Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра вычислительных методов

Построение разреженной матрицы и решение СЛАУ

Параллельные высокопроизводительные вычисления

выполнил:
Петров Т. П.
группа 504

15 ноября
Москва, 2024

Содержание

1 Описание задания и программной реализации

1.1 Краткое описание задания

Необходимо реализовать многопоточную программу для решения систем линейных алгебраически уравнений (СЛАУ) на неструктурированной сетке с использованием OpenMP. Алгоритм должен состоять из нескольких этапов:

1. Генерация графа сетки и его матричного представления – создание графа, связей элементов и его представление в разреженном формате CSR
2. Заполнение матрицы СЛАУ – построение матрицы коэффициентов и вектора правой части с использованием тестовых формул
3. Решение СЛАУ – реализация итерационного метода сопряженных градиентов для решения уравнения с поддержкой параллелизма
4. Проверка производительности – измерение времени выполнения каждого этапа и анализ многопоточного ускорения и эффективности алгоритма

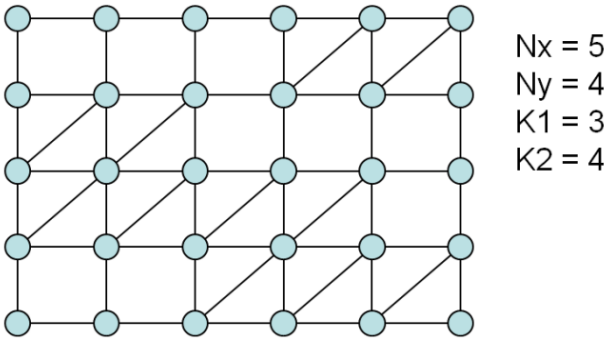


Рис. 1: Пример сетки для генерации графа

1.2 Краткое описание программной реализации

...

1.2.1 Запуск программы

Для запуска на локальных системах достаточно указать количество нитей, а также все необходимые входные данные:

```
OMP_NUM_THREADS=k ./CGSolver Nx Ny K1 K2
```

Для запуска на кластере, использующем систему очередей, запускается скрипт со следующими параметрами:

```
mpisubmit.pl -t k CGSolver -- Nx Ny K1 K2
```

Однако желательно редактирование командного файла для запуска на заданных узлах и привязки к определенному сокету узла.

```
source /polusfs/setenv/setup.SMPI
#BSUB -W 00:15
#BSUB -o CGSolver.%J.out
#BSUB -e CGSolver.%J.err
#BSUB -m polus-c4-ib
#BSUB -R affinity[core(10):distribute=pack(socket=1)]
OMP_NUM_THREADS=k
/polusfs/lsf/openmp/launchOpenMP.py CGSolver Nx Ny K1 K2
```

2 Исследование производительности

2.1 Характеристики вычислительной системы

	PC	Polus
CPU	i5-12400F	IBM POWER 8
Cores	6	10
Threads	2	8
TPP	384 GFLOPS	290 GFLOPS
RAM	2xDDR5-5600	4xDDR4-2400
BW	89.6 GB/s	153.6 GB/s

Для того, чтобы собрать программу и скомпилировать все файлы, необходимо выполнить ряд следующих действий:

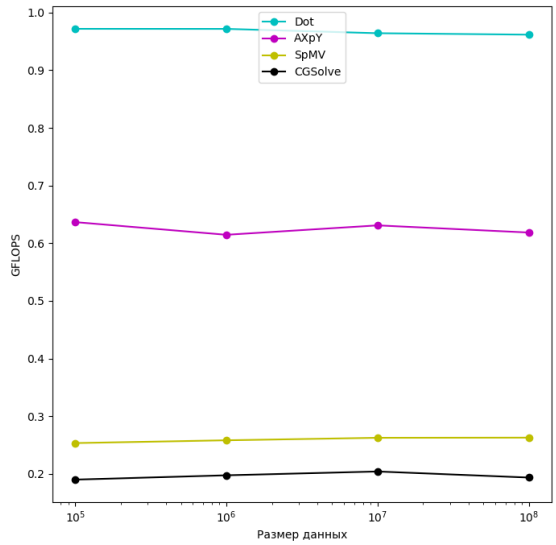
```
mkdir build && cd build
cmake -DENABLE_TESTS=<On|Off> -DUSE_DEBUG_MODE=<On|Off> ..
make -j 4
cd ../bin
```

Также на локальных системах можно запустить несколько простых тестов для проверки корректности работы программы (при условии, что на этапе сборки флаг ENABLE_TESTS был включен):

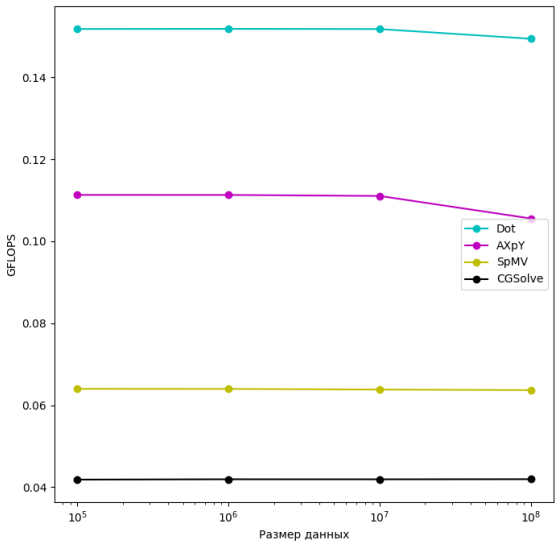
```
cd build/Tests
ctest
```

2.2 Результаты измерений производительности

2.2.1 Последовательная производительность

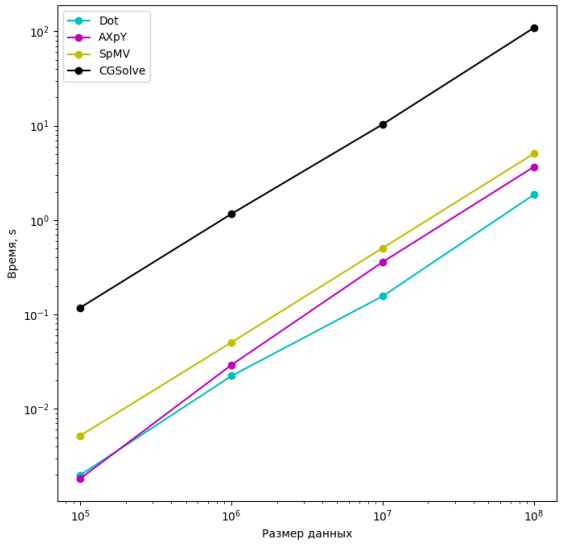


(а) PC

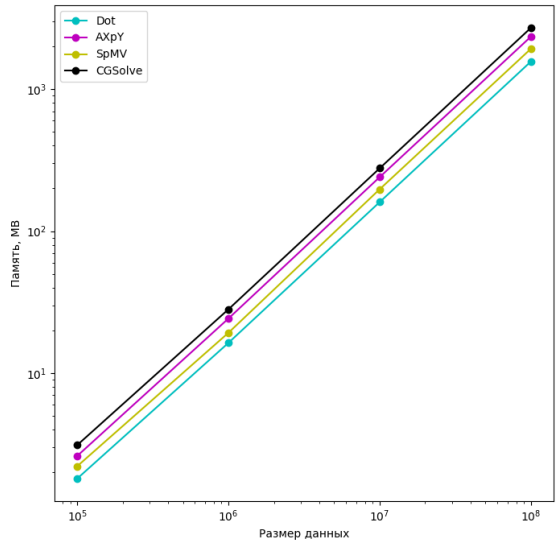


(б) Polus

Рис. 2: Зависимость производительности от размера входных данных



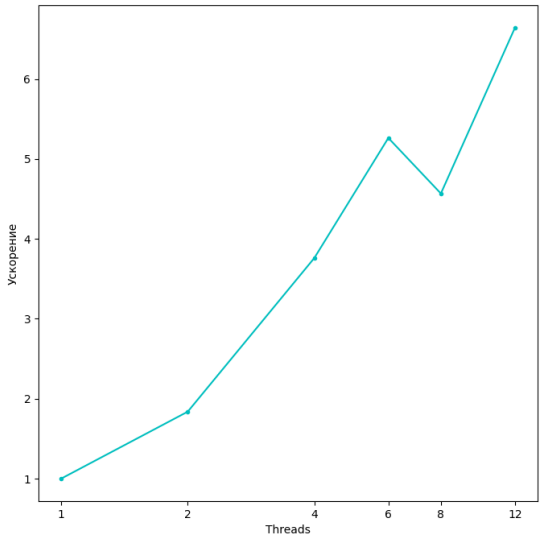
(а)



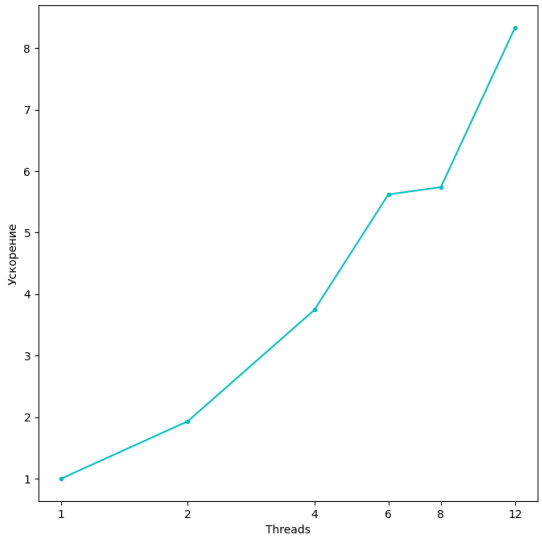
(б)

Рис. 3: Зависимость времени выполнения (а) и выделяемой памяти (б) в зависимости от размера входных данных

2.2.2 Параллельное ускорение

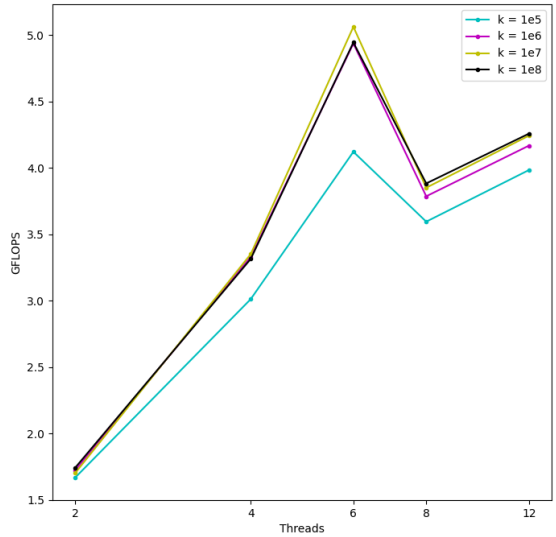


(а)

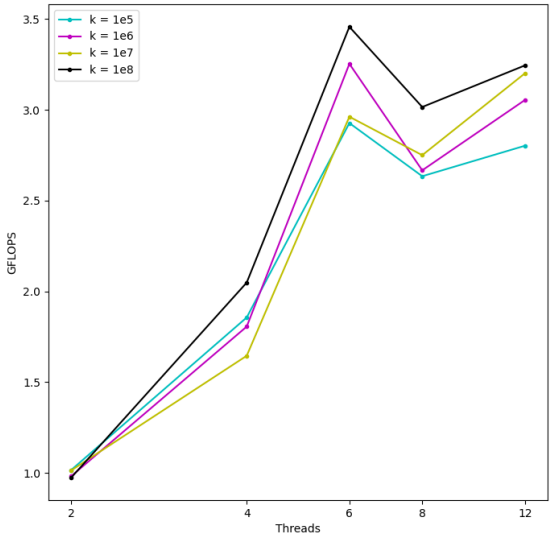


(б)

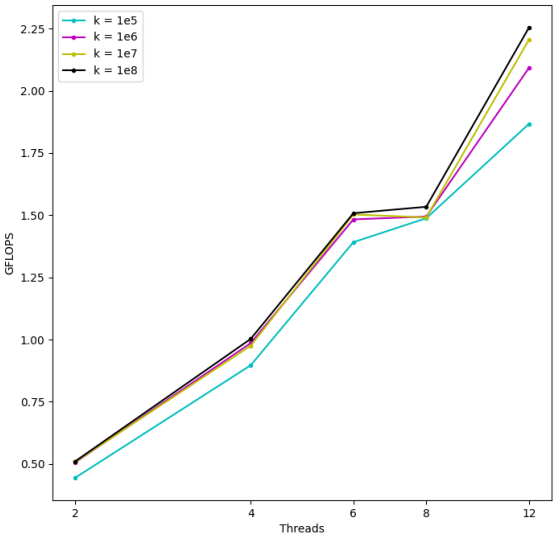
Рис. 4: Ускорение создания (а) и заполнения (б) массивов формата CSR в зависимости от числа нитей



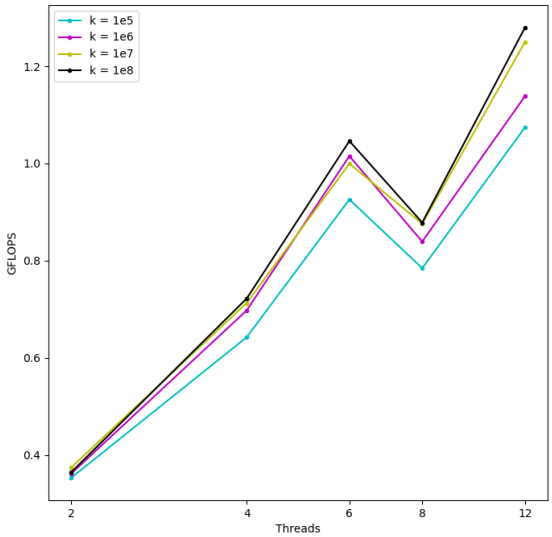
(а) Dot



(б) AXpY

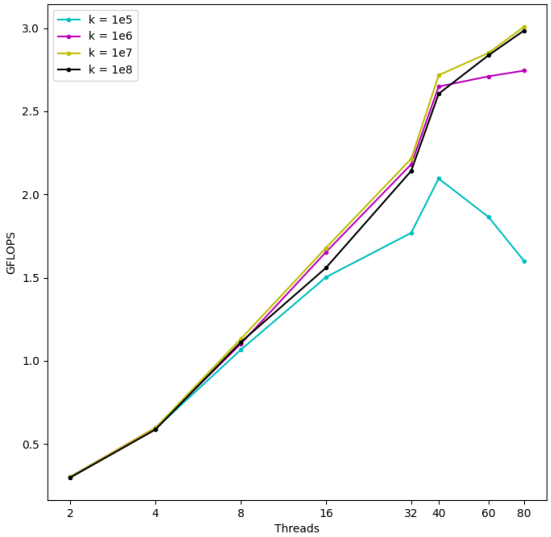


(в) SpMV

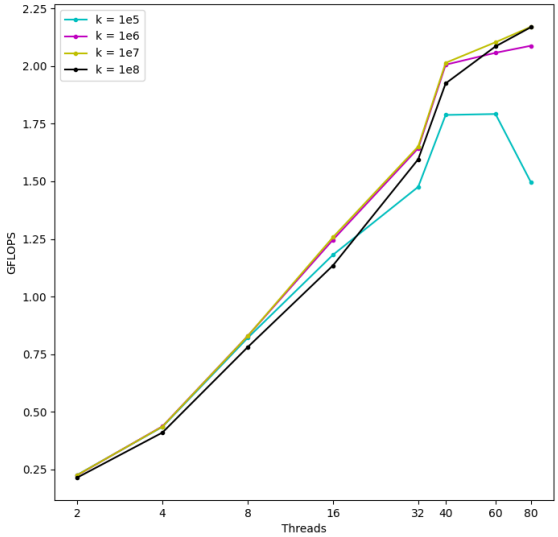


(г) CGSolver

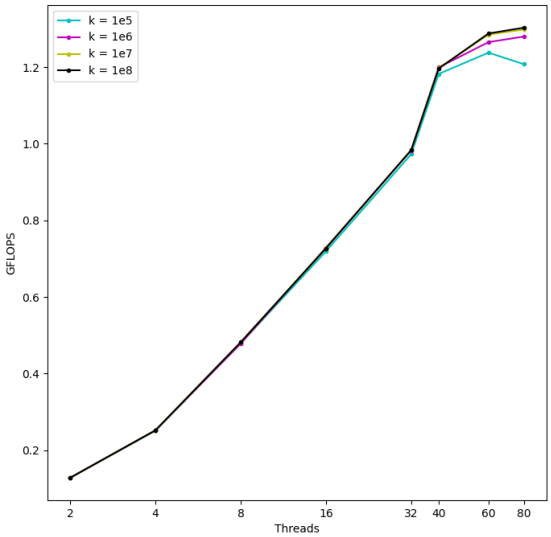
Рис. 5: Зависимость производительности от числа нитей (PC)



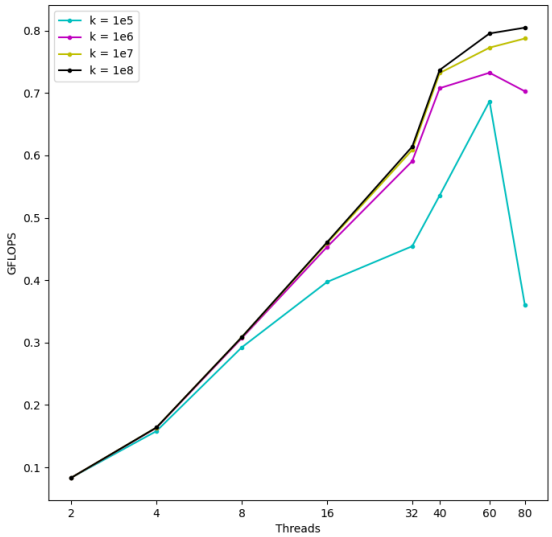
(а) Dot



(б) AXpY



(в) SpMV

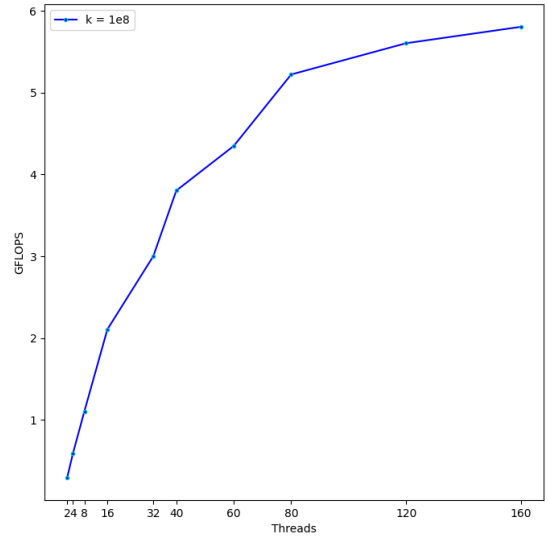


(г) CGSolver

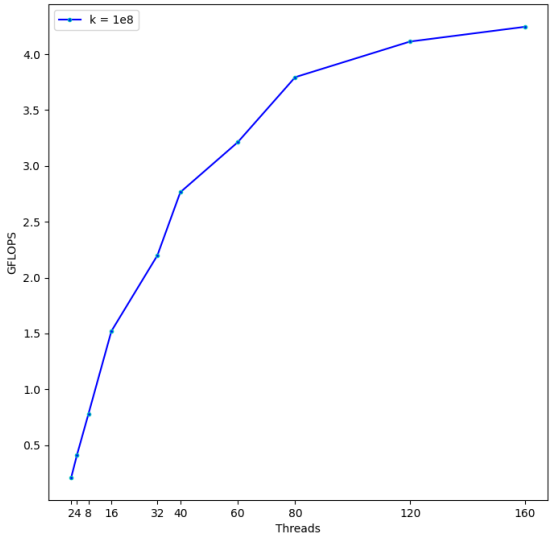
Рис. 6: Зависимость производительности от числа нитей (Polus)

T	2	4	8	16	32	40	64	80
Dot	1.99	3.92	7.44	10.43	14.34	17.44	18.99	19.98
AXpY	2.04	3.89	7.40	10.75	15.12	18.24	19.76	20.55
SpMV	2.01	3.96	7.57	11.42	15.45	18.79	20.23	20.46
CGSolve	1.99	3.91	7.36	10.98	14.64	17.56	18.96	19.18

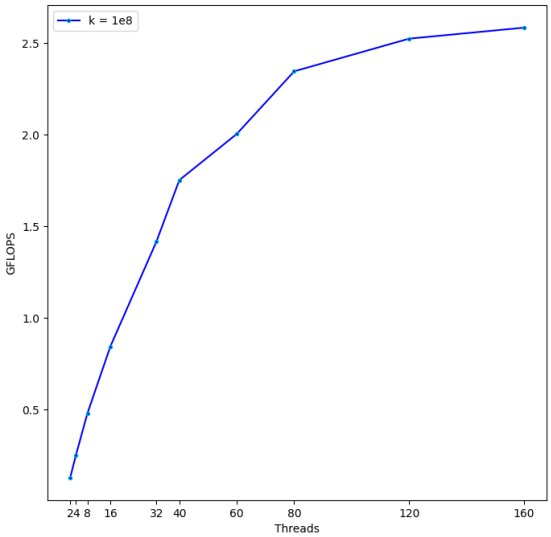
Рис. 7: Расчеты ускорения для каждой из операций при $N = 1e8$ (Polus)



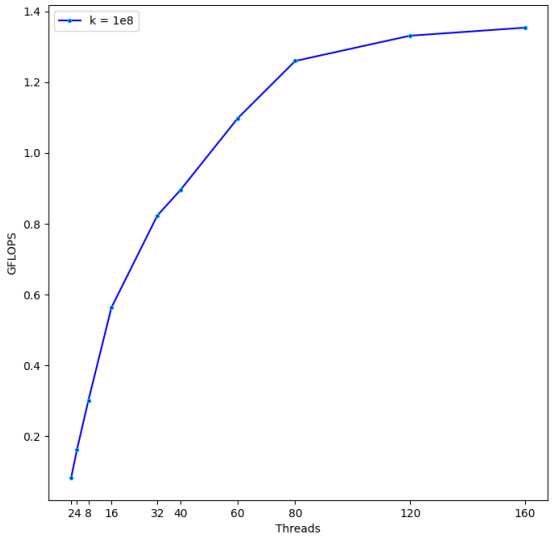
(а) Dot



(б) AXpY



(в) SpMV



(г) CGSolver

Рис. 8: Зависимость производительности от числа нитей (Polus)

T	2	4	8	16	32	40	60	80	120	160
Dot	1.98	3.96	7.42	14.11	20.10	25.46	29.11	34.95	37.51	38.85
AXpY	1.99	3.89	7.38	14.41	20.83	26.19	30.44	35.95	38.97	40.22
SpMV	1.99	3.94	7.51	13.26	22.23	27.52	31.48	36.85	39.65	40.60
CGSolve	1.98	3.86	7.19	13.43	19.63	21.34	26.18	30.03	31.73	32.27

Рис. 9: Расчеты ускорения для каждой из операций на двухсокетной конфигурации при $N = 1e8$ (Polus)

3 Анализ полученных результатов

$TPP_{PC} = 4\ GHz \cdot 6\ Cores \cdot 2\ Threads/Core \cdot 512/64 = 384\ GFLOPS$
$TPP_{Polus} = 290\ GFLOPS$
$BW_{PC} = 2\ Channels \cdot 5600\ MT/s \cdot 8\ bytes = 89,6\ GB/s$
$BW_{Polus} = 8\ Channels \cdot 2400\ MT/s \cdot 8\ bytes = 153,6\ GB/s$

$AI_{dot} = 2\ FLOP/(2 \cdot 8)\ bytes = 1/8$

На каждой итерации считываем из памяти $x[i]$ и $y[i]$, каждый из которых типа 'double' занимает — 8 байт. При этом на каждой итерации происходит 2 операции: умножение $x[i] \cdot y[i]$ и сложение с итоговой суммой

$AI_{AXpY} = 2\ FLOP/(3 \cdot 8)\ bytes = 1/12$

На каждой итерации считываем из памяти $x[i]$ и $y[i]$, каждый из которых типа 'double' занимает — 8 байт, а также записываем в память результат $z[i] = a \cdot x[i] + y[i]$ типа 'double' — тоже 8 байт. Следовательно, всего $3 \cdot 8$ байтов. При этом на каждой итерации происходит 2 операции: умножение $a \cdot x[i]$ и сложение $x[i] + y[i]$

$AI_{SpMV} = 12\ FLOP/(132)\ bytes = 1/11$

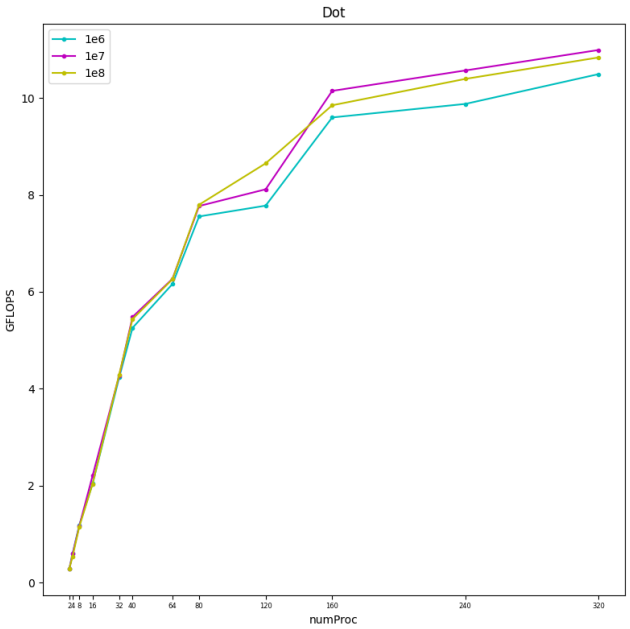
За итерацию будем считать умножение строки i матрицы A на столбец x. Для считывания элементов матрицы A необходимо вначале получить номер $col = ia[i]$, с которого в массиве a начинают храниться элементы i-ой строки — 4 байта (ia имеет тип 'int'). Далее будем считать, что в среднем в строке 6 ненулевых элементов. Для каждого такого элемента необходимо считать значение $a[col]$ типа 'double' — 8 байт, считать $j = ja[col]$ номер столбца — 4 байта (ja имеет тип 'int'), а также считать $x[j]$ типа 'double' — 8 байт. В конце вычислений необходимо записать полученный результат в $y[i]$ типа 'double' — 8 байт. Таким образом получаем, что на каждую строку нам понадобится $4+6\cdot(4+8+8)+8 = 132$. При этом для каждого элемента строки потребуется 2 операции (умножение $a[col] \cdot x[j]$ и сложение в общий результат). Следовательно всего $6 \cdot 2 = 12$ операций.

$TBP = \min(TPP, BW \cdot AI)$
$TBP_{PC,dot} = \min(384\ GFLOPS, 89,6\ GB/s \cdot 1/8) = 11.2\ GFLOPS$
$TBP_{PC,AXpY} = \min(384\ GFLOPS, 89,6\ GB/s \cdot 1/12) = 7.5\ GFLOPS$
$TBP_{PC,SpMV} = \min(384\ GFLOPS, 89,6\ GB/s \cdot 1/11) = 8.1\ GFLOPS$
$TBP_{Polus,dot} = \min(290\ GFLOPS, 153,6\ GB/s \cdot 1/8) = 19.2\ GFLOPS$
$TBP_{Polus,AXpY} = \min(290\ GFLOPS, 153,6\ GB/s \cdot 1/12) = 12.8\ GFLOPS$
$TBP_{Polus,SpMV} = \min(290\ GFLOPS, 153,6\ GB/s \cdot 1/11) = 14\ GFLOPS$

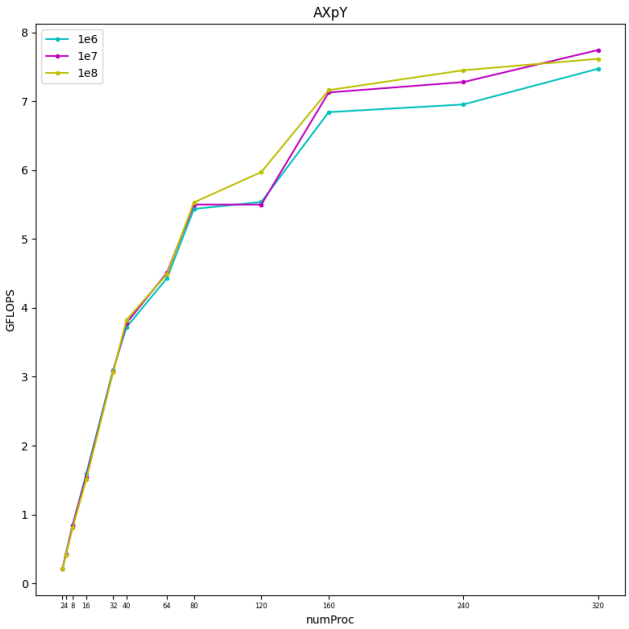
...	Dot	AXpY	SpMV
$PC\ TBP_{Analytical}$	11.2 $GFLOPS$	7.5 $GFLOPS$	8.1 $GFLOPS$
$PC\ RealP$	5.06 $GFLOPS$	3.45 $GFLOPS$	2.25 $GFLOPS$
$Polus\ TBP_{Analytical}$	19.2 $GFLOPS$	12.8 $GFLOPS$	14 $GFLOPS$
$Polus\ RealP$	3.01 $GFLOPS$	2.16 $GFLOPS$	1.31 $GFLOPS$

Рис. 10: Аналитические и реальные значение производительности для каждой из операций

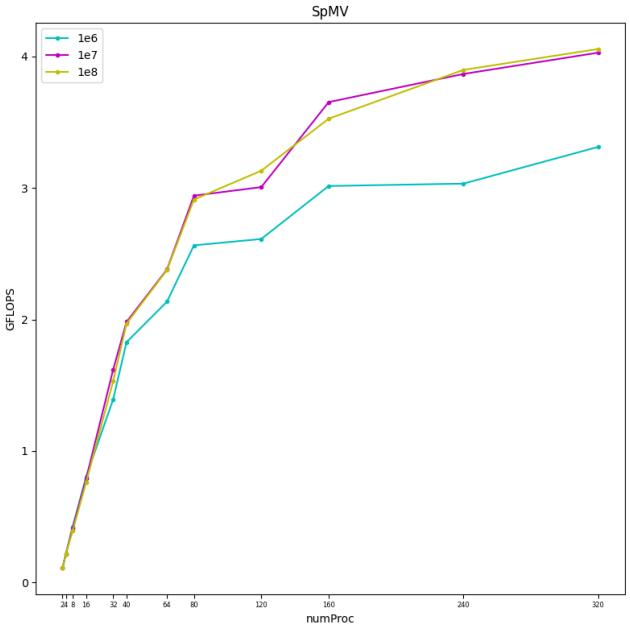
4 MPI



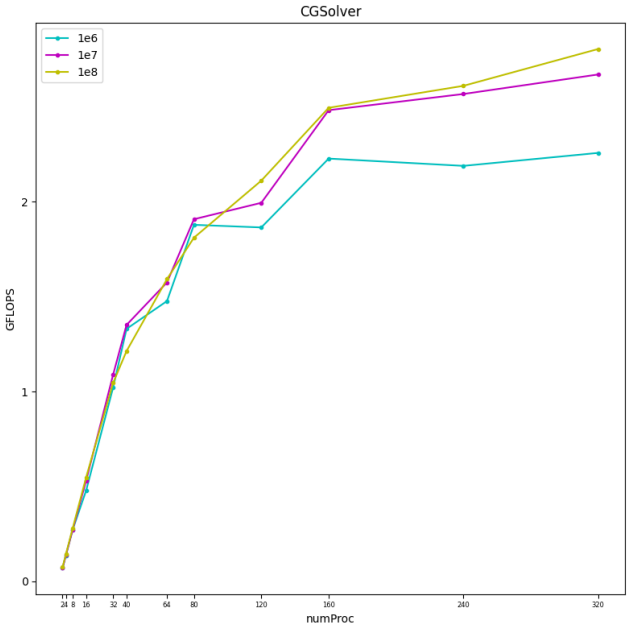
(а) Dot



(б) AXpY

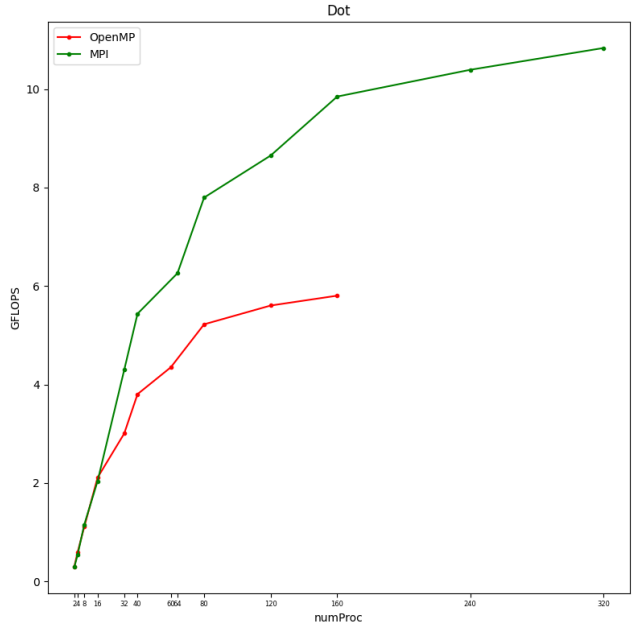


(в) SpMV

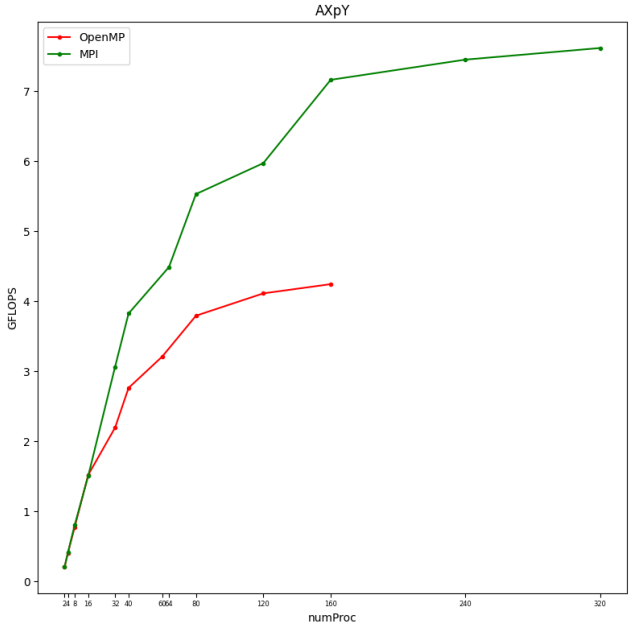


(г) CGSolver

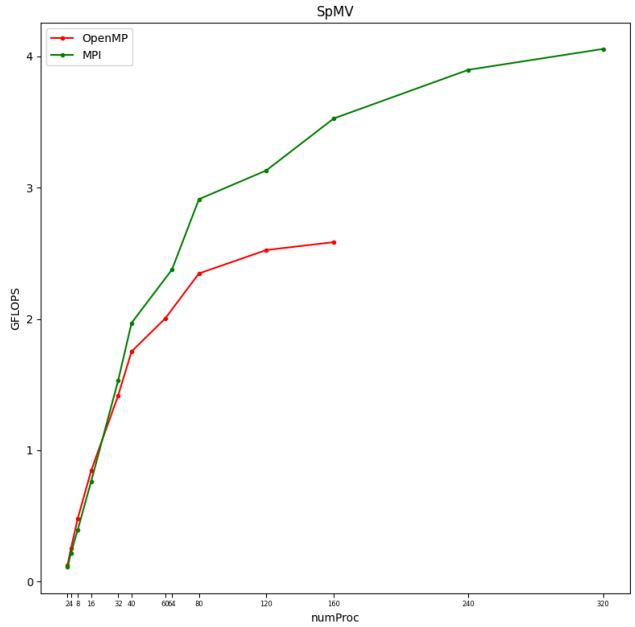
Рис. 11: Зависимость производительности от числа нитей (Polus)



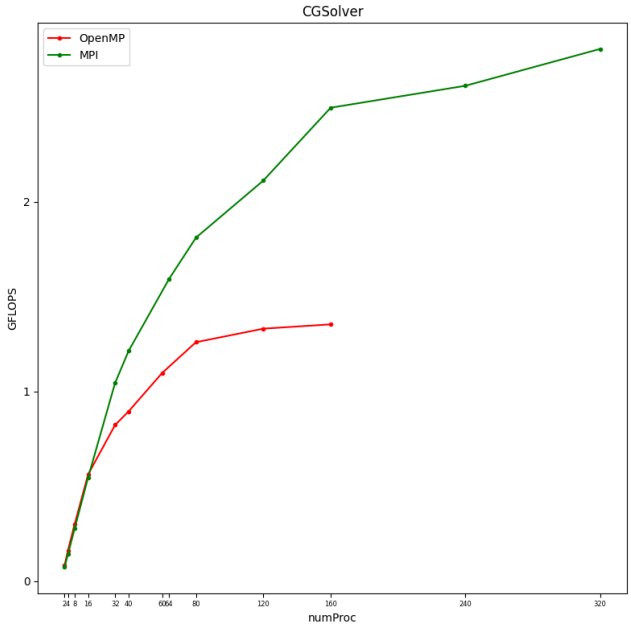
(а) Dot



(б) AXpY



(в) SpMV



(г) CGSolver

Рис. 12: Зависимость производительности от числа нитей (Polus)