**Процессор:**

AMD Ryzen 5 5600H with Radeon Graphics, 3301 МГц,   
ядер: 6, логических процессоров: 12

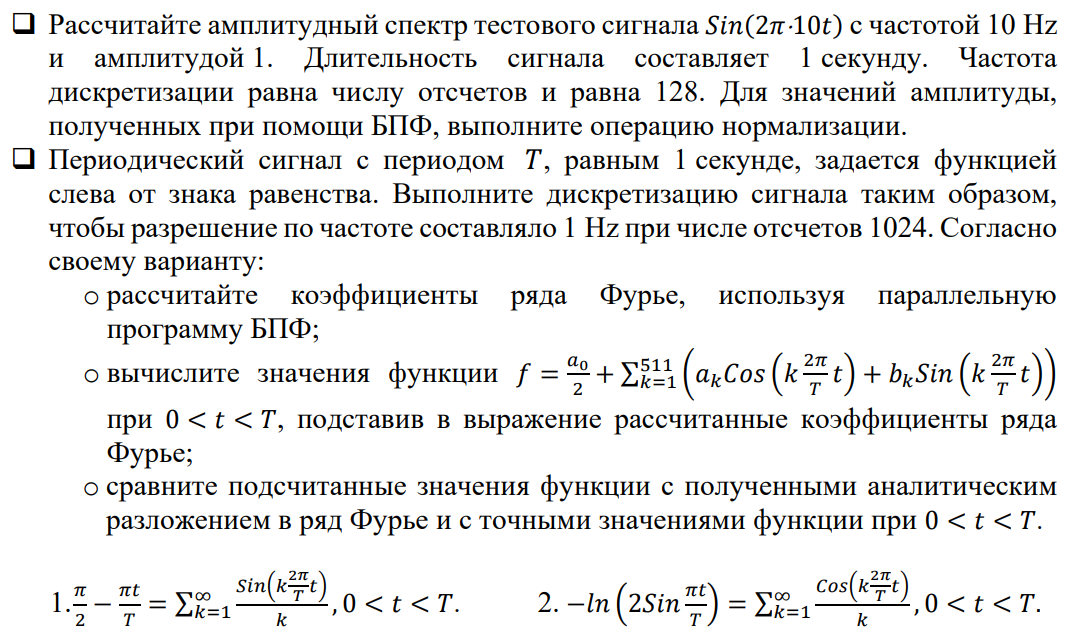
**Память:** 16,0 ГБ (доступно: 15,4 ГБ)

**Задание 18**

Аналогично работе с OMP выполните следующее задание через MPI.

**Предподготовка.**

*Вариант 1.*

****

**Код предподговки:**

void Pretreatment(int NProc, int ProcId) {

int count = 128;

double eps = 1e-6;

complex<double>\* inputSignal = new complex<double>[count];

complex<double>\* outputSignal = new complex<double>[count];

double t;

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

t = 1.0 \* i / count;

inputSignal[i] = complex<double>(sin(2.0 \* PI \* 10 \* t), 0);

}

//SerialFFT(inputSignal, outputSignal, count);

ParallelFFT(inputSignal, outputSignal, count, NProc, ProcId);

if (ProcId == 0) {

cout << "Pretreatment result:" << endl;

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

if (abs(outputSignal[i].real()) > eps || abs(outputSignal[i].imag()) > eps) {

cout << i << " " << outputSignal[i] << endl;

}

}

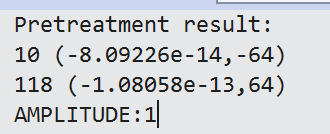
double norm = sqrt(outputSignal[10].imag() \* outputSignal[10].imag() +

outputSignal[10].real() \* outputSignal[10].real());

cout << "AMPLITUDE:" << norm / count \* 2;

}

}



**Вариант 1:**

double f(complex<double>\* signal, int size, double t) {

double res = signal[0].real() / 2.0;

for (int i = 1; i < 512; i++) {

res += signal[i].real() \* cos(i \* 2.0 \* PI \* t / T)

- signal[i].imag() \* sin(i \* 2.0 \* PI \* t / T);

}

return res;

}

double StandardSum(double t) {

//сумма ряда

double eps = 1e-9;

double res = 0;

double s = 0;

int k = 1;

do {

s = sin(k \* 2.0 \* PI \* t / T) / k;

res += s;

k++;

} while (fabs(s) > eps);

return res;

}

void var1(int NProc, int ProcId) {

int count = 1024;

complex<double>\* inputSignal = new complex<double>[count];

complex<double>\* outputSignal = new complex<double>[count];

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

double t = i \* 1.0 / count;

inputSignal[i] = complex<double>((PI / 2 - PI \* t / T), 0);

}

//SerialFFT(inputSignal, outputSignal, count);

ParallelFFT(inputSignal, outputSignal, count, NProc, ProcId);

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

outputSignal[i] = outputSignal[i] / ((double)count / 2.);

}

if (ProcId == 0) {

cout << left << setw(10) << "Function" << " "

<< setw(10) << "Fourier" << " " << setw(10) << "Exact value" << endl;

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

double t = i \* 1.0 / count;

cout << setw(10) << f(outputSignal, count, t) << " "

<< setw(10) << StandardSum(t) << " " << setw(10) << (PI / 2 - PI \* t / T) << endl;

}

}

}

int main()

{

MPI\_Init(NULL, NULL);

int NProc, ProcId;

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

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

if (ProcId == 0) {

Pretreatment(NProc, ProcId);

cout << endl;

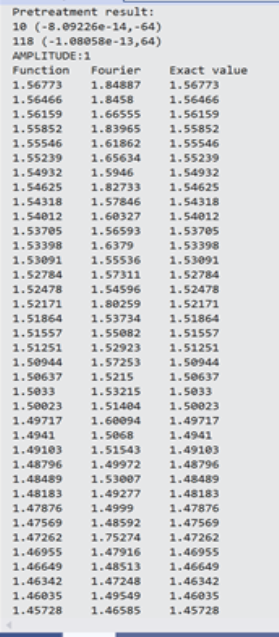
var1(NProc, ProcId);

}

MPI\_Finalize();

return 0;

}

****