UDC 519.687.1::004

V. V. ROMANUKE
Khmelnitskiy National University

LIMITATION OF EFFECTIVENESS IN USING MATLAB GPUARRAY METHOD FOR CALCULATING PRODUCTS OF TRANSPOSE-SYMMETRICALLY SIZED MATRICES

A research of effectiveness in using MATLAB **gpuArray** *method for calculating products of transpose-symmetrically sized matrices is represented. For this, MATLAB* **gpuArray** *method is used on three types of NVIDIA® GPU. It is revealed that, independently of the size, generating matrices directly on GPU is fully inefficient. GeForce GT 610 is inefficient in itself. GeForce GTS 450 is efficient when number of lines and columns of the first matrix is greater than 200 and 50, respectively. The running time efficiency of matrix product calculation for Tesla K40c is stronger, as it comes when number of lines and columns of the first matrix is greater than 70 and 10, respectively.*

Keywords: matrix product, parallelization, effectiveness, MATLAB, **gpuArray** *method, running time efficiency.*

В. В. РОМАНЮК
Хмельницький національний університет

ОБМЕЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ MATLAB-МЕТОДУ GPUARRAY ДЛЯ ОБЧИСЛЕННЯ ДОБУТКУ МАТРИЦЬ ТРАНСПОНОВАНО-СИМЕТРИЧНОГО РОЗМІРУ

Представляється дослідження ефективності використання MATLAB-методу **gpuArray** *для обчислення добутку матриць транспоновано-симетричного розміру. Для цього використовується MATLAB-метод* **gpuArray** *на трьох типах NVIDIA® GPU. Виявляється, що генерування матриць безпосередньо на GPU є повністю неефективним незалежно від розміру. GeForce GT 610 є неефективним по суті. GeForce GTS 450 є ефективним тоді, коли число рядків і стовпців першої матриці є більшим відповідно за 200 та 50. Продуктивність часу рахунку матричного добутку для Tesla K40c більш явна, оскільки вона наступає за числа рядків і стовпців першої матриці, більшого відповідно за 70 та 10.*

Ключові слова**: добуток матриць, паралелізація, ефективність, MATLAB, метод* **gpuArray, продуктивність часу рахунку.*

Motivation of exploring the running time efficiency of nonsquare matrix product calculation

In the article [1], a research of efficient computation of square matrix product on GPU was represented. While researching, MATLAB **gpuArray** method was used on three types of NVIDIA® GPU (GeForce GTS 450, Tesla K40c, GeForce GT 610). The research exposed that MATLAB **gpuArray** method optimal use, if any, requires the matrix order be greater than 120. Generating matrices directly on GPU [2, 3] is fully inefficient for a long sequence of products. The running time is shortened when matrices are already on GPU, and the efficiency holds if matrices are generated directly on GPU just for a few times [1, p. 250].

Generally speaking, the article [1] proves the effectiveness of GPU computations becomes apparent for large sized arrays, but MATLAB **gpuArray** method may have specific optimal use when numbers of lines and columns are different. For instance, calculating on CPU, product of 10×1000 matrix and 1000×10 matrix is calculated faster than product of two 100×100 matrices. And product of 1000×10 matrix and 10×1000 matrix is calculated much slower. This specificity motivates to explore the running time efficiency (RTE) of nonsquare matrix product calculation.

Goal and items to be fulfilled

For determining RTE of using MATLAB **gpuArray** method for calculating nonsquare matrix products, let multiply $M\times N$ matrices by $N\times M$ matrices and measure the running time. Along with results of the article [1], this will allow to ascertain optimal parallelization of matrix computations [4, 5] with MATLAB Parallel Computing Toolbox using its **gpuArray** method. To get it realized, the following items are to be fulfilled:

1. Formalize the problem in mathematical notation.
2. Define the range of numbers of lines and columns in transpose-symmetrically sized matrices (TSSM).
3. Run the MATLAB code for multiplying TSSM increasing progressively numbers of their lines and columns. Make it for an appropriate number of cycles to ensure stable statistical estimation of the running time.
4. Both for CPU and GPU, estimate time for initialization of the matrix elements depending on the size (i. e. numbers M and N). Similarly, estimate time for TSSM product depending on the size.
5. For each type of the applied GPU (GeForce GTS 450, Tesla K40c, GeForce GT 610), find subranges of numbers M and N (and, probably, their combination), where the GPU running time is shorter than the CPU running time.

Formalization of RTE problem

The running time consists of the matrices' initialization period (MIP) and their product calculation period (PCP). Matrices can be initialized by a way among those three ones:

1. CPU matrix initialization, without transferring to GPU (CPUMI). This way presupposes product calculation just on CPU. The CPU running time is

$$t(M, N) = \theta(M, N) + p(M, N) \quad (1)$$

by MIP $\theta(M, N)$ and PCP $p(M, N)$ on CPU.

2. CPU matrix initialization, with subsequent transferring to GPU (CPUMI-GPU). This way presupposes product calculation on GPU. The GPU running time is

$$t_{\mathrm{GPU}}(M, N) = \theta_{\mathrm{GPU}}(M, N) + p_{\mathrm{GPU}}(M, N) \qquad (2)$$

by MIP $\theta_{\mathrm{GPU}}(M, N)$ for CPUMI-GPU and PCP $p_{\mathrm{GPU}}(M, N)$ on GPU.

3. Direct GPU matrix initialization (DGPUMI) [1, p. 244]. The matrix product is calculated on GPU, and here the GPU running time is

$$t^{*}_{\mathrm{GPU}}(M, N) = \theta^{*}_{\mathrm{GPU}}(M, N) + p^{*}_{\mathrm{GPU}}(M, N) \qquad (3)$$

by MIP $\theta^{*}_{\mathrm{GPU}}(M, N)$ for DGPUMI and PCP $p^{*}_{\mathrm{GPU}}(M, N)$ on GPU after DGPUMI.

Let matrices $\mathbf{A} = (a_{ij})_{M\times N}$ and $\mathbf{B} = (b_{lk})_{N\times M}$ to be multiplied have entries which are values of standard normal variates:

$$\mathbf{A} \in \mathrm{N}\,(0, 1, M\times N) \text{ and } \mathbf{B} \in \mathrm{N}\,(0, 1, N\times M) \text{ by } M \in \mathbb{N}\setminus\{1\} \text{ and } N \in \mathbb{N}\setminus\{1\} \qquad (4)$$

for the infinite set $\mathrm{N}\,(0, 1, L\times Q)$ of $L\times Q$-matrices in which every entry is a value drawn from the standard normal distribution. By CPUMI, the product of matrices (4) is the matrix

$$\mathbf{C} = (c_{ik})_{M\times M} = \mathbf{A}\cdot\mathbf{B} = \left(\sum_{j=1}^{N} a_{ij} b_{jk}\right)_{M\times M}. \qquad (5)$$

The period of assignment (4) is MIP $\theta(M, N)$ in the CPU running time (1). By CPUMI-GPU, matrices (4) are preliminarily copied to a GPU device. The copier is a mapping C taking a matrix $\mathbf{Z}$ on CPU and returning the matrix $\mathbf{Z}_{\mathrm{GPU}}$ on GPU, where $\mathbf{Z}_{\mathrm{GPU}} = \mathbf{Z}$ [1], so the CPUMI-GPU product of matrices (4) is the matrix

$$\mathbf{C}_{\mathrm{GPU}} = (c_{ik})_{M\times M} = \mathbf{A}_{\mathrm{GPU}}\cdot\mathbf{B}_{\mathrm{GPU}} = \left(\sum_{j=1}^{N} a_{ij} b_{jk}\right)_{M\times M}. \qquad (6)$$

MIP $\theta_{\mathrm{GPU}}(M, N)$ in the GPU running time (2) consists of the period of assignment (4) and period of transferring

$$\mathbf{A}_{\mathrm{GPU}} = C(\mathbf{A}) \text{ and } \mathbf{B}_{\mathrm{GPU}} = C(\mathbf{B}). \qquad (7)$$

By DGPUMI, the matrix (6) is calculated just after matrices are generated directly on GPU:

$$\mathbf{A}_{\mathrm{GPU}} \in \mathrm{N}\,(0, 1, M\times N) \text{ and } \mathbf{B}_{\mathrm{GPU}} \in \mathrm{N}\,(0, 1, N\times M) \text{ by } M \in \mathbb{N}\setminus\{1\} \text{ and } N \in \mathbb{N}\setminus\{1\}. \qquad (8)$$

Then MIP $\theta^{*}_{\mathrm{GPU}}(M, N)$ in the GPU running time (3) is just the period of assignment (8). Note that the product (6) is not returned back (in MATLAB notation, not gathered) to CPU. And, theoretically, PCP $p_{\mathrm{GPU}}(M, N)$ and $p^{*}_{\mathrm{GPU}}(M, N)$ are expected to be the same. However, some differences occur [1, Figure 7 on p. 247, Figure 10 on p. 248, Figure 13 on p. 249].

For some maximum numbers $M_{\max}$ and $N_{\max}$, the goal is to find those subranges

$$R_M \times R_N \subset \left\{\overline{2, M_{\max}}\right\} \times \left\{\overline{2, N_{\max}}\right\} \qquad (9)$$

at which the following inequalities are true:

$$t_{\mathrm{GPU}}(M, N) < t(M, N),\; t^{*}_{\mathrm{GPU}}(M, N) < t(M, N),\; p_{\mathrm{GPU}}(M, N) < p(M, N),\; p^{*}_{\mathrm{GPU}}(M, N) < p(M, N)$$
$$\text{by } \{M, N\} \in R_M \times R_N \subset \left\{\overline{2, M_{\max}}\right\} \times \left\{\overline{2, N_{\max}}\right\}. \qquad (10)$$

The relationship between $t_{\mathrm{GPU}}(M, N)$ and $t^{*}_{\mathrm{GPU}}(M, N)$ along with $p^{*}_{\mathrm{GPU}}(M, N)$ and $p_{\mathrm{GPU}}(M, N)$ ought to be ascertained. Thus we have three objects to be evaluated and compared pairwise [1].

Range of numbers of lines and columns in TSSM

To complete the range of numbers of lines and columns in TSSM, appoint the maximum numbers $M_{\max}$ and $N_{\max}$. Obviously, they must be identical. The abscissa and ordinate axes in Figures 6 — 14 of the article [1] allow to put $M_{\max} = N_{\max} = 500$. Excepting MIP where singly two matrices on GeForce GTS 450 are preassigned [1, Figure 8 on p. 247], the range may be shortened by to $M_{\max} = N_{\max} = 400$.

Clocking the running times and estimations

For clocking the running times, we use the mentioned three types of NVIDIA® GPU [1, Figures 1 — 3 on p. 245]. The conclusion about MATLAB **gpuArray** method not suitable for any square matrix computations on GeForce GT 610 [1, p. 249] does not restrict us to try this GPU for TSSM. Only DGPUMI will be executed single time because it takes badly increasing MIP by the cycled DGPUMI both for GeForce GTS 450 and Tesla K40c [1, Figure 4 on p. 245, Figure 5 on p. 246]. However, the inequality

$$\theta_{\mathrm{GPU}}(M, N) > \theta(M, N) \quad (11)$$

observed in [1, Figure 6 on p. 246, Figure 8 on p. 247, Figure 9 on p. 247, Figure 11 on p. 248, Figure 12 on p. 248, Figure 14 on p. 249] should be nonetheless checked for TSSM.

An appropriate number of cycles to ensure stable statistical estimation of the running times shouldn't be necessarily equal to 1000. For accelerating estimation procedures, we take 100 cycles. This is enough to make qualitative conclusions.

After the MATLAB code for multiplying TSSM has been run and executed, we visualize 3D graphs of the meshed surfaces in the inequalities (10) and (11) and $\theta^{*}_{\mathrm{GPU}}(M, N)$. Note that CPU are different for GeForce GTS 450 and Tesla K40c (GeForce GT 610 is with the same CPU as Tesla K40c), so graphs of the CPU running time (1) and its MIP $\theta(M, N)$ and PCP $p(M, N)$ will be re-visualized afresh.

When matrices are preassigned during 100 cycles (100-preassignment), GeForce GT 610 is fully inefficient (compare Figures 1 — 3 to Figures 4 — 6). Deplorably, DGPUMI by 100-preassignment for this GPU is impracticable, because it took more than two days to plot the meshes on just $\left\{\overline{2, 23}\right\} \times \left\{\overline{2, 400}\right\}$. Firstly, it took about 16 minutes to plot the line at $M = 3$ after $M = 2$. Further, the time was increasing progressively (half an hour for $M = 4$, 44 minutes for $M = 5$, ...). Finally, the line at $M = 23$ after $M = 22$ was plotted taken 4 hours and 22 minutes.

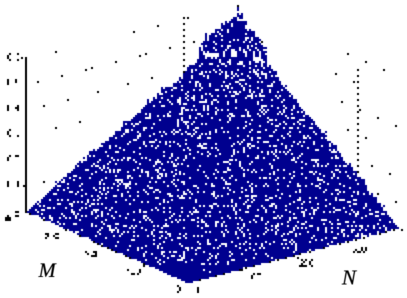

Fig. 1. MIP in the CPU running time (1) before GeForce GT 610 is enabled by 100-preassignment

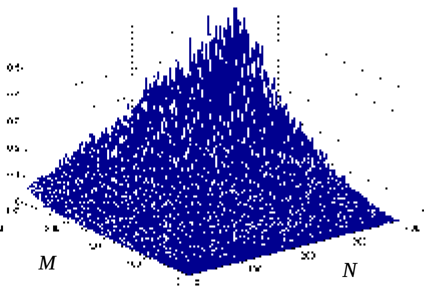

Fig. 2. PCP in the CPU running time (1) before GeForce GT 610 is enabled by 100-preassignment

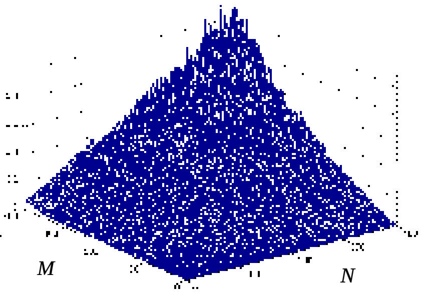

Fig. 3. The CPU running time (1) before GeForce GT 610 is enabled by 100-preassignment

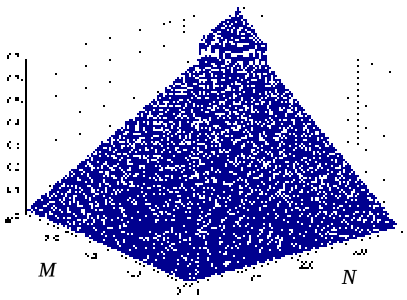

Fig. 4. MIP for CPUMI-GPU in the GPU running time (2) for GeForce GT 610 by 100-preassignment

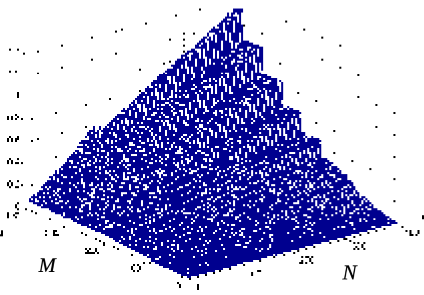

Fig. 5. PCP on GeForce GT 610 in the GPU running time (2) by 100-preassignment

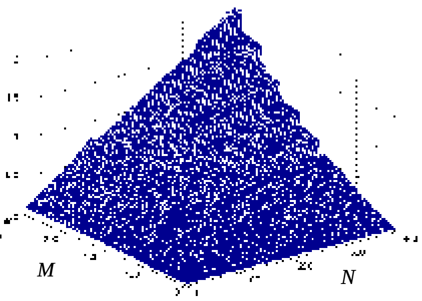

Fig. 6. The GPU running time (2) for GeForce GT 610 by 100-preassignment

When GeForce GTS 450 is on, MIP is shorter for CPU by $M > 150$, $N > 150$ (compare Figures 7 — 9 to Figures 10 — 12). PCP and the GPU running time (2) are surely shorter by $M > 250$ independently of N. DGPUMI by 100-preassignment for this GPU is impracticable also: it took 27, 45, 63, 80, 97, 115, 132, 150, 167 minutes to plot the meshes on $\left\{\overline{2, 11}\right\} \times \left\{\overline{2, 400}\right\}$ going by $M = 2$ through $M = 11$.

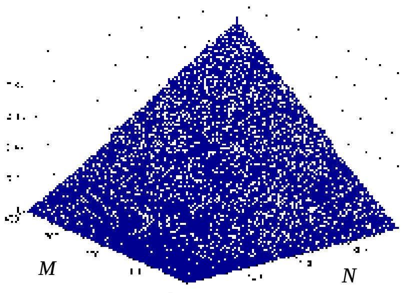

Fig. 7. MIP in the CPU running time (1) before GeForce GTS 450 is enabled by 100-preassignment

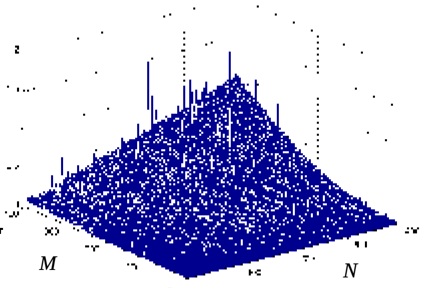

Fig. 8. PCP in the CPU running time (1) before GeForce GTS 450 is enabled by 100-preassignment

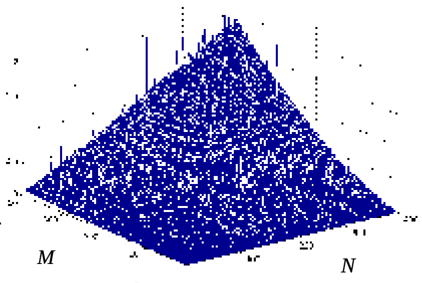

Fig. 9. The CPU running time (1) before GeForce GTS 450 is enabled by 100-preassignment

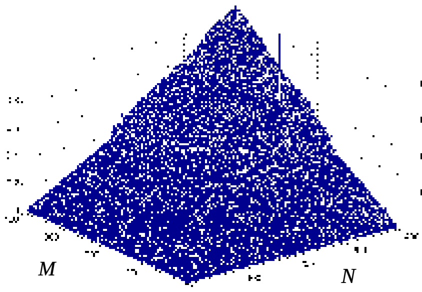

Fig. 10. MIP for CPUMI-GPU in the GPU running time (2) for GeForce GTS 450 by 100-preassignment

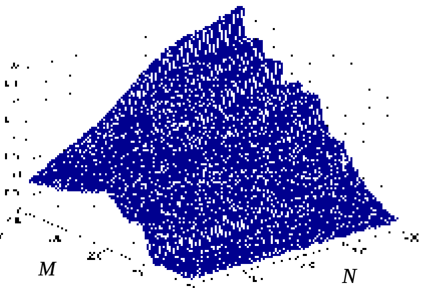

Fig. 11. PCP on GeForce GTS 450 in the GPU running time (2) by 100-preassignment

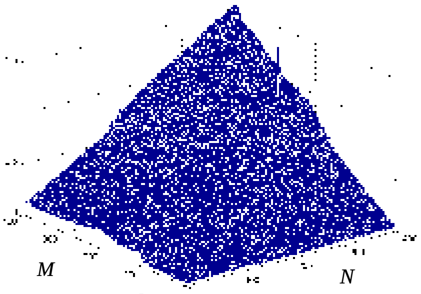

Fig. 12. The GPU running time (2) for GeForce GTS 450 by 100-preassignment

Amazingly enough, the meshes for Tesla K40c by 100-preassignment have strange region over $\left\{\overline{2, 192}\right\}\times\left\{\overline{2, 400}\right\}$ (compare Figures 13 — 15 to Figures 16 — 18). The experiment was canceled after $M=192$. Nevertheless, MIP on $\left\{\overline{193, 400}\right\}\times\left\{\overline{2, 400}\right\}$ are very similar for CPU and Tesla K40c. Resuming (restarting) the experiment since $M=193$, the GPU appears better than CPU (see the abrupt drop in Figures 17 and 18):

$$p_{\mathrm{GPU}}(M, N) < p(M, N) \text{ and } t_{\mathrm{GPU}}(M, N) < t(M, N) \text{ by } \{M, N\} \in \left\{\overline{193, 400}\right\}\times\left\{\overline{50, 400}\right\}. \quad (12)$$

Note, however, that such an effect of hang makes Tesla K40c not so reliable as it might be expected. DGPUMI by 100-preassignment for Tesla K40c is impracticable: it took 17, 31, 44, 56 minutes progressively to plot the meshes on $\left\{\overline{2, 6}\right\}\times\left\{\overline{2, 400}\right\}$ going by $M=2$ through $M=6$.

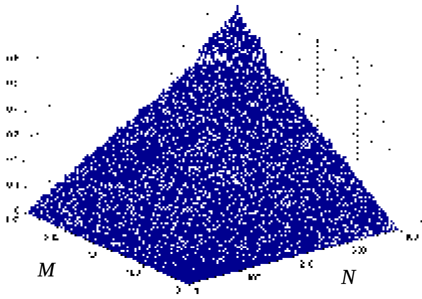

Fig. 13. MIP in the CPU running time (1) before Tesla K40c is enabled by 100-preassignment

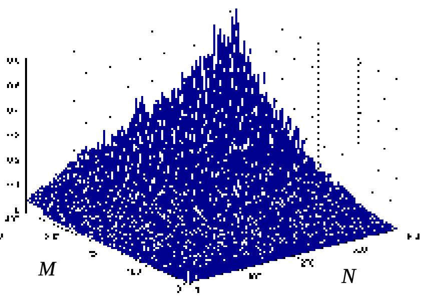

Fig. 14. PCP in the CPU running time (1) before Tesla K40c is enabled by 100-preassignment

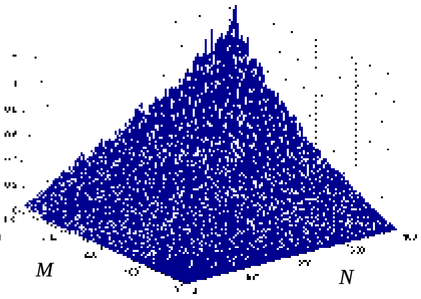

Fig. 15. The CPU running time (1) before Tesla K40c is enabled by 100-preassignment

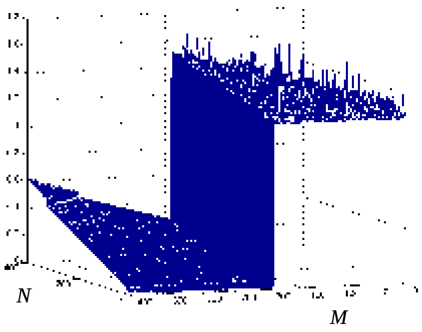

Fig. 16. MIP for CPUMI-GPU in the GPU running time (2) for Tesla K40c by 100-preassignment; the unexpected strange region over $\left\{\overline{2, 192}\right\}\times\left\{\overline{2, 400}\right\}$ is occasional, but it looks like such poor starts of Tesla K40c can be systematic

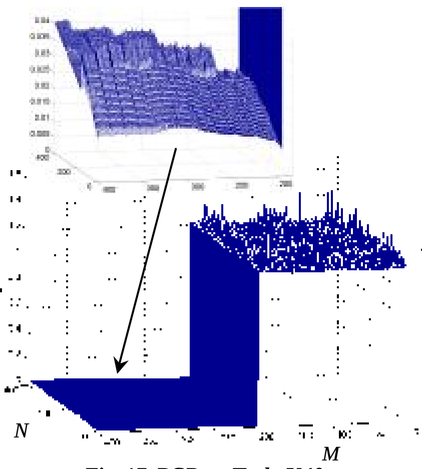

Fig. 17. PCP on Tesla K40c in the GPU running time (2) by 100-preassignment

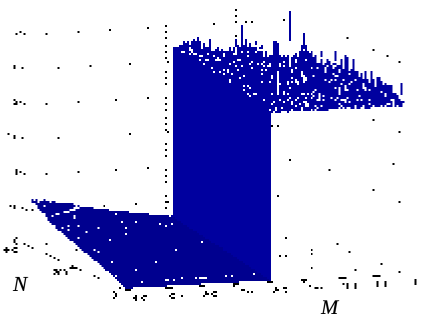

Fig. 18. The GPU running time (2) for Tesla K40c by 100-preassignment; the unexpected strange region over $\left\{\overline{2, 192}\right\}\times\left\{\overline{2, 400}\right\}$ is aftermath of the worst MIP and PCP, whose inapplicability makes Tesla K40c not so reliable

When matrices are preassigned singly (1-preassignment), GeForce GT 610 is fully inefficient again (compare Figures 19 — 21 to Figure 22, where PCP on GeForce GT 610 in the GPU running time (2) by 1-preassignment is very similar to the mesh in Figure 22, and MIP for CPUMI-GPU is close to MIP in Figure 19). DGPUMI is senseless (Figure 23).

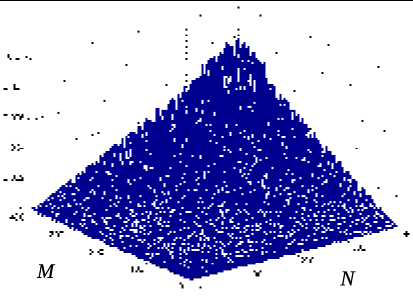

Fig. 19. MIP in the CPU running time (1) before GeForce GT 610 is enabled by 1-preassignment

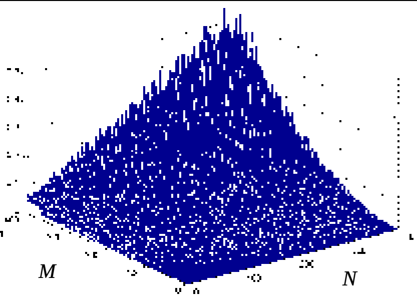

Fig. 20. PCP in the CPU running time (1) before GeForce GT 610 is enabled by 1-preassignment

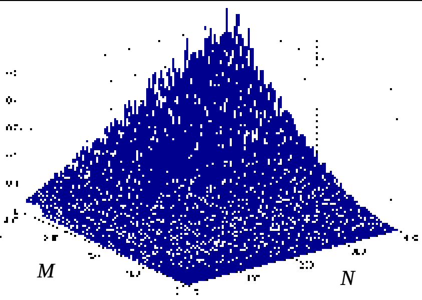

Fig. 21. The CPU running time (1) before GeForce GT 610 is enabled by 1-preassignment

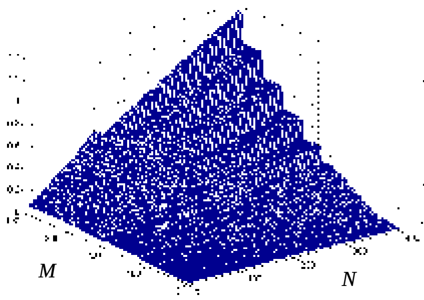

Fig. 22. The GPU running time (2) for GeForce GT 610 by 1-preassignment

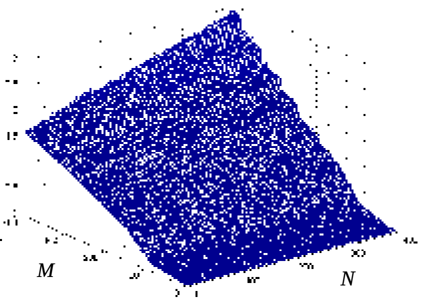

Fig. 23. The GPU running time (3) for GeForce GT 610 by 1-preassignment

GeForce GTS 450 looks pretty efficient by 1-preassignment (Figure 24 and Figure 26) by $M > 200$ and $N > 50$. MIP for DGPUMI is problematic for 2×2-matrices lasting up to, particularly, 0.736 second. Such a long MIP is likely taken for the GPU commutation (porting). Owing to generally short MIP, PCP and the corresponding running times (1) — (3) are very similar. By 1-preassignment, DGPUMI is fully ineffective (Figure 26). Curiously enough, the GPU running time (3) for GeForce GTS 450 by 1-preassignment is independent of N, increasing quasi-linearly as M increases. Below, the same disappointing DGPUMI results will be revealed for Tesla K40c.

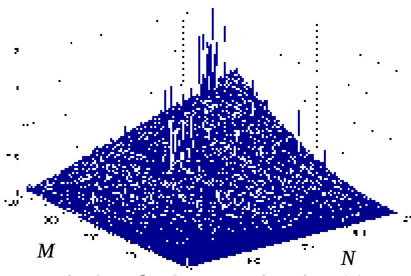

Fig. 24. The CPU running time (1) before GeForce GTS 450 is enabled by 1-preassignment

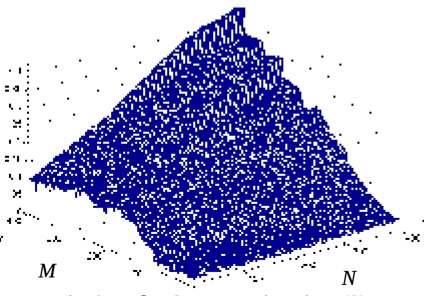

Fig. 25. The GPU running time (2) for GeForce GTS 450 by 1-preassignment

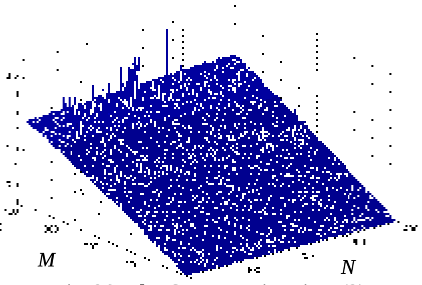

Fig. 26. The GPU running time (3) for GeForce GTS 450 by 1-preassignment

The real RTE is observed when by 1-preassignment Tesla K40c is enabled (compare Figures 27 — 29 to Figures 30 — 32), excepting DGPUMI (Figure 33 and Figure 34, where a strange region over $\{\overline{2,\ 79}\} \times \{\overline{2,\ 400}\}$ befell again forcing to restart the experiment from $M = 80$). Here

$$p_{\mathrm{GPU}}(M, N) < p(M, N) \text{ and } t_{\mathrm{GPU}}(M, N) < t(M, N) \text{ by } \{M, N\} \in \{\overline{70,\ 400}\} \times \{\overline{10,\ 400}\} \qquad (13)$$

is true almost surely. MIP for DGPUMI is again problematic for 2×2-matrices lasting up to, particularly, 2.231 seconds taken for the GPU commutation (porting).

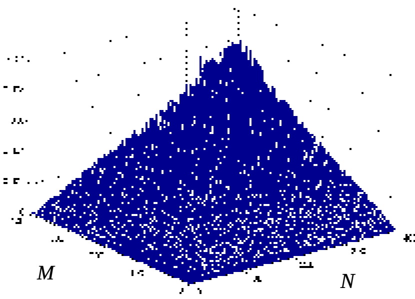

Fig. 27. MIP in the CPU running time (1) before Tesla K40c is enabled by 1-preassignment

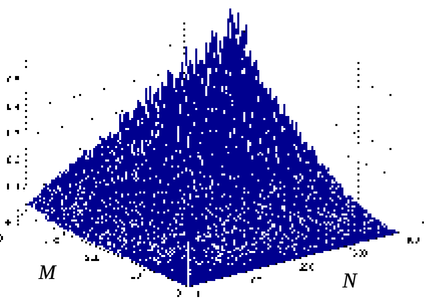

Fig. 28. PCP in the CPU running time (1) before Tesla K40c is enabled by 1-preassignment

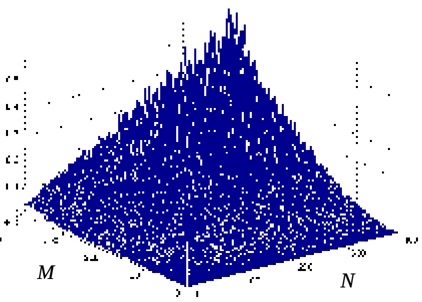

Fig. 29. The CPU running time (1) before Tesla K40c is enabled by 1-preassignment

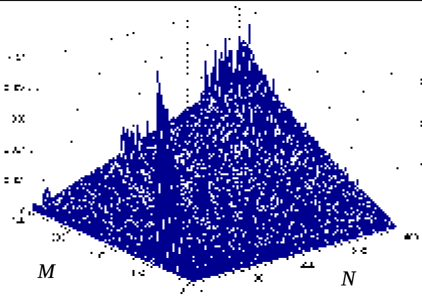

Fig. 30. MIP for CPUMI-GPU in the GPU running time (2) for Tesla K40c by 1-preassignment

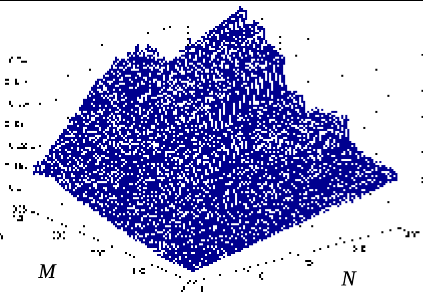

Fig. 31. PCP on Tesla K40c in the GPU running time (2) by 1-preassignment

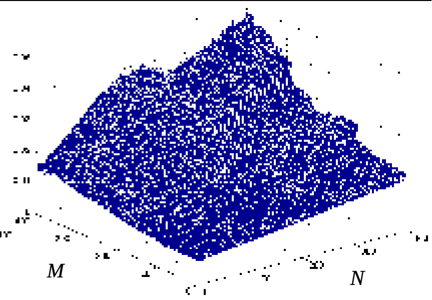

Fig. 32. The GPU running time (2) for Tesla K40c by 1-preassignment

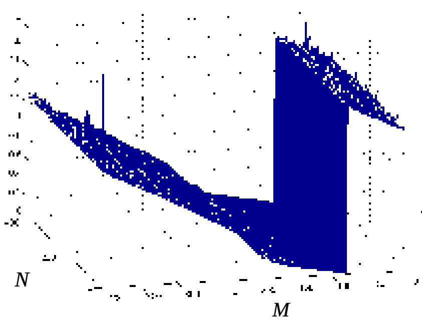

Fig. 33. PCP on Tesla K40c after DGPUMI in the GPU running time (3) by 1-preassignment

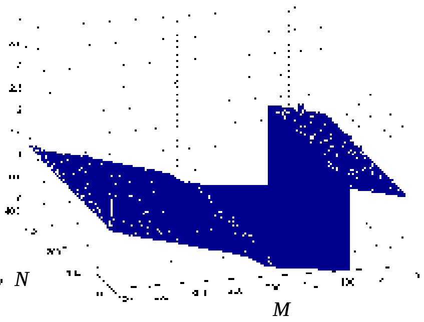

Fig. 34. The GPU running time (3) for Tesla K40c by 1-preassignment

Review of results and conclusion

The research results in Figures 1 — 34 fairly expose the limitation of MATLAB **gpuArray** method effectiveness. Generally, using MATLAB **gpuArray** method for calculating products of TSSM is effective when number of lines of the first matrix is about a hundred and greater. Namely, GeForce GTS 450 is efficient when number of lines and columns of the first matrix is greater than 200 and 50, respectively. Tesla K40c is efficient when number of lines and columns of the first matrix is greater than 70 and 10, respectively. Unlike GeForce GTS 450 and Tesla K40c, GeForce GT 610 is out of competitiveness [1]. However, now such a disappointing conclusion is not only for square matrix product, but also for nonsquare matrix product calculation. Another result repeating a finding in [1] is that, independently of the size, generating matrices directly on GPU is fully inefficient. Obviously, this is not just a MATLAB **gpuArray** method problem. The reason is those hangs of Tesla K40c for CPUMI-GPU (Figures 16 — 18). No hangs of GeForce GTS 450 were registered, although there are some pleated regions in Figure 11 and Figure 25. Consequently, optimal parallelization [6, 7] of matrix computations with MATLAB Parallel Computing Toolbox using its **gpuArray** method comes both with large sized matrices (at least, if nonsquare, starting at about 70×10), and accurate configuration of GPU, the operating system, CPU.

References

1. Romanuke V. V. MATLAB gpuArray method optimal use for square matrix product // Herald of Khmelnytskyi national university. Technical sciences. — 2015. — № 3. — P. 243 — 250.

2. Silber-Chaussumier F. Generating data transfers for distributed GPU parallel programs / F. Silber-Chaussumier, A. Muller, R. Habel // Journal of Parallel and Distributed Computing. — 2013. — Volume 73, Issue 12. — P. 1649 — 1660.

3. Zhang L. High accuracy digital image correlation powered by GPU-based parallel computing / L. Zhang, T. Wang, Z. Jiang, Q. Kemao, Y. Liu, Z. Liu, L. Tang, S. Dong // Optics and Lasers in Engineering. — 2015. — Volume 69. — P. 7 — 12.

4. Kshemkalyani A. D. Distributed Computing Principles, Algorithms, and Systems / A. D. Kshemkalyani, M. Singhal. — Cambridge University Press, 2008. — 754 p.

5. Trobec R. Parallel Computing. Numerics, Applications, and Trends / R. Trobec, M. Vajteršic, P. Zinterhof (Eds.). — Springer, 2009. — 530 p.

6. Coppersmith D. Matrix multiplication via arithmetic progressions / D. Coppersmith, S. Winograd // Journal of Symbolic Computation. — 1990. — Volume 9, Issue 3. — P. 251 — 280.

7. Chou C.-C. Parallelizing Strassen's method for matrix multiplication on distributed-memory MIMD architectures / C.-C. Chou, Y.-F. Deng, G. Li, Y. Wang // Computers & Mathematics with Applications. — 1995. — Volume 30, Issue 2. — P. 49 — 69.

Рецензія/Peer review : 26.9.2015 р. Надрукована/Printed : 2.11.2015 р.
Стаття рецензована редакційною колегією