Задания из файла выполняются в одном проекте, где номер задания можно выбрать путем ввода номера в консоли, например: "mpiexec -n 4 mpiconsole.exe 15". Так, в методе main инициализируется MPI и значения rank и size заносятся в соответствующие переменные для дальнейшей работы с программой. Выбор метода осуществляется оператором switch и переменной task, получаемой значение из консоли. Выбранное приложение выполняется согласно поставленной инструкции и производится выход из среды MPI, где программа завершает свое выполнение.

Код метода main

int main(int argc, char\* argv[]) {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int rank, size;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (argc > 1) {

int task = atoi(argv[1]);

switch (task) {

case 15:

task15(rank, size);

break;

case 16:

task16(rank, size);

break;

case 17:

task17(rank, size);

break;

case 18:

task18(rank, size);

break;

case 19:

task19(rank, size);

break;

case 20:

task20(rank, size);

break;

case 21:

task21(rank, size);

break;

case 22:

task22(rank, size);

break;

case 23:

task23(rank, size);

break;

case 24:

task24(rank, size);

break;

case 25:

task25(rank, size);

break;

case 26:

task26(rank, size);

break;

case 27:

task27(rank, size);

break;

case 28:

task28(rank, size);

break;

case 29:

task29(rank, size);

break;

case 30:

task30(rank, size);

break;

case 31:

task31(rank, size, 3);

break;

case 32:

task32(rank, size);

break;

default:

std::cerr << "Неверный номер.\n";

break;

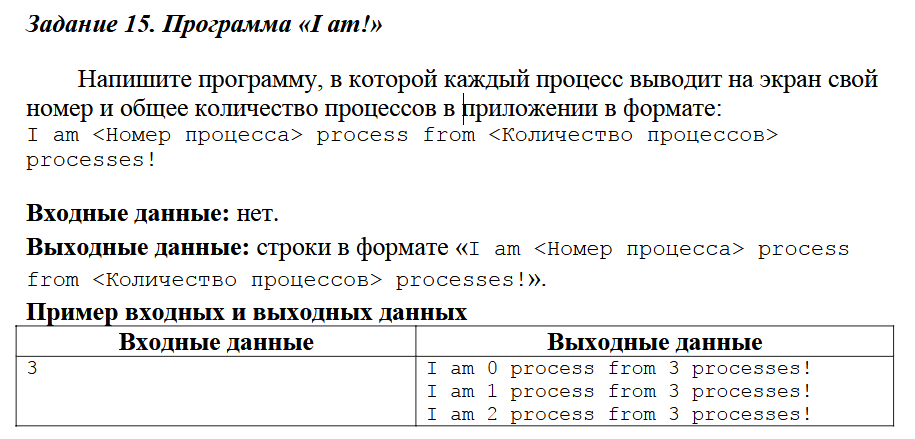
}

}

MPI\_Finalize();

return 0;

}



**Листинг:**

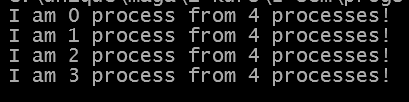
void task15(int rank, int size) {

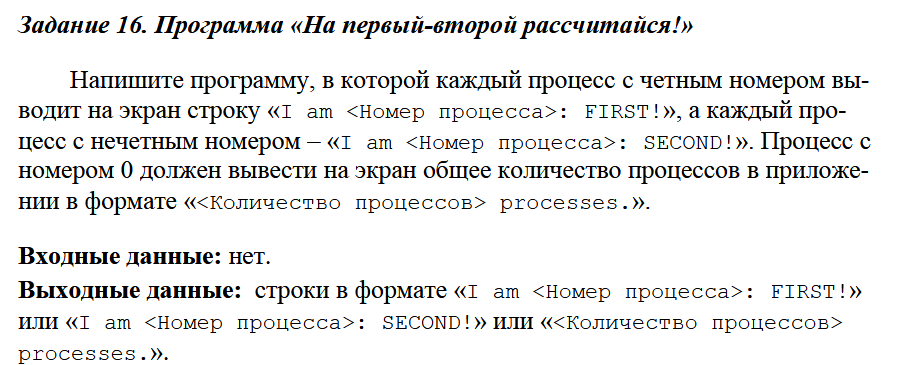
printf("I am %d process from %d processes!\n", rank, size);

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 15





**Листинг:**

void task16(int rank, int size) {

if (rank == 0)

printf("%d processes!\n", size);

else

{

if (rank % 2 == 0)

printf("I am %d: FIRST!\n", rank);

else

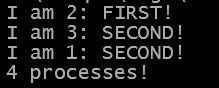
printf("I am %d: SECOND!\n", rank);

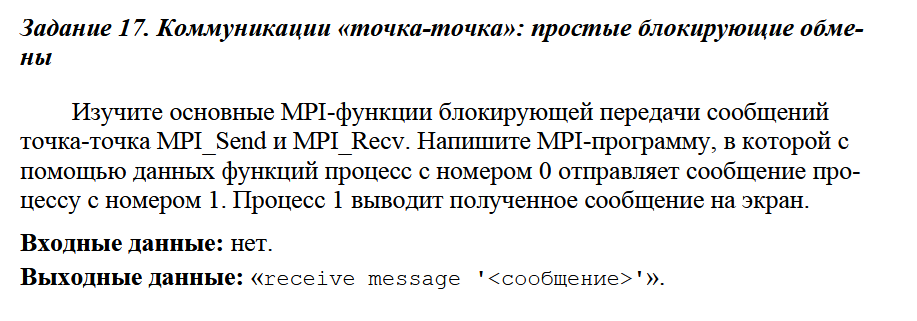
}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 16





**Листинг:**

void task17(int rank, int size) {

if (rank == 0) {

int message = 12;

MPI\_Send(&message, 1, MPI\_INT, 1, 5, MPI\_COMM\_WORLD);

}

else {

MPI\_Status status;

int message;

MPI\_Recv(&message, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

printf("receive message '%d'\n", message);

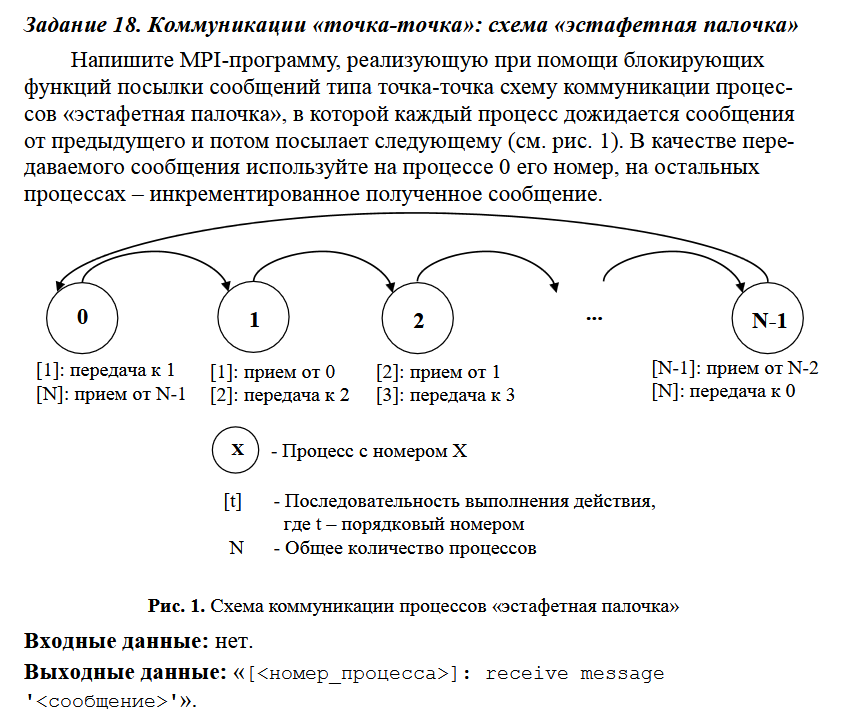
}

}

**Выполнение:**

mpiexec -n 2 mpiconsole.exe 17





**Листинг:**

void task18(int rank, int size) {

if (rank == 0) {

MPI\_Status status;

int procmessage = 0;

MPI\_Ssend(&procmessage, 1, MPI\_INT, 1, 5, MPI\_COMM\_WORLD);

MPI\_Recv(&procmessage, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

printf("[%d]: receive message '%d'\n", rank, procmessage);

}

else {

MPI\_Status status;

int procmessage;

MPI\_Recv(&procmessage, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

printf("[%d]: receive message '%d'\n", rank, procmessage);

procmessage++;

if (rank == size - 1) {

MPI\_Ssend(&procmessage, 1, MPI\_INT, 0, 5, MPI\_COMM\_WORLD);

}

else {

MPI\_Ssend(&procmessage, 1, MPI\_INT, procmessage + 1, 5, MPI\_COMM\_WORLD);

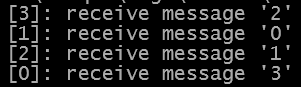
}

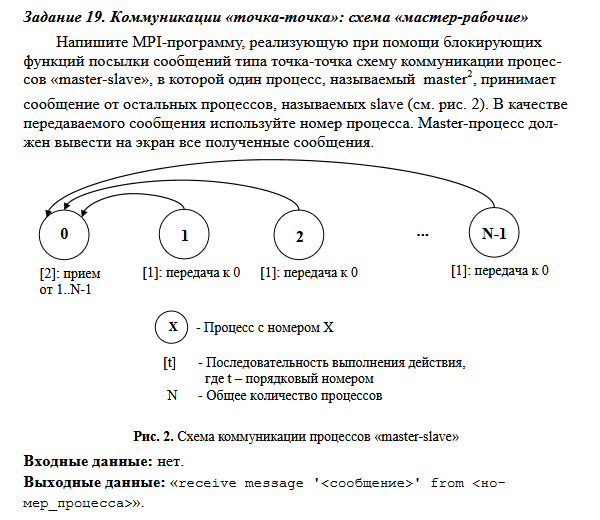
}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 18





**Листинг:**

void task19(int rank, int size) {

if (rank == 0) {

MPI\_Status status;

int procmessage;

for(int i = 1; i < size; i++) {

MPI\_Recv(&procmessage, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

printf("receive message '%d'\n", procmessage);

}

}

else {

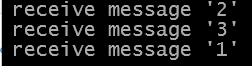
MPI\_Send(&rank, 1, MPI\_INT, 0, rank, MPI\_COMM\_WORLD);

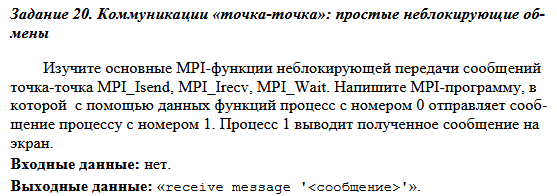
}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 19





**Листинг:**

void task20(int rank, int size) {

if (rank == 0) {

MPI\_Request send\_request;

MPI\_Status status;

int message = 12;

MPI\_Isend(&message, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, &send\_request);

}

else {

int message;

MPI\_Status status;

MPI\_Request recv\_request;

MPI\_Irecv(&message, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

printf("receive message '%d'\n", message);

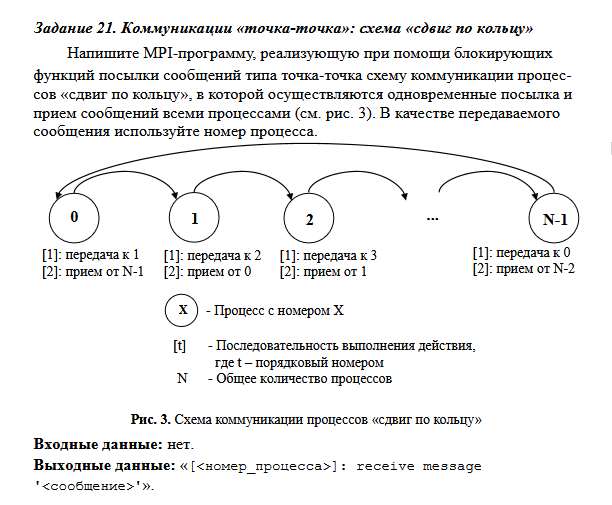
}

}

**Выполнение:**

mpiexec -n 2 mpiconsole.exe 20





**Листинг:**

void task21(int rank, int size) {

MPI\_Request send\_request, recv\_request;

MPI\_Status status;

int message = rank;

if (rank == size - 1)

MPI\_Isend(&message, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &send\_request);

else

MPI\_Isend(&message, 1, MPI\_INT, rank + 1, rank, MPI\_COMM\_WORLD, &send\_request);

if (rank == 0)

MPI\_Irecv(&message, 1, MPI\_INT, size - 1, 0, MPI\_COMM\_WORLD, &recv\_request);

else

MPI\_Irecv(&message, 1, MPI\_INT, rank - 1, rank - 1, MPI\_COMM\_WORLD, &recv\_request);

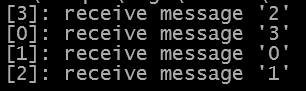
MPI\_Wait(&recv\_request, &status);

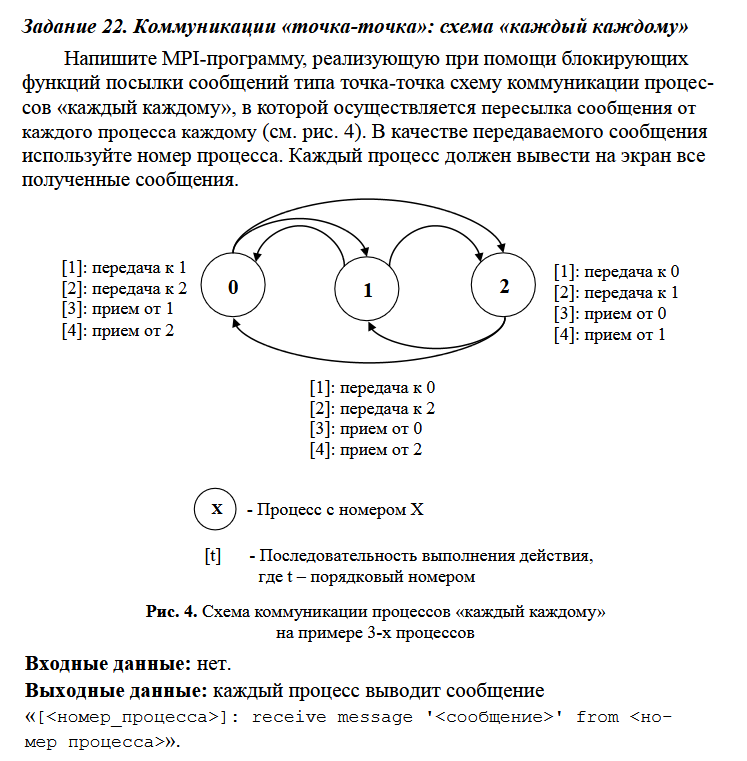
printf("receive message '%d'\n", message);

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 21





**Листинг:**

void task22(int rank, int size) {

MPI\_Request send\_request, recv\_request;

MPI\_Status status;

int message;

switch (rank) {

case 1:

MPI\_Isend(&rank, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &send\_request);

MPI\_Isend(&rank, 1, MPI\_INT, 2, 2, MPI\_COMM\_WORLD, &send\_request);

MPI\_Irecv(&message, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

printf("[%d]; receive message '%d'\n", rank, message);

MPI\_Irecv(&message, 1, MPI\_INT, 2, 1, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

printf("[%d]; receive message '%d'\n", rank, message);

break;

case 2:

MPI\_Isend(&rank, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &send\_request);

MPI\_Isend(&rank, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &send\_request);

MPI\_Irecv(&message, 1, MPI\_INT, 1, 2, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

printf("[%d]; receive message '%d'\n", rank, message);

MPI\_Irecv(&message, 1, MPI\_INT, 0, 2, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

printf("[%d]; receive message '%d'\n", rank, message);

break;

case 0:

MPI\_Isend(&rank, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, &send\_request);

MPI\_Isend(&rank, 1, MPI\_INT, 2, 2, MPI\_COMM\_WORLD, &send\_request);

MPI\_Irecv(&message, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

printf("[%d]; receive message '%d'\n", rank, message);

MPI\_Irecv(&message, 1, MPI\_INT, 2, 0, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

printf("[%d]; receive message '%d'\n", rank, message);

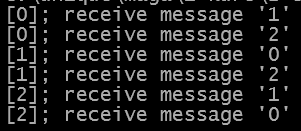
break;

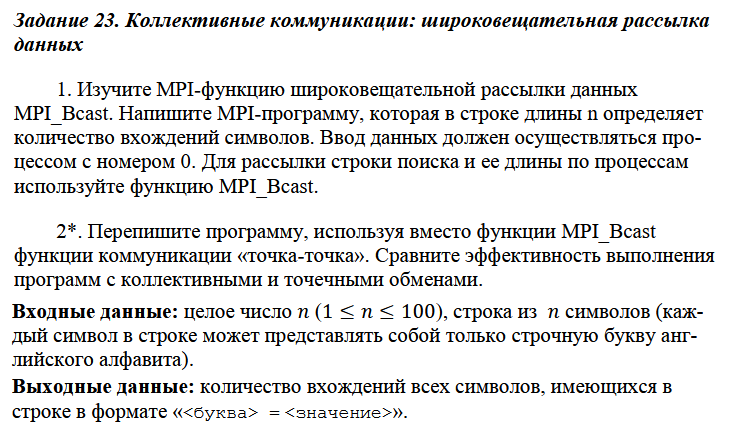
}

}

**Выполнение:**

mpiexec -n 3 mpiconsole.exe 22





**Листинг:**

void task23(int rank, int size) {

string text;

if (rank == 0) {

cout << "Enter text: ";

cin >> text;

MPI\_Bcast(&text, text.length() + 1, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

}

int count[26] = { 0 };

for (int i = 0; i < text.length(); i++) {

count[text[i] - 'a']++;

}

int total\_count[26] = { 0 };

MPI\_Reduce(count, total\_count, 26, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

cout << "Summarized chars: " << endl;

for (char c = 'a'; c <= 'z'; c++) {

if (total\_count[c - 'a'] > 0) {

cout << c << " = " << total\_count[c - 'a'] << endl;

}

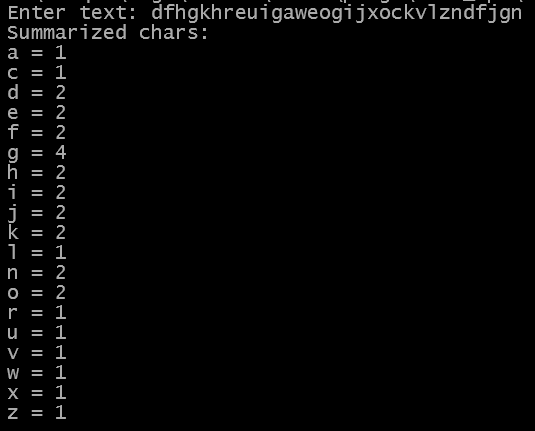
}

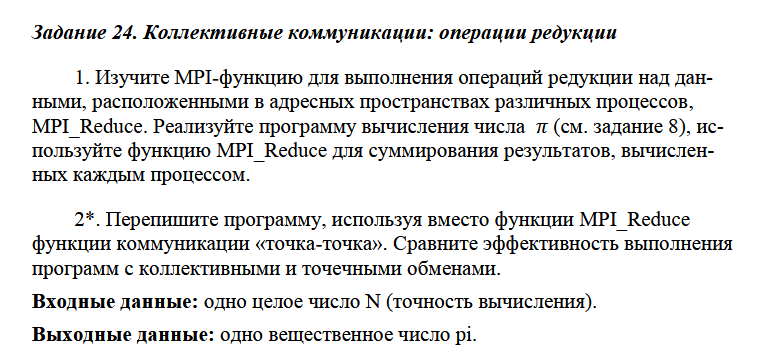
}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 23





**Листинг:**

double CalculatePi(int localStart, int localEnd, unsigned int totalN) {

double sum = 0.0;

double h = 1.0 / totalN;

for (int i = localStart; i < localEnd; i++) {

double xi = (i + 0.5) \* h;

sum += 4.0 / (1.0 + xi \* xi);

}

return sum \* h;

}

void task24(int rank, int size) {

long double massPi = 0.0;

int Nprec = 0;

if (rank == 0) {

cout << "Enter precision: ";

cin >> Nprec;

if (Nprec <= 0) {

cerr << "Precision must be a positive integer." << endl;

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

MPI\_Bcast(&Nprec, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

}

else {

MPI\_Bcast(&Nprec, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

}

if (Nprec < size) {

cerr << "Precision must be greater than or equal to the number of processes." << endl;

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

int localNprec = Nprec / size;

int localStart = rank \* localNprec;

int localEnd = (rank + 1) \* localNprec;

long double localPartialPiSum = CalculatePi(localStart, localEnd, Nprec);

long double globalSum = 0.0;

MPI\_Reduce(&localPartialPiSum, &globalSum, 1, MPI\_LONG\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

cout << std::fixed << std::setprecision(15);

cout << "pi: " << globalSum << endl;

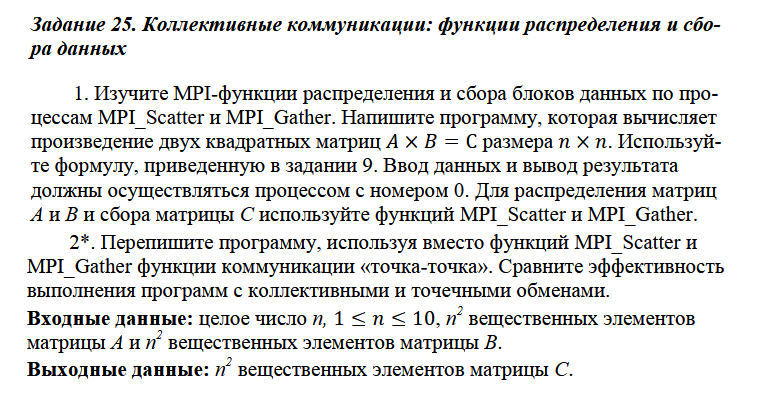
}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 24





**Листинг:**

void task25(int rank, int size) {

int n;

if (rank == 0) {

std::cout << "Input size of matrix: ";

std::cin >> n;

}

MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (n % size != 0) {

if (rank == 0) {

std::cerr << "ERROR! The size of the matrix should be a multiple of the number of processes." << std::endl;

}

}

int rows\_per\_process = n / size;

std::vector<int> A(n \* n), B(n \* n), C(n \* n);

std::vector<int> local\_A(rows\_per\_process \* n), local\_C(rows\_per\_process \* n);

if (rank == 0) {

cout << "A elems: " << endl;

for (int i = 0; i < n \* n; i++) {

cin >> A[i];

}

cout << "B elems :" << endl;

for (int i = 0; i < n \* n; i++) {

cin >> B[i];

}

}

MPI\_Scatter(A.data(), rows\_per\_process \* n, MPI\_INT,

local\_A.data(), rows\_per\_process \* n, MPI\_INT,

0, MPI\_COMM\_WORLD);

MPI\_Bcast(B.data(), n \* n, MPI\_INT, 0, MPI\_COMM\_WORLD);

for (int i = 0; i < rows\_per\_process; ++i) {

for (int j = 0; j < n; ++j) {

local\_C[i \* n + j] = 0;

for (int k = 0; k < n; ++k) {

local\_C[i \* n + j] += local\_A[i \* n + k] \* B[k \* n + j];

}

}

}

MPI\_Gather(local\_C.data(), rows\_per\_process \* n, MPI\_INT,

C.data(), rows\_per\_process \* n, MPI\_INT,

0, MPI\_COMM\_WORLD);

if (rank == 0) {

std::cout << "\nMatrix A:" << std::endl;

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

std::cout << A[i \* n + j] << " ";

}

std::cout << std::endl;

}

std::cout << "\nMatrix B:" << std::endl;

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

std::cout << B[i \* n + j] << " ";

}

std::cout << std::endl;

}

std::cout << "Result matrix C:" << std::endl;

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

std::cout << C[i \* n + j] << " ";

}

std::cout << std::endl;

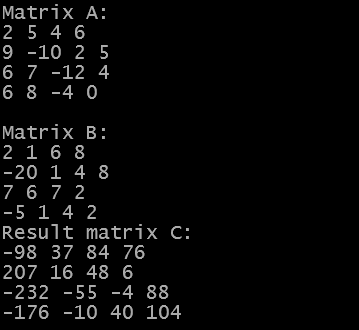
}

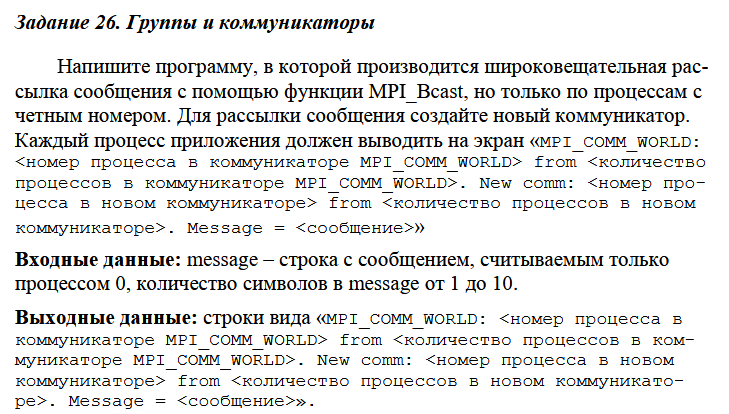
}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 25





**Листинг:**

void task26(int rank, int size) {

char message[11] = "";

MPI\_Comm new\_comm;

int color = rank % 2;

MPI\_Comm\_split(MPI\_COMM\_WORLD, color, rank, &new\_comm);

int new\_rank, new\_size;

MPI\_Comm\_rank(new\_comm, &new\_rank);

MPI\_Comm\_size(new\_comm, &new\_size);

if (rank == 0) {

while (true) {

cout << "Enter message: ";

cin >> message;

if (strlen(message) > 0 && strlen(message) <= 10) {

break;

}

}

}

if (rank % 2 == 0) {

MPI\_Bcast(message, 11, MPI\_CHAR, 0, new\_comm);

cout << "MPI\_COMM\_WORLD: " << rank << " from " << size

<< ". New comm: " << new\_rank << " from " << new\_size

<< ". Message = " << message << endl;

}

else {

cout << "MPI\_COMM\_WORLD: " << rank << " from " << size

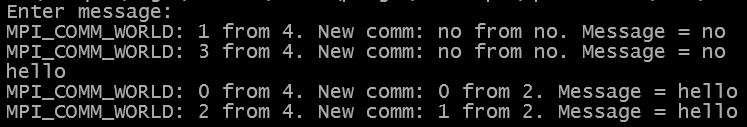
<< ". New comm: no from no. Message = no" << endl;

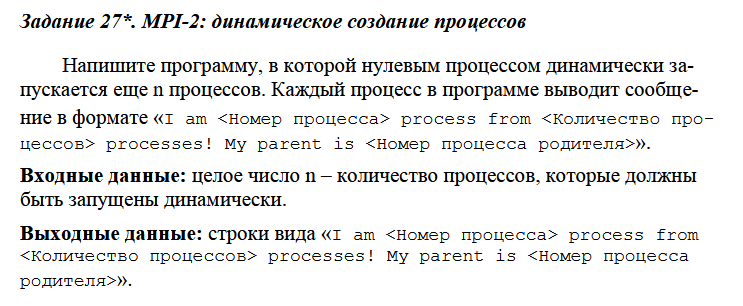
}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 26





**Листинг:**

void task27(int rank, int size) {

if (rank == 0) {

int n;

cout << "Enter the number of processes to spawn: ";

cin >> n;

MPI\_Comm intercomm;

vector<int> spawned\_ranks(n);

MPI\_Comm\_spawn("consoleChild.exe", MPI\_ARGV\_NULL, n, MPI\_INFO\_NULL, 0, MPI\_COMM\_SELF, &intercomm, MPI\_ERRCODES\_IGNORE);

for (int i = 0; i < n; i++) {

spawned\_ranks[i] = i;

}

MPI\_Comm\_free(&intercomm);

}

else {

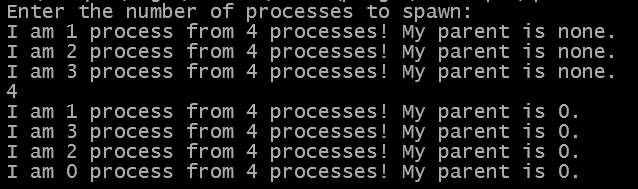
cout << "I am " << rank << " process from " << size << " processes! My parent is none." << endl;

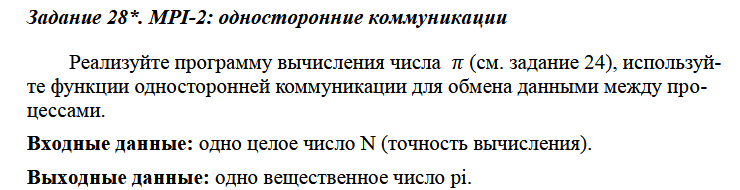
}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 27





**Листинг:**

void task28(int rank, int size) {

double massPi = 0.0;

int Nprec = 0;

int localNprec = 0;

double localPartialPiSum = 0.0;

MPI\_Request send\_request, recv\_request;

MPI\_Status status;

if (rank == 0) {

cout << "Enter precision: ";

cin >> Nprec;

for (int i = 1; i < size; i++) {

MPI\_Isend(&Nprec, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD, &send\_request);

}

}

else {

MPI\_Irecv(&Nprec, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

}

localNprec = Nprec / size;

int localStart = rank \* localNprec;

int localEnd = (rank + 1) \* localNprec;

localPartialPiSum = CalculatePi(localStart, localEnd, Nprec);

if (rank == 0) {

massPi += localPartialPiSum;

for (int i = 1; i < size; i++) {

MPI\_Irecv(&localPartialPiSum, 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD, &recv\_request);

MPI\_Wait(&recv\_request, &status);

massPi += localPartialPiSum;

}

cout << "Final massPi= " << massPi << endl;

}

else {

MPI\_Isend(&localPartialPiSum, 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD, &send\_request);

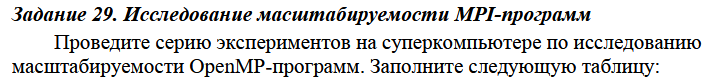
}

}

**Выполнение:**

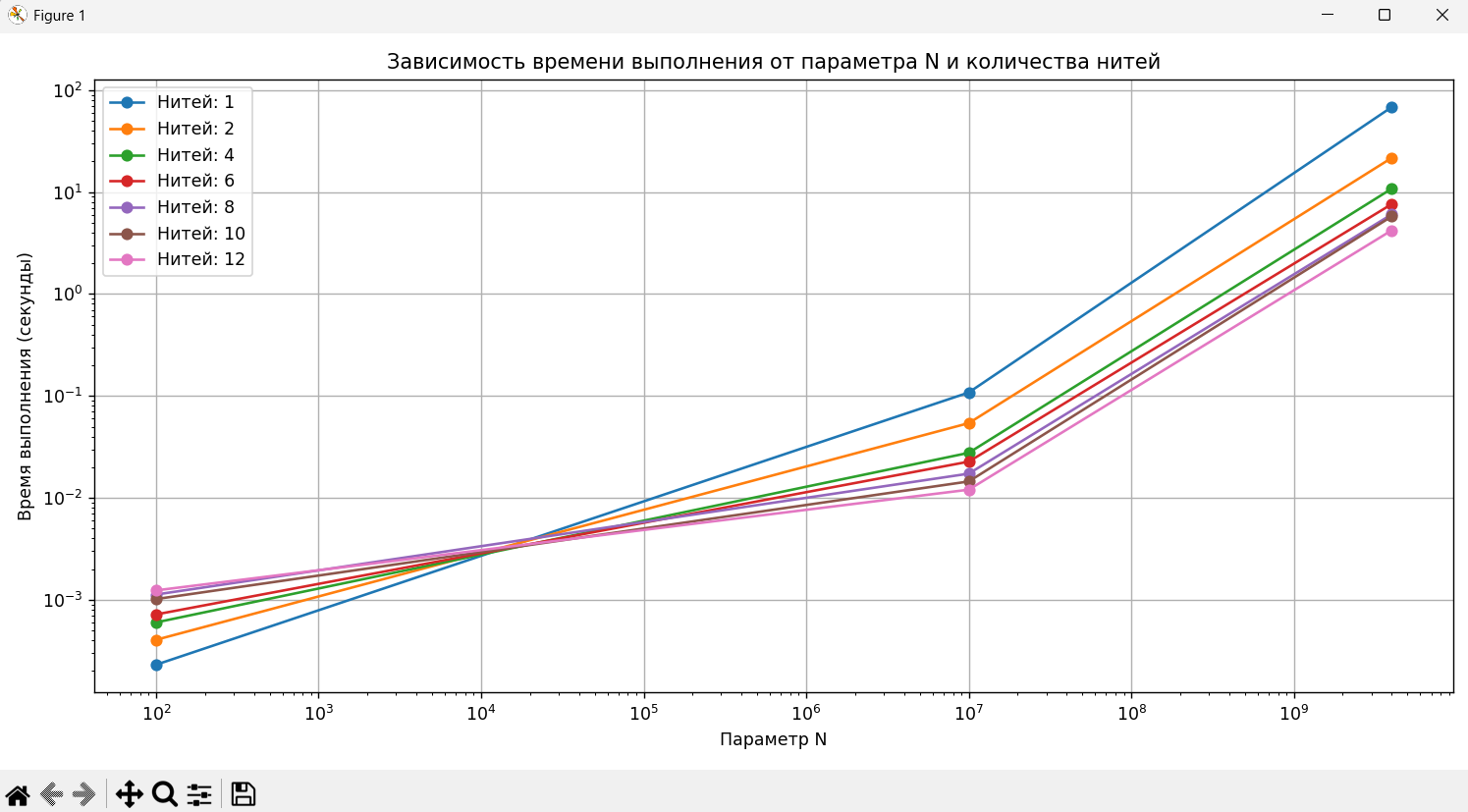
mpiexec -n 4 mpiconsole.exe 28





На основании данных таблицы постройте график масштабируемости для каждого значения параметра N. Определите для каждого графика, при каком количестве нитей достигается максимальное ускорение.

|  |  |  |
| --- | --- | --- |
| Параметр N | Количество нитей | Время выполнения (сек.) |
| 100 | 1 | 0.000230900011957 |
| 10 000 000 | 1 | 0.108121900004335 |
| 4 000 000 000 | 1 | 68.127729012772242 |
| 100 | 2 | 0.000405399943702 |
| 10 000 000 | 2 | 0.054105599992909 |
| 4 000 000 000 | 2 | 21.628441100008786 |
| 100 | 4 | 0.000600300030783 |
| 10 000 000 | 4 | 0.027671099989675 |
| 4 000 000 000 | 4 | 10.793411299935542 |
| 100 | 6 | 0.000719000003301 |
| 10 000 000 | 6 | 0.022676899912767 |
| 4 000 000 000 | 6 | 7.614544700016268 |
| 100 | 8 | 0.001126799965277 |
| 10 000 000 | 8 | 0.017288499977440 |
| 4 000 000 000 | 8 | 6.039494600030594 |
| 100 | 10 | 0.001017199945636 |
| 10 000 000 | 10 | 0.014495700015686 |
| 4 000 000 000 | 10 | 5.765852600103244 |
| 100 | 12 | 0.001237799995579 |
| 10 000 000 | 12 | 0.012035800027661 |
| 4 000 000 000 | 12 | 4.212874599965289 |



Исходя из графика для

N = 100 наибольшая скорость с 1й нитью

N = 10 000 000 наибольшая скорость с 12ю нитями, похожее время с 10ю нитями

N = 4 000 000 000 наибольшая скорость с 12ю нитями

**Листинг:**

void countTimeForPi(int rank, int size, unsigned int Nprec) {

long double massPi = 0.0;

if (Nprec < size) {

cerr << "Precision must be greater than or equal to the number of processes." << endl;

MPI\_Abort(MPI\_COMM\_WORLD, 1);

}

unsigned int localNprec = Nprec / size;

unsigned int remainder = Nprec % size;

if (rank < remainder) {

localNprec += 1;

}

unsigned int localStart = rank \* (Nprec / size) + (rank < remainder ? rank : remainder);

unsigned int localEnd = localStart + localNprec;

long double localPartialPiSum = CalculatePi(localStart, localEnd, Nprec);

long double globalSum = 0.0;

MPI\_Reduce(&localPartialPiSum, &globalSum, 1, MPI\_LONG\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

cout << std::fixed << std::setprecision(15);

cout << "pi: " << globalSum << endl;

}

}

void task29(int rank, int size) {

//unsigned int Nprec = 100;

//unsigned int Nprec = 10000000;

unsigned int Nprec = 4000000000;

MPI\_Bcast(&Nprec, 1, MPI\_UNSIGNED, 0, MPI\_COMM\_WORLD);

double startTime = MPI\_Wtime();

countTimeForPi(rank, size, Nprec);

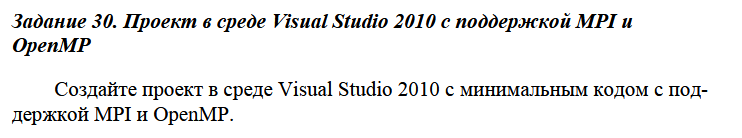
double endTime = MPI\_Wtime();

if (rank == 0) {

cout << "Total time taken: " << (endTime - startTime) << " seconds" << endl;

}

}



**Листинг:**

void task30(int rank, int size) {

#pragma omp parallel for

for (int i = 0; i < 2; ++i) {

std::lock\_guard<std::mutex> lock(cout\_mutex);

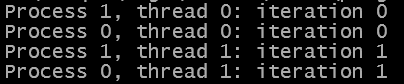
std::cout << "Process " << rank << ", thread " << omp\_get\_thread\_num() << ": iteration " << i << std::endl;

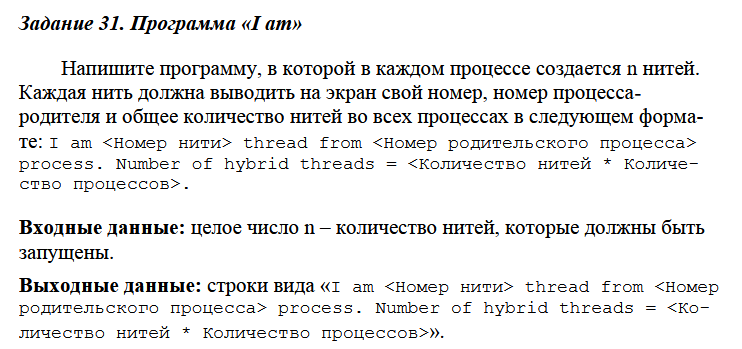
}

}

**Выполнение:**

mpiexec -n 2 mpiconsole.exe 30





**Листинг:**

void task31(int rank, int size, int n) {

int total\_threads = n \* size;

#pragma omp parallel num\_threads(n)

{

int thread\_id = omp\_get\_thread\_num();

{

std::lock\_guard<std::mutex> lock(cout\_mutex);

std::cout << "I am " << thread\_id << " thread from " << rank

<< " process. Number of hybrid threads = " << total\_threads << std::endl;

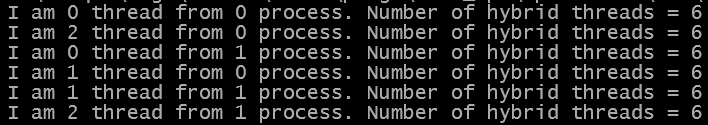
}

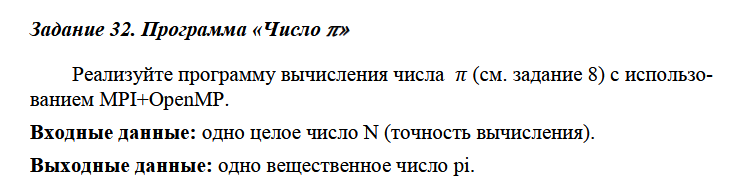
}

}

**Выполнение:**

mpiexec -n 2 mpiconsole.exe 31





**Листинг:**

void task32(int rank, int size) {

unsigned int Nprec;

if (rank == 0) {

std::cout << "Enter precision: ";

std::cin >> Nprec;

}

MPI\_Bcast(&Nprec, 1, MPI\_UNSIGNED, 0, MPI\_COMM\_WORLD);

int localN = Nprec / size;

int localStart = rank \* localN;

int localEnd = (rank + 1) \* localN;

if (rank == size - 1) {

localEnd = Nprec;

}

long double localSum = 0.0;

#pragma omp parallel

{

long double threadSum = 0.0;

long double h = 1.0 / Nprec;

#pragma omp for

for (int i = localStart; i < localEnd; i++) {

long double xi = (i + 0.5) \* h;

threadSum += 4.0 / (1.0 + xi \* xi);

}

#pragma omp atomic

localSum += threadSum;

}

localSum \*= (1.0 / Nprec);

long double totalSum = 0.0;

MPI\_Reduce(&localSum, &totalSum, 1, MPI\_LONG\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

std::cout << std::setprecision(15) << "Calculated value of pi: " << totalSum << std::endl;

}

}

**Выполнение:**

mpiexec -n 4 mpiconsole.exe 32

