Национальный технический университет
«Киевский политехнический институт»
Факультет информатики и вычислительной техники
Кафедра Вычислительной Техники

Расчетно-графическая работа

По курсу: Параллельное программирование

студента III-го курса,
группы ИВ-93
Свинарчука С.В.

Киев 2011 г.

Задание: сравнить скорость выполнение параллельных процессов созданных с помощью библиотек Mpi и OpenMp.

В данной роботе создавалось 4 потока, которые выполнялись параллельно на компьютере с 4 ядрами и засекалось время выполнения программы. Время выполнения программы при использовании разного количества ядер представлены в таблицах. В приложениях представлены исходные коды программ.

OpenMP				
Размер матрицы	Количество ядер			
	1	2	3	4
1000	39	22	20	11
2000	287	164	151	87
3000	1112	624	616	362
4000	6981	3744	3721	2244

Время работы программы при использовании библиотеки OpenMP

OpenMP				
Размер матрицы	Количество ядер			
	1	2	3	4
1000	1	1,77	1,95	3,54
2000	1	1,75	1,90	3,29
3000	1	1,78	1,80	3,07
4000	1	1,86	1,87	3,11

Коэффициент ускорения при использовании библиотеки OpenMP

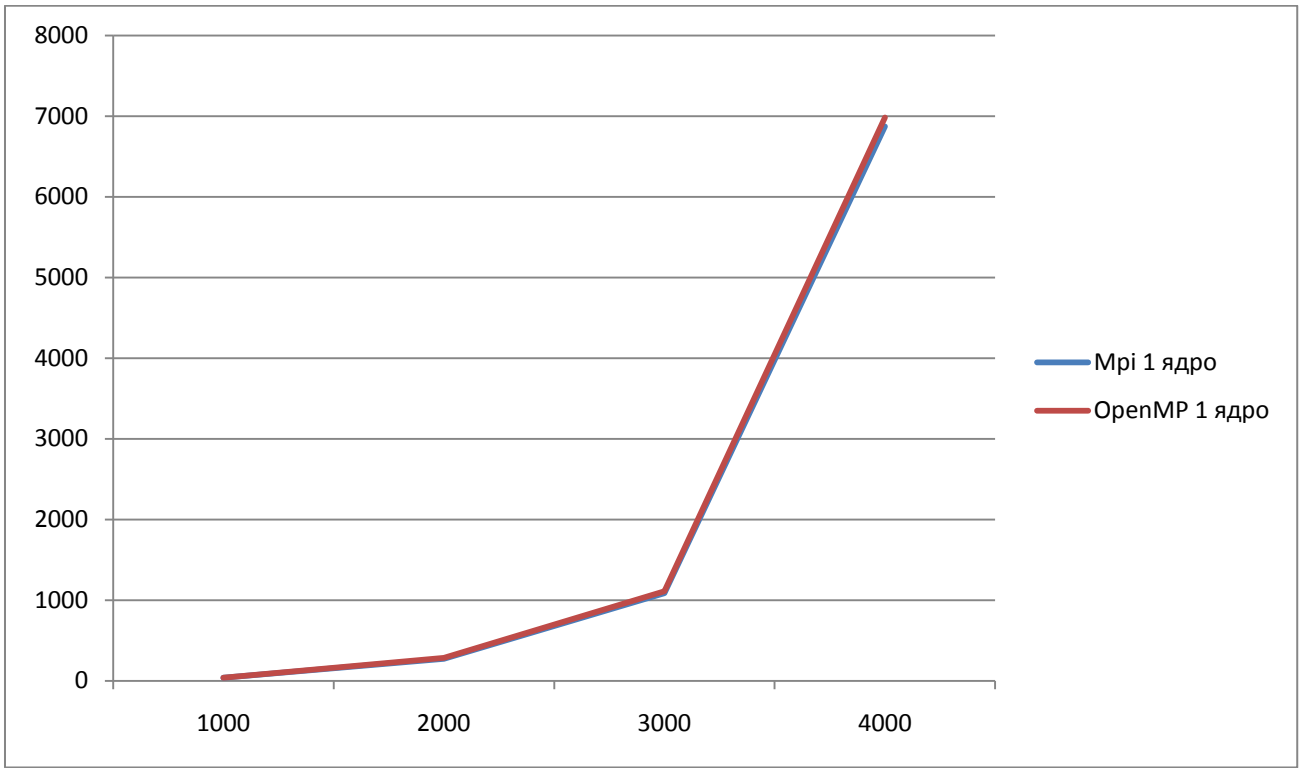
Mpi				
Размер матрицы	Количество ядер			
	1	2	3	4
1000	39	20	20	11
2000	274	161	147	78
3000	1085	621	608	354
4000	6872	3624	3612	2102

Время работы программы при использовании библиотеки Mpi

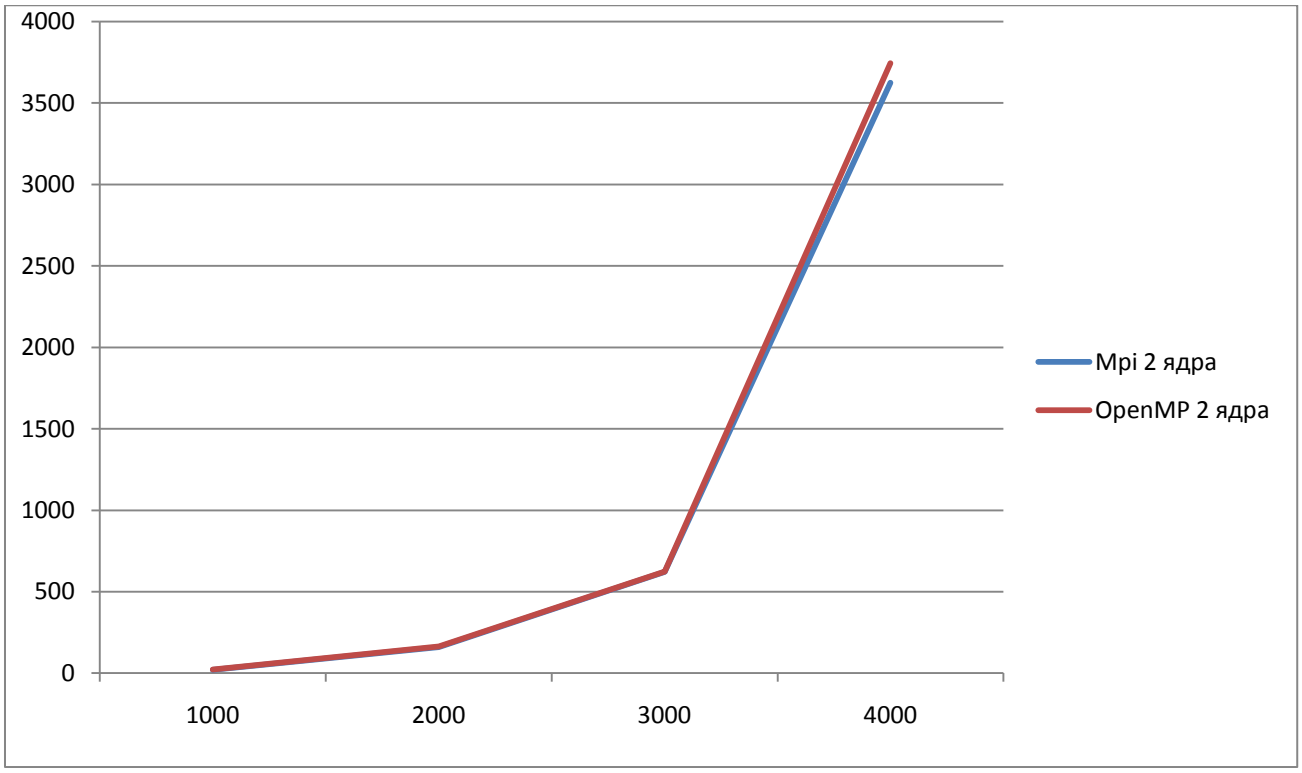
Mpi				
Размер матрицы	Количество ядер			
	1	2	3	4
1000	1	1,95	1,95	3,54
2000	1	1,76	1,87	3,51
3000	1	1,80	1,81	3,06
4000	1	1,89	1,90	3,26

Коэффициент ускорения при использовании библиотеки Mpi

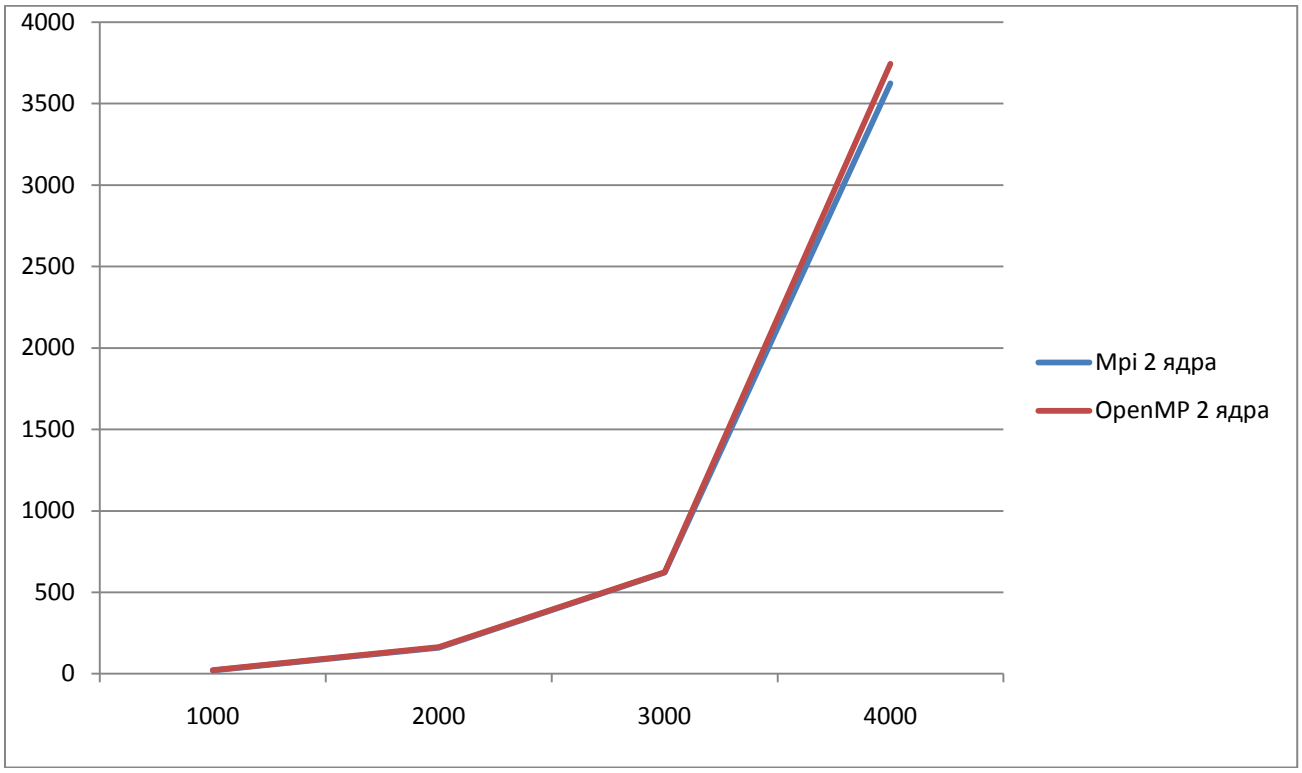
Диаграммы времени для библиотек Mpi и OpenMP



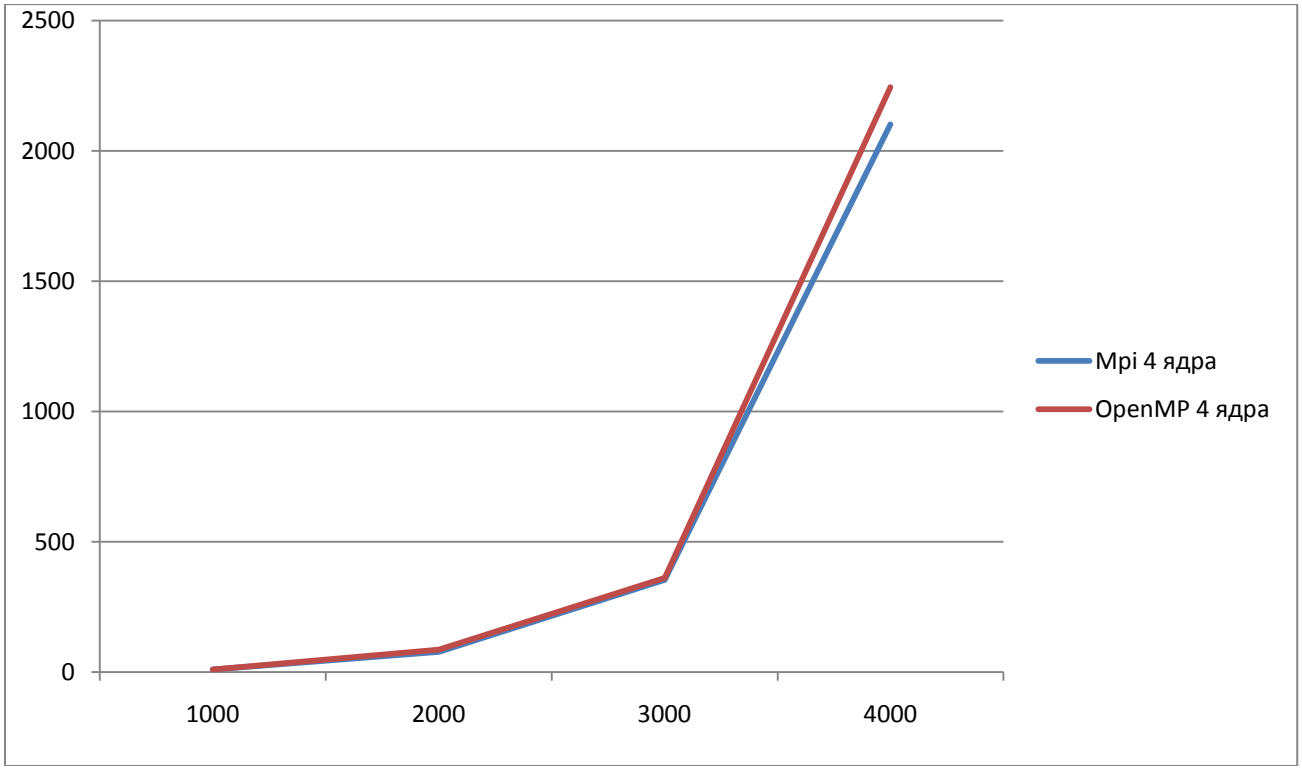
Время выполнения программы Mpi и OpenMP на 1 ядре



Время выполнения программы Mpi и OpenMP на 2 ядрах



Время выполнения программы Mpi и OpenMP на 3 ядрах



Время выполнения программы Mpi и OpenMP на 4 ядрах

Вывод: в результате выполнения расчетно-графической работы было показано что параллельные программы , которые написаны с помощью библиотеки Mpi выполняются немного быстрей чем программы на OpenMP.

Приложение А:
Исходный код программы с использованием библиотеки OpenMp

```
/*
Laboratorna robota №8 OpenMP
Svinarchuk Sergey, IO-93
variant:
1.24:E = A + C - B + D
2.03:MC = MA*MB
3.02:B = TRANS(MC)*A
*/

#include <windows.h>
#include <stdio.h>
#include <omp.h>
#include <time.h>

const int size = 800;
const int value = 1;
void VectorIn(int Vector[size]){
	for(int i=0; i<size; i++){
		Vector[i] = value;
	}
}

void MatrixIn(int Matrix[size][size]){
	for(int i=0; i<size; i++){
		for(int j=0; j<size; j++){
			Matrix[i][j] = value;
		}
	}
}
void VectorOut(int Vector[size]){
	for(int i=0; i<size; i++){
		printf("%d ",Vector[i]);
	}
	printf("\n");

}
void MatrixOut(int Matrix[size][size]){
	for(int i=0; i<size; i++){
		for(int j=0; j<size; j++){
			printf("%d ",Matrix[i][j]);
		}
		printf("\n");
	}
}
void Func1(void){
	int MA[size][size], MB[size][size], MC[size][size];
	for(int i=0; i<size; i++){
		for(int j=0; j<size; j++){
			MC[i][j] = 0;
		}
	}
	MatrixIn(MA);
	MatrixIn(MB);
	for(int i=0; i<size; i++){
		for(int j=0; j<size; j++){
			for(int k=0; k<size; k++){
				MC[i][j] += MA[i][k]*MB[k][j];
			}
		}
	}
	if(size<=8){
		Sleep(10);
		MatrixOut(MC);
	}
}
```

```
int main(void){
	time_t t;
	struct tm *t_m;
	t=time(NULL);
	t_m=localtime(&t);
	printf("start time: %d:%d", t_m->tm_sec, t_m->tm_sec);
	#pragma omp parallel num_threads(4)
	{
		#pragma omp section
		{
			Func1();
		}
		#pragma omp section
		{
			Func1();
		}
		#pragma omp section
		{
			Func1();
		}
		#pragma omp section
		{
			Func1();
		}
	}
	t=time(NULL);
	t_m=localtime(&t);
	printf("stop time: %d:%d", t_m->tm_sec, t_m->tm_sec);
	system("pause");
	}
```

Приложение В:

Исходный код программы с использованием библиотеки Mpi

```
/*
Laboratorna robota №8 Mpi
Svinarchuk Sergey, IO-93
variant:
1.24:E = A + C - B + D
2.03:MC = MA*MB
3.02:B = (MC*MB)*A
*/

#include <windows.h>
#include <stdio.h>
#include <mpi.h>
#include <time.h>

const int size = 1000;
const int value = 1;
void VectorIn(int Vector[size]){
	for(int i=0; i<size; i++){
		Vector[i] = value;
	}
}

void MatrixIn(int Matrix[size][size]){
	for(int i=0; i<size; i++){
		for(int j=0; j<size; j++){
			Matrix[i][j] = value;
		}
	}
}
void VectorOut(int Vector[size]){
	for(int i=0; i<size; i++){
		printf("%d ",Vector[i]);
	}
	printf("\n");
```

```
}
void MatrixOut(int Matrix[size][size]){
	for(int i=0; i<size; i++){
		for(int j=0; j<size; j++){
			printf("%d ",Matrix[i][j]);
		}
		printf("\n");
	}
}
void Func1(void){
	int MA[size][size], MB[size][size], MC[size][size];
	for(int i=0; i<size; i++){
		for(int j=0; j<size; j++){
			MC[i][j] = 0;
		}
	}
	MatrixIn(MA);
	MatrixIn(MB);
	for(int i=0; i<size; i++){
		for(int j=0; j<size; j++){
			for(int k=0; k<size; k++){
				MC[i][j] += MA[i][k]*MB[k][j];
			}
		}
	}
	if(size<=8){
		MatrixOut(MC);
	}
}

int main(int args, char* argvs[]){
	time_t t;
	struct tm *t_m;
	t=time(NULL);
	t_m=localtime(&t);
	printf("start time: %d:%d", t_m->tm_sec, t_m->tm_sec);
	int rank;
	MPI_Init(&args, &argvs);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	if(rank == 0 ){
		Func1();
	}
	if(rank == 1 ){
		Func1();
	}
	if(rank == 2 ){
		Func1();
	}
	if(rank == 3 ){
		Func1();
	}
	MPI_Finalize();
	t=time(NULL);
	t_m=localtime(&t);
	printf("stop time: %d:%d", t_m->tm_sec, t_m->tm_sec);
	system("pause");
	return 0;
}
```