



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

Трифонов Владислав Дмитриевич

Отчет по заданию 2:
Многопоточная MPI-реализация солвера CG для СЛАУ
с разреженной матрицей, заданной в формате ELL

Курс “Параллельные вычисления” (1 курс магистратуры ВМК)

группа 528, дата подачи 25.11.2019

Москва, 2019

Содержание

1	Описание задания и программной реализации	3
1.1	Краткое описание задания	3
1.2	Краткое описание программной реализации	3
1.2.1	Сборка	3
1.2.2	Запуск на локальной машине	3
1.2.3	Запуск на кластере POLUS	4
1.2.4	Реализация	4
2	Исследование производительности	6
2.1	Характеристики локальной вычислительной системы	6
2.1.1	Компиляция	6
2.2	Характеристики кластера POLUS	6
2.2.1	Компиляция	6
2.3	Результаты измерений производительности	6
2.3.1	Сравнение MPI с OpenMP на локальной машине	6
2.3.2	Параллельное ускорение на кластере	7
2.3.3	Масштабирование на кластере	12
3	Заключение	20

1 Описание задания и программной реализации

1.1 Краткое описание задания

В качестве задания 2 по курсу “Параллельные вычисления” предлагалось реализовать численное решение СЛАУ с разреженной матрицей, заданной в формате ELLPACK, методом сопряженных градиентов с предобуславливателем Якоби с помощью библиотеки MPI на основе выполненного задания 1. Для этого дополнительно требовалось:

- провести разбиение декартовой расчетной области Nx, Ny, Nz на Px, Py, Pz областей, соответствующих каждому из MPI-процессов
- для каждой подобласти произвести генерацию локальной матрицы и создание схемы обменов
- реализовать MPI-варианты базовых операций
- реализовать предложенный солвер на основе данных операций
- реализовать проверочные вызовы последовательных и многопоточных реализаций для ручной проверки и оценки времени работы алгоритма
- исследовать эффективность реализованных алгоритмов

1.2 Краткое описание программной реализации

Реализация была выполнена на языке C++ (стандарт C++14).

1.2.1 Сборка

Сборка выполняется с помощью Makefile. Компилятор и его параметры задаются в переменных Makefile *CXX*, *CXXFLAGS* (компилятор по-умолчанию – `mpic++`). Скомпилированная программа находится по пути `./build/bin/main`.

1.2.2 Запуск на локальной машине

Параметры запуска можно узнать с помощью команды:

```
# ./build/bin/main --help
```

.

Пример запуска реализованного солвера с предварительной проверкой операций при размере сетки 5x5x5, 2 MPI-процессах с разбиением на 2 подобласти по z, максимальном числе итераций 6, параметром $eps = 0.1$, усреднением по 3 запускам:

```
# mpirun -np 2 ./build/bin/main --qa --nx=5 --ny=5 --nz=5 --px=1 --py=1 --pz=2 --maxit=6
--tol=0.1 --nseeds=3
```

1.2.3 Запуск на кластере POLUS

На кластере POLUS использовался скрипт запуска *mpisubmit.pl*. Запуск проводился аналогично:

```
# mpisubmit.pl -p 2 --stdout out.log --stderr err.log ./build/bin/main -- --qa --nx=5
--ny=5 --nz=5 --px=1 --py=1 --pz=2 --maxit=6 --tol=0.1 --nseeds=3
```

1.2.4 Реализация

Для работы с матрицами в формате ELLPACK модуль *ell_utils.cpp* был портирован с прошлой реализации на с (задание 1) на с++.

Базовые операции работы с векторами и матрицами в модуле *ops_utils.cpp* были портированы с прошлой реализации на с (задание 1) на с++.

В модуле *mpi_solver.cpp* было реализовано деление расчетной сетки на подобласти, генерация локальной матрицы, создание массивов *part* и *L2G*, создание схемы обмена, тестирование базовых операций, солвер и тестирование солвера. Для этого были написаны следующие основные функции:

- *create_part_L2G* – создание массивов *part* и *L2G* для расширенной подобласти
- *init_local_matrix* – создание локальной матрицы для подобласти с помощью массивов *part* и *L2G*
- *create_messaging_vectors* – создание схемы обмена: списков номеров ячеек для обмена с каждым из соседей
- *update_halo* – обновление гало-ячеек с помощью схемы обмена и MPI-обменов
- *mpi_dot* – MPI-реализация операции dot с помощью Allreduce обмена

- *run_qa* – тестирование базовых операций
- *solve* – реализация солвера с помощью MPI-обменов
- *run_solver* – тестирование солвера
- *test_mpi_solver* – обертка для запуска тестирования

В модуле *main* производится парсинг аргументов командной строки и использование приведенных функций.

2 Исследование производительности

2.1 Характеристики локальной вычислительной системы

Исследование было выполнено на ПК с 4-х ядерным CPU Intel i5-3570K, работающим на частоте 4.1 GHz, с памятью 4*4Gb DDR3-1600MHz, работающей в двухканальном режиме. Пиковая производительность 131,2 GFLOPS, пиковая пропускная способность памяти $12.8 * 2 = 25.6$ Gb/s. ОС – Ubuntu 16.04.

2.1.1 Компиляция

Компиляция проводилась с помощью компилятора `g++ 5.4.0` с флагами `-g -Wall -O3 -Wl,-z,defs -Wextra -std=c++14`. Использовалась библиотека *MPICH* 3.2.

2.2 Характеристики кластера POLUS

На кластере IBM POLUS доступно 4 вычислительных узла, каждый состоит из 2 десятиядерных процессоров IBM POWER8. Пиковая производительность 55,84 TFLOPS. Пиковая пропускная способность памяти на узел 232 GB/s. ОС – Red Hat 7.5.

2.2.1 Компиляция

Компиляция проводилась с помощью компилятора *mpixlC* с флагами `-g -Wall -O3 -Wl,-z,defs -Wextra -std=c++14 -qarch=pwr8`. В качестве MPI-модуля использовался SpectrumMPI (команда *module load SpectrumMPI* до компиляции).

2.3 Результаты измерений производительности

2.3.1 Сравнение MPI с OpenMP на локальной машине

При проведении экспериментов проводилось 3 запуска, далее время усреднялось (`--nseeds=3`). В OpenMP-варианте использовалось 2 и 4 потока (`--nt=2`, `--nt=4`). В MPI-варианте использовалось 2 и 4 MPI-процесса (`-np 2`, `-np 4`) Параметр $eps = 0.1$, $maxit = 20$ (`--tol=0.1 --maxit=20`). Так же вручную проводилась проверка совпадения контрольных значений (нормы L_2 , L_{inf}) для двух версий. Размер системы $N = 100 * 100 * 1000 =$

10^7 . Для MPI версии разбиение производилось по координате z ($--px=1 --py=1 --pz=2$, $--px=1 --py=1 --pz=4$).

	dot	axpby	SpMV	solver
Время последовательной реализации (в секундах) (T_1)	0.009	0.027	0.107	1.963
Время OpenMP, 2 потока (в секундах), (T_{2_o})	0.008	0.019	0.076	1.531
<i>Ускорение OpenMP, 2 потока ($S_{2_o} = T_1/T_{2_o}$)</i>	<i>1.125</i>	<i>1.42</i>	<i>1.4</i>	<i>1.28</i>
<i>Эффективность OpenMP, 2 потока ($S_{2_o}/2$)</i>	<i>56%</i>	<i>71%</i>	<i>70%</i>	<i>64%</i>
Время OpenMP, 4 потока (в секундах), (T_{4_o})	0.009	0.017	0.075	1.496
<i>Ускорение OpenMP, 4 потока ($S_{4_o} = T_1/T_{4_o}$)</i>	<i>1.0</i>	<i>1.59</i>	<i>1.43</i>	<i>1.31</i>
<i>Эффективность OpenMP, 4 потока ($S_{4_o}/4$)</i>	<i>25%</i>	<i>39.75%</i>	<i>35.75%</i>	<i>32.75%</i>
Время MPI, 2 потока (в секундах), (T_{2_m})	0.009	0.022	0.072	1.402
<i>Ускорение MPI, 2 потока ($S_{2_m} = T_1/T_{2_m}$)</i>	<i>1.0</i>	<i>1.23</i>	<i>1.49</i>	<i>1.4</i>
<i>Эффективность MPI, 2 потока ($S_{2_m}/2$)</i>	<i>50%</i>	<i>61.36%</i>	<i>74%</i>	<i>70%</i>
Время MPI, 4 потока (в секундах), (T_{4_m})	0.011	0.023	0.074	1.380
<i>Ускорение MPI, 4 потока ($S_{4_m} = T_1/T_{4_m}$)</i>	<i>0.81</i>	<i>1.17</i>	<i>1.45</i>	<i>1.42</i>
<i>Эффективность MPI, 4 потока ($S_{4_m}/4$)</i>	<i>20%</i>	<i>29%</i>	<i>36%</i>	<i>35%</i>

Таблица 1: Сравнение OpenMP с MPI на локальной машине

Результаты обеих версий совпадают по контрольным значениям. Времена выполнения сопоставимы.

2.3.2 Параллельное ускорение на кластере

Для измерения параллельного ускорения были проведены аналогичные запуски на кластере с размером системы $N = 10^7$, $N = 10^8$ и разным числом MPI процессов, соответствующих одному ядру кластера – $P = 1$, $P = 4$, $P = 8$, $P = 16$, $P = 32$, $P = 64$.

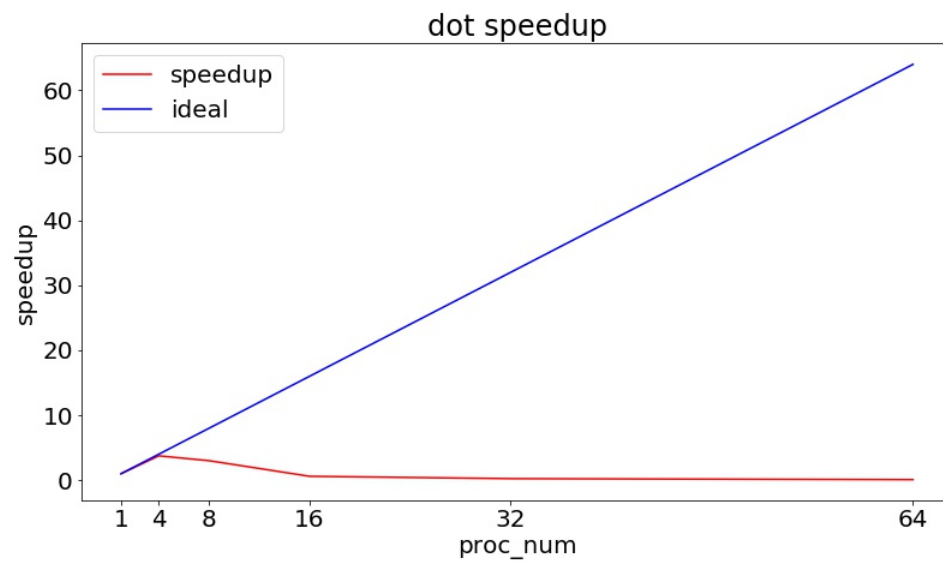


Рис. 1: Ускорение dot, $N = 10^7$

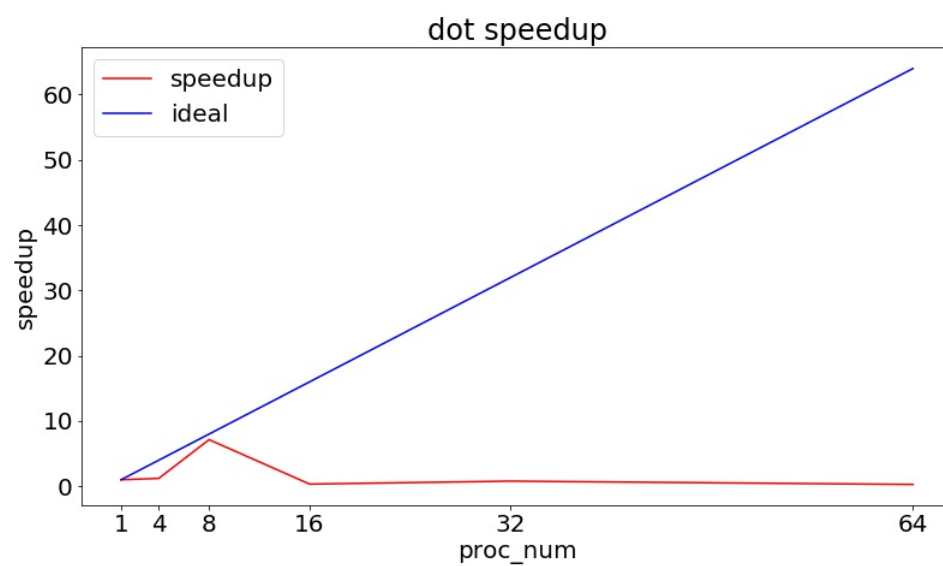


Рис. 2: Ускорение dot, $N = 10^8$

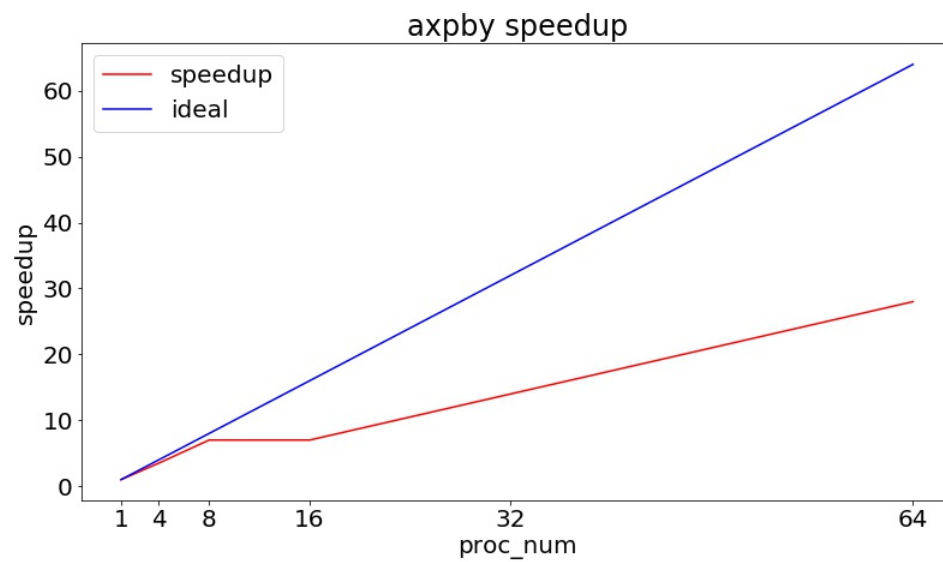


Рис. 3: Ускорение axpby, $N = 10^7$

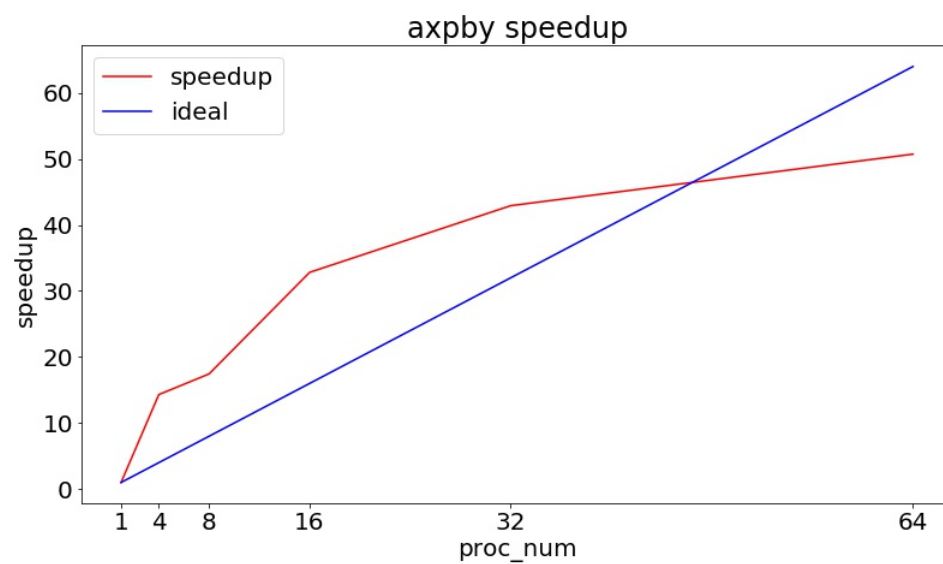


Рис. 4: Ускорение axpby, $N = 10^8$

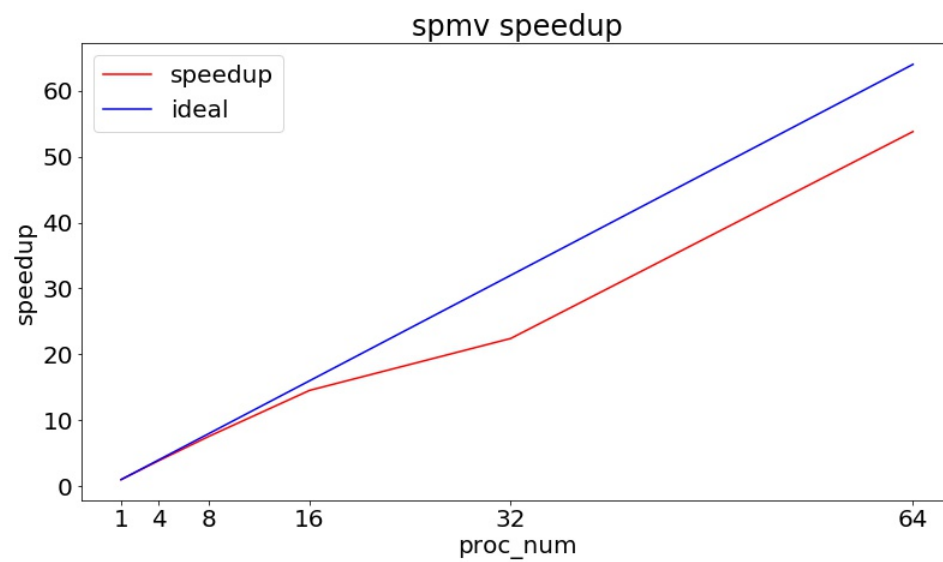


Рис. 5: Ускорение SpMV, $N = 10^7$

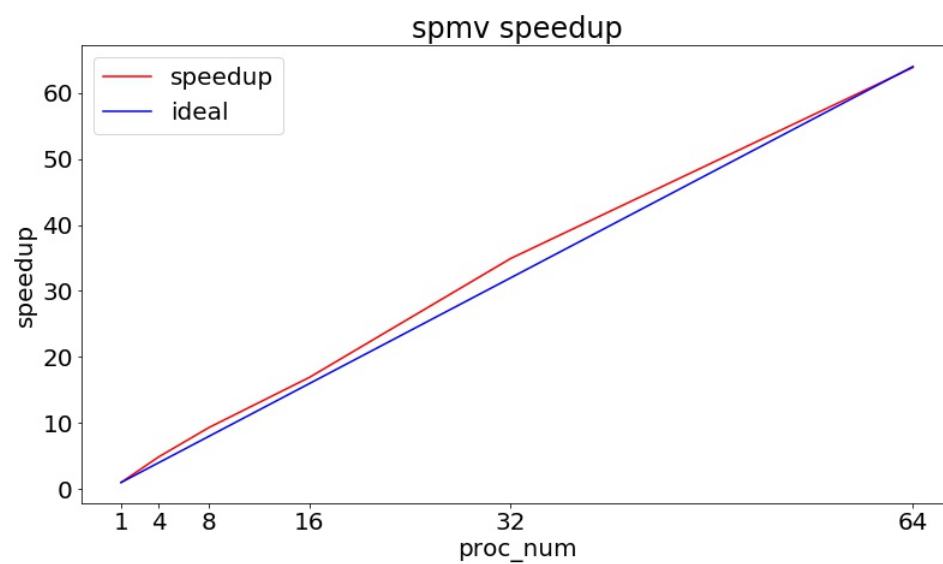


Рис. 6: Ускорение SpMV, $N = 10^8$

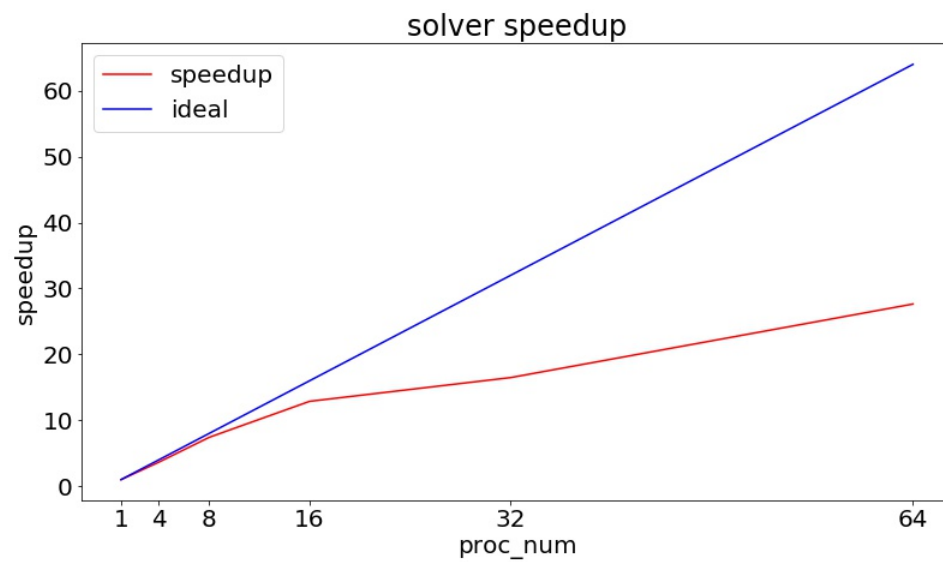


Рис. 7: Ускорение солвера, $N = 10^7$

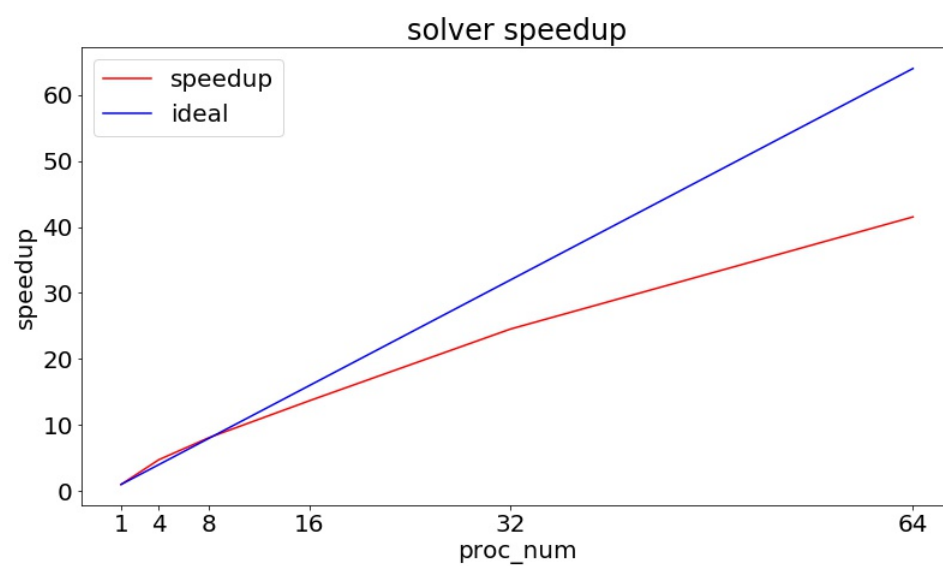


Рис. 8: Ускорение солвера, $N = 10^8$

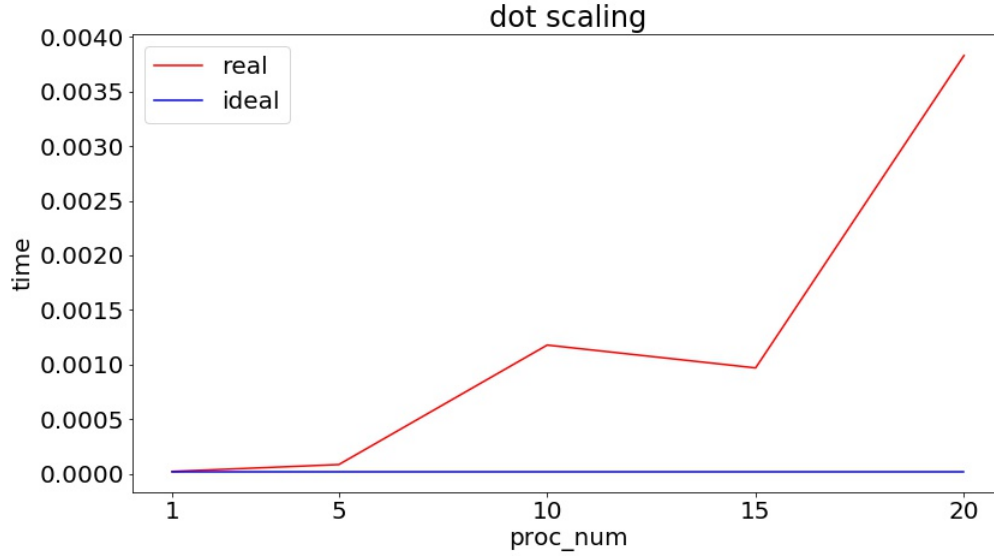


Рис. 9: Масштабирование dot, $N = 10^4 * P$

Вид графика для операции dot обусловлен тем, что в реализации используется операция MPI_Allreduce, при которой все узлы кластера передают частичную сумму мастер-процессу, он производит редукцию и рассылку всем полученной суммы.

Вид графика для операции axrbv обусловлен тем, что в ее реализации не производится обменов, векторы линейно комбинируются по расширенной подобласти.

2.3.3 Масштабирование на кластере

Для измерения масштабирования были проведены запуски на кластере с разным числом MPI процессов, соответствующих одному ядру кластера – $P = 1$, $P = 5$, $P = 10$, $P = 15$, $P = 20$. Размерность системы увеличивалась пропорционально количеству процессов – $N = 10^4 * P$, $N = 10^5 * P$, $N = 10^6 * P$.

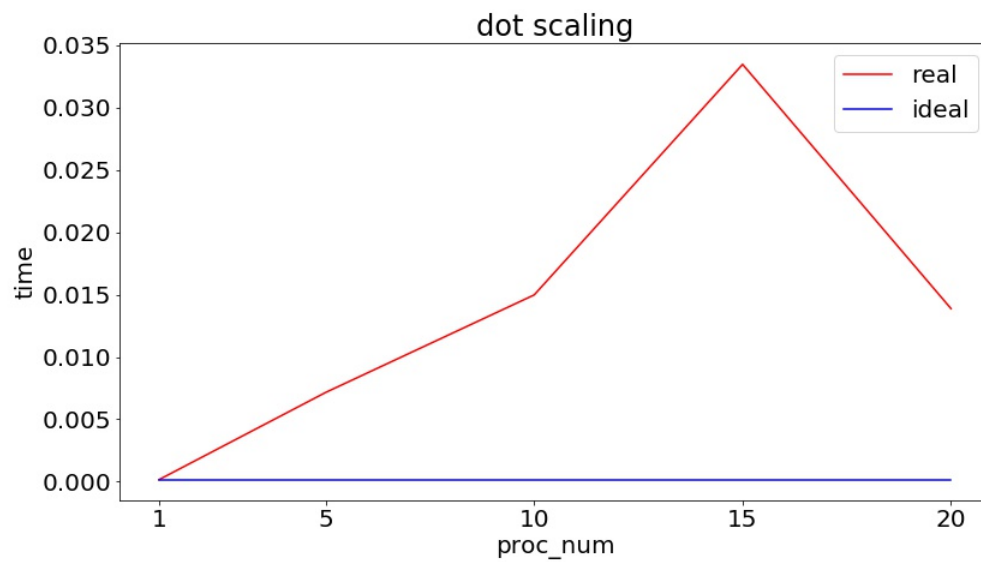


Рис. 10: Масштабирование dot, $N = 10^5 * P$

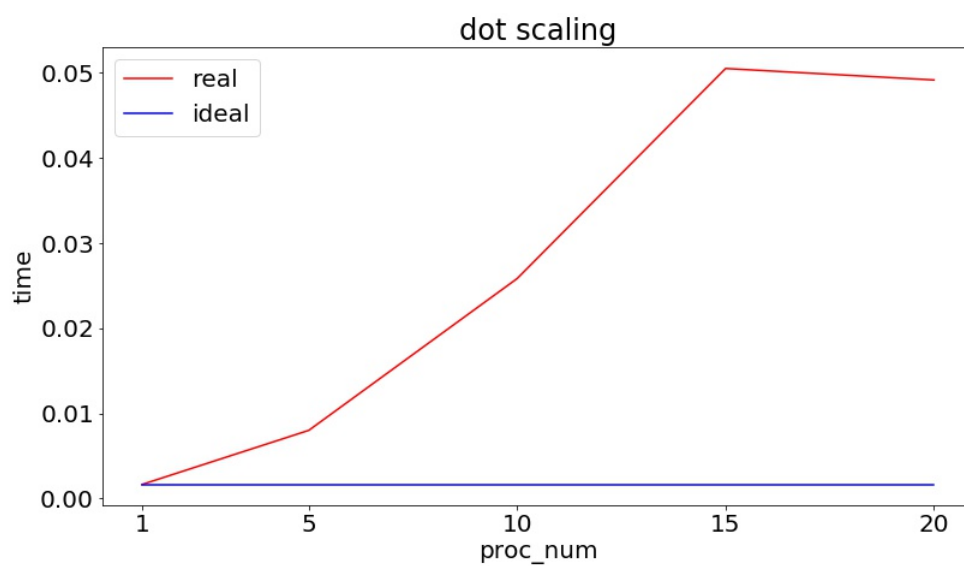


Рис. 11: Масштабирование dot, $N = 10^6 * P$

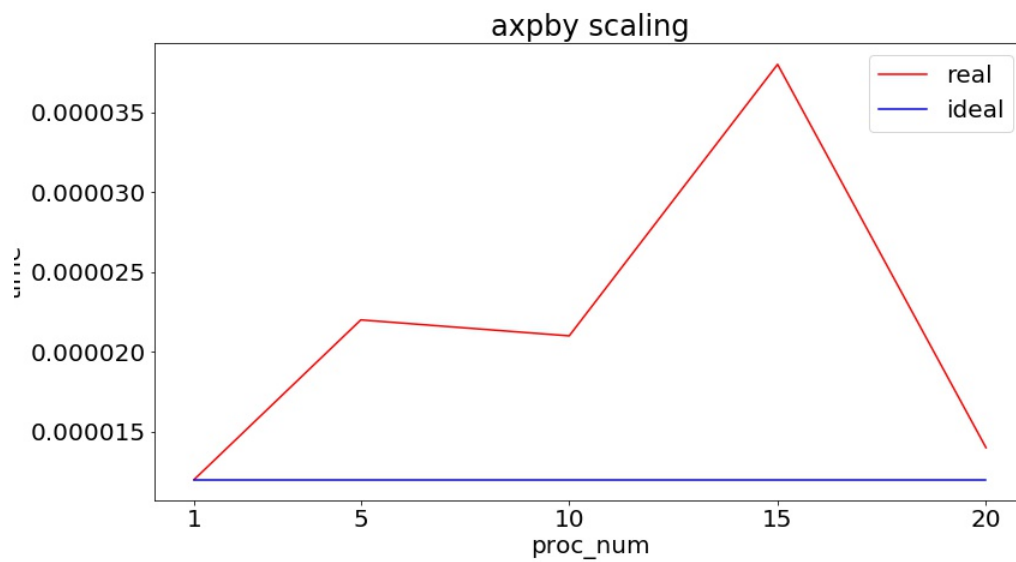


Рис. 12: Масштабирование axpby, $N = 10^4 * P$

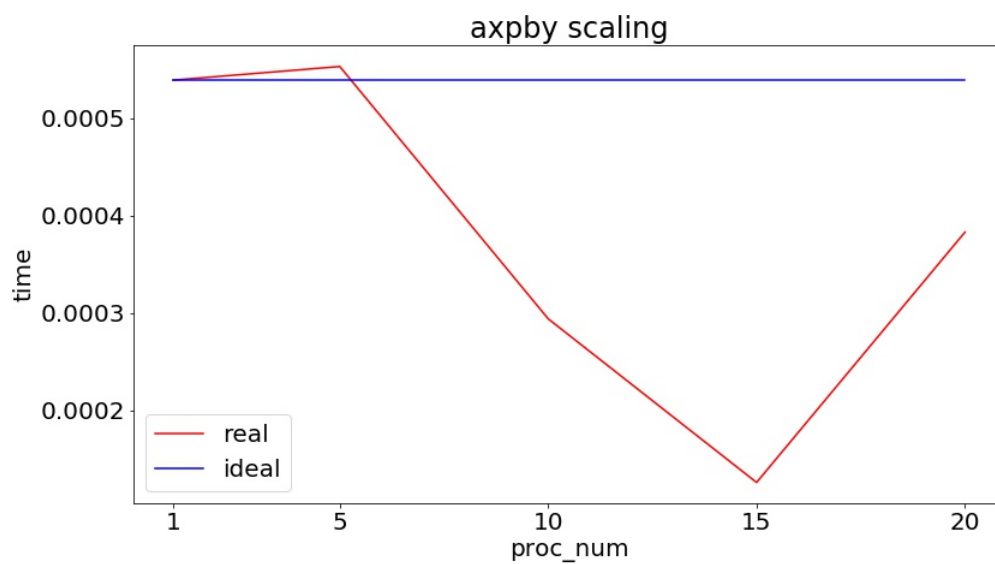


Рис. 13: Масштабирование axpby, $N = 10^5 * P$

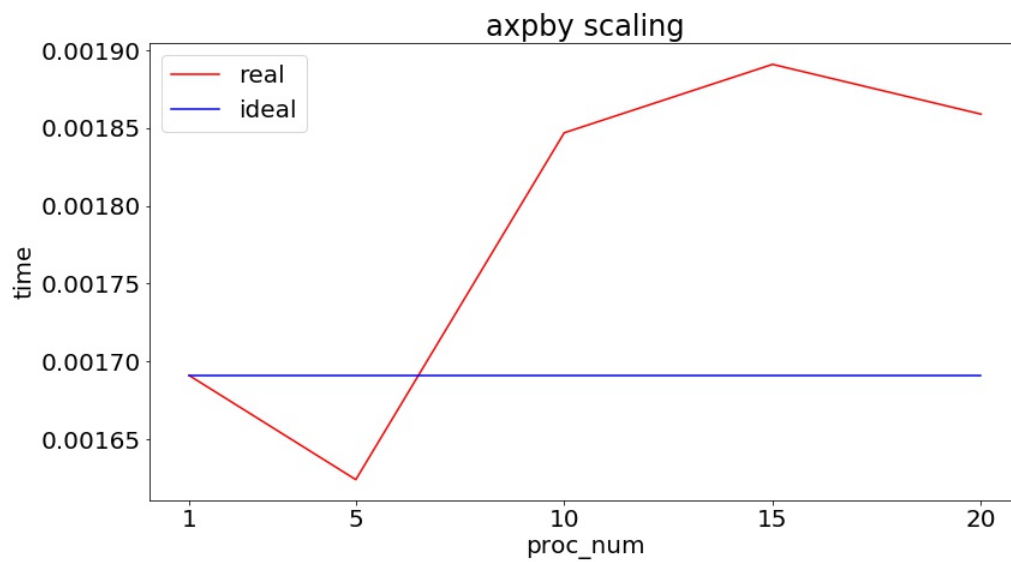


Рис. 14: Масштабирование axpby, $N = 10^6 * P$

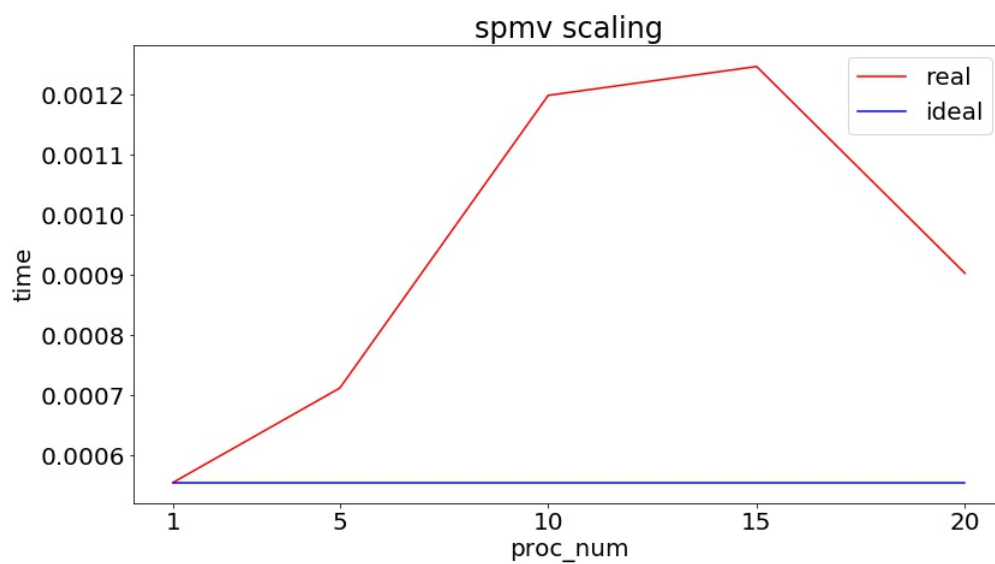


Рис. 15: Масштабирование SpMV, $N = 10^4 * P$

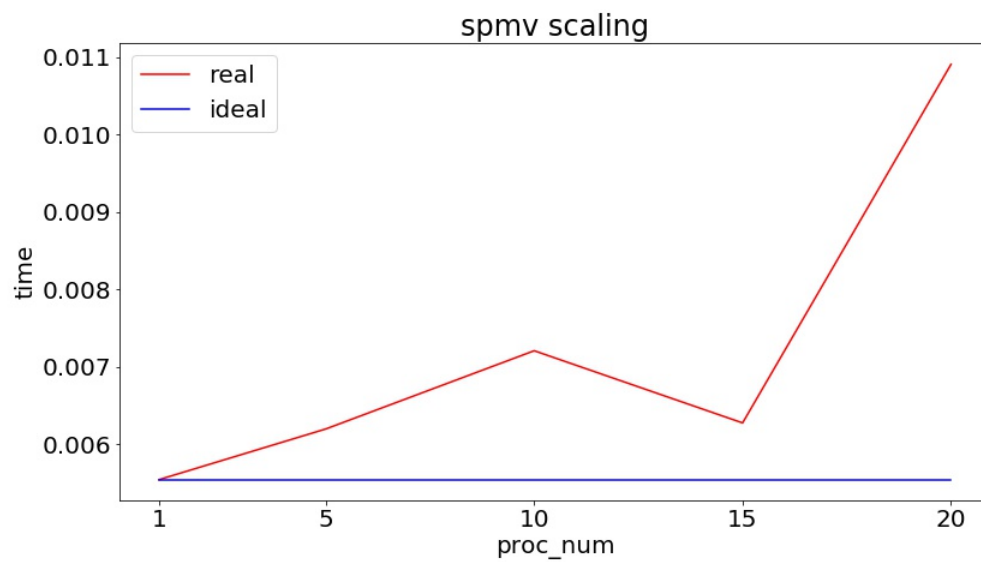


Рис. 16: Масштабирование SpMV, $N = 10^5 * P$

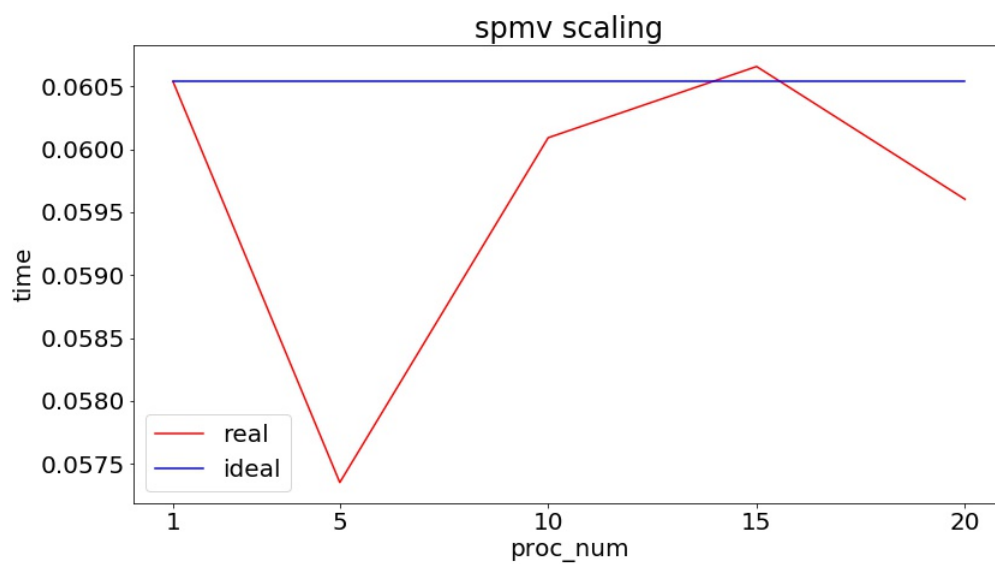


Рис. 17: Масштабирование SpMV, $N = 10^6 * P$

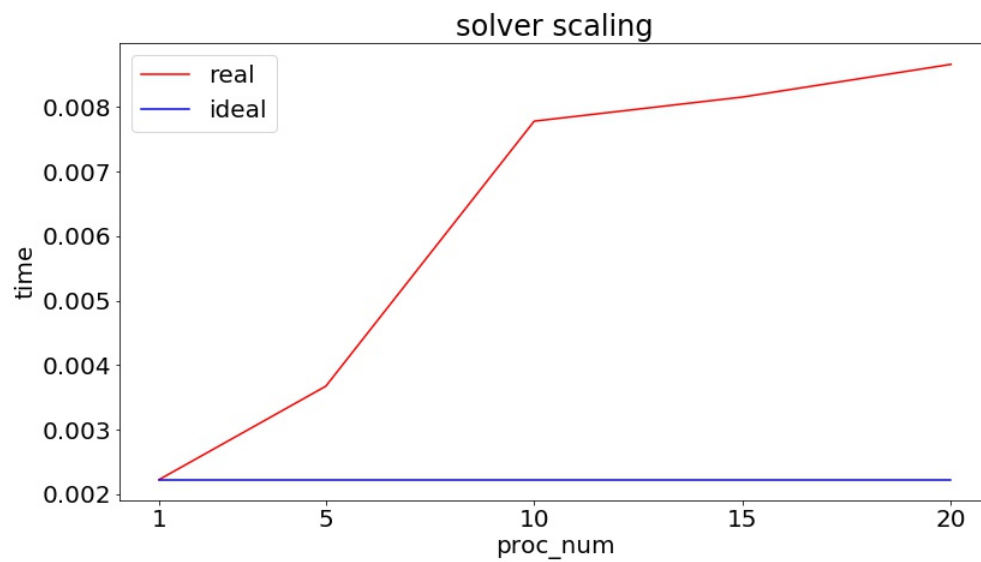


Рис. 18: Масштабирование солвера, $N = 10^4 * P$

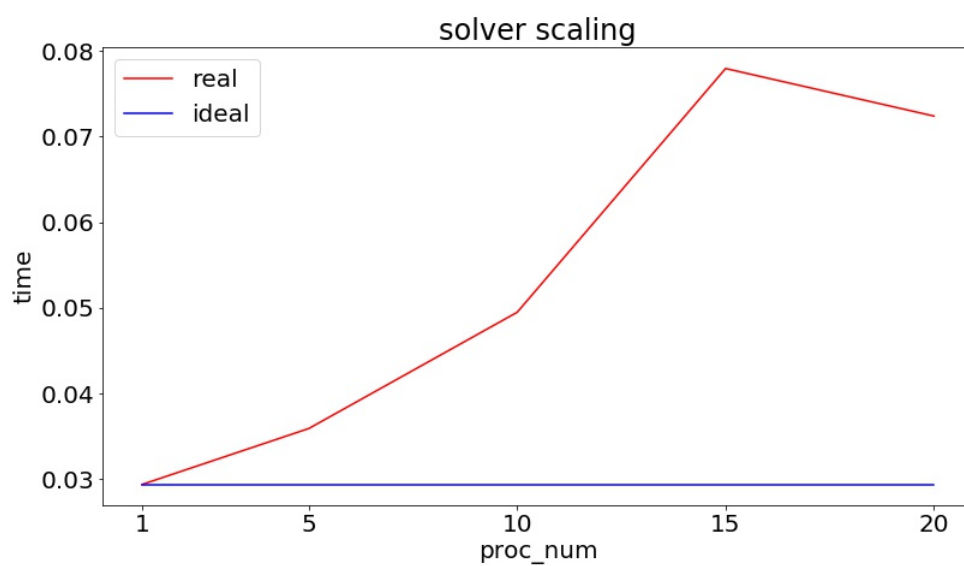


Рис. 19: Масштабирование солвера, $N = 10^5 * P$

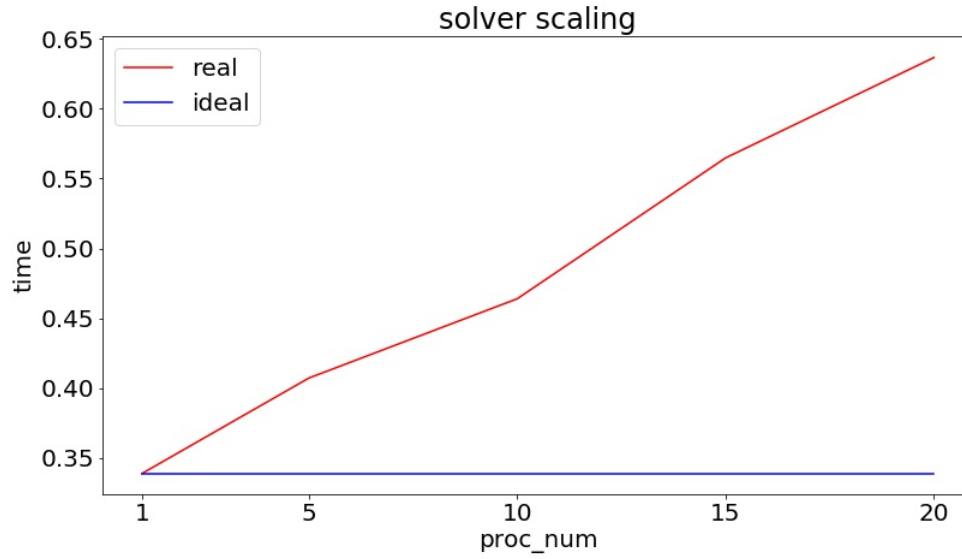


Рис. 20: Масштабирование солвера, $N = 10^6 * P$

	$N = 10^4 * P$	$N = 10^5 * P$	$N = 10^6 * P$
$P = 1$	0.000021	0.000161	0.001666
$P = 5$	0.000084	0.007158	0.008021
$P = 10$	0.001178	0.014975	0.025855
$P = 15$	0.000969	0.033467	0.050523
$P = 20$	0.003829	0.013876	0.049177

Таблица 2: Время выполнения (в секундах) для операции dot

	$N = 10^4 * P$	$N = 10^5 * P$	$N = 10^6 * P$
$P = 1$	0.000012	0.000539	0.001691
$P = 5$	0.000022	0.000553	0.001624
$P = 10$	0.000021	0.000294	0.001847
$P = 15$	0.000038	0.000126	0.001891
$P = 20$	0.000014	0.000383	0.001859

Таблица 3: Время выполнения (в секундах) для операции ахрбу

	$N = 10^4 * P$	$N = 10^5 * P$	$N = 10^6 * P$
$P = 1$	0.000555	0.005546	0.060537
$P = 5$	0.000712	0.006200	0.057352
$P = 10$	0.001199	0.007210	0.060092
$P = 15$	0.001247	0.006278	0.060658
$P = 20$	0.000903	0.010908	0.059603

Таблица 4: Время выполнения (в секундах) для операции SpMV

	$N = 10^4 * P$	$N = 10^5 * P$	$N = 10^6 * P$
$P = 1$	0.002227	0.029375	0.338978
$P = 5$	0.003674	0.035908	0.407232
$P = 10$	0.007781	0.049482	0.464001
$P = 15$	0.008155	0.077948	0.564827
$P = 20$	0.008661	0.072395	0.636663

Таблица 5: Время выполнения (в секундах) солвера

3 Заключение

В результате практического задания была выполнена многопоточная MPI-реализация солвера СЛАУ с разреженной матрицей. Также были измерены параллельное ускорение и масштабирование реализаций базовых операций и солвера в целом на кластере POLUS.