МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №1**

**по курсу «Параллельная обработка данных»**

**Изучение технологии MPI**

Выполнил: В.А. Петросян

Группа: 8О-408Б-17

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2020

**Условие**

**Цель работы :** Знакомство с технологией MPI. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода.

**Вариант 2**: обмен граничными слоями через bsend, контроль сходимости allgather;

**Программное и аппаратное обеспечение**

Сведения о системе:

1. Процессор: Intel Core i7-Q720 1.60GHz
2. Количество ядер 4.
3. Количество потоков 8.

 4. Оперативная память: 8 ГБ

 5. HDD: 465 ГБ

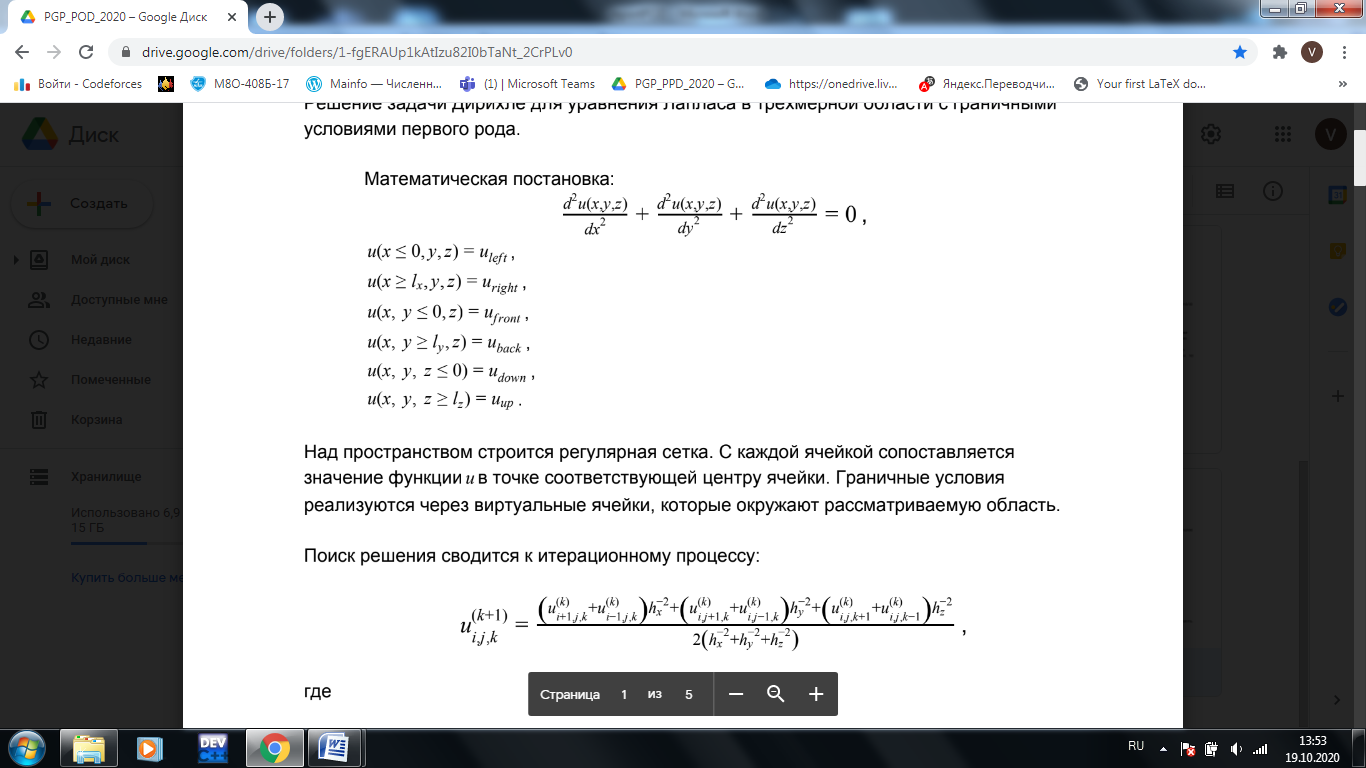
Программное обеспечение:

 1. OS: Windows 7

 2. IDE: Visual Studio 2019

 3. Компилятор: mpic++

**Метод решения**



Опишу немного логику работы с данными. Допустим размер блока, который обсчитывает один процесс это x \* y \* z. Мы выделим на каждое измерение два дополнительных элемента для хранения граничных условий. Теперь блок имеет размер (x + 2) \* (y + 2) \* (z + 2) . Храним данные блока в виде одномерного массива, но обращаемся к нему как к трёхмерному. Чтобы было проще взаимодействовать с массивом, напишем пару макросов для правильного доступа по индексу к данным.

// Индексация внутри блока

#define \_i(i, j, k) (((k) + 1)\*(blockY + 2)\*(blockX + 2) + ((j) + 1)\*(blockX + 2) + (i) + 1)

#define \_ix(id) (((id) % (blockX + 2)) - 1)

#define \_iy(id) ((((id) % ((blockY + 2) \* (blockX + 2))) / (blockX + 2)) - 1)

#define \_iz(id) ( ( (id) / ((blockY + 2) \* (blockX + 2)) ) - 1)

Пока не достигнем нужной точности *ε* будем делиться нашими данными с соседними по сетке процессами.

**Описание программы**

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numproc); - общее количество процессов.

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id); - номер нашего процесса 0 <= id < numproc.

MPI\_Bcast(&blockX, 1, MPI\_INT, 0, MPI\_COMM\_WORLD); - “Широковещательное сообщение” передает всем процессам значение переменной blockX.

ib = \_ibx(id); - индексация процесса в сетке блоков по x

jb = \_iby(id); - индексация процесса в сетке блоков по y

kb = \_ibz(id); - индексация процесса в сетке блоков по z

int buffer\_size = 12 \* sizeOfBuff \* sizeof(double) + 12 \* MPI\_BSEND\_OVERHEAD;

double \*buffer = (double \*)malloc(buffer\_size);

MPI\_Buffer\_attach(buffer, buffer\_size); - через этот буфер процесс будет общаться со всеми остальными. Я выделим место с двойным запасом.

for(i = -1; i <= blockX; i++){

   for(j = -1; j <= blockY; j++){

       for(k = -1; k <= blockZ; k++){

           data[\_i(i, j, k)] = startU; - инициализация начальным условием

       }

   }

}

MPI\_Barrier(MPI\_COMM\_WORLD); - синхронизирует процессы внутри коммутатора. В коммутаторе MPI\_COMM\_WORLD содержатся все процессы.

if (ib + 1 < gridX){

   for(j = 0; j < blockY; j++){

       for(k = 0; k < blockZ; k++){

           buff[j \* blockZ + k] = data[\_i(blockX - 1, j, k)];

       }

   }

Посылка данных процессу соседу. Так как пространство трехмерное, то мы посылаем двумерный массив. Посылка производится через Bsend.

   int tmpSize = blockY \* blockZ;

   MPI\_Bsend(buff, tmpSize, MPI\_DOUBLE, \_ib(ib + 1, jb, kb), id, MPI\_COMM\_WORLD);

}

if (ib > 0) {

    int tmpSize = blockY \* blockZ;

    MPI\_Recv(buff, tmpSize, MPI\_DOUBLE, \_ib(ib - 1, jb, kb),

\_ib(ib - 1, jb, kb), MPI\_COMM\_WORLD, &status);

    for(j = 0; j < blockY; j++){

        for(k = 0; k < blockZ; k++){

            data[\_i(-1, j, k)] = buff[j \* blockZ + k];

        }

    }

}

else {

    for(j = 0; j < blockY; j++){

        for(k = 0; k < blockZ; k++){

            data[\_i(-1, j, k)] = leftU;

        }

    }

}

**Результаты**

**Общий размер задачи 30 х 30 х 30**

|  |  |  |  |
| --- | --- | --- | --- |
| **Time** | **gridX** | **gridY** | **gridZ** |
| 4.757188 | 1 | 1 | 1 |
| 2.455298 | 1 | 1 | 2 |
| 1.725504 | 1 | 2 | 2 |
| 1.209881 | 3 | 2 | 1 |
| 2.059756 | 2 | 2 | 2 |
| 4.510396 | 3 | 2 | 3 |
| 3.558537 | 3 | 3 | 3 |
| 3.904495 | 6 | 1 | 5 |
| 2.965461 | 2 | 3 | 5 |
| 3.873532 | 6 | 6 | 1 |
| 3.732131 | 3 | 2 | 6 |

**Общий размер задачи 40 х 40 х 40**

|  |  |  |  |
| --- | --- | --- | --- |
| **Time** | **gridX** | **gridY** | **gridZ** |
| 19.223992 | 1 | 1 | 1 |
| 9.992391 | 1 | 1 | 2 |
| 7.297924 | 1 | 2 | 2 |
| 9.911798 | 2 | 2 | 2 |
| 17.197795 | 2 | 2 | 4 |
| 11.785805 | 2 | 4 | 4 |

**Общий размер задачи 52 х 52 х 52**

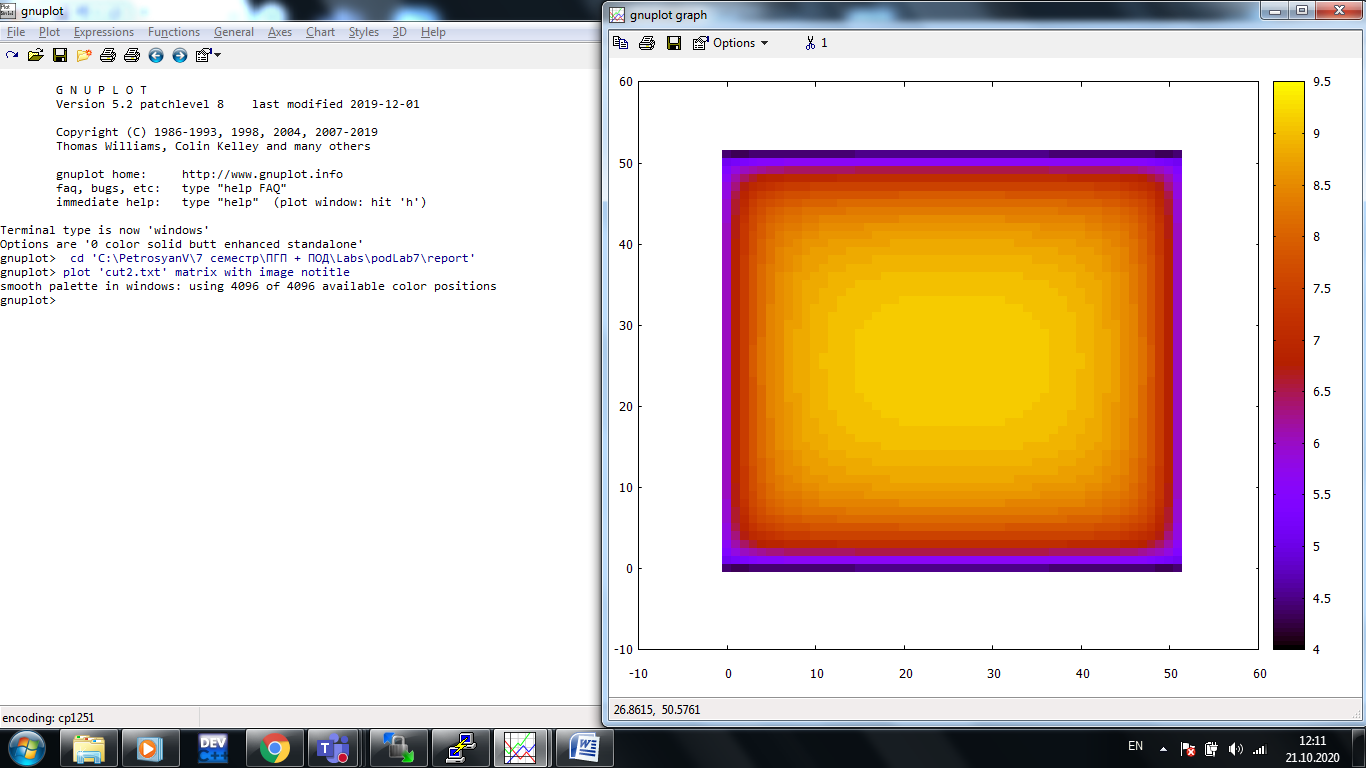
|  |  |  |  |
| --- | --- | --- | --- |
| **Time** | **gridX** | **gridY** | **gridZ** |
| 31.797320 | 1 | 1 | 2 |
| 22.126758 | 1 | 2 | 2 |
| 31.648761 | 2 | 2 | 2 |
| 42.379538 | 2 | 2 | 4 |
| 28.337448 | 2 | 4 | 4 |

**Общий размер задачи 64 х 64 х 64**

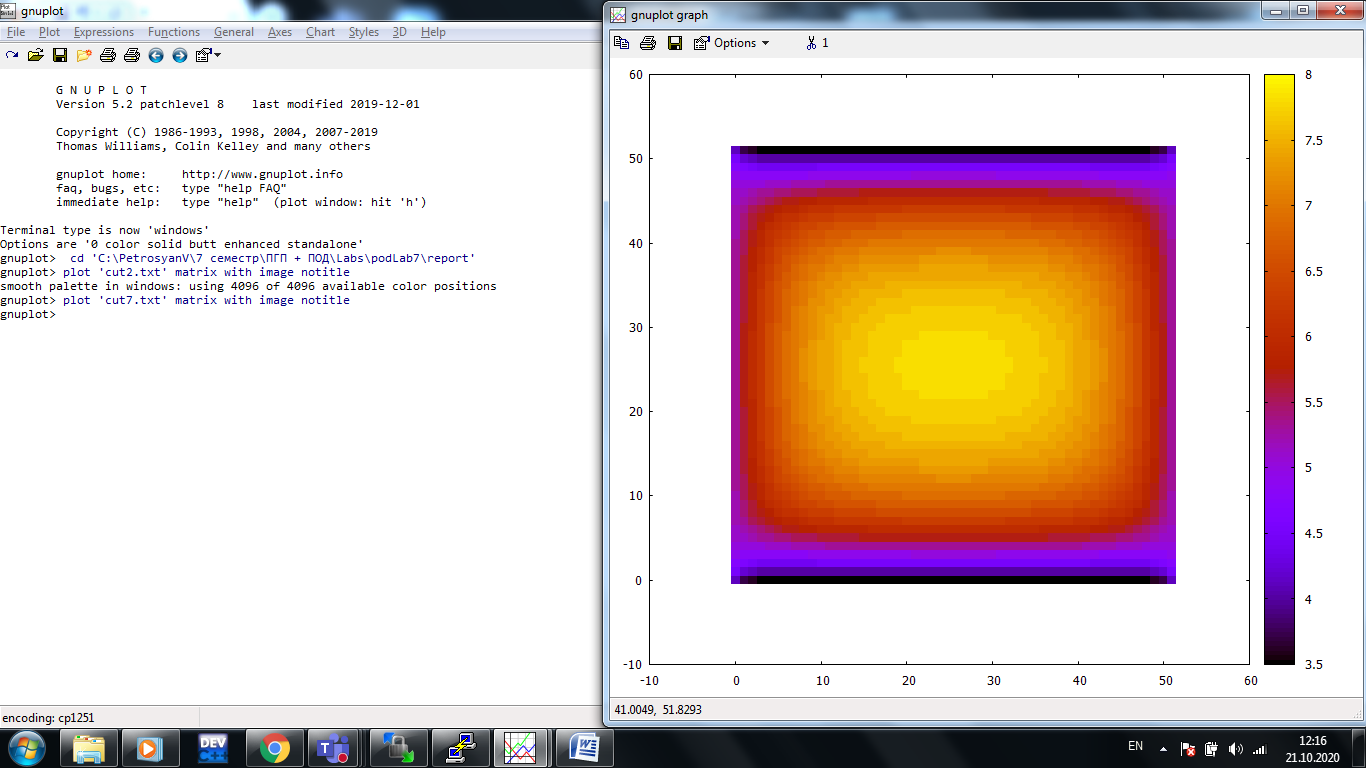
|  |  |  |  |
| --- | --- | --- | --- |
| **Time** | **gridX** | **gridY** | **gridZ** |
| 53.447667 | 1 | 2 | 2 |
| 54.648724 | 2 | 2 | 2 |
| 73.770564 | 2 | 2 | 4 |
| 55.249143 | 2 | 4 | 4 |

**Температурный срез для задачи 52 х 52 х 52**

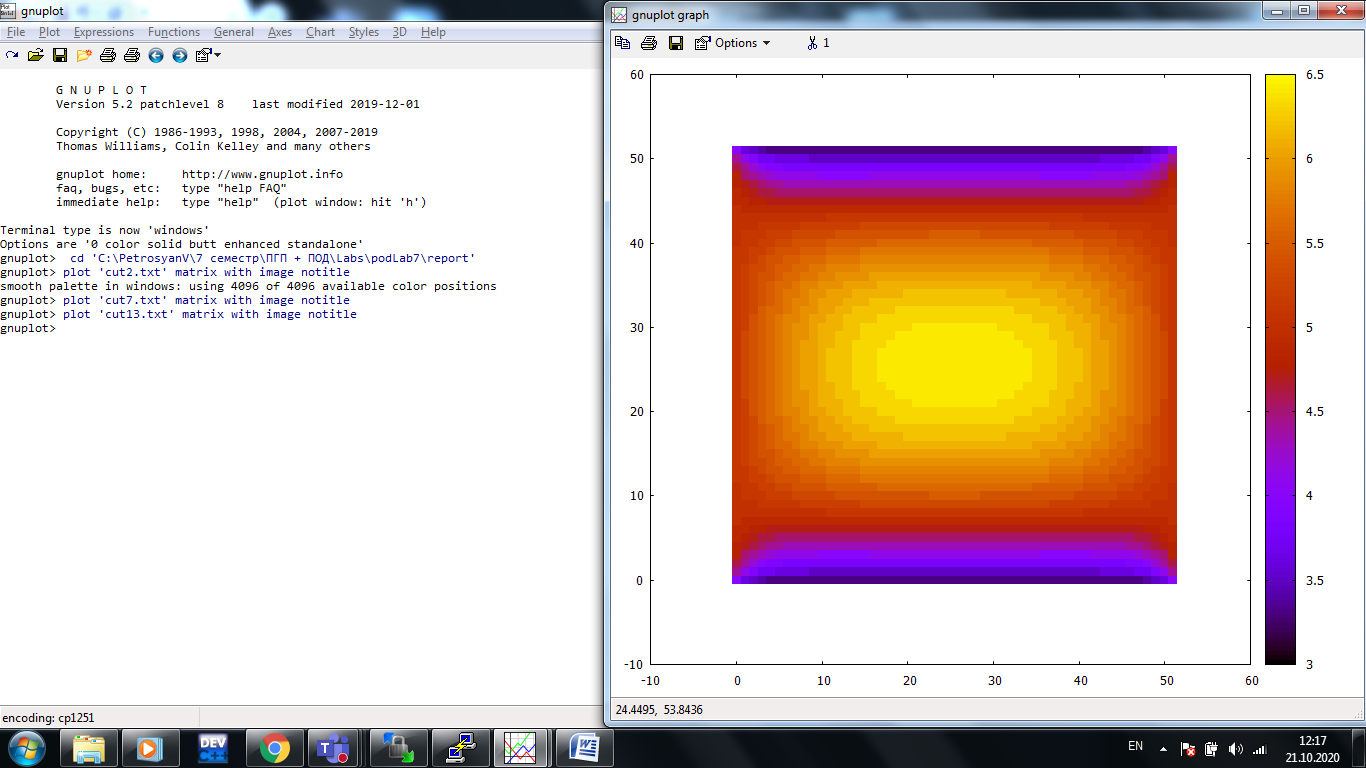
* Для z = 2

****

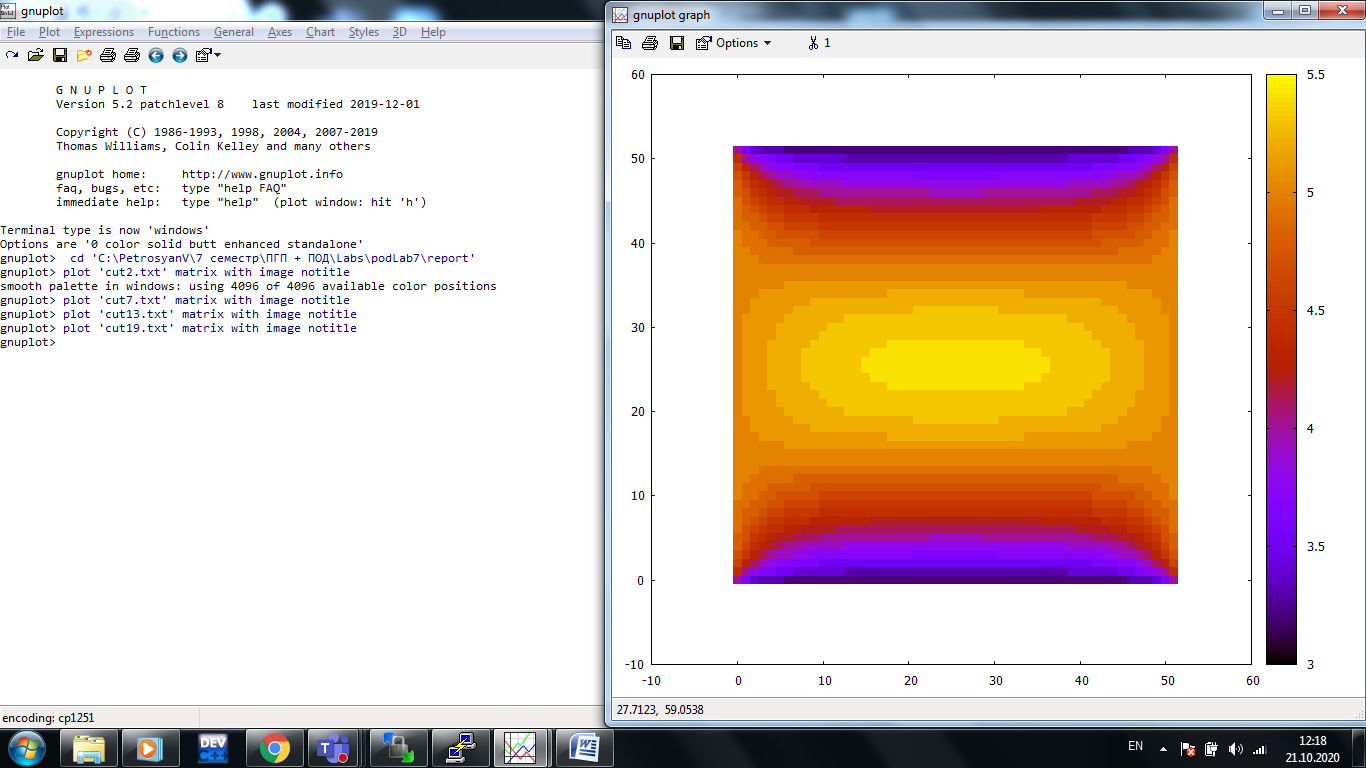
* Для z = 7

****

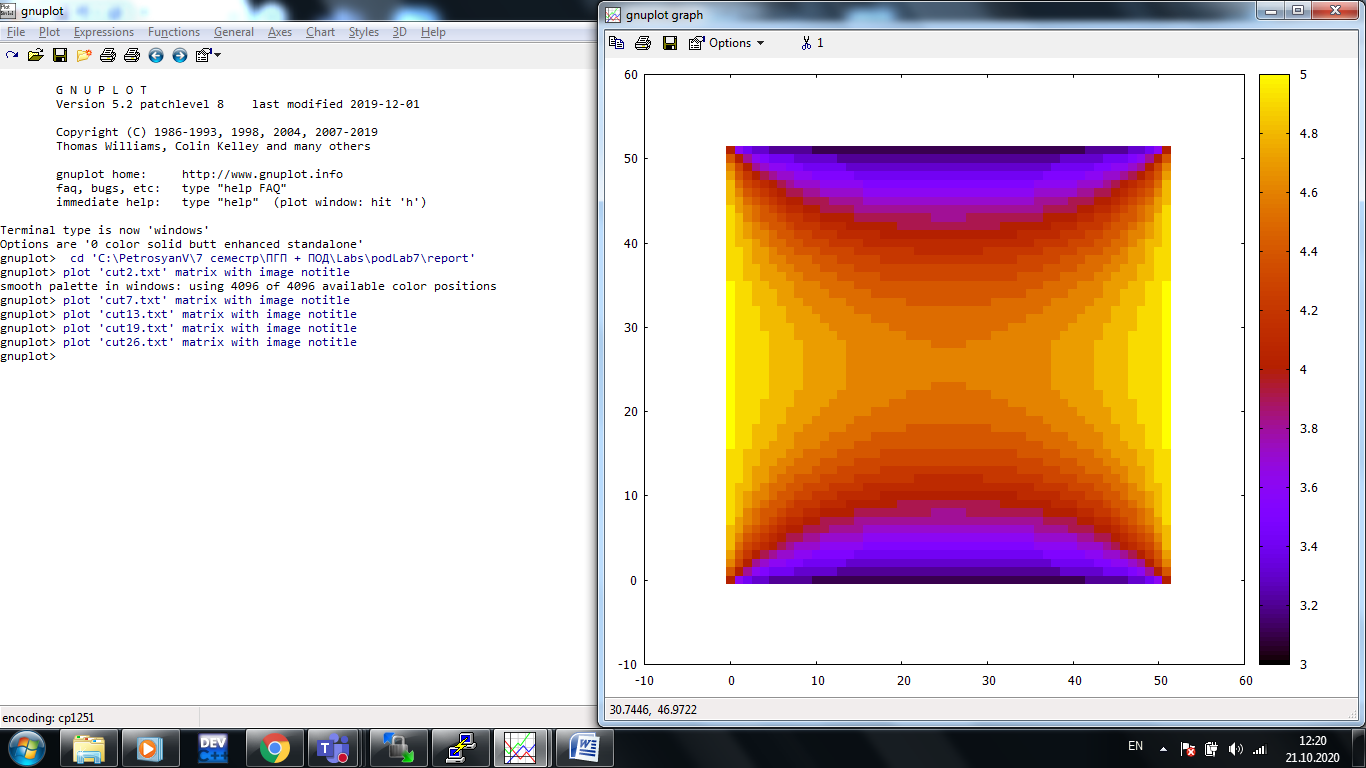
* Для z = 13

****

* Для z = 19



* Для z = 25



**Выводы**

Сложность в программировании данной лр не испытал. Весь код был написан за 3 - 4 часа. Единственное, что вызвало затруднение это ошибка по невнимательности. На поиск бага ушел день.

Люди улучшали производительность CPU выпуская всё более мощные процессоры, но чем мощнее процессор, тем больше тепла он выделяет. Уже чисто физически трудно делать процессоры мощнее. Поняв это люди стали компенсировать мощность количеством. Именно поэтому задача распараллеливания программы на нескольких CPU актуальна. Google Chrome использует технологию распараллеливания создавая несколько процессов при вызове браузера.