9th International Conference in Deep Learning
in Computational Physics

ML-Based Optimum Sub-system Size Heuristic for the GPU Implementation of the Tridiagonal Partition Method

Milena Veneva

✓✉ milena.p.veneva@gmail.com Milena_Veneva mveneva

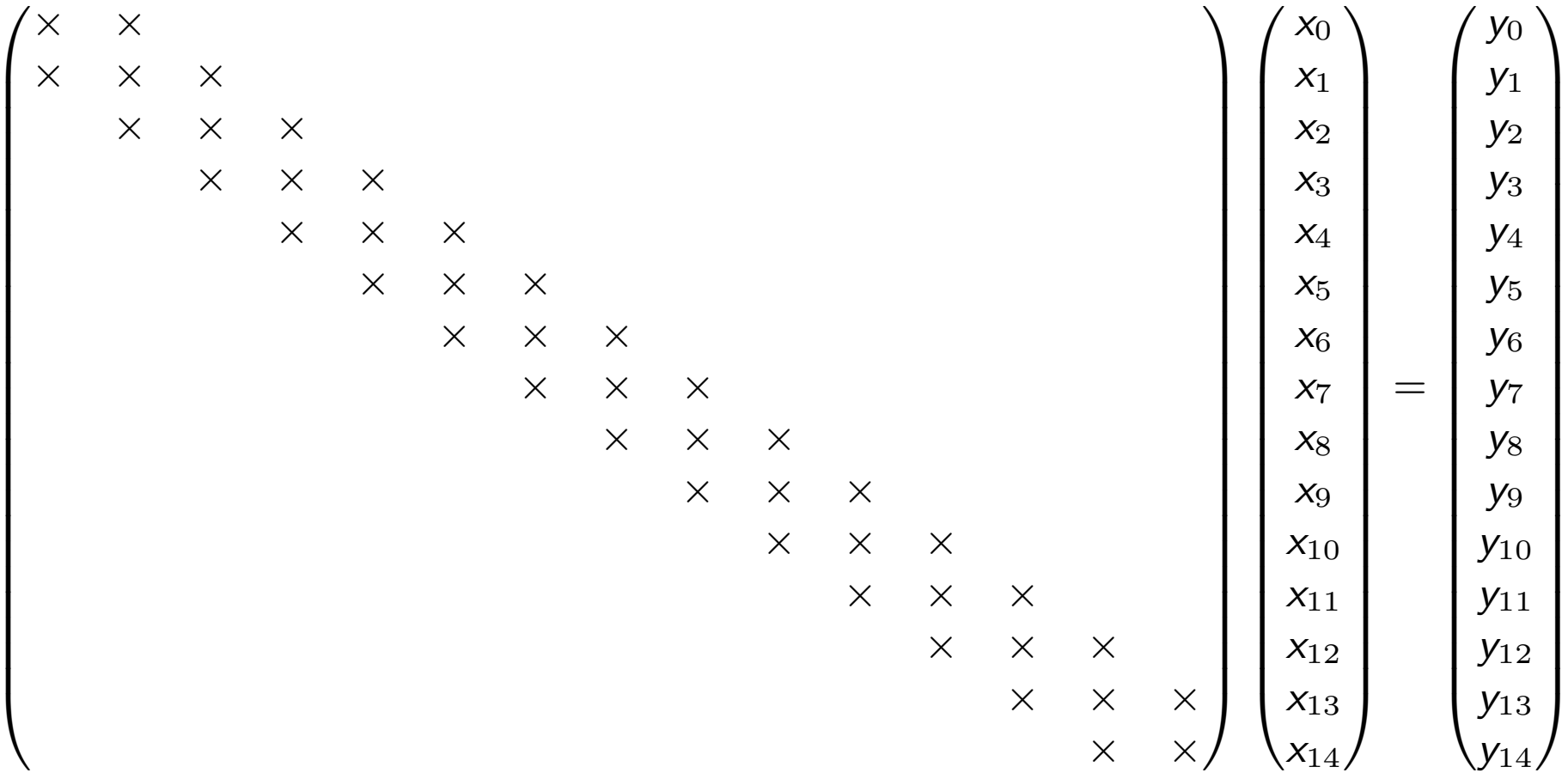
✓RIKEN Center for Computational Science, supervisor
Dr. Toshiyuki Imamura

✓in collaboration with the Joint Institute for Nuclear Research, supervisor **Dr. Alexander Ayriyan**

July 2025, Moscow, Russia.

Tridiagonal partition method (0/3)

0. Starting from the initial SLAE with the following coefficient matrix.



[1] Austin T. M., Berndt M., and Moulton J. D. *A memory efficient parallel tridiagonal solver*, pp .1–13, Preprint LA-VR-03-4149 (2004).

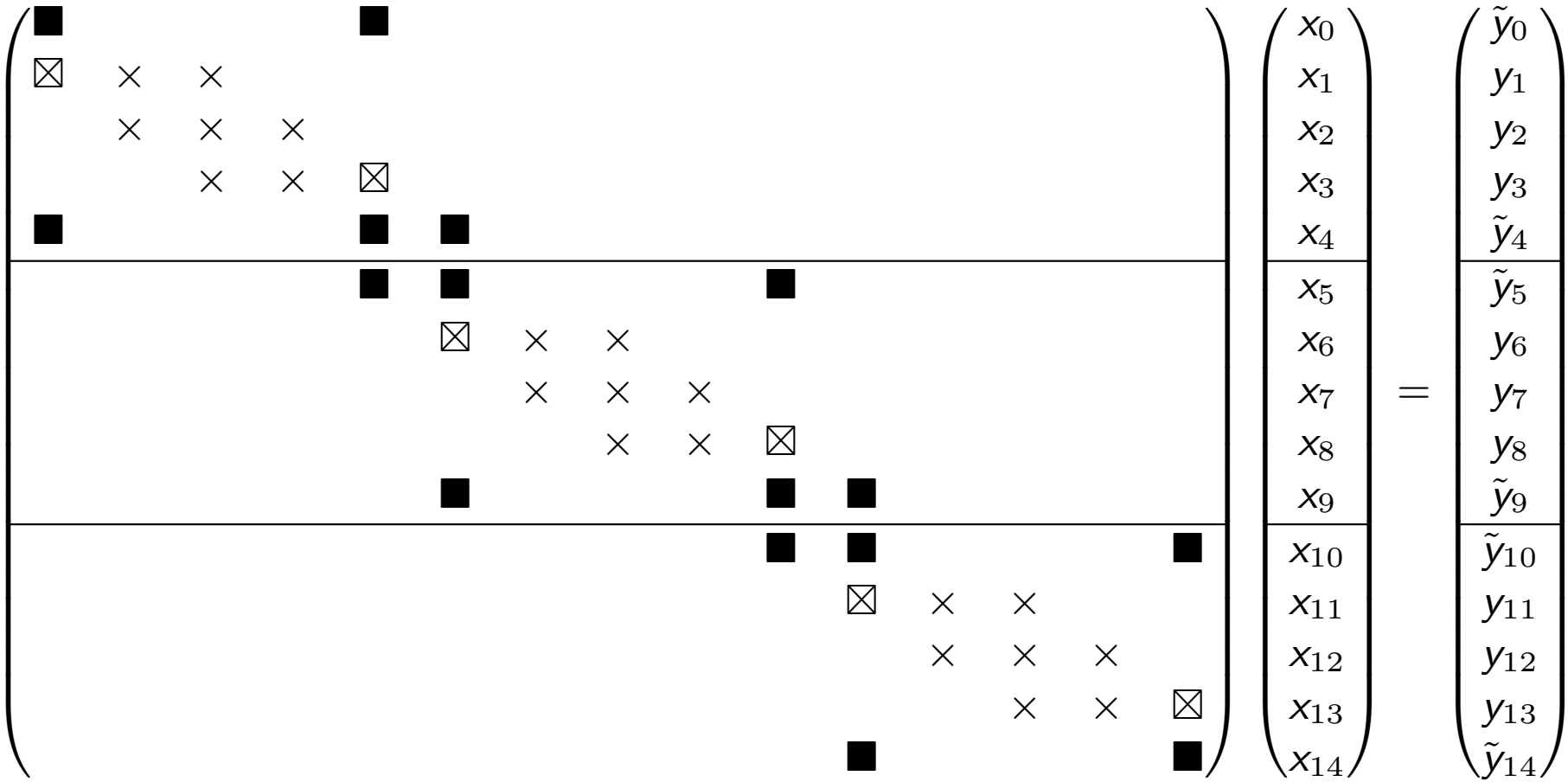
Tridiagonal partition method (1a/3)

1a. Partitioning.

$$
\left(\begin{array}{ccccccccccccc}
\times & \times & & & & & & & & & & & \\
\times & \times & \times & & & & & & & & & & \\
 & \times & \times & \times & & & & & & & & & \\
 & & \times & \times & \times & & & & & & & & \\
 & & & \times & \times & \times & & & & & & & \\
\hline
 & & & & \times & \times & \times & & & & & & \\
 & & & & & \times & \times & \times & & & & & \\
 & & & & & & \times & \times & \times & & & & \\
 & & & & & & & \times & \times & \times & & & \\
 & & & & & & & & \times & \times & \times & & \\
\hline
 & & & & & & & & & \times & \times & \times & \\
 & & & & & & & & & & \times & \times & \times \\
 & & & & & & & & & & & \times & \times
\end{array}\right)
\left(\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ \hline x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ \hline x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \end{array}\right)
=
\left(\begin{array}{c} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ \hline y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ \hline y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \end{array}\right)
$$

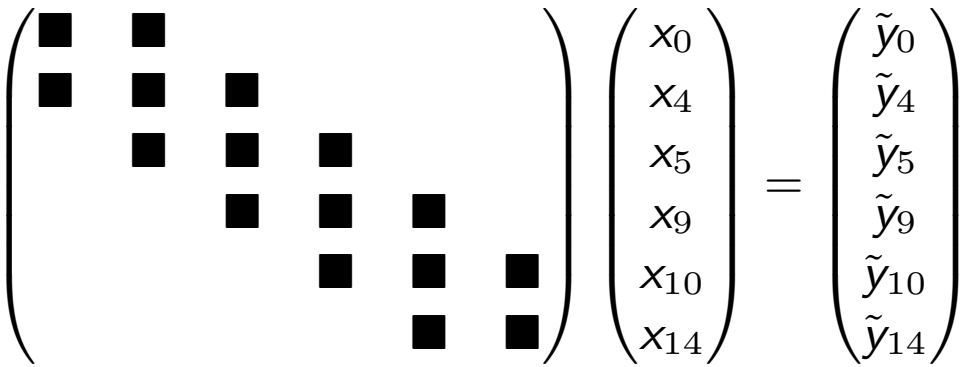
Tridiagonal partition method (1b/3)

1b. Obtaining the interface equations (on the device).



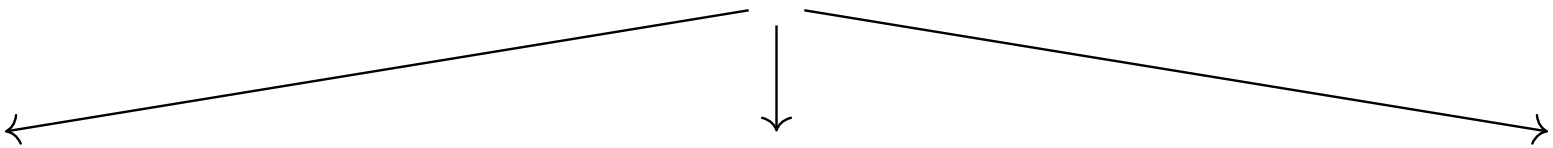
Tridiagonal partition method (2/3)

2. Assembling the interface system and solving it with the Thomas method (on the host).



Tridiagonal partition method (3/3)

3. Substituting the already found unknowns corresponding to the $\boxtimes$ coefficients, and solving the rest of the tridiagonal sub-systems (on the device).



$$\begin{pmatrix} \times & \times & \\ \times & \times & \times \\ & \times & \times \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \tilde{y}_1 \\ y_2 \\ \tilde{y}_3 \end{pmatrix} \quad \begin{pmatrix} \times & \times & \\ \times & \times & \times \\ & \times & \times \end{pmatrix} \begin{pmatrix} x_6 \\ x_7 \\ x_8 \end{pmatrix} = \begin{pmatrix} \tilde{y}_6 \\ y_7 \\ \tilde{y}_8 \end{pmatrix} \quad \begin{pmatrix} \times & \times & \\ \times & \times & \times \\ & \times & \times \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \end{pmatrix} = \begin{pmatrix} \tilde{y}_{11} \\ y_{12} \\ \tilde{y}_{13} \end{pmatrix}$$

Computational experiment

- on the basis of NVIDIA GPU RTX 2080 Ti;
- SLAE sizes: 10^i, 2×10^i, 4×10^i, 5×10^i, 8×10^i, $i = 2, 3, \ldots, 7$, and 4.5×10^3, 2.5×10^4, 3×10^4, 6×10^4, 7×10^4, 7.5×10^4;
- Parameters: 256 CUDA threads within a block were used; number of CUDA streams: according to [2]; precision FP64, _NO_WD interface option; no recursions; no overwriting of RHS;
- very many different sub-system sizes were tested;
- event synchronization when collecting the times.

---

[2] Veneva M., Imamura T., *ML-Based Optimum Number of CUDA Streams for the GPU Implementation of the Tridiagonal Partition Method*, arXiv:2501.05938 [cs.DC] (2025), to be published.

Considerations

- Consideration about blockSize: there should be enough warps in a block so as to keep the SM busy while one warp is waiting for resources (e. g. loading data from memory). Choosing the blockSize to be 256 gives the best performance. BlockSize smaller than 128 is not enough to supply the SM with enough active warps, and blockSize bigger than 256 requires too much resources (registers and shared memory), therefore the performance starts deteriorating.

- Balance between latency hiding, resource utilization, and occupancy.

- Within the partition method, the gridSize depends on the sub-system size m.

---

[3] NVIDIA, *CUDA C++ Best Practices Guide*, Release 12.9,

`https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/` (2025).

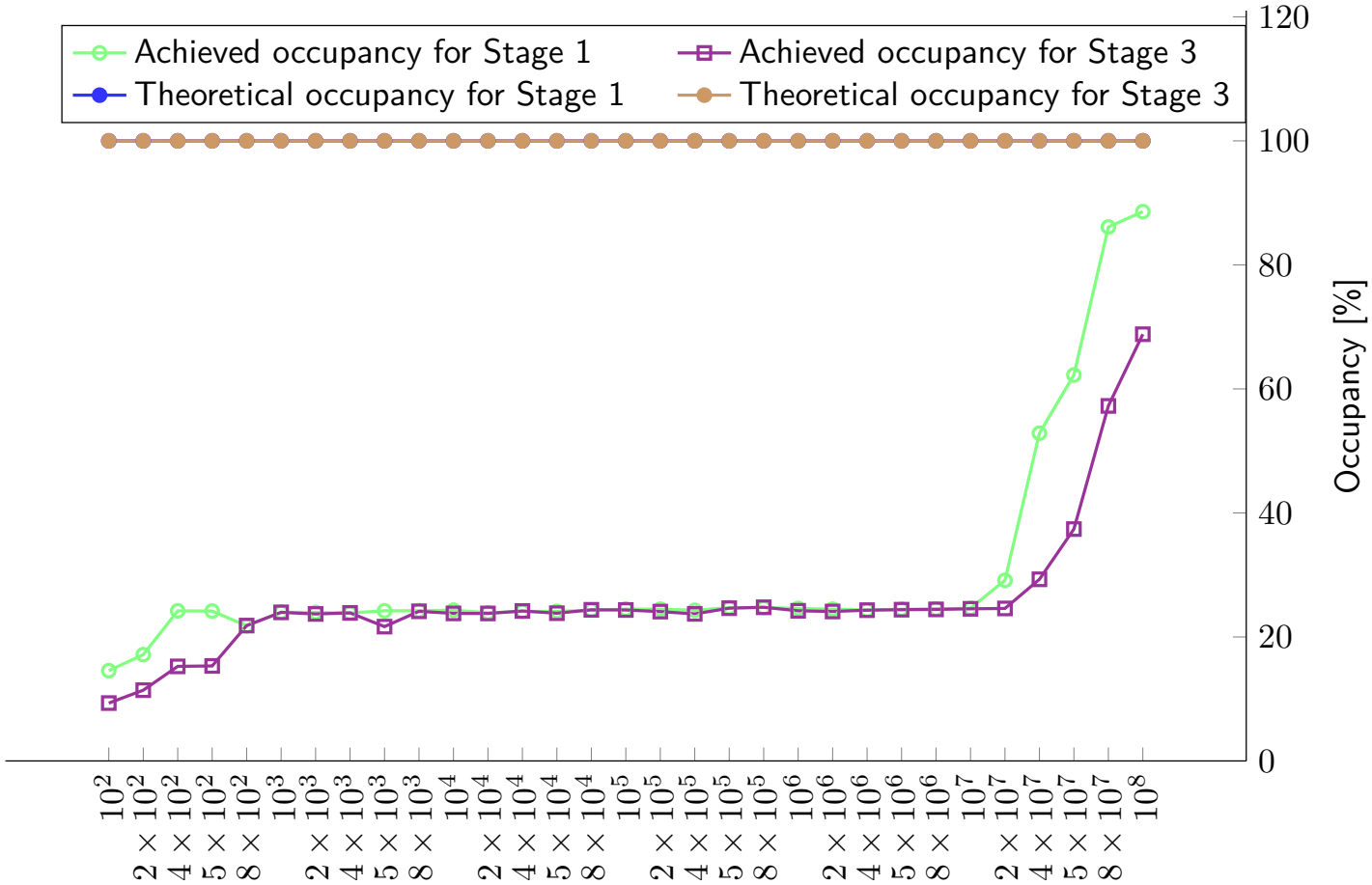
Existing heuristics

- Exhaustive search over all possible combinations of parameters (e. g. `QUDA`), but this requires additional runs of the application and is energy-costly. This approach is beneficial for applications that are run many times with the same parameters.
- Some promote a performance characteristic hoping that in the worst case scenario this approach is not going to lead to pathologically bad performance (e. g. occupancy in the case of `Thrust`).
- Some use two-step approach so as to gather information about the GPU limitations, and the kernel that needs to be launched (e. g. `KLARAPTOR`).
- Some use machine learning techniques to predict the combinations of parameters that would lead to optimal or near optimal performance (e. g. [4]).

---

[4] Sato K., Takizawa H., Komatsu K., and Kobayashi H.. *Automatic tuning of CUDA execution parameters for stencil processing*, In: Software Automatic Tuning, From Concepts to State-of-the-Art Results (2010).

Comparison between the achieved and the theoretical occupancy



Thus, the occupancy cannot be our main reference point.

SLAE size	opt m	#str	comments	corrected opt m
10^2	4	1	–	4
2×10^2	4	1	–	4
4×10^2	4	1	–	4
5×10^2	4	1	–	4
8×10^2	4	1	–	4
10^3	4	1	–	4
2×10^3	4	1	–	4
4×10^3	4	1	–	4
4.5×10^3	4	1	–	4
5×10^3	8	1	–	8
8×10^3	8	1	–	8
10^4	8	1	–	8
2×10^4	8	1	–	8
2.5×10^4	8	1	–	8
3×10^4	16	1	–	16
4×10^4	16	1	–	16
5×10^4	16	1	–	16
6×10^4	20	1	–	20
7×10^4	35	1	small diff with the time when $m = 20$: 0.0008099	20
7.5×10^4	40	1	small diff with the time when $m = 20$: 0.00719	20
8×10^4	32	1	–	32
10^5	40	1	small diff with the time when $m = 32$: 0.000621	32
2×10^5	64	2	small diff with the time when $m = 32$: 0.068788	32
4×10^5	64	4	small diff with the time when $m = 32$: 0.073638	32
5×10^5	40	8	small diff with the time when $m = 32$: 0.045666	32
8×10^5	64	8	diff with the time when $m = 32$: 0.182118	32
10^6	32	8	–	32
2×10^6	32	16	–	32
4×10^6	32	32	–	32
5×10^6	32	32	–	32
8×10^6	64	32	small diff with the time when $m = 32$: 0.44834	32
10^7	32	32	–	32
2×10^7	64	32	–	64
4×10^7	64	32	–	64
5×10^7	64	32	–	64
8×10^7	64	32	–	64
10^8	64	32	–	64

ML model for the optimum sub-system size (1/2)

- k-nearest neighbors (kNN) classification (independent variable: SLAE size; dependent (target variable): the optimum sub-system size).

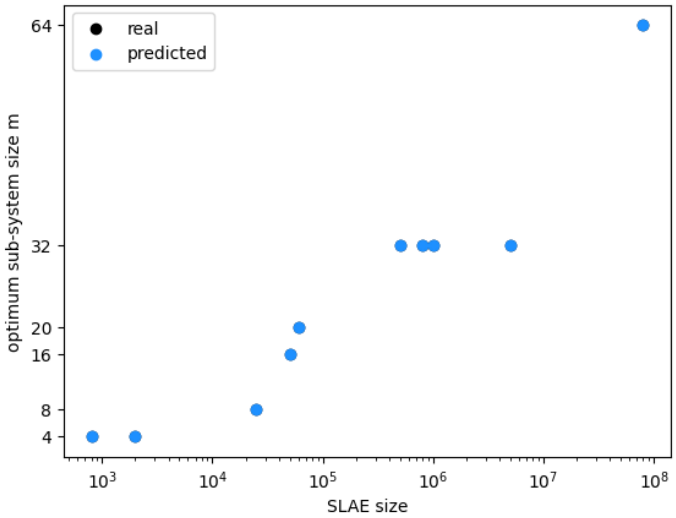
- k was found empirically by using `scikit-learn` tool `GridSearchCV` (which looks for the best combination of hyper-parameters, in particular we looked for the best k (between 1, and the number of unique sub-system sizes), and best weight – 'uniform' or 'distance') to be equal to 1 (and 'uniform' weight).

- The data was split into two data sets – training (that is, neighbours), and test (for which we make predictions) – using the Python scikit-learn routine `train_test_split` with shuffle option turned on, and splitting ratio $3:1$.

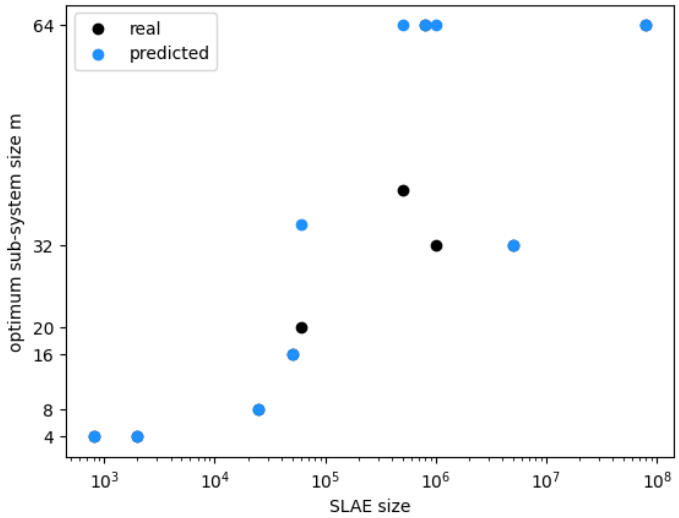
ML model for the optimum sub-system size (2/2)

- Two approaches when fitting the data: (1) using the experimentally found optimum sub-system sizes, and (2) using the corrected sub-system sizes which takes into account the trend we have noticed.
- The former approach gave us 0.7 normalised accuracy score for the test set, which means that it manages to find the expected sub-system size in 7 out of 10 cases.
- On the other hand, using the latter idea, we get 1.0 normalised accuracy score, that is, 100% accuracy.
- The null accuracy (accuracy that could be achieved by always predicting the most frequently met sub-system size) was found to be 0.4.
- The results are in a good correspondence with the memory alignments considerations (aligned to at least 256 bytes).

Results from the kNN classification model



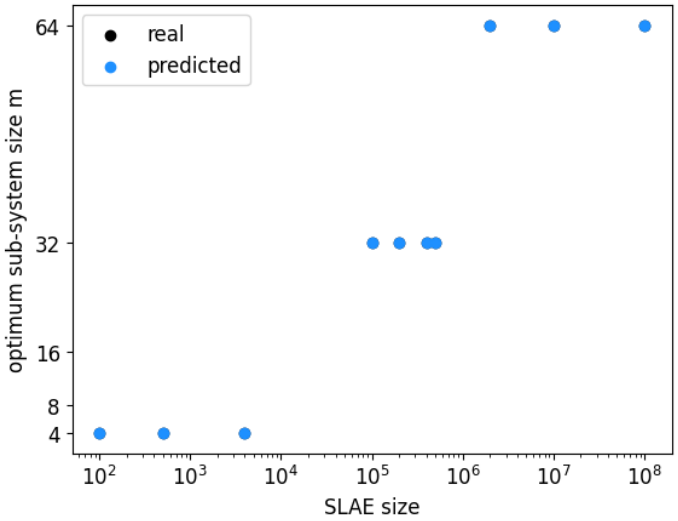
(a) Results from the kNN classification model for the optimum sub-system size using the corrected data for m.



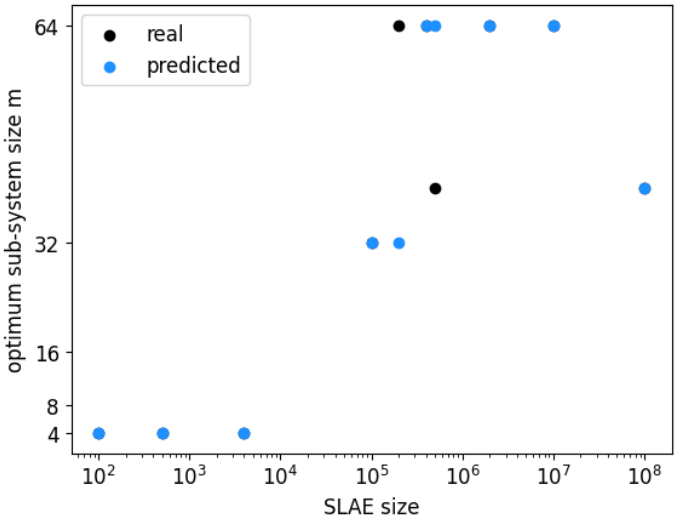
(b) Results from the kNN classification model for the optimum sub-system size using the observed data for m.

Changing from sub-optimal to optimal (bigger) sub-system size might give us speed-up up to 1.70 (for number_of_lines $= 8 \times 10^7$ compare $m = 64$ and $m = 4$).

Results for FP32



(a) Results from the kNN classification model for the optimum sub-system size using the corrected data for m.



(b) Results from the kNN classification model for the optimum sub-system size using the observed data for m.

- 0.8 normalised accuracy score for the test set for the observed data,
- 1.0 normalised accuracy score for the test set for the corrected data,
- null accuracy of 0.4.

SLAE size	#str	opt m 2080 Ti	heuristic on 2080 Ti	opt m A5000	diff with the heuristic	opt m 4080	diff with the heuristic
10^2	1	4	4	4	–	4	–
2×10^2	1	4	4	4	–	4	–
4×10^2	1	4	4	4	–	4	–
5×10^2	1	4	4	4	–	4	–
8×10^2	1	4	4	4	–	8	small
10^3	1	4	4	4	–	4	–
2×10^3	1	4	4	4	–	4	–
4×10^3	1	4	4	8	small	8	small
4.5×10^3	1	4	4	4	–	4	–
5×10^3	1	8	8	4	small	4	small
8×10^3	1	8	8	8	–	4	small
10^4	1	8	8	8	–	8	–
2×10^4	1	8	8	8	–	16	small
2.5×10^4	1	8	8	8		8	–
3×10^4	1	16	16	16	–	16	–
4×10^4	1	16	16	16	–	16	–
5×10^4	1	16	16	16	–	16	–
6×10^4	1	20	20	32	2.65%	40	small
7×10^4	1	35	20	20	–	20	–
7.5×10^4	1	40	20	20	–	40	small
8×10^4	1	32	32	40	small	32	–
10^5	1	40	32	32	–	32	–
2×10^5	2	64	32	**64**	6.26%	**64**	4.59%
4×10^5	3	64	32	64	3.54%	64	small
5×10^5	8	40	32	40	2.38%	40	4.19%
8×10^5	8	64	32	64	6.03%	64	2.50%
10^6	8	32	32	**64**	9.44%	**64**	7.13%
2×10^6	16	32	32	**64**	8.15%	**64**	6.00%
4×10^6	32	32	32	**64**	5.60%	**64**	6.90%
5×10^6	32	32	32	**64**	3.65%	**64**	5.66%
8×10^6	32	64	32	64	5.63%	64	7.09%
10^7	32	32	32	**64**	6.06%	**64**	6.75%
2×10^7	32	64	64	64	–	64	–
4×10^7	32	64	64	64	–	64	–
5×10^7	32	64	64	64	–	64	–
8×10^7	32	64	64	64	–	64	–
10^8	32	64	64	64	–	64	–

Thank you for your attention!

Acknowledgements:

R-CCS team (RIKEN),

Dr. Toshiyuki Imamura (RIKEN),

Dr. Alexander Ayryan (JINR).