DOI: 10.15439/2018F271
ISSN 2300-5963 ACSIS, Vol. 15

An effective sparse storage scheme for GPU-enabled uniformization method

Beata Bylina, Jarosław Bylina, Marek Karwacki
Marie Curie-Skłodowska University, Institute of Mathematics,
Pl. M. Curie-Skłodowskiej 5, 20-031 Lublin, Poland
Email: `{beata.bylina,jaroslaw.bylina}@umcs.pl`

***Abstract*—The authors developed a GPU approach to the uniformization method for the computing transient solution of Markov models. The authors use two techniques to reduce the memory size of storing matrices. One of them is a modification of a storage sparse matrix format HYB; second is to utilize two GPU cards and the multicore CPU. The modified HYB format is suitable for sparse Markovian transition rate matrices and oversized matrices on single GPU, also improving computation performance at the same time. The use of two GPUs enables processing matrices of even bigger sizes.**

I. Introduction

A POWERFUL tool used widely for modeling a lot of processes and systems (natural and artificial ones) are Markov chains.

In [2], we provided details of a heterogeneous (CPU-GPU) implementation of the uniformization method [6], [11], [12] for solving Markov chains. Markov chains transition rate matrices (which are very sparse) were stored with the use of the HYB format which is a hybrid of other well-known sparse formats (ELL and COO). The HYB format was chosen because it gave the best results in experiments described in [1].

However, matrices in question are usually very large and do not fit into one GPU memory. Thus, in [3], we presented an implementation of uniformization for many GPU. Notwithstanding, the communication between GPUs was slow and the results were not satisfactory. Hence, in [4] we described an effective storage scheme for Markov chains transition rate matrices, namely HYBIV, which reduced the memory requirements and the number of miss caches and thereby improved the overall performance. That format was not studied for uniformization though.

This work shows numerical experiments where that format (HYBIV) is used for uniformization and tested for four groups of transition rate matrices (from PRISM). It also compares the HYB format with HYBIV (on 1 and 2 GPUs) and with CSR (on CPU).

The structure of the article is following. Section II describes the conducted numerical experiments. Section III presents the memory usage for formats used in experiments. In Section IV, the performance time of the experiments is analyzed. Section V concludes the experiments and the paper.

II. Methodology of Numerical Experiments

The memory requirements were tested and compared in this section, as well as the time of the uniformization algorithm, with the use of three storage schemes:

- HYB — the original format from the CUSP library;
- HYBIV — the modified format;
- the well-known CSR format (as the most common and often the most efficient format for CPUs), on CPU, with the use of the MKL library [13].

We were interested in studying and comparing memory required by the original HYB format and for our modified HYBIV format — on one GPU and on two GPUs. Also, the times elapsed by the uniformization method with the use of these formats for one and two GPUs were compared. The comparison between times of these formats on GPUs with the use of the CUSP library and the CSR format on CPU with the use of the MKL library was done.

All the codes are written in C++. We tested the uniformization on CPU and GPU under Linux with `gcc` and NVIDIA `nvcc` compilers with optimization flag `-O3`. The experiments were run on an Intel system with 12 cores. The Intel system has two sockets with six-core Intel Xeon X5650 clocked at 2.67 GHz and 48 GB memory. In the performance evaluation were used NVIDIA GPUs (2× Tesla M2050 with 3 GB memory) and libraries: CUDA Toolkit 4.0, CUSP 0.2, MKL 10.3.

We tested the implementations on four widely used benchmark models: mutex [7], a Kanban system [5], a cyclic server Polling system [9], tandem queueing network [10].

These protocols were chosen due to their scalability and the possibility to verify their properties by numerical solving. The first model (MUTEX) is generated by the authors, which allowed scaling up this model. The remaining three models were generated using PRISM [8] (unfortunately, we were unable to generate matrices of bigger size), a probabilistic model checker developed at the University of Birmingham.

Tables I and II present details of test matrices, where:

- *name* is the name of the matrix with parameters describing the model
- n is the number of rows,
- nz is the number of non-zero elements,
- nz/n represents the matrix density,
- uv is the number of unique values of matrix' elements,
- x is the size of the ELL part,
- c is the size of the COO part.

TABLE I
PROPERTIES OF THE MATRICES FOR THE 'MUTEX' MODELS

#	name	n	nz	nz/n	uv	x	c
1	MUTEX__N_16__R_4	2517	20949	8.32	654	0	20949
2	MUTEX__N_12__R_7	3302	38966	11.80	1545	0	38966
3	MUTEX__N_12__R_8	3797	47381	12.48	1871	0	47381
4	MUTEX__N_12__R_9	4017	51561	12.84	2030	0	51561
5	MUTEX__N_12__R_10	4083	52947	12.97	2078	0	52947
6	MUTEX__N_12__R_11	4095	53223	13.00	2081	0	53223
7	MUTEX__N_20__R_4	6196	52596	8.49	1259	5	21616
8	MUTEX__N_16__R_5	6885	68997	10.02	2008	6	27687
9	MUTEX__N_20__R_5	21700	223140	10.28	5067	6	92940
10	MUTEX__N_16__R_6	14893	173101	11.62	4943	17	0
11	MUTEX__N_20__R_9	431910	7222550	16.72	132862	21	0
12	MUTEX__N_20__R_16	1047225	21972345	20.98	225590	21	0
13	MUTEX__N_24__R_10	4540386	86052066	18.95	329608	25	0
14	MUTEX__N_24__R_12	9740686	211067278	21.67	364622	25	0
15	MUTEX__N_24__R_13	12236830	278463166	22.76	379255	25	0
16	MUTEX__N_24__R_14	14198086	335339590	23.62	392677	25	0

TABLE II
PROPERTIES OF THE MATRICES FOR THE 'KANBAN', 'POLL' AND 'TANDEM' MODELS

#	name	n	nz	nz/n	uv	x	c
17	kanban_sm-1	160	776	4.85	73	0	776
18	kanban_sm-2	4600	32720	7.11	142	5	9845
19	kanban_sm-3	58400	504800	8.64	191	9	25910
20	kanban_sm-4	454475	4434325	9.76	200	10	245084
21	kanban_sm-5	2546432	27006448	10.61	200	11	1268920
22	poll17_sm	3342336	34537472	10.33	51	11	1730158
23	poll18_sm	7077888	76677120	10.83	45	12	2653722
24	tandem_sm-2047	8386560	37724163	4.50	14	5	0

We divide matrices into the COO and ELL parts with the HYB format using CUSP. With very small matrices format COO is faster, therefore the ELL part remains empty. In Table I, there are matrix properties describing MUTEX model; the matrices differ in size and also in the proportion of COO and ELL. Six matrices with empty ELL part were chosen, three matrices with not empty COO and ELL parts and seven matrices with empty COO part.

In Table II we describe properties of the matrices from three remaining models: 'kanban', 'poll' and 'tandem'. Among the describe matrices only one has an empty COO part, and only one an empty ELL part. The other have not empty COO and ELL parts.

III. GPU MEMORY REQUIREMENTS

Memory usage was checked by the function `cudaMemGetInfo`. This function returns the total GPU memory and free GPU card memory which gives us information about memory occupation by the application. The method of the measurement is not precise enough to enable comparison of such small matrices. It prints total GPU memory usage, not only of matrices but also all additional values. The 'basic' value of 63 MB consists of not only explicitly allocated data but also the inner CUDA variables. It is visible that minimal, constant size is about 63 MB (for $n = 13$). The memory usage depends directly on nz. With small differences in nz it may happen that the memory use will be smaller for a bigger matrix. It is difficult to find dependencies of n because the matrices have different sparsity patterns.

Tables III and IV show the memory usage in MB on one and two GPUs (respectively) for the 'kanban', 'poll' and 'tandem' models. This memory was counted by function

TABLE III
EXPERIMENTAL MEMORY USAGE (IN MB) ON ONE GPU FOR THE 'KANBAN', 'POLL' AND 'TANDEM' MODELS ($M_{exp} = \frac{\mathbf{HYB}}{\mathbf{HYBIV}}$)

#	name	**HYB**	**HYBIV**	M_{exp}
17	kanban_sm-1	64.45	65.45	0.98
18	kanban_sm-2	64.45	64.45	1.00
19	kanban_sm-3	69.70	67.82	1.03
20	kanban_sm-4	118.45	95.20	1.24
21	kanban_sm-5	402.60	254.58	1.58
22	poll17_sm	509.73	315.46	1.62
23	poll18_sm	1075.04	621.11	1.73
24	tandem_sm-2047	542.49	334.46	1.62

TABLE IV
EXPERIMENTAL MEMORY USAGE (IN MB) ON TWO GPUS FOR THE 'KANBAN', 'POLL' AND 'TANDEM' MODELS

#	name	**HYB2**			**HYBIV2**		
		gpu1	+	gpu2	gpu1	+	gpu2
17	kanban_sm-1	64.45	+	64.45	64.45	+	64.45
18	kanban_sm-2	64.45	+	65.45	64.45	+	64.45
19	kanban_sm-3	69.70	+	68.70	67.57	+	67.57
20	kanban_sm-4	103.95	+	99.32	89.82	+	86.32
21	kanban_sm-5	301.97	+	280.59	217.84	+	197.09
22	poll17_sm	375.61	+	349.98	265.46	+	239.96
23	poll18_sm	757.91	+	703.91	507.37	+	453.37
24	tandem_sm-2047	526.49	+	462.49	406.48	+	342.48

`cudaMemGetInfo` and therefore we call this memory experimental. Basing on Tables III and IV we can say that:

- The HYBIV format on larger matrices required almost twice less memory than HYB which results from experimental data.
- Memory usage per GPU was smaller in HYB2 and HYBIV2 than in HYB and HYBIV but it was higher by half because some variables were stored on each GPU.
- The number of non-zeros (nz) and the number of unique elements (uv) had the biggest influence on the memory occupation.

IV. TIME

All the processing times are reported in seconds. The time is measured with an MKL function `dsecnd`. Computations were made in double precision. Let us assume that $pt = \pi(0) = [1, 0, \ldots, 0]^T$. The choice of e_1 as the starting vector may seem too special, but it reflects the fact that we order the state space by reachability and that the Markov chain starts in the first state before evolving into other states.

Tables V and VI show run-time on CPU (CSR format, SpMV operation from MKL library), on one GPU (HYB and HYBIV formats, SpMV operation from CUSP library), on two GPUs (HYB2 and HYBIV2 formats, modified SpMV operation from CUSP library) respectively for $t = 10$, $t = 100$ and $\varepsilon = 10^{-10}$ (t and ε are parameters of the uniformization method), for matrices from 'MUTEX' model with numbers: 13, 14, 15 and 16. The bold values denote the fastest computation times and '—' denotes that the matrices could not be stored in the device memory.

Figures 1–6 show SpMV run-time (in seconds) on CPU, on one GPU, on two GPUs for the 'MUTEX' model.

On the basis of the charts of run-time it can be concluded that computation on CPU on CSR format takes the longest

TABLE V
RUN-TIME (IN SECONDS) ON CPU (CSR, SPMV FROM MKL); ON ONE GPU (HYB AND HYBIV, CUSP); ON TWO GPUS (HYB2 AND HYBIV2, CUSP) — THE 'MUTEX' MODELS FOR $t = 10$ AND $\varepsilon = 10^{-10}$

#	name	CPU	HYB	HYB2	HYBIV	HYBIV2
13	MUTEX__N_24__R_10	635.18	290.60	213.88	272.15	**207.79**
14	MUTEX__N_24__R_12	1844.90	—	552.48	757.49	**536.78**
15	MUTEX__N_24__R_13	1910.35	—	—	1048.57	**724.38**
16	MUTEX__N_24__R_14	3420.17	—	—	1314.93	**891.61**

TABLE VI
RUN-TIME (IN SECONDS) ON CPU (CSR, SPMV FROM MKL); ON ONE GPU (HYB AND HYBIV, CUSP); ON TWO GPUS (HYB2 AND HYBIV2, CUSP) — THE 'MUTEX' MODELS FOR $t = 100$ AND $\varepsilon = 10^{-10}$

#	name	CPU	HYB	HYB2	HYBIV	HYBIV2
13	MUTEX__N_24__R_10	6293.35	2923.13	2073.12	2758.9	**2001.43**
14	MUTEX__N_24__R_12	15206.1	—	5386.58	7647.88	**5212.61**
15	MUTEX__N_24__R_13	20242.1	—	—	10628.6	**7076.16**
16	MUTEX__N_24__R_14	33267.8	—	—	13396.4	**8767.05**

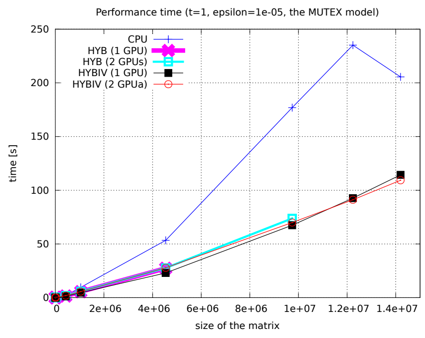

Fig. 1. Run-time (in seconds) on CPU (CSR, SpMV from MKL); on one GPU (HYB and HYBIV, CUSP); on two GPUs (HYB2 and HYBIV2, modified CUSP) — the 'MUTEX' model ($t = 1$, $\varepsilon = 10^{-5}$)

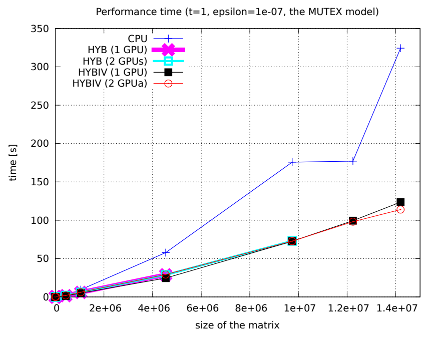

Fig. 2. Run-time (in seconds) on CPU (CSR, SpMV from MKL); on one GPU (HYB and HYBIV, CUSP); on two GPUs (HYB2 and HYBIV2, modified CUSP) — the 'MUTEX' model ($t = 1$, $\varepsilon = 10^{-7}$)

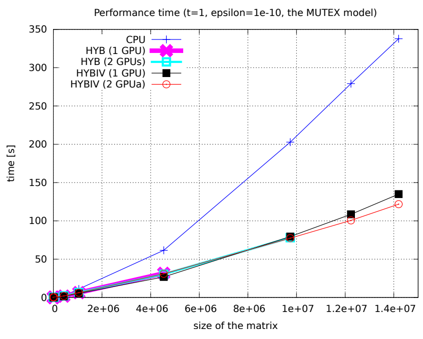

Fig. 3. Run-time (in seconds) on CPU (CSR, SpMV from MKL); on one GPU (HYB and HYBIV, CUSP); on two GPUs (HYB2 and HYBIV2, modified CUSP) — the 'MUTEX' model ($t = 1$, $\varepsilon = 10^{-10}$)

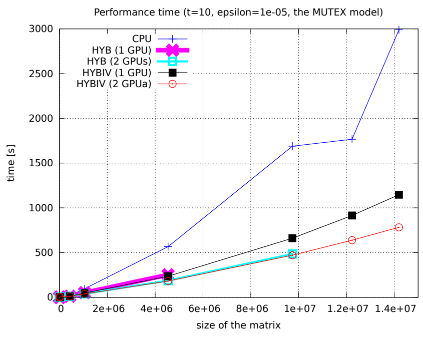

Fig. 4. Run-time (in seconds) on CPU (CSR, SpMV from MKL); on one GPU (HYB and HYBIV, CUSP); on two GPUs (HYB2 and HYBIV2, modified CUSP) — the 'MUTEX' model ($t = 10$, $\varepsilon = 10^{-5}$)

time, while GPU application speedup the run-time considerably.

The value of the variable l depends on t variable value. For 'MUTEX' model it is worth using two GPUs. Clearly, computations were done faster for proposed HYBIV format than for HYB format.

Tables VII and VIII show the time in seconds for the double precision uniformization method using the HYB and HYBIV storage formats on one GPU and two GPUs and the CSR

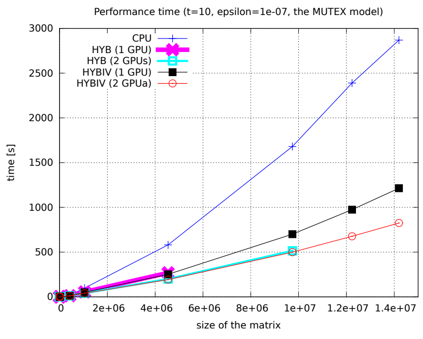

Fig. 5. Run-time (in seconds) on CPU (CSR, SpMV from MKL); on one GPU (HYB and HYBIV, CUSP); on two GPUs (HYB2 and HYBIV2, modified CUSP) — the 'MUTEX' model ($t = 10$, $\varepsilon = 10^{-7}$)

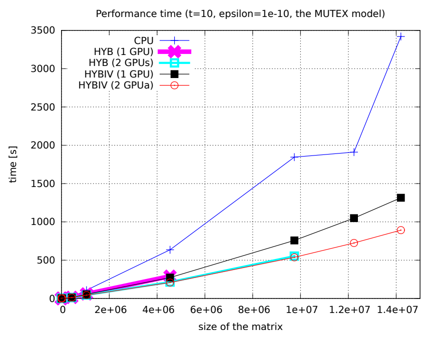

Fig. 6. Run-time (in seconds) on CPU (CSR, SpMV from MKL); on one GPU (HYB and HYBIV, CUSP); on two GPUs (HYB2 and HYBIV2, modified CUSP) — the 'MUTEX' model ($t = 10$, $\varepsilon = 10^{-10}$)

TABLE VII
SPMV RUN-TIME ON CPU (CSR, SPMV FROM MKL); ON ONE GPU (HYB AND HYBIV, CUSP); ON TWO GPUS (HYB2 AND HYBIV2, MODIFIED CUSP) — THE 'KANBAN', 'POLL' AND 'TANDEM' MODELS, $t = 1, \varepsilon = 10^{-10}$)

name	CPU	HYB	HYB2	HYBIV	HYBIV2
`kanban_sm-1`	**0.01**	0.02	0.02	**0.01**	0.02
`kanban_sm-2`	**0.01**	0.02	0.07	0.02	0.07
`kanban_sm-3`	0.03	0.04	0.12	**0.02**	0.12
`kanban_sm-4`	0.25	0.19	0.59	**0.06**	0.72
`kanban_sm-5`	1.58	1.04	3.40	**0.24**	5.98
`poll17_sm`	21.10	3.75	11.39	**2.10**	10.73
`poll18_sm`	35.70	8.37	26.26	**4.59**	24.59
`tandem_sm-2047`	1410.19	121.03	482.84	**88.31**	466.98

storage format from the MKL library on CPU for $\varepsilon = 10^{-10}$, $t = 1$ and $t = 10$ for the other three models - 'kanban', 'poll' and 'tandem'. The bold values denote the fastest computation times.

Run-times on two GPUs are slower than on one GPU for matrices which $\frac{nz}{n} < 16$ (for all matrices 'kanban', 'poll' and 'tandem' models and matrices number from 1 to 10 for 'MUTEX' model). There are too few computations to sensibly use two GPUs. For 'poll' and 'tandem' models we achieve considerable computation speedup on one GPU.

The best storage scheme — that is, the fastest and the most compact — for bigger transition rate matrices is HYBIV2 (HYBIV on 2 GPUs). The HYB storage format performs not quite efficiently in many cases. It is because the granularity (one thread per row) of the sparse matrix-vector multiplication is not fine enough for them, so the bigger the matrix, the better the utilization of the GPU. The performance of HYBIV was a little better than HYB, because in HYBIV less data is stored in slow global memory. Using two GPUs we almost doubled the performance in comparison to single GPU. For smaller matrices, HYBIV and splitting data across two GPUs were not useful.

TABLE VIII
SPMV RUN-TIME (IN SECONDS) ON CPU (CSR, SPMV FROM MKL); ON ONE GPU (HYB AND HYBIV); ON TWO GPUS (HYB2 AND HYBIV2) — THE 'KANBAN', 'POLL' AND 'TANDEM' MODELS, $t = 10, \varepsilon = 10^{-10}$)

name	CPU	HYB	HYB2	HYBIV	HYBIV2
`kanban_sm-1`	**0.01**	0.04	0.05	**0.01**	0.06
`kanban_sm-2`	**0.02**	0.05	0.26	0.05	0.26
`kanban_sm-3`	**0.06**	0.08	0.30	**0.06**	0.31
`kanban_sm-4`	1.06	0.36	1.14	**0.20**	1.24
`kanban_sm-5`	6.84	1.80	5.20	**0.85**	7.90
`poll17_sm`	197.15	25.98	73.42	**18.82**	68.49
`poll18_sm`	330.33	56.86	151.52	**40.85**	137.82
`tandem_sm-2047`	14475.40	1220.19	4873.97	**898.60**	4716.94

V. CONCLUSION

In this article, we investigated the use of a modified sparse memory format on the GPU in a practical problem, namely calculating probabilities from Markov transition matrices. Our results showed that in the case of small size matrices, we did not achieve high performance in the HYBIV format or significant memory savings. However, the proposed method reduces the memory size for storing larger matrices. In addition, the use of HYBIV does not degrade performance and the use of two GPUs allows the processing of larger matrices than one GPU. In our future work, we try to transfer codes to the CUDA version beyond 4 and use streams to optimize overall performance.

REFERENCES

[1] Bylina, B., Bylina, J., Karwacki, M.: Computational Aspects of GPU-accelerated Sparse Matrix-Vector Multiplication for Solving Markov Models. Theoretical and Applied Informatics 23, 127–145 (2011)

[2] Bylina, B., Karwacki, M., Bylina, J.: A CPU-GPU Hybrid Approach to the Uniformization Method for Solving Markovian Models — A Case Study of a Wireless Network. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2012. CCIS, vol. 291, pp. 401–410. Springer, Heidelberg (2012)

[3] Bylina, B., Karwacki, M., Bylina, J.: Multi-GPU Implementation of the Uniformization Method for Solving Markov Models. In: Proceedings of Federated Conference on Computer Science and Information Systems (FedCSIS) 2012, pp. 533–537 (2012)

[4] Bylina, J., Bylina, B., Karwacki, M.: An Efficient Representation on GPU for Transition Rate Matrices for Markov Chains; Lecture Notes in Computer Science 8384, 663–672. Springer, Heidelberg (2014)

[5] G. Ciardo and M. Tilgner.: On the use of Kronecker Operators for the Solution of Generalized Stochastic Petri Nets. ICASE Report 96-35, Institute for Computer Applications in Science and Engineering, 1996.

[6] N. J. Dingle, P. G. Harrison, W. J. Knottenbelt: *Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models*, Journal of parallel and distributed computing, 64 (2004), 908-920

[7] Fernandes, P., Plateau, B., Stewart, W.J.: Efficient Descriptor-Vector Multiplication in Stochastic Automata Networks. J. ACM 45, 381–414 (1998)

[8] M. Z. Kwiatkowska, G. Norman, D. Parker: PRISM: Probabilistic Symbolic Model Checker, Computer Performance Evaluation, Modelling Techniques and Tools 12th International Conference, TOOLS 2002, LNCS 2324, pp.200-204, Springer, 2005.

[9] Resing, J. A. C. (1993). Polling systems and multitype branching processes. Queueing Systems 13 (4): 409–426

[10] H. Hermanns, J. Meyer-Kayser and M. Siegle. Multi Terminal Binary Decision Diagrams to Represent and Analyse Continuous Time Markov Chains. In B. Plateau and W. Stewart and M. Silva (editors), Proc. 3rd International Workshop on Numerical Solution of Markov Chains (NSMC'99), pages 188-207, Prensas Universitarias de Zaragoza. 1999.

[11] R. B. Sidje: *Expokit: A software package for computing matrix exponentials*, ACM Trans. Math. Software, 24 (1998), pp. 130–156.

[12] R. B. Sidje, K. Burrage, S. MacNamara: *Inexact Uniformization Method for Computing Transient Distributions of Markov Chains.* SIAM J. Scientific Computing 29(6): 2562–2580 (2007).

[13] Intel Math Kernel Library, `http://software.intel.com/en-us/articles/intel-mkl/`