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

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 c (задание 1) на c++.

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

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 10 запусков, далее время усреднялось (--nseeds=10). 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 L2, Linf) 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)	**0.010**	**0.027**	**0.103**	**1.866**
Время OpenMP, 2 потока (в секундах), (T_{2_o})	**0.008**	**0.017**	**0.070**	**1.460**
Ускорение OpenMP, 2 потока ($S_{2_o} = T_1/T_{2_o}$)	***1.125***	***1.58***	***1.47***	***1.28***
Эффективность OpenMP, 2 потока ($S_{2_o}/2$)	*56%*	*79%*	*73.5%*	*64%*
Время OpenMP, 4 потока (в секундах), (T_{4_o})	**0.009**	**0.016**	**0.068**	**1.439**
Ускорение OpenMP, 4 потока ($S_{4_o} = T_1/T_{4_o}$)	***1.1***	***1.69***	***1.51***	***1.30***
Эффективность OpenMP, 4 потока ($S_{4_o}/4$)	*27.5%*	*42.25%*	*37.75%*	*32.5%*
Время MPI, 2 потока (в секундах), (T_{2_m})	**0.008**	**0.024**	**0.074**	**1.417**
Ускорение MPI, 2 потока ($S_{2_m} = T_1/T_{2_m}$)	***1.125***	***1.125***	***1.39***	***1.32***
Эффективность MPI, 2 потока ($S_{2_m}/2$)	*56%*	*56%*	*69.5%*	*66%*
Время MPI, 4 потока (в секундах), (T_{4_m})	**0.011**	**0.027**	**0.073**	**1.391**
Ускорение MPI, 4 потока ($S_{4_m} = T_1/T_{4_m}$)	***0.91***	***1.0***	***1.41***	***1.34***
Эффективность MPI, 4 потока ($S_{4_m}/4$)	*22.75%*	*25%*	*35.25%*	*33.5%*

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 – $P = 1$, $P = 10$, $P = 20$, $P = 30$, $P = 40$, $P = 50$, $P = 60$.

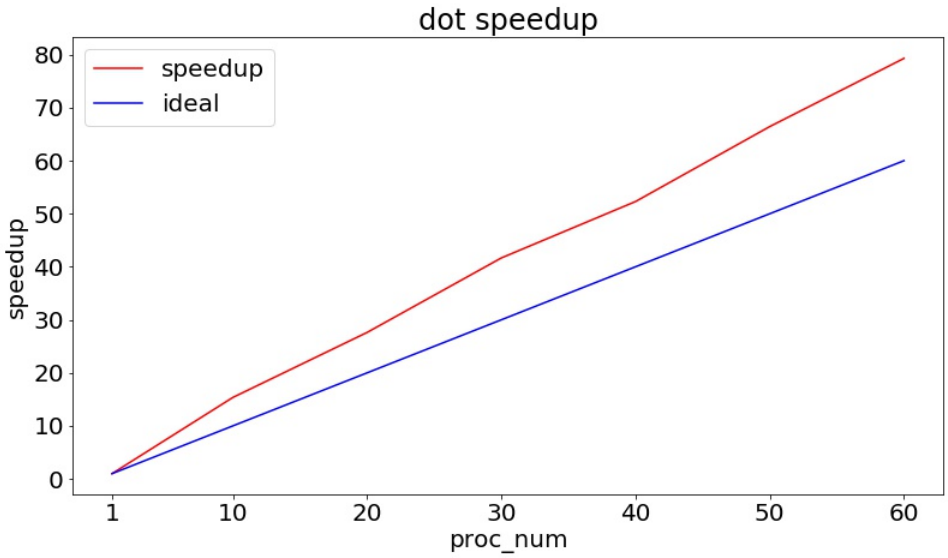

Рис. 1: Ускорение dot, $N = 10^7$

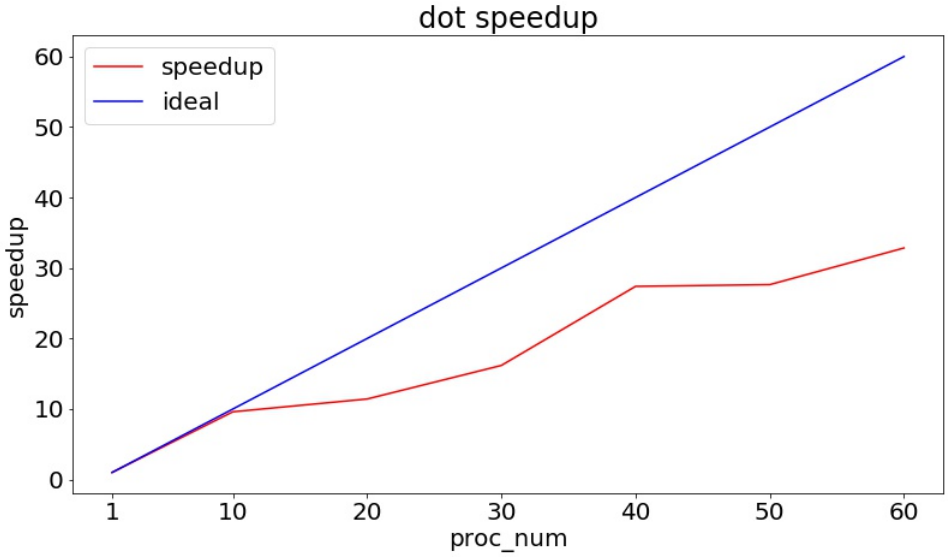

Рис. 2: Ускорение dot, $N = 10^8$

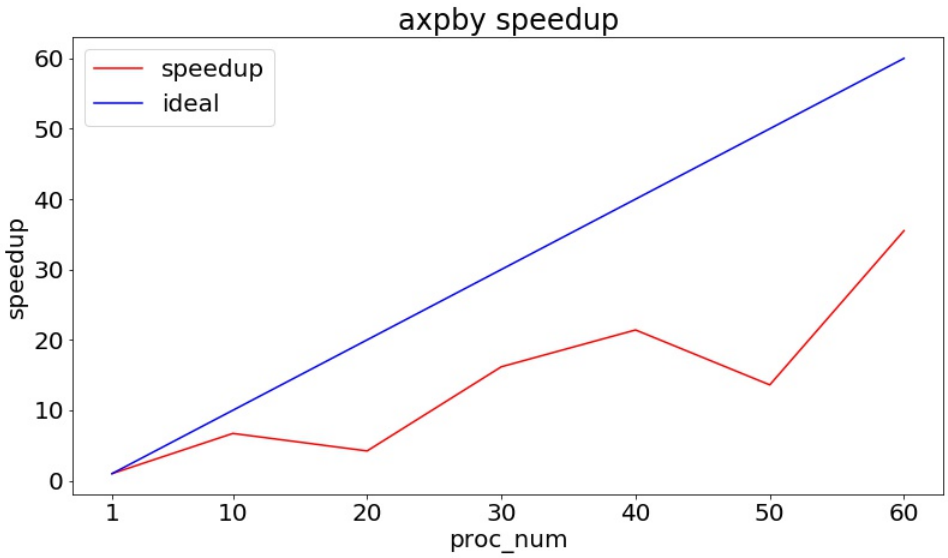

Рис. 3: Ускорение axpby, $N = 10^7$

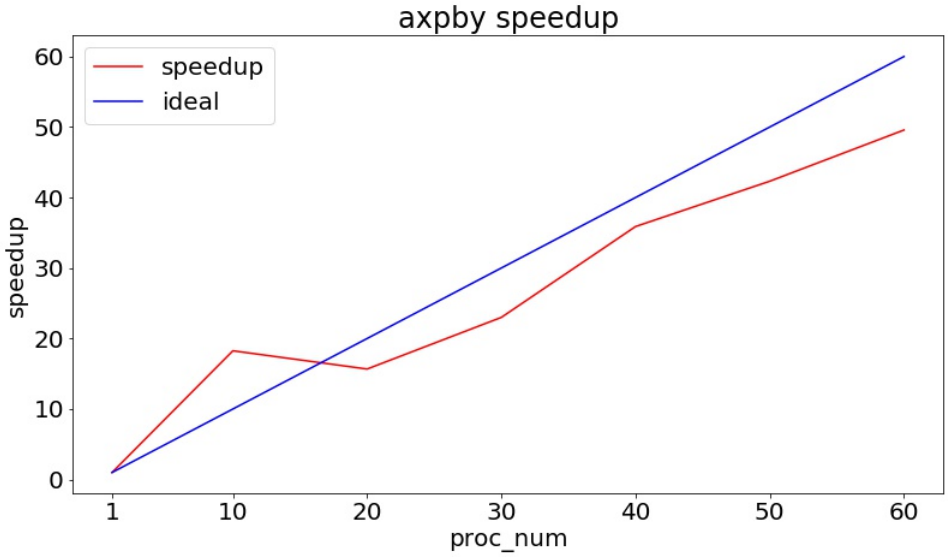

Рис. 4: Ускорение axpby, $N = 10^8$

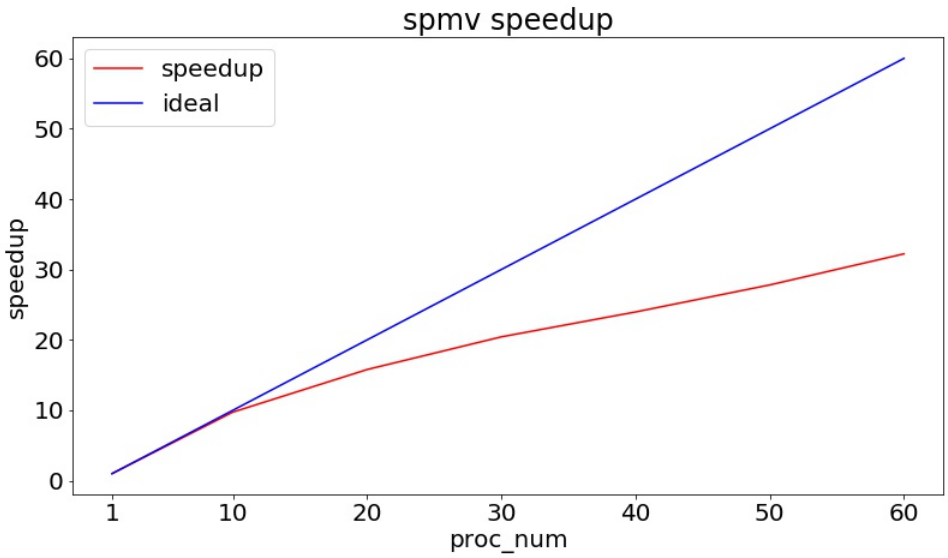

Рис. 5: Ускорение SpMV, $N = 10^7$

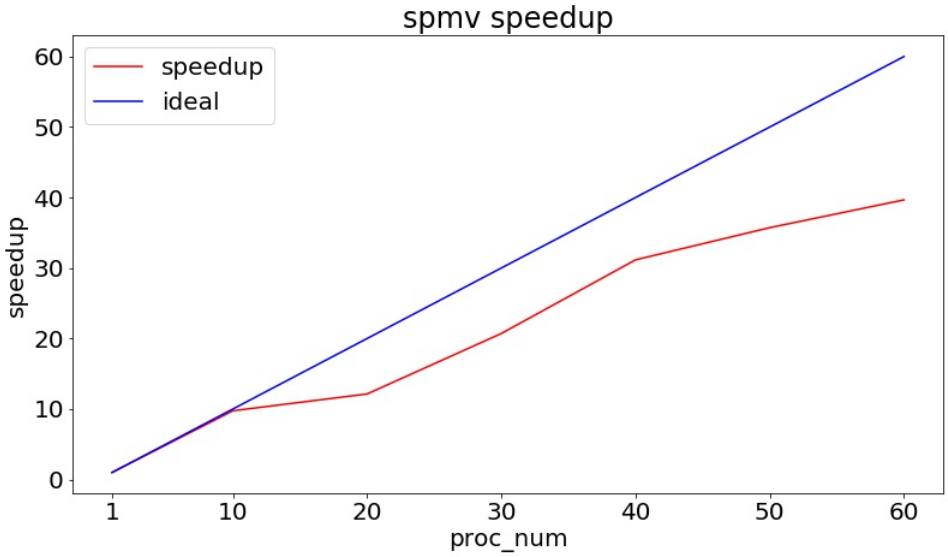

Рис. 6: Ускорение SpMV, $N = 10^8$

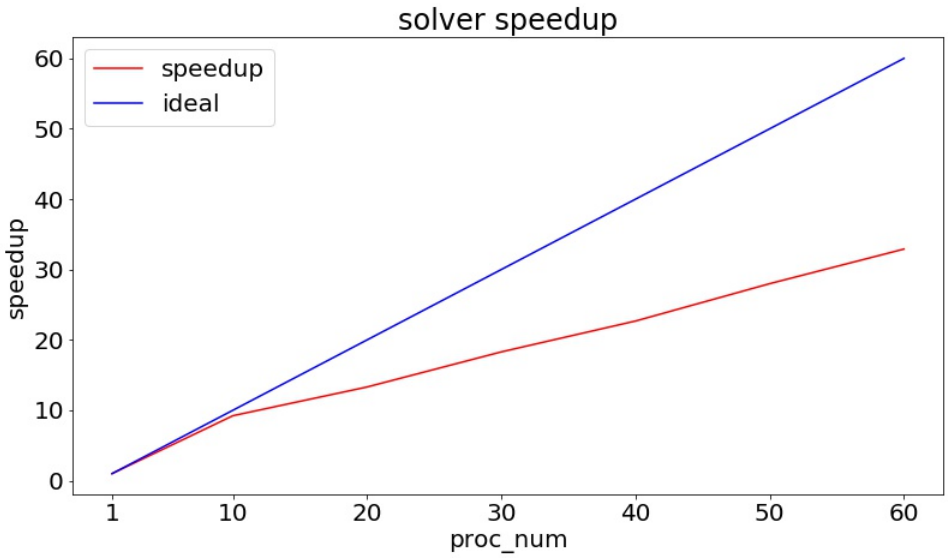

Рис. 7: Ускорение солвера, $N = 10^7$

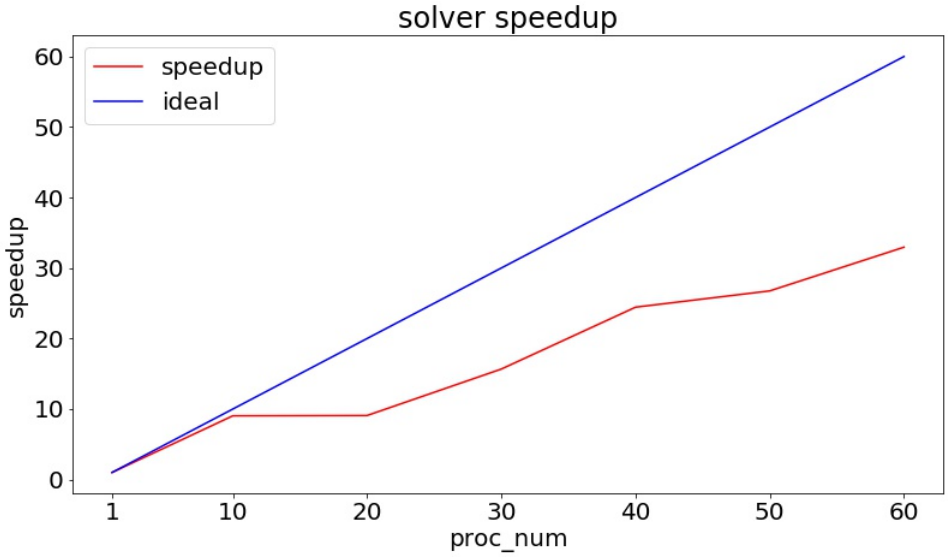

Рис. 8: Ускорение солвера, $N = 10^8$

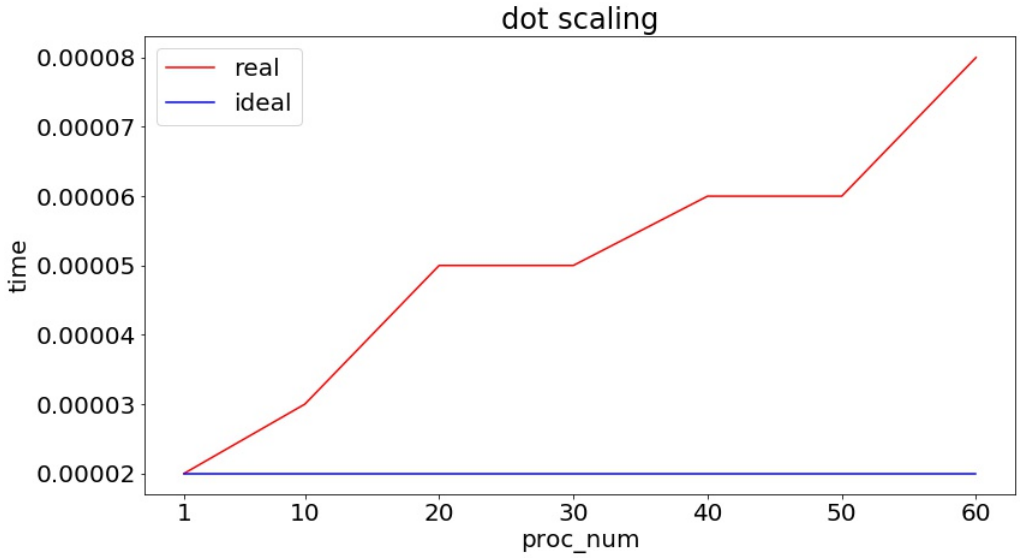

Рис. 9: Масштабирование dot, $N = 10^4 * P$

2.3.3 Масштабирование на кластере

Для измерения масштабирования были проведены запуски на кластере с разным числом MPI процессов, соответствующих одному ядру кластера – $P = 1$, $P = 10$, $P = 20$, $P = 30$, $P = 40$, $P = 50$, $P = 60$. Размерность системы увеличивалась пропорционально количеству процессов – $N = 10^4 * P$, $N = 10^5 * P$, $N = 10^6 * P$.

Поведение результатов меняется при переходе от 20 к 30 процессам и от 40 к 50. Так как один узел кластера включает в себя 20 ядер, а значит 20 MPI процессов, то такое поведение закономерно, потому что у каждого узла отдельное ОЗУ.

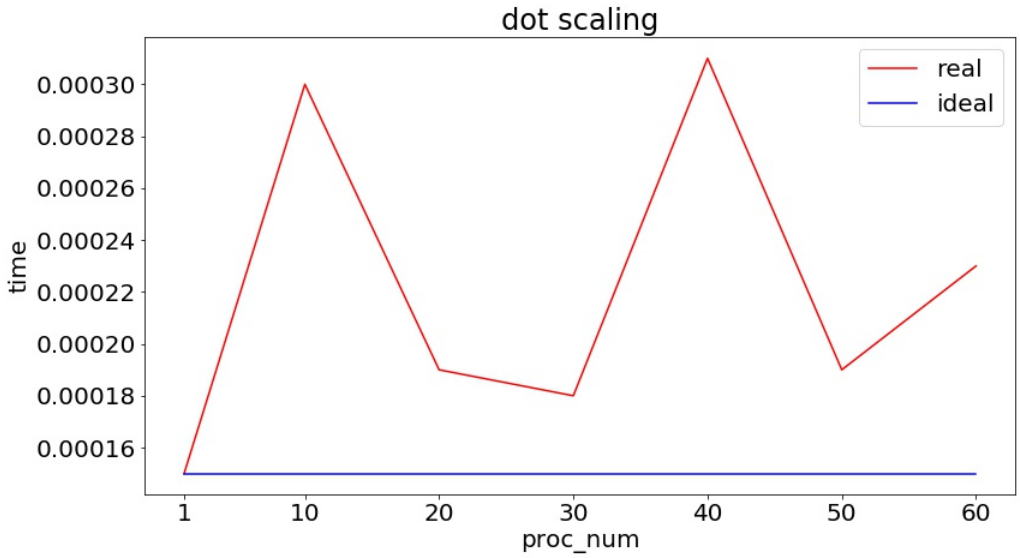

Рис. 10: Масштабирование dot, $N = 10^5 * P$

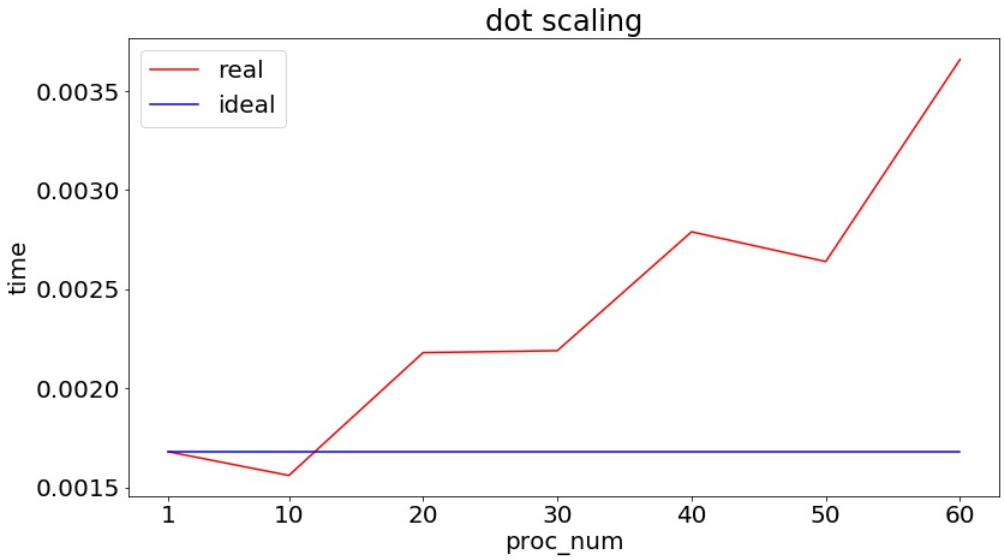

Рис. 11: Масштабирование dot, $N = 10^6 * P$

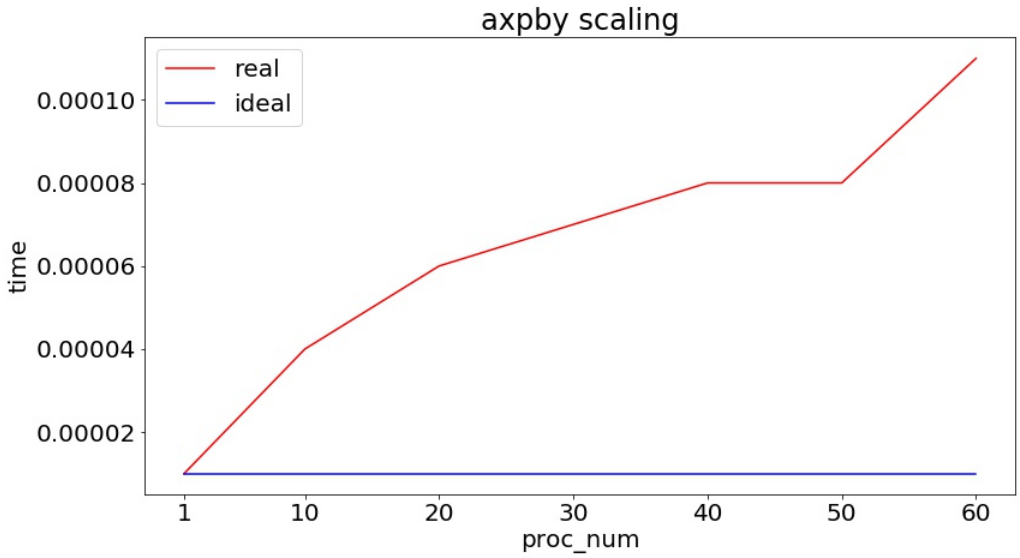

Рис. 12: Масштабирование axpby, $N = 10^4 * P$

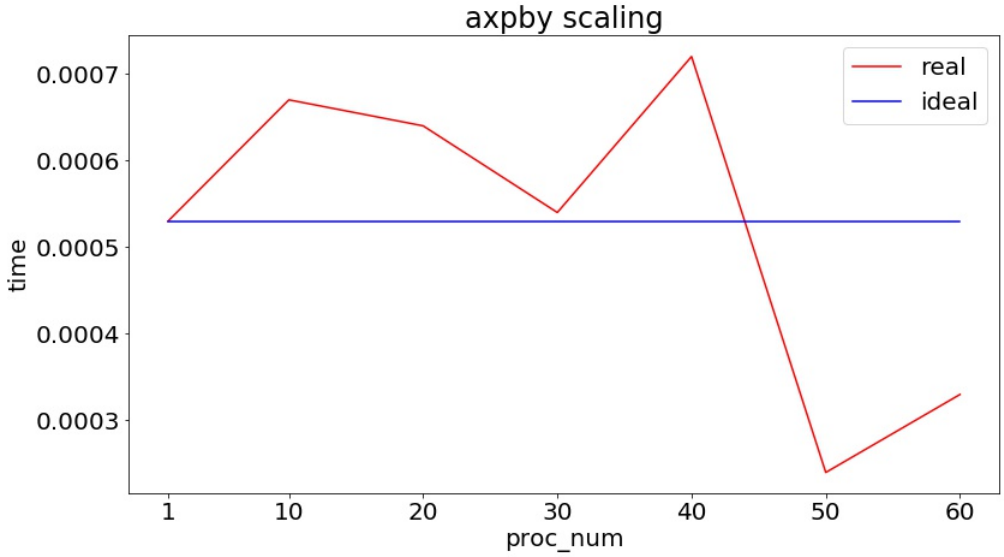

Рис. 13: Масштабирование axpby, $N = 10^5 * P$

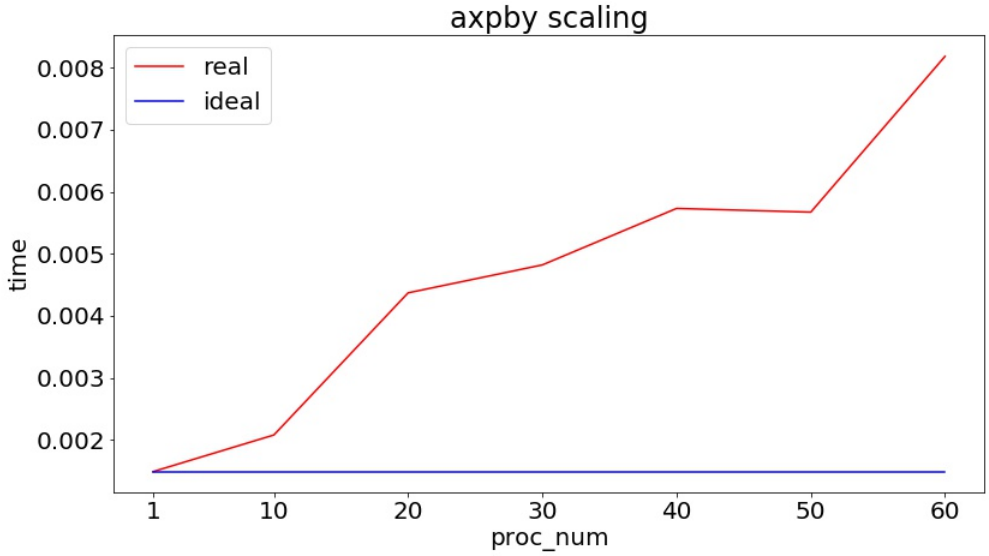

Рис. 14: Масштабирование axpby, $N = 10^6 * P$

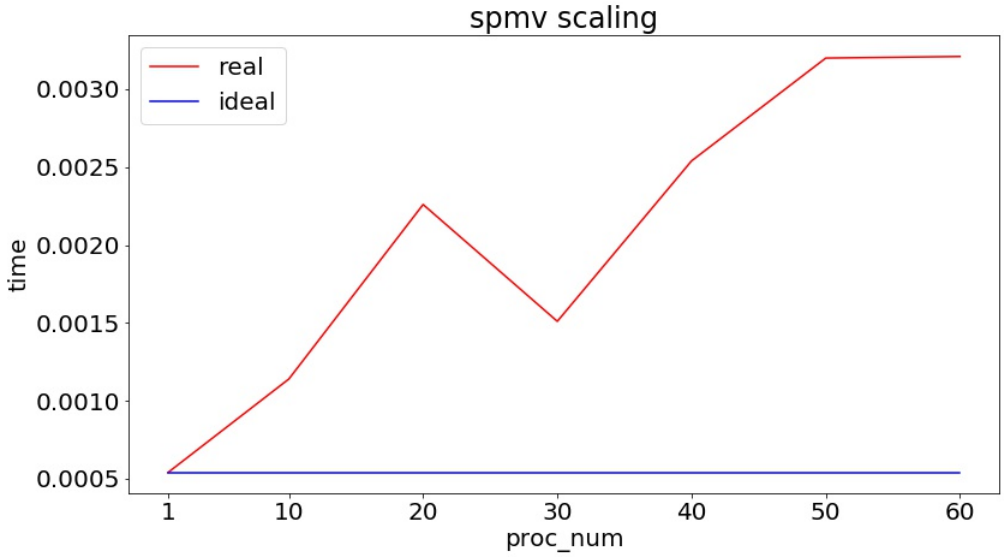

Рис. 15: Масштабирование SpMV, $N = 10^4 * P$

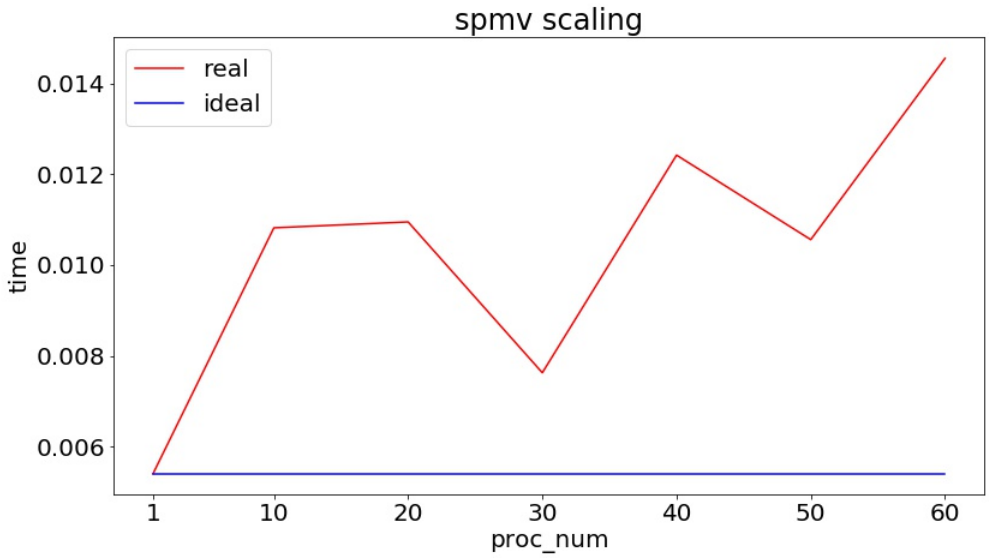

Рис. 16: Масштабирование SpMV, $N = 10^5 * P$

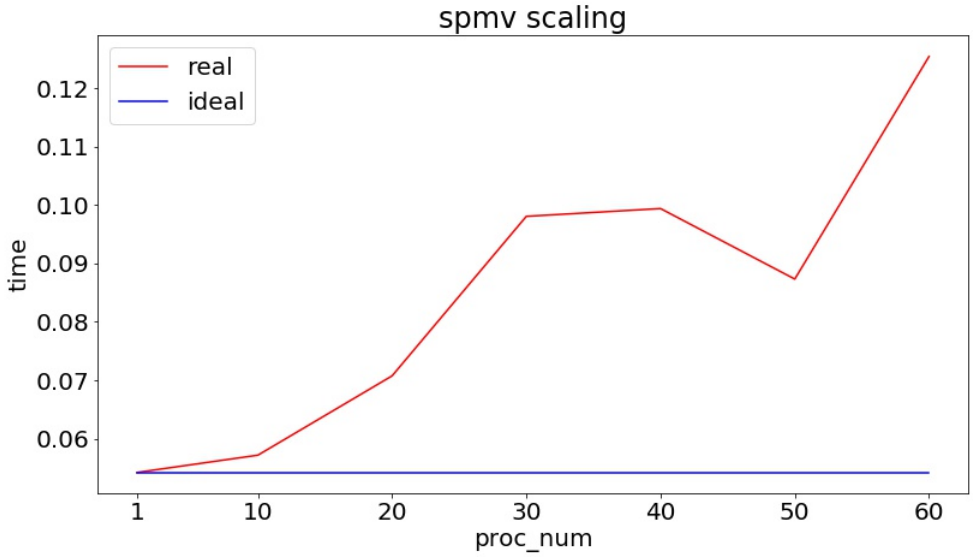

Рис. 17: Масштабирование SpMV, $N = 10^6 * P$

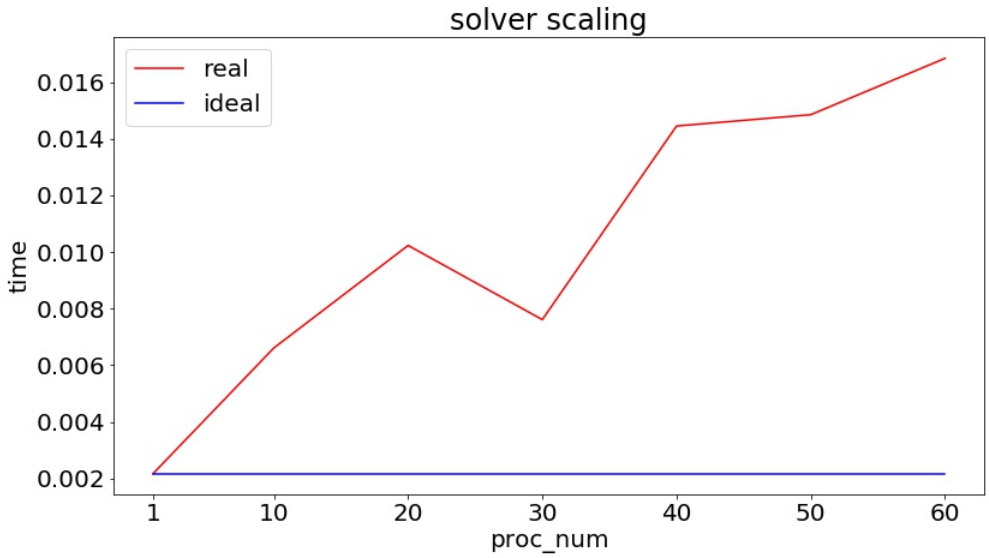

Рис. 18: Масштабирование солвера, $N = 10^4 * P$

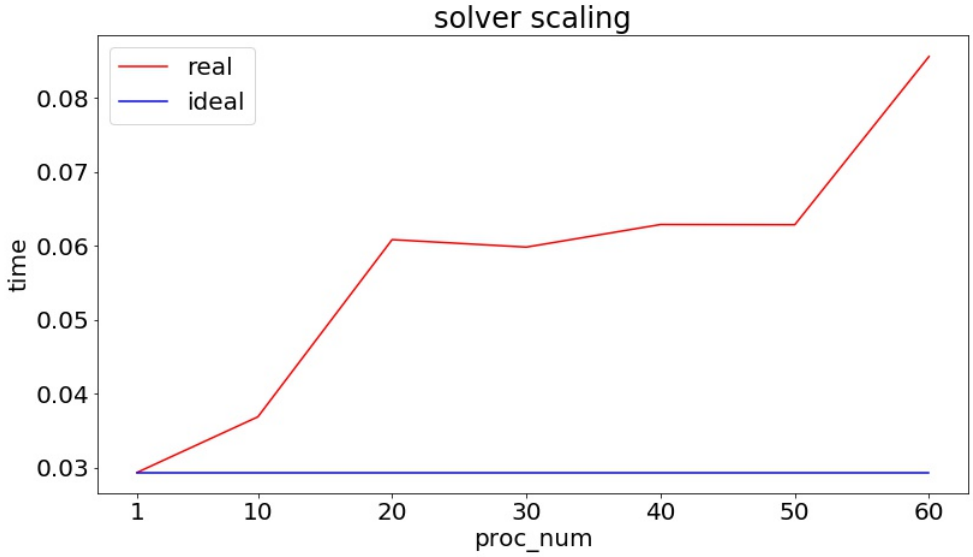

Рис. 19: Масштабирование солвера, $N = 10^5 * P$

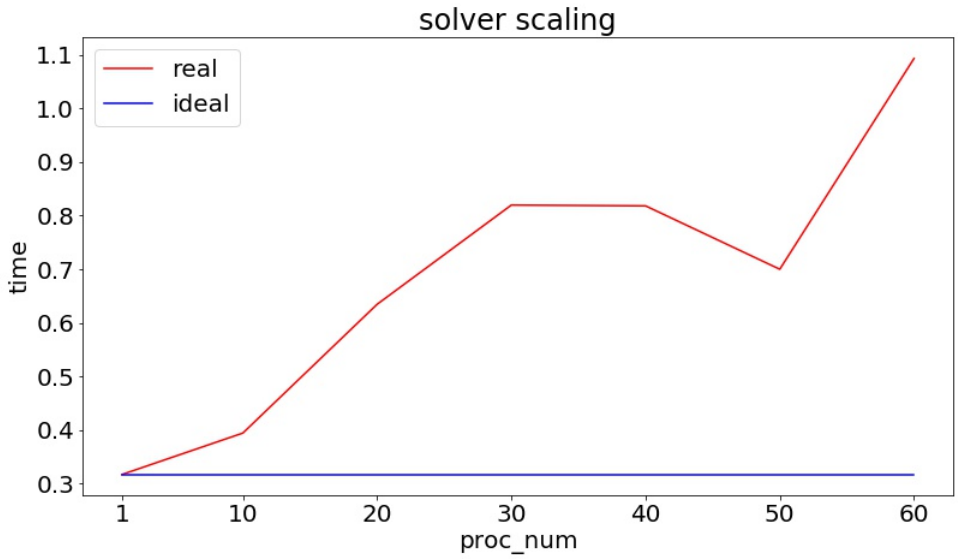

Рис. 20: Масштабирование солвера, $N = 10^6 * P$

	$N = 10^4 * P$	$N = 10^5 * P$	$N = 10^6 * P$
$P = 1$	$2 * 10^{-5}$	$1.5 * 10^{-4}$	0.00168
$P = 10$	$3 * 10^{-5}$	$3 * 10^{-4}$	0.00156
$P = 20$	$5 * 10^{-5}$	$1.9 * 10^{-4}$	0.00218
$P = 30$	$5 * 10^{-5}$	$1.8 * 10^{-4}$	0.00219
$P = 40$	$6 * 10^{-5}$	$3.1 * 10^{-4}$	0.00279
$P = 50$	$6 * 10^{-5}$	$1.9 * 10^{-4}$	0.00264
$P = 60$	$8 * 10^{-5}$	$2.3 * 10^{-4}$	0.00366

Таблица 2: Время выполнения (в секундах) операции dot

	$N = 10^4 * P$	$N = 10^5 * P$	$N = 10^6 * P$
$P = 1$	$1 * 10^{-5}$	$5.3 * 10^{-4}$	0.00149
$P = 10$	$4 * 10^{-5}$	$6.7 * 10^{-4}$	0.00208
$P = 20$	$6 * 10^{-5}$	$6.4 * 10^{-4}$	0.00437
$P = 30$	$7 * 10^{-5}$	$5.4 * 10^{-4}$	0.00482
$P = 40$	$8 * 10^{-5}$	$7.2 * 10^{-4}$	0.00573
$P = 50$	$8 * 10^{-5}$	$2.4 * 10^{-4}$	0.00567
$P = 60$	$1.1 * 10^{-4}$	$3.3 * 10^{-4}$	0.00818

Таблица 3: Время выполнения (в секундах) операции axpby

	$N = 10^4 * P$	$N = 10^5 * P$	$N = 10^6 * P$
$P = 1$	0.00054	0.00541	0.0542
$P = 10$	0.00114	0.01082	0.05716
$P = 20$	0.00226	0.01095	0.07075
$P = 30$	0.00151	0.00763	0.09808
$P = 40$	0.00254	0.01242	0.09941
$P = 50$	0.0032	0.01056	0.08733
$P = 60$	0.00321	0.01455	0.12546

Таблица 4: Время выполнения (в секундах) операции SpMV

	$N = 10^4 * P$	$N = 10^5 * P$	$N = 10^6 * P$
$P = 1$	0.00217	0.02939	0.3172
$P = 10$	0.00661	0.0369	0.39443
$P = 20$	0.01023	0.06083	0.63464
$P = 30$	0.00761	0.05982	0.81976
$P = 40$	0.01445	0.06287	0.8183
$P = 50$	0.01485	0.06284	0.70004
$P = 60$	0.01684	0.08556	1.09319

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.