Akademia Nauk Stosowanych w Nowym Sączu			
Programowanie współbieżne i rozproszone **PWIR_11**			
Imię i Nazwisko: Karol Wolski		Ocena sprawozdania:	Zaliczenie:
Data: 23.05.2023	Grupa: L3		

Zad 1.

Przeanalizuj załączony kod.

Program wykonuje prostą operację dodawania wektorów wykonywaną przez wiele procesów wykorzystując bibliotekę MPI. W funkcji mainProcess przydzielana jest pamięć dla tablic va, vb i vc o rozmiarze 5 * (size - 1). Wygenerowane zostają losowe wartości i zapisane w va i vb, natomiast vc jest wypełnione zerami. Następnie tablice va i vb są rozgłaszane do wszystkich pozostałych procesów za pomocą MPI_Bcast. Tablice requests i statuses są dynamicznie przydzielane w celu przechowywania obiektów żądań i statusów MPI. Żądania są używane do operacji odbioru bez blokowania. Po zakończeniu operacji odbioru za pomocą MPI_Waitall, zawartość tablic jest wyświetlana. Na końcu pamięć dynamicznie przydzielana jest zwalniana. W funkcji workerProcess przydzielana jest pamięć dla v, va i vb. Tablice va i vb są odbierane za pomocą MPI_Bcast od głównego procesu. Następnie proces roboczy wykonuje dodawanie wektorów poprzez sumowanie odpowiadających elementów z va i vb i zapisuje wynik w v. Następnie tablica v jest wysyłana z powrotem do głównego procesu za pomocą MPI_Send. Na końcu pamięć dynamicznie przydzielana jest zwalniana.

Funkcja main inicjalizuje środowisko MPI za pomocą MPI_Init i pobiera id procesu oraz liczbę procesów. Jeśli identyfikator procesu wynosi 0, wywoływana jest funkcja mainProcess, przekazując łączną liczbę procesów jako argument. W przeciwnym razie, dla procesów roboczych, wywoływana jest funkcja workerProcess, przekazując identyfikator procesu i łączną liczbę procesów jako argumenty. Na końcu jest wykonywane czyszczenie środowiska MPI za pomocą MPI_Finalize.

Zad 2.

Zmodyfikuj załączony kod tak by na jeden proces przypadało więcej elementów do zsumowania (obecnie jest 5).

```cpp
#include "mpi.h"
#include <cstdio>
#include <iostream>
#include <time.h>

void mainProcess(int size) {
    srand(time(NULL));

    //alokujemy wektory o rozmiarze(5*(ilosc procesów-1))
    unsigned int* va = new unsigned int[10 * (size - 1)];
    unsigned int* vb = new unsigned int[10 * (size - 1)];
    unsigned int* vc = new unsigned int[10 * (size - 1)];

    //wypełniamy a i b losowymi danymi, a vc zerujemy
    for (unsigned int i = 0; i < 10 * (size - 1); i++) {
        va[i] = rand() % 10;
        vb[i] = rand() % 10;
        vc[i] = 0;
    }

    //broadcastujemy wektor a do pozostałych procesów
    MPI_Bcast(va, 10 * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    //broadcastujemy wektor b do pozostałych procesów
    MPI_Bcast(vb, 10 * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    //odpalamy nasłuch
    MPI_Request* requests = new MPI_Request[size - 1];
    MPI_Status* statuses = new MPI_Status[size - 1];
    for (unsigned int i = 0; i < size-1; i++) {
        MPI_Irecv(vc + i * 10, 10, MPI_UNSIGNED, i + 1, 0, MPI_COMM_WORLD, &requests[i]);
    }
    MPI_Waitall(size - 1, requests, statuses);

    //wypisujemy wyniki
    for (unsigned int i = 0; i < (10 * (size - 1)); i++) printf("%d\t", va[i]);
    printf("\r\n");
    for (unsigned int i = 0; i < (10 * (size - 1)); i++) printf("%d\t", vb[i]);
    printf("\r\n");
    for (unsigned int i = 0; i < (10 * (size - 1)); i++) printf("%d\t", vc[i]);
    printf("\r\n");

    //zwalniamy pamięć
    delete[] va;
    delete[] vb;
    delete[] vc;
    delete[] requests;
    delete[] statuses;
}
```

```cpp
void workerProcess(int id, int size) {
    //alokujemy buffor na moją część zadania
    unsigned int* v = new unsigned int[10];

    //alokujemny miejsce na wektor a oraz b
    unsigned int* va = new unsigned int[10 * (size - 1)];
    unsigned int* vb = new unsigned int[10 * (size - 1)];

    //nasłuchujemy bcasta wektora a
    MPI_Bcast(va, 10 * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    //nasłuchujemy bcasta wektora b
    MPI_Bcast(vb, 10 * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    //liczymy sumę
    for (unsigned int i = 0; i < 10; i++) {
        v[i] = va[(id - 1) * 10 + i] + vb[(id - 1) * 10 + i];
    }

    //odsyłamy wynik
    MPI_Send(v, 10, MPI_UNSIGNED, 0, 0, MPI_COMM_WORLD);

    //zwalniamy pamięc
    delete[] v;
    delete[] va;
    delete[] vb;
}

int main()
{
    int PID, PCOUNT;

    MPI_Init(NULL, NULL);

    MPI_Comm_rank(MPI_COMM_WORLD, &PID);
    MPI_Comm_size(MPI_COMM_WORLD, &PCOUNT);

    if (PID == 0) { //jestem procesem głównym
        mainProcess(PCOUNT);
    }
    else { //jestem procesem roboczym
        workerProcess(PID, PCOUNT);
    }

    MPI_Finalize();

    return 0;
}
```

Zad 3.

Zmodyfikuj załączony program tak, by rozmiar wektorów był pobierany od użytkownika. Program musi wyliczyć jaka część danych przypada na każdy proces, oraz przekazać każdemu z nich tę informację.

```cpp
#include "mpi.h"
#include <cstdio>
#include <iostream>
#include <time.h>

void mainProcess(int size) {
    srand(time(NULL));

    unsigned int vectorSize;
    std::cout << "Podaj rozmiar wektora: ";
    std::cin >> vectorSize;

    // Przekazanie rozmiaru wektora do procesów roboczych
    MPI_Bcast(&vectorSize, 1, MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // Alokacja wektorów
    unsigned int* va = new unsigned int[vectorSize * (size - 1)];
    unsigned int* vb = new unsigned int[vectorSize * (size - 1)];
    unsigned int* vc = new unsigned int[vectorSize * (size - 1)];

    // Inicjalizacja wektorów
    for (unsigned int i = 0; i < vectorSize * (size - 1); i++) {
        va[i] = rand() % 10;
        vb[i] = rand() % 10;
        vc[i] = 0;
    }

    // Broadcast wektora `va` do procesów roboczych
    MPI_Bcast(va, vectorSize * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // Broadcast wektora `vb` do procesów roboczych
    MPI_Bcast(vb, vectorSize * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // Odbieranie wyników od procesów roboczych
    MPI_Request* requests = new MPI_Request[size - 1];
    MPI_Status* statuses = new MPI_Status[size - 1];
    for (unsigned int i = 0; i < size - 1; i++) {
        MPI_Irecv(vc + i * vectorSize, vectorSize, MPI_UNSIGNED, i + 1, 0, MPI_COMM_WORLD, &requests[i]);
    }
    MPI_Waitall(size - 1, requests, statuses);

    // Wypisywanie wyników
    for (unsigned int i = 0; i < (vectorSize * (size - 1)); i++) {
        printf("%d\t", va[i]);
    }
    printf("\r\n");
    for (unsigned int i = 0; i < (vectorSize * (size - 1)); i++) {
        printf("%d\t", vb[i]);
    }
    printf("\r\n");
    for (unsigned int i = 0; i < (vectorSize * (size - 1)); i++) {
        printf("%d\t", vc[i]);
    }
    printf("\r\n");

    // Zwalnianie pamięci
    delete[] va;
    delete[] vb;
    delete[] vc;
    delete[] requests;
```

```
    delete[] statuses;
}

void workerProcess(int id, int size) {
    unsigned int vectorSize;

    // Odbieranie rozmiaru wektora od procesu głównego
    MPI_Bcast(&vectorSize, 1, MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // Alokacja wektorów
    unsigned int* va = new unsigned int[vectorSize * (size - 1)];
    unsigned int* vb = new unsigned int[vectorSize * (size - 1)];

    // Odbieranie wektorów `va` i `vb` od procesu głównego
    MPI_Bcast(va, vectorSize * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);
    MPI_Bcast(vb, vectorSize * (size - 1), MPI_UNSIGNED, 0, MPI_COMM_WORLD);

    // Obliczanie sumy
    unsigned int* v = new unsigned int[vectorSize];
    for (unsigned int i = 0; i < vectorSize; i++) {
        v[i] = va[(id - 1) * vectorSize + i] + vb[(id - 1) * vectorSize + i];
    }

    // Wysyłanie wyniku do procesu głównego
    MPI_Send(v, vectorSize, MPI_UNSIGNED, 0, 0, MPI_COMM_WORLD);

    // Zwalnianie pamięci
    delete[] va;
    delete[] vb;
    delete[] v;
}



int main()
{
    int PID, PCOUNT;

    MPI_Init(NULL, NULL);

    MPI_Comm_rank(MPI_COMM_WORLD, &PID);
    MPI_Comm_size(MPI_COMM_WORLD, &PCOUNT);

    if (PID == 0) { //jestem procesem głównym
        mainProcess(PCOUNT);
    }
    else { //jestem procesem roboczym
        workerProcess(PID, PCOUNT);
    }

    MPI_Finalize();

    return 0;
}
```

```
PS C:\Users\200iq\source\repos\PWIR_11_\x64\Debug> mpiexec -n 10 MPI_HELLOWORLD.exe
Podaj rozmiar wektora: 10
2       5       4       5       7       0       2       2       2       8       9       3       0       7       2       4       8       0       5       4       3       4       0       6
1       2       6       8       2       0       4       3       1       3       5       0       9       6       6       5       5       5       8       7       5       0       3       2
1       2       6       8       2       0       4       3       1       3       5       0       9       6       6       5       5       5       8       7       5       0       3       2
3       7       10      13      9       0       6       5       3       11      14      3       9       13      8       9       13      5       13      11      8       4       3       1
1       11      5       15      8       2       13      4       10      9       17      3       4       12      8       13      16      14      8       5       3       13      1       1
4       12      9       12      14      11      6       14      13      11      6       6       3       12      0       8       17      9       8       11      5       16      6       1
0       2       5       15      9       10      7       15      8
PS C:\Users\200iq\source\repos\PWIR_11_\x64\Debug>
```

W tym programie została dodana interakcja z użytkownikiem, aby pobrać rozmiar wektora od niego. Następnie rozmiar ten jest rozgłaszany (MPI_Bcast) do wszystkich procesów. Na podstawie rozmiaru wektora i liczby procesów, obliczamy, jaką część danych powinien przetwarzać każdy proces (chunkSize). Informacja ta jest wysyłana do każdego procesu za pomocą MPI_Send. W funkcji workerProcess procesy odbierają informację o rozmiarze części danych za pomocą MPI_Recv.