Министерство образования и науки РФ
Санкт-Петербургский Политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта

КУРСОВОЙ ПРОЕКТ

Тема: «Метод сопряженных градиентов»
по дисциплине «Параллельное программирование на суперкомпьютерных системах»

Выполнил:
студент гр. 5140201/30301 _____ Семкин Д.Е.
подпись, дата

Проверил:
Доцент, к.т.н. _____ Лукашин А.А.
подпись, дата

Санкт-Петербург
2024

Оглавление

Введение

В этом задании реализован алгоритм метода сопряженных градиентов на суперкомпьютере. Алгоритм метода сопряженных градиентов распараллеливается следующим методом.

- C & Linux pthreads
- C & MPI
- Python & MPI
- C & OpenMP

Наконец, сравниваются и анализируются результаты реализации метода сопряженных градиентов различными методами.

1 Математическое описание метода сопряженных градиентов

Метод сопряженных градиентов — численный метод решения систем линейных алгебраических уравнений, является итерационным методом Крыловского типа.

Пусть дана система линейных уравнений: $A_x = b$. Причём матрица системы — это действительная матрица, обладающая следующими свойствами: $A = A^T > 0$, то есть это симметричная положительно определённая матрица. Тогда процесс решения СЛАУ можно представить как минимизацию следующего функционала:

$$(Ax, x) - 2(b, x) \to min$$

Шаги алгоритма метода сопряженных градиентов показаны вниз на Рисунке 1:

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat
$\alpha_k := \dfrac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
if r_{k+1} is sufficiently small, then exit loop
$\beta_k := \dfrac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$
$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
$k := k + 1$
end repeat

Рисунок 1 – Алгоритм метода сопряженных градиентов

Поскольку минимизируемый функционал квадратичный, то процесс должен дать ответ на n-й итерации, однако при реализации метода на компьютере существует погрешность представления вещественных чисел, в

результате чего может потребоваться и больше итераций. Так же оценивать точность можно по относительной невязке; $\frac{\|r^k\|}{\|b\|}$, и прекращать итерационный процесс, когда невязка станет меньше заданного числа.

2 Суперкомпьютерный центр «Политехнический»

2.1 Состав и характеристики

В качестве вычислительного ресурса для решения поставленной задачи используется суперкомпьютерный комплекс Санкт-Петербургского Политехнического Университета. В вычислительном центре представлены три суперкомпьютера:

• «Политехник - РСК Торнадо» - кластер с пиковой производительностью 943 Тфлопс; содержит 612 узлов с прямым жидкостным охлаждением серии "Торнадо"(производитель "РСК Технологии РФ), имеющие каждый два CPU Intel Xeon E5-2697 v3 (14 ядер, 2.6 ГГц) и 64 ГБ оперативной памяти DDR4 и 56 узлов с прямым жидкостным охлаждением серии "Tornado содержащие каждый два CPU Intel Xeon E5-2697 v3 и два ускорителя вычислений NVIDIA Tesla K40X, 64 ГБ оперативной памяти DDR4.

• «Политехник - РСК ПетаСтрим» - массивно-параллельный компьютер с ультравысокой многопоточностью; единственная в России система, способная поддержать более 70 тыс. потоков; пиковая производительность 291 Тфлопс; содержит 288 узлов на сопроцессорах Intel Xeon Phi;

• «Политехник - NUMA» - массивно-параллельная система с кеш-когерентной глобально адресуемой памятью объемом более 12 ТБ; содержит 64 узла, 192 процессора; пиковая производительность 30 ТФлопс. Все вычислительные системы работают с общей системой хранения данных (Seagate ClusterStore, 1.1ПБ), имеют единую систему управления и мониторинга.

2.2 Технология подключения

Подключение к суперкомпьютеру возможно при наличии логина и ключа доступа, который выдаётся сотрудниками суперкомпьютерного центра. С помощью этого ключа и производится доступ к суперкомпьютеру вместо использования пароля. В данном задании я использовал подключение через PuTTy. Также возможно подключение через VScode, так как он предоставляет хорошую функцию редактирования.

Сначала я отправил свою папку с файлами на сервер суперкомпьютера.

```
PS C:\Users\Dima\Desktop\prog1> pscp -i C:\Users\Dima\Documents\kluch1.ppk -r C:\Users\Dima\Desktop\prog1 tm3u17@login1.hpc.spbstu.ru:
```

Рисунок 2 – Перенос файлов на сервер

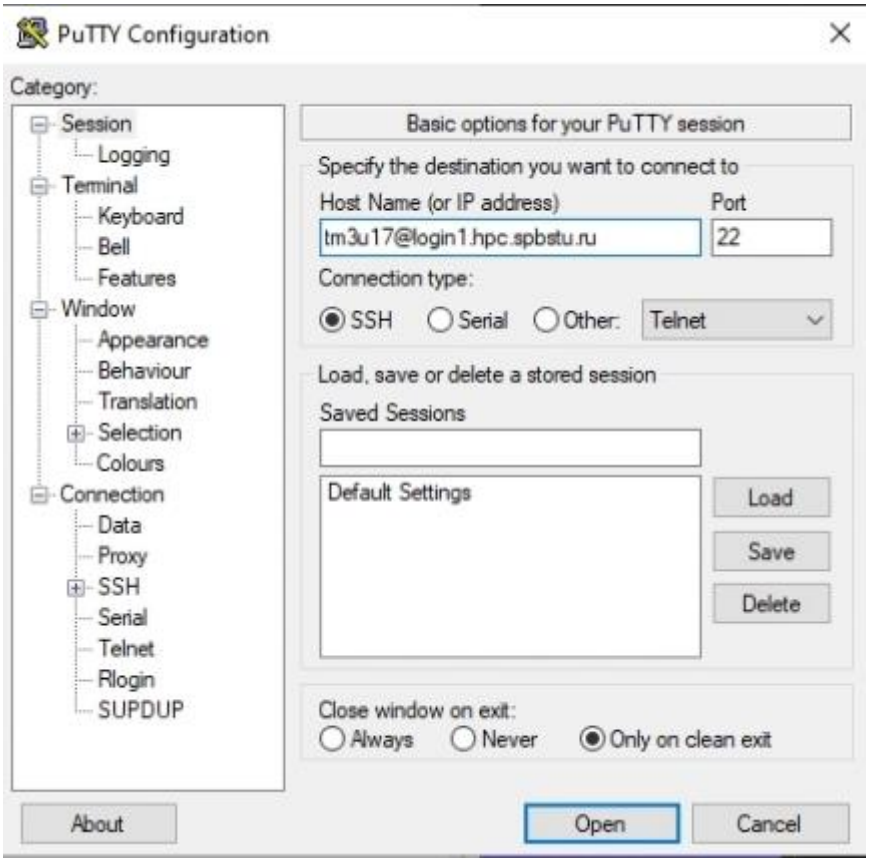

Рисунок 3 – Ввод логина и пароля

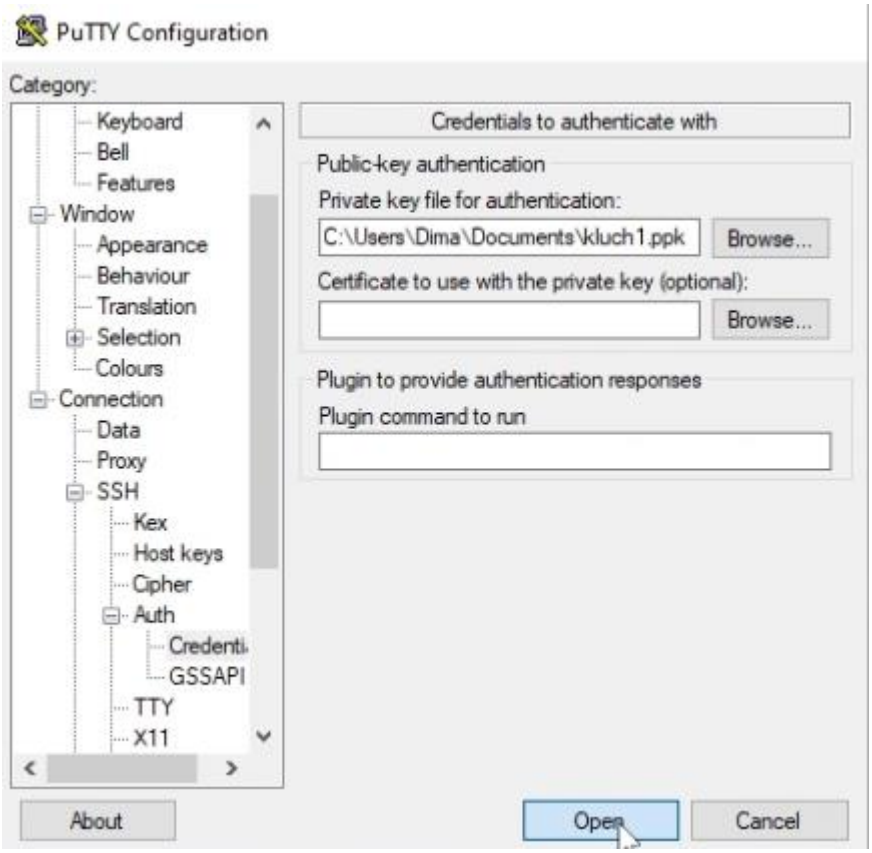

Рисунок 4 – Ввод логина и пароля

После ввода логина и пароля, мы открываем консоль суперкомпьютера

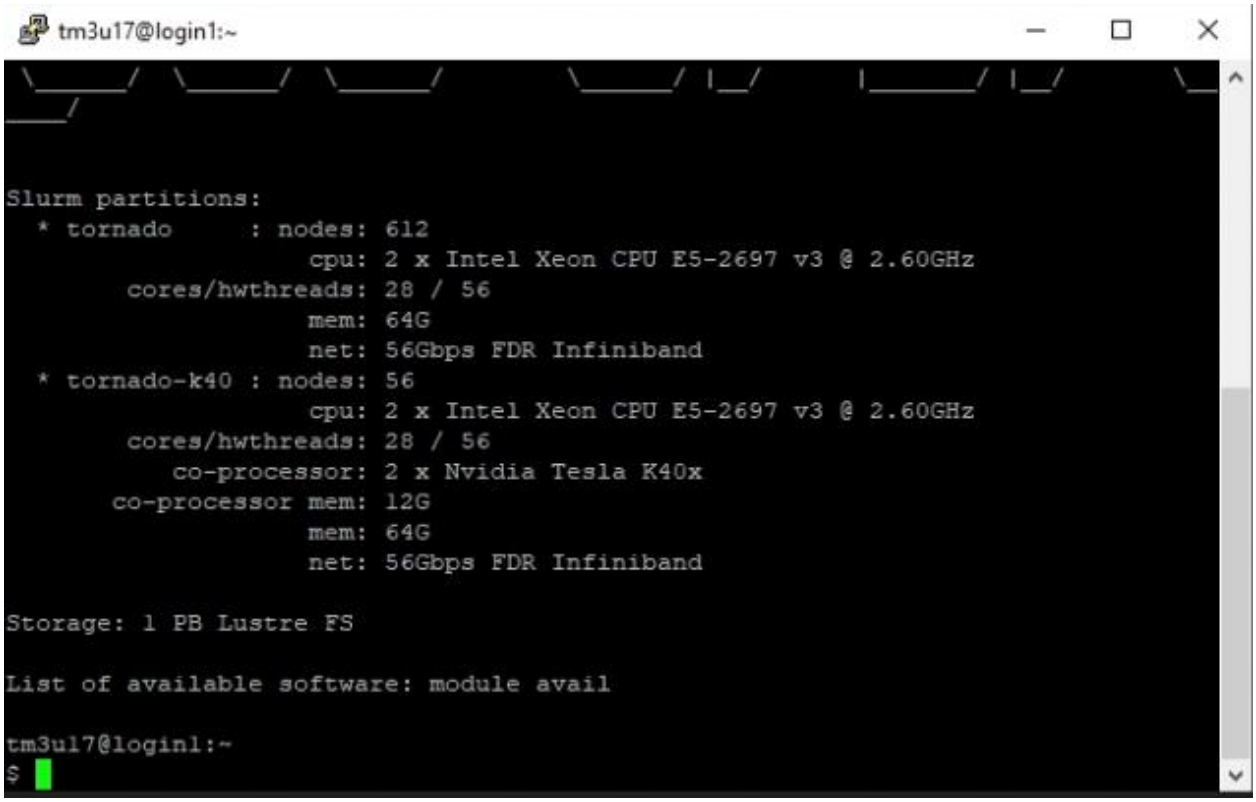

Рисунок 5 – Успешно подключение

2.3 Настройка окружения

Для управления окружением на ресурсах СКЦ «Политехнический» используется система так называемых Envirement Modules. С помощью них модифицируется окружение пользователя. Список доступных модулей можно посмотреть так:

```
$ module avail

------------------------------ /usr/share/Modules/modulefiles ------------------------------
dot         module-git  module-info modules     null        use.own

------------------------------------ /opt/basis/modules ------------------------------------
abinit/8.10.3                       library/fftw/3.3.5/gcc
ansys/apdl/16.2                     library/fftw/3.3.5/icc
ansys/apdl/17.2                     library/fftw/3.3.6/gcc
ansys/apdl/18.2                     library/fftw/3.3.6/gcc_threaded
ansys/apdl/19.2                     library/fltk/1.3.4/gcc
ansys/apdl/2019r3                   library/gflags/2.2.1
ansys/apdl/2020r2                   library/glog/0.3.4
ansys/apdl/2021r2                   library/gsl/2.5.0
ansys/apdl/2022r1                   library/hdf5/1.10.1/gcc
ansys/apdl/2022r2                   library/hdf5/1.10.1/gcc72
ansys/cfx/16.2                      library/hdf5/1.10.1/icc
ansys/cfx/17.2                      library/hdf5/1.10.5/gcc74
ansys/cfx/18.0                      library/hdf5/1.8.13/intel
ansys/cfx/18.2                      library/hdf5/1.8.14/gcc
```

Рисунок 6 – Список модулей

Таким образом можно выбрать наборы компиляторов, библиотек, а также предустановленного программного обеспечения. Например, как для файла pmpi.py мы используем python/3.10.

Загрузка требуемых модулей осуществляется с помощью команды module load.

module load compiler/gcc/6.2.0

2.4 Запуск задач

Управление ресурсами кластера осуществляется с помощью программного пакета SLURM. Принцип его работы можно описать следующим образом: пользователь запрашивает некоторый ресурс (процессорные ядра, память и т. п.), размещая свою задачу в очереди; система, основываясь на приоритетах пользователя и текущем заполнении очереди, выбирает момент запуска задачи. Под очередью понимается

последовательность задач, которая должны решаться на определенном вычислительном ресурсе (группе узлов).

При этом каждый узел в текущий момент времени может быть занят только одной задачей, одного пользователя; таким образом узел отводится в монопольное использование размещенной на нем задаче, т. е. другие задачи на занятом узле выполнятся не будут.

Пример скрипта sbatch для компиляции и выполнения файла:

```
cmpi > $ run.sl
  1  #!/bin/bash
  2  #SBATCH --nodes=1
  3  #SBATCH -p tornado-k40
  4  #SBATCH -t 00-02:00:00
  5  #SBATCH -J Ren_cgm
  6  #SBATSH -o ./result.txt
  7  #SBATSH -e ./error.txt
  8  if [ -f /etc/profile.d/modules-basis.sh ];
  9  then
 10  source /etc/profile.d/modules-basis.sh
 11  fi
 12  module purge
 13  module load compiler/gcc/9
 14  module add mpi/openmpi/4.0.1/gcc/9
 15
 16  mpicc cmpi.c -o cmpi
 17  mpiexec -n 20 ./cmpi
```

Рисунок 7 – Запуск задач

3 Распараллеливание алгоритма метода сопряженных градиентов

Видно, из рисунка 1. Алгоритм метода сопряженного градиента содержит большое количество повторяющихся операций между векторами и матрицами. Следовательно, общий алгоритм метода градиента можно оптимизировать для ускорения за счет распараллеливания этих матричных и векторных операций. Среди этих операций между векторами и матрицами умножение векторов и матриц является наиболее сложным (алгоритм не

включает умножение между матрицами). Поэтому в этом задании мы в основном изучаем влияние различных методов на результаты распараллеливания путем распараллеливания умножения между матрицами и векторами. Когда вектор умножается на матрицу, результатом является вектор. Чтобы распараллелить эту операцию, можно заставить каждый поток вычислять часть элементов результирующего вектора. Вычисления с несколькими потоками могут ускорить вычисления. Код выглядит следующим образом:

```
void MultMatrix(int id,int numprocs,struct Matrix mul1,struct Matrix mul2,struct Matrix prod){
    resetMatrix(prod);
    int i,j,k,X,Y;
    long num=dim/numprocs+1;
    long indexProd=id*num;
    for(i=0;i<num;i++){
        if(indexProd+i==prod.x*prod.y)
            break;
        X=(indexProd+i)/prod.y;
        Y=(indexProd+i)%prod.y;
        double sum=0;
        for(j=0;j<mul1.y;j++){
            sum+=mul1.m[X*mul1.y+j]*mul2.m[j*mul2.y+Y];
        }
        prod.m[indexProd+i]=sum;
    }
```

Рисунок 8 – Умножение матрицы на вектор

4 Исследование результата

4.1 Проверки корректности алгоритма

Поскольку алгоритм метода сопряженного градиента требует, чтобы матрица коэффициентов A была положительно определенной матрицей. Поэтому в данной работе реализован алгоритм построения положительно определенных матриц.

```
void*positive_definite_matrix(double*m){
    int i, j;
    double sum = 1;
    for (i = 0;i < dim;i++) {
        for (j = 0;j < dim;j++) {
            if (j < i)
                m[i * dim + j] = 0;
            else
                m[i * dim + j] = sum++;
        }
    }
}
```

Рисунок 9 – Алгоритм построения положительно определенных матриц

Положительно определенные матрицы, сгенерированные этим алгоритмом, имеют следующий вид.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

Первый элемент матрицы равен единице, а затем постепенно увеличивается на единицу. Элементы ниже диагонали матрицы равны 0. Из определения положительно определенной матрицы мы знаем, что этот тип матрицы является положительно определенной.

В этом задании предполагается, что все элементы вектора B равны 10, а элементы исходного вектора решения X равны 0. В силу особенностей матрицы коэффициентов A. Последние несколько элементов вектора решения X могут быть легко получены по теореме Гаусса. Поскольку размер матрицы X слишком велик, в этой работе последние две цифры вектора X находятся с помощью теоремы Гаусса. Затем сравнить его с двумя последними элементами вектора X, рассчитанными алгоритмом сопряженного градиента, чтобы проверить точность алгоритма.

При размерах вектора 10, 50 и 100 максимальное количество итераций не превышает 20, и результаты показаны ниже.

```
Количество итерации:20
Метод Гаусса:x(n):0.181818,x(n-1)=0.003431
Метод сопряженных градиентов:x(n)=0.181858,x(n-1)=0.002904
Размерность матрицы:10
Количество потоков:1
Время:0.001469s
```

Рисунок 10 – Размер 10

```
Количество итерации:20
Метод Гаусса:x(n):0.007843,x(n-1)=0.000006
Метод сопряженных градиентов:x(n)=0.007843,x(n-1)=0.000007
Размерность матрицы:50
Количество потоков:1
Время:0.003175s
```

Рисунок 11 – Размер 50

```
Количество итерации:20
Метод Гаусса:x(n):0.001980,x(n-1)=0.000000
Метод сопряженных градиентов:x(n)=0.001980,x(n-1)=0.000001
Размерность матрицы:100
Количество потоков:1
Время:0.004853s
```

Рисунок 12 – Размер 100

Результаты испытаний показывают, что результаты, полученные двумя методами, практически совпадают. Разница может быть вызвана недостаточным количеством итераций алгоритма метода сопряженного градиента. Таким образом, можно сделать вывод, что реализованный алгоритм метода сопряженных градиентов эффективен.

4.2 Результат работы

Установить размерность матрицы коэффициентов A на 1000,Значение элемента вектора B равно 100,максимальное количество итераций 20, использовать 1, 2, 4 и 8 потоков в разных методах для сравнения времени

алгоритма. Результаты при использовании технологии c+pthread показаны ниже на рисунках:

```
1  Количество итерации:20
2  Метод Гаусса:x(n):0.000200,x(n-1)=0.000000
3  Метод сопряженных градиентов:x(n)=0.000109,x(n-1)=0.000074
4  Размерность матрицы:1000
5  Количество потоков:1
6  Время:0.228925s
```

Рисунок 13 – Количество потоков 1 при c+pthread

```
1  Количество итерации:20
2  Метод Гаусса:x(n):0.000200,x(n-1)=0.000000
3  Метод сопряженных градиентов:x(n)=0.000109,x(n-1)=0.000074
4  Размерность матрицы:1000
5  Количество потоков:2
6  Время:0.149630s
```

Рисунок 13 – Количество потоков 2 при c+pthread

```
1  Количество итерации:20
2  Метод Гаусса:x(n):0.000200,x(n-1)=0.000000
3  Метод сопряженных градиентов:x(n)=0.000109,x(n-1)=0.000074
4  Размерность матрицы:1000
5  Количество потоков:4
6  Время:0.121615s
```

Рисунок 13 – Количество потоков 4 при c+pthread

```
1  Количество итерации:20
2  Метод Гаусса:x(n):0.000200,x(n-1)=0.000000
3  Метод сопряженных градиентов:x(n)=0.000109,x(n-1)=0.000074
4  Размерность матрицы:1000
5  Количество потоков:8
6  Время:0.108339s
```

Рисунок 13 – Количество потоков 8 при c+pthread

Результаты при использовании технологии c+mpi показаны ниже на рисунках:

```
18  Количество итерации:20
19  Метод Гаусса:x(n):0.000200,x(n-1)=0.000000
20  Метод сопряженных градиентов:x(n)=0.000109,x(n-1)=0.000074
21  Размерность матрицы:1000
22  Количество потоков:1
23  Время:0.180364s
```

Рисунок 14 – Количество потоков 1 при c+mpi

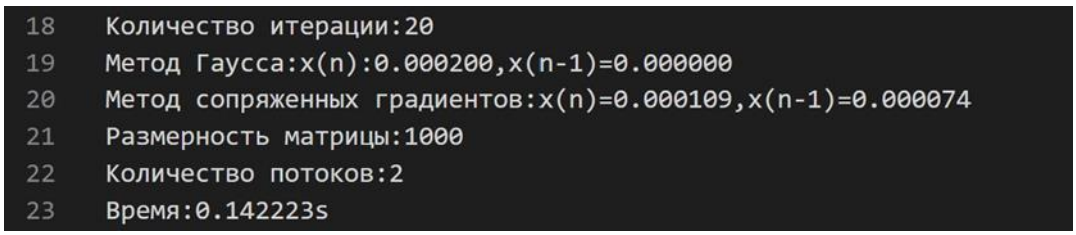

Рисунок 15 – Количество потоков 2 при c+mpi

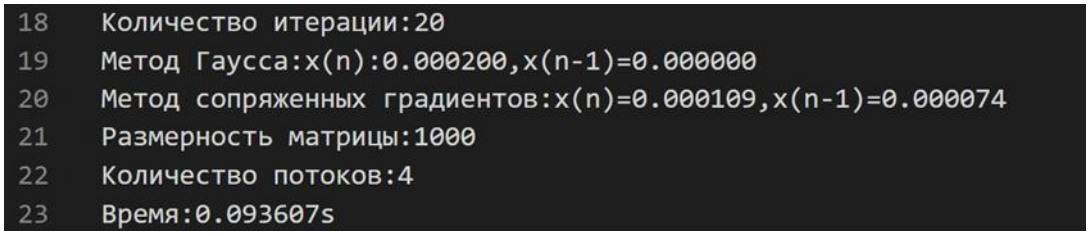

Рисунок 16 – Количество потоков 4 при c+mpi

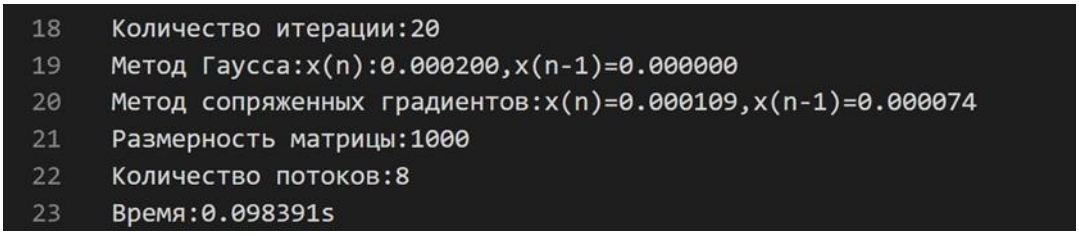

Рисунок 17 – Количество потоков 8 при c+mpi

Результаты при использовании технологии C+OpenMP показаны ниже на рисунках:

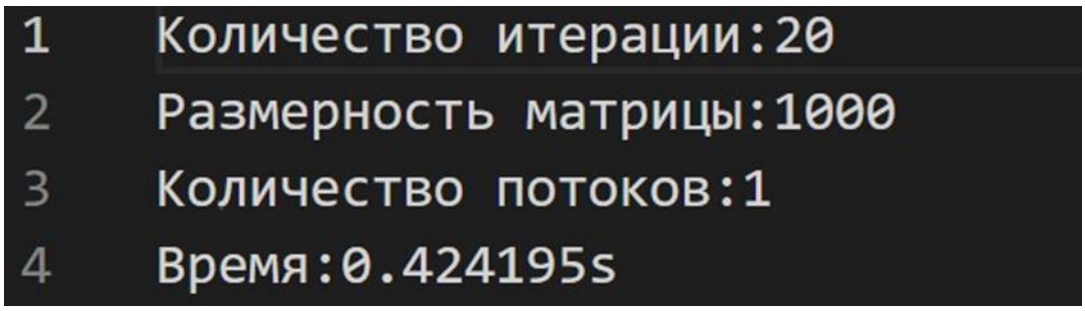

Рисунок 18 – Количество потоков 1 при C+OpenMP

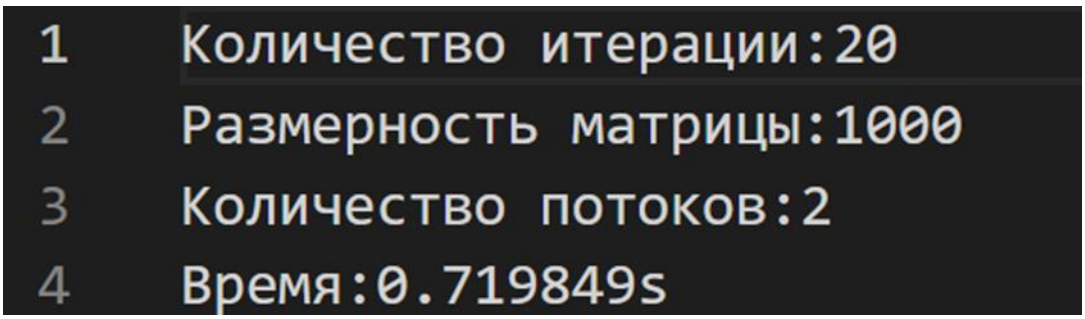

Рисунок 19 – Количество потоков 2 при C+OpenMP

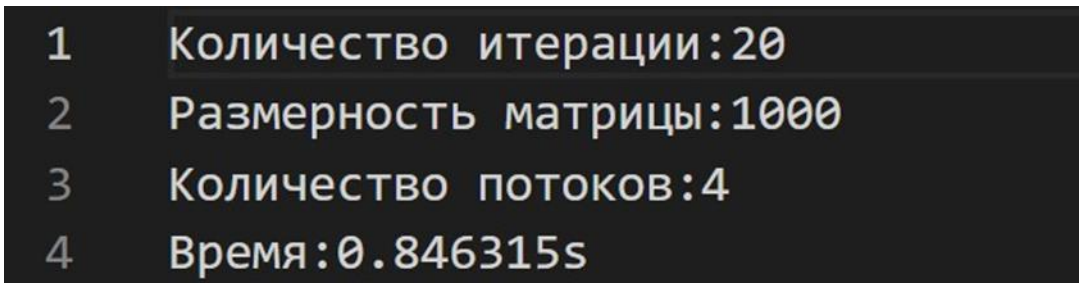

Рисунок 20 – Количество потоков 4 при C+OpenMP

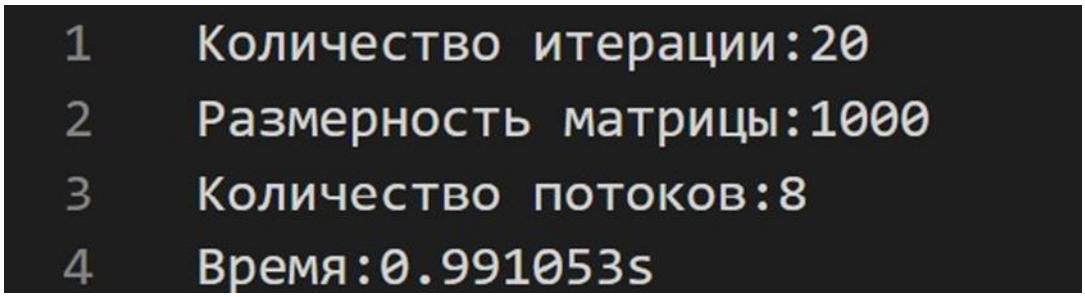

Рисунок 21 – Количество потоков 8 при C+OpenMP

Результаты при использовании технологии Python+mpi показаны ниже на рисунках:

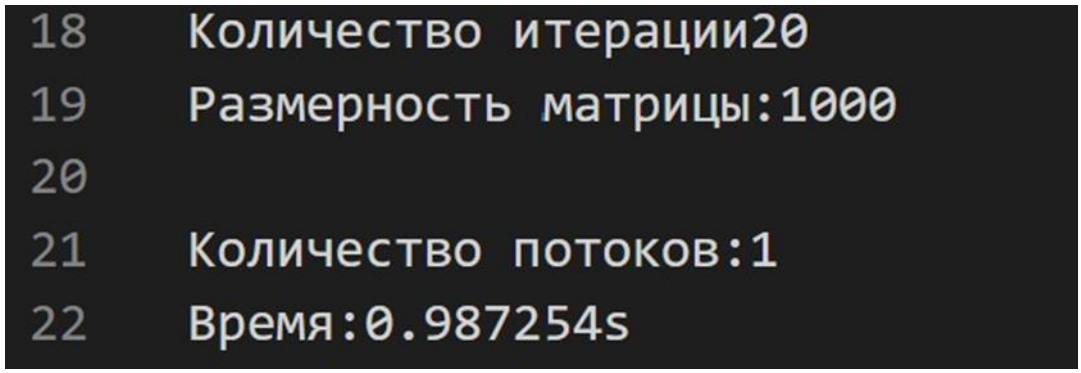

Рисунок 22 – Количество потоков 1 при Python+mpi

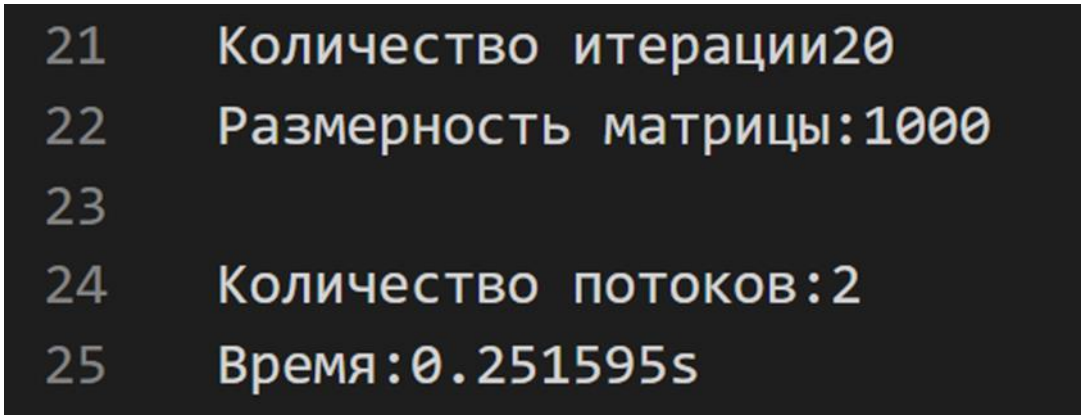

Рисунок 23 – Количество потоков 2 при Python+mpi

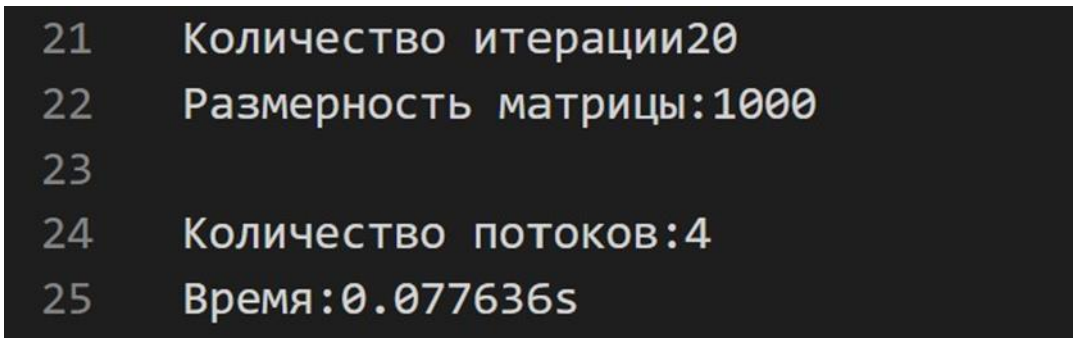

Рисунок 24 – Количество потоков 4 при Python+mpi

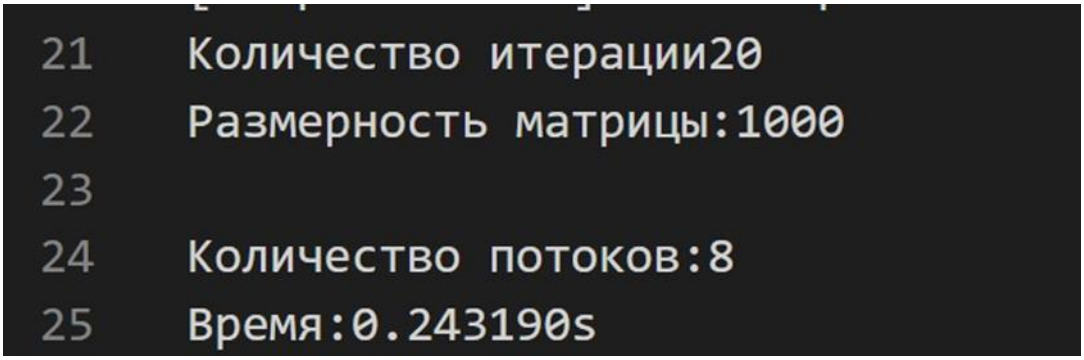

Рисунок 25 – Количество потоков 8 при Python+mpi

Через файл конфигурации можно выбрать несколько ресурсов узла, как показано ниже на рисунке 26.

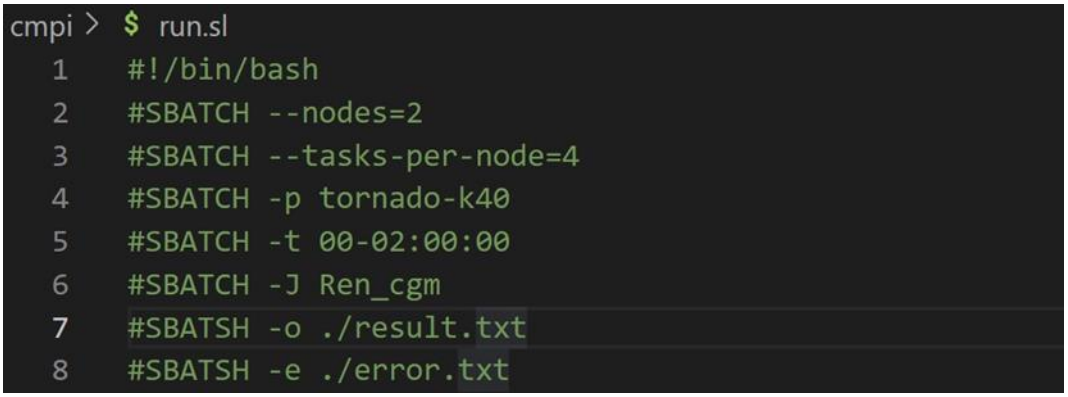

Рисунок 26 – Файл конфигурации

Результаты при выполнении на нескольких узлах.

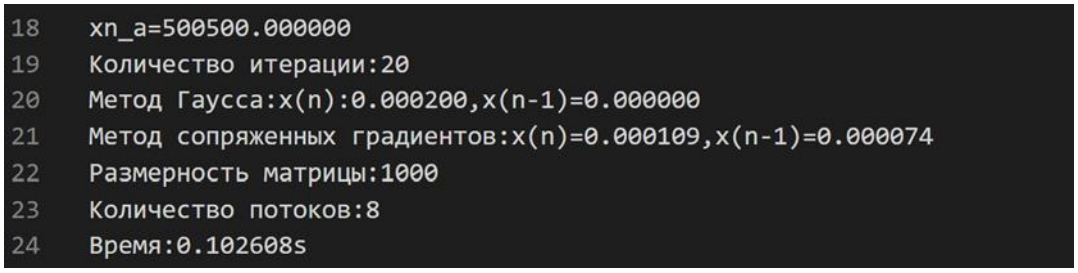

Рисунок 27 – 1 узел

```
18  xn_a=500500.000000
19  Количество итерации:20
20  Метод Гаусса:x(n):0.000200,x(n-1)=0.000000
21  Метод сопряженных градиентов:x(n)=0.000109,x(n-1)=0.000074
22  Размерность матрицы:1000
23  Количество потоков:8
24  Время:0.162772s
```

Рисунок 27 – 2 узла

```
18  xn_a=500500.000000
19  Количество итерации:20
20  Метод Гаусса:x(n):0.000200,x(n-1)=0.000000
21  Метод сопряженных градиентов:x(n)=0.000109,x(n-1)=0.000074
22  Размерность матрицы:1000
23  Количество потоков:8
24  Время:0.170941s
```

Рисунок 27 – 4 узла

5 Заключение

Был реализован алгоритм сопряженного градиента. При размерности исходной матрицы коэффициентов A 1000 и значении постоянного вектора 100. Последние два элемента полученного вектора решения X равны соответственно 0.000109 и 0.000074. Сравнение времени работы алгоритма при использовании разных методов показано ниже в Таблице 1.

Таблица 1 – Время работы алгоритмов

Количество потоков/Метод	c+pThread	c+mpi	c+openmp	Python+mpi
1	0,23 сек.	0,18 сек.	0,42 сек.	0,99 сек.
2	0,15 сек.	0,14 сек.	0,72 сек.	0,25 сек.
4	0,12 сек.	0,09 сек.	0,85 сек.	0,07 сек.
8	0,10 сек.	0,10 сек.	1,00 сек.	0,24 сек.

Помимо использования технологии openmp, чем больше потоков, тем короче время. Среди них самым эффективным стал c+mpi.

Когда максимальное количество итераций равно 20, размерность матрицы равна 1000, а количество потоков равно 8, при запуске программы на 1, 2 и 4 узлах, время будет следующим:

Таблица 2 – Время работы при разном количестве узлов

Количество узла	1	2	4
Время	0,10 сек.	0,16 сек.	0,17 сек.

Как видно из результатов в таблице, чем больше узлов, тем дольше работает алгоритм.

Приложение: https://github.com/SemkinDE/parallprog17