Mathematical Modeling and Computational Physics

ML-Based Optimum Number of CUDA Streams for the GPU Implementation of the Tridiagonal Partition Method

Milena Veneva, Toshiyuki Imamura

✓RIKEN Center for Computational Science

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

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

October 2024, Yerevan, Armenia.

Tridiagonal partition method (0/3)

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

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

[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).

1a. Partitioning.

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

Tridiagonal partition method (1b/3)

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

$$
\left(\begin{array}{ccccccccccccccc}
\blacksquare & & & & \blacksquare & & & & & & & & & & \\
\boxtimes & \times & \times & & & & & & & & & & & & \\
 & \times & \times & \times & & & & & & & & & & & \\
 & & \times & \times & \boxtimes & & & & & & & & & & \\
\blacksquare & & & & \blacksquare & \blacksquare & & & & & & & & & \\
\hline
 & & & & \blacksquare & \blacksquare & & & & \blacksquare & & & & & \\
 & & & & & \boxtimes & \times & \times & & & & & & & \\
 & & & & & & \times & \times & \times & & & & & & \\
 & & & & & & & \times & \times & \boxtimes & & & & & \\
 & & & & & \blacksquare & & & & \blacksquare & \blacksquare & & & & \\
\hline
 & & & & & & & & & \blacksquare & \blacksquare & & & & \blacksquare \\
 & & & & & & & & & & \boxtimes & \times & \times & & \\
 & & & & & & & & & & & \times & \times & \times & \\
 & & & & & & & & & & & & \times & \times & \boxtimes \\
 & & & & & & & & & & \blacksquare & & & & \blacksquare
\end{array}\right)
\left(\begin{array}{c}
x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ \hline x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ \hline x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14}
\end{array}\right)
=
\left(\begin{array}{c}
\tilde{y}_0 \\ y_1 \\ y_2 \\ y_3 \\ \tilde{y}_4 \\ \hline \tilde{y}_5 \\ y_6 \\ y_7 \\ y_8 \\ \tilde{y}_9 \\ \hline \tilde{y}_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ \tilde{y}_{14}
\end{array}\right)
$$

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

$$\begin{pmatrix} \blacksquare & \blacksquare & & & & \\ \blacksquare & \blacksquare & \blacksquare & & & \\ & \blacksquare & \blacksquare & \blacksquare & & \\ & & \blacksquare & \blacksquare & \blacksquare & \\ & & & \blacksquare & \blacksquare & \blacksquare \\ & & & & \blacksquare & \blacksquare \end{pmatrix} \begin{pmatrix} x_0 \\ x_4 \\ x_5 \\ x_9 \\ x_{10} \\ x_{14} \end{pmatrix} = \begin{pmatrix} \tilde{y}_0 \\ \tilde{y}_4 \\ \tilde{y}_5 \\ \tilde{y}_9 \\ \tilde{y}_{10} \\ \tilde{y}_{14} \end{pmatrix}$$

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, 2.5×10^i, 4×10^i, 5×10^i, 7.5×10^i, and 8×10^i, $i = \overline{3, 7}$;
- Parameters: the sub-system size was set to $m = 10$; 256 CUDA threads within a block were used; precision FP64, _NO_WD interface option; no recursions; no overwriting of RHS;
- number of CUDA streams: powers of 2, up to 32 (hardware working queues);
- event synchronization when collecting the times.

---

[2] Werkhoven B. van, Maassen J., Seinstra F. J., and Bal H. E., *Performance models for CPU-GPU data transfers*, 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 11–20 (2014).

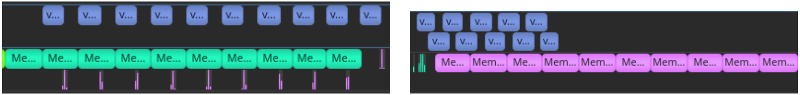
Time complexity model

$$T = \left(T_1^{H2D} + T_1^{COMP} + T_1^{D2H}\right) + T_2^{H} + \left(T_3^{H2D} + T_3^{COMP} + T_3^{D2H}\right),$$

where T_i is the memory transfer/kernel time of Stage i, $i = \overline{1,3}$.

$$T_{\text{str}} = T_1^{H2D} + \frac{T_1^{COMP} + T_1^{D2H} + T_3^{H2D} + T_3^{COMP}}{\text{num\_str}} + T_2^{H} + T_3^{D2H} + T_{\text{overhead}},$$

where T_{overhead} is the overhead from the creation of CUDA streams.



(a) Stage 1 (b) Stage 3

Existing heuristics

$$T_{\text{overhead}} = \tau \times \text{num\_str (according to [2])},$$

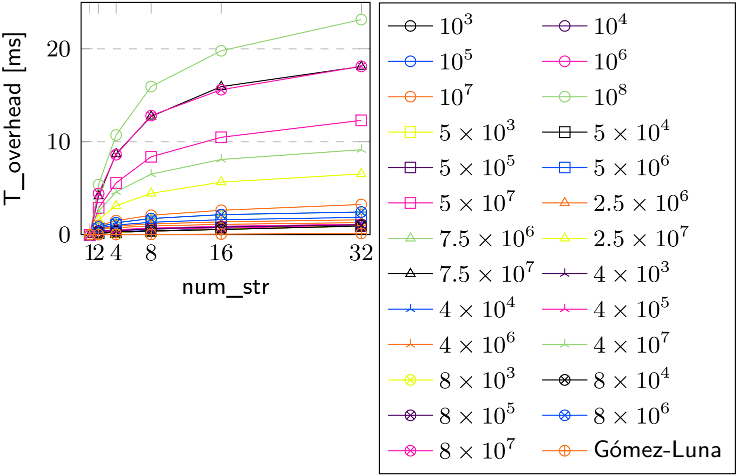
where τ is the time for creating one stream ($\tau = 0.004448$ ms for NVIDIA GPU RTX 2080 Ti). Hence,

$$\text{num\_str\_opt} = \sqrt{\frac{T_1^{COMP} + T_1^{D2H} + T_3^{H2D} + T_3^{COMP}}{\tau}}.$$

SLAE size	T_1^{COMP}	T_1^{D2H}	T_3^{H2D}	T_3^{COMP}	sum	opt num_str Gómez-Luna	opt num_str actual
4×10^3	0.221312	0.014848	0.006592	0.030688	0.27344	7.8	1
4×10^4	0.216544	0.057312	0.015456	0.038112	0.327424	8.6	1
4×10^5	0.393184	0.402944	0.102784	0.205408	1.10432	15.8	4
4×10^6	1.99398	3.89741	0.975392	2.1305	8.997282	45.0	32
4×10^7	17.4515	38.8368	9.60672	20.9816	86.87662	139.8	32

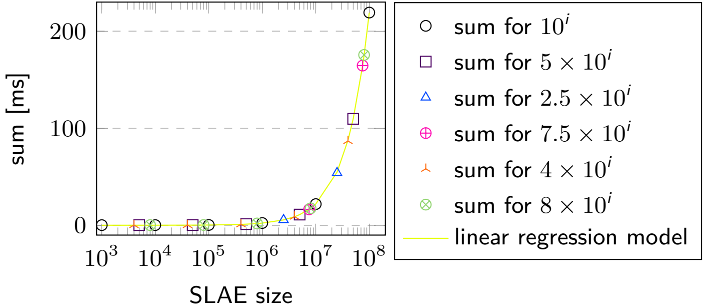
[3] Gómez-Luna J., González-Linares J. M., Benavides J. I. and Guil N., *Performance models for CUDA streams on NVIDIA GeForce series*, J. Parallel Distrib. Comput., 72, **9**, pp. 1117–1126 (2011).

T_overhead for different SLAE sizes, on different number of streams



Mathematical model for sum

$$\text{sum} = T_1^{COMP} + T_1^{D2H} + T_3^{H2D} + T_3^{COMP}.$$



Model (regression analysis, splitting ratio $3:1$, shuffle turned on):

$$\text{sum\_model} = 0.0000021890017149 \times \text{SLAE\_size} + 0.1470644998564126.$$

R^2: training – 0.9999813476643502, test – 0.9999942108504311.
The mean squared error (MSE) is 0.02 (test).

Mathematical model for T_overhead

$$T_{overhead} = (T_{str} - T_{non\_str}) + \frac{\mathrm{num\_str} - 1}{\mathrm{num\_str}} \times \mathrm{sum}.$$

Algorithm: the optimum number of streams comes when $T_{overhead}$ is smaller than the second term on the right-hand side of the equation above, and the difference between them is the biggest among the ones that fulfill the inequality.

Models (`SciPy` routine `curve_fit`, the form of the functions is preset, splitting ratio $3:1$, shuffle turned on; *small* for SLAE sizes $\leq 10^6$, and *big* – sizes $> 10^6$):

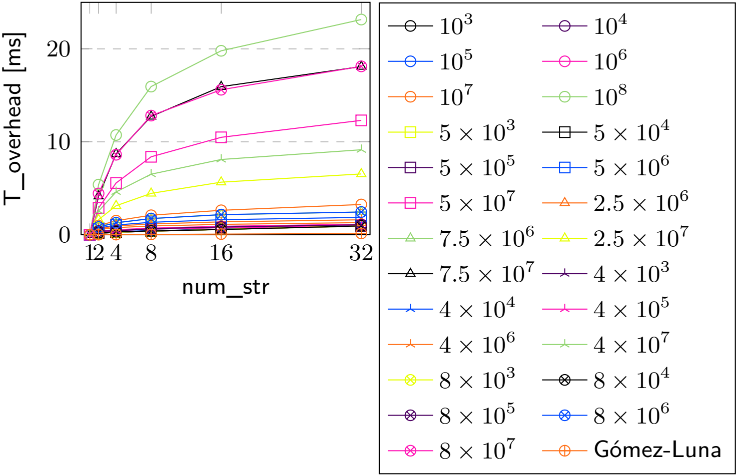
$$\mathrm{T\_overhead\_model\_small} = 0.0000002245645331 \times \mathrm{SLAE\_size}$$

$$+ 0.6009426920043296 \times \log_{10}(\mathrm{num\_streams}) - 0.0605183610625299,$$

$$\mathrm{T\_overhead\_model\_big} = (0.000000356594859 \times \mathrm{SLAE\_size}$$

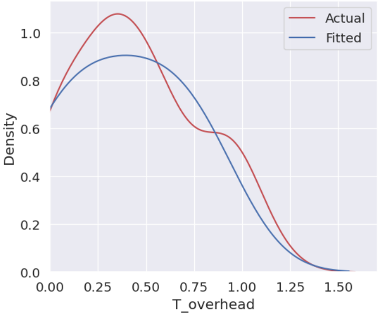
$$+ 0.0522781620855163) \times \log_2(\mathrm{num\_streams}^{\frac{4}{3}}) + 0.3941472844770443.$$

T_overhead for different SLAE sizes, on different number of streams

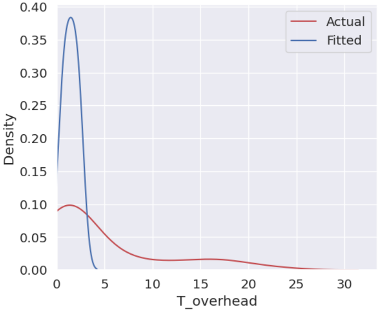


Models' metrics

set	metric	model_small	model_big
training	R-squared	0.9531711290769591	0.9933780389080090
	MSE = mean squared error	0.0050126881205798	0.2451169015984794
	RMSE $= \sqrt{\mathrm{MSE}}$	0.0708003398337877	0.4950928211946518
test	R-squared	0.9549695579010460	0.9896761975222511
	MSE = mean squared error	0.0044441139999724	0.1447752928068124
	RMSE $= \sqrt{\mathrm{MSE}}$	0.0666641882870588	0.3804934858927448



(a) model_small



(b) model_big

Predictions based on the models

SLAE size	actual	predicted	size	actual	predicted	size	actual	predicted
10^3	1	1	4×10^5	4	4	10^7	32	32
4×10^3	1	1	5×10^5	8	**4**	2.5×10^7	32	32
5×10^3	1	1	8×10^5	8	8	4×10^7	32	32
8×10^3	1	1	10^6	8	8	5×10^7	32	32
10^4	1	1	2.5×10^6	16	16	7.5×10^7	32	32
4×10^4	1	1	4×10^6	32	32	8×10^7	32	32
5×10^4	1	1	5×10^6	32	32	10^8	32	32
8×10^4	1	1	7.5×10^6	32	32			
10^5	1	**2**	8×10^6	32	32			

- SLAE size 10^5: 2 vs. 1 stream
  time diff $= 1.239040 - 1.230176 = 0.008864$ ms.
- SLAE size 5×10^5: 4 vs. 8 streams
  time diff $= 4.046112 - 4.029888 = 0.016224$ ms.

Performance improvement when using optimum number of CUDA streams: up to 1.30 (for SLAE sizes 8×10^7 and 10^8).

Thank you for your attention!

Acknowledgements:
R-CCS team (RIKEN),
Dr. Alexander Ayryan (JINR).