



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

Трифонов Владислав Дмитриевич

Отчет по заданию 1:
Многопоточная реализация солвера CG для СЛАУ
с разреженной матрицей, заданной в формате ELL

Курс “Параллельные вычисления” (1 курс магистратуры ВМК)

группа 528, дата подачи 13.10.2019

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	4
1.2.3	Реализация	4
2	Исследование производительности	6
2.1	Характеристики вычислительной системы	6
2.1.1	Компиляция	6
2.2	Результаты измерений производительности	6
3	Анализ полученных результатов	11
3.1	Процент от пика	11
3.2	Процент от достижимой производительности	13
4	Заключение	17

1 Описание задания и программной реализации

1.1 Краткое описание задания

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

- реализовать генератор матрицы с диагональным преобладанием для расчетной области, представленной трехмерной декартовой решеткой заданного размера N_x, N_y, N_z
- выполнить последовательные реализации операций $dot, axpby, SpMV$, а также вспомогательные функции для работы с векторами
- реализовать предложенный солвер на основе данных операций
- выполнить многопоточные реализации данных операций с помощью библиотеки OpenMP
- реализовать многопоточный солвер
- 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.

1.2.1 Сборка

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

1.2.2 Запуск

Параметры запуска можно узнать с помощью команды:

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

Пример запуска реализованных солверов с предварительной проверкой операций при размере сетки 5x5x5, 2 потоках, максимальном числе итераций 6, параметром $eps = 0.1$, усреднением по 3 запускам:

```
# ./build/bin/main --qa --nx=5 --ny=5 --nz=5 --nt=2 --maxit=6 --tol=0.1 --nseeds=3
```

1.2.3 Реализация

Для работы с матрицами в формате ELLPACK был реализован модуль *ell_utils.c*, в нем содержатся функции:

- `generate_ELL_3D_DECART` – генерация случайной матрицы с диагональным преобладанием для трехмерной декартовой решетки заданных размеров
- `delete_ELL` – корректное освобождение выделенной памяти

Для работы с векторами был реализован модуль *vector_utils.c*, в нем содержатся функции:

- `create_uninit_Vector` – создание вектора заданной размерности без его инициализации
- `create_const_Vector` – создание вектора заданной размерности с константной инициализацией
- `create_cosine_Vector` – создание вектора заданной размерности с инициализацией $x_i = \cos(i * i)$
- `create_sin_Vector` – создание вектора заданной размерности с инициализацией $x_i = \sin(i * i)$
- `copy_Vector` – создание копии вектора

- `copy_from_Vector_to_Vector` – копирование значений одного вектора в другой. Векторы должны быть одинаковых размеров
- `compute_sum` – подсчет суммы компонент вектора
- `compute_L2_norm` – подсчет L2 нормы
- `compute_L1_norm` – подсчет L1 нормы
- `compute_Linf_norm` – подсчет Linf нормы
- `delete_Vector` – корректное освобождение выделенной памяти

Базовые операции работы с векторами и матрицами выполнены в модуле *ops_utils.c*:

- `dot` – скалярное произведение двух векторов
- `axpby_store` – операция $r\vec{e}s = a * \vec{x} + b * \vec{y}$, вектор результат должен быть указан
- `axpby` – аналогичная операция, для результата аллоцируется новая память
- `SpMV_store` – операция умножения разреженной матрицы на вектор, вектор результат должен быть указан
- `SpMV` – аналогичная операция, для результата аллоцируется новая память
- `inv_diag_SpMV_store` – операция умножения обратной диагональной части разреженной матрицы на вектор, вектор результат должен быть указан
- `inv_diag_SpMV` – аналогичная операция, для результата аллоцируется новая память

Аналогичные многопоточные операции реализованы в модуле *omp_ops_utils.c*.

В модулях *solver.c* и *omp_solver.c* реализованы функции *solve* и *omp_solve* для решения СЛАУ с разреженной матрицей, заданной правой частью, начальным приближением, параметром *eps* и максимальным числом итераций.

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

2 Исследование производительности

2.1 Характеристики 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 *gcc* 5.4.0 с флагами *-g -Wall -O3 -Werror -Wl,-z,defs -Wextra -fopenmp*.

2.2 Результаты измерений производительности

При проведении экспериментов проводилось 3 запуска, далее время усреднялось (*--nseeds=3*). В многопоточном варианте использовалось 2 и 4 потока (*--nt=2, --nt=4*). Параметр *eps = 0.1*, *maxit = 20* (*--tol=0.1 --maxit=20*).

Посчитаем количество FLOP для каждой из операций в зависимости от размера N :

- $FLOP(dot) = N + (N - 1) = 2N - 1$, N умножений и $N - 1$ сложение
- $FLOP(axpby) = 2N + 1N = 3N$, $2N$ умножений и N сложений
- $FLOP(SpMV) = (m + m - 1) * N = 13N$, где m – количество ненулевых элементов матрицы в строке, в предложенной декартовой сетке $m = 7$
- $FLOP(inv_diag_SpMV) = 2N$, N умножений и N делений
- $FLOP(solver) = FLOP(SpMV) + FLOP(axpby) + iter_num * (FLOP(inv_diag_SpMV) + FLOP(dot) + FLOP(axpby) + FLOP(SpMV) + FLOP(dot) + 1 + 2 * FLOP(axpby)) = FLOP(SpMV) + FLOP(axpby) + iter_num * (FLOP(inv_diag_SpMV) + 2 * FLOP(dot) + 3 * FLOP(axpby) + FLOP(SpMV) + 1) = 16N + iter_num * (2N + 4N - 2 + 9N + 13N + 1) = N(16 + iter_num * 28)$, где $iter_num$ – количество проведенных итераций

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
GFLOP	$2 * 10^{-3}$	$2 * 10^{-2}$	$1 * 10^{-1}$	$1.5 * 10^{-1}$
Время последовательной реализации (в секундах) (T_1)	0.001	0.009	0.047	0.070
GFLOPS (последовательная версия)	2.0	2.22	2.12	2.14
Время многопоточной реализации, 2 потока (T_2)	0.001	0.008	0.039	0.058
GFLOPS (2 потока)	2.0	2.5	2.56	2.59
<i>Ускорение (2 потока) (T_1/T_2)</i>	<i>1.0</i>	<i>1.125</i>	<i>1.20</i>	<i>1.21</i>
Время многопоточной реализации, 4 потока (T_4)	0.003	0.009	0.039	0.057
GFLOPS (4 потока)	0.67	2.22	2.56	2.63
<i>Ускорение (4 потока) (T_1/T_4)</i>	<i>0.33</i>	<i>1.0</i>	<i>1.20</i>	<i>1.23</i>
<i>Относительное ускорение (T_2/T_4)</i>	<i>0.33</i>	<i>0.89</i>	<i>1.0</i>	<i>1.01</i>

Таблица 1: Результаты производительности операции dot

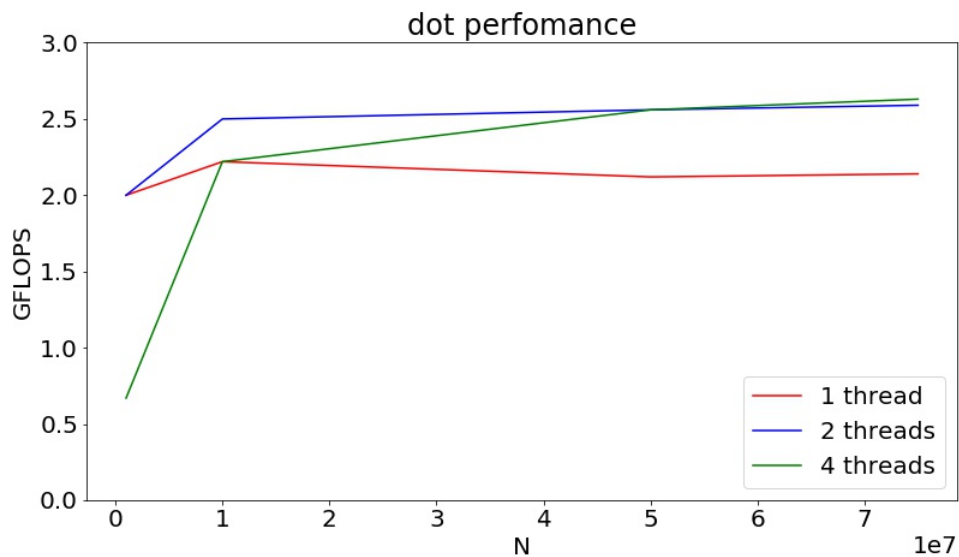


Рис. 1: Графики производительности операции dot

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
GFLOP	$3 * 10^{-3}$	$3 * 10^{-2}$	$1.5 * 10^{-1}$	$2.25 * 10^{-1}$
Время последовательной реализации (в секундах) (T_1)	0.003	0.027	0.135	0.204
GFLOPS (последовательная версия)	1.0	1.11	1.11	1.10
Время многопоточной реализации, 2 потока (T_2)	0.001	0.019	0.084	0.140
GFLOPS (2 потока)	3.0	1.58	1.78	1.61
<i>Ускорение (2 потока) (T_1/T_2)</i>	<i>3.0</i>	<i>1.42</i>	<i>1.61</i>	<i>1.46</i>
Время многопоточной реализации, 4 потока (T_4)	0.002	0.017	0.070	0.101
GFLOPS (4 потока)	2.0	1.76	2.14	2.23
<i>Ускорение (4 потока) (T_1/T_4)</i>	<i>1.5</i>	<i>1.59</i>	<i>1.93</i>	<i>2.0</i>
<i>Относительное ускорение (T_2/T_4)</i>	<i>0.5</i>	<i>1.12</i>	<i>1.2</i>	<i>1.39</i>

Таблица 2: Результаты производительности операции axpby

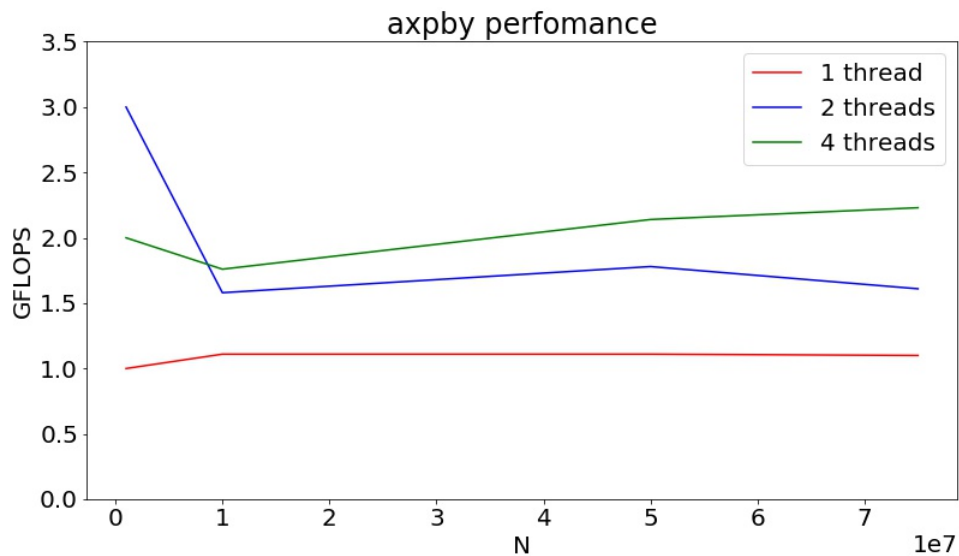


Рис. 2: Графики производительности операции axpby

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
GFLOP	$1.6 * 10^{-2}$	$1.6 * 10^{-1}$	$8 * 10^{-1}$	1.2
Время последовательной реализации (в секундах) (T_1)	0.009	0.107	0.535	0.807
GFLOPS (последовательная версия)	1.77	1.49	1.49	1.49
Время многопоточной реализации, 2 потока (T_2)	0.007	0.076	0.369	0.552
GFLOPS (2 потока)	2.28	2.10	2.17	2.17
<i>Ускорение (2 потока) (T_1/T_2)</i>	<i>1.28</i>	<i>1.4</i>	<i>1.45</i>	<i>1.46</i>
Время многопоточной реализации, 4 потока (T_4)	0.008	0.075	0.341	0.514
GFLOPS (4 потока)	2	2.13	2.35	2.33
<i>Ускорение (4 потока) (T_1/T_4)</i>	<i>1.125</i>	<i>1.43</i>	<i>1.57</i>	<i>1.57</i>
<i>Относительное ускорение (T_2/T_4)</i>	<i>0.875</i>	<i>1.01</i>	<i>1.08</i>	<i>1.07</i>

Таблица 3: Результаты производительности операции SpMV

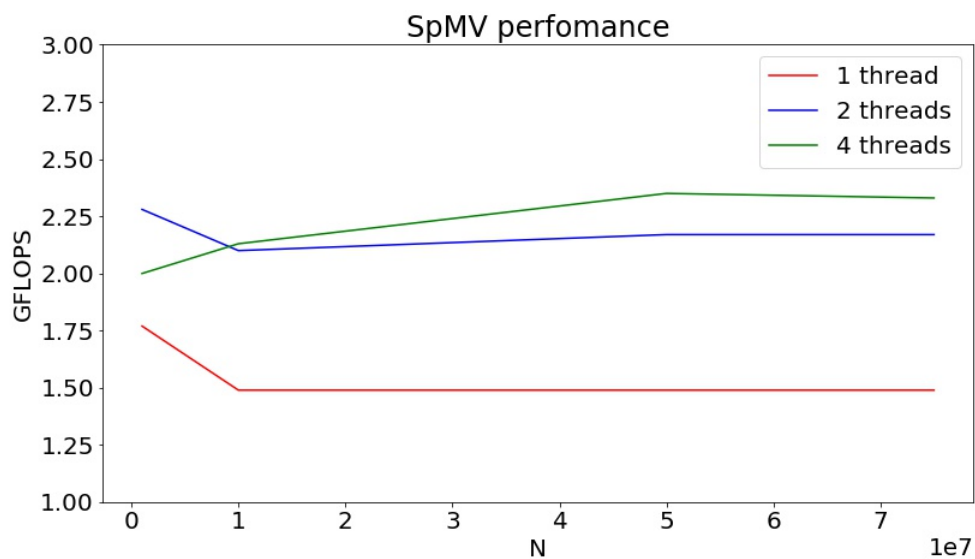


Рис. 3: Графики производительности операции SpMV

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
Количество итераций	7	8	9	9
GFLOP	$2.12 * 10^{-1}$	2.40	$1.34 * 10^1$	$2.01 * 10^1$
Время последовательной реализации (в секундах) (T_1)	0.161	1.963	10.719	15.925
GFLOPS (последовательная версия)	1.32	1.22	1.25	1.26
Время многопоточной реализации, 2 потока (T_2)	0.131	1.531	8.492	12.760
GFLOPS (2 потока)	1.62	1.57	1.58	1.57
<i>Ускорение (2 потока) (T_1/T_2)</i>	<i>1.23</i>	<i>1.28</i>	<i>1.26</i>	<i>1.25</i>
Время многопоточной реализации, 4 потока (T_4)	0.139	1.496	8.151	12.331
GFLOPS (4 потока)	1.52	1.6	1.64	1.63
<i>Ускорение (4 потока) (T_1/T_4)</i>	<i>1.16</i>	<i>1.31</i>	<i>1.31</i>	<i>1.29</i>
<i>Относительное ускорение (T_2/T_4)</i>	<i>0.94</i>	<i>1.02</i>	<i>1.04</i>	<i>1.03</i>

Таблица 4: Результаты производительности солвера

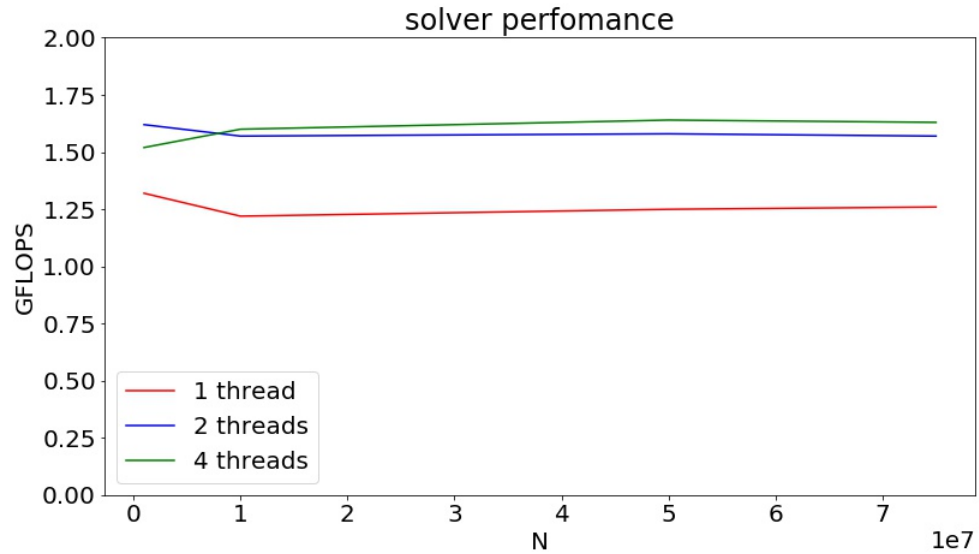


Рис. 4: Графики производительности солвера

3 Анализ полученных результатов

Пиковая производительность $TPP = 131,2$ GFLOPS, пиковая пропускная способность памяти $BW = 25.6$ Gb/s. На одно ядро $TPP_p = 32.8$ GFLOPS.

3.1 Процент от пика

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
GFLOPS (последовательная версия)	2.0	2.22	2.12	2.14
Процент от пика ($GFLOPS/TPP_p$)	6.1%	6.77%	6.46%	6.5%
GFLOPS (2 потока)	2.0	2.5	2.56	2.59
Процент от пика (2 потока) ($GFLOPS/(2 * TPP_p)$)	3.05%	3.81%	3.9%	3.95%
GFLOPS (4 потока)	0.67	2.22	2.56	2.63
Процент от пика (4 потока) ($GFLOPS/(4 * TPP_p)$)	0.51%	1.69%	1.95%	2.00%

Таблица 5: Анализ достигаемой производительности для операции dot

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
GFLOPS (последовательная версия)	1.0	1.11	1.11	1.10
Процент от пика ($GFLOPS/TPP_p$)	3.04%	3.38%	3.38%	3.35%
GFLOPS (2 потока)	3.0	1.58	1.78	1.61
Процент от пика (2 потока) ($GFLOPS/(2 * TPP_p)$)	4.57%	2.4%	2.71%	2.45%
GFLOPS (4 потока)	2.0	1.76	2.14	2.23
Процент от пика (4 потока) ($GFLOPS/(4 * TPP_p)$)	1.52%	1.34%	1.63%	1.70%

Таблица 6: Анализ достигаемой производительности для операции ахрбу

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
GFLOPS (последовательная версия)	1.77	1.49	1.49	1.49
Процент от пика ($GFLOPS/TPP_p$)	5.40%	4.54%	4.54%	4.54%
GFLOPS (2 потока)	2.28	2.10	2.17	2.17
Процент от пика (2 потока) ($GFLOPS/(2 * TPP_p)$)	3.47%	3.20%	3.30%	3.30%
GFLOPS (4 потока)	2	2.13	2.35	2.33
Процент от пика (4 потока) ($GFLOPS/(4 * TPP_p)$)	1.52%	1.62%	1.79%	1.77%

Таблица 7: Анализ достигаемой производительности для операции SpMV

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
GFLOPS (последовательная версия)	1.32	1.22	1.25	1.26
Процент от пика ($GFLOPS/TPP_p$)	4.02%	3.72%	3.81%	3.84%
GFLOPS (2 потока)	1.62	1.57	1.58	1.57
Процент от пика (2 потока) ($GFLOPS/(2 * TPP_p)$)	2.47%	2.39%	2.41%	2.39%
GFLOPS (4 потока)	1.52	1.6	1.64	1.63
Процент от пика (4 потока) ($GFLOPS/(4 * TPP_p)$)	1.16%	1.22%	1.25%	1.24%

Таблица 8: Анализ достигаемой производительности солвера

3.2 Процент от достижимой производительности

Для оценки вычислительной интенсивности посчитаем количество обращений в памяти для каждой из операций, учитывая что $sizeof(double) = 8$ байт:

- $DATA(dot) = 8 * 2N = 16N$ байт, чтение компонент 2 векторов
- $DATA(axpby) = 8 * 3N = 24N$ байт, 2 чтения и 1 запись
- $DATA(SpMV) = 8 * N(3 * m + 1) = 176N$ байт, чтение m номеров столбцов и значений матрицы, m значений вектора для каждой строки матрицы и произвести 1 запись результата
- $DATA(inv_diag_SpMV) = 8 * 3N = 24N$ байт, 2 чтения и 1 запись
- $DATA(solver) = DATA(SpMV) + DATA(axpby) + iter_num * (DATA(inv_diag_SpMV) + DATA(dot) + DATA(axpby) + 1 + DATA(SpMV) + DATA(dot) + 2 * DATA(axpby)) = DATA(SpMV) + DATA(axpby) + iter_num * (DATA(inv_diag_SpMV) + 2 * DATA(dot) + 3 * DATA(axpby) + DATA(SpMV) + 3) = 200N + iter_num * 304 = N(200 + iter_num * 304)$ байт

Теперь вычислим значение вычислительной интенсивности:

- $AI(dot) = \frac{2N - 1}{16N} = \frac{1}{8} = 0.125 \text{ FLOP/байт}$

- $AI(axpby) = \frac{3N}{24N} = \frac{1}{8} = 0.125 \text{ FLOP/байт}$

- $AI(SpMV) = \frac{13N}{176N} = 0.07 \text{ FLOP/байт}$

- $AI(solver) = \frac{16 + 28 * iter_num}{200 + 304 * iter_num} \text{ FLOP/байт}$

Теоретически достижимая производительность $TBP = \min(TPP, BW * AI)$:

- $TBP(dot) = 25.6 * \frac{1}{8} = 3.2 \text{ GFLOPS}$

- $TBP(axpby) = 25.6 * \frac{1}{8} = 3.2 \text{ GFLOPS}$

- $TBP(SpMV) = 25.6 * \frac{13}{176} = 1.9 \text{ GFLOPS}$

- $TBP(solver) = 25.6 * \frac{16 + 28 * iter_num}{200 + 304 * iter_num} \text{ GFLOPS}$

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
TBP в GFLOPS	3.2	3.2	3.2	3.2
GFLOPS (последовательная версия)	2.0	2.22	2.12	2.14
Процент от пика ($GFLOPS/TBP$)	62.5%	69.35%	66.25%	66.85%
GFLOPS (2 потока)	2.0	2.5	2.56	2.59
Процент от пика (2 потока) ($GFLOPS/TBP$)	62.5%	78.1%	80.0%	80.94%
GFLOPS (4 потока)	0.67	2.22	2.56	2.63
Процент от пика (4 потока) ($GFLOPS/TBP$)	20.94%	69.35%	80.0%	82.2%

Таблица 9: Анализ достигаемой производительности операции dot с учетом пропускной способности памяти

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
TBP в GFLOPS	3.2	3.2	3.2	3.2
GFLOPS (последовательная версия)	1.0	1.11	1.11	1.10
Процент от пика ($GFLOPS/TBP$)	31.25%	34.68%	34.68%	34.37%
GFLOPS (2 потока)	3.0	1.58	1.78	1.61
Процент от пика (2 потока) ($GFLOPS/TBP$)	93.1%	49.37%	55.62%	50.31%
GFLOPS (4 потока)	2.0	1.76	2.14	2.23
Процент от пика (4 потока) ($GFLOPS/TBP$)	62.5%	55.5%	66.87%	69.68%

Таблица 10: Анализ достигаемой производительности операции ahrbu с учетом пропускной способности памяти

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
TBP в GFLOPS	1.9	1.9	1.9	1.9
GFLOPS (последовательная версия)	1.77	1.49	1.49	1.49
Процент от пика ($GFLOPS/TBP$)	93.1%	78.4%	78.4%	78.4%
GFLOPS (2 потока)	2.28	2.10	2.17	2.17
Процент от пика (2 потока) ($GFLOPS/TBP$)	120.0%	110.5%	114.2%	114.2%
GFLOPS (4 потока)	2.0	2.13	2.35	2.33
Процент от пика (4 потока) ($GFLOPS/TBP$)	105.2%	112.1%	123.6%	122.6%

Таблица 11: Анализ достигаемой производительности операции SpMV с учетом пропускной способности памяти

Размер системы (N)	10^6	10^7	$5 * 10^7$	$7.5 * 10^7$
Количество итераций	7	8	9	9
TBP в GFLOPS	2.33	2.33	2.33	2.33
GFLOPS (последовательная версия)	1.32	1.22	1.25	1.26
Процент от пика ($GFLOPS/TBP$)	56.65%	52.36%	54.08%	54.08%
GFLOPS (2 потока)	1.62	1.57	1.58	1.57
Процент от пика (2 потока) ($GFLOPS/TBP$)	69.5%	67.38%	67.81%	67.38%
GFLOPS (4 потока)	1.52	1.6	1.64	1.63
Процент от пика (4 потока) ($GFLOPS/TBP$)	65.24%	68.67%	69.96%	69.96%

Таблица 12: Анализ достигаемой производительности солвера с учетом пропускной способности памяти

4 Заключение

В результате практического задания была выполнена последовательная и многопоточная реализация солвера СЛАУ с разреженной матрицей. В ряде экспериментов было установлено, что, несмотря на высокую теоретическую пиковую производительность процессора, фактически достигаемая производительность реализованных алгоритмов сильно зависит от пропускной способности памяти, что ограничивает эффективность многопоточной версии солвера.