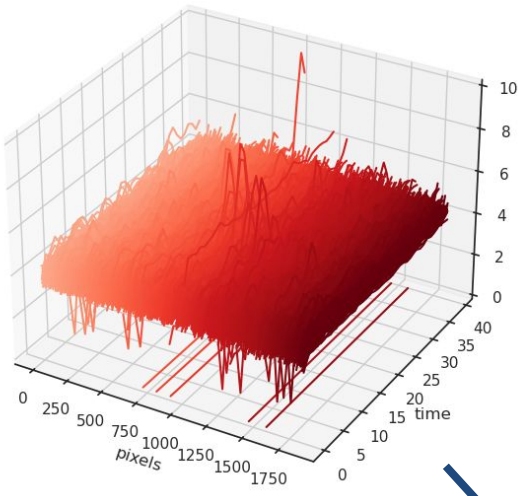
CTAO Real Time Analysis : investigating porting to GPU

LAPP: **Vincent Pollet**, Pierre Aubert, Sami Caroff, Thomas Vuillaume

11/07/2024

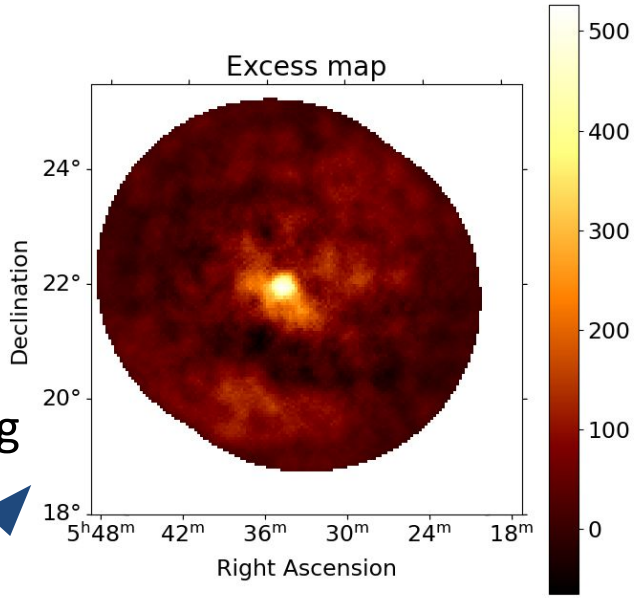
Funded by the European Unions's Horizon 2020 – Grant N˚824064

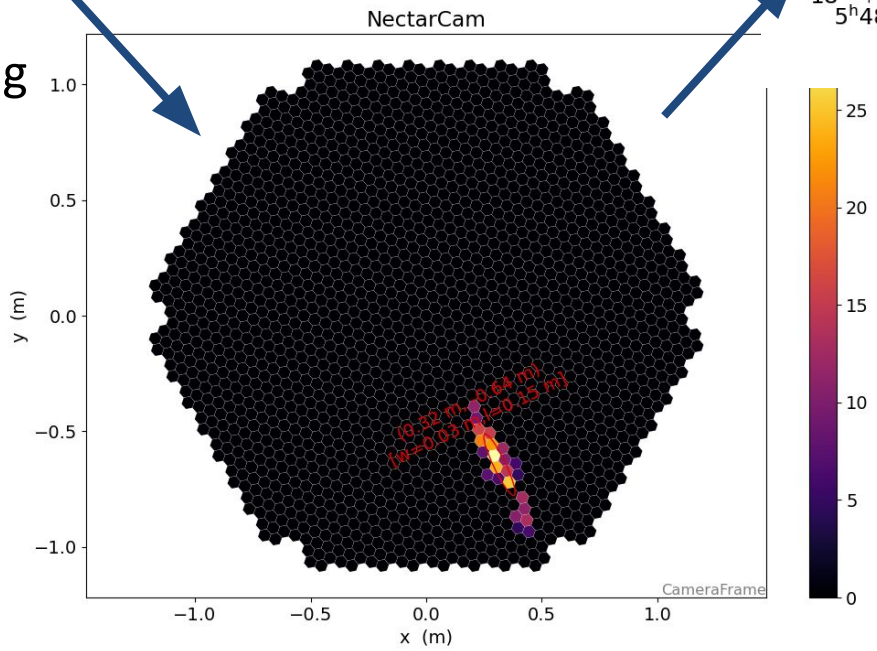
Cherenkov Telescope Array Observatory (CTAO)



Signal processing
~3GB/s

Machine learning
~20kB/s





R&D project: numpy implementation that translates to gpu

cuNumeric

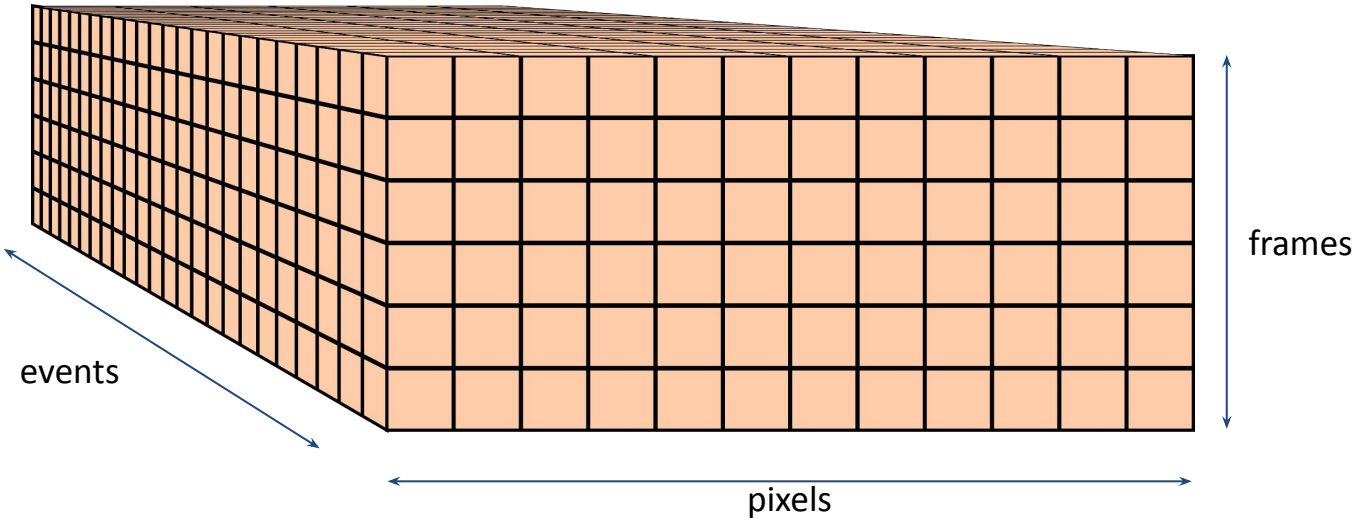
https://gitlab.in2p3.fr/CTA-LAPP/rta/pyhiperta/

- Data:
  - Events: ~10^7
  - samples: 40
  - pixels: 1855

→

- **batch** of few events on CPU
- **batch** of many events on GPU

- Spatial & temporal memory locality → data tensor
- Parallelization → independent events processed in parallel
- vectorization → independent pixel processing per time slice



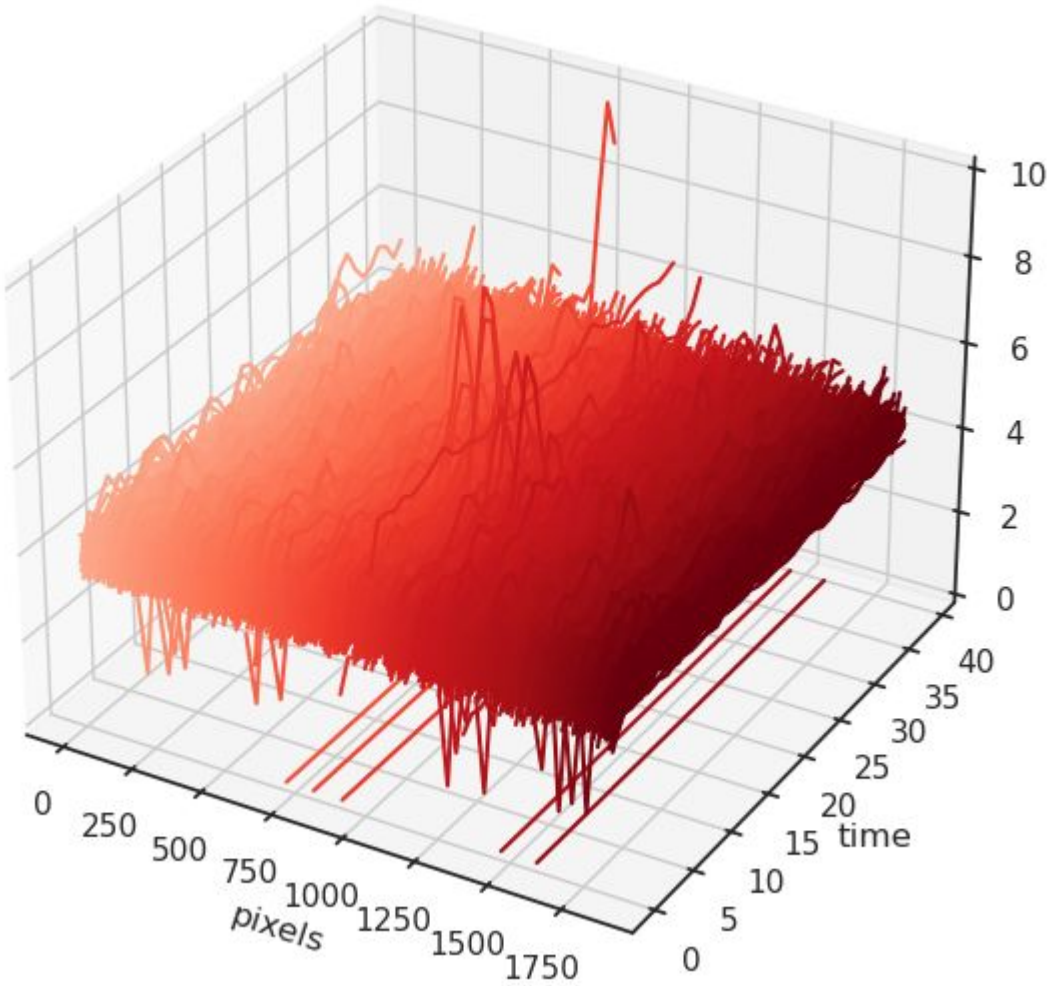
Processing steps:

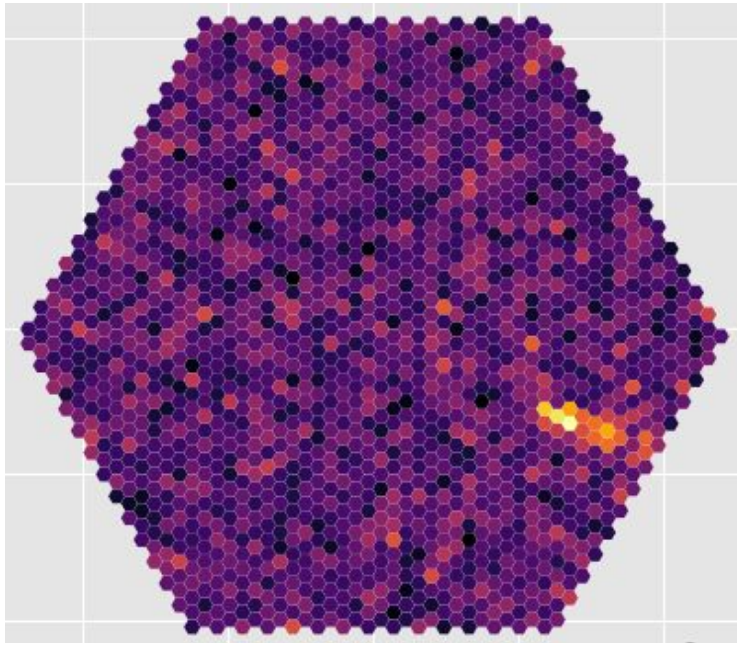
- calibration
- integration
- cleaning
- hillas parameters

$$w_{e,t,p} = (w_{e,t,p} - ped_{e,p}) \cdot gain_{e,p}$$

```
def calibrate(waveform, gains, pedestals):
    """
    waveform
      Shape: (N_batch, N_frames, N_pixels)
    gains
        Shape: (N_batch, N_pixels).
    pedestals
        Shape (N_batch, N_pixels)
    """

    return (waveform - pedestals[..., np.newaxis, :]) * gains[..., np.newaxis, :]
```

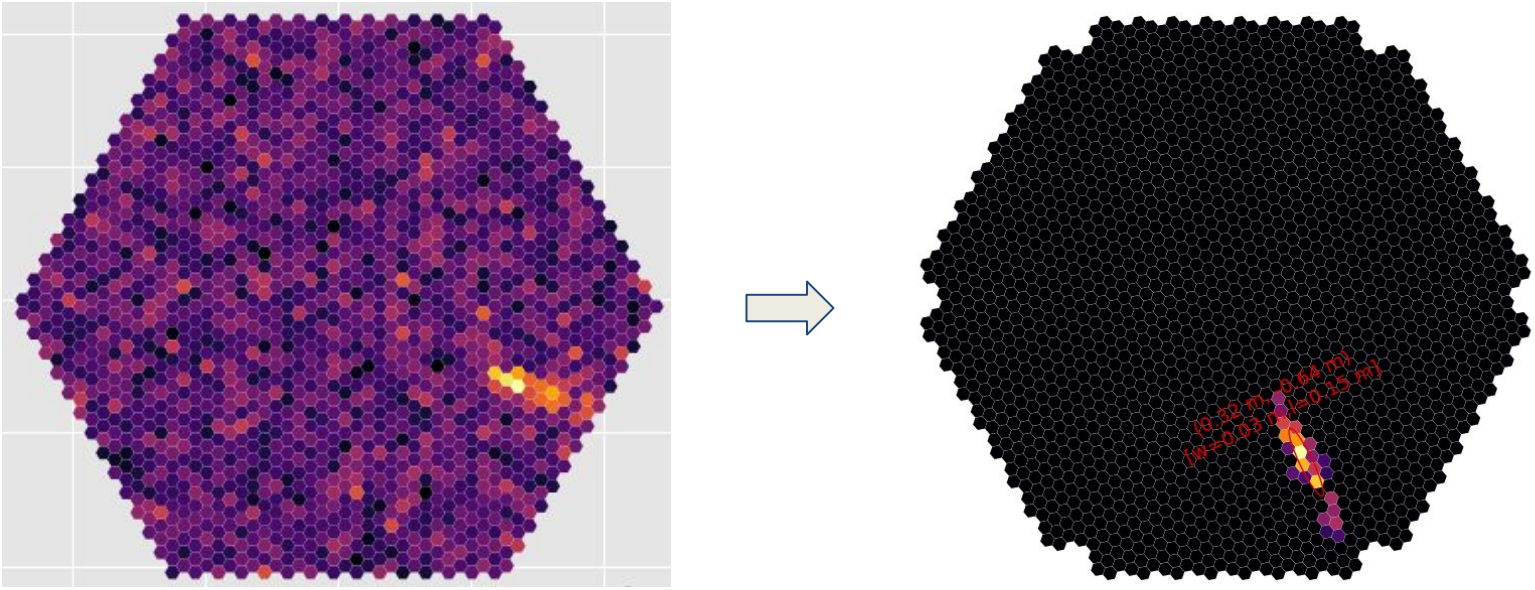

10
8
6
4
2
0
40
35
30
25
20
15
10
5
0
time
0
250
500
750
1000
1250
1500
1750
pixels



```
def integrate_local_pe                s, nb_frames_b                b_frames_after_max):
    charge = np.empty                     [0], wa
    for w in wavefor                                    uential operations
        for p in wavefor
            max_idx = np.ar
            charge[w, p] = wave
                w,
                max_idx-n                                   frames_after_max+1,
                p
            ].sum()                    on vec
    return charge
```

Integration: what to avoid

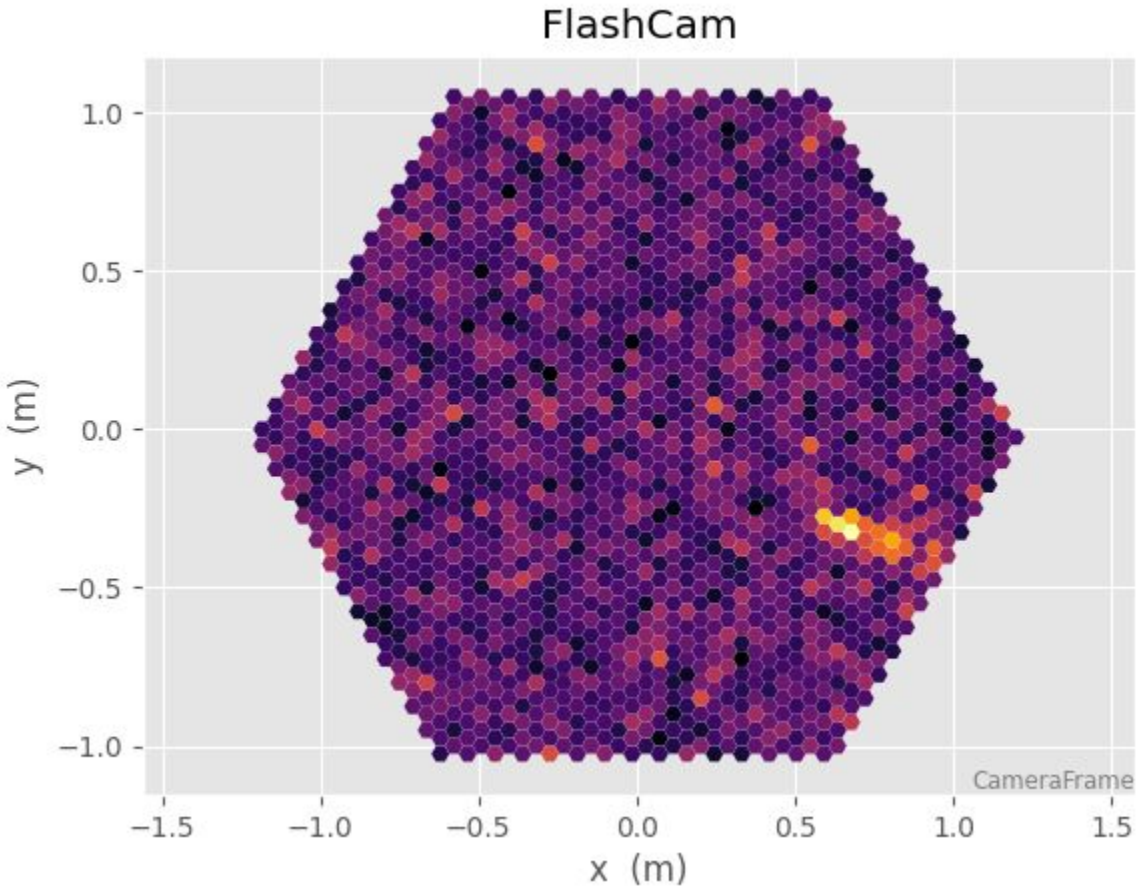
```
def integrate_local_peak(
   waveform: np.ndarray, nb_frames_before_max: int, nb_frames_after_max: int
):
    peaks_idx = np.argmax(waveform, axis=-2, keepdims=True)
    window_idx = peaks_idx + \
        np.arange(-nb_frames_before_max,
                  nb_frames_after_max + 1,
                  dtype=np.int32)[..., np.newaxis]
    waveform_windows = np.take_along_axis(waveform, window_idx, axis=-2)
    charge = waveform_windows.sum(axis=-2)
    return charge
```

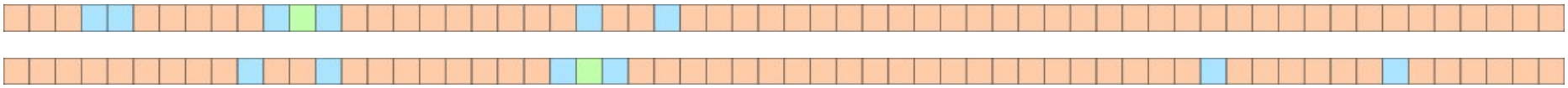


Keep pixels that verify 2 conditions:

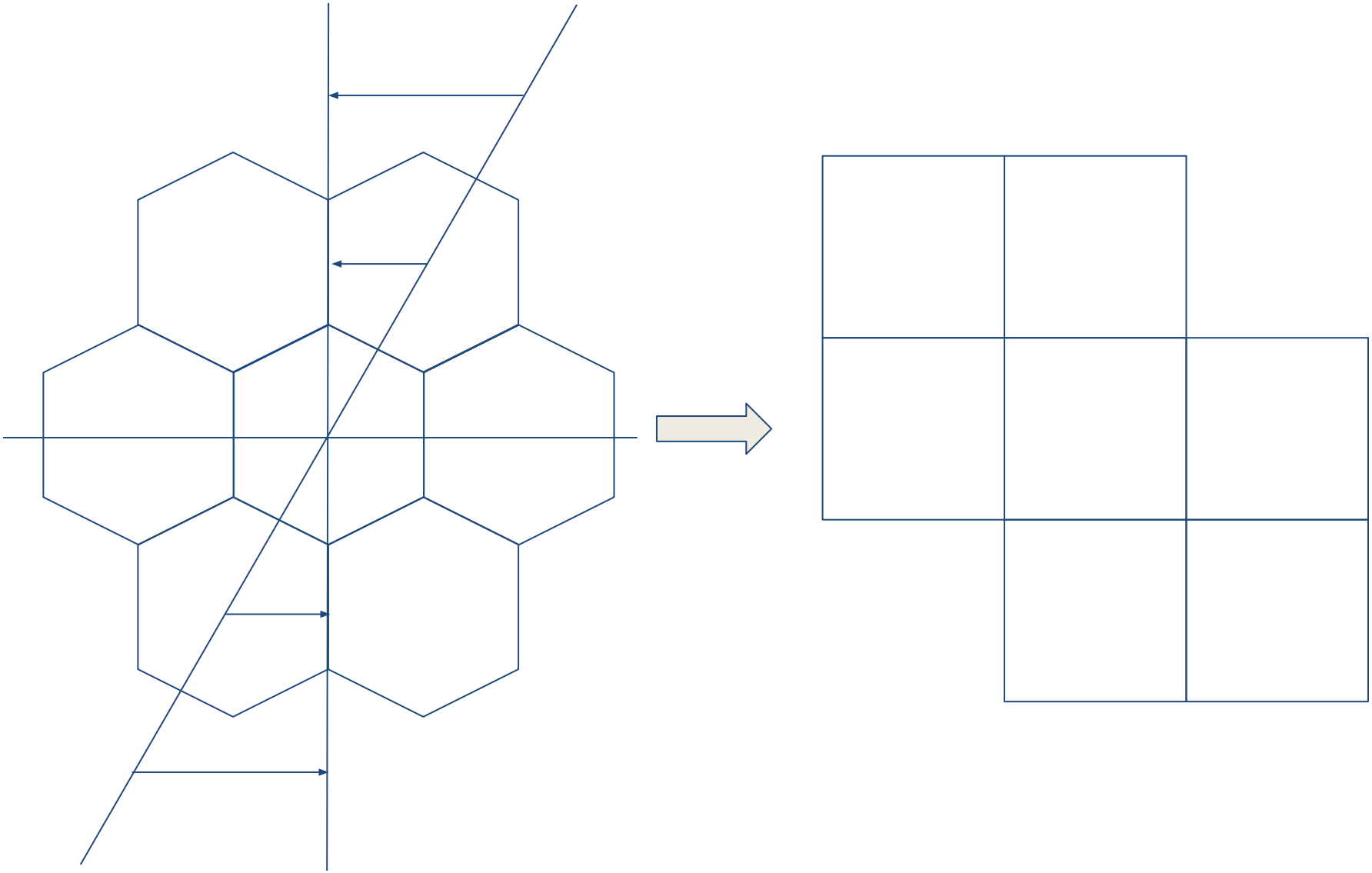
- condition 1: Pixels are above T1 and have at least N neighbors above T1
- condition 2: Pixels are above T2 and have at least 1 neighbor passing condition 1

Cleaning - Memory access





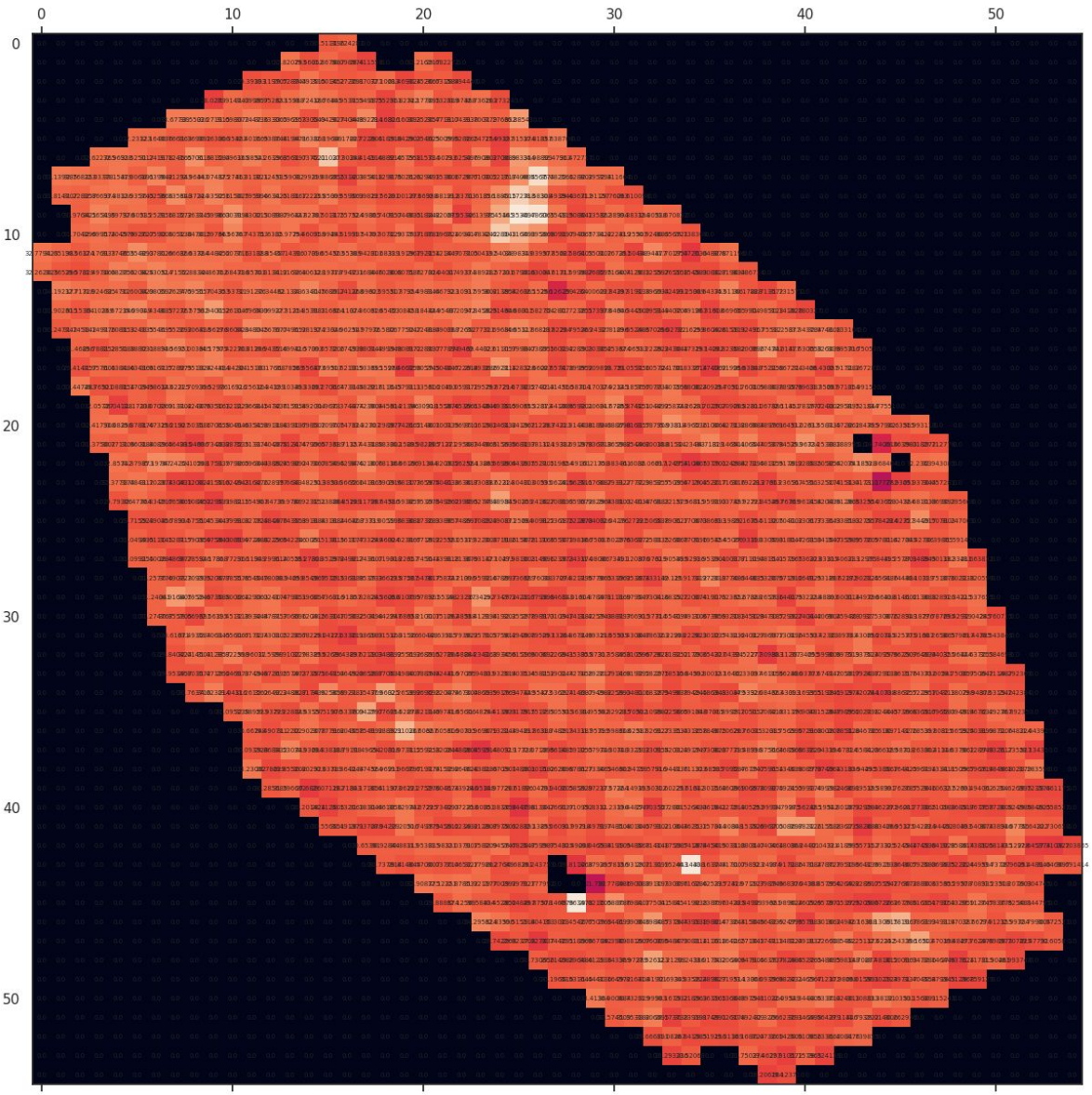
Cleaning - Hexagonal to square



Cleaning - Square stencil

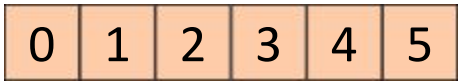
Stencil

1	1	0
1	0	1
0	1	1



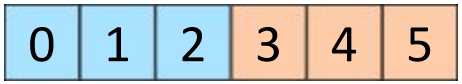
Cleaning - Sliding window view

```
>>> from numpy.lib.stride_tricks import sliding_window_view
>>> x = np.arange(6)
>>> x.shape
(6,)
>>> v = sliding_window_view(x, 3)
>>> v.shape
(4, 3)
>>> v
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4],
       [3, 4, 5]])
```



Cleaning - Sliding window view

```
>>> from numpy.lib.stride_tricks import sliding_window_view
>>> x = np.arange(6)
>>> x.shape
(6,)
>>> v = sliding_window_view(x, 3)
>>> v.shape
(4, 3)
>>> v
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4],
       [3, 4, 5]])
```



Cleaning - Sliding window view

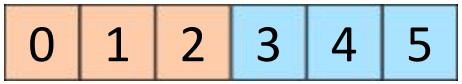
```
>>> from numpy.lib.stride_tricks import sliding_window_view
>>> x = np.arange(6)
>>> x.shape
(6,)
>>> v = sliding_window_view(x, 3)
>>> v.shape
(4, 3)
>>> v
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4],
       [3, 4, 5]])
```

Cleaning - Sliding window view

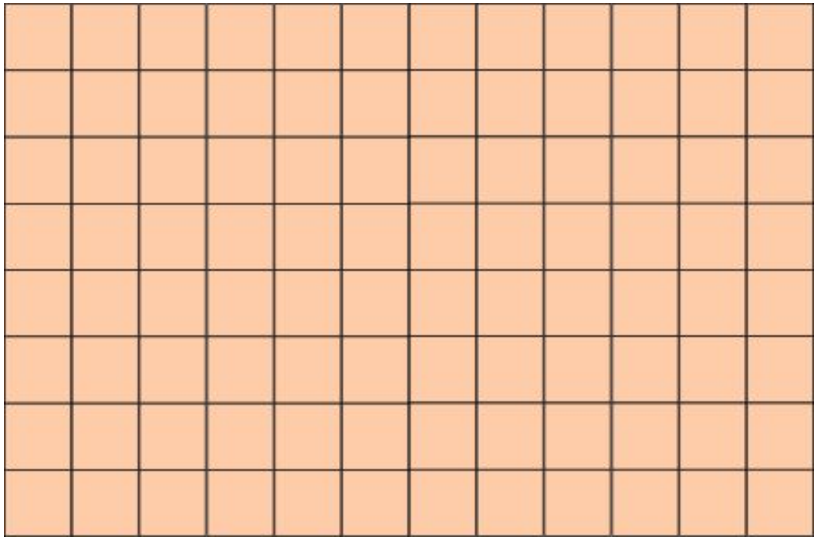
```
>>> from numpy.lib.stride_tricks import sliding_window_view
>>> x = np.arange(6)
>>> x.shape
(6,)
>>> v = sliding_window_view(x, 3)
>>> v.shape
(4, 3)
>>> v
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4],
       [3, 4, 5]])
```

Cleaning - Sliding window view

```
>>> from numpy.lib.stride_tricks import sliding_window_view
>>> x = np.arange(6)
>>> x.shape
(6,)
>>> v = sliding_window_view(x, 3)
>>> v.shape
(4, 3)
>>> v
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4],
       [3, 4, 5]])
```

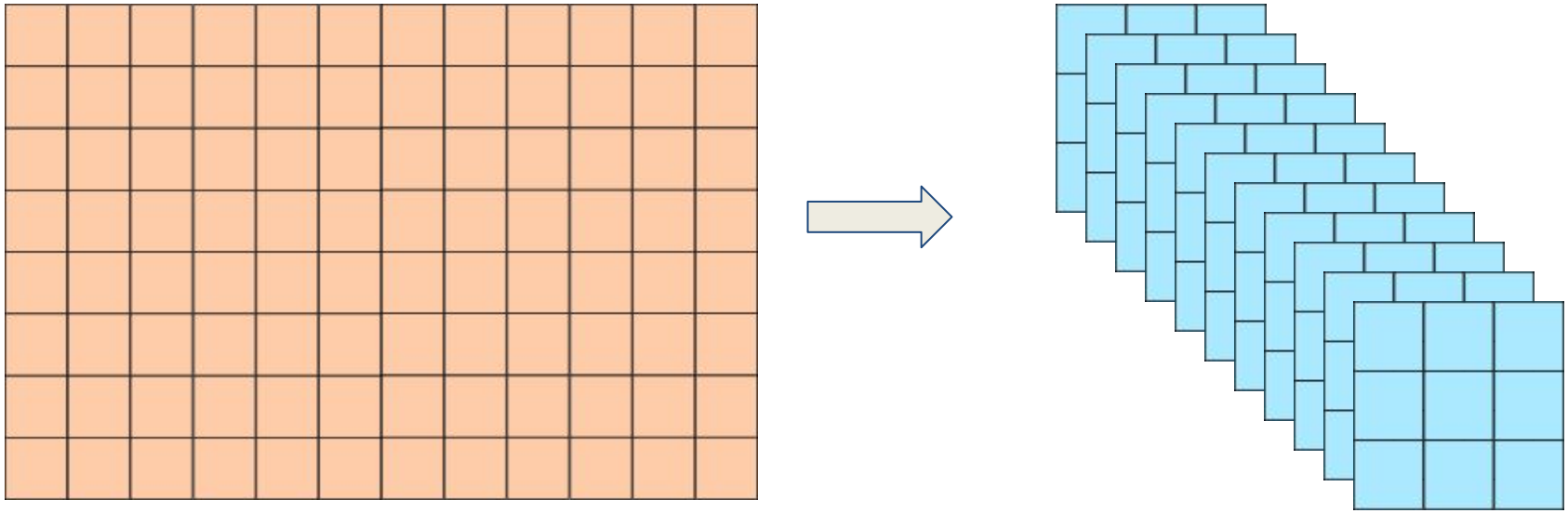


```
    neighbors_only_view = sliding_window_view(
        waveforms_2D_padded,
        neighbors_stencil.shape
    ) * neighbors_stencil
    # condition 1:
    mask = (waveforms_2D >= pixel_threshold) \
        & ((neighbors_only_view >= pixel_threshold).sum(axis=(-2, -1)) \
            >= min_number_neighbors)
```



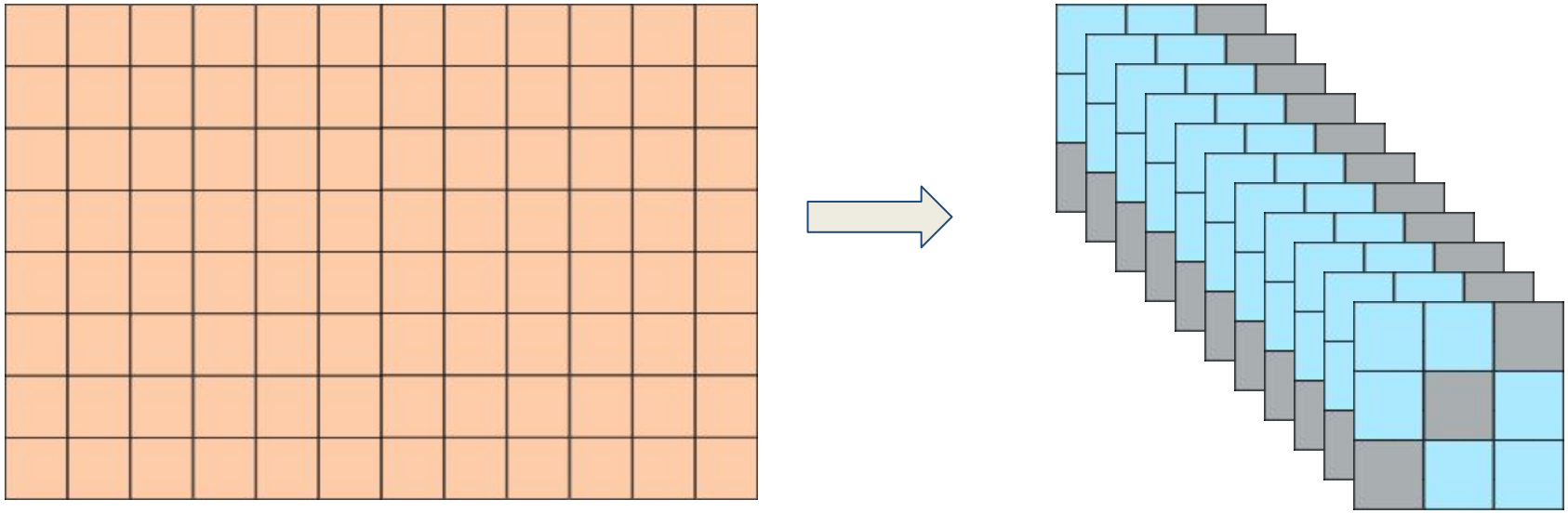
Cleaning - Sliding window view

```
    neighbors_only_view = sliding_window_view(
        waveforms_2D_padded,
        neighbors_stencil.shape
    ) * neighbors_stencil
    # condition 1:
    mask = (waveforms_2D >= pixel_threshold) \
        & ((neighbors_only_view >= pixel_threshold).sum(axis=(-2, -1)) \
            >= min_number_neighbors)
```



Cleaning - Sliding window view

```
    neighbors_only_view = sliding_window_view(
        waveforms_2D_padded,
        neighbors_stencil.shape
    ) * neighbors_stencil
    # condition 1:
    mask = (waveforms_2D >= pixel_threshold) \
        & ((neighbors_only_view >= pixel_threshold).sum(axis=(-2, -1)) \
            >= min_number_neighbors)
```

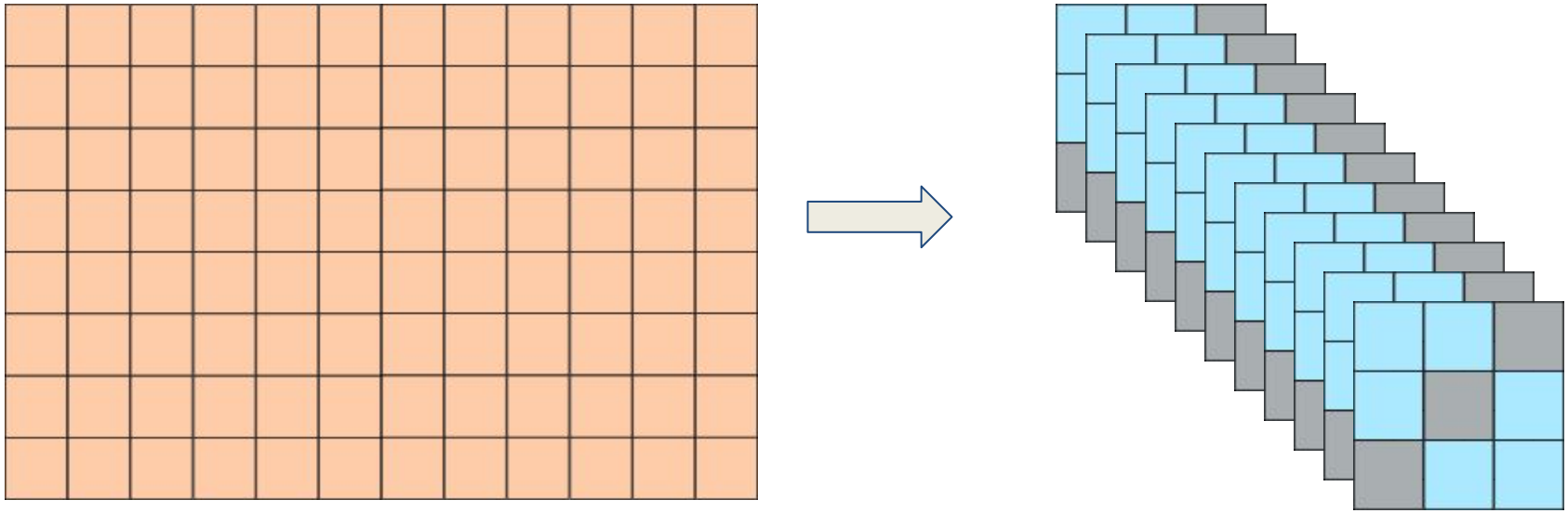


Cleaning - Sliding window view

```
    neighbors_only_view = sliding_window_view(
        waveforms_2D_padded,
        neighbors_stencil.shape
    ) * neighbors_stencil
    # condition 1:
    mask = (waveforms_2D >= pixel_threshold) \
        & ((neighbors_only_view >= pixel_threshold).sum(axis=(-2, -1)) \
            >= min_number_neighbors)
```

Pixel above threshold

N neighbors above threshold

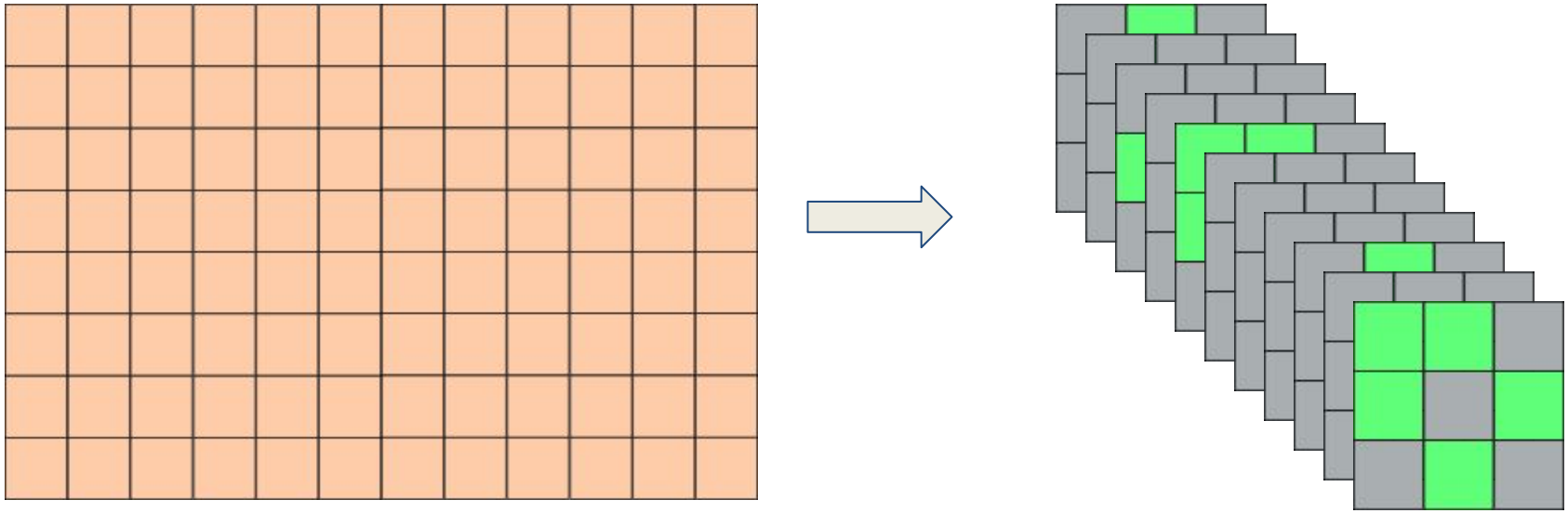


Cleaning - Sliding window view

```
    neighbors_only_view = sliding_window_view(
        waveforms_2D_padded,
        neighbors_stencil.shape
    ) * neighbors_stencil
    # condition 1:
    mask = (waveforms_2D >= pixel_threshold) \
        & ((neighbors_only_view >= pixel_threshold).sum(axis=(-2, -1)) \
            >= min_number_neighbors)
```

Pixel above threshold

N neighbors above threshold

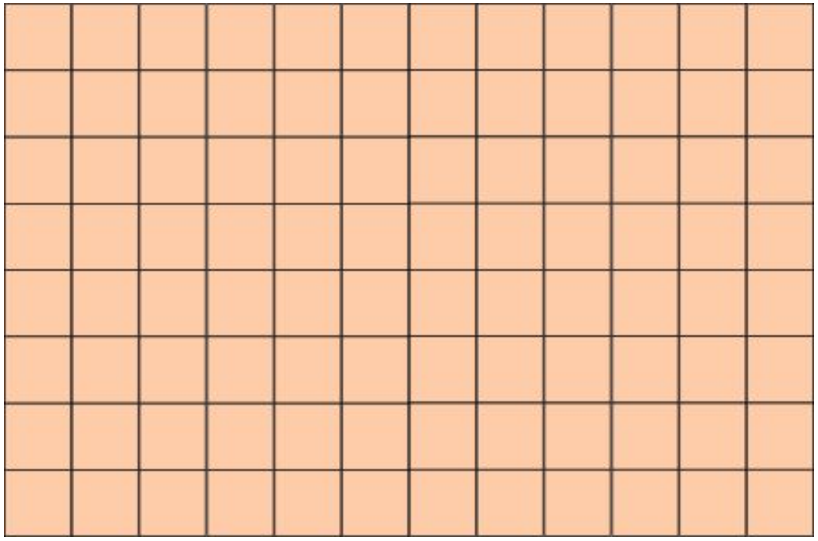


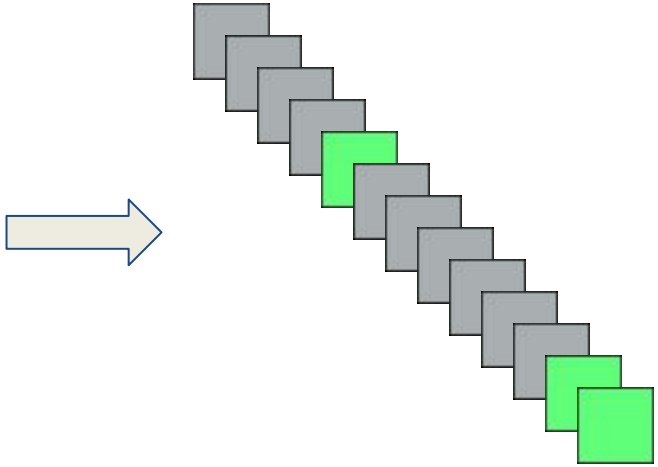
Cleaning - Sliding window view

```
    neighbors_only_view = sliding_window_view(
        waveforms_2D_padded,
        neighbors_stencil.shape
    ) * neighbors_stencil
    # condition 1:
    mask = (waveforms_2D >= pixel_threshold) \
        & ((neighbors_only_view >= pixel_threshold).sum(axis=(-2, -1)) \
            >= min_number_neighbors)
```

Pixel above threshold

N neighbors above threshold





Cleaning - Sliding window view

```
    neighbors_only_view = sliding_window_view(
        waveforms_2D_padded,
        neighbors_stencil.shape
    ) * neighbors_stencil
    # condition 1:
    mask = (waveforms_2D >= pixel_threshold) \
        & ((neighbors_only_view >= pixel_threshold).sum(axis=(-2, -1)) \
            >= min_number_neighbors)
```

Pixel above threshold

N neighbors above threshold

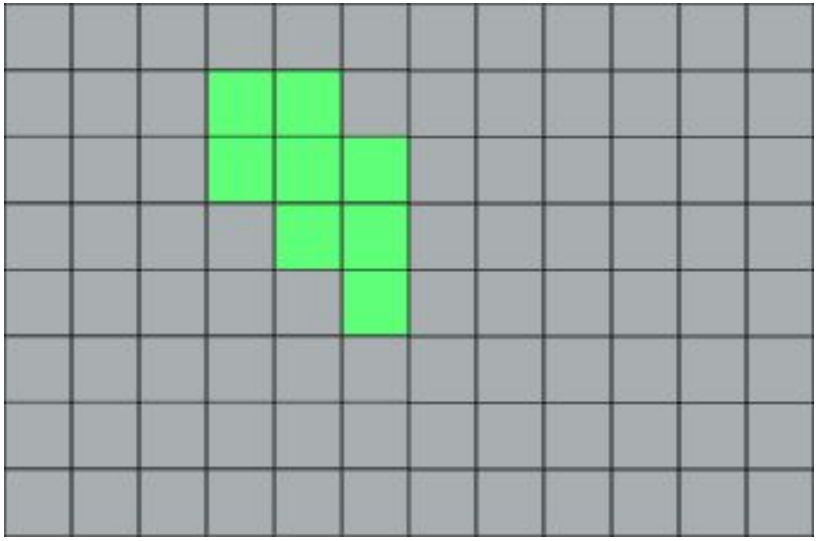


Image Parametrization - Hillas Parameters

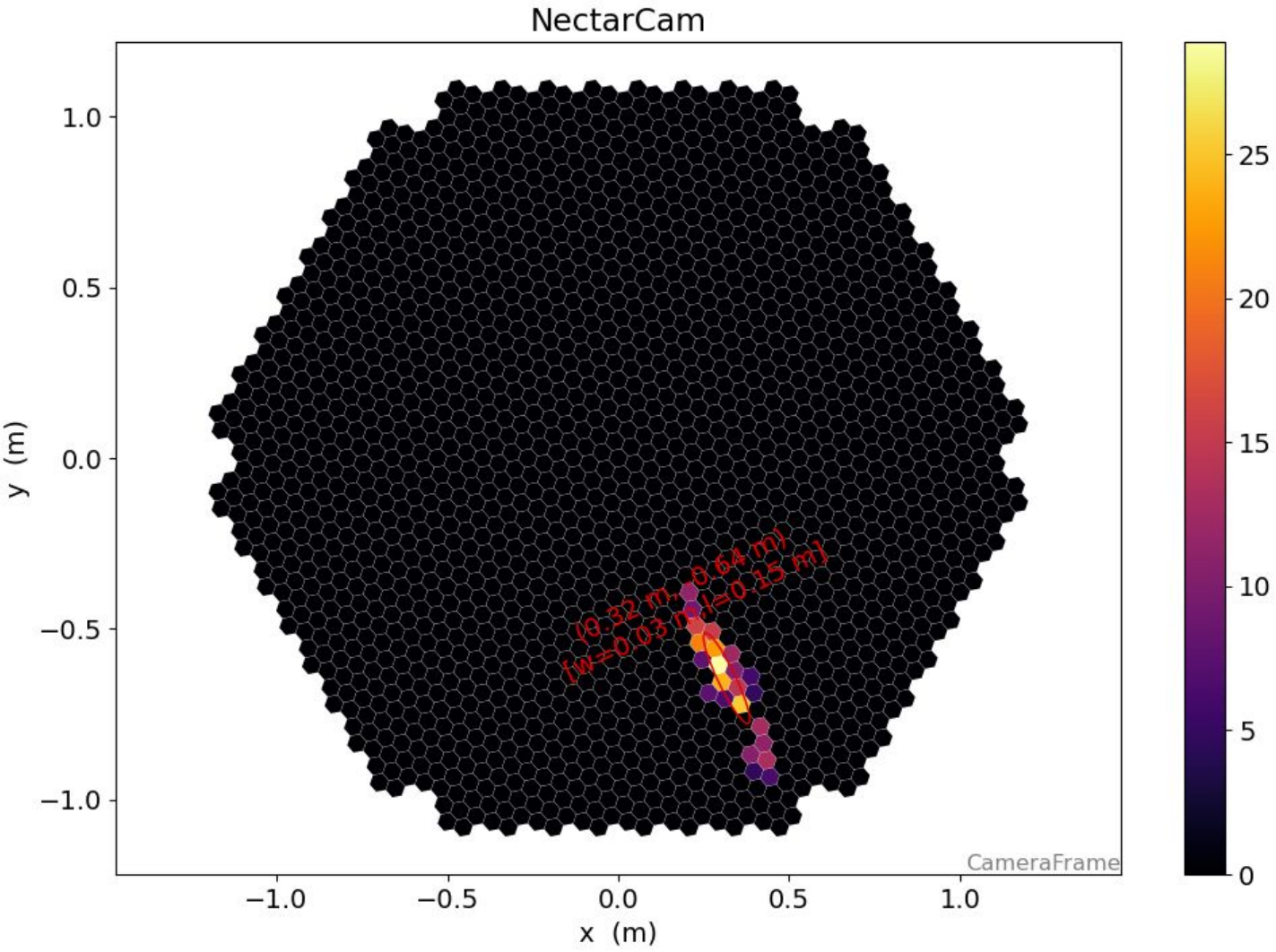
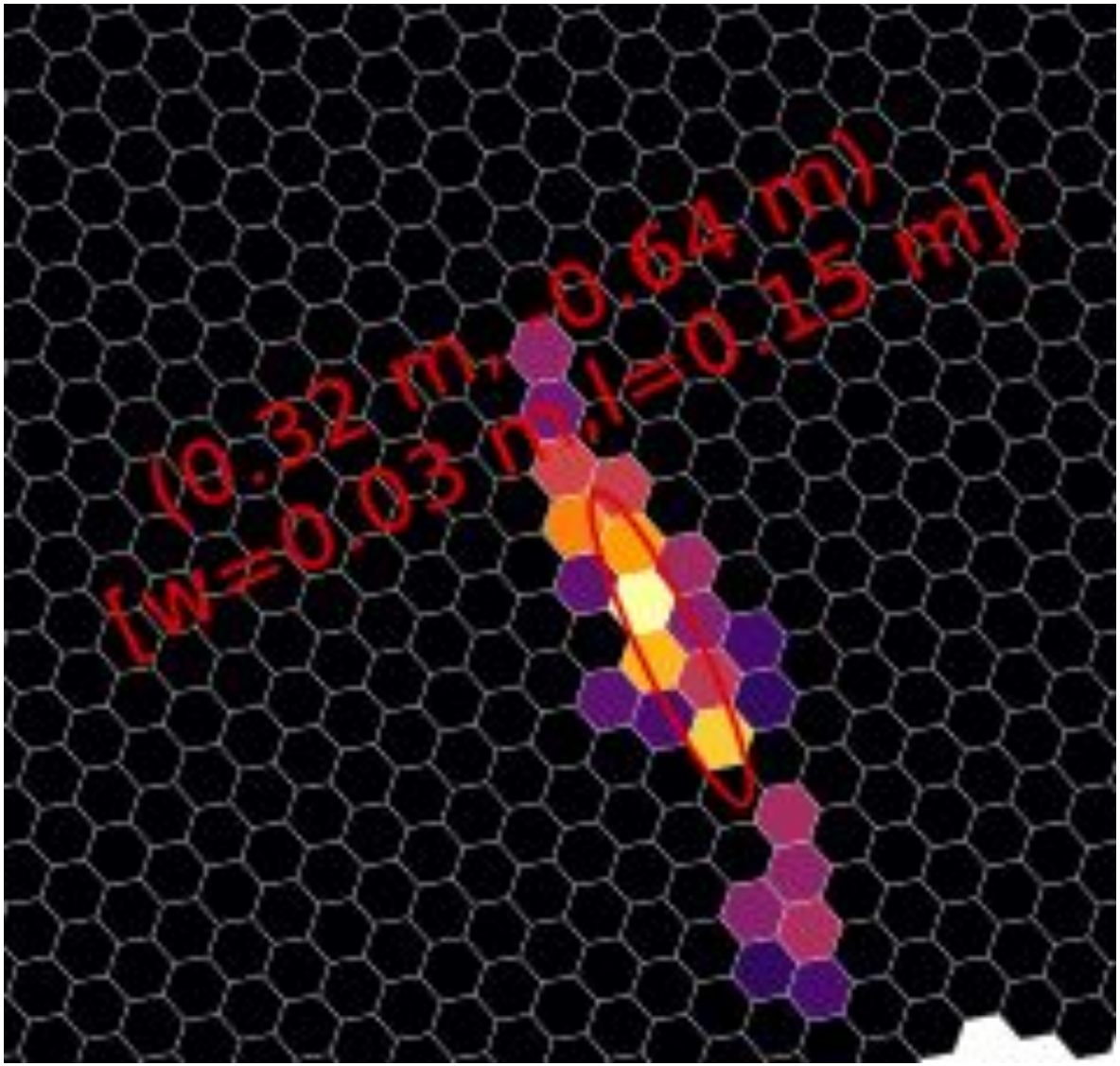


Image Parametrization - Hillas Parameters



Ellipse fit: Principal Component Analysis transformation

➢ Estimation of the covariance matrix of the image pixels

```
def hillas(pix_x, pix_y, image_batch):
    for image in image_batch:
        # center of gravity
        cog_x = np.average(pix_x, weights=image)
        cog_y = np.average(pix_y, weights=image)
        # PCA
        delta_x = pix_x - cog_x
        delta_y = pix_y - cog_y
        cov = np.cov(delta_x, delta_y, aweights=image, ddof=0)
        eig_vals, eig_vecs = np.linalg.eigh(cov)
        # width and length are eigen values of the PCA
        width, length = np.sqrt(eig_vals)
```

sequential operations

Issue: Numpy covariance, eigenvalue functions do not handle batches

Image Parametrization - Hillas Parameters

$$\operatorname{Var}(\vec{X}) = \begin{pmatrix} \operatorname{Var}(X_1) & \operatorname{Cov}(X_1, X_2) & \cdots & \operatorname{Cov}(X_1, X_p) \\ \operatorname{Cov}(X_2, X_1) & \ddots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{Cov}(X_p, X_1) & \cdots & \cdots & \operatorname{Var}(X_p) \end{pmatrix}$$

$$\operatorname{E}[X] = \sum_{i=1}^{\infty} x_i \, p_i,$$

$$\operatorname{Var}(X) = \operatorname{E}\left[X^2\right] - \operatorname{E}[X]^2$$

$$\operatorname{cov}(X, Y) = \operatorname{E}\left[(X - \operatorname{E}[X])(Y - \operatorname{E}[Y])\right]$$

```python
def hillas(images, position):
    intensity = waveform.sum(axis=-1)

    moments = (
    waveform[..., np.newaxis, :]  # shape (N_batch, 1, N_pixels) to broadcast for all moments
        * np.stack([position[0], position[1], position[0] ** 2, position[1] ** 2, position.prod(axis=0)])
    ).sum(axis=-1) / intensity[..., np.newaxis]

    E_x_y_square = moments[..., 0:2] ** 2
    variance = moments[..., 2:4] - E_x_y_square
    covariance = moments[..., 4] - moments[..., 0:2].prod(axis=-1)
    varx_plus_vary = variance.sum(axis=-1)
    char_pol_delta = np.sqrt(varx_plus_vary**2 + 4.0 * (covariance**2 - variance.prod(axis=-1)))
    eigenValueHigh = (varx_plus_vary + char_pol_delta) / 2.0
    eigenValueLow = (varx_plus_vary - char_pol_delta) / 2.0

    length = np.sqrt(eigenValueHigh)
    width = np.sqrt(eigenValueLow)
```

Translation from numpy to cuNumeric

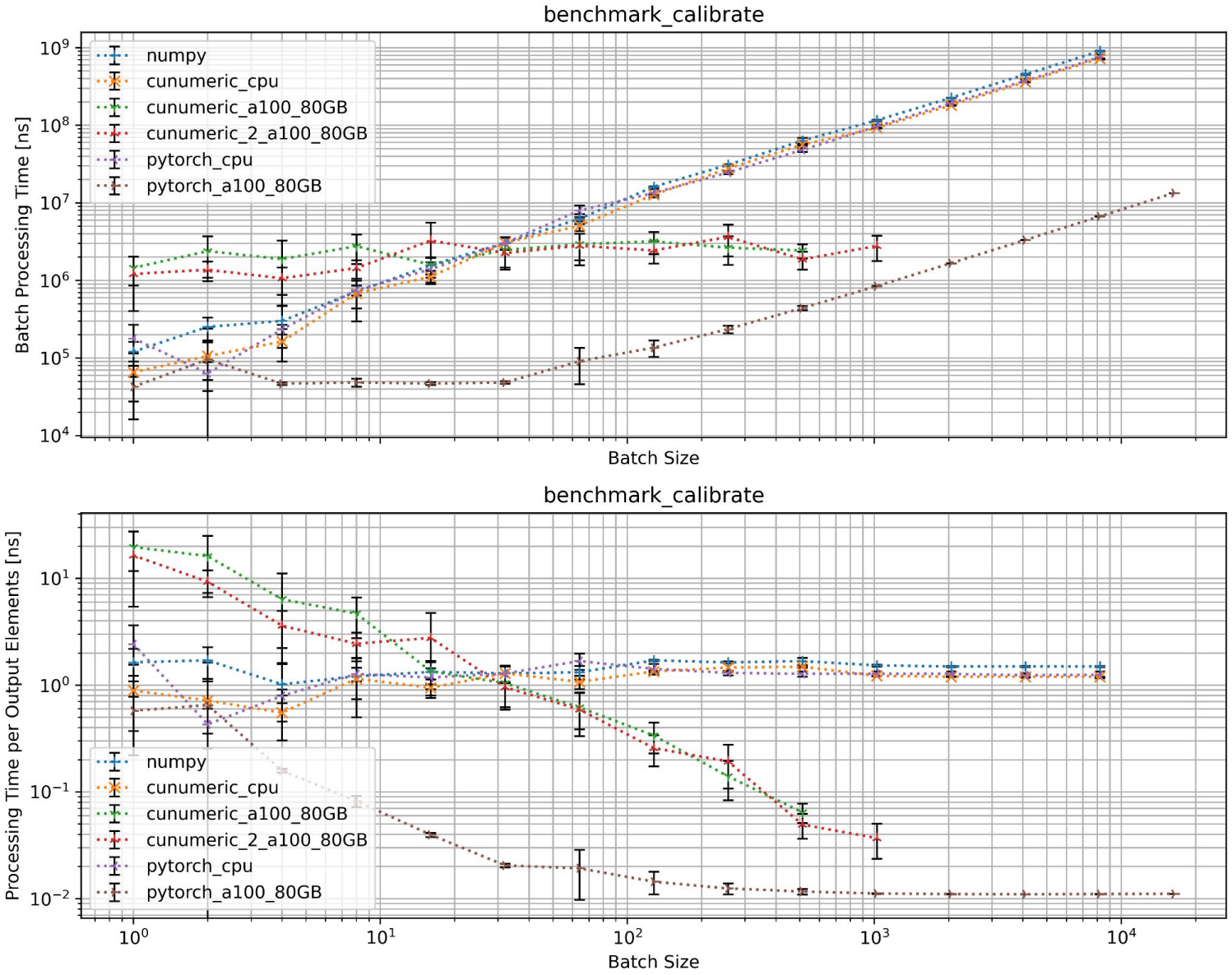
- Installation
  - version 24.03 from source (first legate.core, then cuNumeric)
  - dependencies installed with cuda (cuTensor cuTensor-cuda missing)
  - requires machine with gpu to enable gpu support
- Translation
  - `import cunumeric as np`
  - Differences in API of `cunumeric.random.default_rng`
  - no “stride_tricks” -> no cleaning
- Execution
  - using legate driver with several cpus, gpus on same node
  - much more memory used
  - crash when not enough memory is available (even on cpu)

“Whenever possible, use the `out` parameter in the APIs, to avoid allocating an intermediate array in our implementation.” cuNumeric guidelines

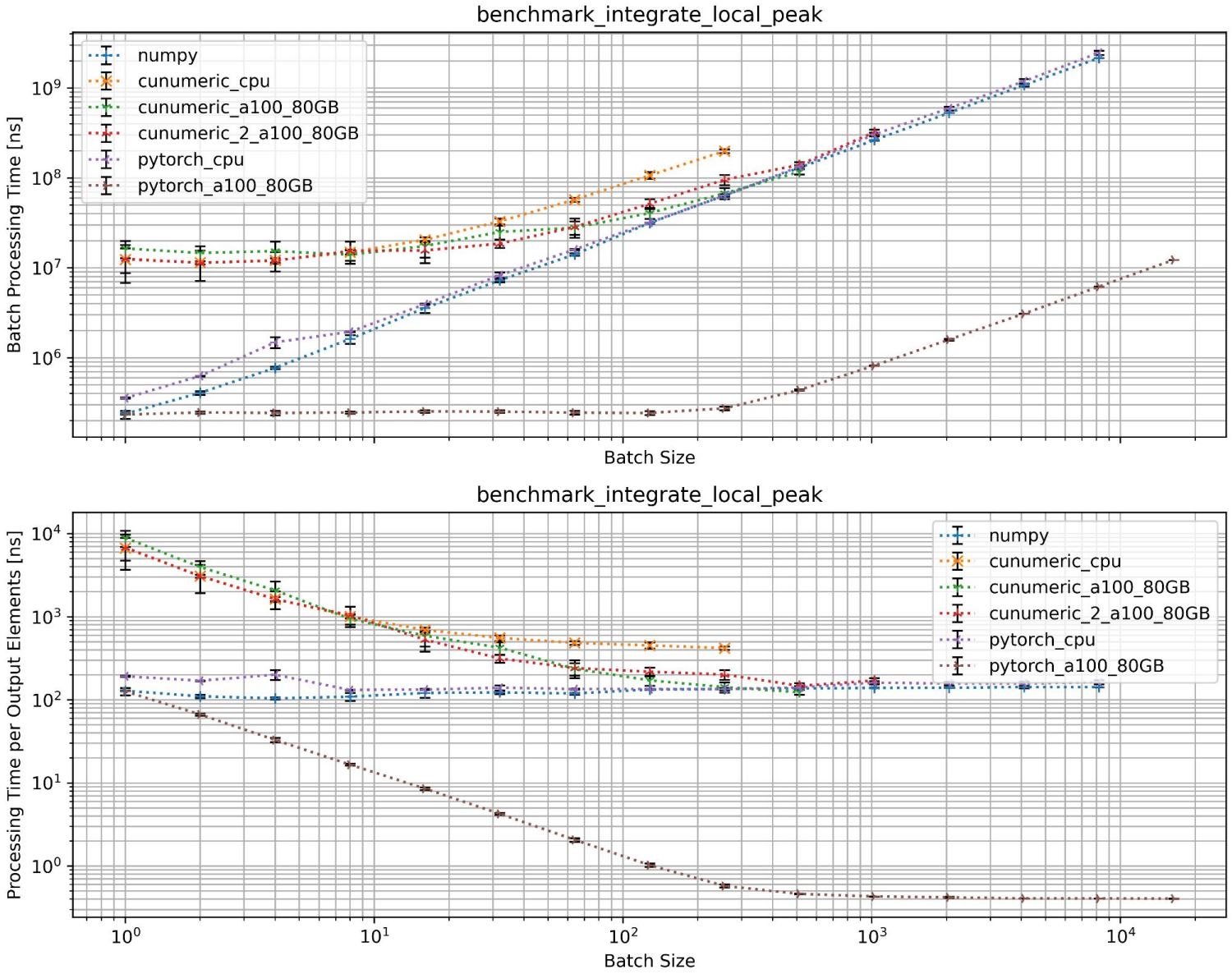
Translation from numpy to pytorch

- Installation
  - from pytorch conda channel, with cuda 12.1
- Translation
  - tedious but easy as API is very similar with some name changes
    - `np.* -> torch.*`
    - `np.ndarray -> torch.Tensor`
    - `np.array(...) -> torch.tensor(...)`
    - slightly different padding
    - no `striding_window_view` but `as_strided` is supported
    - introduce `torch.device`
    - fix `dtype` to `float32` or `int32`
- Execution
  - multi-cpu (mono node)
  - mono gpu

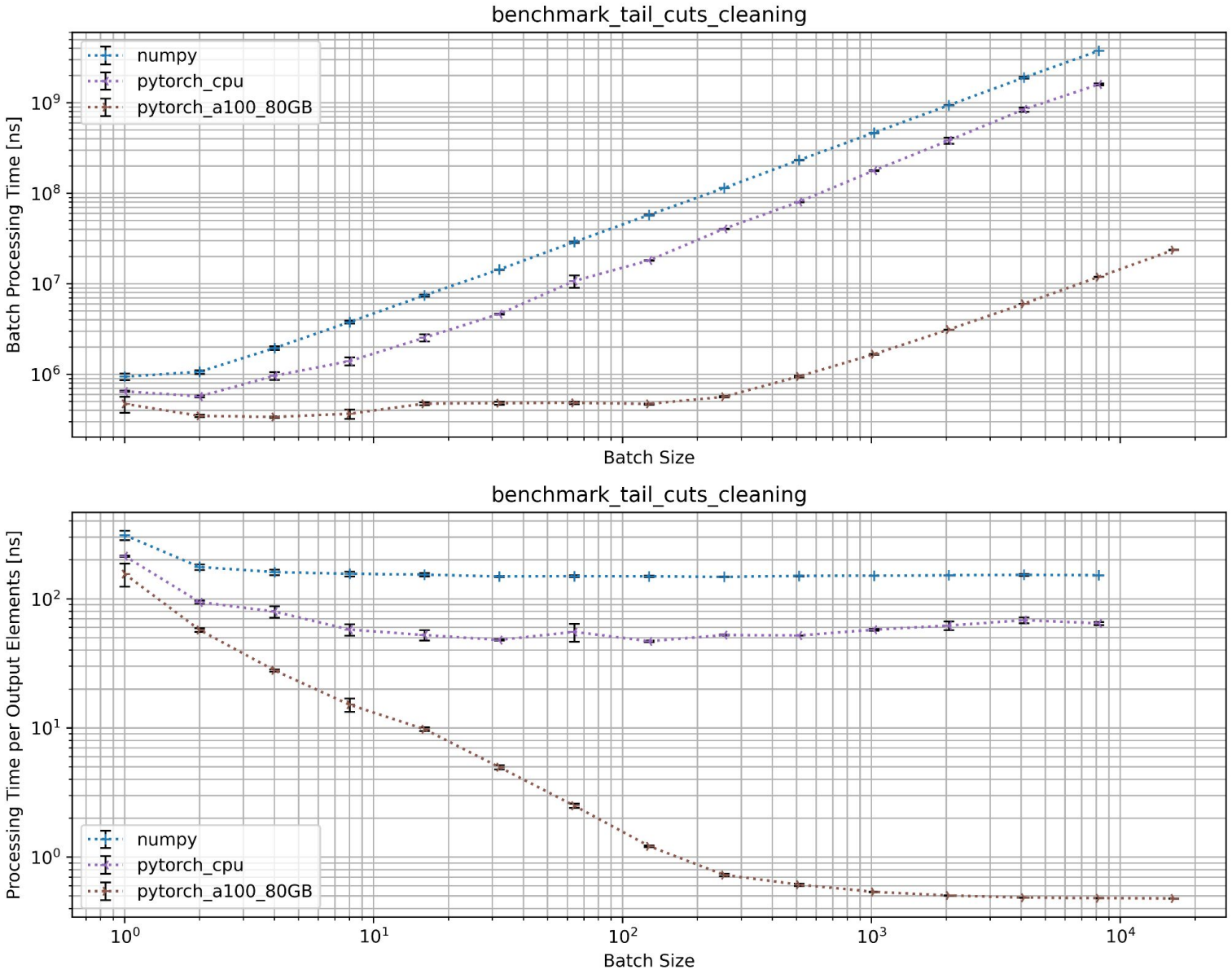
Benchmarks


benchmark_calibrate
Batch Processing Time [ns]
Batch Size
numpy
cunumeric_cpu
cunumeric_a100_80GB
cunumeric_2_a100_80GB
pytorch_cpu
pytorch_a100_80GB
benchmark_calibrate
Processing Time per Output Elements [ns]
Batch Size
numpy
cunumeric_cpu
cunumeric_a100_80GB
cunumeric_2_a100_80GB
pytorch_cpu
pytorch_a100_80GB

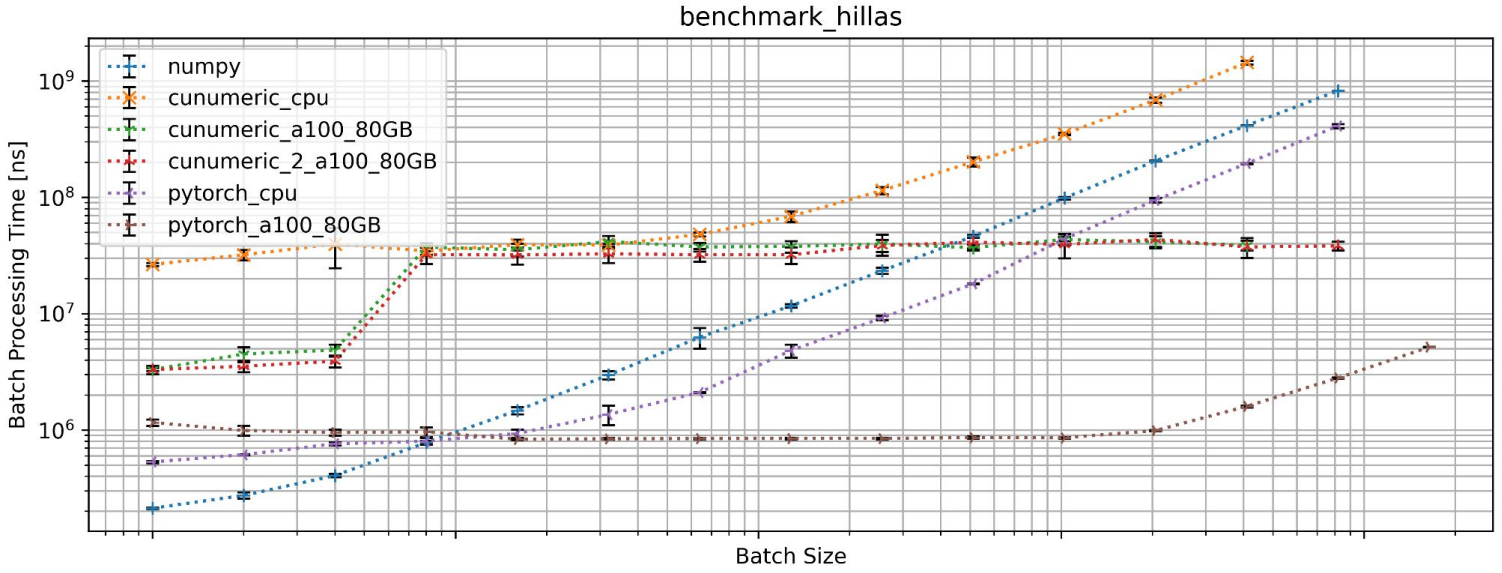
Benchmarks

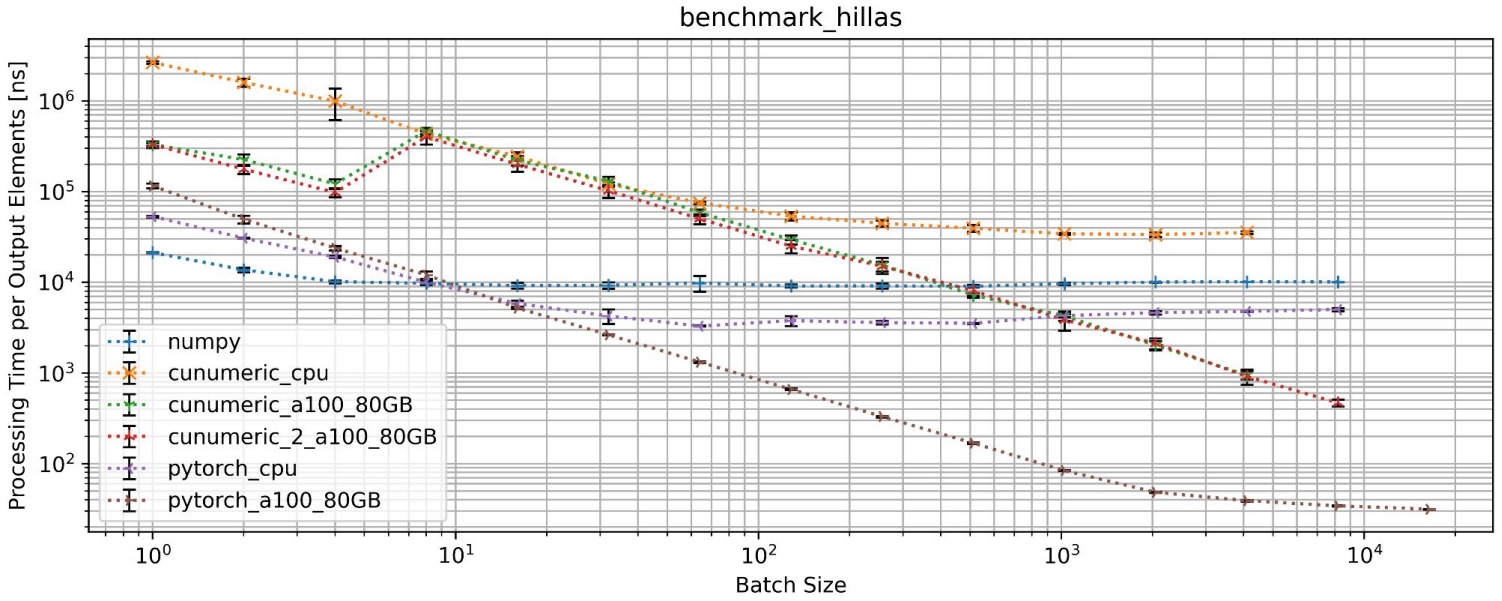

benchmark_integrate_local_peak
numpy
cunumeric_cpu
cunumeric_a100_80GB
cunumeric_2_a100_80GB
pytorch_cpu
pytorch_a100_80GB
Batch Processing Time [ns]
Batch Size
benchmark_integrate_local_peak
Processing Time per Output Elements [ns]
Batch Size

Benchmarks


benchmark_tail_cuts_cleaning
numpy
pytorch_cpu
pytorch_a100_80GB
Batch Processing Time [ns]
10^9
10^8
10^7
10^6
Batch Size
benchmark_tail_cuts_cleaning
Processing Time per Output Elements [ns]
10^2
10^1
10^0
numpy
pytorch_cpu
pytorch_a100_80GB
10^0
10^1
10^2
10^3
10^4
Batch Size

Benchmarks


benchmark_hillas
numpy
cunumeric_cpu
cunumeric_a100_80GB
cunumeric_2_a100_80GB
pytorch_cpu
pytorch_a100_80GB
Batch Processing Time [ns]
Batch Size


benchmark_hillas
numpy
cunumeric_cpu
cunumeric_a100_80GB
cunumeric_2_a100_80GB
pytorch_cpu
pytorch_a100_80GB
Processing Time per Output Elements [ns]
Batch Size

Conclusion

- Possible to have a single code base with *decent* performance on cpu and gpu implemented with python
- cuNumeric
  - still in development
  - doesn’t deliver all its promises (yet?)
  - transparent scaling to multi-gpu (multi-nodes?)
- pytorch
  - mature library
  - API very similar to numpy
  - good performances on cpu and gpu

Next steps:

- pre-allocated memory layout
- benchmark cpu to gpu data transfer
  - cuNumeric GPU direct storage
  - pytorch: mitigate transfer cost behind more computation ?

https://gitlab.in2p3.fr/CTA-LAPP/rta/pyhiperta/