Michał Dybaś

Temat projektu:

Program, który oblicza iloczyn skalarny n-elementowych wektorów rzeczywistych równolegle w środowisku MPI. Program wykorzystuje dwie architektury środowiska MPI.

- Dla środowiska gdzie nie została ustalona relacja pomiędzy procesami
- Dla środowiska w którym relacja pomiędzy procesami (procesorami) została ustalona przy pomocy funkcji MPI_Graph_Create, a zdefiniowana struktura ma postać pierścienia (architektura Hypercube d=3)

Spis treści

1. Kod źródłowy środowiska MPI projekt_1.c, w którym nie została ustalona relacja między procesami.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>

//  • Dla środowiska gdzie nie została ustalona relacja pomiędzy procesami
//  kompilacja   mpicc -o projekt_1 projekt_1.c
//  uruchamianie mpirun --
host localhost:liczba_procesorów/wątków ./projekt_1

int main(int argc, char ** argv)
{
    //Deklaracja i incjalizacja zmienych
    double *vt_1, *vt_1_part, *vt_2, *vt_2_part, *scores, score = 0; // Wsk
aźniki do tablic liczb typu double (nasze wektory i pod_wektory) i tablicy
na wyniki oraz sumy dla każdego procesu z osobna
    int vt_normal_size; // Zmienna przechowująca ilość elementów dla każdeg
o wektora, jest ona inicjalizowana przez użytkownika
    int vt_extend_size; // Zmienna przechowująca ilość elementów (równych 0
), o które został rozszerzony wektor
    int vt_real_size; // Zmienna przechowująca realną ilość elementów dla k
ażdego wektora
    int vt_part_size; // Zmienna przechowująca ilość elementów wektorów jak
i przypada na każdy z procesów
    int rank; // Numer mojego procesu
    int size; // Liczba procesów
    struct timespec start, end; // Zmienna przechowująca czas rozpoczęcia i
 końca obliczeń

    MPI_Init(&argc, &argv); // Rozpoczęcie obliczeń MPI
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Pobranie aktualnego numeru pro
cesu
    MPI_Comm_size(MPI_COMM_WORLD, &size); // Pobranie liczby procesów

    if(rank == 0) // Instrukcje wykonywane dla głównego procesu programu (p
roces root)
    {
        fprintf(stderr, "Podaj długośc wektorów (liczba całkowita większa o
d 0): "); // Wyświetlenie komunikatu z prośbą o podanie długości wektora
        scanf("%d", &vt_normal_size); // Pobranie długości wektów od użytko
wnika

        if(vt_normal_size < 0) // Wyjście z programu dla wektora mniejszego
 niż 1
        {
```

```
            printf("Podany rozmiar wektorów jest zbyt mały, ilość elementów
 jest mniejsza niż ilość procesów: %d.\n", size);
            MPI_Finalize(); // Koniec obliczeń MPI
            exit(1); // Wyjście z programu z kodem błędu
        }

        vt_extend_size = size - (vt_normal_size % size); // Obliczenie o il
e elementów należy rozszerzyć wektor, aby można było podzielić je równo dla
 każdego procesu
        vt_real_size = vt_normal_size + vt_extend_size; // Obliczenie realn
ej długości wektorów
        vt_part_size = vt_real_size / size; // Obliczenie ilości elementów
przypadających na każdy proces
        vt_1 = malloc(vt_real_size * sizeof(double)); // Zaalokowanie pamię
ci na pierwszy wektor
        vt_2 = malloc(vt_real_size * sizeof(double)); // Zaalokowanie pamię
ci na drugi wektor
        vt_1_part = malloc(vt_part_size * sizeof(double)); // Zaalokowanie
pamięci na pierwszy wektor częściowy
        vt_2_part = malloc(vt_part_size * sizeof(double)); // Zaalokowanie
pamięci na drugi wektor częściowy
        scores = malloc(size * sizeof(double)); // Zaalokowanie pamięci na
tablice wyników

        for(int i = 0; i < vt_real_size; i++) // Inicjalizacja wektorów lic
zbami losowymi
        {
            if(i < vt_normal_size) // Wypełnienie elementów wektorów liczba
mi typu rzeczywistego z przedziału (-25, 25)
            {
                srand(time(NULL) + rand());
                vt_1[i] = (double) rand() / RAND_MAX * 50 - 25;
                vt_2[i] = (double) rand() / RAND_MAX * 50 - 25;
            }
            else // Dla dodatkowo wygenerowanych elementów wartość wynosi 0
, aby wynik obliczeń nie został zakłamany
            {
                vt_1[i] = 0;
                vt_2[i] = 0;
            }
        }

        clock_gettime(CLOCK_MONOTONIC_RAW, &start); // Pobranie czasu rozpo
częcia obliczeń
        MPI_Bcast(&vt_real_size, 1, MPI_INT, 0, MPI_COMM_WORLD); // Wysłani
e rozmiaru wektora
        MPI_Bcast(&vt_part_size, 1, MPI_INT, 0, MPI_COMM_WORLD); // Wysłani
e rozmiaru części wektora
```

```
        MPI_Scatter(vt_1, vt_part_size, MPI_DOUBLE, vt_1_part, vt_part_size
, MPI_DOUBLE, 0, MPI_COMM_WORLD); // Wysłanie podzielonego wektora 1 na kaw
ałki do procesów w grupie
        MPI_Scatter(vt_2, vt_part_size, MPI_DOUBLE, vt_2_part, vt_part_size
, MPI_DOUBLE, 0, MPI_COMM_WORLD); // Wysłanie podzielonego wektora 1 na kaw
ałki do procesów w grupie
        MPI_Barrier(MPI_COMM_WORLD); // Bariera synchronizacyjna

        for (int i = 0; i < vt_part_size; i++) // Obliczenie iloczynu częsc
iowych wektorów
        {
            score += (vt_1[i] * vt_2[i]);
        }
        fprintf(stderr, "Iloczyn skalarny procesora %d wynosi: %f\n", rank,
 score); // Wyświetlenie informacji o obliczonym częściowym iloczynie skala
rnym
        MPI_Barrier(MPI_COMM_WORLD); // Bariera synchronizacyjna
        MPI_Gather(&score, 1, MPI_DOUBLE, scores, 1, MPI_DOUBLE, 0, MPI_COM
M_WORLD); // Zebranie wyników z wszystkich procesów grupie

        score = 0; // Wyzerowanie iloczynu skalarnego
        for (int i = 0; i < size; i++) // Połączenie wyników częściowych
        {
            score += scores[i];
        }

        fprintf(stderr, "Iloczyn skalarny wektorów %d-
elementowych dla %d procesorów wynosi: %f\n", vt_normal_size, size, score);
 // Wyświetlenie informacji o obliczonym iloczynie skalarnym

        free(vt_1); // Zwolnienie obszaru pamięci wektora 1
        free(vt_2); // Zwolnienie obszaru pamięci wektora 2
        free(vt_1_part); // Zwolnienie obszaru pamięci wektora częściowego
1
        free(vt_2_part); // Zwolnienie obszaru pamięci wektora częściowego
2
        free(scores); // Zwolnienie obszaru pamięci tablicy wyników
    }
    else // Instrukcje wykonywane przez procesy podległe
    {
        MPI_Barrier(MPI_COMM_WORLD); // Bariera synchronizacyjna
        MPI_Bcast(&vt_real_size, 1, MPI_INT, 0, MPI_COMM_WORLD); // Odebran
ie rozmiaru wektora
        MPI_Bcast(&vt_part_size, 1, MPI_INT, 0, MPI_COMM_WORLD); // Odebran
ie rozmiaru części wektora
        vt_1 = malloc(vt_real_size * sizeof(double)); // Zaalokowanie pamię
ci na pierwszy wektor
        vt_2 = malloc(vt_real_size * sizeof(double)); // Zaalokowanie pamię
ci na drugi wektor
```

```
        vt_1_part = malloc(vt_part_size * sizeof(double)); // Zaalokowanie
pamięci na pierwszy wektor częsciowy
        vt_2_part = malloc(vt_part_size * sizeof(double)); // Zaalokowanie
pamięci na drugi wektor częściowy

        MPI_Scatter(vt_1, vt_part_size, MPI_DOUBLE, vt_1_part, vt_part_size
, MPI_DOUBLE, 0, MPI_COMM_WORLD); // Odebranie podzielonego wektora 1 na ka
wałki
        MPI_Scatter(vt_2, vt_part_size, MPI_DOUBLE, vt_2_part, vt_part_size
, MPI_DOUBLE, 0, MPI_COMM_WORLD); // Odebranie podzielonego wektora 1 na ka
wałki
        for (int i = 0; i < vt_part_size; i++) // Obliczenie iloczynu częsc
iowych wektorów
        {
            score += (vt_1_part[i] * vt_2_part[i]);
        }
        fprintf(stderr, "Iloczyn skalarny procesora %d wynosi: %f\n", rank,
 score); // Wyświetlenie informacji o obliczonym częściowym iloczynie skala
rnym
        MPI_Barrier(MPI_COMM_WORLD); // Bariera synchronizacyjna
        MPI_Gather(&score, 1, MPI_DOUBLE, NULL, 1, MPI_DOUBLE, 0, MPI_COMM_
WORLD); // Wysłanie wyników do procesu zbierającego wyniki

        free(vt_1); // Zwolnienie obszaru pamięci wektora 1
        free(vt_2); // Zwolnienie obszaru pamięci wektora 2
        free(vt_1_part); // Zwolnienie obszaru pamięci wektora częściowego
1
        free(vt_2_part); // Zwolnienie obszaru pamięci wektora częściowego
2
    }

    clock_gettime(CLOCK_MONOTONIC_RAW, &end); // Pobranie informacji o czas
ie zakończenia obliczeń
    MPI_Finalize(); // Koniec obliczeń MPI

    if(rank == 0)
    {
        u_int64_t delta_us = (end.tv_sec - start.tv_sec) * 1000000 + (end.t
v_nsec - start.tv_nsec) / 1000; // Obliczenie różnicy między czasem końca o
bliczeń, a rozpoczęciem w mikrosekundach
        fprintf(stderr, "Obliczenia trwały %6.2f milisekund\n", (double) de
lta_us / 1000); // Wyświetlenie informacji na temat czasu obliczeń
    }

    return 0; // Wyjście z programu
}
```

2. Kod źródłowy środowiska MPI projekt_2.c, w którym została ustalona relacja między procesami

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>

//  • Dla środowiska gdzie nie została ustalona relacja pomiędzy procesami
//  kompilacja   mpicc -o projekt_1 projekt_1.c
//  uruchamianie mpirun --host localhost:liczba_procesorów/wątków ./projekt_1

int main(int argc, char ** argv)
{
    //Deklaracja i incjalizacja zmienych
    double *vt_1, *vt_1_part, *vt_2, *vt_2_part, *scores, score = 0; // Wskaźn
iki do tablic liczb typu double (nasze wektory i pod_wektory) i tablicy na wyn
iki oraz sumy dla każdego procesu z osobna
    int vt_normal_size; // Zmienna przechowująca ilość elementów dla każdego w
ektora, jest ona inicjalizowana przez użytkownika
    int vt_extend_size; // Zmienna przechowująca ilość elementów (równych 0),
o które został rozszerzony wektor
    int vt_real_size; // Zmienna przechowująca realną ilość elementów dla każd
ego wektora
    int vt_part_size; // Zmienna przechowująca ilość elementów wektorów jaki p
rzypada na każdy z procesów
    int rank; // Numer mojego procesu
    int size; // Liczba procesów
    struct timespec start, end; // Zmienna przechowująca czas rozpoczęcia i ko
ńca obliczeń

    MPI_Comm ring_hyper_d3; // Zmienna przechowująca moją grupę procesów
    /* Definicja struktury grafu!!! */
    int nnodes = 8; // Liczba węzłów
    int index[8] = {0, 1, 2, 3, 4, 5, 6, 7}; // Definicja indeksów
    int edges[16] = {0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 0}; // Defin
icja krawędzi
    //int edges[16] = {1, 0, 2, 1, 3, 2, 4, 3, 5, 4, 6, 5, 7, 6, 0, 7}; // Def
inicja krawędzi
    int reorder = 1; // Zezwolenie na zmianę kolejności procesów w celu popraw
y efektywności

    MPI_Init(&argc, &argv); // Rozpoczęcie obliczeń MPI
    MPI_Graph_create(MPI_COMM_WORLD, nnodes, index, edges,reorder, &ring_hyper
_d3); // Utworzenie struktury grafu, według powyższej definicji sturktury
    MPI_Comm_rank(ring_hyper_d3, &rank); // Pobranie aktualnego numeru procesu
    MPI_Comm_size(ring_hyper_d3, &size); // Pobranie liczby procesów
```

```
    if(rank == 0) // Instrukcje wykonywane dla głównego procesu programu (proc
es root)
    {
        fprintf(stderr, "Podaj długośc wektorów (liczba całkowita większa od 0
): "); // Wyświetlenie komunikatu z prośbą o podanie długości wektora
        scanf("%d", &vt_normal_size); // Pobranie długości wektów od użytkowni
ka

        if(vt_normal_size < 1) // Wyjście z programu dla wektora mniejszego ni
ż 1
        {
            printf("Podany rozmiar wektorów jest zbyt mały, ilość elementów je
st mniejsza niż ilość procesów: %d.\n", size);
            MPI_Finalize(); // Koniec obliczeń MPI
            exit(1); // Wyjście z programu z kodem błędu
        }

        vt_extend_size = size - (vt_normal_size % size); // Obliczenie o ile e
lementów należy rozszerzyć wektor, aby można było podzielić je równo dla każde
go procesu
        vt_real_size = vt_normal_size + vt_extend_size; // Obliczenie realnej
długości wektorów
        vt_part_size = vt_real_size / size; // Obliczenie ilości elementów prz
ypadających na każdy proces
        vt_1 = malloc(vt_real_size * sizeof(double)); // Zaalokowanie pamięci
na pierwszy wektor
        vt_2 = malloc(vt_real_size * sizeof(double)); // Zaalokowanie pamięci
na drugi wektor
        vt_1_part = malloc(vt_part_size * sizeof(double)); // Zaalokowanie pam
ięci na pierwszy wektor częściowy
        vt_2_part = malloc(vt_part_size * sizeof(double)); // Zaalokowanie pam
ięci na drugi wektor częściowy
        scores = malloc(size * sizeof(double)); // Zaalokowanie pamięci na tab
lice wyników

        for(int i = 0; i < vt_real_size; i++) // Inicjalizacja wektorów liczba
mi losowymi
        {
            if(i < vt_normal_size) // Wypełnienie elementów wektorów liczbami
typu rzeczywistego z przedziału (-25, 25)
            {
                srand(time(NULL) + rand());
                vt_1[i] = (double) rand() / RAND_MAX * 50 - 25;
                vt_2[i] = (double) rand() / RAND_MAX * 50 - 25;
            }
            else // Dla dodatkowo wygenerowanych elementów wartość wynosi 0, a
by wynik obliczeń nie został zakłamany
            {
                vt_1[i] = 0;
```

```
                vt_2[i] = 0;
            }
        }

        clock_gettime(CLOCK_MONOTONIC_RAW, &start); // Pobranie czasu rozpoczę
cia obliczeń
        MPI_Bcast(&vt_real_size, 1, MPI_INT, 0, ring_hyper_d3); // Wysłanie ro
zmiaru wektora
        MPI_Bcast(&vt_part_size, 1, MPI_INT, 0, ring_hyper_d3); // Wysłanie ro
zmiaru części wektora
        MPI_Scatter(vt_1, vt_part_size, MPI_DOUBLE, vt_1_part, vt_part_size, M
PI_DOUBLE, 0, ring_hyper_d3); // Wysłanie podzielonego wektora 1 na kawałki do
 procesów w grupie
        MPI_Scatter(vt_2, vt_part_size, MPI_DOUBLE, vt_2_part, vt_part_size, M
PI_DOUBLE, 0, ring_hyper_d3); // Wysłanie podzielonego wektora 1 na kawałki do
 procesów w grupie
        MPI_Barrier(ring_hyper_d3); // Bariera synchronizacyjna

        for (int i = 0; i < vt_part_size; i++) // Obliczenie iloczynu częściow
ych wektorów
        {
            score += (vt_1[i] * vt_2[i]);
        }
        fprintf(stderr, "Iloczyn skalarny procesora %d wynosi: %f\n", rank, sc
ore); // Wyświetlenie informacji o obliczonym częściowym iloczynie skalarnym
        MPI_Barrier(ring_hyper_d3); // Bariera synchronizacyjna
        MPI_Gather(&score, 1, MPI_DOUBLE, scores, 1, MPI_DOUBLE, 0, ring_hyper
_d3); // Zebranie wyników z wszystkich procesów grupie

        score = 0; // Wyzerowanie iloczynu skalarnego
        for (int i = 0; i < size; i++) // Połączenie wyników częściowych
        {
            score += scores[i];
        }

        fprintf(stderr, "Iloczyn skalarny wektorów %d-
elementowych dla %d procesorów wynosi: %f\n", vt_normal_size, size, score); //
 Wyświetlenie informacji o obliczonym iloczynie skalarnym
        clock_gettime(CLOCK_MONOTONIC_RAW, &end); // Pobranie informacji o cza
sie zakończenia obliczeń
        u_int64_t delta_us = (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_n
sec - start.tv_nsec) / 1000; // Obliczenie różnicy między czasem końca oblicze
ń, a rozpoczęciem w mikrosekundach
        fprintf(stderr, "Obliczenia trwały %6.2f milisekund\n", (double) delta
_us / 1000); // Wyświetlenie informacji na temat czasu obliczeń

        free(vt_1); // Zwolnienie obszaru pamięci wektora 1
        free(vt_2); // Zwolnienie obszaru pamięci wektora 2
        free(vt_1_part); // Zwolnienie obszaru pamięci wektora częściowego 1
```

```
        free(vt_2_part); // Zwolnienie obszaru pamięci wektora częściowego 2
        free(scores); // Zwolnienie obszaru pamięci tablicy wyników
    }
    else // Instrukcje wykonywane przez procesy podległe
    {
        MPI_Barrier(ring_hyper_d3); // Bariera synchronizacyjna
        MPI_Bcast(&vt_real_size, 1, MPI_INT, 0, ring_hyper_d3); // Odebranie r
ozmiaru wektora
        MPI_Bcast(&vt_part_size, 1, MPI_INT, 0, ring_hyper_d3); // Odebranie r
ozmiaru części wektora
        vt_1 = malloc(vt_real_size * sizeof(double)); // Zaalokowanie pamięci
na pierwszy wektor
        vt_2 = malloc(vt_real_size * sizeof(double)); // Zaalokowanie pamięci
na drugi wektor
        vt_1_part = malloc(vt_part_size * sizeof(double)); // Zaalokowanie pam
ięci na pierwszy wektor częsciowy
        vt_2_part = malloc(vt_part_size * sizeof(double)); // Zaalokowanie pam
ięci na drugi wektor częściowy

        MPI_Scatter(vt_1, vt_part_size, MPI_DOUBLE, vt_1_part, vt_part_size, M
PI_DOUBLE, 0, ring_hyper_d3); // Odebranie podzielonego wektora 1 na kawałki
        MPI_Scatter(vt_2, vt_part_size, MPI_DOUBLE, vt_2_part, vt_part_size, M
PI_DOUBLE, 0, ring_hyper_d3); // Odebranie podzielonego wektora 1 na kawałki
        for (int i = 0; i < vt_part_size; i++) // Obliczenie iloczynu częsciow
ych wektorów
        {
            score += (vt_1_part[i] * vt_2_part[i]);
        }
        fprintf(stderr, "Iloczyn skalarny procesora %d wynosi: %f\n", rank, sc
ore); // Wyświetlenie informacji o obliczonym częściowym iloczynie skalarnym
        MPI_Barrier(ring_hyper_d3); // Bariera synchronizacyjna
        MPI_Gather(&score, 1, MPI_DOUBLE, NULL, 1, MPI_DOUBLE, 0, ring_hyper_d
3); // Wysłanie wyników do procesu zbierającego wyniki

        free(vt_1); // Zwolnienie obszaru pamięci wektora 1
        free(vt_2); // Zwolnienie obszaru pamięci wektora 2
        free(vt_1_part); // Zwolnienie obszaru pamięci wektora częściowego 1
        free(vt_2_part); // Zwolnienie obszaru pamięci wektora częściowego 2
    }

    MPI_Finalize(); // Koniec obliczeń MPI
    return 0; // Wyjście z programu
}
```

3. Działanie aplikacji i wykonanie eksperymentów

```
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:2 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 10
Iloczyn skalarny procesora 0 wynosi: -186.440553
Iloczyn skalarny wektorów 10-elementowych dla 2 procesorów wynosi: -49.836273
Iloczyn skalarny procesora 1 wynosi: 136.604280
Obliczenia trwały   0.10 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:2 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 100
Iloczyn skalarny procesora 0 wynosi: -262.454303
Iloczyn skalarny wektorów 100-elementowych dla 2 procesorów wynosi: -172.900917
Iloczyn skalarny procesora 1 wynosi: 89.553387
Obliczenia trwały   0.12 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:2 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 500
Iloczyn skalarny procesora 0 wynosi: -4403.571732
Iloczyn skalarny wektorów 500-elementowych dla 2 procesorów wynosi: -1024.317182
Iloczyn skalarny procesora 1 wynosi: 3379.254551
Obliczenia trwały   0.12 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:2 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: 4669.461028
Iloczyn skalarny wektorów 1000-elementowych dla 2 procesorów wynosi: 3894.631786
Iloczyn skalarny procesora 1 wynosi: -774.829242
Obliczenia trwały   0.16 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$
```

```
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:4 ./projekt_1
Podaj długosc wektorów (liczba całkowita większa od 0): 10
Iloczyn skalarny procesora 0 wynosi: 48.412275
Iloczyn skalarny procesora 1 wynosi: 799.462943
Iloczyn skalarny procesora 2 wynosi: 567.978182
Iloczyn skalarny procesora 3 wynosi: -13.002447
Iloczyn skalarny wektorów 10-elementowych dla 4 procesorów wynosi: 1402.850954
Obliczenia trwały   0.16 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:4 ./projekt_1
Podaj długosc wektorów (liczba całkowita większa od 0): 100
Iloczyn skalarny procesora 0 wynosi: 1442.418474
Iloczyn skalarny procesora 1 wynosi: -503.093973
Iloczyn skalarny procesora 2 wynosi: -393.765378
Iloczyn skalarny procesora 3 wynosi: 1179.647504
Iloczyn skalarny wektorów 100-elementowych dla 4 procesorów wynosi: 1725.206627
Obliczenia trwały   0.34 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:4 ./projekt_1
Podaj długosc wektorów (liczba całkowita większa od 0): 500
Iloczyn skalarny procesora 0 wynosi: 2524.628912
Iloczyn skalarny procesora 1 wynosi: 2536.854259
Iloczyn skalarny procesora 2 wynosi: 643.260073
Iloczyn skalarny procesora 3 wynosi: -1003.690324
Iloczyn skalarny wektorów 500-elementowych dla 4 procesorów wynosi: 4701.052920
Obliczenia trwały   0.22 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:4 ./projekt_1
Podaj długosc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: 1955.485313
Iloczyn skalarny procesora 1 wynosi: 2937.441640
Iloczyn skalarny procesora 2 wynosi: -4246.731934
Iloczyn skalarny procesora 3 wynosi: 7060.888077
Iloczyn skalarny wektorów 1000-elementowych dla 4 procesorów wynosi: 7707.083096
Obliczenia trwały   0.26 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$
```

```
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:6 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 10
Iloczyn skalarny procesora 0 wynosi: 295.549645
Iloczyn skalarny procesora 1 wynosi: -295.212071
Iloczyn skalarny procesora 2 wynosi: 529.566166
Iloczyn skalarny procesora 4 wynosi: 109.388409
Iloczyn skalarny wektorów 10-elementowych dla 6 procesorów wynosi: 759.312944
Iloczyn skalarny procesora 3 wynosi: 120.020794
Iloczyn skalarny procesora 5 wynosi: 0.000000
Obliczenia trwały   0.25 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:6 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 100
Iloczyn skalarny procesora 0 wynosi: -499.736052
Iloczyn skalarny procesora 1 wynosi: 606.519869
Iloczyn skalarny procesora 2 wynosi: -234.230892
Iloczyn skalarny procesora 3 wynosi: 637.982852
Iloczyn skalarny procesora 4 wynosi: 990.547228
Iloczyn skalarny procesora 5 wynosi: -648.891238
Iloczyn skalarny wektorów 100-elementowych dla 6 procesorów wynosi: 852.191768
Obliczenia trwały   0.52 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:6 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 500
Iloczyn skalarny procesora 0 wynosi: -6.758893
Iloczyn skalarny procesora 1 wynosi: 3833.761587
Iloczyn skalarny procesora 4 wynosi: -1445.936226
Iloczyn skalarny procesora 5 wynosi: 759.061295
Iloczyn skalarny wektorów 500-elementowych dla 6 procesorów wynosi: 1139.858482
Iloczyn skalarny procesora 2 wynosi: 30.589793
Iloczyn skalarny procesora 3 wynosi: -2030.859074
Obliczenia trwały   0.54 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:6 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: -3408.629862
Iloczyn skalarny procesora 2 wynosi: 3289.728458
Iloczyn skalarny procesora 3 wynosi: 3505.656977
Iloczyn skalarny procesora 4 wynosi: -1859.254029
Iloczyn skalarny procesora 5 wynosi: -3771.584363
Iloczyn skalarny procesora 1 wynosi: 1036.427254
Iloczyn skalarny wektorów 1000-elementowych dla 6 procesorów wynosi: -1207.655565
Obliczenia trwały   1.33 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$
```

```
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 10
Iloczyn skalarny procesora 0 wynosi: 86.813372
Iloczyn skalarny procesora 1 wynosi: -355.466613
Iloczyn skalarny procesora 2 wynosi: -248.276774
Iloczyn skalarny procesora 3 wynosi: 42.058255
Iloczyn skalarny procesora 4 wynosi: -531.239240
Iloczyn skalarny procesora 5 wynosi: 0.000000
Iloczyn skalarny procesora 6 wynosi: 0.000000
Iloczyn skalarny procesora 7 wynosi: 0.000000
Iloczyn skalarny wektorów 10-elementowych dla 8 procesorów wynosi: -1006.111000
Obliczenia trwały   3.52 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 100
Iloczyn skalarny procesora 0 wynosi: 105.104900
Iloczyn skalarny procesora 1 wynosi: 1547.603142
Iloczyn skalarny procesora 2 wynosi: 150.686749
Iloczyn skalarny procesora 3 wynosi: 298.464402
Iloczyn skalarny procesora 4 wynosi: 135.193452
Iloczyn skalarny procesora 5 wynosi: -1238.709970
Iloczyn skalarny procesora 6 wynosi: 863.602201
Iloczyn skalarny procesora 7 wynosi: -297.042922
Iloczyn skalarny wektorów 100-elementowych dla 8 procesorów wynosi: 1564.901956
Obliczenia trwały   1.78 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 500
Iloczyn skalarny procesora 0 wynosi: 1236.493816
Iloczyn skalarny procesora 1 wynosi: -980.703768
Iloczyn skalarny procesora 2 wynosi: -1223.942411
Iloczyn skalarny procesora 3 wynosi: 521.063385
Iloczyn skalarny procesora 4 wynosi: -1516.487219
Iloczyn skalarny procesora 5 wynosi: -680.499233
Iloczyn skalarny procesora 6 wynosi: 2395.527941
Iloczyn skalarny procesora 7 wynosi: -273.571600
Iloczyn skalarny wektorów 500-elementowych dla 8 procesorów wynosi: -522.119088
Obliczenia trwały  15.80 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_1
Podaj długośc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: -608.054431
Iloczyn skalarny procesora 1 wynosi: 191.862229
Iloczyn skalarny procesora 2 wynosi: 3106.694763
Iloczyn skalarny procesora 3 wynosi: 652.299401
Iloczyn skalarny procesora 4 wynosi: 157.675396
Iloczyn skalarny procesora 5 wynosi: 338.390061
Iloczyn skalarny procesora 6 wynosi: -1723.506472
Iloczyn skalarny procesora 7 wynosi: -317.340606
Iloczyn skalarny wektorów 1000-elementowych dla 8 procesorów wynosi: 1798.020340
Obliczenia trwały   6.22 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$
```

```
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_2
Podaj długośc wektorów (liczba całkowita większa od 0): 10
Iloczyn skalarny procesora 0 wynosi: -60.500559
Iloczyn skalarny procesora 1 wynosi: 137.809717
Iloczyn skalarny procesora 2 wynosi: 201.214365
Iloczyn skalarny procesora 3 wynosi: -289.665622
Iloczyn skalarny procesora 4 wynosi: -63.885206
Iloczyn skalarny procesora 5 wynosi: 0.000000
Iloczyn skalarny procesora 6 wynosi: 0.000000
Iloczyn skalarny procesora 7 wynosi: 0.000000
Iloczyn skalarny wektorów 10-elementowych dla 8 procesorów wynosi: -75.027305
Obliczenia trwały   1.02 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_2
Podaj długośc wektorów (liczba całkowita większa od 0): 100
Iloczyn skalarny procesora 0 wynosi: 254.168100
Iloczyn skalarny procesora 4 wynosi: -706.701013
Iloczyn skalarny procesora 5 wynosi: 409.536120
Iloczyn skalarny procesora 6 wynosi: -169.011404
Iloczyn skalarny procesora 7 wynosi: -826.705882
Iloczyn skalarny procesora 1 wynosi: 796.862801
Iloczyn skalarny procesora 2 wynosi: 398.910374
Iloczyn skalarny procesora 3 wynosi: 147.982477
Iloczyn skalarny wektorów 100-elementowych dla 8 procesorów wynosi: 305.041573
Obliczenia trwały   4.31 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_2
Podaj długośc wektorów (liczba całkowita większa od 0): 500
Iloczyn skalarny procesora 0 wynosi: 1160.205698
Iloczyn skalarny procesora 1 wynosi: 1902.445738
Iloczyn skalarny procesora 2 wynosi: -1169.025886
Iloczyn skalarny procesora 3 wynosi: -935.037589
Iloczyn skalarny procesora 4 wynosi: -1007.235201
Iloczyn skalarny procesora 5 wynosi: -4912.929034
Iloczyn skalarny procesora 6 wynosi: 2783.819902
Iloczyn skalarny procesora 7 wynosi: 1558.286741
Iloczyn skalarny wektorów 500-elementowych dla 8 procesorów wynosi: -619.469631
Obliczenia trwały   1.28 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_2
Podaj długośc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: 1000.296770
Iloczyn skalarny procesora 1 wynosi: 1181.304372
Iloczyn skalarny procesora 2 wynosi: 3537.837456
Iloczyn skalarny procesora 3 wynosi: 2059.954475
Iloczyn skalarny procesora 4 wynosi: 1446.922747
Iloczyn skalarny procesora 5 wynosi: -331.423440
Iloczyn skalarny procesora 6 wynosi: -882.294538
Iloczyn skalarny procesora 7 wynosi: 2836.399030
Iloczyn skalarny wektorów 1000-elementowych dla 8 procesorów wynosi: 10848.996871
Obliczenia trwały   4.89 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$
```

```
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_2
Podaj długośc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: -642.895185
Iloczyn skalarny procesora 1 wynosi: -1463.049881
Iloczyn skalarny procesora 2 wynosi: 1414.369121
Iloczyn skalarny procesora 3 wynosi: -2138.903525
Iloczyn skalarny procesora 6 wynosi: -3528.951039
Iloczyn skalarny procesora 7 wynosi: -45.668524
Iloczyn skalarny procesora 4 wynosi: -1347.380721
Iloczyn skalarny procesora 5 wynosi: -3423.808847
Iloczyn skalarny wektorów 1000-elementowych dla 8 procesorów wynosi: -11176.288602
Obliczenia trwały   2.40 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_2
Podaj długośc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: -1199.335455
Iloczyn skalarny procesora 1 wynosi: -1539.756092
Iloczyn skalarny procesora 2 wynosi: 111.145440
Iloczyn skalarny procesora 3 wynosi: -461.904377
Iloczyn skalarny procesora 4 wynosi: 68.503967
Iloczyn skalarny procesora 5 wynosi: -2776.335664
Iloczyn skalarny procesora 6 wynosi: 1850.552745
Iloczyn skalarny procesora 7 wynosi: -876.126877
Iloczyn skalarny wektorów 1000-elementowych dla 8 procesorów wynosi: -4823.256312
Obliczenia trwały   6.05 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_2
Podaj długośc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: -1690.551863
Iloczyn skalarny procesora 1 wynosi: -1070.979523
Iloczyn skalarny procesora 2 wynosi: 227.280711
Iloczyn skalarny procesora 3 wynosi: 246.588611
Iloczyn skalarny procesora 4 wynosi: 2540.955717
Iloczyn skalarny procesora 5 wynosi: -4652.044374
Iloczyn skalarny procesora 7 wynosi: 1366.840392
Iloczyn skalarny procesora 6 wynosi: 1715.623043
Iloczyn skalarny wektorów 1000-elementowych dla 8 procesorów wynosi: -1316.287287
Obliczenia trwały   4.69 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$ mpirun --host localhost:8 ./projekt_2
Podaj długośc wektorów (liczba całkowita większa od 0): 1000
Iloczyn skalarny procesora 0 wynosi: 1188.806799
Iloczyn skalarny procesora 1 wynosi: -1079.534425
Iloczyn skalarny procesora 4 wynosi: 974.214041
Iloczyn skalarny procesora 5 wynosi: -3210.249197
Iloczyn skalarny procesora 6 wynosi: -1082.475126
Iloczyn skalarny procesora 2 wynosi: -2241.241129
Iloczyn skalarny procesora 3 wynosi: -294.079324
Iloczyn skalarny procesora 7 wynosi: -2444.185687
Iloczyn skalarny wektorów 1000-elementowych dla 8 procesorów wynosi: -8188.744048
Obliczenia trwały   1.56 milisekund
michal@DESKTOP-JO2L2LJ:~/Projekt_MPI$
```

Rysunek 1 Dowód na zakłamanie wyników

4. Podsumowanie wyników

Typ relacji	Liczba procesorów	Rozmiar wektorów	Czas w milisekundach
Brak	2	10	0.10
Brak	4	10	0.16
Brak	6	10	0.25
Brak	8	10	3.52
Pierścień (hypercube d=3)	8	10	1.02
Brak	2	100	0.12
Brak	4	100	0.34
Brak	6	100	0.52
Brak	8	100	1.78
Pierścień (hypercube d=3)	8	100	4.31
Brak	2	500	0.12
Brak	4	500	0.22
Brak	6	500	0.54
Brak	8	500	15.8
Pierścień (hypercube d=3)	8	500	1.28
Brak	2	1000	0.16
Brak	4	1000	0.26
Brak	6	1000	1.33
Brak	8	1000	6.22
Pierścień (hypercube d=3)	8	1000	4.89

5. Wnioski

Pierwsze rzecz na którą należy zwrócić uwagę jest częściowe zakłamanie wyników, ponieważ program uruchamiany jest na komputerze z zainstalowaną sporą ilością usług np. kilka baz danych, komunikatory internetowe itd. Z tego względu obciążenie procesora, może być różne. Wpływ na wyniki ma również to, iż wektory generowane są losowo, dlatego dane wejściowe nigdy nie są takie same. Jeśli porównamy wyniki wykonania programu z ustaloną relacją procesów i programu z brakiem takiej relacji dla takiej samej liczby procesów, to na pierwszy rzut oka widać, że średni czas obliczeń jest krótszy dla programu z ustaloną relacją procesów. Natomiast jeśli porównamy wyniki obliczeń dla różnej ilości procesów, dla nieustalonej relacji procesorów, to nie wygląda to już tak kolorowo. Większa liczba procesów nie powoduje przyśpieszenia obliczeń, jeśli same obliczenia nie są zbyt skomplikowane i dane wejściowe są małe. Wynika to prawdopodobnie z faktu, iż samo wygenerowanie struktury procesów i wysłanie tych informacji do innych procesów, także wymaga czasu. W połączeniu z prostymi algorytmami, może to spowodować sumarycznie dłuższy czas obliczeń, niż dla programu bez podziału na procesy. Dlatego zawsze należy się zastanowić, czy wdrożenie technologii systemów rozproszonych ma sens dla każdego danego przykładu z osobna.