Empty on purpose

Acceleration of image processing algorithms for single particle analysis by electron microscopy

RNDr. David Střelák

Supervisors: prof. Luděk Matyska (MU)
prof. José María Carazo (CNB)
Consultants: Jiří Filipovič, Ph.D. (MU)
Carlos Óscar Sánchez Sorzano, Ph.D. (CNB)

RNDr. David Střelák

Masaryk University, CZ

Universidad Autónoma de Madrid, ES

- 2018 - 2022
- Computer Science, doctoral degree programme
- Programa de Doctorado en Ingeniería Informática
- INPhINIT “la Caixa” fellowship
- degree conferred: RNDr., in 2020

MUNI FI

Goals

Improve the algorithms used in Cryo-EM

- single-node performance optimizations
- heterogeneous computing
- autotuning
- novel algorithms

Identify the limitations of state-of-the-art HPC techniques

- application in Cryo-EM
- extend those techniques to overcome their limitations

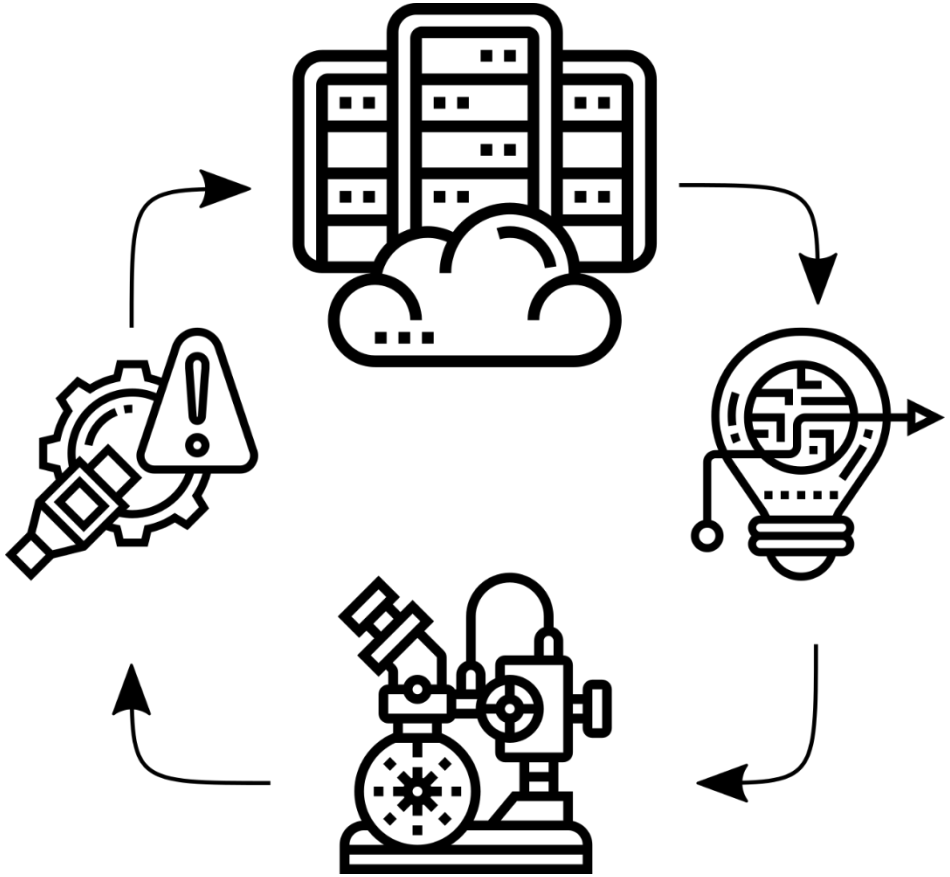
Collaboration between HPC and Cryo-EM

Generally applicable tools in HPC

- how to find them?
- how to test them?

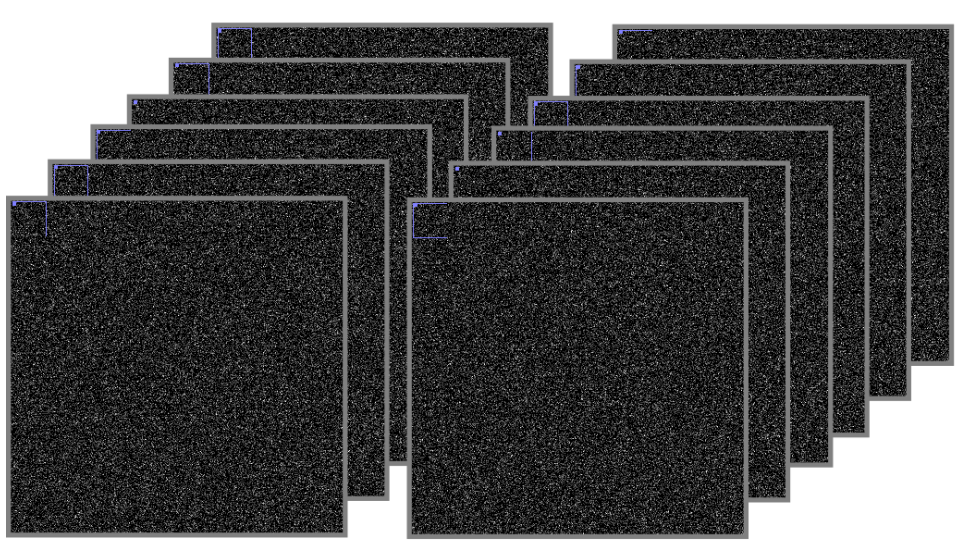
Cryo-EM

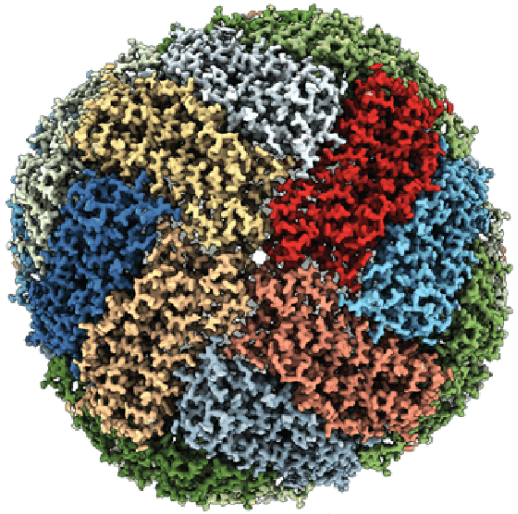
- important part of the structural biology
- complex workflow
- high computational demands



MUNI FI

Cryogenic Electron Microscopy

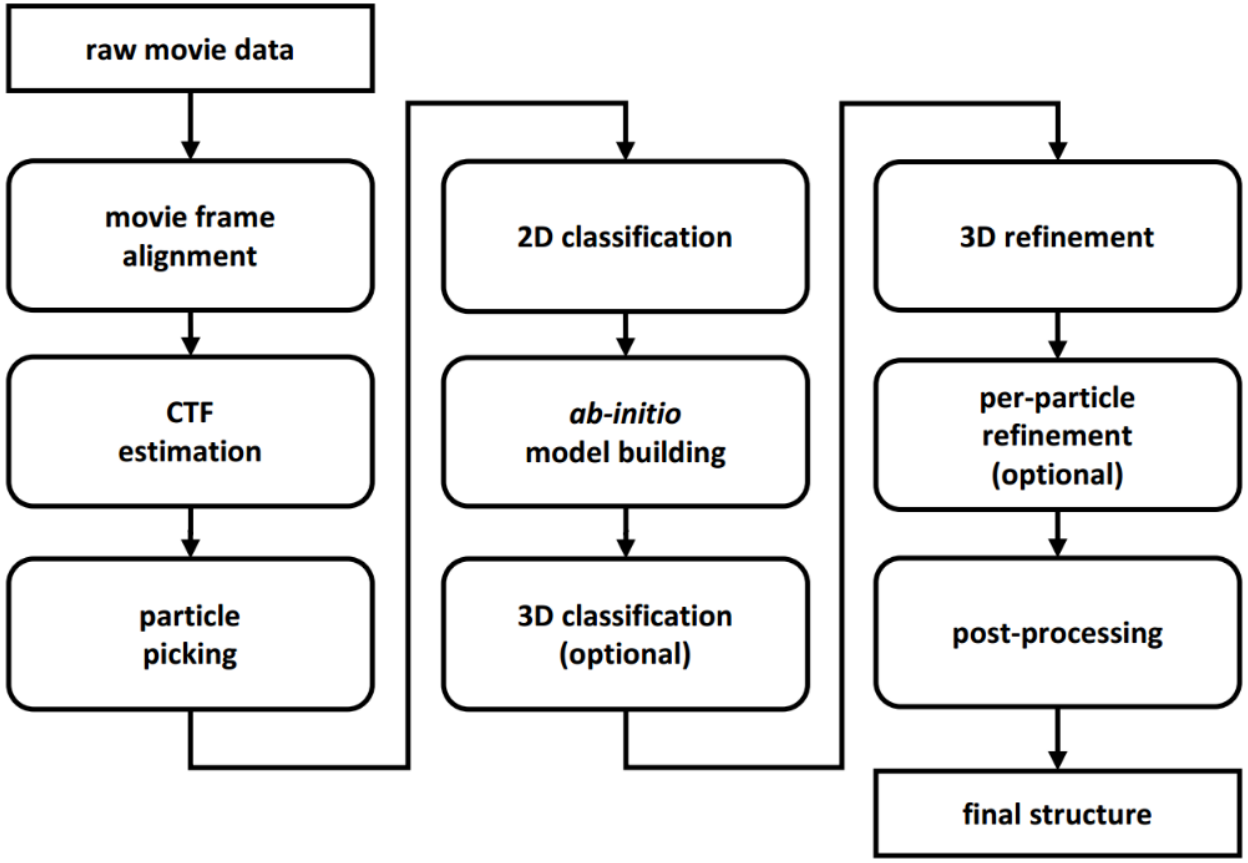




MUNI FI

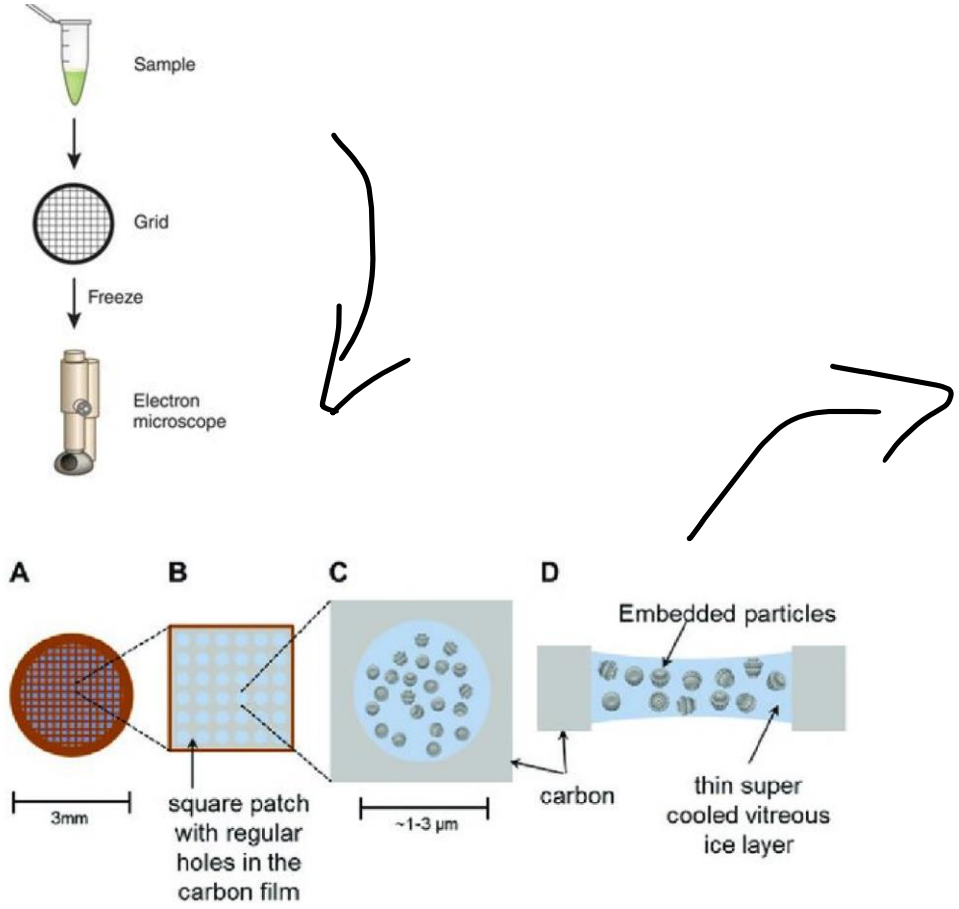
Cryo-EM

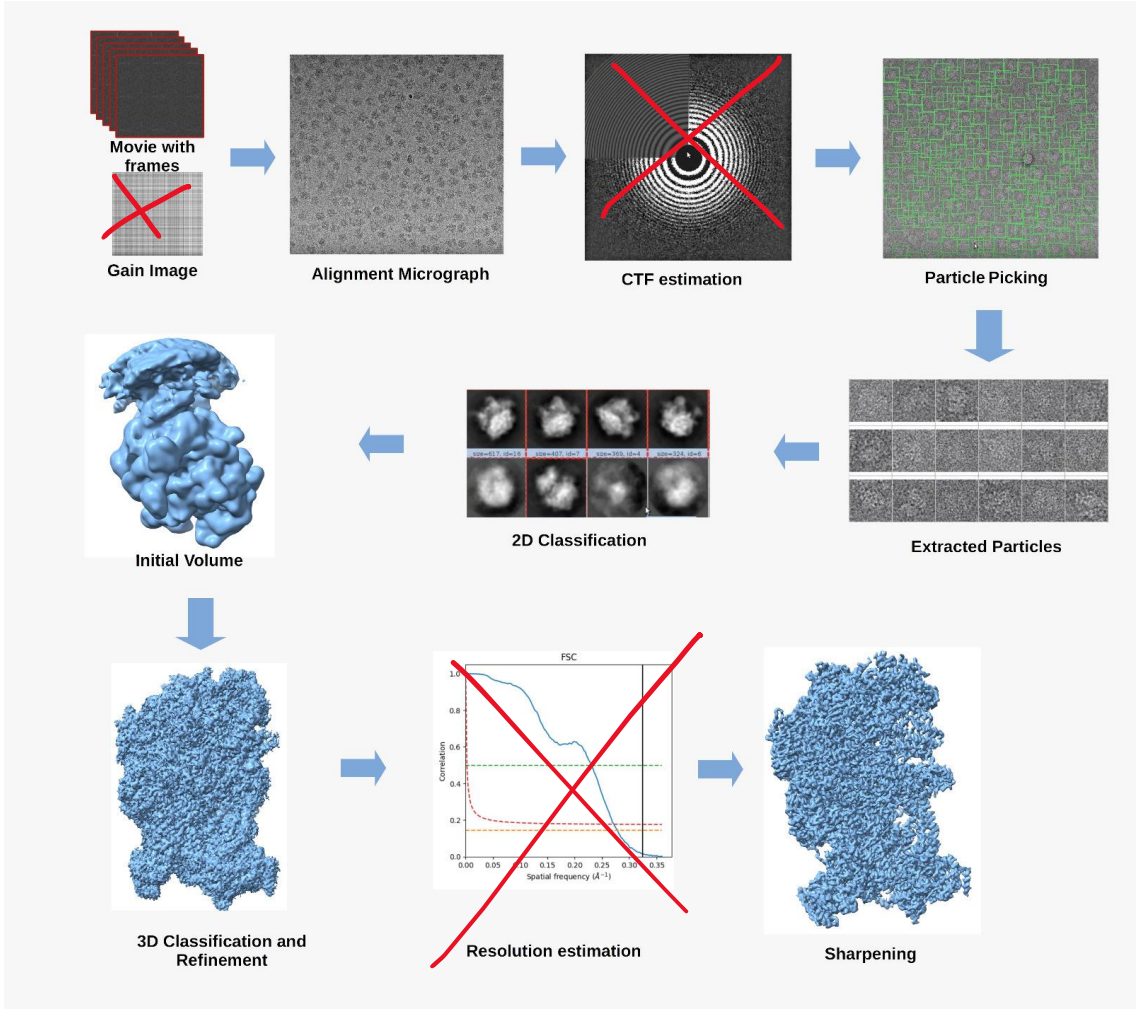
- applicable techniques differ depending on the part of the pipeline
- new advanced algorithms are being researched
  - computationally very expensive conformational landscapes



MUNI FI

Cryogenic Electron Microscopy





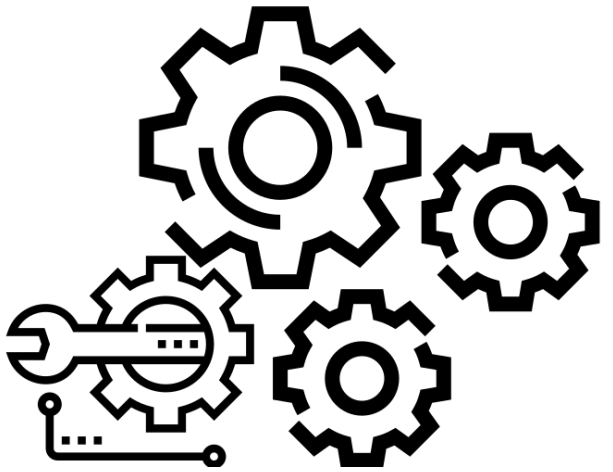
MUNI FI

Acceleration by …

- Using accelerator
  - Sheer brute force
- Faster algorithm
  - More efficient
  - Better fit the HW
- Adjusting implementation
  - To HW
  - To Data
- Adjusting invocations
  - Necessary for close-sourced libraries
- Utilizing all resources
  - Heterogeneous computing

Table 2.1: Raw performance (TFLOPS) comparison of multiple accelerators and processors[47][48][49][50][51][52])

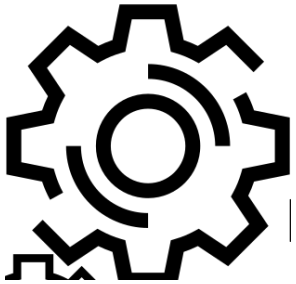
Precision	DP	SP	HP
Intel® Xeon Phi 7290	2.92 (1:2)	5.84	—
AMD Radeon VII	3.360 (1:4)	13.44	26.88 (2:1)
NVIDIA GeForce RTX 2080 Ti	0.41 (1:32)	13.45	26.90 (2:1)
NVIDIA TITAN RTX	0.51 (1:32)	16.31	32.62 (2:1)
NVIDIA Tesla V100	7.06 (1:2)	14.13	28.26 (2:1)
Intel Core I9-7980XE	1.12 (1:2)	2.24	—

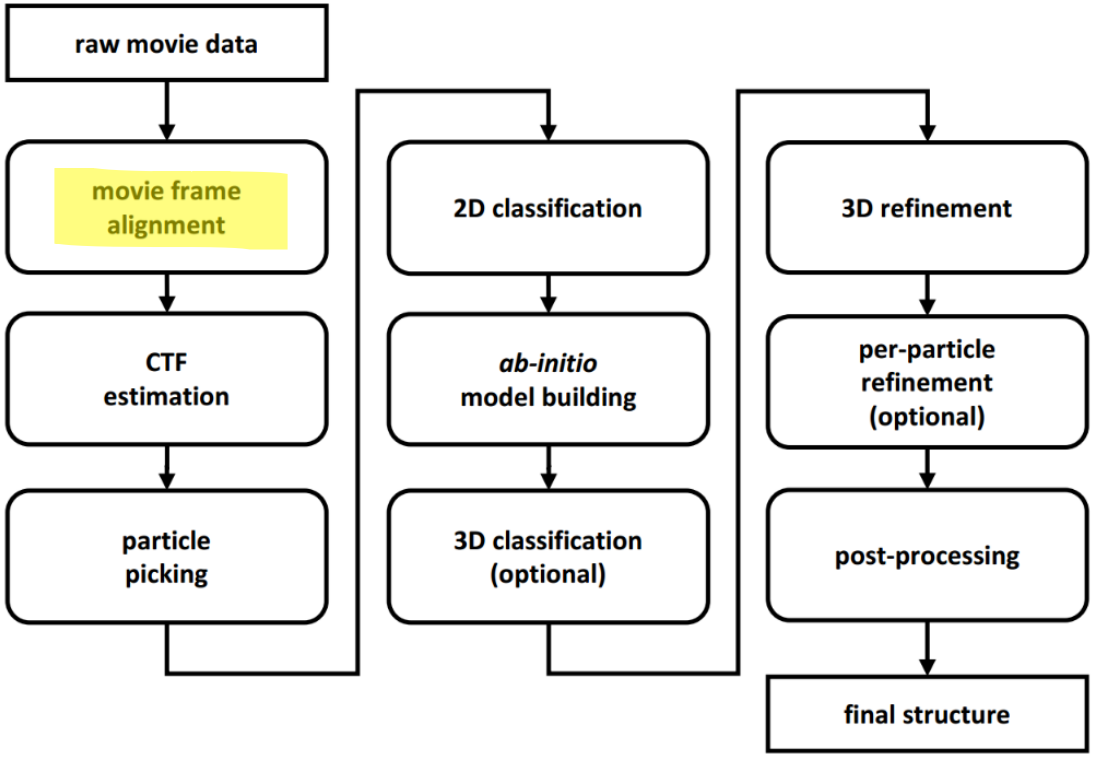


MUNI FI

Acceleration by …

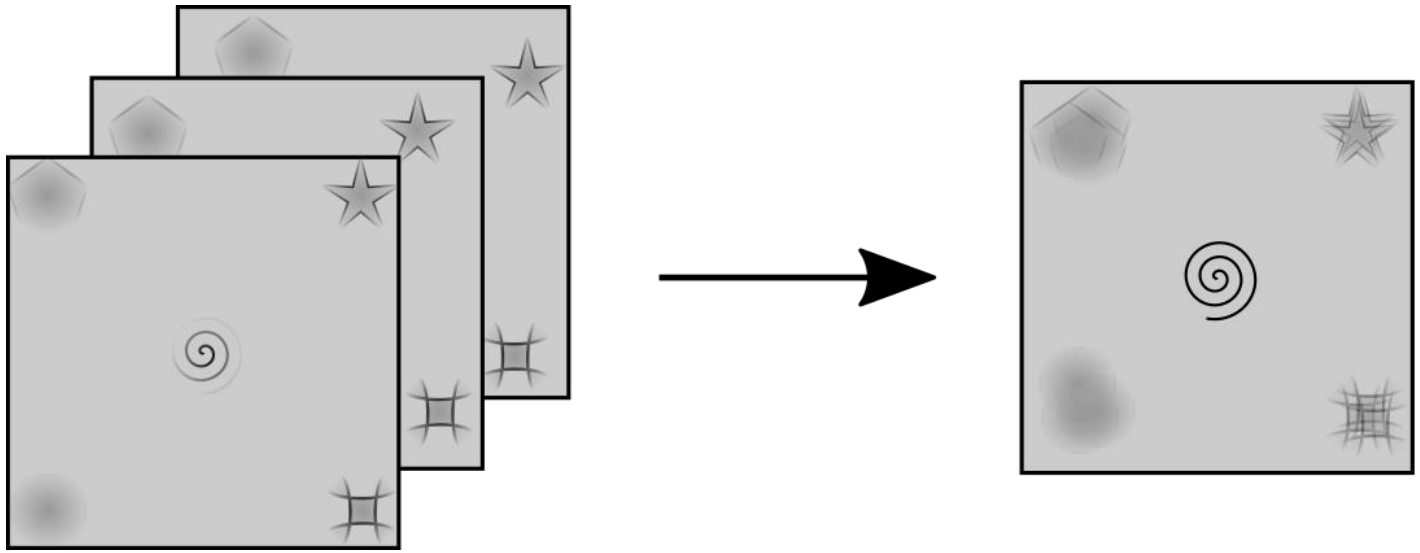
- Using accelerator
  - Sheer brute force





MUNI FI

FlexAlign



[1] STŘELÁK et al. FlexAlign: An Accurate and Fast Algorithm for Movie Alignment in Cryo-Electron Microscopy. Electronics. Switzerland: MDPI, 2020, vol. 9, No 6, p. 1-25. ISSN 2079-9292. doi:10.3390/electronics9061040. Scopus Q2

FlexAlign – global alignment

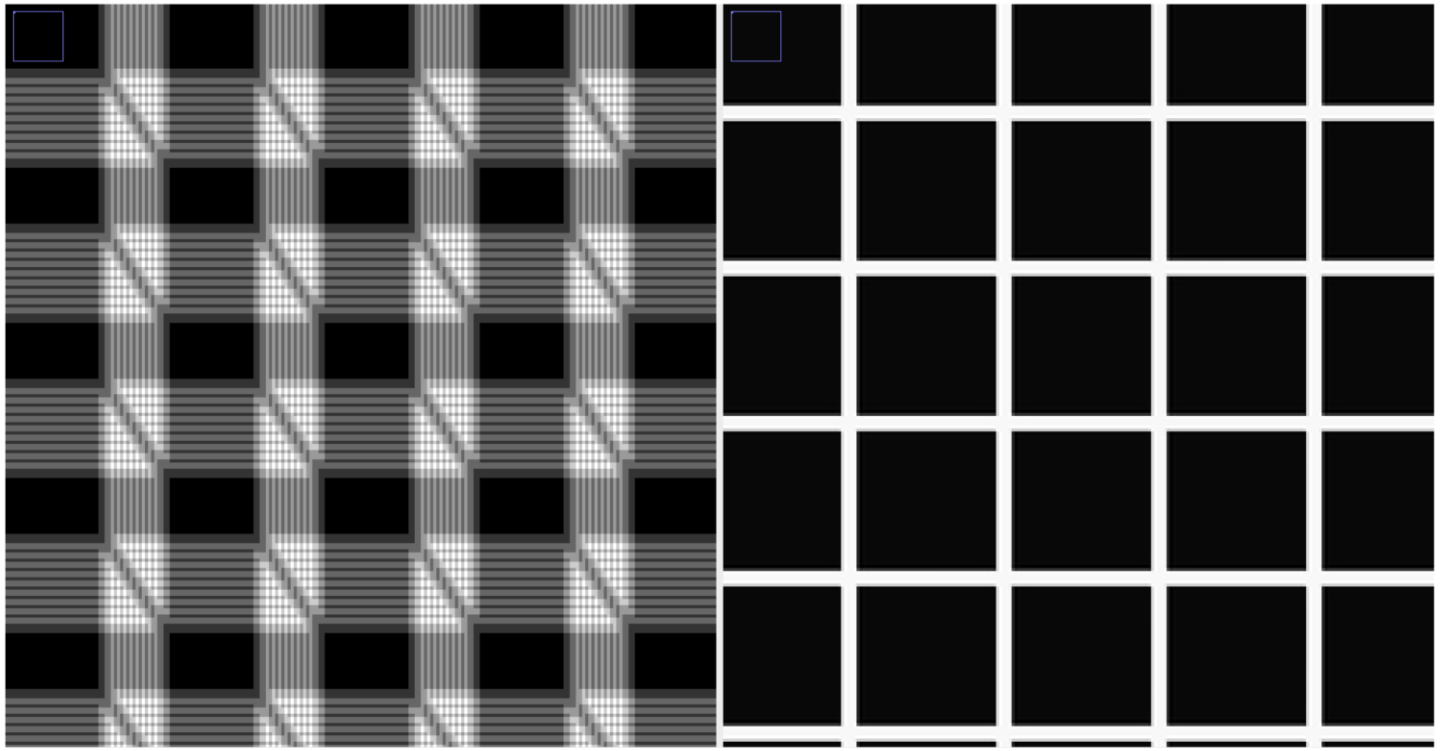
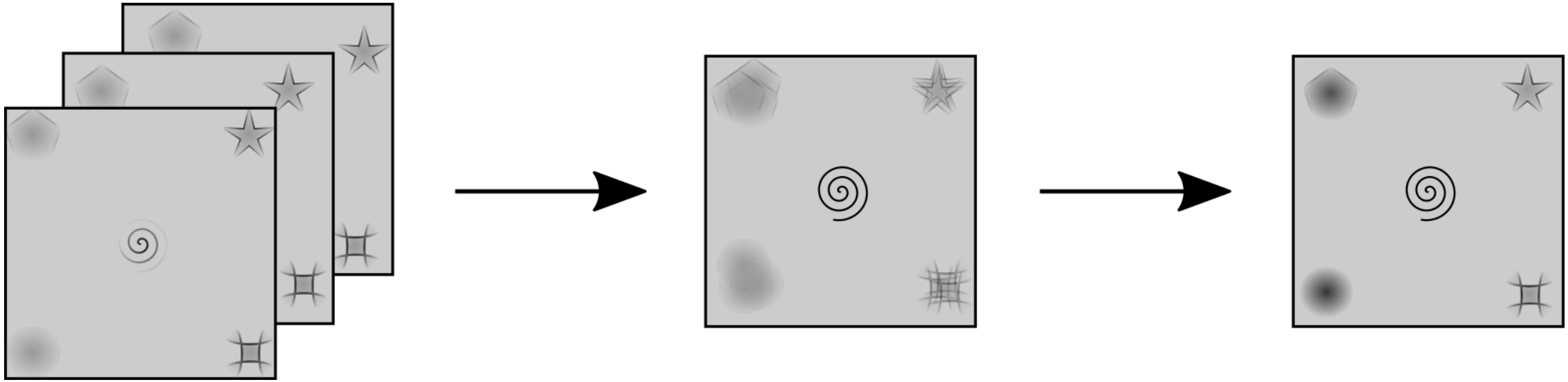


Figure 2. Phantom movie (grid, detail), an average of 10 frames, nth frame shifted by the vector $[2n, 3n]$, before global alignment (**left**), after global alignment (**right**).

MUNI FI

FlexAlign



[1] STŘELÁK et al. FlexAlign: An Accurate and Fast Algorithm for Movie Alignment in Cryo-Electron Microscopy. Electronics. Switzerland: MDPI, 2020, vol. 9, No 6, p. 1-25. ISSN 2079-9292. doi:10.3390/electronics9061040. Scopus Q2

MUNI FI

FlexAlign – local alignment

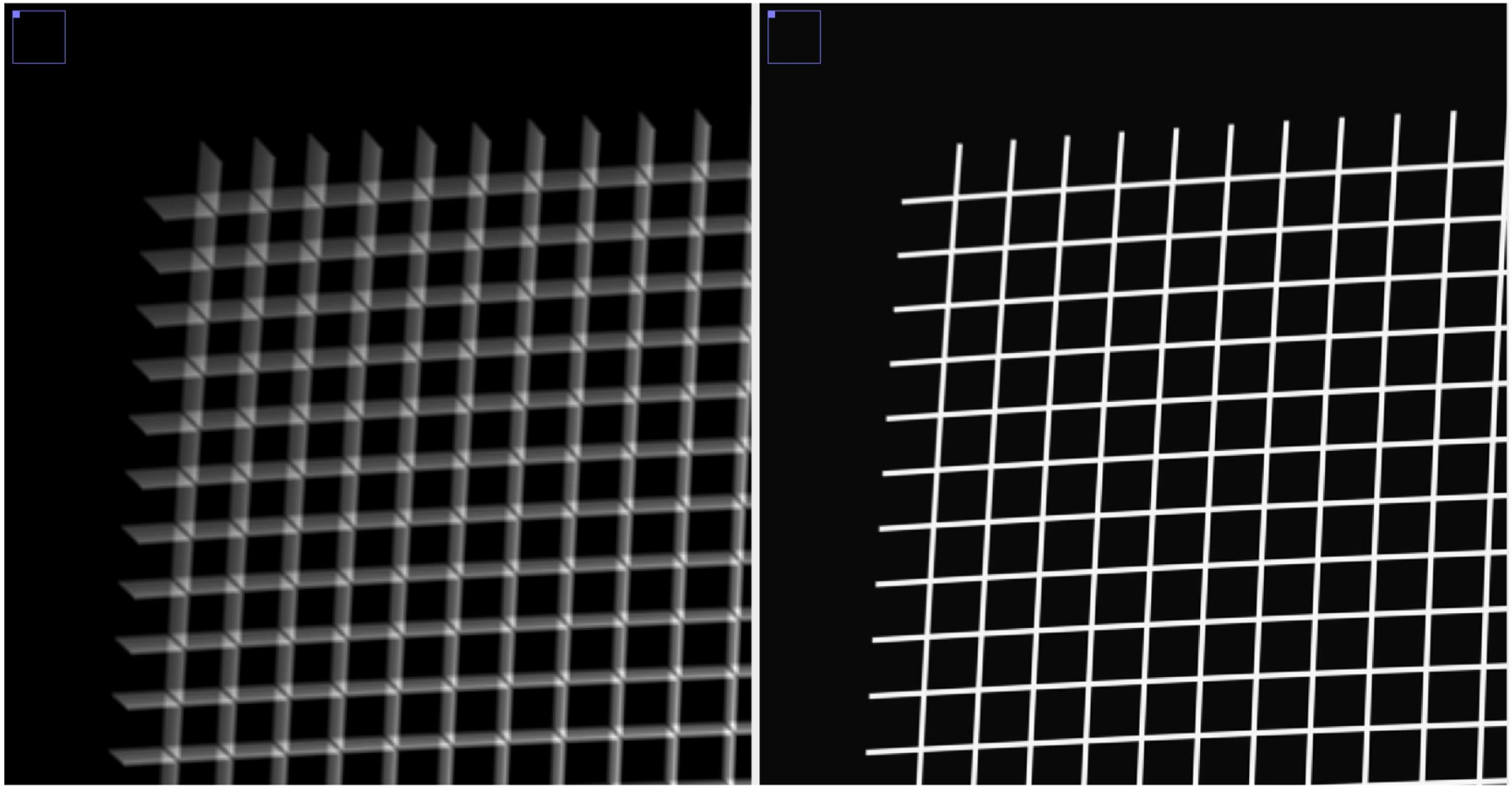
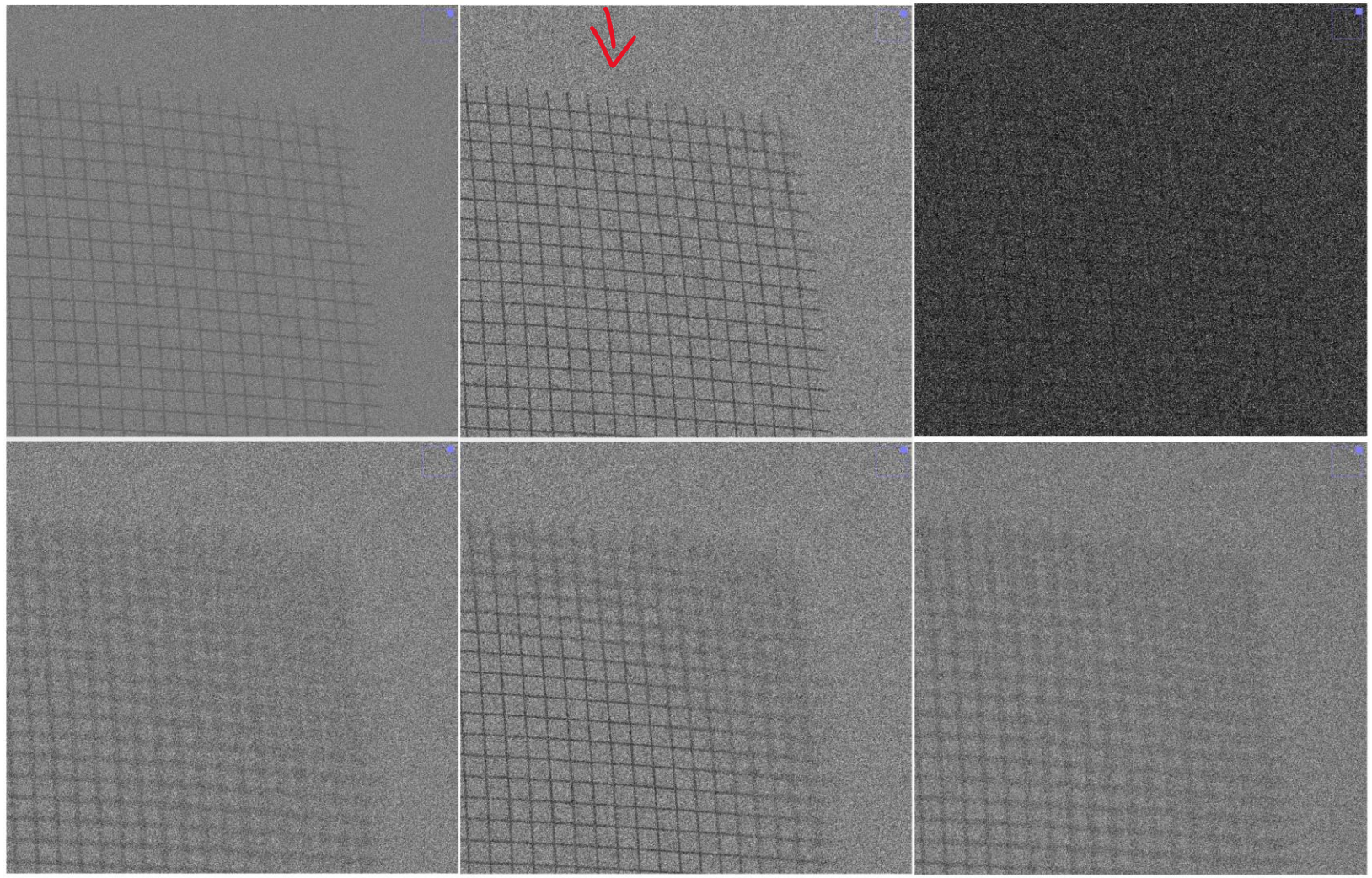


Figure 3. Phantom movie (grid, detail), an average of 50 frames, frames shifted + doming applied, using only global alignment (**left**), after local alignment (**right**).

MUNI FI

FlexAlign - quality



Phantom

Figure 7. Detail of the frame of the phantom movie (**top left**), detail of produced micrograph, normalized: cryoSPARC (**top right**), FlexAlign (**center left**), Warp (**center right**), MotionCor2 (**bottom left**), Relion MotionCor (**bottom right**).

MUNI FI

FlexAlign - quality

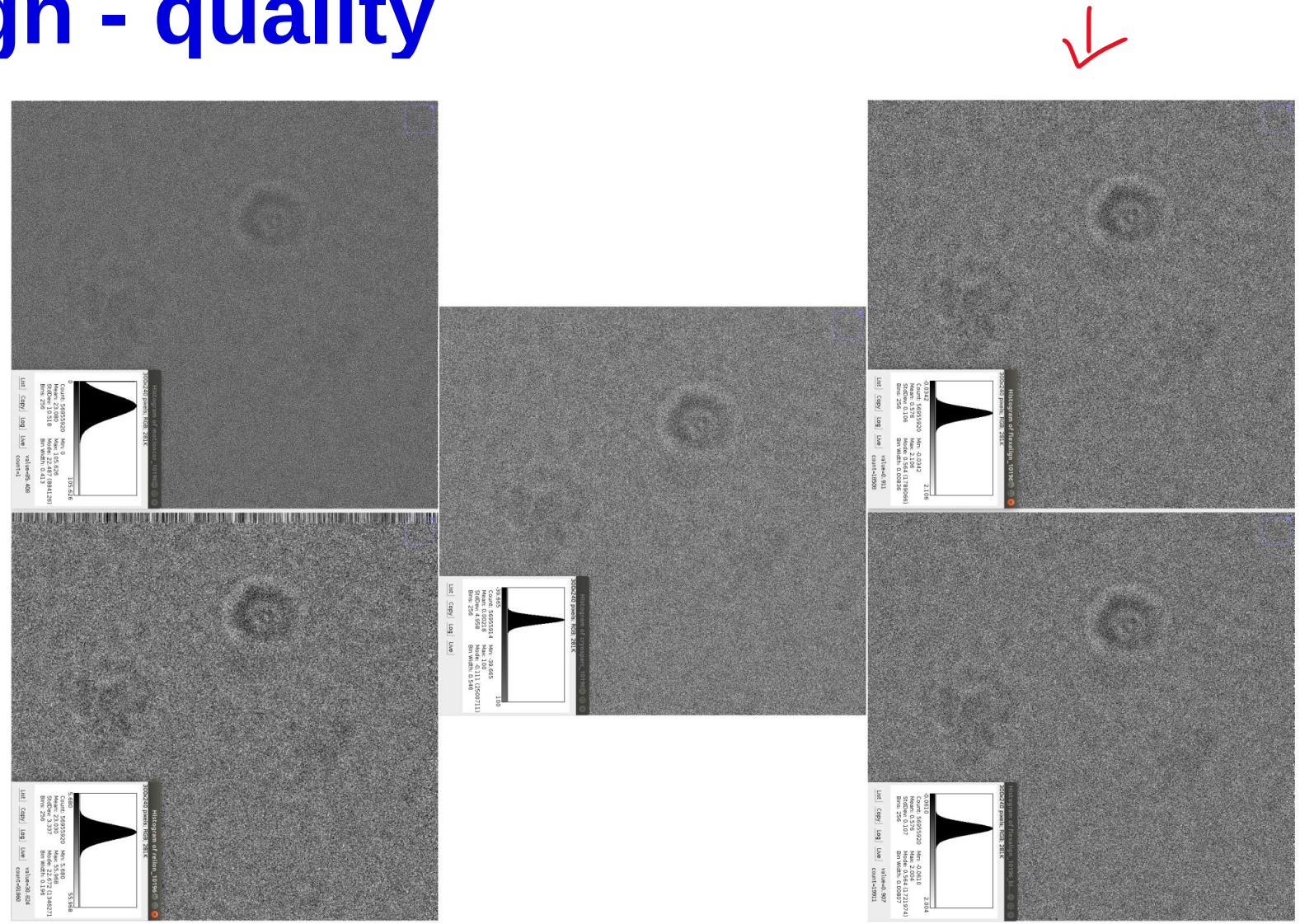


Figure 15. Details of the produced micrograph using EMPIAR 10196 dataset (normalized) with histogram (before normalization): FlexAlign **(top left)**, FlexAlign with max shift of 80 px **(top right)**, cryoSPARC **(center)**, MotionCor2 **(bottom left)**, Relion MotionCor **(bottom right)**.

MUNI FI

FlexAlign - flexibility

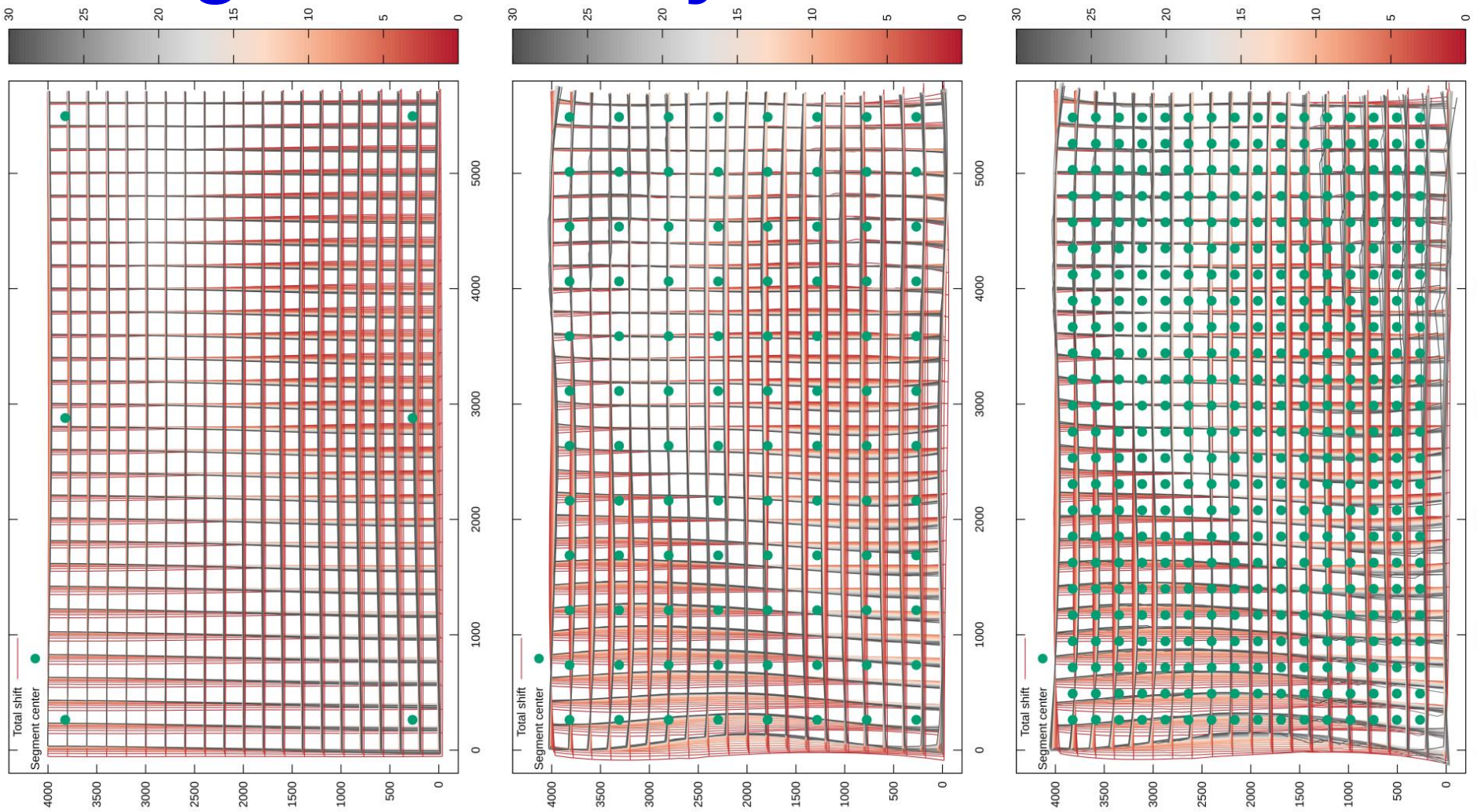


Figure 17. Influence on shift estimation using a different number of patches: 3×2 (**top**), 12×8 (**middle**, default value), 24×16 (**bottom**).

MUNI FI

FlexAlign - performance

Table 2. HW used for benchmarking.

	Testbed 1	**Testbed 2**	**Testbed 3**
CPU	Intel(R) Core(TM) i7-8700 (12 cores, 3.20 GHz)		Intel(R) Core(TM) i7-7700HQ (4 cores, 2.80 GHz)
GPU	GeForce RTX 2080	GeForce GTX 1070	GeForce GTX 1060
CUDA/driver	10.1/418.39	10.1/418.67	8.0.61/436.02 (Win 10)/390.116 (Ubuntu 18.04)
SSD	Samsung SSD 970 EVO 500 GB		NVMe TOSHIBA 1024 GB
RAM	2 × 16 GB DDR4 @ 2.6 GHz		2 × 16 GB DDR4 @ 2.4 GHz

Table 3. Resolution and number of frames used for testing.

	Size	**No. of Patches**
Falcon	4096 × 4096 × 40	9 × 9
K2	3838 × 3710 × 40	8 × 8
K2 super (resolution)	7676 × 7420 × 40	16 × 15
K3	5760 × 4092 × 30	12 × 9
K3 super (resolution)	11,520 × 8184 × 20	24 × 17

FlexAlign - performance

Table 4. Execution time on Testbed 1.

	Falcon	K2	K2 Super	K3	K3 Super
MotionCor2	4.6 s	4.3 s	15.7 s	5.0 s	13.1 s
FlexAlign ~~(tuned)~~	9.2 s	7.6 s	25.6 s	8.8 s	20.5 s
~~FlexAlign (autotuning)~~	~~49.2 s~~	~~34.4 s~~	~~71.9 s~~	~~59.3 s~~	~~72.2 s~~
~~FlexAlign (non-tuned)~~	~~10.8 s~~	~~9.1 s~~	~~31.5 s~~	~~10.7 s~~	~~22.9 s~~
~~Movies to pay off~~	~~25~~	~~18~~	~~8~~	~~27~~	~~22~~

Table 5. Execution time on Testbed 2.

	Falcon	K2	K2 Super	K3	K3 Super
MotionCor2	5.5 s	5.2 s	20.3 s	5.9 s	15.7 s
FlexAlign ~~(tuned)~~	9.0 s	8.2 s	27.3 s	9.3 s	21.1 s
~~FlexAlign (autotuning)~~	~~47.8 s~~	~~38.7 s~~	~~73.3 s~~	~~56.2 s~~	~~65.7 s~~
~~FlexAlign (non-tuned)~~	~~11.7 s~~	~~10.4 s~~	~~35.2 s~~	~~11.5 s~~	~~26.9 s~~
~~Movies to pay off~~	~~15~~	~~14~~	~~6~~	~~22~~	~~8~~

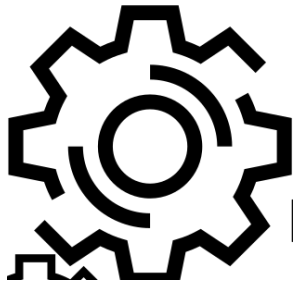
Table 6. Execution time on Testbed 3.

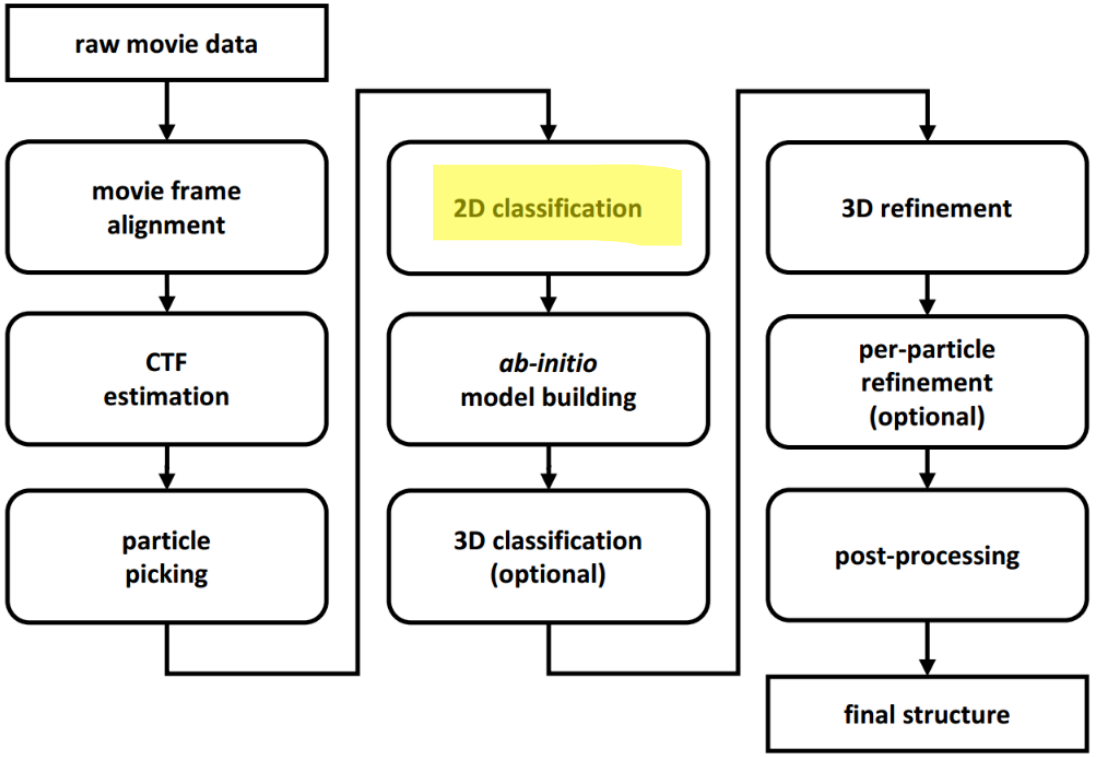
	Falcon	K2	K2 Super	K3	K3 Super
Warp	11.7 s	10 s	14.2 s	13.1 s	15.6 s
FlexAlign ~~(tuned)~~	11.9 s	9.9 s	35.5 s	11.3 s	27.7 s

MUNI FI

Acceleration by …

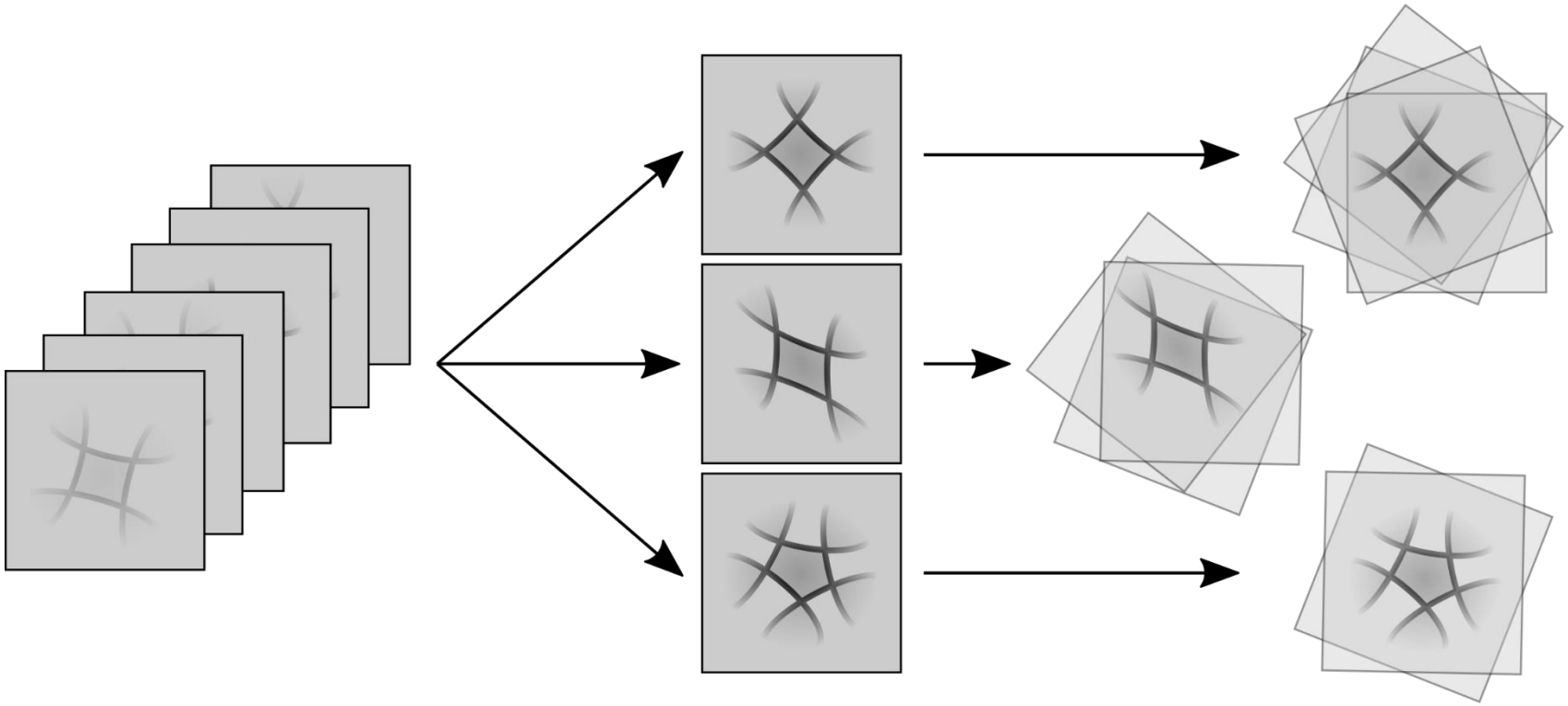
- Using accelerator
  - Sheer brute force
- Faster algorithm
  - More efficient





MUNI FI

Align Significant



[2] JIMÉNEZ-MORENO et al., DeepAlign, a 3D Alignment Method based on Regionalized Deep Learning for Cryo-EM. Journal of Structural Biology. San Diego,USA: Academic Press, 2021, vol. 213, No 2, 14 pp. ISSN 1047-8477. doi:10.1016/j.jsb.2021.107712. Scopus Q2

MUNI FI

Align Significant – DL part

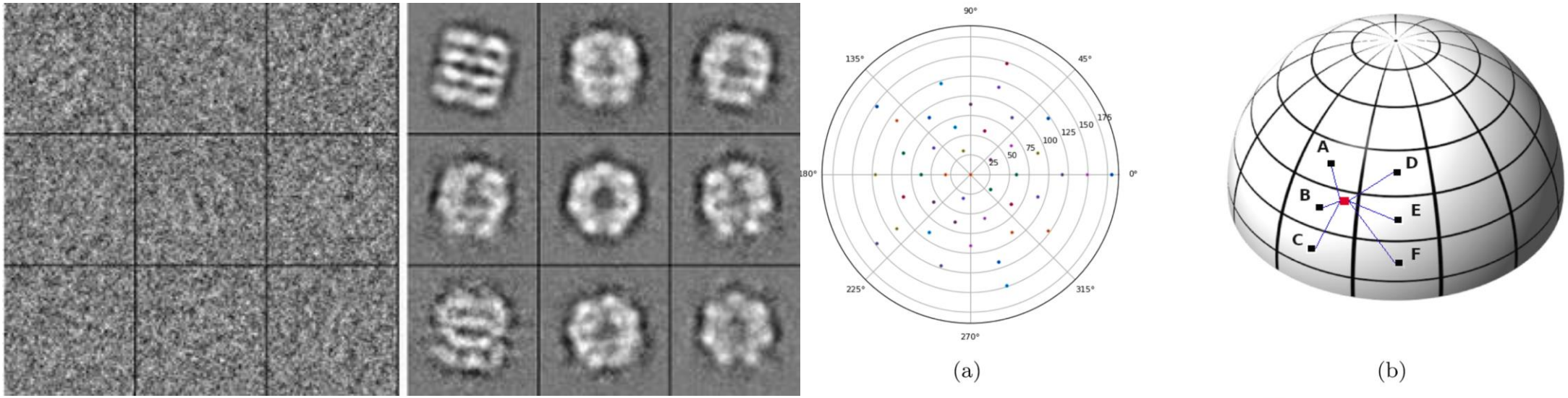


Fig. 2. (a) Top view of region centers shown in dots, example with regions separated 30°. (b) Illustrative example of the labeling for a particle: the distance between the particle (red point) and all the region centers (for clearness just six regions are drawn, A, B, C, D, E and F) is calculated, the minimum distance give the label for the particle (B in this example).

MUNI FI

Align Significant - quality

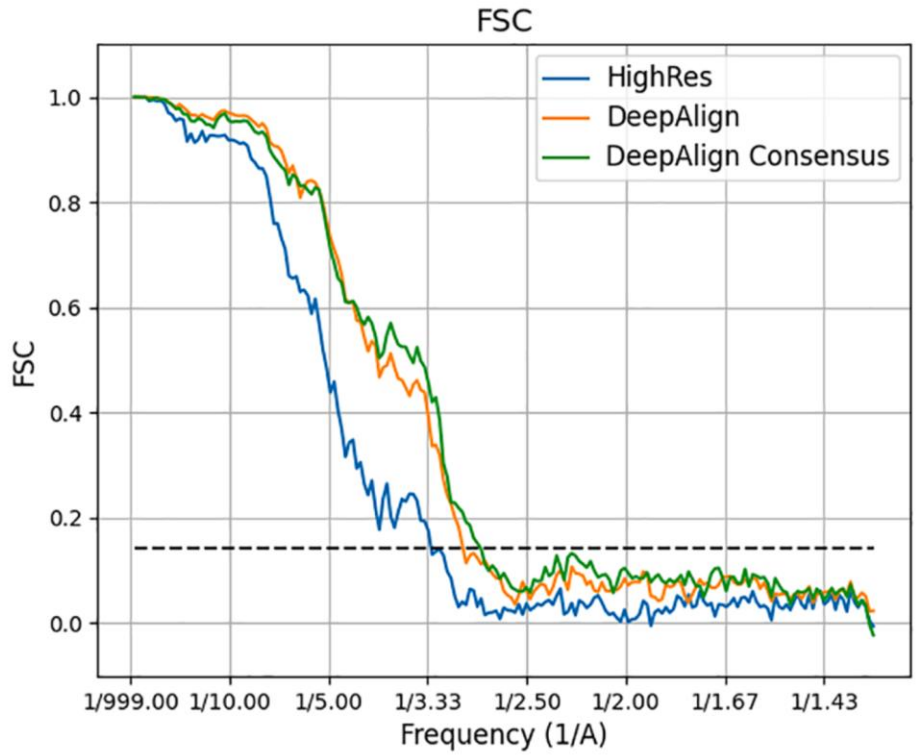


Fig. 10. FSC curves obtained for *proteasome*. Xmipp Highres 3.3 Å, DeepAlign 2.9 Å, and DeepAlign consensus 2.7 Å are compared.

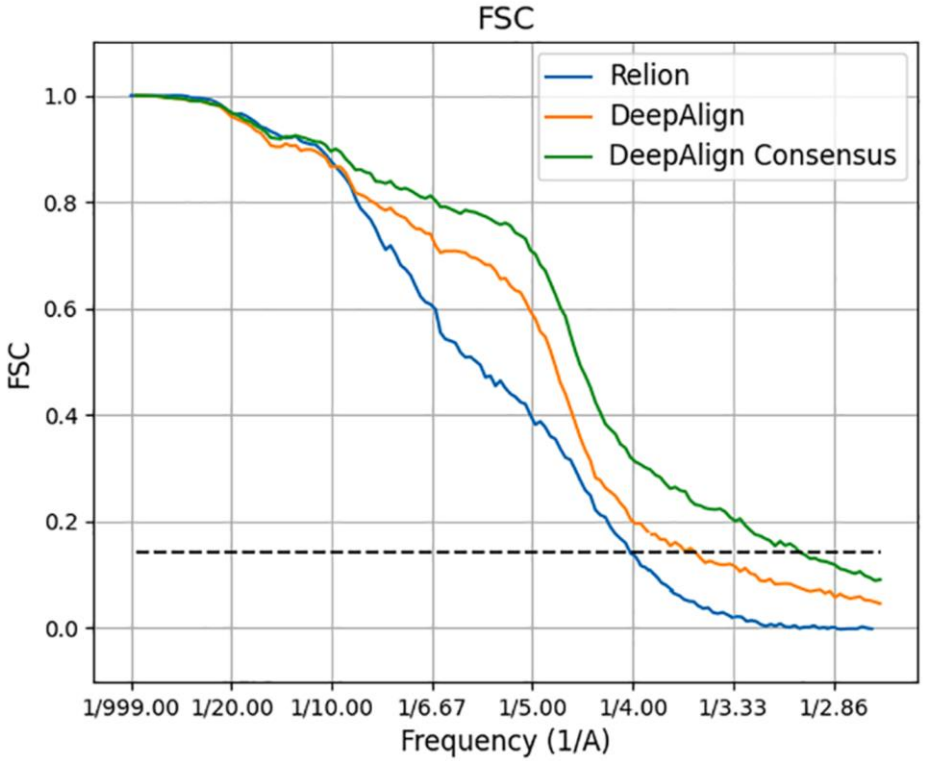


Fig. 6. FSC curves obtained for *ribosome*. Relion 4.0 Å, DeepAlign 3.5 Å, and DeepAlign consensus 2.9 Å are compared.

MUNI FI

Align Significant - quality

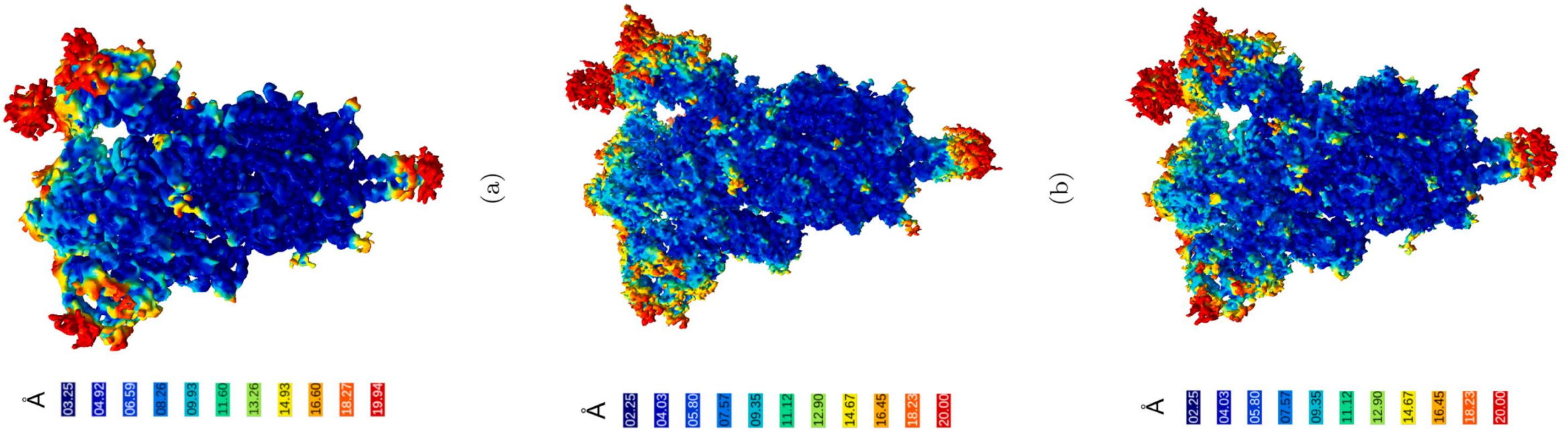


Fig. 15. Local resolution of the reconstructed 3D maps for *SARS-CoV-2 Spike*. (a) CryoSparc, (b) DeepAlign, and (c) DeepAlign consensus.

MUNI FI

Align Significant - quality

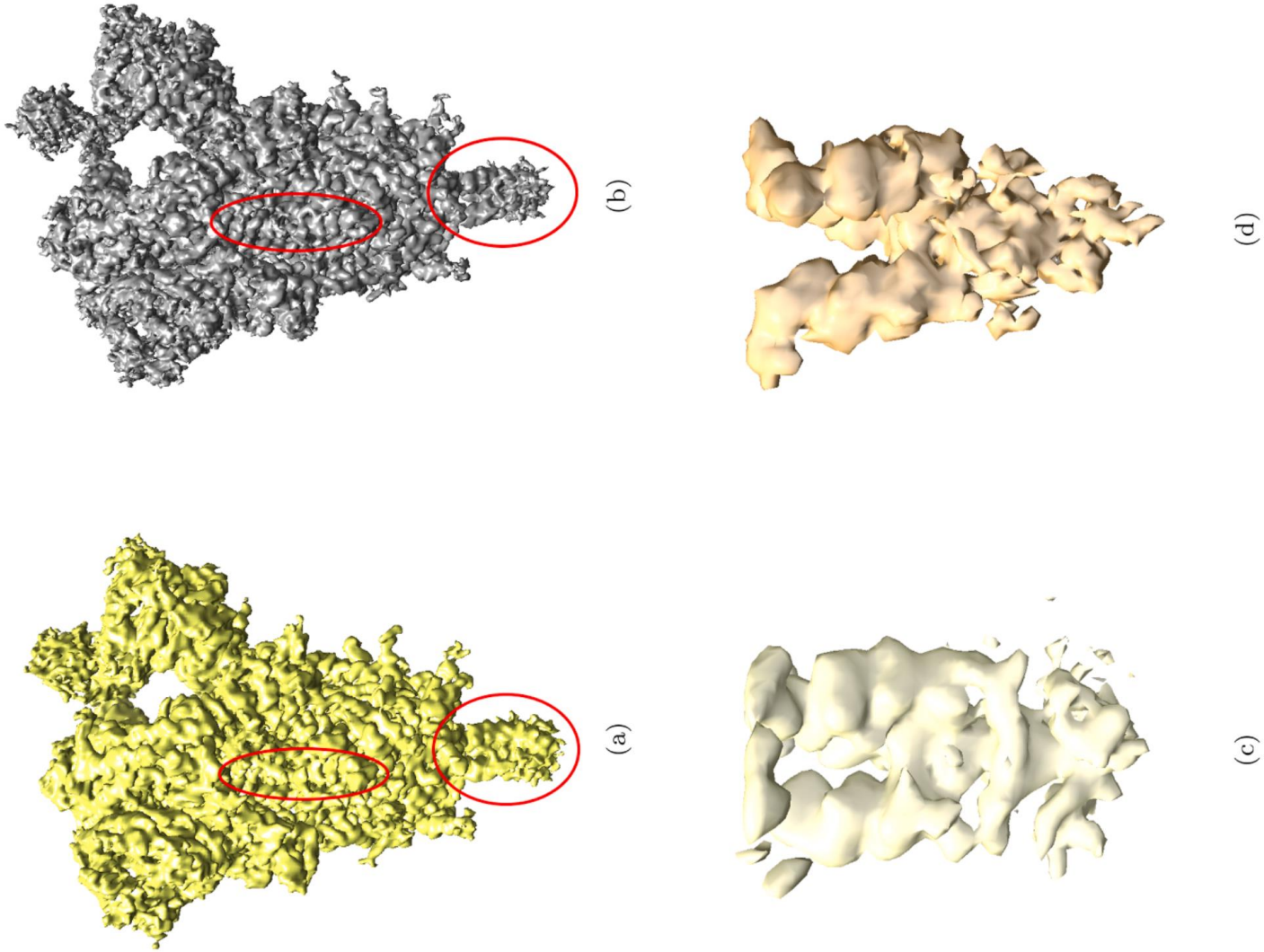


Fig. 17. 3D reconstructed maps for *SARS-CoV-2 Spike*. (a) and (b) Whole 3D maps reconstructed by CryoSparc and DeepAlign, respectively. (c) and (d) Zoom in a specific area showing several helices for CryoSparc and DeepAlign, respectively.

MUNI FI

Align Significant – performance

– [placeholder]

challenging data set. We use the SARS-CoV-2 Spike data set (Melero et al., in press) whose characteristics are: size of 400 × 400 pixels, pixel size of 1.05 Å/pixel, and no symmetry. We considered a distance between regions of 30°, which results in 42 regions, and a target resolution of 4 Å to rescale the input particle images and volume to a size of 314 × 314 pixels. The data set consisted of 36,558 images, from which we

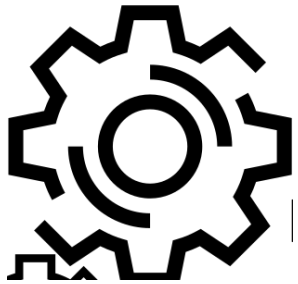
We used 7 GPUs to run DeepAlign with these data. The time required to train one region was, on average, 9 h, so to train the 42 regions we needed 54 h. The prediction time was 3 h, and the final alignment required 20 min, on average, per region, so a total of 2 h were dedicated to this step. The entire process, taking into account some additional steps, took approximately 2 days and a half. These times are higher than

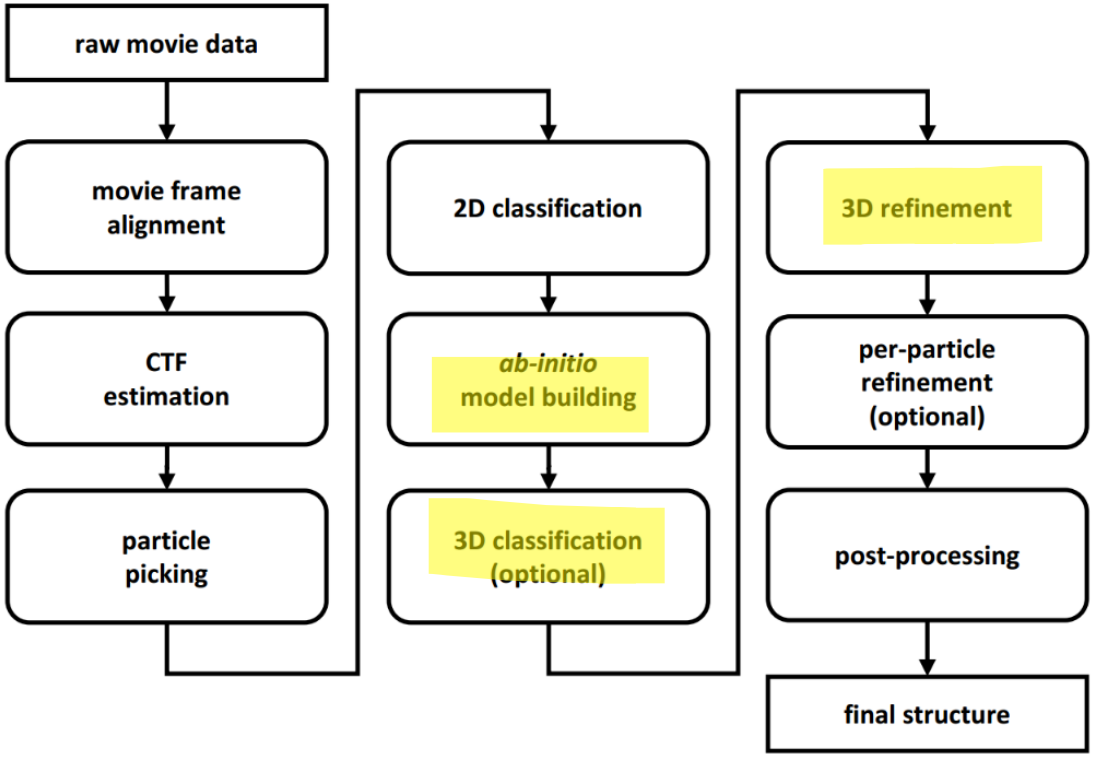
5 regions

MUNI FI

Acceleration by …

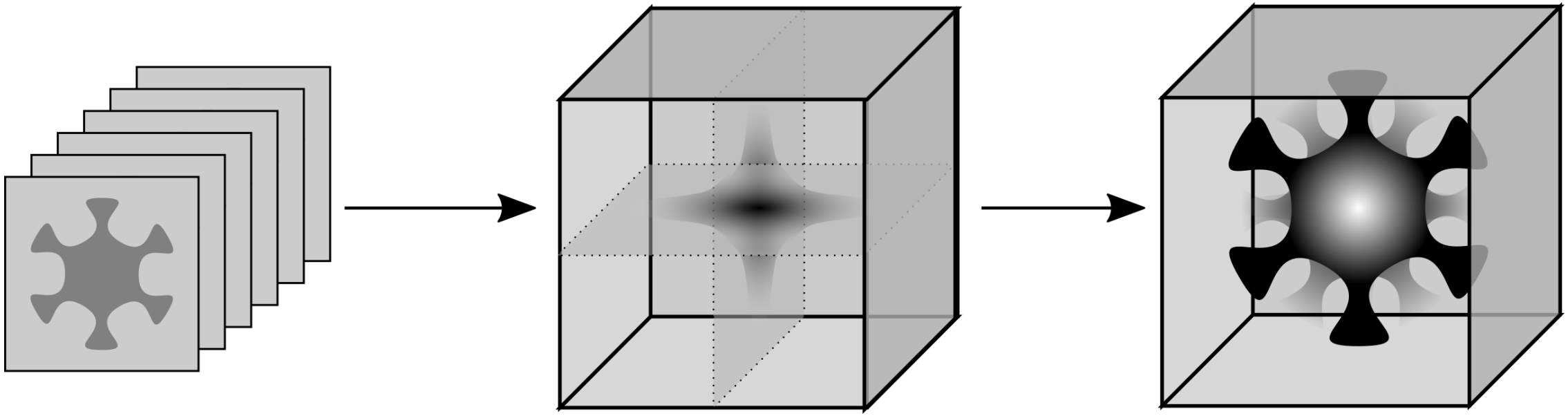
- Faster algorithm
  - More efficient
  - Better fit the HW





MUNI FI

3D Fourier reconstruction



[3] Střelák et al. A GPU acceleration of 3-D Fourier reconstruction in cryo-EM, The International Journal of High Performance Computing Applications. 2019, vol. 33, no. 5, pp. 948–959. Available from DOI:10.1177/1094342019832958. Scopus Q2

MUNI FI

3D Fourier reconstruction

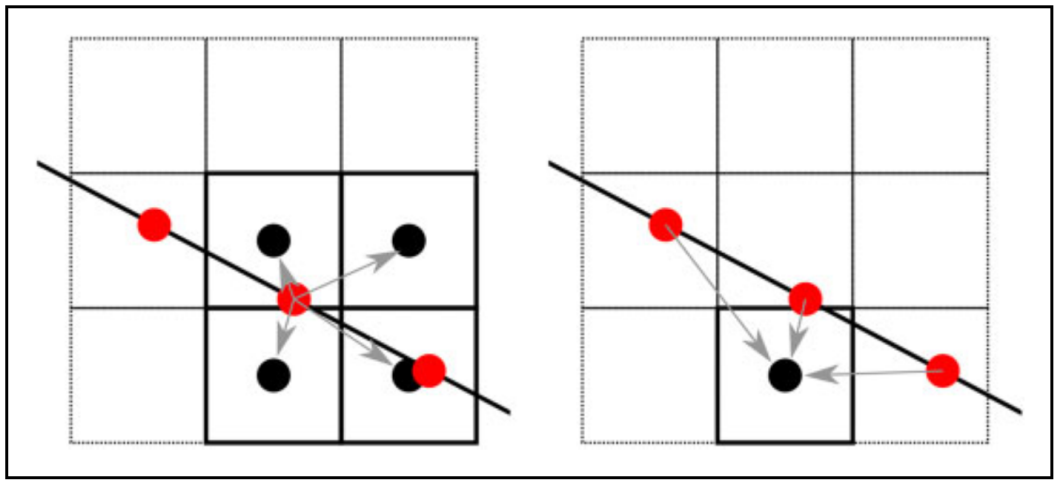


Figure 1. Comparison of the scatter (left) and the gather (right) approach in a cut of the 3-D grid. The solid line represents a sample s, red dots represent pixels, and black dots represent written voxels. With the scatter pattern, the pixels weighted value is written into multiple voxels. With the gather pattern, the voxel value is computed using multiple pixels.

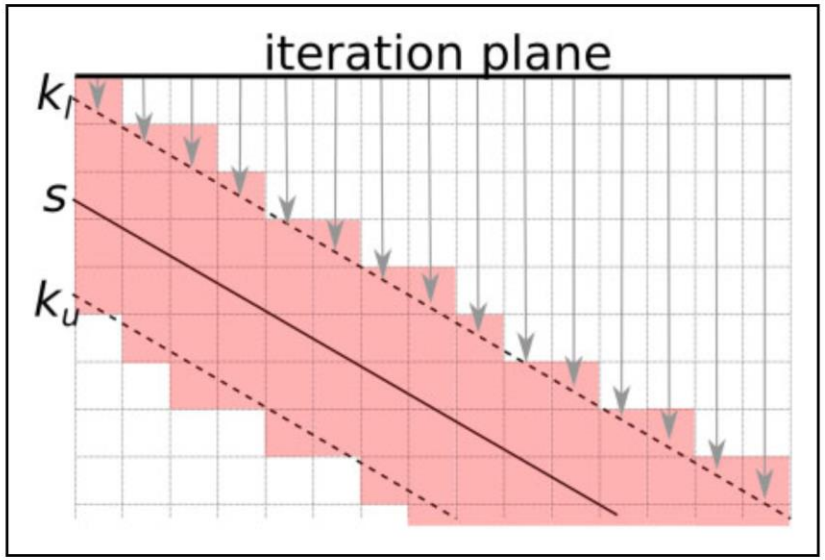


Figure 2. Schematic view of the iteration space in the cut of the 3-D grid. The solid line represents a sample s, and dashed lines represent boundaries of an area affected by the interpolation window. Arrows show computation of the initial iteration in the third dimension (i.e. dimension not iterated at the iteration plane). The updated voxels are emphasized.

MUNI FI

3D Fourier reconstruction – quality

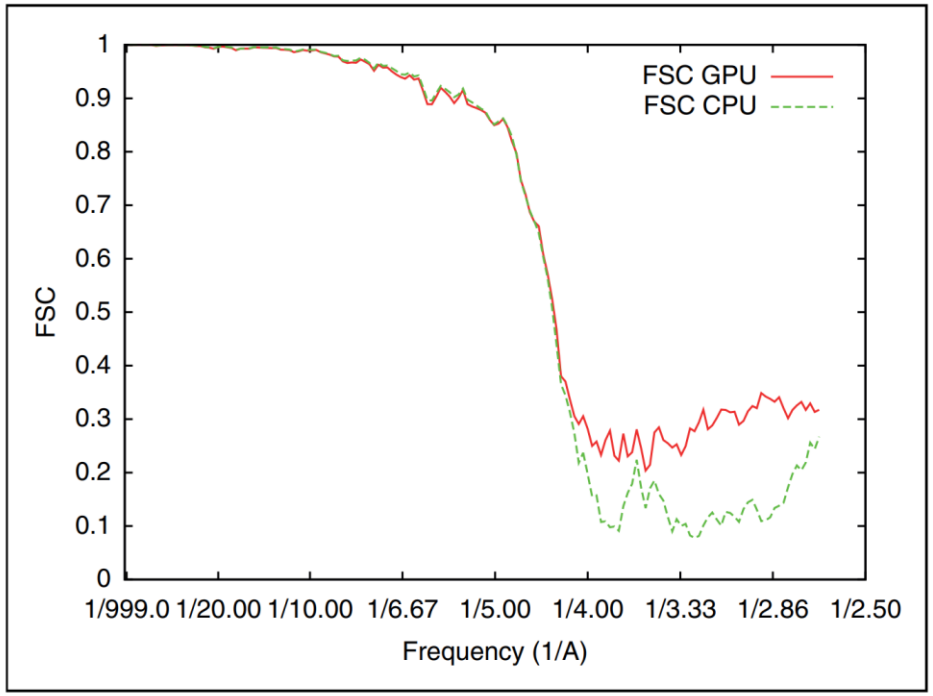


Figure 4. FSC between two halves of samples computed by the original and proposed GPU implementations. FSC: Fourier shell correlation; GPU: graphics processing unit.

MUNI FI

3D Fourier reconstruction – performance

Table 1. Theoretical performance (in single-precision Tflops), memory bandwidth (in GB/s), and power consumption (in Watts) of hardware used in the evaluation.

Processor	Single-precision performance	Memory bandwidth	TDP
2× Xeon E5-2650 v4	0.845	154	210
1× Tesla P100	9519	732	300
4× Tesla P100	38,076	2928	1200
~~GeForce GTX 1070~~	~~5783~~	~~256~~	~~150~~
~~GeForce GTX 750~~	~~1044~~	~~80.2~~	~~55~~
~~GeForce GTX 680~~	~~3090~~	~~192~~	~~195~~

TDP: Thermal Design Power.

Table 5. Performance comparison of the original CPU and our GPU 3-D Fourier reconstruction using different numbers of GPUs. The walltime shows overall application time, the parallel region shows time of the parallelized code of samples insertion into the 3-D grid. The speedup is computed as the relative difference of the walltime.

Configuration	Walltime	Parallel region	Speedup
CPU only	155 min 00 s	150 min	n/a
1× P100	13 min 35 s	12 min 42 s	11.4×
2× P100	8 min 14 s	6 min 50 s	18.8×
4× P100	4 min 53 s	3 min 26 s	31.7×

GPU: graphics processing unit.

MUNI FI

3D Fourier reconstruction – power consumption

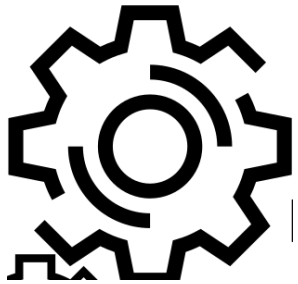
Table 6. Power consumption of CPU and GPU 3-D Fourier reconstruction.

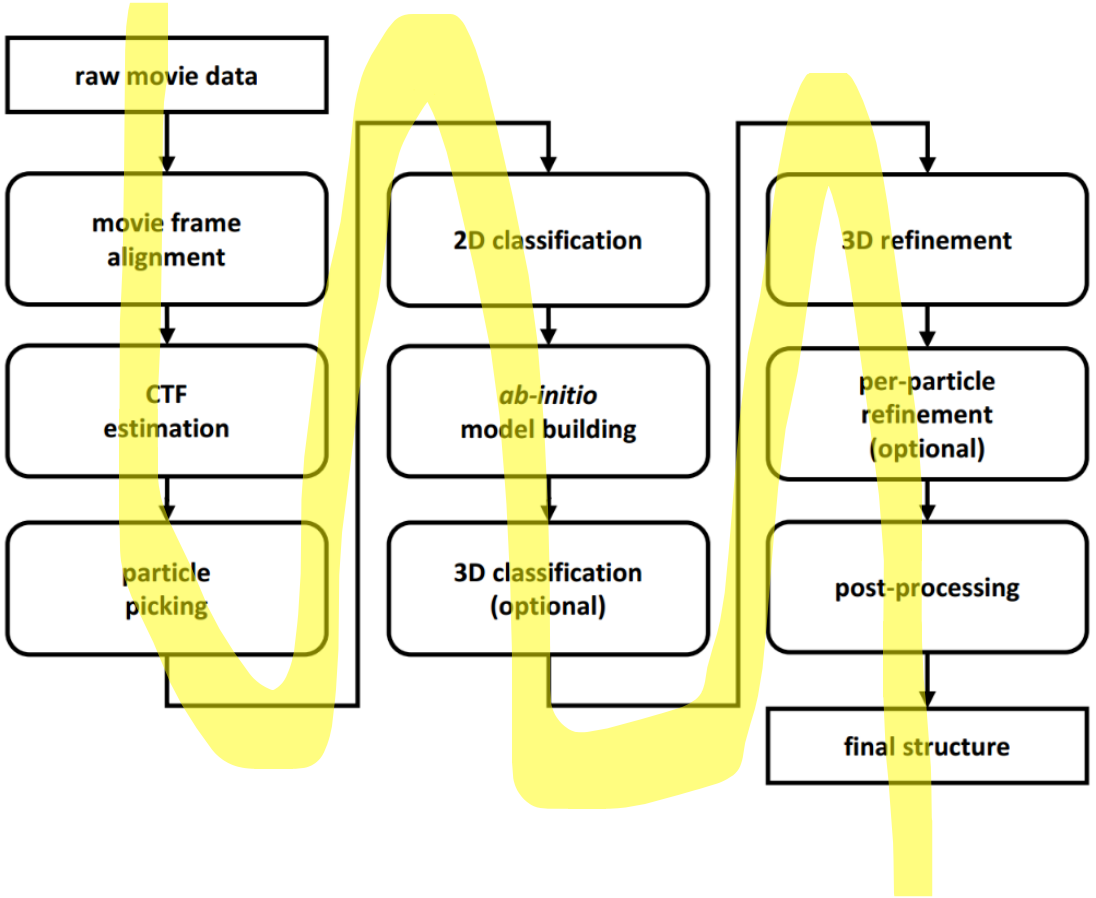
Configuration	Time	Input (W)	Used power (kJ)
CPU only	150 m	206	1845
1× P100	12 min 42 s	253	182.2
2× P100	6 min 50 s	397	159.6
4× P100	3 min 26 s	679	139.9

GPU: graphics processing unit.

Acceleration by …

- Adjusting implementation
  - To HW
  - To Data





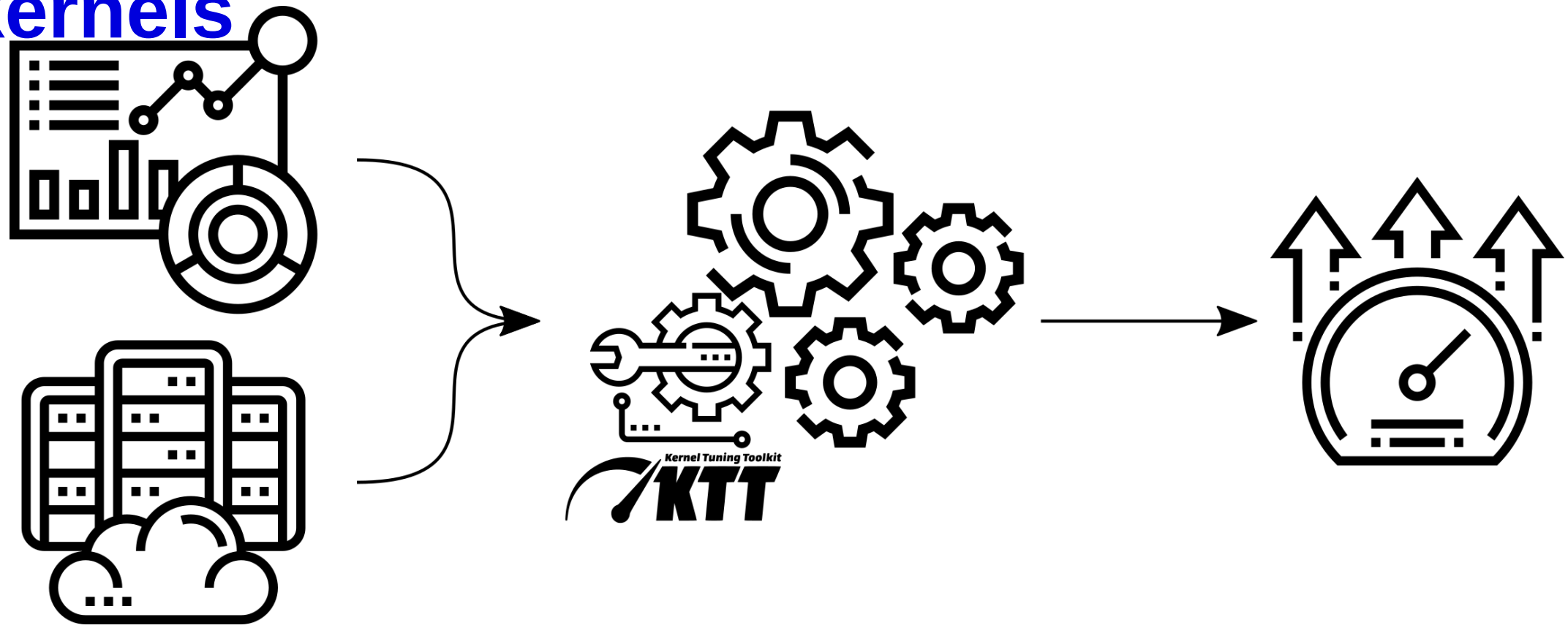
MUNI FI

Autotuning

- Autotuning allows optimizing the application's tuning parameters (properties influencing the application performance) in order to perform the execution more efficiently.

- Offline autotuning
  - before the execution of a tuned code
  - easier to implement
  - does not allow an application to re-tune when its environment changes
- Dynamic autotuning
  - the application can even build the space of different variants during runtime, i. e., it is able to compile tuned kernels during the tuning process

Autotuning of the CUDA and OpenCL kernels



[4] Petrovič et al. A benchmark set of highly-efficient CUDA and OpenCL kernels and its dynamic autotuning with Kernel Tuning Toolkit. Future Generation Computer Systems, Elsevier, 2020, vol. 108, pp. 161-177. ISSN 0167-739X. doi:10.1016/j.future.2020.02.069. Scopus Q1

MUNI FI

Autotuning of the CUDA and OpenCL kernels

Table 3
A list of the benchmarks and the size and dimensionality (i.e., the number of tuning parameters) of their tuning spaces.

Benchmark	Dimensions	Configurations
BiCG	11	5,122
Convolution	10	5,248
Coulomb 3D	8	1,260
GEMM	15	241,600
GEMM batched	11	424
Hotspot	6	480
Transpose	9	10,752
N-body	8	9,408
Reduction	5	175
Fourier	6	360

Autotuning of the CUDA and OpenCL kernels

Table 6

Relative performance of benchmarks ported across GPU architecture
average and standard deviation of the relative performance, worst sh
cannot be executed on a device. 3D Fourier Reconstruction has beer

Benchmark	GPU→GPU		
	avg±stdev	Worst	Failed
BiCG	89.0%±12.3%	57%	1
Convolution	79.4%±14.9%	55%	3
Coulomb 3D	95.8%±6.5%	67%	0
GEMM	83.6%±16.4%	31%	0
GEMM batched	85.4%±17%	37%	0
Hotspot	80.3%±17.5%	46%	3
Transpose	85.0%±21.9%	8%	3
N-body	78.8%±24.2%	2%	3
Reduction	88.4%±24%	12%	3
Fourier	74.5%±30%	31%	0

MUNI FI

Autotuning of the CUDA and OpenCL kernels

Table 8
Performance portability of 3D Fourier reconstruction with 128 × 128 samples. The rows represent GPUs used for offline tuning; the columns represent GPUs used for execution. The percentage shows how performance differs compared to the code using the best combination of tuning parameters (for example, the code tuned for GeForce GTX 1070 and executed on GeForce GTX 750 runs at only 31% of the speed of the code both tuned and executed on GeForce GTX 750).

	2080Ti	1070	750	680
2080Ti	100%	99%	31%	49%
1070	99%	100%	31%	50%
750	43%	67%	100%	94%
680	60%	72%	71%	100%

Table 9
Performance portability on GeForce GTX1070. The rows represent samples resolution used for offline tuning, the columns represent samples resolution used for execution. The percentage shows relative performance compared to the code autotuned for the used resolution.

	128 × 128	91 × 91	64 × 64	50 × 50	32 × 32
128 × 128	100%	100%	77%	70%	32%
91 × 91	100%	100%	76%	68%	33%
64 × 64	94%	94%	100%	91%	67%
50 × 50	79%	78%	98%	100%	86%
32 × 32	65%	67%	80%	92%	100%

MUNI FI

Autotuning of the CUDA and OpenCL kernels

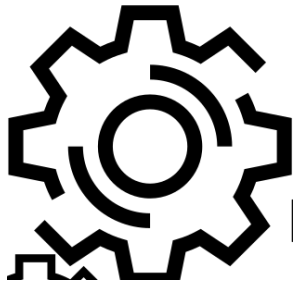
Table 10

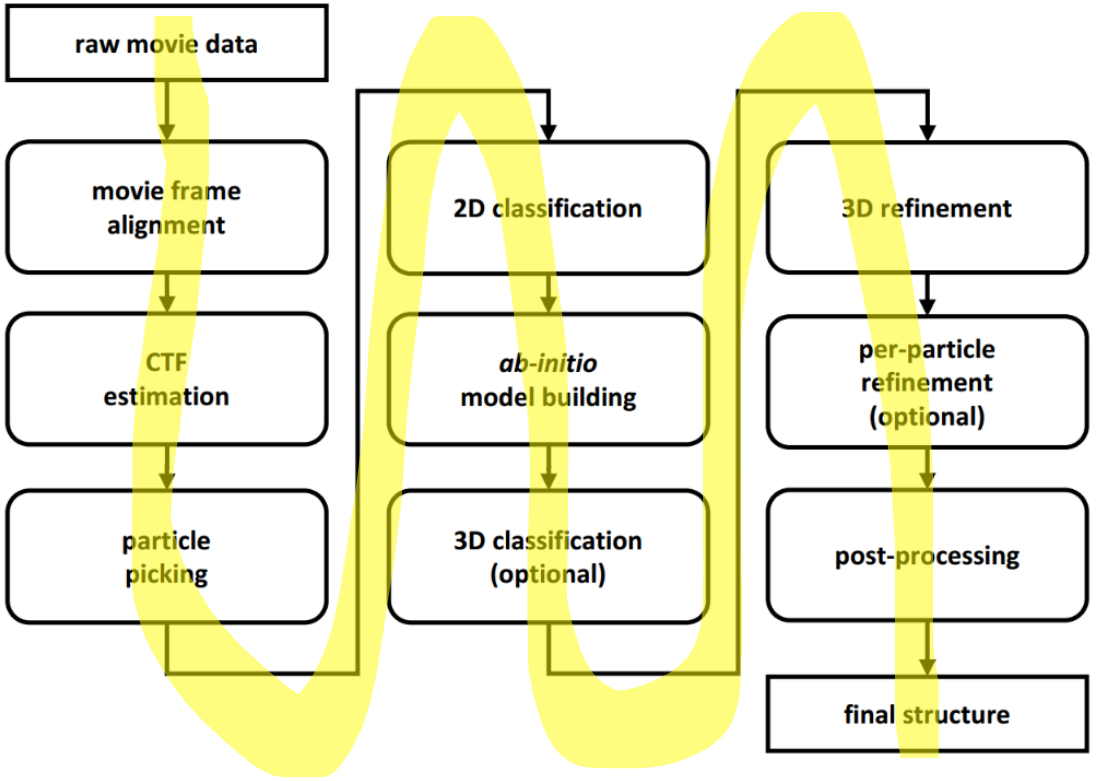
The relative performance of dynamically-tuned 3D Fourier reconstruction. The best runtime is measured with ōrāculum, i.e., the fastest kernel is selected immediately, and no tuning is performed. The relative performance of tuning with searching 50 configurations and with searching the entire tuning space is measured as a percentage of the best runtime. Results for "tuning 50" are shown as an average and standard deviation, whereas other results are shown as an average only (their performance is very stable across multiple executions).

	Best runtime	Tuning 50	Tuning full
2080Ti	1 m 40 s	88% ± 3%	54%
1070	5 m 49 s	96% ± 2%	79%
750	16 m 59 s	92% ± 4%	72%
680	15 m 12 s	94% ± 2%	75%

Acceleration by …

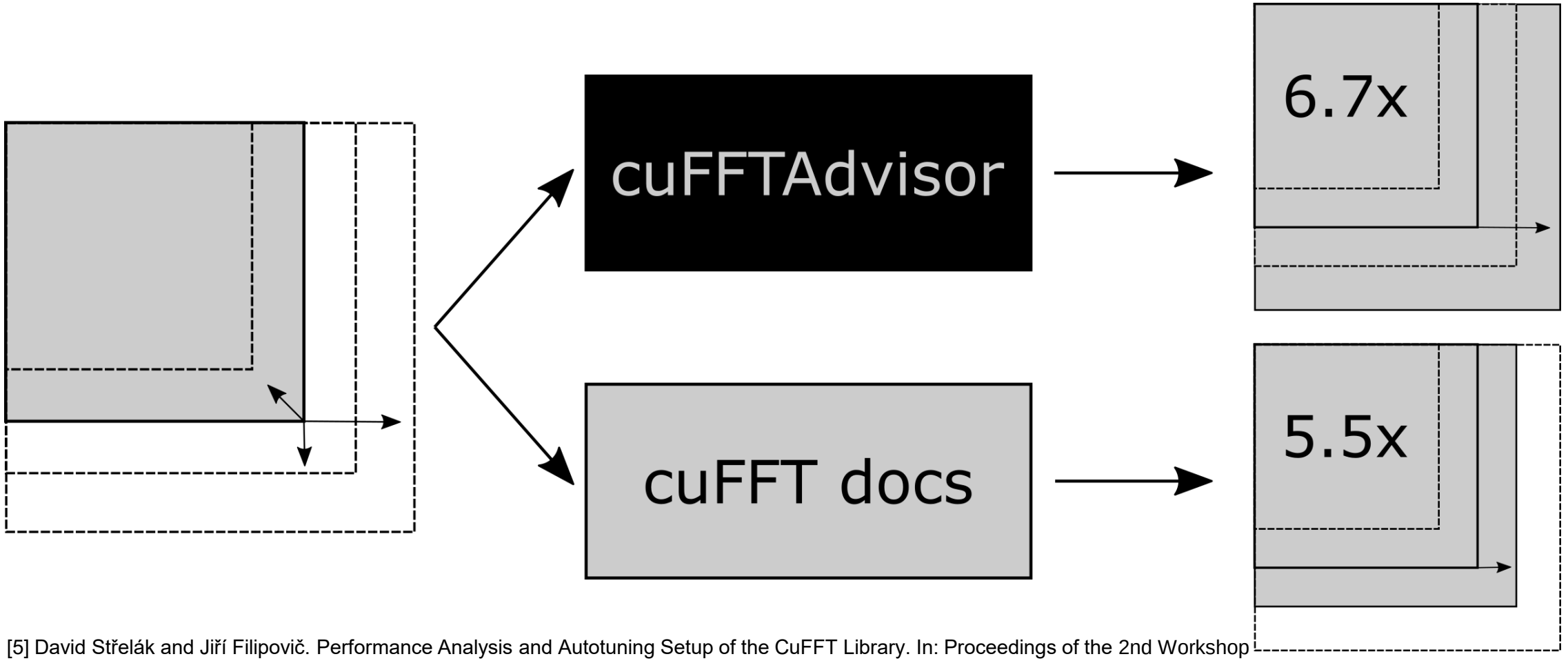
- Adjusting invocations
  - Necessary for close-sourced libraries





MUNI FI

cuFFTAdvisor



[5] David Střelák and Jiří Filipovič. Performance Analysis and Autotuning Setup of the CuFFT Library. In: Proceedings of the 2nd Workshop on AutotuniNg and ADaptivity AppRoaches for Energy Efficient HPC Systems. ANDARE '18. Limassol, Cyprus: Association for Computing Machinery, 2018. isbn: 9781450365918. doi: 10.1145/3295816.3295817

MUNI FI

cuffAdvisor

Table 4. Execution time on Testbed 1.

	Falcon	K2	K2 Super	K3	K3 Super
MotionCor2	4.6 s	4.3 s	15.7 s	5.0 s	13.1 s
FlexAlign (tuned)	9.2 s	7.6 s	25.6 s	8.8 s	20.5 s
FlexAlign (autotuning)	49.2 s	34.4 s	71.9 s	59.3 s	72.2 s
FlexAlign (non-tuned)	10.8 s	9.1 s	31.5 s	10.7 s	22.9 s
Movies to pay-off	25	18	8	27	22

Table 5. Execution time on Testbed 2.

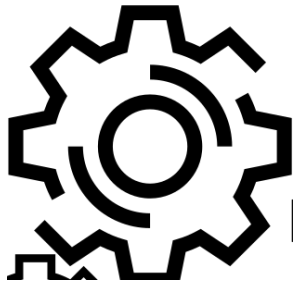
	Falcon	K2	K2 Super	K3	K3 Super
MotionCor2	5.5 s	5.2 s	20.3 s	5.9 s	15.7 s
FlexAlign (tuned)	9.0 s	8.2 s	27.3 s	9.3 s	21.1 s
FlexAlign (autotuning)	47.8 s	38.7 s	73.3 s	56.2 s	65.7 s
FlexAlign (non-tuned)	11.7 s	10.4 s	35.2 s	11.5 s	26.9 s
Movies to pay-off	15	14	6	22	8

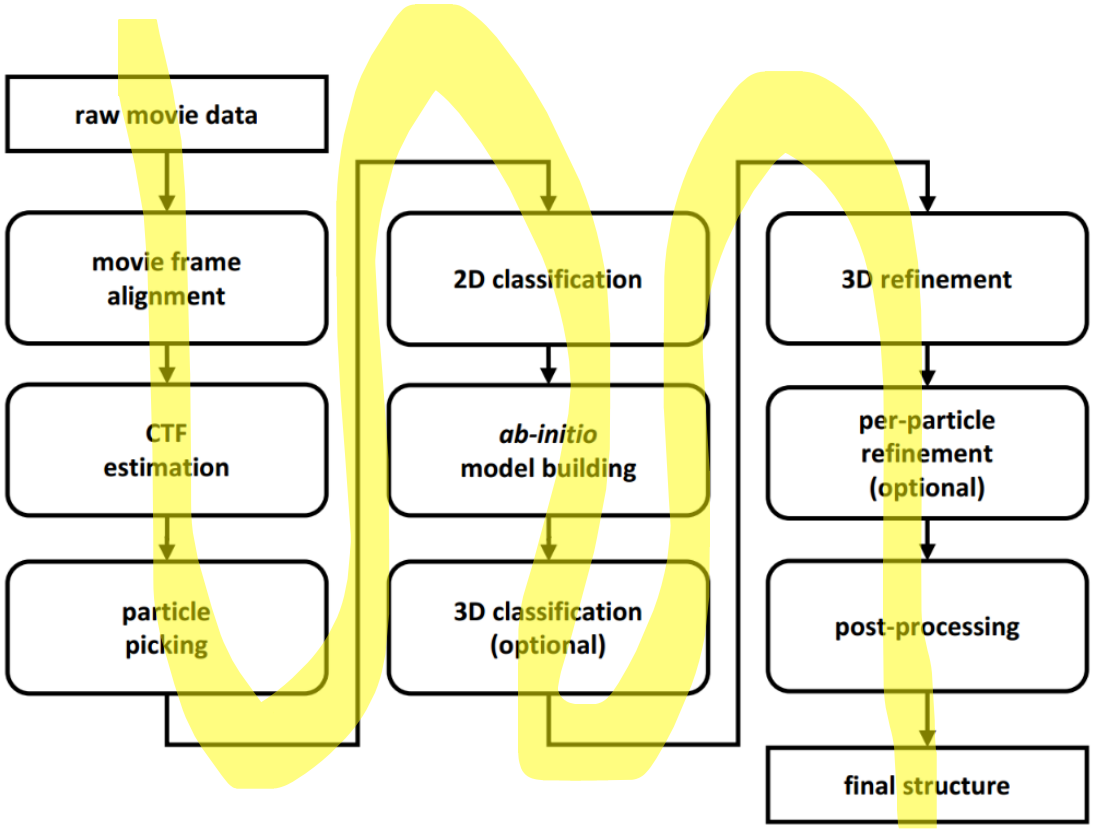
Table 6. Execution time on Testbed 3.

	Falcon	K2	K2 Super	K3	K3 Super
Warp	11.7 s	10 s	14.2 s	13.1 s	15.6 s
FlexAlign (tuned)	11.9 s	9.9 s	35.5 s	11.3 s	27.7 s

Acceleration by …

- Utilizing all resources
  - Heterogeneous computing





MUNI FI

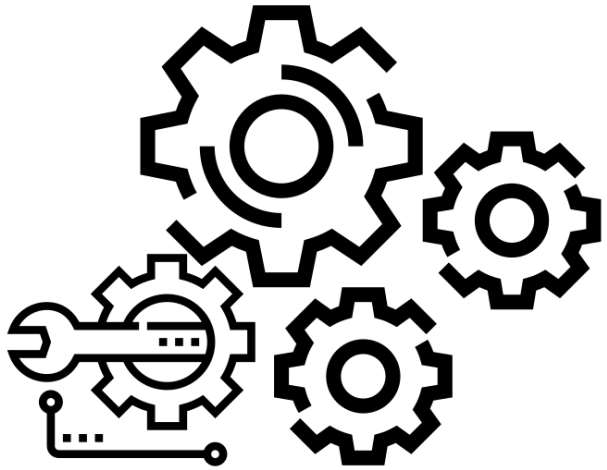
Task-based runtime systems

- Program is defined as a set of individual tasks that can be executed in parallel on various hardware resources
  - respect the execution order defined by their data dependencies
- The tasks are scheduled to different processors available in the system
  - the task-based framework automatically handles data movement between memory spaces
- A task can have multiple implementations
  - for different hardware
  - for other properties

Acceleration by …

- Using accelerator
  - Sheer brute force
- Faster algorithm
  - More efficient
  - Better fit the HW
- Adjusting invocations
  - Necessary for close-sourced libraries
- Adjusting implementation
  - To HW
  - To Data
- Utilizing all resources
  - Heterogeneous computing

- Gotta Catch 'Em All



Umpalumpa framework

- Aims to manage complex workloads on heterogeneous computers
- Combines three aspects that ease programming and optimize code performance

1. it implements data-centric design, where data are described by their physical properties (e. g., location in memory, size) and logical properties (e. g., dimensionality, shape, padding);
2. utilizes task-based parallelism to schedule tasks on heterogeneous nodes.
3. tasks can be dynamically autotuned on a source code level

Umpalumpa – data centric design

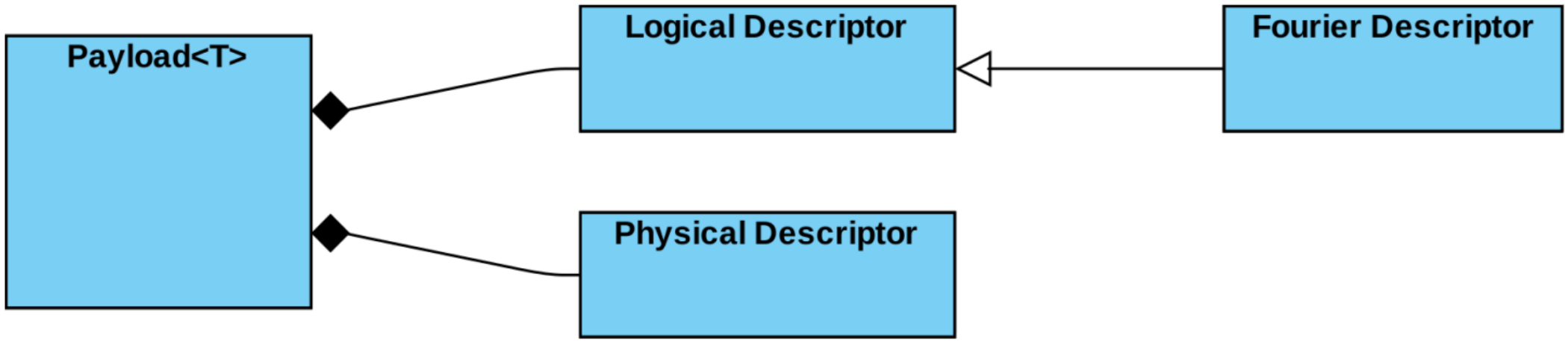


Figure 1: UML of the Payload, the Logical Descriptor and the Physical Descriptor

MUNI FI

Umpalumpa – Algorithms & Strategies

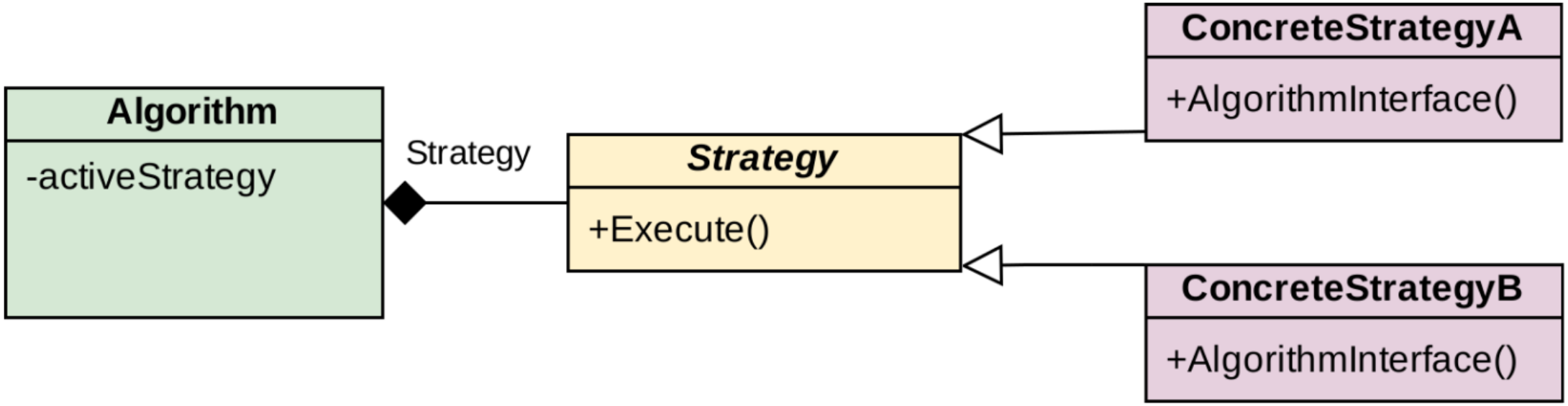


Figure 2: UML of the Algorithm with different Strategies

MUNI FI

Umpalumpa – FlexAlign

Algorithm 1 FlexAlign pseudocode, Umpalumpa version.

Input $F, batch$

```
1:  frame_payloads ⇐ {}
2:  for j = 0; j < size(F); j+ = batch do
3:      frame_payloads ← create_payload(batch)
4:  end for
5:  frames_fd ← {}
6:  shifts ← {}
7:  for j = 0; j < size(F); j+ = batch do
8:      frame ← load_frame(j, batch, frame_payloads)
9:      frame_fd ← convert_to_fd(frame)
10:     frames_fd ← crop(frame_fd)
11:     for i = 0; i ≤ j; i+ = batch do
12:         correlation ← correlate(frames_fd[i], frames_fd[j])
13:         corr_func ← convert_from_fd(correlation)
14:         shifts ← find_max(corr_func)
15:     end for
16: end for
17: for all shift ∈ shifts do
18:     // extract shift from the Payload
19: end for
```

Algorithm 2 Method which converts batch of frames to the Fourier domain

Input $frames$

```
1:  in_payload ← Payload(FourierDescriptor(frames), frames.pd)
2:  out_payload ← {
3:      ld = FourierDescriptor(frames, FourierSpaceDescriptor())
4:      pd = create_pd(ld)
5:      return Payload(ld, pd)
6:  }
7:  in ← PayloadWrapper(in_payload)
8:  out ← PayloadWrapper(out_payload)
9:  alg ← get_fft_alg()
10: if alg is not initialized then
11:     settings ← Settings(out_of_place, forward)
12:     alg.init(out, in, settings)
13: end if
14: alg.execute(out, in)
15: return out_payload
```


MUNI FI

Umpalumpa – FlexAlign

Table 1: HW used for the performance testing.

CPU	AMD EPYC 7402 24-Core Processor
GPU	2 × NVIDIA GeForce RTX 3090 (24 GB)
CUDA/driver	11.4 / 470.103.01
RAM	64 GB DDR4 @ 3.2 GHz

Table 3: Runtime of FlexAlign in seconds, single GPU. Speedup is computed relative to Xmipp implementation.

	Xmipp	**Umpalumpa, One GPU worker**	**Speedup**	**Umpalumpa, All workers**	**Speedup**
Falcon	7,5	5,7	133 %	6,5	115 %
K2	6,4	4,7	136 %	5,1	126 %
K2 super	23,3	21,2	110 %	15,1	154 %
K3	7,3	5,7	128 %	6,2	118 %
K3 super	19,6	13,2	148 %	15,0	131 %

Table 4: Performance of Umpalumpa FlexAlign, all CPUs, batch 5 / 1. Relative performance is computed as a fraction of Umpalumpa execution on a single GPU.

	Wall time (s)	**Rel. performance**
Falcon	27,5 / 12,1	27 % / 62 %
K2	19,2 / 8,6	33 % / 74 %
K2 super	89,7 / 38,0	26 % / 61 %
K3	28,3 / 10,9	26 % / 67 %
K3 super	86,8 / 24,1	23 % / 81 %

Umpalumpa – Fourier Reconstruction

Algorithm 3 Fourier Reconstruction pseudocode, Umpalumpa version.

Input $projections, batch$

1: $volume \leftarrow$ create_volume_payload()
2: **for** $i = 0;\ i <$ size($projections$); $i+ = batch$ **do**
3: $proj\_payload \leftarrow$ create_payload($batch$)
4: $aux\_payload \leftarrow$ create_payload($batch$) // holds auxiliary data, e. g. projection orientation
5: $proj \leftarrow$ load_projection($i, batch, proj\_payload$)
6: $aux \leftarrow$ load_aux($i, batch, aux\_payload$)
7: $proj\_fd \leftarrow$ convert_to_fd($proj$)
8: $proj\_fd \leftarrow$ crop($proj\_fd$)
9: insert_to_volume($proj\_fd, aux, volume$)
10: **end for**
11: get_insert_alg().synchronize()

```cpp
void FourierReconstruction<T>::Execute(const umpalumpa::data::Size &imgSize,
  size_t noOfSymmetries,
  size_t batchSize,
  const umpalumpa::fourier_reconstruction::Settings::Type &type,
  const umpalumpa::fourier_reconstruction::Settings::Interpolation &interpolation)
{
  assert(imgSize.x % 2 == 0);// we can process only odd size of the images
  using umpalumpa::fourier_reconstruction::Settings;
  using umpalumpa::fourier_reconstruction::Settings;

  auto imgBatchSize = umpalumpa::data::Size(imgSize.x, imgSize.y, 1, batchSize);
  auto volumeSize = umpalumpa::data::Size(imgSize.x + 1, imgSize.y + 1, imgSize.y + 1, 1);
  auto imgCroppedBatchSize = umpalumpa::data::Size(
    imgSize.x / 2, imgSize.y, 1, batchSize);// This should probably be .x / 2 + 1 (i.e. normal FFT
                                            // size), but in Xmipp it's like that
  auto traverseSpaceBatchSize = umpalumpa::data::Size(1, 1, 1, batchSize * noOfSymmetries);
  auto settings = umpalumpa::fourier_reconstruction::Settings{};
  settings.SetType(type);
  settings.SetInterpolation(interpolation);

  spdlog::info(
    "\nRunning Fourier Reconstruction.\nImage size: {}*{} ({})\nBatch: {}\nSymmetries: "
    "{}\nInterpolation type: {}\nInterpolation coefficient type: {}",
    imgSize.x,
    imgSize.y,
    imgSize.n,
    batchSize,
    noOfSymmetries,
    settings.GetType() == Settings::Type::kPrecise ? "immediate interpolation"
                                                   : "delayed interpolation",
    settings.GetInterpolation() == Settings::Interpolation::kDynamic ? "dynamic computation"
                                                                     : "precomputed table");

  auto symmetries = GenerateSymmetries(noOfSymmetries);
  auto filter = CreatePayloadFilter(imgCroppedBatchSize);
  auto volume = CreatePayloadVolume(volumeSize);
  auto weight = CreatePayloadWeight(volumeSize);
  // FIXME init volume and weight to 0
  auto table = CreatePayloadBlobTable(settings);

  for (size_t i = 0; i < imgSize.n; i += batchSize) {
    auto name = std::to_string(i) + "-" + std::to_string(i + batchSize - 1);
    spdlog::debug("Loop {}", name);
    auto img = CreatePayloadImage(imgBatchSize, name);
    GenerateData(i, img);
    auto space = CreatePayloadTraverseSpace(traverseSpaceBatchSize, name);
    GenerateTraverseSpaces(imgCroppedBatchSize, volumeSize, space, symmetries, settings);
    auto fft = ConvertToFFT(img, name);
    auto croppedFFT = Crop(fft, filter, name);
    InsertToVolume(croppedFFT, volume, weight, space, table, settings);
    RemovePD(img.dataInfo);
    RemovePD(fft.dataInfo);
    RemovePD(space.dataInfo);
    RemovePD(croppedFFT.dataInfo);
    OptionalSynch();
  }

  GetFRAlg().Synchronize();// wait till the work is done

  // Show results
  // Print(volume, "Volume data");
  // Print(weight, "Weight data");

  RemovePD(table.dataInfo);
  RemovePD(weight.dataInfo);
  RemovePD(volume.dataInfo);
  RemovePD(filter.dataInfo);
}
```


MUNI FI

Umpalumpa – Fourier Reconstruction

Barracuda

- OS: Ubuntu 18.04.2 LTS
- RAM: 32 GiB @ 2666 MHz
- CPU: Intel® Core™ i7-8700 @ 3.20 GHz
- GPU: GeForce GTX 1070 (8 GiB)
- SSD: Samsung NVMe SM981/PM981

Supercuda

- OS: Ubuntu 18.04.2 LTS
- RAM: 32 GiB @ 2666 MHz
- CPU: Intel® Core™ i7-8700 @ 3.20 GHz
- GPU: GeForce RTX 2080 (8 GiB)
- SSD: Samsung NVMe SM981/PM981

jsem naměřil a komentuji je. Vstupní dataset čítá 31 547 snímků viru (Brome mosaic virus) [12], každý o rozměrech 83×83 pixelů. V testech se symetrií byl každý snímek umístěn 60krát, podle icosahedrální symetrie. Bez symetrie byly snímky umístěny pouze jednou.

MUNI FI

Umpalumpa – Fourier Reconstruction

Tabulka 3: Výsledky měření na jednom stroji, bez symetrií a bez CPU implementace codeletu rekonstrukce. Údaje v sekundách.

	Původní	eager	random	ws	lws	dm	dmda	dmdas	dmdasd
Barracuda	8,18 ± 0,08 %	4,49 ± 0,70 %	5,21 ± 0,80 %	4,46 ± 0,63 %	6,64 ± 0,41 %	4,51 ± 0,75 %	4,54 ± 0,44 %	4,45 ± 0,42 %	4,51 ± 0,72 %
Supercuda	8,10 ± 0,06 %	4,54 ± 0,39 %	5,26 ± 0,63 %	4,55 ± 0,39 %	5,97 ± 0,27 %	4,57 ± 0,70 %	4,61 ± 0,54 %	4,57 ± 0,47 %	4,55 ± 0,34 %

Tabulka 4: Výsledky měření na jednom stroji, se symetriemi a bez CPU implementace codeletu rekonstrukce. Údaje v sekundách.

	Původní	eager	random	ws	lws	dm	dmda	dmdas	dmdasd
Barracuda	98,23 ± 0,05 %	96,91 ± 0,01 %	99,01 ± 0,05 %	96,81 ± 0,01 %	97,37 ± 0,07 %	98,73 ± 0,03 %	98,74 ± 0,03 %	98,73 ± 0,04 %	98,68 ± 0,03 %
Supercuda	38,40 ± 0,02 %	39,70 ± 0,04 %	41,10 ± 0,07 %	39,79 ± 0,02 %	39,90 ± 0,02 %	40,74 ± 0,04 %	40,77 ± 0,04 %	40,82 ± 0,05 %	40,80 ± 0,05 %

MUNI FI

Umpalumpa – Fourier Reconstruction

Table 5: Runtime of Fourier Reconstruction in seconds, entire machine. Speedup is computed relative to Xmipp implementation

Resolution	Symmetries	Projections	Xmipp	Umpalumpa	Speedup
64×64	78	100 000	43,6	6,3	696 %
128×128	78	30 000	14,4	7,6	190 %
256×256	78	10 000	12,2	11,7	104 %
512×512	78	10 000	46,0	39,4	117 %

Summary

Cryo-EM

- several performance critical

  algorithms have been accelerated
  - increased productivity
  - quality consensus
- all contributions are open-source

  and used by the community via

  Xmipp suite*

* https://github.com/I2PC/xmipp

HPC

- three novel tools

cuFFTAdvisor
- open-source*
- its core functionality might be added directly to cuFFT

KTT
- open-source*
- introduced dynamic autotuning

Umpalumpa
- open-source*
- combining tasks with autotuning

* https://github.com/HiPerCoRe/cuFFTAdvisor

https://github.com/HiPerCoRe/KTT

MUNI FI

Thank you for your attention

[1] **STŘELÁK, David**, Jiří FILIPOVIČ, Amaya JIMÉNEZ-MORENO, José María Carazo and Carlos Óscar S. SORZANO. “FlexAlign: an accurate and fast algorithm for movie alignment in Cryo-Electron Microscopy". Special Issue of Electronics "FPGA/GPU Acceleration of Biomedical Engineering Applications" (2020). ISSN: 2079-9292. Scopus Q2

[2] JIMÉNEZ-MORENO, Amaya, **David STŘELÁK**, Jiří FILIPOVIČ, José María CARAZO and Carlos Óscar S. SORZANO. DeepAlign, a 3D Alignment Method based on Regionalized Deep Learning for Cryo-EM. Journal of Structural Biology. San Diego,USA: Academic Press, 2021, vol. 213, No 2, 14 pp. ISSN 1047-8477. doi:10.1016/j.jsb.2021.107712. Scopus Q2

[3] **STŘELÁK, David**, Carlos Óscar S. SORZANO, José María CARAZO and Jiří FILIPOVIČ. A GPU acceleration of 3-D Fourier reconstruction in cryo-EM. The International Journal of High Performance Computing Applications, SAGE Publishing, 2019, vol. 33, No 5, p. 948-959. ISSN 1094-3420. doi:10.1177/1094342019832958. Scopus Q2

[4] PETROVIČ, Filip, **David STŘELÁK**, Jana HOZZOVÁ, Jaroslav OĽHA, Richard TREMBECKÝ, Siegfried BENKNER and Jiří FILIPOVIČ. A benchmark set of highly-efficient CUDA and OpenCL kernels and its dynamic autotuning with Kernel Tuning Toolkit. Future Generation Computer Systems, Elsevier, 2020, vol. 108, p. 161-177. ISSN 0167-739X. doi:10.1016/j.future.2020.02.069. Scopus Q1

[5] **STŘELÁK, David** and Jiří FILIPOVIČ. Performance analysis and autotuning setup of the cuFFT library. In ACM International Conference Proceeding Series. Limassol, Cyprus: ACM, 2018., 6 pp. ISBN 978-1-4503-6591-8. doi:10.1145/3295816.3295817.

[6] **STŘELÁK, David**, Amaya JIMÉNEZ-MORENO, José VILAS, Erney RAMÍREZ-APORTELA, Ruben SÁNCHEZ-GARCÍA, David MALUENDA, Javier VARGAS, David HERREROS, Estrella FERNÁNDEZ-GIMÉNEZ, Federico DE ISIDRO-GÓMEZ, Jan HORÁČEK, David MYŠKA, Martin HORÁČEK, Pablo CONESA, Yunior FONSECA-REYNA, Jorge JIMÉNEZ, Marta MARTÍNEZ, Mohamad HARASTANI, Slavica JONIĆ, Jiří FILIPOVIČ, Roberto MARABINI, José CARAZO and Carlos SORZANO. Advances in Xmipp for Cryo–Electron Microscopy: From Xmipp to Scipion. Molecules. Mayer und Muller, 2021, vol. 26, No 20, p. 1-14. ISSN 1420-3049. doi:10.3390/molecules26206224. Scopus Q2

[7] **STŘELÁK, David**, David MYSKA, Filip Petrovic, Jan Polak, Jaroslav Olha and Jiri Filipovic . Umpalumpa: a framework for efficient execution of complex image processing workloads on heterogeneous nodes. Computing (Submitted, Scopus Q2)

In addition, I have co-authored 5 other articles and one book chapter

MUNI FI

Sources

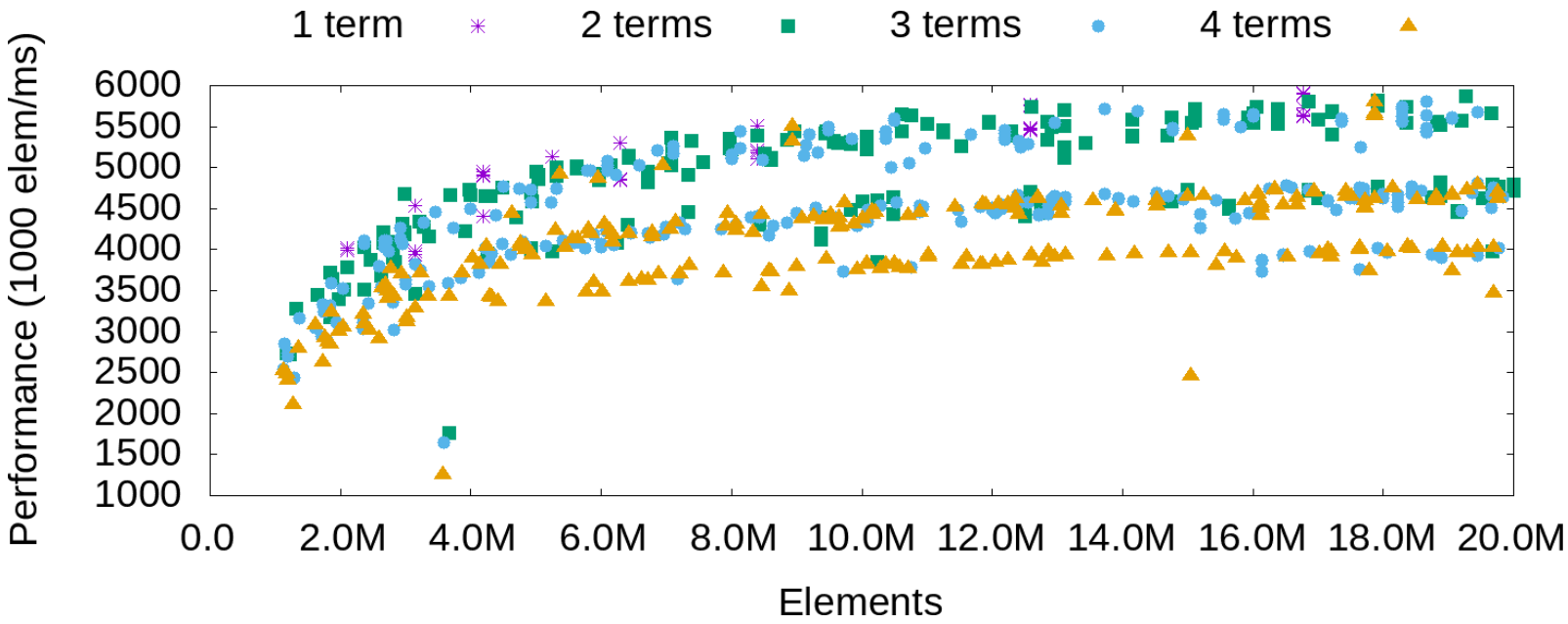
https://www.ebi.ac.uk/pdbe/entry/emdb/EMD-21024/experiment

https://www.ebi.ac.uk/pdbe/emdb/empiar/entry/10337/

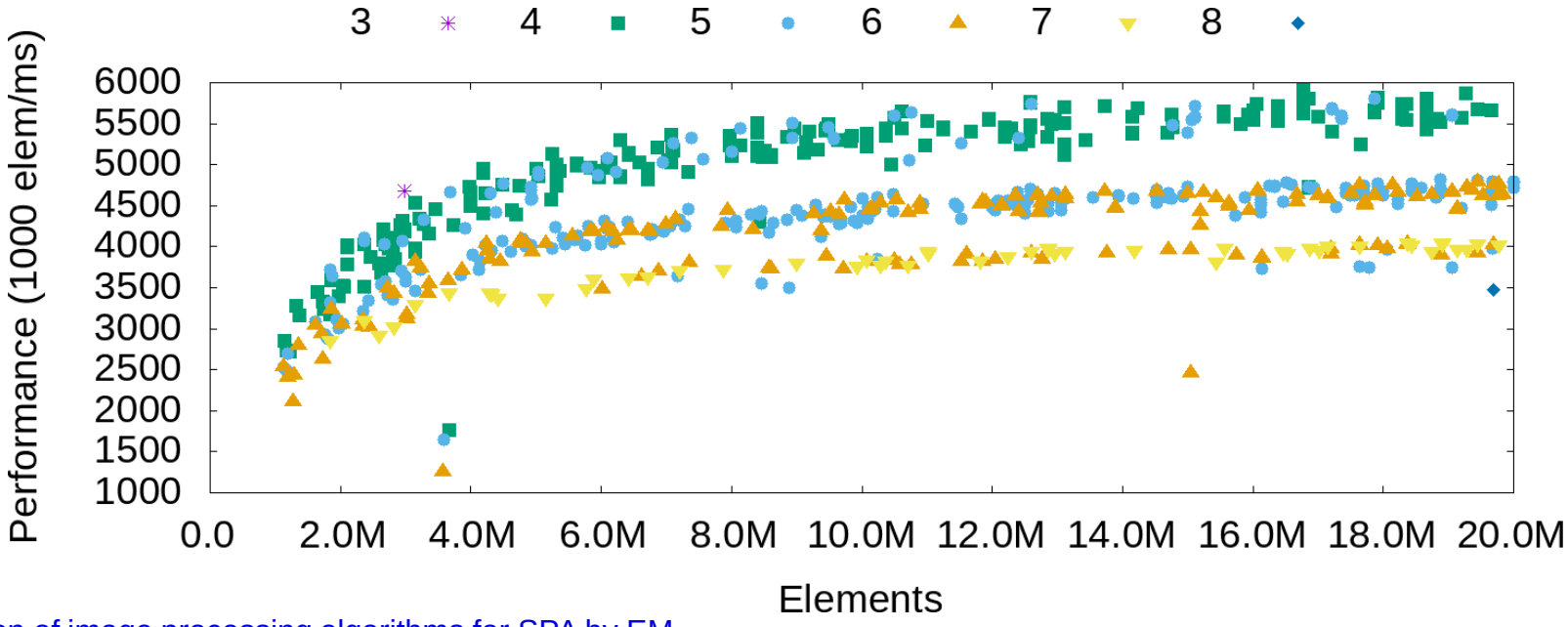
Icon made by Eucalyp from www.flaticon.com

Additional materials

$2^3 \times 5^2$
(two terms)



$2^3 \times 5^2$
(single kernel)



MUNI FI

cuFFTAdvisor - terms

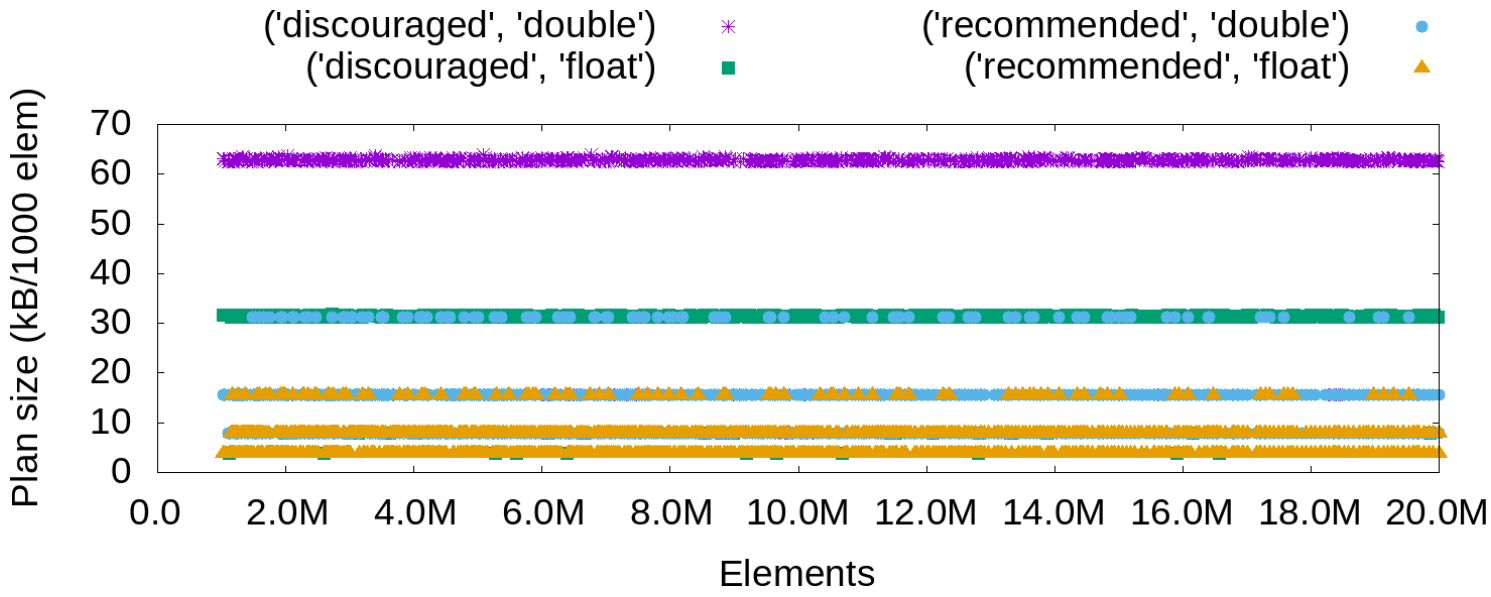
- $2^a \times 3^b \times 5^c \times 7^d$, where $a \neq 0$
- make sure the size of the input can be decomposed to as few kernel calls as possible (more info)

	1 term	2 terms	3 terms	4 terms	1-4 terms	heuristics	autotuned
count	2000	1927	1963	1972	2000	2000	2000
mean	5.05	5.94	5.84	5.61	5.90	6.03	6.94
std	1.70	1.80	1.71	1.68	1.67	1.68	1.81
min	0.54	0.93	0.82	0.80	1.00	1.00	1.01
25%	3.79	4.59	4.57	4.21	4.64	4.73	5.46
50%	4.73	5.49	5.43	5.36	5.51	5.69	6.68
75%	5.98	7.25	7.11	6.84	7.22	7.39	8.22
max	11.65	12.65	14.42	12.16	14.42	14.42	13.77

cuFFTAdvisor – memory usage

Our primary focus was performance

- using ‘recommended’ sizes automatically reduces memory footprint
- using multiple of two reduces memory
- more details in the paper [6]



MUNI FI