GNA:

Global Neutrino Analysis framework and GPU based computations

A. Fatkina M. Gonchar D. Naumov K. Treskov
L. Kolupaeva A. Kalitkina

JINR DLNP

NEC 2019

Introduction

GNA (Global Neutrino Analysis) — flexible, extensible framework for the statistical data analysis; cuGNA is a GPU support library for GNA framework.

GNA goals

- Comprehensive models with a large number of parameters.
- Data analysis for JUNO and Daya Bay experiments.
- Global analysis of neutrino data (experiments: Daya Bay, JUNO, NOvA, T2K, etc).

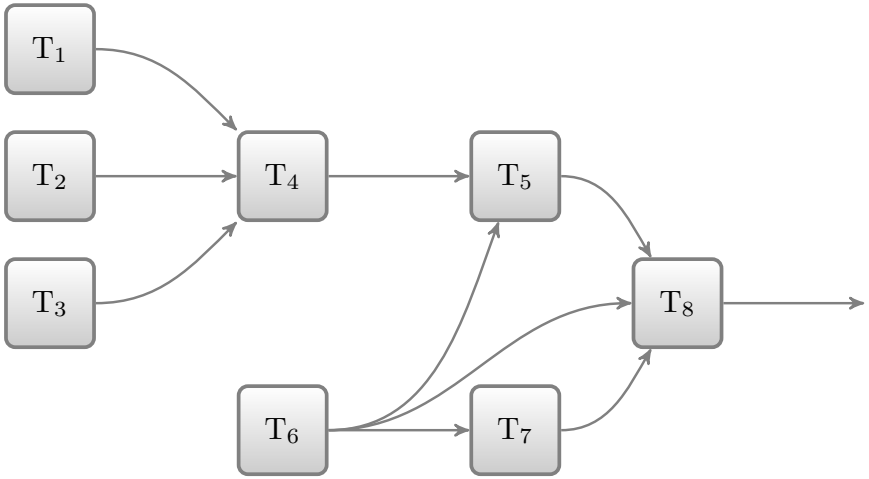
cuGNA goals

- Accelerate computations.
- Use a full power of target machine.
- Do it smoothly and hide data management details from the end user.
- Keep GNA performance features for CPU computations.

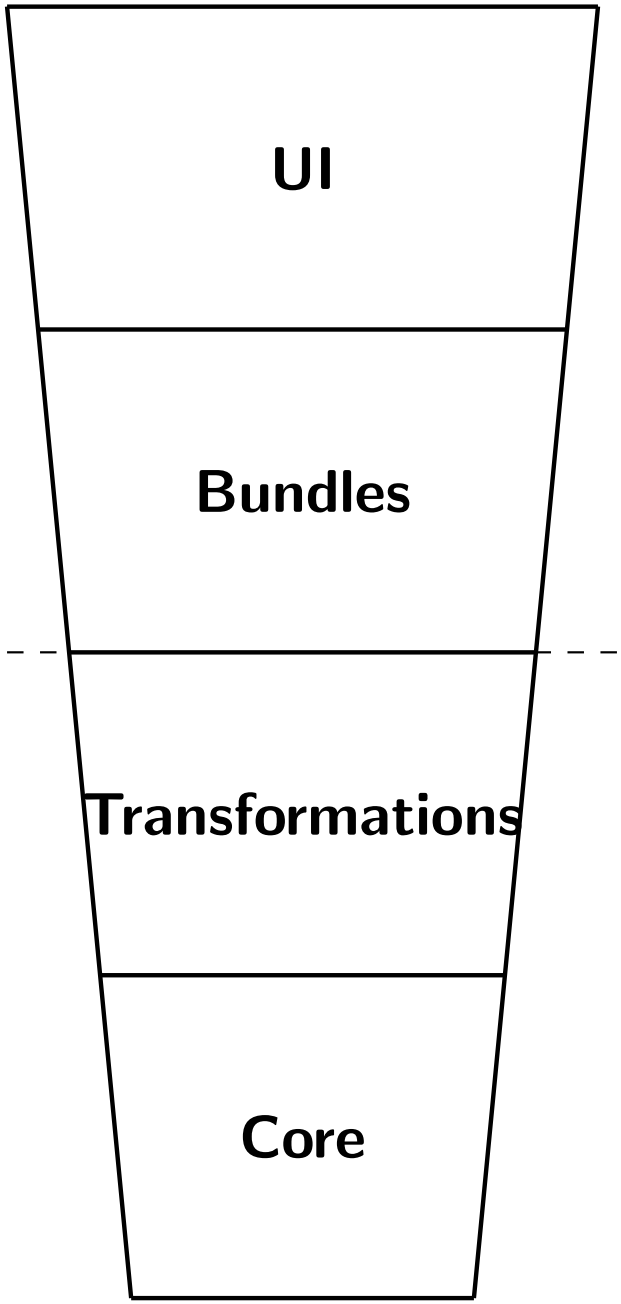
GNA overview

The idea of GNA

- Dataflow paradigm.
- Computations are represented by the graph, where nodes are transformations.
- Physical and programming issues are separated.
- Computations on demand in lazy manner.



GNA Structure



- Comprehensive command line chain.
- Computational graphs.
- Statistical analysis.

- Read configuration.
- Variables.
- Small computational graph.

Python (flexibility)

C++ (efficiency)

- Linear algebra
- Integration

- Statistics
- Physics

- Data
- Variable
- Transformation

- GPU transformation
- Host-Device data management

Computations

Number of parameters

- Daya Bay — 15 free parameters and 400 at all,
- JUNO antineutrino spectra — 5 free parameters and 100-1200 parameters in general (depands on task) .

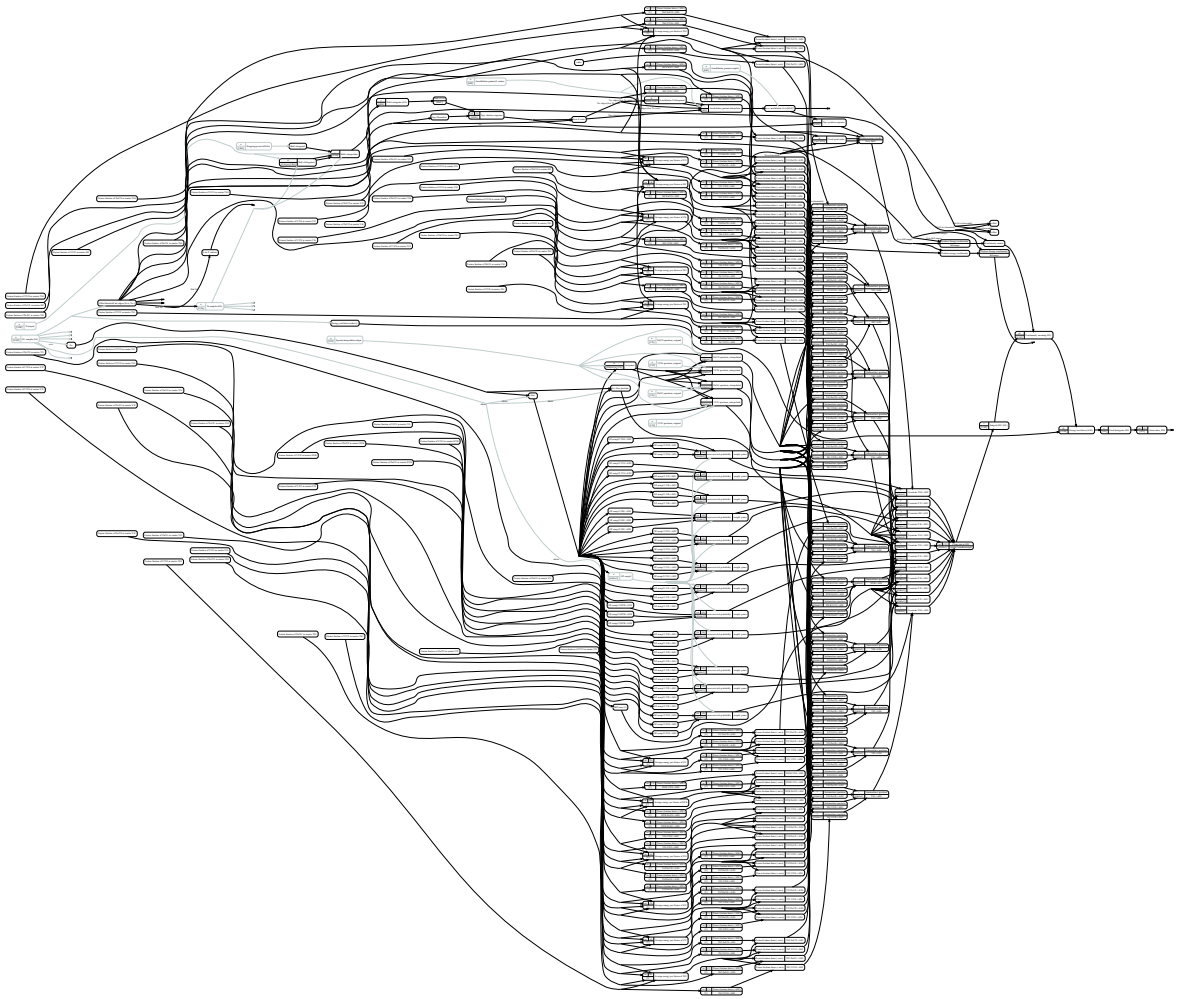
Expected execution time

- **Seconds** for a single model evaluation,
- **Minutes or hours** for multidimensional fit,
- **Days or months** for MC based methods.

```
Variables in namespace 'acc_norm':
  AD11                =          1 |          1±        0.01 [          1%] |
  AD21                =          1 |          1±        0.01 [          1%] |
Variables in namespace 'bkg_rate_acc':
  AD11                =     8.4636 |               [fixed]                  |
  AD21                =    6.29076 |               [fixed]                  |
Variables in namespace 'efflivetime':
  AD11                =7.73792e+07 |                                        |
  AD21                = 8.0053e+07 |                                        |
Variables in namespace 'acc_num_bf':
  AD11                =6.54907e+08 |                                        |
  AD21                =5.03594e+08 |                                        |
```

Computational graph example

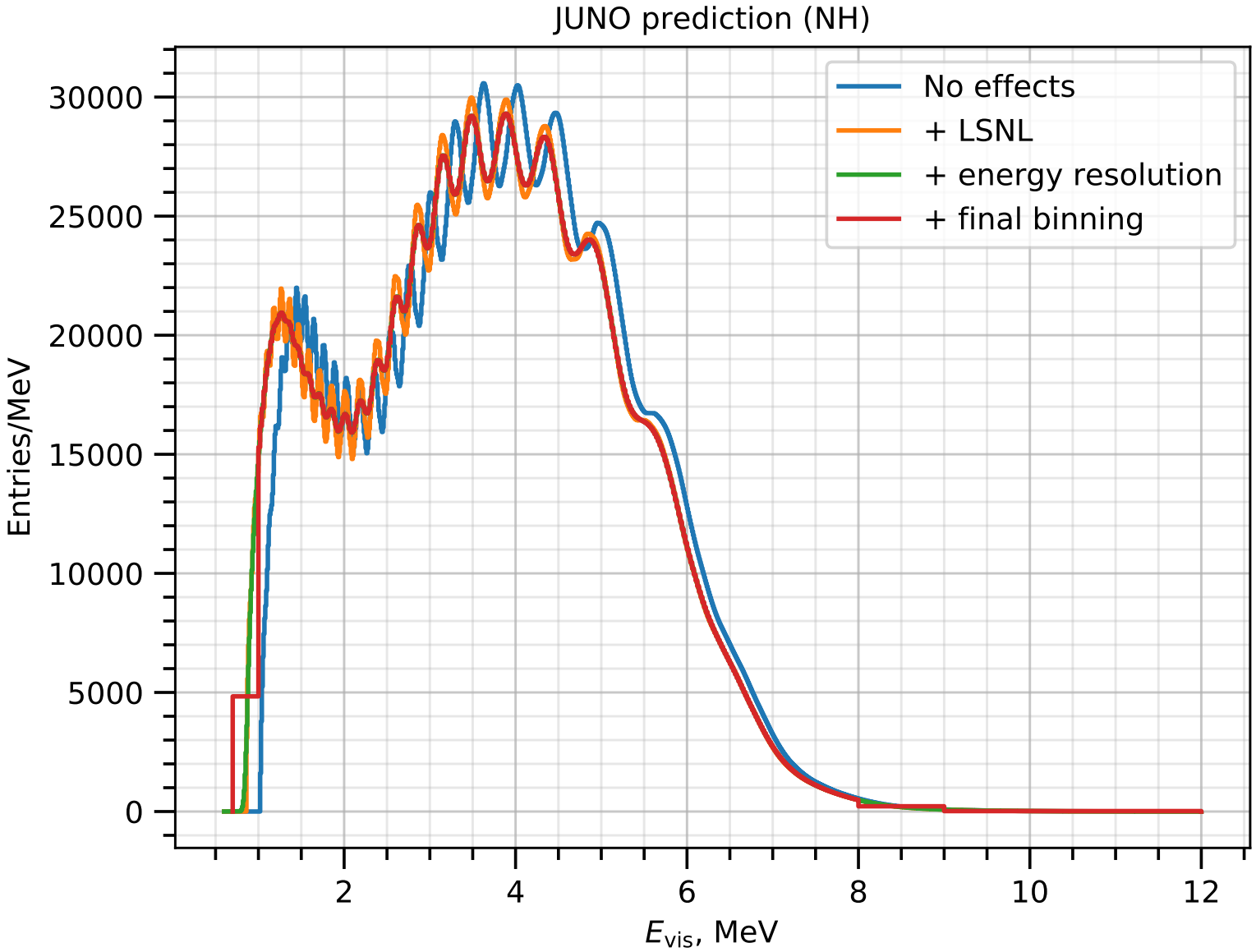
JUNO model



Graph: 481 nodes, 969 edges

Computational graph example

JUNO model

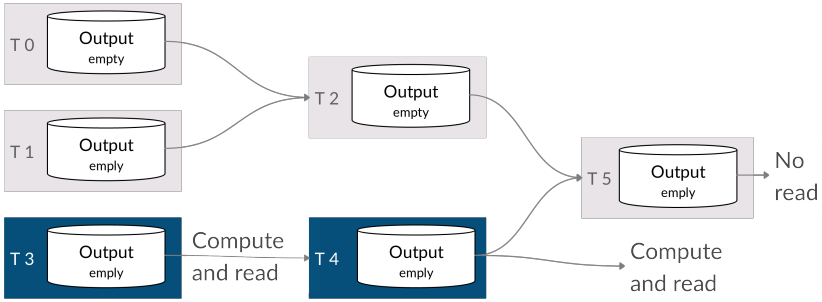


General features

Performance on any target machine

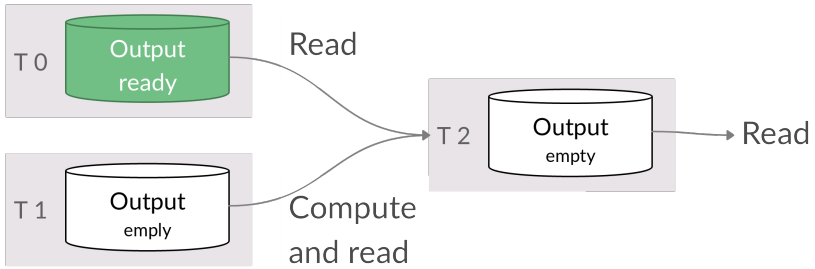
Lazy evaluation

- Subgraphs are to be computed after try to read its output. Only dependent subgraph is computed.



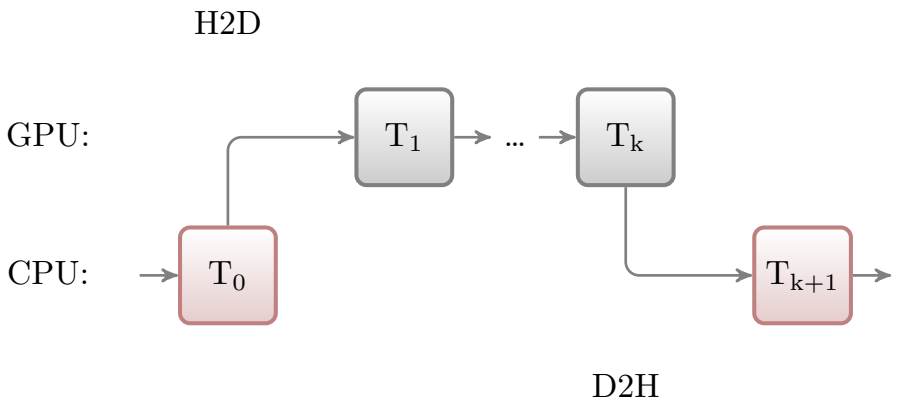
Caching

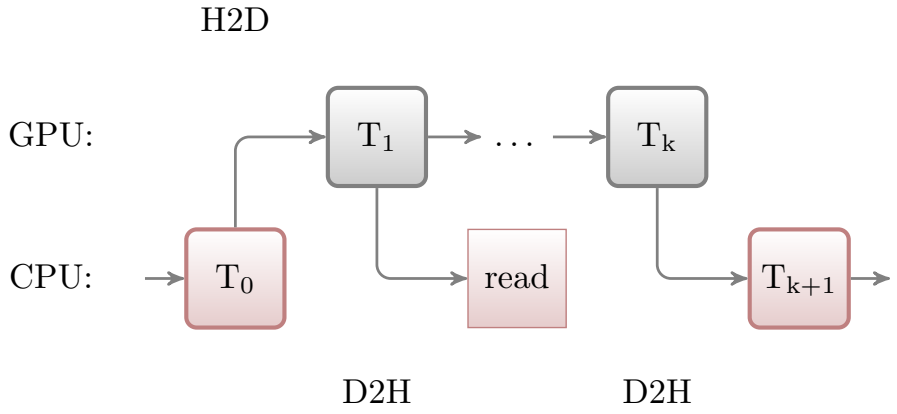
- If transformation was already computed for given input data there is no need to recompute it one more time.



What is cuGNA about?

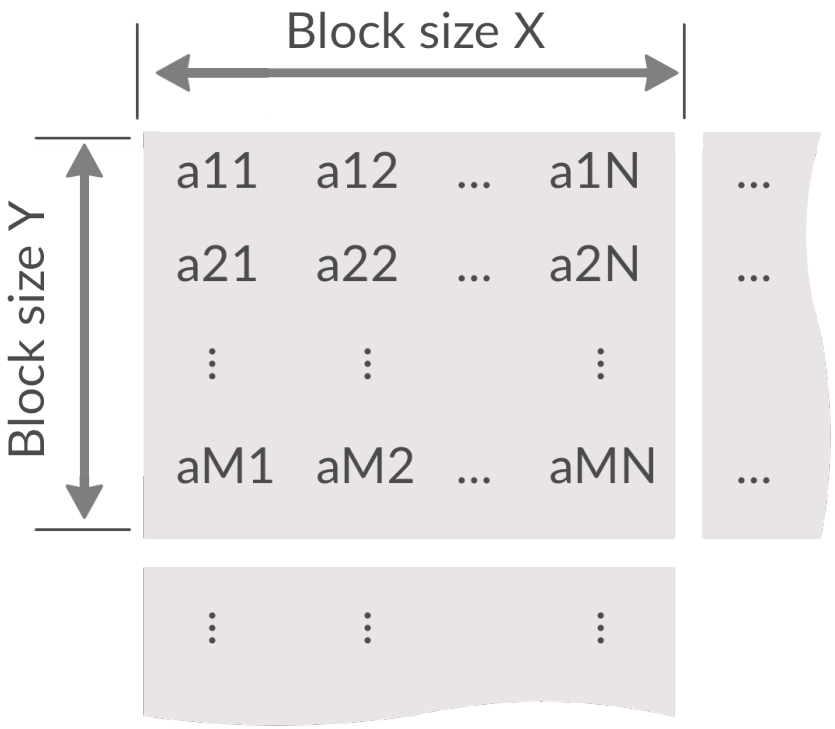
- Lazy data transfer tools.
- Unified data containers.
- An effortless switch between GPU and CPU computations.
- A set of predefined GPU-based transformations.
- General GNA style API.





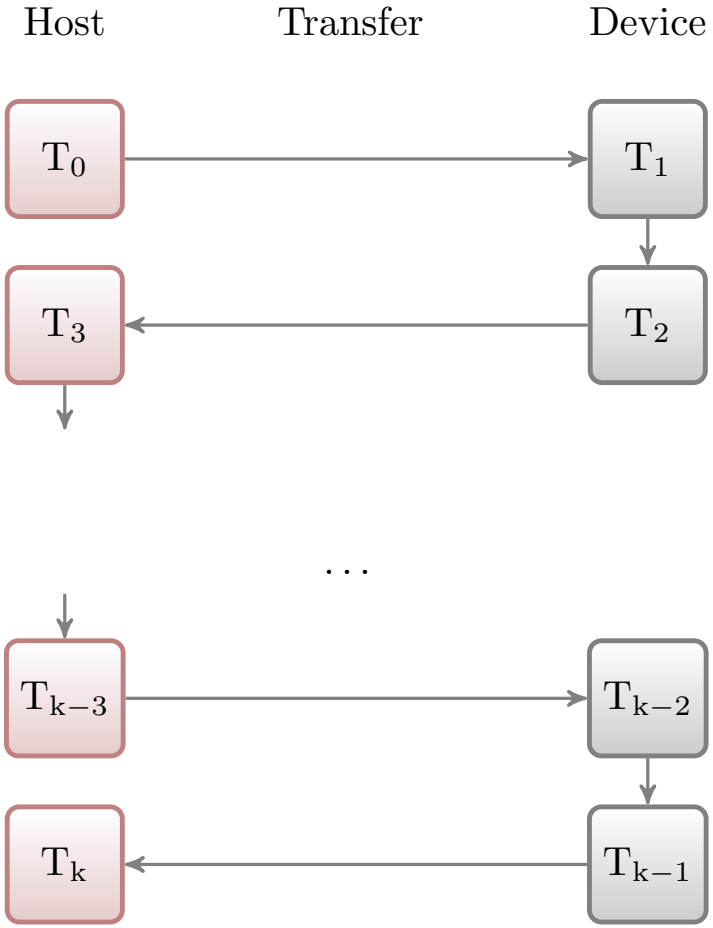
Why GPU?

- Most of GNA transformations operate on arrays/matrices.
- For most transformations almost no data dependency within a single transformation.
- Single memory allocation — multiple calls.



When you should not use it?

- Small size of input data arrays.
- Many data transfers are expected.
- Many data dependencies within a single transformation.
- Computationally easy tasks (addition, assignment, etc).



When it's definitely worth it?

Some simple rules

GPU architecture specific points

- More input data size.
- Less data dependecies.

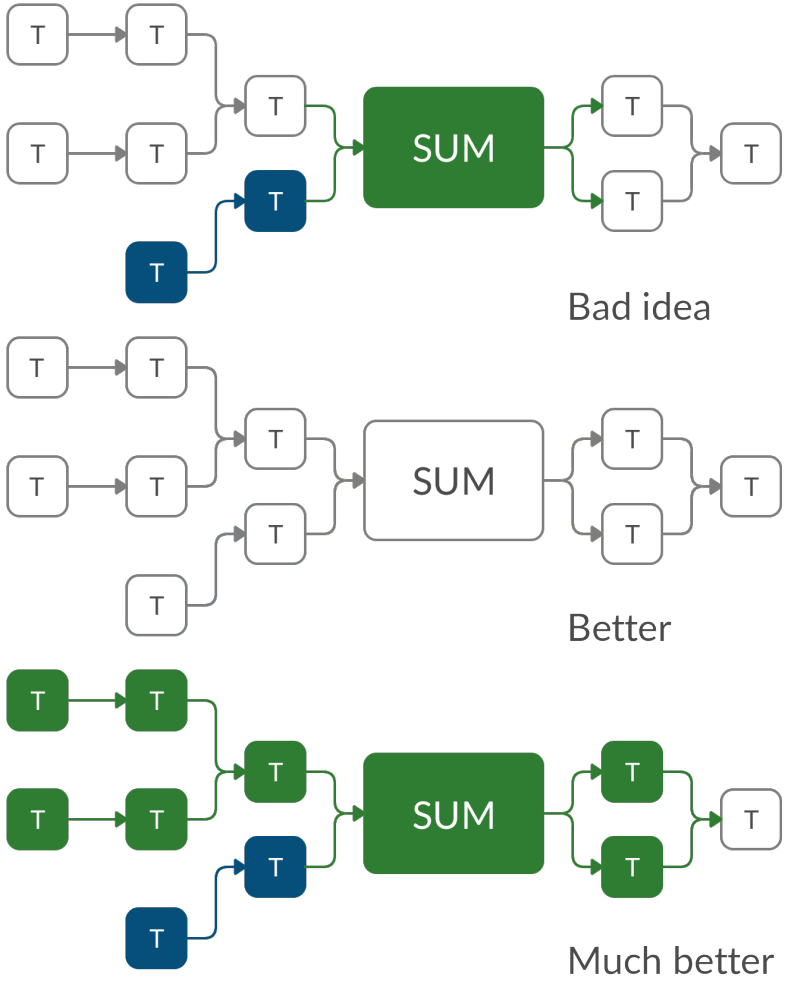
GNA specific points

- Port continuous subgraphs.
- Operations on big arrays or matrices.

GPU transformation or H2D/D2H.

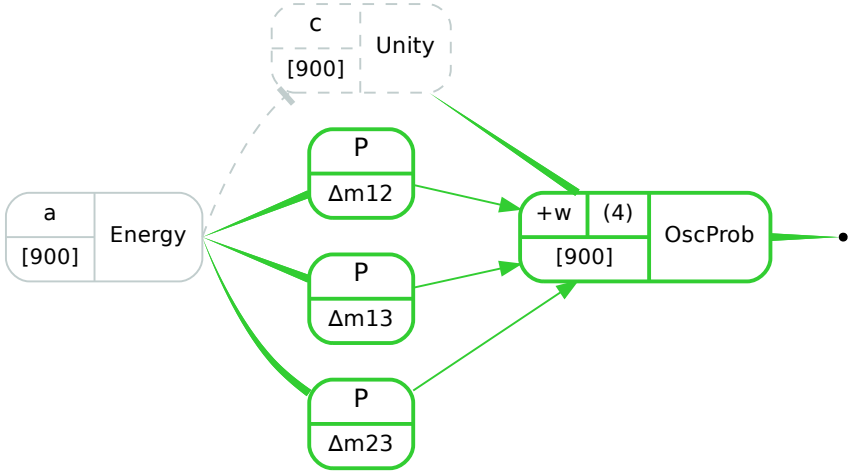
Recomputable CPU transformations.

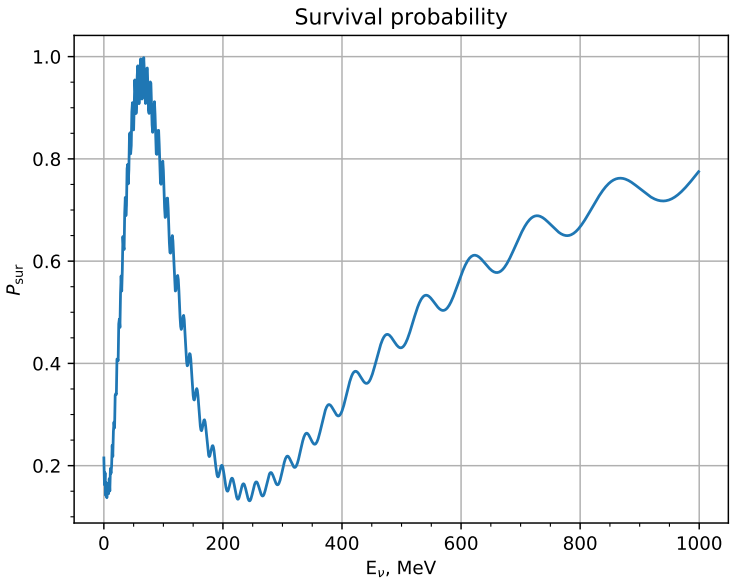
Static CPU transformations.



Examples

Oscillation probability





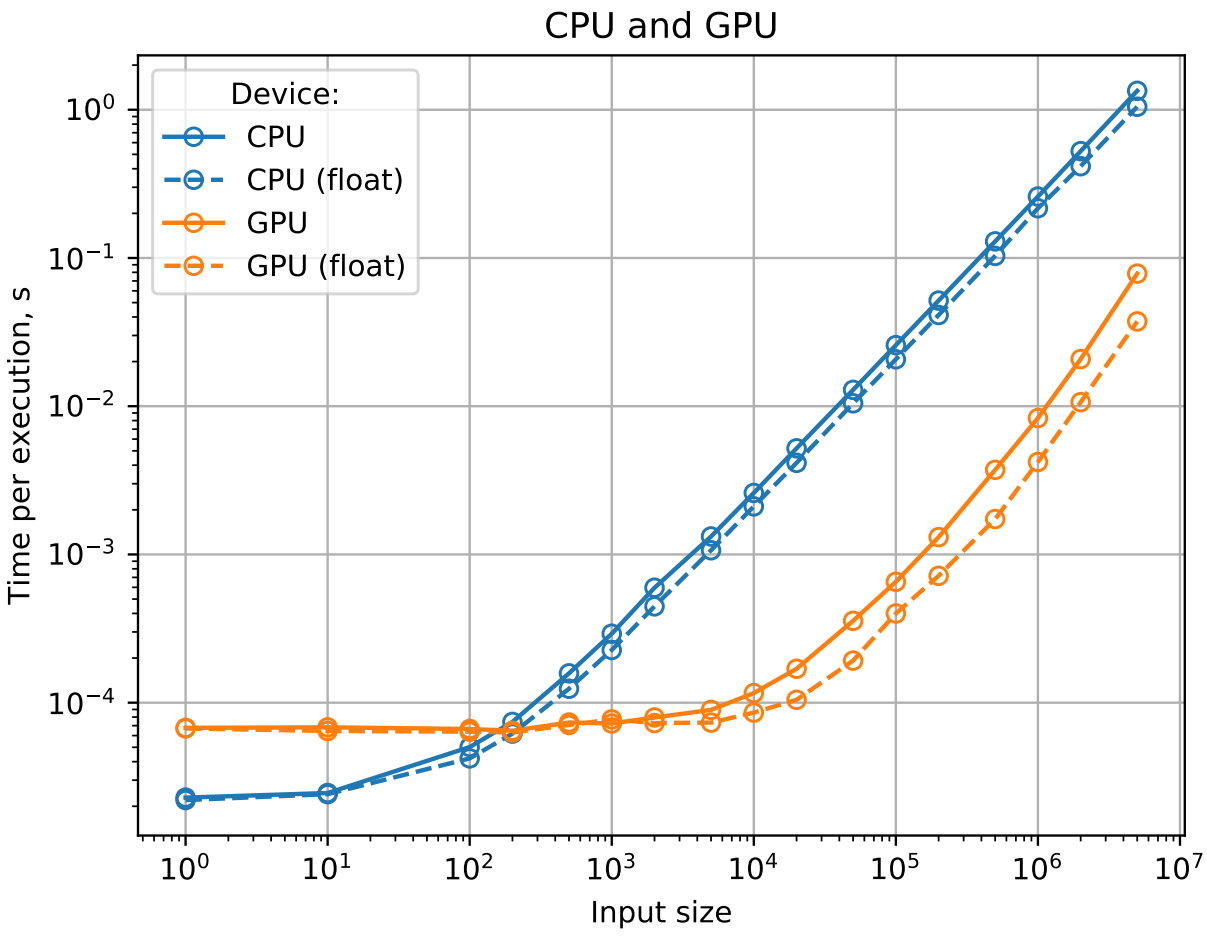
Functions to be compuited on GPU
Green arrow — D2D transfer (low costs)

Expanding edge — H2D transfer (costly)
Tapering edge — D2H tranfer (costly)

Examples

Oscillation probability

$$P(\nu_\alpha \to \nu_\beta) = \delta_{\alpha\beta} - 4\sum_{i>j} \mathrm{Re}(V^*_{\alpha i} V_{\beta i} V_{\alpha j} V^*_{\beta j}) \sin^2 \frac{\Delta m^2_{ij} L}{4E_\nu} + 2\sum_{i>j} \mathrm{Im}(V^*_{\alpha i} V_{\beta i} V_{\alpha j} V^*_{\beta j}) \sin \frac{\Delta m^2_{ij} L}{2E_\nu}$$



GPU: GeForce GTX 950M, CPU: Intel Core i7-7500U

Conclusion

GNA framework

- Flexible framework for data analysis of neutrino experiments.
- May be extended by user-defined transformations.
- Implemented Daya Bay and JUNO models.

GPU computations in GNA

- An effortless GPU support for statistical data analysis tasks.
- Common GNA style API.
- Acceleration on real size GNA tasks in tens times.

GNA:

Global Neutrino Analysis framework and GPU based computations

A. Fatkina M. Gonchar D. Naumov K. Treskov
L. Kolupaeva A. Kalitkina

JINR DLNP

NEC 2019

http://gna.pages.jinr.ru/gna/