Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

ОТЧЕТ О ПРОХОЖДЕНИИ КУРСА

«Параллельное программирование на суперкомпьютерных системах»

Решение СЛАУ с ленточными матрицами

Выполнила
студентка гр.3540201/20301 <подпись> Т.А. Мишарина

Проверил
Преподаватель <подпись> А. А. Лукашин

«___» _____ 2023 г.

Санкт-Петербург
2023

СОДЕРЖАНИЕ

Постановка задачи

1) Проработать реализацию алгоритма, допускающего распараллеливание на несколько потоков / процессов для решения СЛАУ с ленточными матрицами.

2) Разработать тесты для проверки корректности алгоритма (входные данные, выходные данные, код для сравнения результатов).

3) Реализовать алгоритмы с использованием выбранных технологий.

4) Провести исследование эффекта от использования многоядерности / многопоточности / многопроцессности на СКЦ, варьируя узлы от 1 до 4 (для MPI) и варьируя количество процессов / потоков.

5) Подготовить отчет в электронном виде.

1 Алгоритм для параллельного решения СЛАУ с ленточными матрицами

Ленточная матрица – это разреженная матрица, в которой все ненулевые элементы расположены симметрично относительно главной диагонали. Расширенная ленточная матрица размером 5 на 5 и шириной 3 имеет вид:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & b_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & b_3 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & b_4 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & b_5 \end{array}\right)$$

Частный случай ленточной матрицы – трехдиагональная матрица.

Для решения СЛАУ с ленточной матрицей используется метод Гаусса. При прямом ходе метода матрица приводится к треугольному виду, при обратном ходе вычисляется значение последней переменной СЛАУ x_5, которое затем подставляется во все предыдущее уравнения для поиска остальных переменных данной СЛАУ.

На первой итерации метода Гаусса обнуляются все ненулевые элементы первого столбца кроме элемента первой строки и первого столбца. Для данной матрицы будут обнуляться элементы a_{21} и a_{31}, при этом будут изменяться значения элементов второй строки a_{22}, a_{23} и b_2 и третьей строки a_{32}, a_{33} и b_3:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & b_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & b_3 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & b_4 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & b_5 \end{array}\right) \rightarrow \left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ 0 & a_{22}^* & a_{23}^* & a_{24} & 0 & b_2^* \\ 0 & a_{32}^* & a_{33}^* & a_{34} & a_{35} & b_3^* \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & b_4 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & b_5 \end{array}\right)$$

На второй итерации метода обнуляются все ненулевые элементы второго столбца кроме элементов первой строки второго столбца и второй строки второго столбца. Для данной матрицы будут обнуляться элементы a_{32}^*

и a_{42}, при этом будут изменяться значения элементов третьей строки a_{33}^*, a_{34} и b_3^* и четвертой строки a_{43}, a_{44} и b_4:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ 0 & a_{22}^* & a_{23}^* & a_{24} & 0 & b_2^* \\ 0 & a_{32}^* & a_{33}^* & a_{34} & a_{35} & b_3^* \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & b_4 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & b_5 \end{array}\right) \rightarrow \left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ 0 & a_{22}^* & a_{23}^* & a_{24} & 0 & b_2^* \\ 0 & 0 & a_{33}^{**} & a_{34}^{**} & a_{35} & b_3^{**} \\ 0 & 0 & a_{43}^{**} & a_{44}^{**} & a_{45} & b_4^{**} \\ 0 & 0 & a_{53} & a_{54} & a_{55} & b_5 \end{array}\right)$$

После всех итераций прямого хода метода Гаусса получаем треугольную матрицу:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ 0 & a_{22}^* & a_{23}^* & a_{24} & 0 & b_2^* \\ 0 & 0 & a_{33}^{**} & a_{34}^{**} & a_{35} & b_3^{**} \\ 0 & 0 & 0 & a_{44}^{***} & a_{45}^{***} & b_4^{***} \\ 0 & 0 & 0 & 0 & a_{55}^{****} & b_5^{****} \end{array}\right)$$

При обратном ходе вычисляется значение последней переменной СЛАУ x_5, как произведение b_5^{****} на a_{55}^{****}. Это значение затем подставляется во все предыдущее уравнения для поиска остальных переменных данной СЛАУ. Для x_4 значение будет вычисляться следующим образом:

$$x_4 = \frac{b_4^{***} - a_{45}^{***} x_5}{a_{44}^{***}}$$

Таким образом вычисляются все переменные СЛАУ.

Данный алгоритм можно распараллелить. На первой итерации метода Гаусса обнуляются все ненулевые элементы первого столбца, то есть вычисляются новые значения элементов второй и третьей строки. Пусть один поток вычисляет вторую строку, а другой третью. Когда оба потока закончат вычисления, переходим к следующей итерации метода Гаусса, где также распределяем строки между потоками.

Пусть вычисления, выполненные 1 потоком, будут обозначены зеленым цветом, а вычисления, выполненные 2 потоком – синим:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & b_2 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & b_3 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & b_4 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & b_5 \end{array}\right) \rightarrow \left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ 0 & a_{22}^{*} & a_{23}^{*} & a_{24} & 0 & b_2^{*} \\ 0 & a_{32}^{*} & a_{33}^{*} & a_{34} & a_{35} & b_3^{*} \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & b_4 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & b_5 \end{array}\right) \rightarrow$$

$$\rightarrow \left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & 0 & 0 & b_1 \\ 0 & a_{22}^{*} & a_{23}^{*} & a_{24} & 0 & b_2^{*} \\ 0 & 0 & a_{33}^{**} & a_{34}^{**} & a_{35} & b_3^{**} \\ 0 & 0 & a_{43}^{**} & a_{44}^{**} & a_{45} & b_4^{**} \\ 0 & 0 & a_{53} & a_{54} & a_{55} & b_5 \end{array}\right) \rightarrow \ldots$$

Существенный эффект от распараллеливания будет виден на матрицах большего размера (порядка 1000 на 1000), ширина которых составляет 50–70% от общего размера матрицы.

2 Реализация алгоритма для параллельного решения СЛАУ с ленточными матрицами

2.1 Реализация на языке C, с помощью технологии pthreads

Для работы с данной технологией сначала необходимо подключить библиотеку следующим образом:

#include <pthread.h>

Для того, чтобы создать поток необходимо использовать функцию:

pthread_create(…)

Эта функция принимает четыре аргумента: указатель на переменную pthread_t, которая будет содержать идентификатор потока; указатель на переменную pthread_attr_t, которая может использоваться для установки атрибутов потока, таких как размер стека; указатель на функцию, которая будет выполняться потоком; и указатель на любые аргументы, которые необходимо передать функции.

Для того, чтобы дождаться завершения всех потоков перед выходом из программы необходимо использовать функцию:

pthread_join()

2.2 Реализация на языке C, с помощью технологии OpenMP

Open Multi-processing (OpenMP) — это метод распараллеливания разделов кода C/C++/Fortran. OpenMP относится к концепции разделяемой памяти. При этом разные процессоры будут иметь доступ к одной и той же ячейке памяти.

Для работы с данной технологией сначала необходимо подключить библиотеку следующим образом:

#include <omp.h>

Далее, для того чтобы можно было установить количество потоков необходимо добавить функцию:

omp_set_num_threads(p), где p – это количество потоков.

Теперь необходимо выделить части, которые можно распараллелить. Так как распараллеливание идет по строкам матрицы, то перед циклом, который работает со строками необходимо объявить:

#pragma omp parallel for private(jmax,a,j), где jmax – максимальный номер столбца, a – значение элемента до обнуления, j – текущий номер столбца.

2.3 Реализация на языке C, с помощью технологии MPI

MPI — это библиотека интерфейса передачи сообщений, позволяющая выполнять параллельные вычисления путем отправки кода на несколько процессоров.

Для работы с данной технологией сначала необходимо подключить библиотеку следующим образом:

#include <mpi.h>

Далее, необходимо идентифицировать MPI с помощью следующей функции:

MPI_Init(&argc, &argv)

Данная функция создает идентификатор MPI_COMM_WORLD. С его помощью можно получить идентификатор данного процесса:

MPI_Comm_rank(MPI_COMM_WORLD, &rank)

Также с его помощью можно получить общее количество процессов:

MPI_Comm_size(MPI_COMM_WORLD, &size)

Для того, чтобы распределить строки матрицы по разным процессам можно использовать функцию коллективной коммуникации MPI:

MPI_Scatter(…)

Для того, чтобы отправлять обновленные строки другим процессам можно использовать функции:

MPI_Send(…) – отправить,

MPI_Recv(…) – принять.

Затем, чтобы собрать результаты вычислений от всех процессов можно использовать:

MPI_Gather(…)

Чтобы удалить окружение MPI необходимо воспользоваться функцией:

MPI_Finalize()

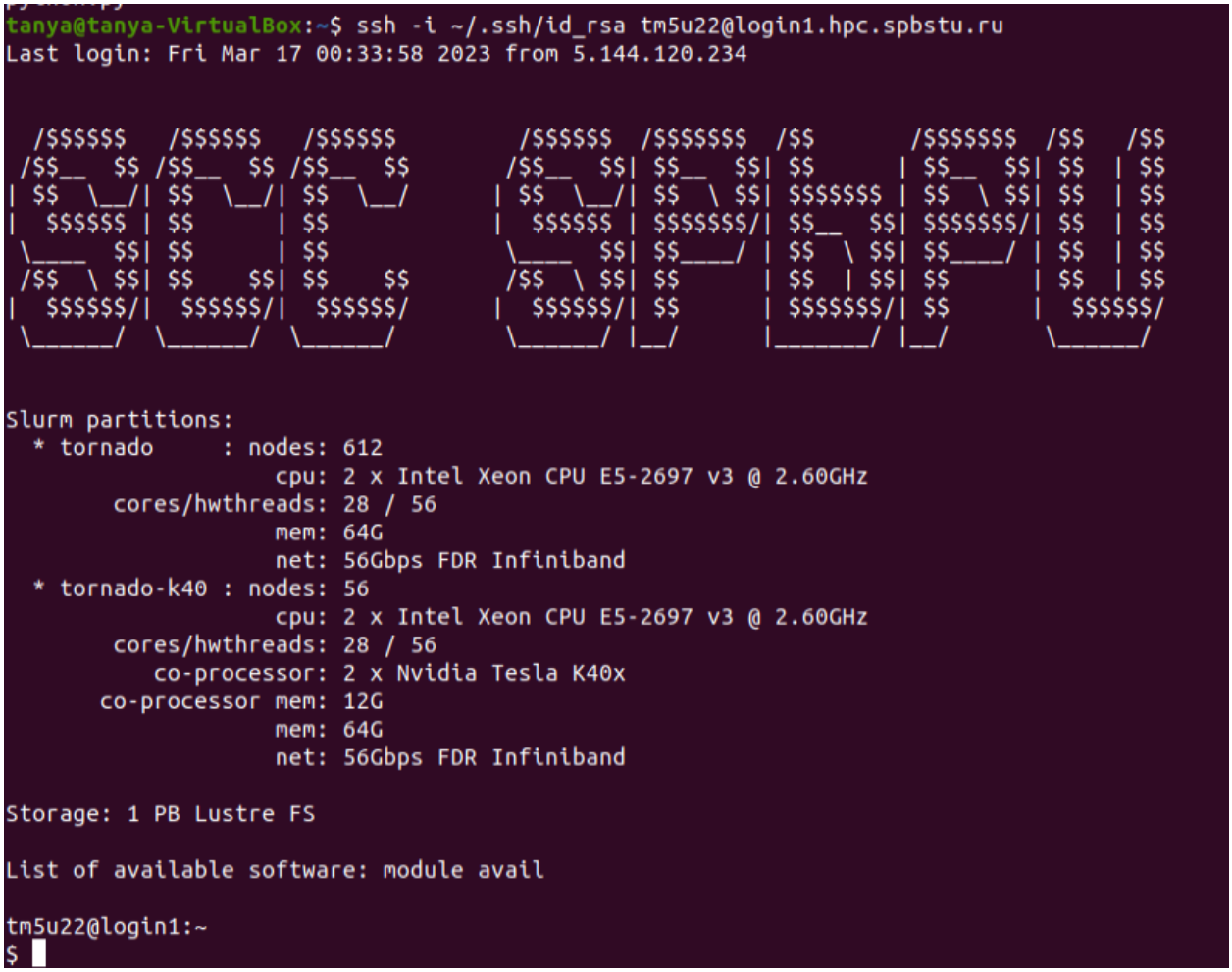
В качестве начальных данных будем использовать квадратную матрицу размером 1000 на 1000 и шириной 500.

2.4 Реализация на языке python, с помощью технологии MPI

Для реализации параллельного алгоритма на языке python с помощью технологии MPI использовались те же функции, что и для параллельного алгоритма на языке C.

3 Запуск алгоритма для параллельного решения СЛАУ с ленточными матрицами на суперкомпьютере

Для того, чтобы запустить программу на суперкомпьютере, сначала необходимо попасть на этот суперкомпьютер. Для этого нужно использовать следующую команду:



Теперь необходимо выделить сервер. Сначала с помощью команды *sinfo* выведем информацию о всех серверах, затем с помощью *salloc* выделим свободный сервер:

```
tm5u22@login1:~
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
cascade      up 14-00:00:0      1  drain n06p011
cascade      up 14-00:00:0     80   idle n06p[001-010,012-081]
g2           up 7-00:00:00      1   resv n01p012
g2           up 7-00:00:00      4   idle n01p[013-016]
tornado*     up 14-00:00:0    346 alloc@ n01p[033-036,038-040,042-065,067,076-
171,180,194-197,200-202,204-205,218-235,237-258,267-269,274,281-282,286-287,32
2-543,569-576,580,589-591,605]
tornado*     up 14-00:00:0     28   resv n01p[001-011,017-032,596]
tornado*     up 14-00:00:0     77  alloc n01p[037,041,066,068-073,081,089,094,
270-273,275,280,283-285,288-291,322,385,388,394-395,409,433-438,468,501,544,55
tornado*     up 14-00:00:0    152   idle n01p[074-075,082-083,110-111,129,131,
321,326-327,390-393,396-397,430-432,439-443,462-467,469-470,484-491,494-500,54
tm5u22@login1:~
$ salloc -N 1 -p cascade
salloc: Pending job allocation 2790123
salloc: job 2790123 queued and waiting for resources
salloc: job 2790123 has been allocated resources
salloc: Granted job allocation 2790123
```

Далее нужно зайти на выделенный сервер. С помощью команды *squeue* узнаем номер нашего сервера и с помощью *ssh* перейдем на него:

```
tm5u22@login1:~
$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           2790123   cascade       sh   tm5u22  R       0:10      1 n06p001
tm5u22@login1:~
$ ssh n06p001
Last login: Mon Mar 20 21:46:46 2023 from 10.10.0.3
```

Для компиляции кода необходимо подгрузить модуль:

```
[tm5u22@n06p053 ~]$ module load compiler/gcc/9
```

Теперь с помощью *gcc* компилируем выходной файл и запускаем его.

Пример работы программы без распараллеливания для матрицы 1000 на 1000 и шириной 500:

```
tm5u22@login1:~/mydir
$ gcc gause.c -o gause
tm5u22@login1:~/mydir
$ ./gause
Time: 1.019027 secs
```

Содержимое файла result.txt, который содержит решение СЛАУ:

```
[tm5u22@n06p053 mydir]$ cat result.txt
x[0] =3.722490
x[1] =-3.667003
x[2] =1.317471
x[3] =2.480734
x[4] =1.644241
x[5] =-4.986271
```

…

```
x[997] =-2.074508
x[998] =5.256492
x[999] =-0.081746
```

Пример работы программы с технологией *pthreads* на 5 потоках для матрицы 1000 на 1000 и шириной 500:

```
[tm5u22@n06p001 mydir]$ gcc pthreads.c -o pthreads
[tm5u22@n06p001 mydir]$ ./pthreads
Time: 0.386758 secs
```

Содержимое файла result_pthreads.txt, который содержит решение СЛАУ:

```
[tm5u22@n06p053 mydir]$ cat result_pthreads.txt
x[0] =3.722490
x[1] =-3.667003
x[2] =1.317471
x[3] =2.480734
x[4] =1.644241
x[5] =-4.986271
```

…

```
x[997] =-2.074508
x[998] =5.256492
x[999] =-0.081746
```

Пример работы программы с технологией *OpenMP* на 5 потоках для матрицы 1000 на 1000 и шириной 500:

```
[tm5u22@n06p001 mydir]$ gcc openmp.c -o openmp
[tm5u22@n06p001 mydir]$ ./openmp
Time: 0.255938 secs
```

Содержимое файла result_openmp.txt, который содержит решение СЛАУ:

```
tm5u22@login1:~/mydir
$ cat result_openmp.txt
x[0] =3.722490
x[1] =-3.667003
x[2] =1.317471
x[3] =2.480734
x[4] =1.644241
x[5] =-4.986271
```

…

```
x[997] =-2.074508
x[998] =5.256492
x[999] =-0.081746
tm5u22@login1:~/mydir
```

Для работы программы с технологией *MPI* на языке C необходимо подгрузить следующий модуль:

```
[tm5u22@n06p001 ~]$ module load mpi/openmpi/4.0.1/gcc/9
[tm5u22@n06p001 ~]$ cd mydir
```

Теперь с помощью *mpicc* компилируем выходной файл и запускаем его с помощью *mpirun*.

Пример работы программы с технологией *MPI* на 5 процессах для матрицы 1000 на 1000 и шириной 500:

```
[tm5u22@n06p001 mydir]$ mpicc mpi.c -o mpi
[tm5u22@n06p001 mydir]$ mpirun -np 5 ./mpi
[n06p001:12578] mca_base_component_repository_open: unable to open mca_btl_openib:
r directory (ignored)
[n06p001:12580] mca_base_component_repository_open: unable to open mca_btl_openib:
r directory (ignored)
[n06p001:12582] mca_base_component_repository_open: unable to open mca_btl_openib:
r directory (ignored)
[n06p001:12581] mca_base_component_repository_open: unable to open mca_btl_openib:
r directory (ignored)
[n06p001:12579] mca_base_component_repository_open: unable to open mca_btl_openib:
r directory (ignored)
Time: 0.194620s
[tm5u22@n06p001 mydir]$
```

Содержимое файла result_mpi.txt, который содержит решение СЛАУ:

```
[tm5u22@n06p053 mydir]$ cat result_mpi.txt
x[0] =1.550784
x[1] =0.605687
x[2] =0.771769
x[3] =2.516819
x[4] =-0.413855
x[5] =-0.970473
```

…

```
x[997] =-2.074508
x[998] =5.256492
x[999] =-0.081746
```

Для работы программы с технологией *MPI* на языке python необходимо подгрузить виртуальное окружение:

```
[tm5u22@n06p015 ~]$ source python-envs/ex/bin/activate
(ex) [tm5u22@n06p015 ~]$ cd mydir
```

Также нужно подгрузить следующий модуль:

```
(ex) [tm5u22@n06p053 mydir]$ module load python/3.8
```

Пример работы программы с технологией *MPI* на языке python на 4 процессах для матрицы 100 на 100 и шириной 50:

```
(ex) [tm5u22@n06p001 mydir]$ mpirun -np 4 python3 python.py
[n06p001:23150] mca_base_component_repository_open: unable to open mca_btl_openib: libosmco
r directory (ignored)
[n06p001:23151] mca_base_component_repository_open: unable to open mca_btl_openib: libosmco
r directory (ignored)
[n06p001:23153] mca_base_component_repository_open: unable to open mca_btl_openib: libosmco
r directory (ignored)
[n06p001:23152] mca_base_component_repository_open: unable to open mca_btl_openib: libosmco
r directory (ignored)
[n06p001:23150] pml_ucx.c:285  Error: UCP worker does not support MPI_THREAD_MULTIPLE
[n06p001:23152] pml_ucx.c:285  Error: UCP worker does not support MPI_THREAD_MULTIPLE
[n06p001:23153] pml_ucx.c:285  Error: UCP worker does not support MPI_THREAD_MULTIPLE
[n06p001:23151] pml_ucx.c:285  Error: UCP worker does not support MPI_THREAD_MULTIPLE
Time: 3.014509
```

Содержимое файла result_ python.txt, который содержит решение СЛАУ:

```
(ex) [tm5u22@n06p001 mydir]$ cat result_python.txt
x[0] =1.550784
x[1] =0.605687
x[2] =0.771769
x[3] =2.516819
x[4] =-0.413855
x[5] =-0.970473
```

…

```
x[97] =-0.133151
x[98] =-0.354409
x[99] =1.168031
```

4 Тестирование алгоритма для параллельного решения СЛАУ с ленточными матрицами

4.1 Зависимость времени работы алгоритма от размера матрицы

Сравним время работы алгоритма при различных размерах матрицы N. При этом ширина матрицы составляет 50% от количества строк (столбцов), а количество потоков процессов равно 5.

N	*pthreads*	*OpenMP*	*MPI (C)*	*MPI (python)*
100	0.032446	0.020198	0.016040	2.991276
500	0.170023	0.112303	0.092767	15.645573
1000	0.386758	0.255938	0.194620	31.567643
5000	2.134014	1.356689	0.997856	210.506170
10000	5.178767	4.967009	2.908766	501.222340

4.2 Зависимость времени работы алгоритма от ширины матрицы

Сравним время работы алгоритма при фиксированном размере матрицы 1000 на 1000 и различных значениях ширины C матрицы (50%, 70%, 90%). Количество потоков процессов равно 5.

C	*pthreads*	*OpenMP*	*MPI (C)*	*MPI (python)*
500	0.386758	0.255938	0.194620	31.567643
700	1.220343	1.543879	0.966231	85.652877
900	4.890960	4.778952	4.560078	125.760442

4.3 Зависимость времени работы алгоритма на C, с технологией pthreads от числа потоков

Сравним время работы алгоритма с технологией *pthreads* при фиксированном размере матрицы 1000 на 1000 с шириной 500 при различном количестве потоков p.

p	*Время, с*
2	0.829090

5	0.386758
10	0.327666
20	0.317801

4.4 Зависимость времени работы алгоритма на C, с технологией OpenMP от числа потоков

Сравним время работы алгоритма с технологией *OpenMP* при фиксированном размере матрицы 1000 на 1000 с шириной 500 при различном количестве потоков p.

p	*Время, с*
2	0.595311
5	0.255938
10	0.176858
20	0.102232

4.5 Зависимость времени работы алгоритма на C, с технологией MPI от числа процессов

Сравним время работы алгоритма с технологией *MPI* на языке C при фиксированном размере матрицы 1000 на 1000 с шириной 500 при различном количестве процессов p.

p	*Время, с*
2	0.456767
5	0.194620
10	0.112934
20	0.086274

4.6 Зависимость времени работы алгоритма на python, с технологией MPI от числа процессов

Сравним время работы алгоритма с технологией *MPI* на языке python при фиксированном размере матрицы 1000 на 1000 с шириной 500 при различном количестве процессов p.

p	*Время, с*
2	46.700425
5	31.567643
10	22.135655
20	19.168434

4.7 Зависимость времени работы алгоритма на C и python, с технологией MPI от числа узлов

Сравним время работы алгоритма с технологией *MPI* на языках C и python при фиксированном размере матрицы 1000 на 1000 с шириной 500 при различном количестве узлов n.

n	*MPI (C)*	*MPI (python)*
1	1.446338	46.288950
2	1.243311	45.363604
3	1.129899	45.157188
4	1.176438	45.265055

Заключение

В ходе проведения данной работы был разработан алгоритм решения СЛАУ с ленточными матрицами на языках C и python. Далее, данный алгоритм был распараллелен с помощью следующих технологий: для C – технологии *pthreads*, *OpenMP* и *MPI*, для python технология *MPI*.

Также был выполнен запуск программ на суперкомпьютере и проведены следующие исследования: исследование зависимости времени работы алгоритма от размера матрицы для каждой технологии распараллеливания, исследование зависимости времени работы алгоритма от ширины матрицы для каждой технологии распараллеливания, исследование зависимости времени работы алгоритма от числа потоков/процессов для каждой технологии распараллеливания, исследование зависимости времени работы алгоритма от числа узлов для технологии *MPI* на языках C и python.

Было установлено, что наиболее эффективной технологией распараллеливания, при решении СЛАУ с ленточными матрицами, является технология *MPI* на языке C. Также было установлено, что с увеличением размера матрицы время работы алгоритма увеличивается, с увеличением ширины матрицы время работы алгоритма также увеличивается, а с увеличением числа потоков/процессов время работы алгоритма уменьшается. При увеличении числа узлов для технологии *MPI* на языках C и python время работы алгоритма изменилось незначительно.

Приложение 1. Код без распараллеливания

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include<sys/time.h>

#define N 1000

const int C = 500;
double A[N][N];
double x[N];
double b[N];
int up = 5;
int down = 1;

double rand_between(const int from, const int to) {
   if (to == from)
      return to;
   if (to < from)
      return rand_between(to, from);
   return from + rand() % (to - from + 1);
}

void createMatrix() {
   //int i, j;
   for (int i = 0; i < N; i++) {
      b[i] = rand_between(down, up);
      for (int j = 0; j < N; j++) {
         if (i - j<C && i - j>-C) {
            A[i][j] = rand_between(down, up);
         }
         else {
```

```
            A[i][j] = 0;
        }
    }
  }
}

int gause() {
  for (int k = 0; k < N - 1; k++) {
    int imin, imax;
    imin = k + 1;
    imax = k + C - 1;
    if (imax > N - 1) { imax = N - 1; }
    for (int i = imin; i <= imax; i++) {
      int jmin, jmax;
      double a;
      jmin = k;
      //jmax = C + (i - k) + k - 1;
      jmax = (2 * C - 1) + (i - C);
      if (jmax > N - 1) { jmax = N - 1; }
      a = A[i][jmin];
      for (int j = jmin; j <= jmax; j++) {
        A[i][j] = A[i][j] - A[k][j] * a / A[k][jmin];
      }
      b[i] = b[i] - b[k] * a / A[k][jmin];
    }
  }
  x[N - 1] = b[N - 1] / A[N - 1][N - 1];
  for (int i = N - 2; i >= 0; i--) {
    int jmin, jmax;
    double sum = 0;
    jmin = i + 1;
    jmax = i + C - 1;
```

```
        if (jmax > N - 1) { jmax = N - 1; }
        for (int j = jmin; j <= jmax; j++) {
            sum = sum + x[j] * A[i][j];
        }
        x[i] = (b[i] - sum) / A[i][i];
    }
    return x[N - 1];
}

int main()
{
float time_use = 0;
struct timeval start_time;
struct timeval end_time;
    createMatrix();
    gettimeofday(&start_time, NULL);
    gause();
    gettimeofday(&end_time, NULL);
    time_use = (end_time.tv_sec - start_time.tv_sec) * 1000000 + (end_time.tv_usec -
start_time.tv_usec);
            FILE* f = fopen("result.txt", "w+");
            for (int j = 0; j < N; j++) {
            fprintf(f,"x[%d] =%f \n", j, x[j]);
            }
    printf("Time: %lf secs\n", time_use / 1000000);
}
```

Приложение 2. Код на языке C, с помощью технологии pthreads

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include<sys/time.h>
#include<pthread.h>

#define N 1000
#define p 10

//using namespace std;

const int C = 500;
double A[N][N];
double x[N];
double b[N];
int up = 5;
int down = 1;
int pthread_t[p];
int imin[p];
int imax[p];

/*
struct param{
	int pthread_t;
	int imin;
	int imax;
};
*/

double rand_between(const int from, const int to) {
   if (to == from)
```

```
    return to;
  if (to < from)
    return rand_between(to, from);
  return from + rand() % (to - from + 1);
}

void createMatrix() {
  //int i, j;
  for (int i = 0; i < N; i++) {
    b[i] = rand_between(down, up);
    for (int j = 0; j < N; j++) {
      if (i - j<C && i - j>-C) {
        A[i][j] = rand_between(down, up);
      }
      else {
        A[i][j] = 0;
      }
    }
  }
}

int gause(imin,imax){
      for (int i = imin; i <= imax; i++) {
      int jmin, jmax;
      double a;
      jmin = k;
      //jmax = C + (i - k) + k - 1;
      jmax = (2 * C - 1) + (i - C);
      if (jmax > N - 1) { jmax = N - 1; }
      a = A[i][jmin];
      for (int j = jmin; j <= jmax; j++) {
        A[i][j] = A[i][j] - A[k][j] * a / A[k][jmin];
```

```
            }
            b[i] = b[i] - b[k] * a / A[k][jmin];
        }
    }

int parall(){
            int size=C/p;
            for (int k = 0; k < N - 1; k++) {

                        for(int t =0;t<p;t++){
                                    pthread_t[t]=t;
                                    imin[t]=k + 1 + t*size;
                                    iman[t]=k + C - 1 + t*size;
                                    if (imax[t] > N - 1) { imax[t] = N - 1; }
                                    pthread_create(&pthread_t[t],NULL,gause,&imin[t],&iman[t]);
                        }
                        for(int t =0;t<p;t++){
                                    pthread_join(&pthread_t[t],NULL);
                        }
            }
            x[N - 1] = b[N - 1] / A[N - 1][N - 1];
    for (int i = N - 2; i >= 0; i--) {
        int jmin, jmax;
        double sum = 0;
        jmin = i + 1;
        jmax = i + C - 1;
        if (jmax > N - 1) { jmax = N - 1; }
        for (int j = jmin; j <= jmax; j++) {
            sum = sum + x[j] * A[i][j];
        }
        x[i] = (b[i] - sum) / A[i][i];
    }
```

```
    return x[N - 1];
}

int main()
{
float time_use = 0;
struct timeval start_time;
struct timeval end_time;
    createMatrix();
    gettimeofday(&start_time, NULL);
    gause();
    gettimeofday(&end_time, NULL);
    time_use = (end_time.tv_sec - start_time.tv_sec) * 1000000 + (end_time.tv_usec -
start_time.tv_usec);

            FILE* f = fopen("result_pthreads.txt", "w+");
            for (int j = 0; j < N; j++) {
            fprintf(f,"x[%d] =%f \n", j, x[j]);
            }

    printf("Time: %lf secs\n", time_use / 1000000);
}
```

Приложение 3. Код на языке C, с помощью технологии OpenMP

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include<sys/time.h>
#include <omp.h>

#define N 1000
#define p 10

//using namespace std;

const int C = 500;
double A[N][N];
double x[N];
double b[N];
int up = 5;
int down = 1;

double rand_between(const int from, const int to) {
   if (to == from)
      return to;
   if (to < from)
      return rand_between(to, from);
   return from + rand() % (to - from + 1);
}

void createMatrix() {
   //int i, j;
   for (int i = 0; i < N; i++) {
      b[i] = rand_between(down, up);
      for (int j = 0; j < N; j++) {
```

```
            if (i - j<C && i - j>-C) {
                A[i][j] = rand_between(down, up);
            }
            else {
                A[i][j] = 0;
            }
        }
    }
}

int gause() {
    for (int k = 0; k < N - 1; k++) {
        int imin, imax;
        imin = k + 1;
        imax = k + C - 1;
        if (imax > N - 1) { imax = N - 1; }
        {
        for (int i = imin; i <= imax; i++) {
            int jmin, jmax;
            double a;
            jmin = k;
            //jmax = C + (i - k) + k - 1;
            jmax = (2 * C - 1) + (i - C);
            if (jmax > N - 1) { jmax = N - 1; }
            a = A[i][jmin];
            #pragma omp parallel for
            for (int j = jmin; j <= jmax; j++) {
                A[i][j] = A[i][j] - A[k][j] * a / A[k][jmin];
            }
            b[i] = b[i] - b[k] * a / A[k][jmin];
            #pragma omp barrier
        }
```

```
        }
    }
    x[N - 1] = b[N - 1] / A[N - 1][N - 1];
    #pragma omp parallel for
    for (int i = N - 2; i >= 0; i--) {
        int jmin, jmax;
        double sum = 0;
        jmin = i + 1;
        jmax = i + C - 1;
        if (jmax > N - 1) { jmax = N - 1; }
        for (int j = jmin; j <= jmax; j++) {
            sum = sum + x[j] * A[i][j];
        }
        x[i] = (b[i] - sum) / A[i][i];
    }
    return x[N - 1];
}

int main()
{
float time_use = 0;
struct timeval start_time;
struct timeval end_time;
    createMatrix();
    omp_set_num_threads(p);
    gettimeofday(&start_time, NULL);
    gause();
    gettimeofday(&end_time, NULL);
    time_use = (end_time.tv_sec - start_time.tv_sec) * 1000000 + (end_time.tv_usec -
start_time.tv_usec);

FILE* f = fopen("result_openmp.txt", "w+");
```

```
            for (int j = 0; j < N; j++) {
            fprintf(f,"x[%d] =%f \n", j, x[j]);
            }
    printf("Time: %lf secs\n", time_use / 1000000);
}
```

Приложение 4. Код на языке C, с помощью технологии MPI

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include<sys/time.h>
#include <mpi.h>

#define N 1000
#define p 10

const int C = 500;
double A[N][N];
double x[N];
double b[N];
int up = 5;
int down = 1;

double rand_between(const int from, const int to) {
   if (to == from)
      return to;
   if (to < from)
      return rand_between(to, from);
   return from + rand() % (to - from + 1);
}

void createMatrix() {
   //int i, j;
   for (int i = 0; i < N; i++) {
      b[i] = rand_between(down, up);
      for (int j = 0; j < N; j++) {
         if (i - j<C && i - j>-C) {
            A[i][j] = rand_between(down, up);
```

```
            }
            else {
                A[i][j] = 0;
            }
        }
    }
}

int gause(int rank, int size) {
    int chunk_size = N / size;
    int start_row = rank * chunk_size;
    int end_row = (rank == size - 1) ? N - 1 : (start_row + chunk_size - 1);

    for (int k = 0; k < N - 1; k++) {
        int imin, imax;
        imin = k + 1;
        imax = k + C - 1;
        if (imax > N - 1) { imax = N - 1; }

        for (int i = imin; i <= imax; i++) {
            int jmin, jmax;
            double a;
            jmin = k;
            jmax = (2 * C - 1) + (i - C);
            if (jmax > N - 1) { jmax = N - 1; }
            a = A[i][jmin];

            for (int j = jmin; j <= jmax; j++) {
                A[i][j] = A[i][j] - A[k][j] * a / A[k][jmin];
            }

            b[i] = b[i] - b[k] * a / A[k][jmin];
```

```
    }
    // Send updated rows to other processes
    for (int p = 0; p < size; p++) {
      if (p != rank) {
        int row_start = p * chunk_size;
        int row_end = (p == size - 1) ? N - 1 : (row_start + chunk_size - 1);
        int row_count = row_end - row_start + 1;

        MPI_Send(&A[start_row], chunk_size * N, MPI_DOUBLE, p, k, MPI_COMM_WORLD);
        MPI_Send(&b[start_row], chunk_size, MPI_DOUBLE, p, k, MPI_COMM_WORLD);

        MPI_Recv(&A[row_start], row_count * N, MPI_DOUBLE, p, k, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        MPI_Recv(&b[row_start], row_count, MPI_DOUBLE, p, k, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
      }
    }
  }
  // Backward substitution
  for (int i = end_row; i >= start_row; i--) {
    int jmin, jmax;
    double sum = 0;
    jmin = i + 1;
    jmax = i + C - 1;
    if (jmax > N - 1) { jmax = N - 1; }

    for (int j = jmin; j <= jmax; j++) {
      sum += x[j] * A[i][j];
    }

    x[i] = (b[i] - sum) / A[i][i];
  }
  // Gather results from all processes
```

```
   MPI_Allgather(MPI_IN_PLACE, chunk_size, MPI_DOUBLE, &x[0], chunk_size, MPI_DOUBLE,
MPI_COMM_WORLD);
   return x[N - 1];
}

int main(int argc, char** argv)
{
int rank, size;
float time_use = 0;
struct timeval start_time;
struct timeval end_time;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

   createMatrix();

   gettimeofday(&start_time, NULL);
   int result = gause(rank, size);
   gettimeofday(&end_time, NULL);
   time_use = (end_time.tv_sec - start_time.tv_sec) * 1000000 + (end_time.tv_usec -
start_time.tv_usec);

   MPI_Finalize();
FILE* f = fopen("result_mpi.txt", "w+");
          for (int j = 0; j < N; j++) {
          fprintf(f,"x[%d] =%f \n", j, x[j]);
          }
   printf("Time: %lf secs\n", time_use / 1000000);
   return 0;
}
```

Приложение 5. Код на языке python, с помощью технологии MPI

```
import numpy as np
import random
import time
from mpi4py import MPI

N = 1000
p = 10
C = 500
up = 5
down = 1

A = np.zeros((N, N))
x = np.zeros(N)
b = np.zeros(N)

def rand_between(from_, to):
	if to == from_:
		return to
	if to < from_:
		return rand_between(to, from_)
		return from_ + random.randint(0, to - from_)

def createMatrix():
	for i in range(N):
		b[i] = rand_between(down, up)
	for j in range(N):
		if i-j<C and i-j>-C:
		A[i][j] = rand_between(down, up)
	else:
		A[i][j] = 0
```

```
def gause(rank, size):
        chunk_size = N // size
        start_row = rank * chunk_size
        end_row = N - 1 if rank == size - 1 else (start_row + chunk_size - 1)

        for k in range(N - 1):
                imin, imax = k + 1, k + C - 1
                imax = N - 1 if imax > N - 1 else imax

        for i in range(imin, imax+1):
                jmin, jmax = k, 2*C-1+(i-C)
                jmax = N - 1 if jmax > N - 1 else jmax
                a = A[i][jmin]

        for j in range(jmin, jmax+1):
                A[i][j] -= A[k][j] * a / A[k][jmin]

                b[i] -= b[k] * a / A[k][jmin]

        for p in range(size):
                if p != rank:
                row_start = p * chunk_size
                row_end = N - 1 if p == size - 1 else (row_start + chunk_size - 1)
                row_count = row_end - row_start + 1

                MPI.COMM_WORLD.Send(A[start_row:start_row+chunk_size], dest=p, tag=k)
                MPI.COMM_WORLD.Send(b[start_row:start_row+chunk_size], dest=p, tag=k)

                MPI.COMM_WORLD.Recv(A[row_start:row_start+row_count], source=p, tag=k)
                MPI.COMM_WORLD.Recv(b[row_start:row_start+row_count], source=p, tag=k)

        for i in range(end_row, start_row-1, -1):
```

```
                jmin, jmax = i+1, i+C-1
                jmax = N - 1 if jmax > N - 1 else jmax
                sum_ = sum(x[j]*A[i][j] for j in range(jmin, jmax+1))
                x[i] = (b[i] - sum_) / A[i][i]

                MPI.COMM_WORLD.Allgather(MPI.IN_PLACE, [x[rank*chunk_size:(rank+1)*chunk_size],
MPI.DOUBLE],
                [x, MPI.DOUBLE])

        return x[N-1]

if __name__ == '__main__':
        comm = MPI.COMM_WORLD
        rank = comm.Get_rank()
        size = comm.Get_size()

        createMatrix()

        start_time = time.time()
        result = gause(rank, size)
        end_time = time.time()

        with open("result_python.txt", "w+") as f:
        for j in range(N):
                f.write(f"x[{j}] = {x[j]}\n")

        print(f"Time: {end_time - start_time} secs")
```