

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ  
УНИВЕРСИТЕТ**

**Факультет информационных технологий**

**Кафедра параллельных вычислений**

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**“Умножение матрицы на матрицу в MPI 2D решетка”**

**студента 2 курса, 20212 группы**

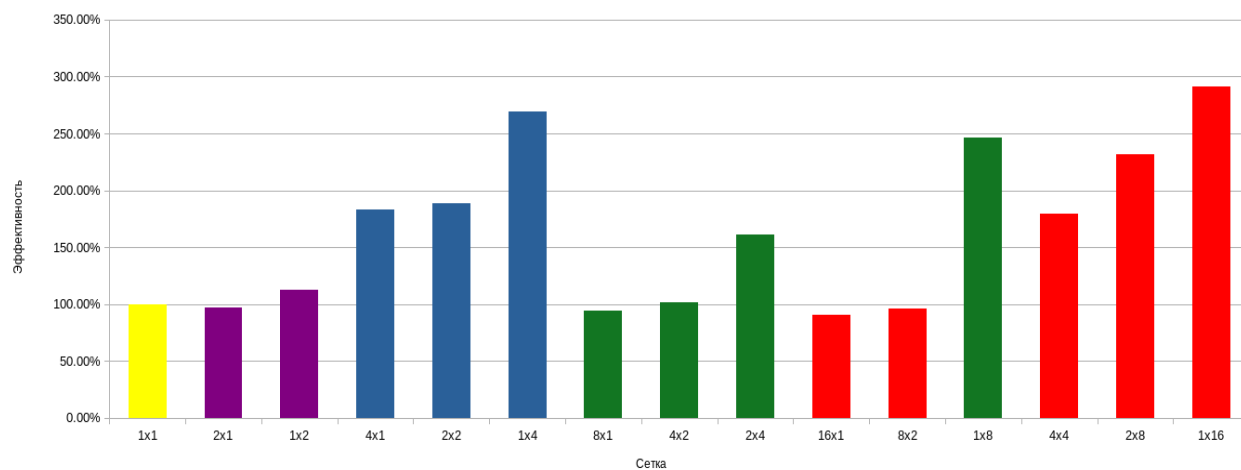
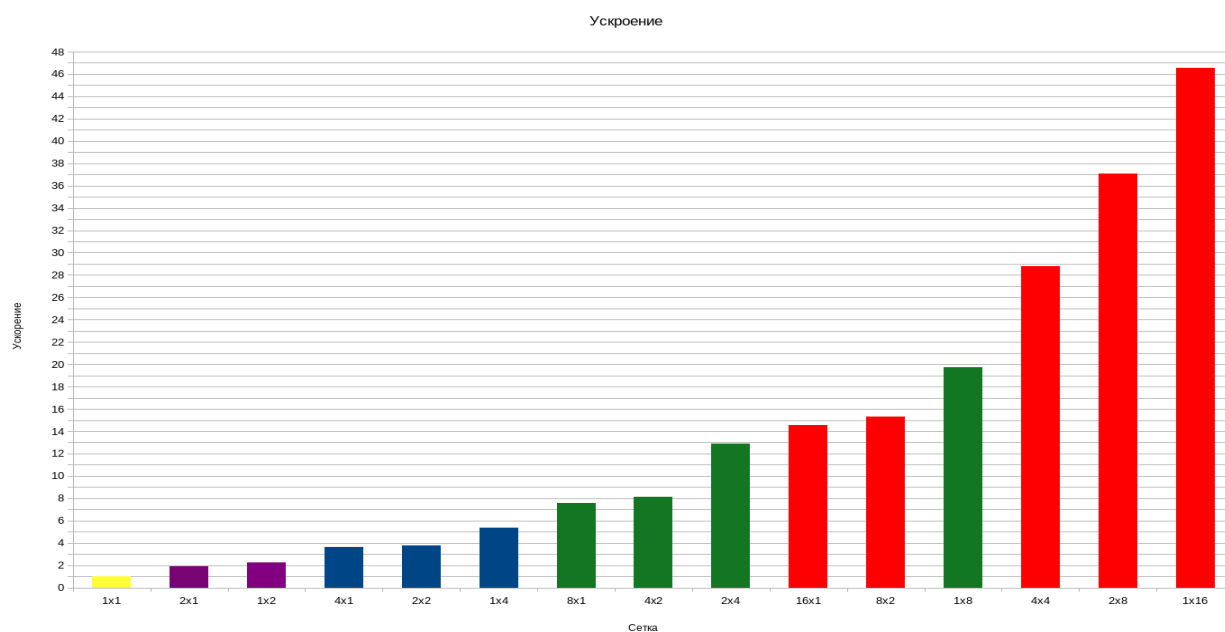
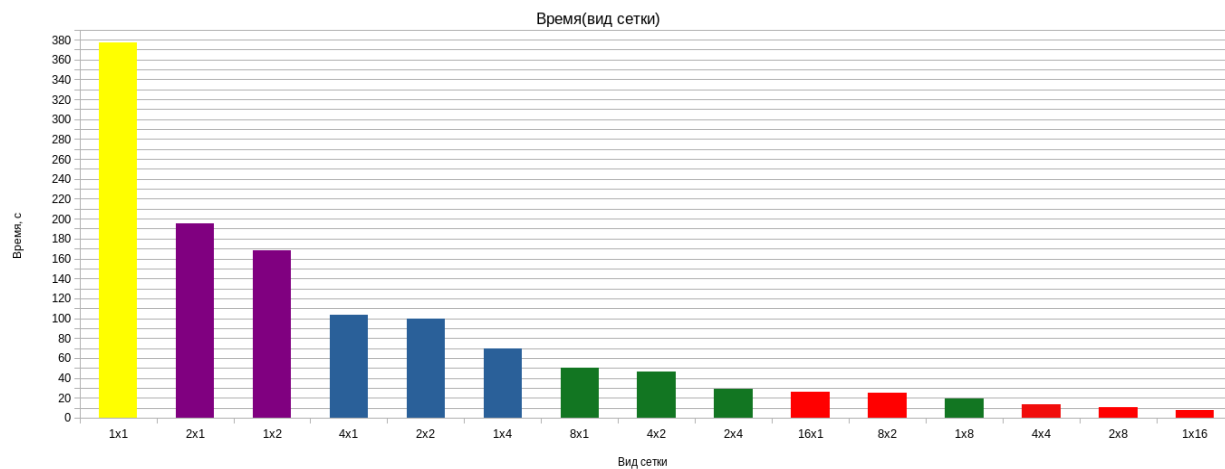
**Курбатова Максима Андреевича**

**Направление 09.03.01 – «Информатика и вычислительная техника»**

**Преподаватель: Кудинов А.Ю.**

**Новосибирск 2022**

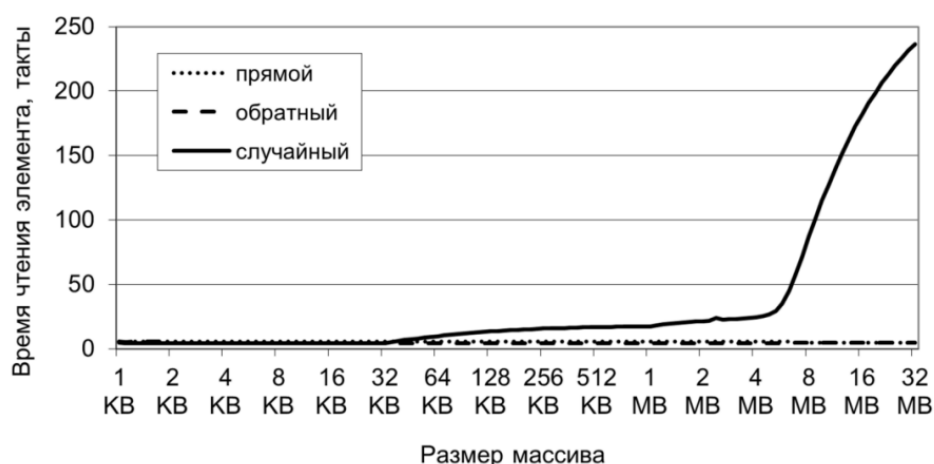
Замеры были сделаны при размере матрицы 2048x2048



## Анализ полученных результатов

Сразу бросается в глаза эффективность более 100%. Однако это можно легко объяснить. Все дело в иерархии памяти.

**Примерный** график времени доступа к памяти матрицы:



Дело в том, что когда мы обходим матрицу B, то обход ее по столбцам нарушает парадигму локальности данных. При каждом обращении к новому элементу матрицы B будет происходить кеш-промах.

1 x 1	1 x 2	1x4	1x8	1 x 16
2048x2048	2048x1024	2048x512	2048x256	2048x128
32 Mb	16 Mb	8 Mb	4 Mb	2 Mb

Теперь посмотрим уровни кеша хоста на кластере

Level 1 cache size ?

4 x 32 KB 4-way set associative instruction caches  
4 x 32 KB 8-way set associative data caches

Level 2 cache size ?

4 x 256 KB 8-way set associative caches

Level 3 cache size

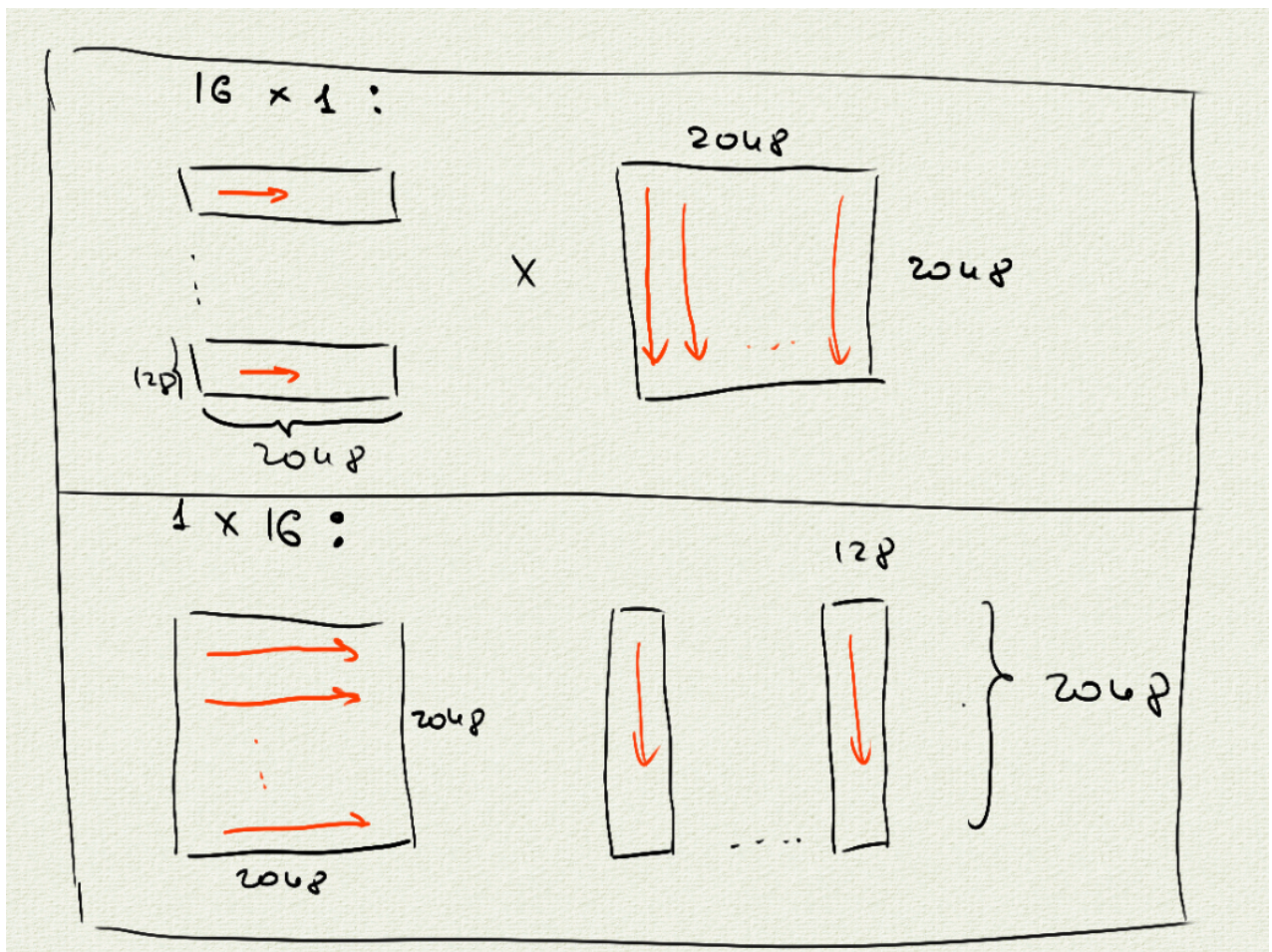
8 MB 16-way set associative shared cache













Становится понятно, что при одном процессе количество вычислений тоже самое, что и на 16, однако хвост нашей матрицы будет храниться в кеш-памяти третьего уровня и доступ к таким элементам будет 30-50 тактов. А если мы поделим матрицу между процессами, то уже кусочек матрицы меньшего размера будет лежать в кеш-памяти 3 уровня. Прибавим еще к этому, что на 16 процессах у нас половина ядер на другом хосте, у которого свой кеш, это дает еще более быстрый доступ к памяти, тк кеш 3 уровня разделяемый.

Еще один момент. Почему  $1 \times 16$  работает гораздо быстрее, чем  $16 \times 1$ ?

При  $16 \times 1$  матрица  $B$  не делится и мы вынуждены всю ее загрузить в кеш, сильно залезая на 3 уровень.

При  $16 \times 1$  такое происходит с матрицей  $A$ , но тк мы ее обходим последовательно, то аппаратная предвыборка сделает свою работу и доступ ко всем элементам этой матрицы будет примерно одинаков. А матрицу  $B$  мы разрезали, то не так сильно потеряем во времени.

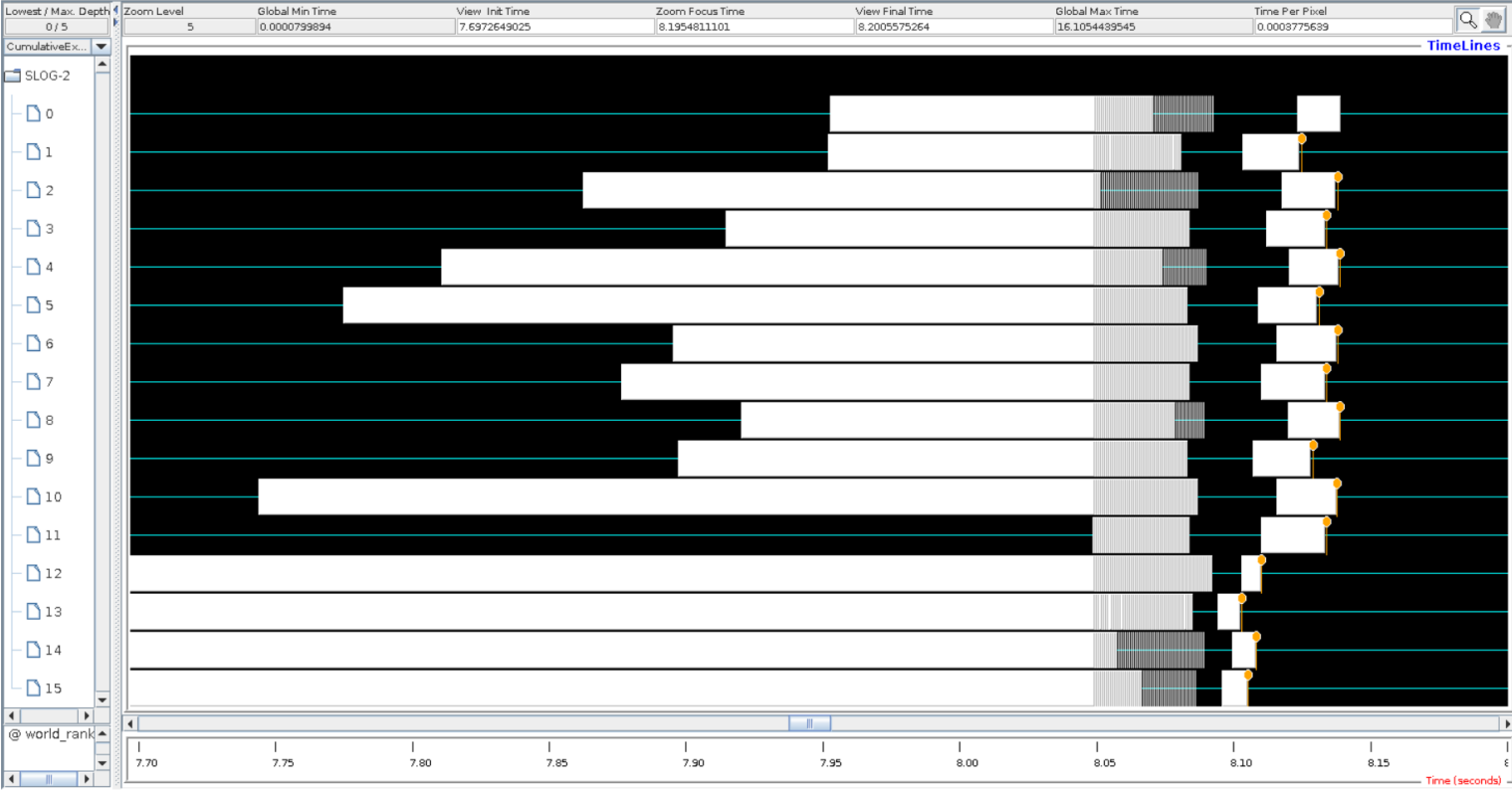
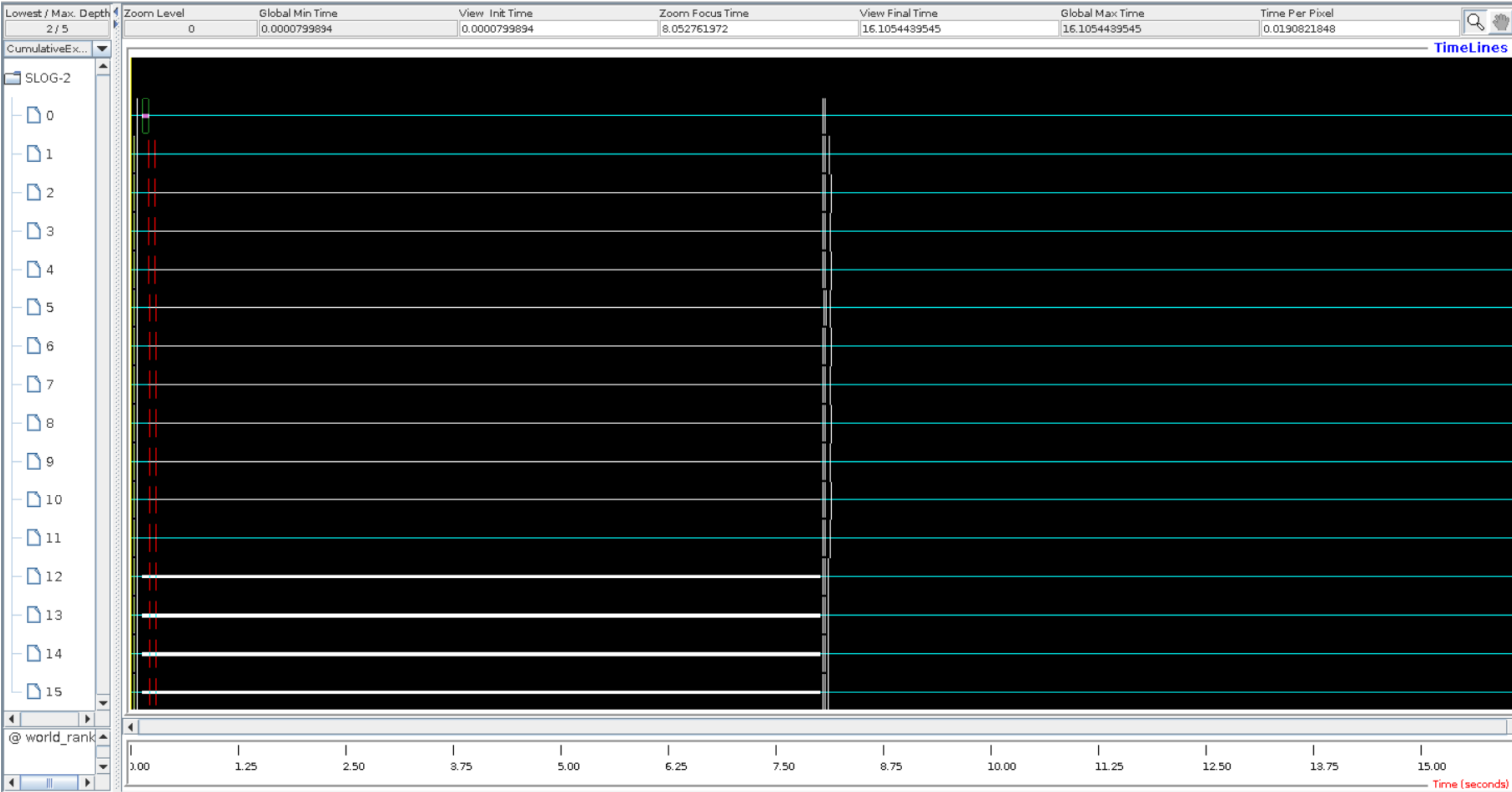


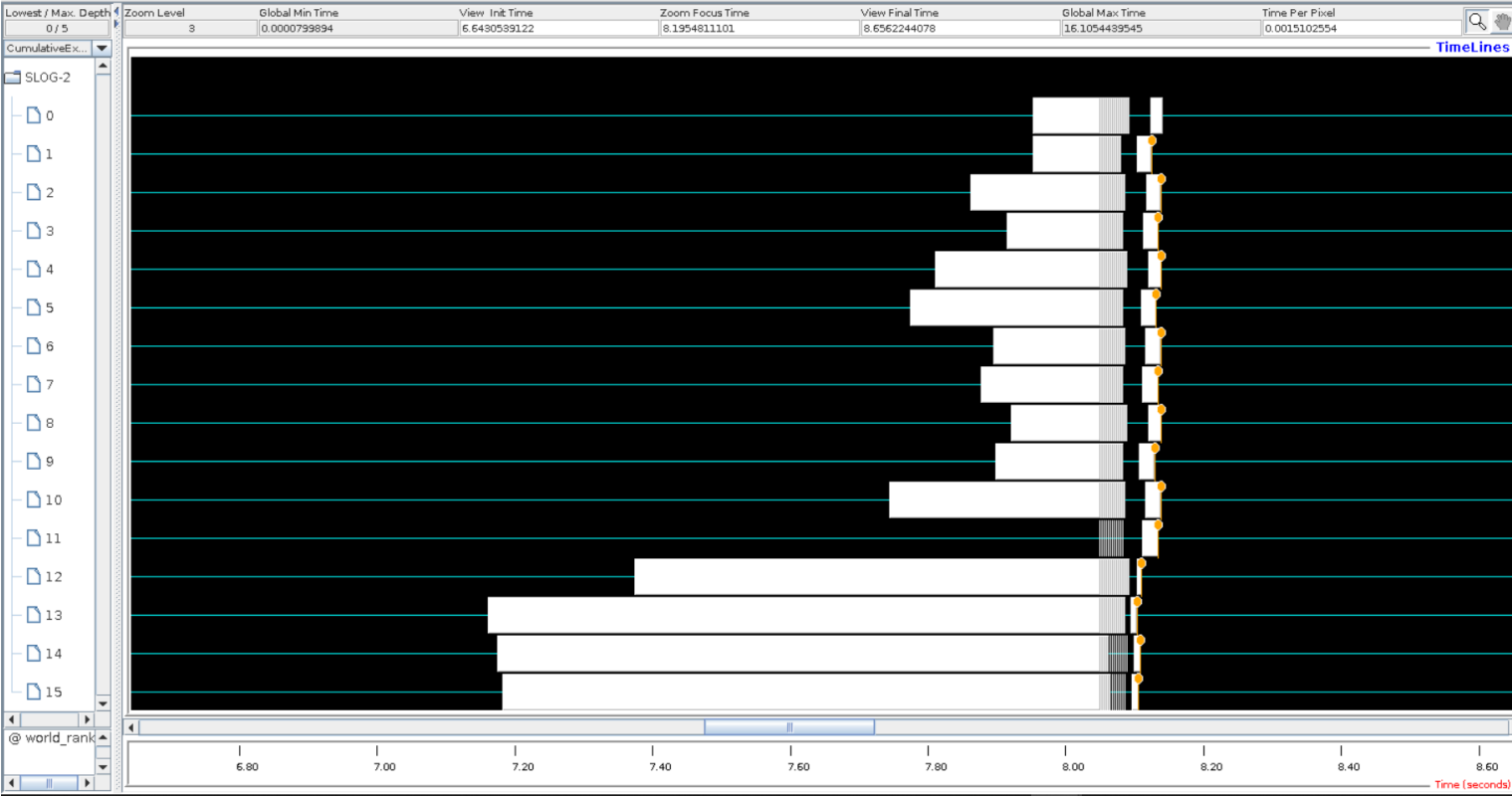
Topo	Name ▼	V ▼	S ▼
	Preview_State	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPI_Bcast	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPI_Cart_coords	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPI_Cart_create	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPI_Cart_sub	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPI_Comm_rank	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPI_Comm_size	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPI_Gather	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPI_Scatter	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	Preview_Event	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPE_Comm_finalize	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	MPE_Comm_init	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

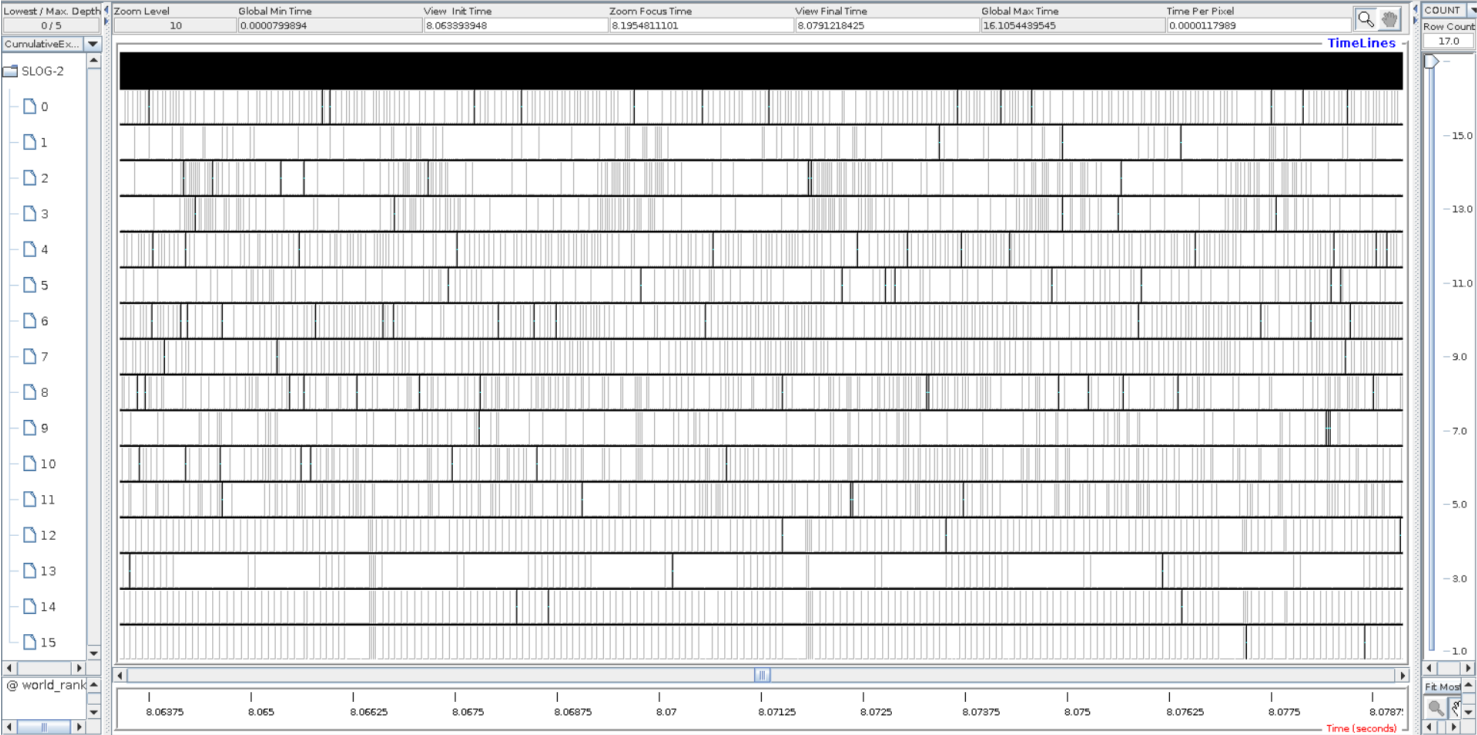
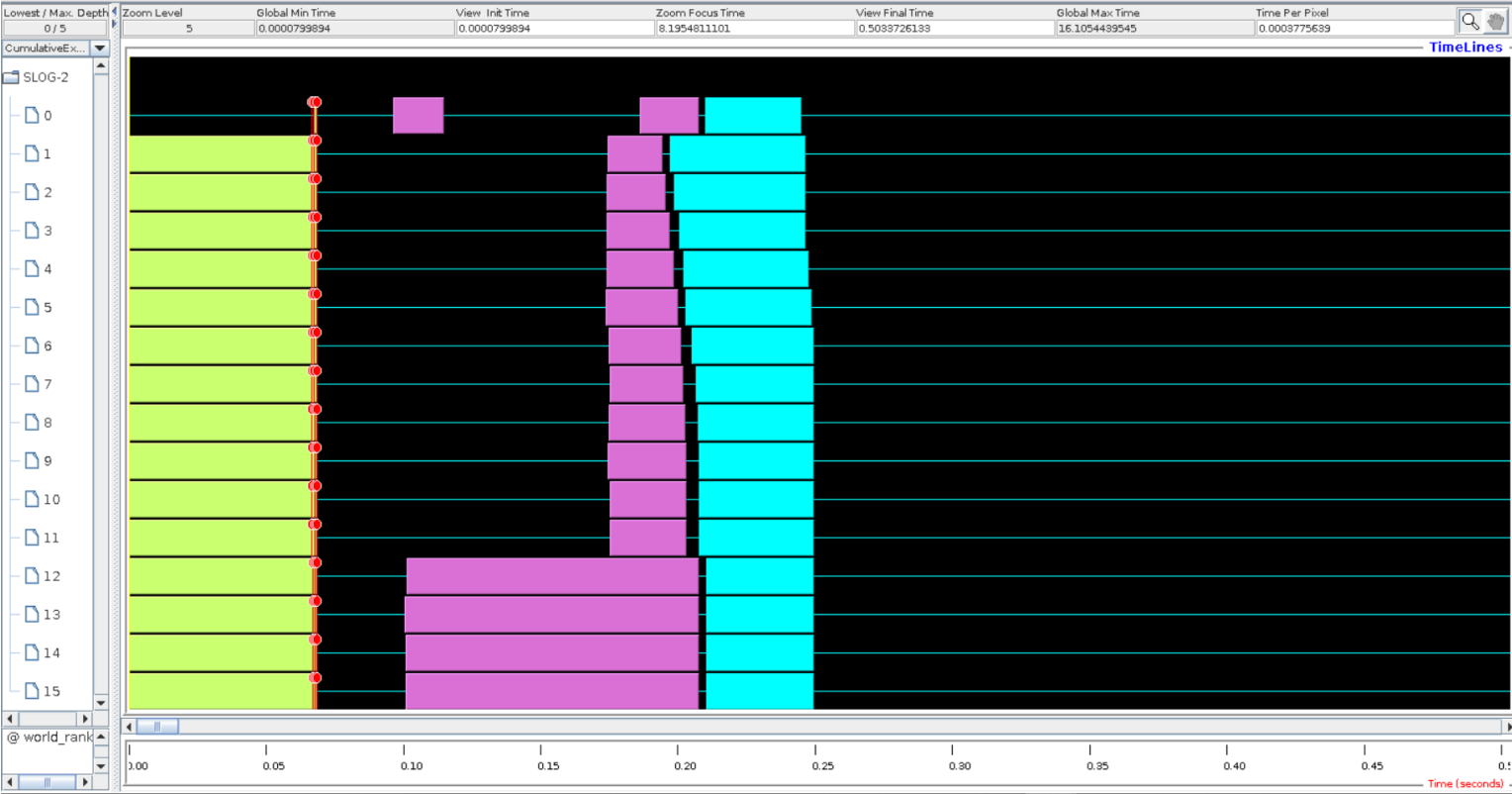
All

SelectDeselect

close









## Вывод:

Разбивать матрицу на подматрицы и отдельно их умножать может очень сильно повысить эффективность программы. Это связано с иерархией памяти, а также запретом транспонировать матрицу В по заданию, вследствие чего мы получили то, что у нас отсутствует локальность данных при обходе матрицы В. Однако это работает только при больших матрицах, иначе доступ к элементам по столбцам будет не сильно дольше, чем при последовательном обходе матрицы(см. график выше).