1. Формулировка задачи

Выполнение операции AES ShiftRows над набором из N-матриц. Данные передаются по протоколу TCP.

2. Алгоритм выполнения бизнес-логики

В процедуре `ShiftRows` байты в каждой строке матрицы циклически сдвигаются влево. Размер смещения байтов каждой строки зависит от её номера. Алгоритм выполнения операции представлен на рисунке 1.

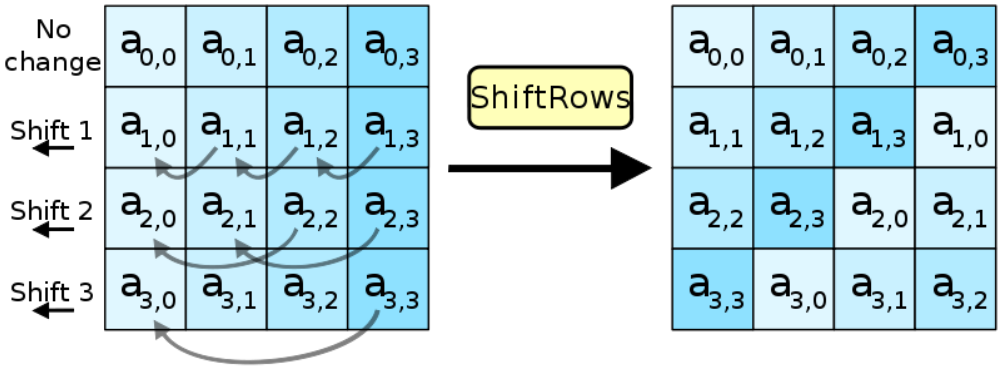


Рисунок 1 – Алгоритм выполнения операции ShiftRows

Данный алгоритм применяется для каждой из N матриц.

3. Последовательный вариант исполнения
Исходный код программы-вычислителя представлен в листинге 1.

```
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <time.h>
#define PORT 8080
#define SA struct sockaddr

int create() {
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");

    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) < 0)
      printf("setsockport(SO_REUSEADDR) failed \n");

    bzero(&servaddr, sizeof(servaddr));
```

```
    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client...\n");

    return connfd;
}

void func(int sockfd)
{
        int n = 0;
        read(sockfd, &n, sizeof(n));
        int width = 0;
        read(sockfd, &width, sizeof(width));
        int height = 0;
        read(sockfd, &height, sizeof(height));
        n = ntohl(n);
        width = ntohl(width);
        height = ntohl(height);


        printf("Received header:\n Number of matrices = %d\n Width of each = %d\n Height
of each = %d\n", n, width, height);

        u_char *in = malloc(sizeof(u_char) * n * width * height);
        u_char *out = malloc(sizeof(u_char) * n * width * height);

        u_char buff[height * width];

        for (size_t i = 0; i < n; i++) {
            read(sockfd, in + i * height * width, sizeof(buff));
        }

       struct timeval start, end;
       gettimeofday(&start, NULL);

        process(n, height, width, in, out);

       gettimeofday(&end, NULL);

       double delta = ((end.tv_sec  - start.tv_sec) * 1000000u +
         end.tv_usec - start.tv_usec) / 1.e6;
       printf("\nElapsed: %lf ms\n", delta);

       printf("Result send.\n");

        printf("Sending result...\n");
        for (size_t i = 0; i < n; i++) {
            write(sockfd, out + i * height * width, sizeof(buff));
        }

       write(sockfd, &delta, sizeof(delta));
```

```
       printf("Result send.\n");
         free(in);
         free(out);
}

void process(int n, int height, int width, u_char *in, u_char *out) {
    printf("Start calculations\n");
    int i, j, k;
    for(i = 0; i < n; i++) {
        for(j = 0; j < height; j++) {
            // printf("i = %d, j= %d, threadId = %d \n", i, j, omp_get_thread_num());
            for(k = 0; k < width; k++) {
                int current = i * width * height + j * width + k;
                if (k + j < width) {
                    out[current] = in[current + j];
                } else {
                    out[current] = in[current + j - width];
                }
            }
        }
    }
    printf("Calculations complete\n");
}

// Driver function
int main()
{
    int sockfd;
    sockfd = create();
    func(sockfd);
    close(sockfd);
}
```

Исходный код программы-генератора преведён в листинге 2.

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <stdbool.h>
#include <time.h>
#include <errno.h>
#include <string.h>
#define SA struct sockaddr

u_char randomByte()
{
  return (u_char) randInRange( 0, 255 );
}

int randInRange( int min, int max )
{
  double scale = 1.0 / (RAND_MAX + 1);
  double range = max - min + 1;
  return min + (int) ( rand() * scale * range );
}

int create(char* addr, char* port) {
    int sockfd;
    struct sockaddr_in servaddr, cli;


    // socket create and varification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));


    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr(addr);
    servaddr.sin_port = htons(port);


    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("connection with the server failed...\n");
```

```
        exit(0);
    }
    else
        printf("connected to the server..\n");


    return sockfd;
}

void send_data(int sockfd, int count, int width, int height, int silent)
{
    printf("Sending data...");

    int n = htonl(count);
    write(sockfd, &n, sizeof(n));
    int w = htonl(width);
    write(sockfd, &w, sizeof(w));
    int h = htonl(height);
    write(sockfd, &h, sizeof(h));

    int sz = width * height;
    u_char buff[sz];
    for(size_t i = 0; i < count; i++)
    {
        bzero(buff, sizeof(buff));
        for(size_t j = 0; j < sz; j++)
        {
            buff[j] = randomByte();


            if (!silent) {
                if (j % width == 0) {
                     printf("\n");
                 }
                 if (j % sz == 0) {
                     printf("\n");
                 }
                printf("\t%x", buff[j]);
            }
        }
        write(sockfd, buff, sizeof(buff));
    }
    printf("\nData have been send.");
}

double receive_data(int sockfd, int count, int width, int height, int silent) {

    int size = width * height;

    u_char buff[size];
    bzero(buff, sizeof(buff));


    printf("\nReceiving result...");

    u_char *in = malloc(sizeof(u_char) * count * width * height);

    for(size_t i = 0; i < count; i++)
    {
        read(sockfd, in + i * size, sizeof(buff));
        if (!silent) {
            for(size_t j = 0; j < size; j++) {
                if (j % width == 0) {
                     printf("\n");
                 }
                 if (j % size == 0) {
                     printf("\n");
                 }
                printf("\t%x", in[i * size + j]);
            }
       }
    }

    double time = 0;
    read(sockfd, &time, sizeof(time));

    printf("\nResult received.");
    free(in);


    return time;
}

void save_report(char* file_name, long int size_bytes, double time_ms) {
    FILE *file;
    file = fopen(file_name, "ab");
    if (!file) {
        printf("something went wrong: %s", strerror(errno));
       exit(1);
    }
    fprintf(file, "%ld:%lf\n", (long int) size_bytes, (double) time_ms);
    fclose(file);
```

```
    printf("Report saved.\n");
}


int main(int argc, char **argv)
{
    int sockfd;
    char* host = argv[1];
    int port = atoi(argv[2]);
    int count = atoi(argv[3]);
    int height = atoi(argv[4]);
    int width = atoi(argv[4]);
    char* file_name = argv[5];
    bool silent = false;
    if (argv[6])
       silent = true;

    sockfd = create(host, port);

    clock_t begin = clock();
    send_data(sockfd, count, width, height, silent);
    double time = receive_data(sockfd, count, width, height, silent);
    clock_t end = clock();
    printf("\nElapsed: %ld ms", (long int)(end - begin));

    close(sockfd);
    printf("\nSocket successfully closed.\n");
    save_report(file_name, count * height * width, time);
}
```

График зависимости времени выполнения от размера данных представлен на рисунке 2. Операция проводилась над N матрицами размером 1024x1024, таким образом N – количество данных в Мб. Использовался процессор Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz. Исходные данные для постороения графиков были взяты из сгенерированного файла results.txt. Содержимое файла results.txt приведено в таблице 1.

Таблица 1. Результаты работы программы с последовательным выполнением

bytes	Sequential
1	0.003178
128	0.406489
256	0.80658
384	1.198283
512	1.599309
640	2.018421
768	2.403031
896	2.837906
1024	3.194413

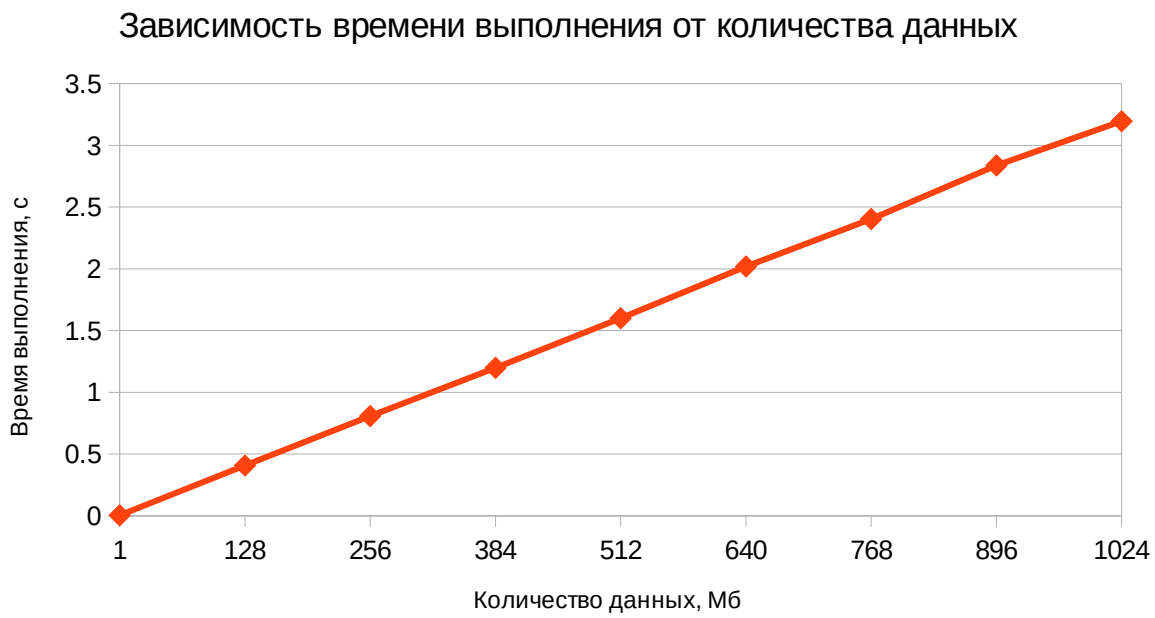


Рисунок 2 – График зависимости для варианта программы с последовательным исполнением

4. OpenMP

В качестве начального варианта распараллеливания программы было выбрано поматричное распараллеливание.
Исходный код программы-вычислителя представлен в листинге 3.

```
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <omp.h>
#include <time.h>
#define PORT 8080
#define SA struct sockaddr


int create() {
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;


    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");


    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) < 0)
       printf("setsockport(SO_REUSEADDR) failed \n");


    bzero(&servaddr, sizeof(servaddr));


    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);


    // Binding newly created socket to given IP and verification
```

```
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");


    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);


    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client...\n");


    return connfd;
}


void func(int sockfd, int n_threads)
{
        int n = 0;
        read(sockfd, &n, sizeof(n));
        int width = 0;
        read(sockfd, &width, sizeof(width));
        int height = 0;
        read(sockfd, &height, sizeof(height));
        n = ntohl(n);
        width = ntohl(width);
        height = ntohl(height);



        printf("Received header:\n Number of matrices = %d\n Width of each = %d\n Height of each =
%d\n", n, width, height);


        u_char *in = malloc(sizeof(u_char) * n * width * height);
        u_char *out = malloc(sizeof(u_char) * n * width * height);


        u_char buff[height * width];


        for (size_t i = 0; i < n; i++) {
            read(sockfd, in + i * height * width, sizeof(buff));
        }


       struct timeval start, end;
       gettimeofday(&start, NULL);


        process(n, height, width, in, out, n_threads);


       gettimeofday(&end, NULL);


       double delta = ((end.tv_sec  - start.tv_sec) * 1000000u +
         end.tv_usec - start.tv_usec) / 1.e6;
       printf("\nElapsed: %lf ms\n", delta);


       printf("Result send.\n");


        printf("Sending result...\n");
        for (size_t i = 0; i < n; i++) {
            write(sockfd, out + i * height * width, sizeof(buff));
        }


       write(sockfd, &delta, sizeof(delta));


       printf("Result send.\n");
        free(in);
        free(out);
}
```

```
void process(int n, int height, int width, u_char *in, u_char *out, int n_threads) {
    printf("Start calculations\n");
    omp_set_dynamic(0);
    omp_set_num_threads(n_threads);


int i, j, k;
#pragma omp parallel shared(in, out) private(i,j,k)
{
   #pragma omp for // collapse(2)
    for(i = 0; i < n; i++) {
        for(j = 0; j < height; j++) {
            // printf("i = %d, j= %d, threadId = %d \n", i, j, omp_get_thread_num());
            for(k = 0; k < width; k++) {
                int current = i * width * height + j * width + k;
                if (k + j < width) {
                    out[current] = in[current + j];
                } else {
                    out[current] = in[current + j - width];
                }
            }
        }
    }
}
    printf("Calculations complete\n");
}


// Driver function
int main(int argc, char **argv)
{
    int sockfd;
    int n_threads = atoi(argv[1]);
    sockfd = create();
    func(sockfd, n_threads);
    close(sockfd);
}
```

Таблица 2. Результаты работы программы с использованием OpenMP

bytes	Sequential	Threads = 2	Threads = 4	Threads = 8	Threads = 16	Threads = 64
1	0.003178	0.003282	0.003523	0.003728	0.003591	0.003632
128	0.406489	0.205686	0.1047	0.093005	0.075036	0.074668
256	0.80658	0.409162	0.208748	0.165908	0.151323	0.147849
384	1.198283	0.613895	0.309704	0.240023	0.22233	0.218263
512	1.599309	0.819062	0.412197	0.318836	0.294659	0.2909
640	2.018421	1.026453	0.518227	0.400625	0.36333	0.362997
768	2.403031	1.234458	0.61907	0.477736	0.437699	0.430699
896	2.837906	1.431896	0.730553	0.559527	0.509409	0.50061
1024	3.194413	1.637582	0.824429	0.629518	0.577687	0.569694

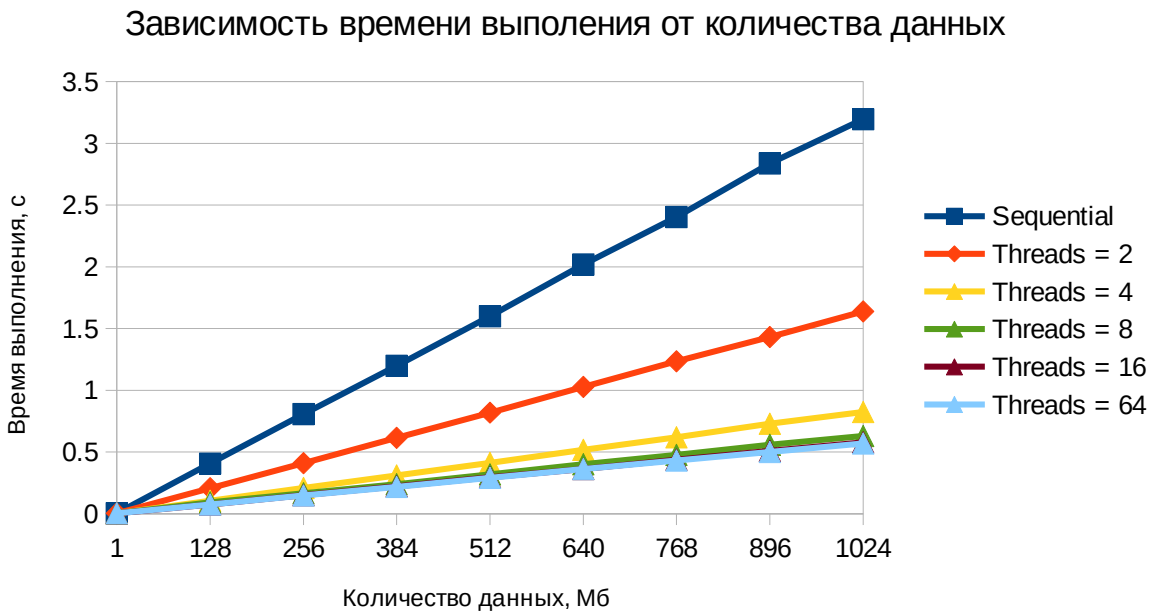


Рисунок 3 – График зависимости для варианта программы с использованием OpenMP

В результате анализа результатов выполнения программы было установлено, что оптимальное количество потоков равно 8. Для этого количества потоков были проведены эксперименты с различными уровнями распараллеливания: поматричное, построчное, построчное с использование collapse. Результаты в таблице 3 и на рисунке 4. Код в листингах 4 и 5.

Листинг 4. Построчное распараллеливание

```
void process(int n, int height, int width, u_char *in, u_char *out, int n_threads) {
    printf("Start calculations\n");
    omp_set_dynamic(0);
    omp_set_num_threads(n_threads);


int i, j, k;


    for(i = 0; i < n; i++) {
#pragma omp parallel shared(in, out, i) private(j,k)
{
   #pragma omp for
        for(j = 0; j < height; j++) {
            // printf("i = %d, j= %d, threadId = %d \n", i, j, omp_get_thread_num());
            for(k = 0; k < width; k++) {
                int current = i * width * height + j * width + k;
                if (k + j < width) {
                    out[current] = in[current + j];
                } else {
                    out[current] = in[current + j - width];
                }
            }
        }
    }
}
    printf("Calculations complete\n");
}
```

Листинг 5. Построчное распараллеливание с использование collapse

```
void process(int n, int height, int width, u_char *in, u_char *out, int n_threads) {
```

```
    printf("Start calculations\n");
    omp_set_dynamic(0);
    omp_set_num_threads(n_threads);


int i, j, k;
#pragma omp parallel shared(in, out) private(i,j,k)
{
   #pragma omp for collapse(2)
    for(i = 0; i < n; i++) {
        for(j = 0; j < height; j++) {
            // printf("i = %d, j= %d, threadId = %d \n", i, j, omp_get_thread_num());
            for(k = 0; k < width; k++) {
                int current = i * width * height + j * width + k;
                if (k + j < width) {
                    out[current] = in[current + j];
                } else {
                    out[current] = in[current + j - width];
                }
            }
        }
    }
}
    printf("Calculations complete\n");
}
```

Таблица 3. Результаты работы программы с использованием OpenMP для 8 потоков с различными уровнями распараллеливания

bytes	External loop	Collapse	Internal loop
1	0.003728	0.003523	0.003728
128	0.093005	0.085098	0.121985
256	0.165908	0.160387	0.248125
384	0.240023	0.257133	0.360398
512	0.318836	0.32447	0.474778
640	0.400625	0.397761	0.596767
768	0.477736	0.4929	0.713646
896	0.559527	0.557298	0.831878
1024	0.629518	0.634791	0.948625

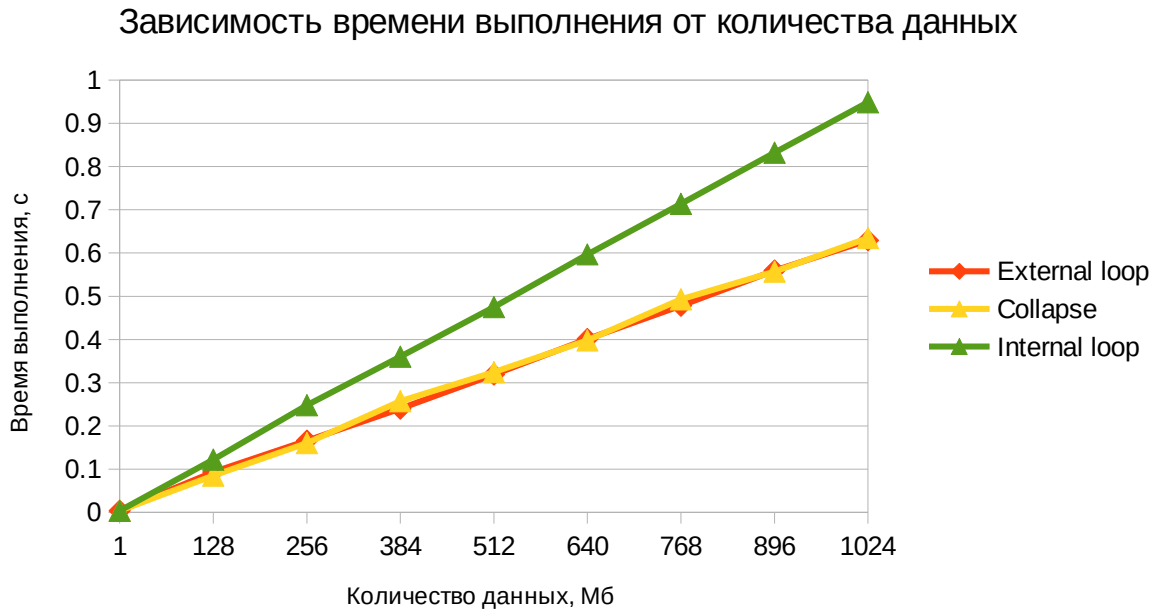
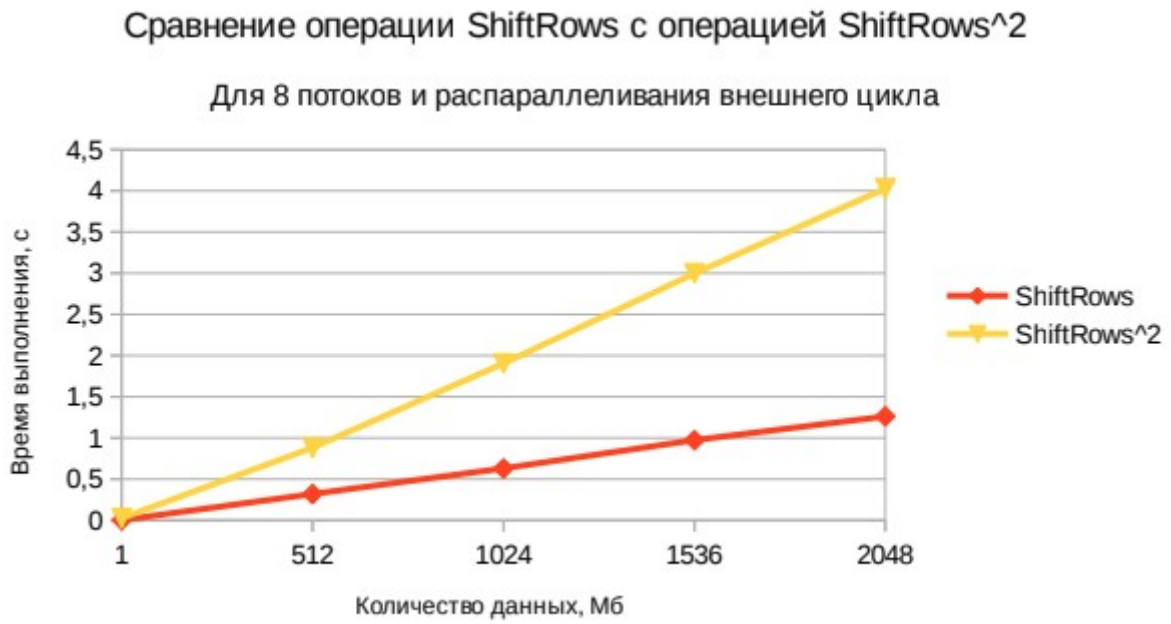


Рисунок 4 – График зависимости для варианта программы с использованием OpenMP для 8 потоков с различными уровнями распараллеливания

В результате анализа результатов выполнения программы было установлено, что наиболее эффективными видами распараллеливания является поматричное и построчное с использованием collapse, который заранее (до выполнения внешнего цикла) распределяет строки между потоками, в отличии от обычного построчного распараллеливания, которое на каждой итерации внешнего цикла производит распределение строк между потоками.



5. MPI.

Исходный код программы-вычислителя представлен в листинге 6.

```
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <time.h>
#include <mpi.h>
#define PORT 8080
#define SA struct sockaddr

int create() {
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");

    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) < 0)
       printf("setsockport(SO_REUSEADDR) failed \n");

    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);
```

```
    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccept the client...\n");

    return connfd;
}

void process(int n, int height, int width, u_char *in/*, u_char *out*/) {

u_char *out = (u_char *) malloc(height * width * sizeof(u_char) * n);
    printf("Start calculations\n width = %d height = %d", height, width);
    int i, j, k;
    for(i = 0; i < n; i++) {
        for(j = 0; j < height; j++) {
            for(k = 0; k < width; k++) {
                int current = i * width * height + j * width + k;
                if (k + j < width) {
                    out[current] = in[current + j];
                } else {
                    out[current] = in[current + j - width];
                }
            }
        }
    }

    printf("Calculations complete\n");

//Тестовый вывод
    for(size_t i = 0; i < n; i++)
        {
            for(size_t j = 0; j < height * width; j++) {
               if (j % width == 0) {
                    printf("\n");
                }
                if (j % height * width == 0) {
                    printf("\n");
                }
               printf("\t%x", out[i * height * width + j]);
            }
        }
}

// Driver function
int main(int argc, char **argv)
{
    int sockfd;
    int rank, size, rc;
    int n, width, height;
    u_char *in = NULL;
    u_char *out = NULL;
    u_char buff[height * width];

    if ((rc = MPI_Init(&argc, &argv)) != MPI_SUCCESS) {
        fprintf(stderr, "Error starting MPI program. Terminating.\n");
        MPI_Abort(MPI_COMM_WORLD, rc);
    }
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) {

    sockfd = create();

    read(sockfd, &n, sizeof(n));
    read(sockfd, &width, sizeof(width));
    read(sockfd, &height, sizeof(height));
    n = ntohl(n);
    width = ntohl(width);
    height = ntohl(height);
```

```
    printf("Received header:\n Number of matrices = %d\n Width of each = %d\n Height of each = %d\
n", n, width, height);

    in = malloc(sizeof(u_char) * n * width * height);
    out = malloc(sizeof(u_char) * n * width * height);



    for (size_t i = 0; i < n; i++) {
        read(sockfd, in + i * height * width, sizeof(buff));
    }
}

    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&width, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&height, 1, MPI_INT, 0, MPI_COMM_WORLD);
    int p = n / size;
    printf("process %d of %d всего матриц %ld  количество матриц на поток %ld\n", rank, size, n, p);

   printf("%d \n", height * width * p);
       //получить данные от рута
    u_char *recvbuf = (u_char *) malloc(height * width * sizeof(u_char) * p);
    MPI_Scatter(in, height * width * p, MPI_UNSIGNED_CHAR, recvbuf, height * width * p,
                MPI_UNSIGNED_CHAR, 0,
                MPI_COMM_WORLD);

    //что то сделать с данными
    process(p, height, width, recvbuf);

    MPI_Gather(recvbuf, height * width * p, MPI_UNSIGNED_CHAR, out, height * width * p,
               MPI_UNSIGNED_CHAR, 0,
               MPI_COMM_WORLD);

    printf("\nGather complete!\n");


if (rank == 0) {
        printf("Sending result...\n");

        for (size_t i = 0; i < n; i++) {
            write(sockfd, out + i * height * width, sizeof(buff));
        }

       printf("Result send.\n");
        free(in);
        free(out);
close(sockfd);
}

MPI_Finalize();
}
```

Результаты выполнения в таблице 4 и на рисунке 5.

Таблица 3. Результаты работы программы с использованием MPI для 8 потоков

bytes	MPI
1	0.003728
128	0.107495
256	0.2070165
384	0.3002105
512	0.396807
640	0.498696
768	0.595691
896	0.6957025
1024	0.7890715

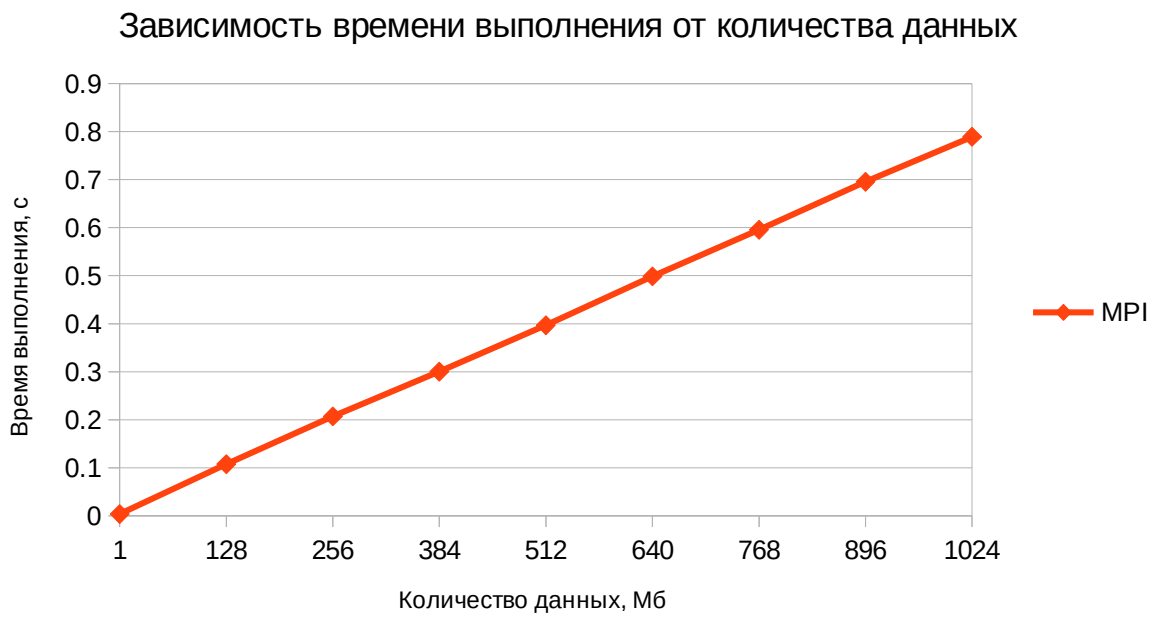


Рисунок 4 – График зависимости для варианта программы с использованием MPI для 8 потоков