonance (NMR)
- cryo-electron microscopy (cryo-EM)

Cryo-EM has some superiority over other methods

- catches molecules in natural environment (diffraction needs crystalization)
- usable for large molecules (NMR is restricted to smaller proteins)

Cryo-electron microscopy

Rapidly-developed recently

- in 2012, there was only four structures at near-atomic resolution
- in 2015, 115 structures was discovered
- this progress is allowed by direct electron detectors, viterious ice and **image reconstruction methods**

In 2017, Nobel price in chemistry was given for cryo-EM

- Jacques Dubochet, Joachim Frank, Richard Henderson
- Joachim Frank got his price for image processing methods allowing to obtain 3D structure from electron microscope data

Cryo-electron microscopy

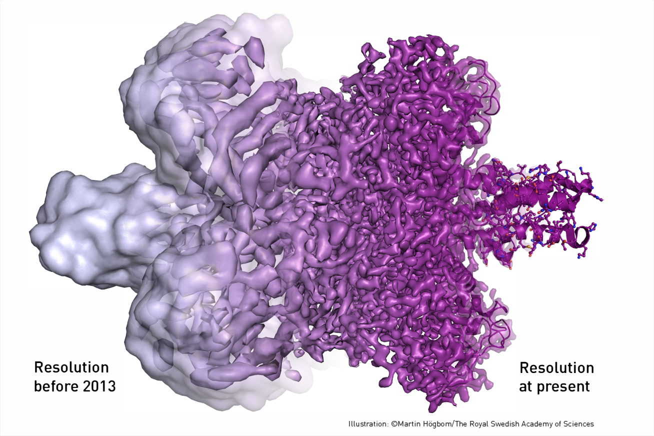


Illustration: ©Martin Högbom/The Royal Swedish Academy of Sciences

Specimens in the Ice

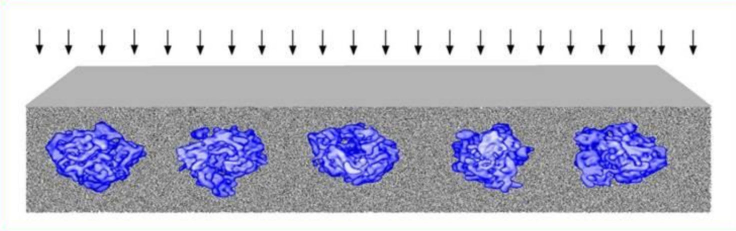


Image Analysis in Cryo-EM

Reconstruction of 3D volume is challenging

- electron beam causes damages, so it must be weak, so a noise-to-signal distance is very low
- surrounding water adds another source of noise
- specimens are captured in random positions, possibly with conformational changes
- when captured multiple times, the image is moving and deforming

Raw data from microscope

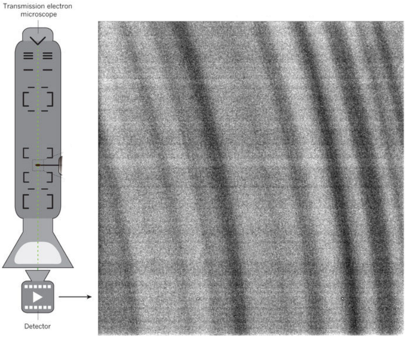
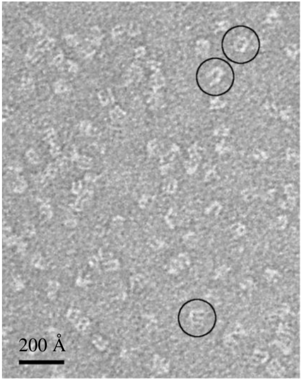
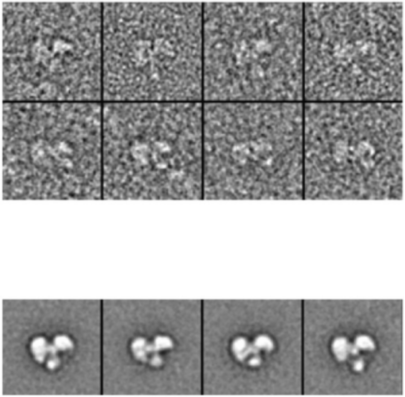


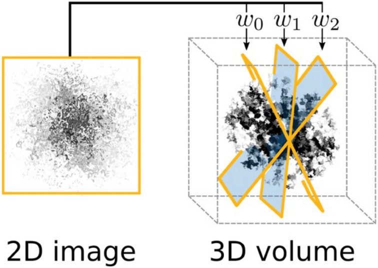
Image of Specimens



Aligning Images



3D Volume Reconstruction



Our Focus

Image reconstruction is very computationally-demanding

- requires thousands of CPU hours at least
- 3D reconstruction is one of main bottlenecks

We focus on acceleration of 3D volume reconstruction in Xmipp software

- software developed in Spanish National Center for Biotechnology (CNB-CSIC)
- production use, not a prototype-toy

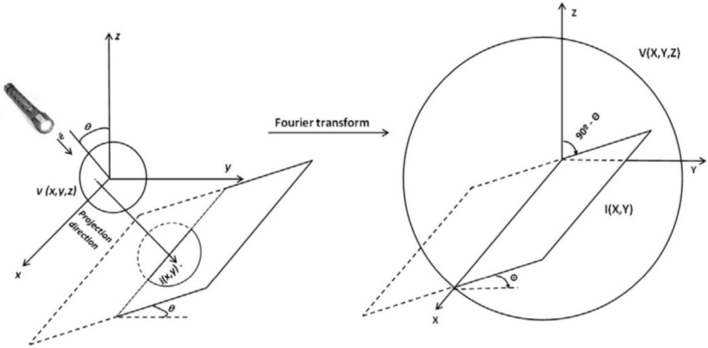
Getting 3D Volume from Images?

Central slice theorem

- let *i* be real-space projection image, which has concentrated information about 3D volume *v*
- let *I* be a Fourier transform of image *i* and *V* be Fourier transform of *v*
- *I* forms a slice of *V* with the same orientation as *i* holds with respect to *v*, moreover, slice *I* is going through center of *V*

So, we need to transform our images into Fourier space, create 3D Fourier volume and transform the volume back to real space.

3D Volume Reconstruction



3D Volume Reconstruction

We need to guess orientation of each 2D image

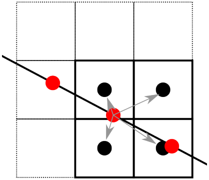
- computed iteratively
- bottleneck is creating 3D volume from 2D images

We have accelerated the creation of 3D volume on GPUs.

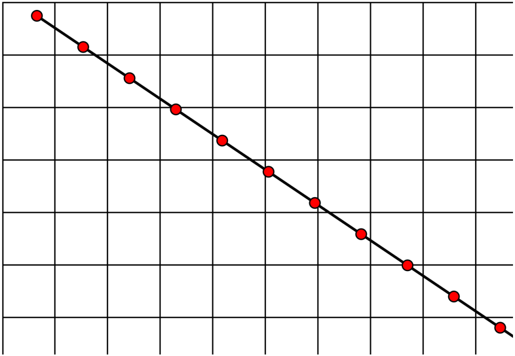
State-of-the-art

Multiple papers deal with GPU acceleration of 3D reconstruction, all implementing a scatter method

- GPU threads are associated to 2D pixels of the image
- each thread computes projection of the pixel into volume (resulting in floating-point position)
- the pixel value is put into multiple voxels (integer position) using interpolation



State-of-the-art



State-of-the-art

Drawbacks of scatter pattern

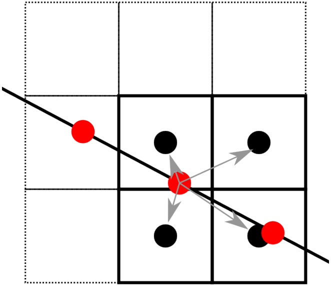
- race conditions in writing (distances within a voxel up to $\sqrt{3}\times$ longer than distance between two pixels), requires atomic writes
- some wrong optimizations removing atomics have been published
- frequent writing into 3D domain with poor spatial locality

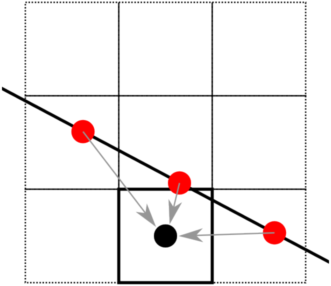
The Gather Pattern

The image value is computed for each 3D voxel

- so each voxel is written only once
  - no race conditions in reading
- image data are interpolated (we obtain floating-point position in the image), so they are accessed multiple times
  - much better memory locality (we are now repeating accesses into 2D image, not 3D volume)

The Gather Pattern





The Gather Pattern

Projecting 3D voxels to 2D image

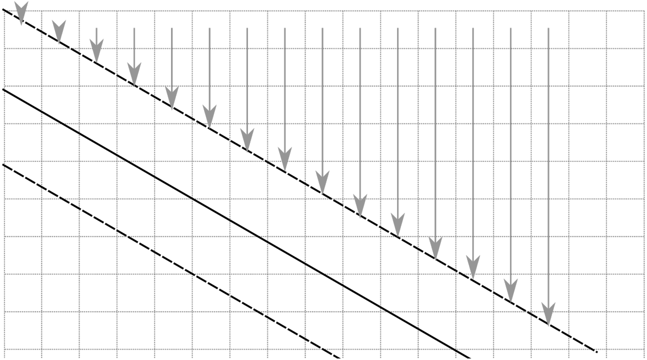
- when going into image space, we get position in the image and z-distance from the image
- a lot of voxels do not hit the image (z-distance is too high, or they are out of image boundaries)
    - we have $\mathcal{O}(n^2)$ pixels, but $\mathcal{O}(n^3)$ voxels – a lot of them is not used

The Gather Pattern

Projection planes optimization

- we look at the image from some plane orthogonal to coordinate axes (XY, XZ, YZ), which maximizes projected image size
- the iteration space is reduced to the projection plane
- for each point of the projection plane, we compute the distance of the image and start to process voxels from there

The Gather Pattern



Basic GPU Implementation

The gather pattern can be rewritten for GPU directly

- one GPU thread is assigned to one point of projection plane

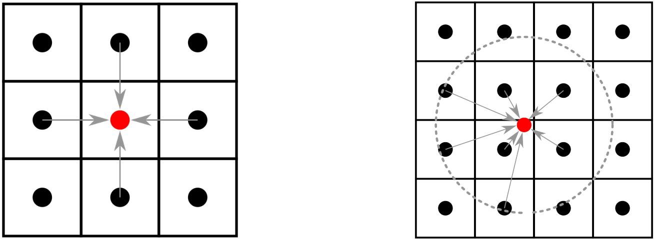
Optimization opportunities

- the advanced interpolation method may be computationally demanding
- GPU cache system is limited in maintaining data locality

Interpolation

Our computation is a kind of stencil, but with floating-point positions

- interpolation coefficients cannot be precomputed easily



Interpolation

We have implemented two strategies

- on-the-fly interpolation
- precomputed table for very fine steps (originally in Xmipp)
    - can be cached or preloaded into shared memory

Explicit Caching of Image Data

A thread block accesses only a part of the image

- can be cached in fast shared memory
- however, its size may vary depending on image rotation
    - we upper-bound image size to $\lceil\sqrt{2}\sqrt{3}(b+2i)\rceil$, where b is thread block size and i is interpolation radius
    - shared memory is allocated to upper-bound prior GPU kernel execution
- for each image, AABB is computed and proper size is preloaded in shared memory

Additional Optimizations

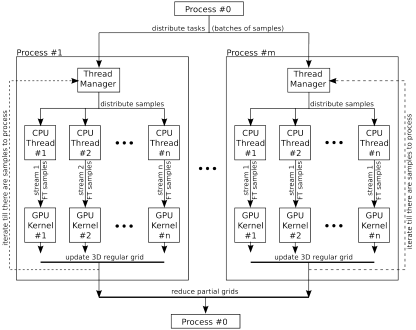
Register consumption optimization

- many parameters into templates or macros
- allows to increase GPU parallelism

CPU-GPU load balancing

- CPU prepares images for GPU, one core is not powerful to do so
- multiple threads are preparing images and sharing GPU time, also allows copy and computation overlay

Architecture



Obrázek: Architecture of 3D Fourier Reconstruction.

Autotuning

parameter	values
BLOCK_DIM	8, 12, 16, 20, 24, 28, 32
ATOMICS	0, 1
GRID_DIM_Z	1, 4, 8, 16
PRECOMP_INT	0, 1
SHARED_INT	0, 1
SHARED_IMG	0, 1
TILE_SIZE	1, 2, 4, 8

Evaluation

Tested on a GPGPU cluster node

Processor	performance	memory BW
2× Xeon E5-2650 v4	845 GFlops	154 GB/s
1× Tesla P100	9,519 TFlops	732 GB/s
4× Tesla P100	38,076 TFlops	2928 GB/s
theoretical speedup (1 GPU)	11.3×	4.75×

Evaluation

execution	time	speedup over original
2× CPU	155m	n/a
1× GPU	13m35s	11.4×
4× GPU	4m53s	31.7×

Performance Portability

Tabulka: Performance portability of 3D Fourier Reconstruction

	P100	GTX1070	GTX750	GTX680
Tesla P100	100%	95%	44%	96%
GTX 1070	88%	100%	31%	50%
GTX 750	65%	67%	100%	94%
GTX 680	71%	72%	71%	100%

We can gain over 3× speedup when tuning for each GPU architecture.

Performance Portability

Tabulka: Sensitivity on input images in 3D Fourier Reconstruction (GTX 1070)

	128x128	91x91	64x64	50x50	32x32
128x128	100%	100%	77%	70%	32%
91x91	100%	100%	76%	68%	33%
64x64	94%	94%	100%	91%	67%
50x50	79%	78%	98%	100%	86%
32x32	65%	67%	80%	92%	100%

We can gain over 3× speedup when tuning for specific input size.

Conclusion

We have implemented fast, production-ready algorithm

- significant performance boost
- implemented in Xmipp from beginning

Advantages over state-of-the-art

- gather approach is already significantly faster (about 2x on Pascal architecture)
- we suppose it will be even faster with further architectures (bigger flops-to-memory gap, higher parallelism)
- we do not rely on HW implementation of atomics (negligible slowdown when e.g. result is stored in double-precision)

Programming Complexity

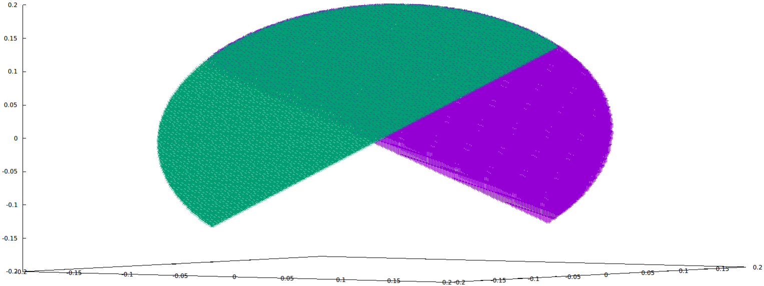
The basic idea of the algorithm is pretty simple

- we are just putting 2D slices into 3D space, right?

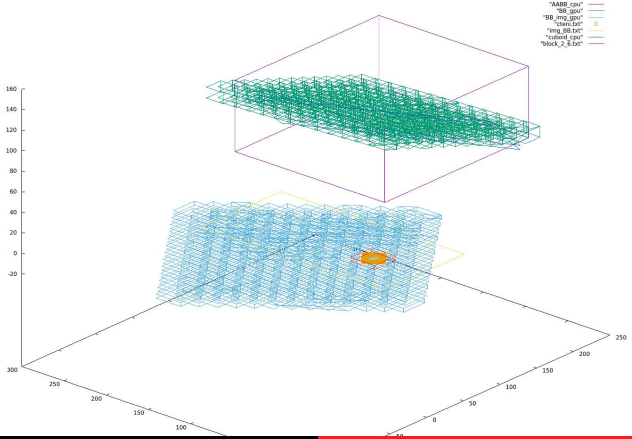
Indexing hell

- Fourier space is symmetric, we need to deal with its boundaries
- going from 3D integer position to 2D real position with handling of 3D symmetry, 2D symmetry and space padding in both 3D and 2D is challenging
- because of real position, it is not always clear if we compute correctly (e.g. how to trace boundary conditions?)
- the indexed space is extremely large
- gnuplot seems really good tool

Debuging with gnuplot



Debuging with gnuplot



Finding Bugs

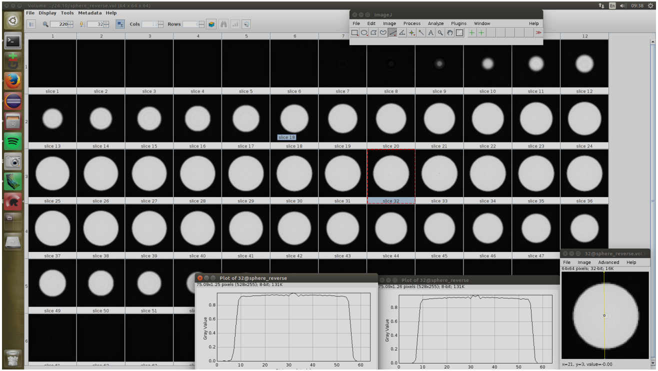
It is not simple to determine what is a bug in noisy data

- just by moving from scatter to gather, we already compute something else

Errors in some part of pipeline are difficult to interpret

- e.g., bad indexing in Fourier space looks really weird in real space
- too long chain: 2D real → 2D Fourier → 3D Fourier → 3D real

Sphere



Volume: ../26.10/sphere_reverse.vol (64 x 64 x 64)

Plot of 32@sphere_reverse

75.09x1.25 pixels (528x255); 8-bit; 131K

Plot of 32@sphere_reverse

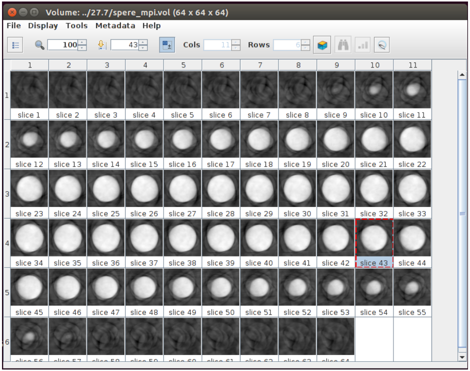
75.09x1.26 pixels (528x255); 8-bit; 131K

32@sphere_reverse.vol

64x64 pixels; 32-bit; 16K

x=21, y=3, value=-0.00

Sphere with Indexing Bug



Volume: ../27.7/spere_mpi.vol (64 x 64 x 64)

100

43

Cols 11

Rows 6