Sprawozdanie 1

Alicja Niewiadomska | Metody Programowania Równoległego AGH 2023

Komunikacja PP

Celem eksperymentów było zmierzenie wartości opóźnienia i charakterystyki przepustowości w klastrze dla różnych rodzajów komunikacji oraz różnych konfiguracji środowiska.

Konfiguracja eksperymentu

W ramach sprawozdania przetestowałam 3 rodzaje komunikacji:

- ➔ komunikację standardową *MPI_Send*
- ➔ komunikację synchroniczną *MPI_Ssend*
- ➔ komunikację buforowaną *MPI_Bsend*

Każdy rodzaj komunikacji został przetestowany:

- ➔ między 2 procesami w obrębie 1 węzła
- ➔ między 2 procesami na 2 węzłach

W ramach pojedynczego eksperymentu 2 procesy wymieniały ze sobą wiadomości o rozmiarach payloadu równych kolejnym potęgom 2 z zakresu *<1B; 2.5E6)*. Dla każdego rozmiaru payloadu zmierzony został czas dla 1000 wymian wiadomości.

Program testujący

Plik *ping_pong.c.*

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

#define STANDARD 0
#define SYNC 1
#define BUFF 2

int BUFF_SIZE = 1E7;
int MSG_COUNT = 1E3;
```

```
int MAX_MSG_SIZE = 2.5E6;

double sender(int size, int mode) {
    MPI_Barrier(MPI_COMM_WORLD);
    char *buff = malloc(size);
    int i;
    double start = MPI_Wtime();
    for (i = 0; i < MSG_COUNT; i++) {

      if (mode == STANDARD) {
            MPI_Send(buff, size, MPI_BYTE, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(buff, size, MPI_BYTE, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      } else if (mode == SYNC) {
            MPI_Ssend(buff, size, MPI_BYTE, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(buff, size, MPI_BYTE, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        } else if (mode == BUFF) {
            MPI_Bsend(buff, size, MPI_BYTE, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(buff, size, MPI_BYTE, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
    }
    double end = MPI_Wtime();
    free(buff);
    return end - start;
}

void receiver(int size, int mode) {
    char *buff = malloc(size);
    int i;
    for (i = 0; i < MSG_COUNT; i++) {
        if (mode == STANDARD) {
            MPI_Recv(buff, size, MPI_BYTE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            MPI_Send(buff, size, MPI_BYTE, 0, 0, MPI_COMM_WORLD);
      } else if (mode == SYNC) {
            MPI_Recv(buff, size, MPI_BYTE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            MPI_Ssend(buff, size, MPI_BYTE, 0, 0, MPI_COMM_WORLD);
        } else if (mode == BUFF) {
           MPI_Recv(buff, size, MPI_BYTE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
           MPI_Bsend(buff, size, MPI_BYTE, 0, 0, MPI_COMM_WORLD);
        }
    }
    free(buff);
}

double ping_pong(int rank, int size, int mode) {
    if (rank == 0) {
        double time = sender(size, mode);
```

```
        return time;
    } else if (rank == 1) {
        receiver(size, mode);
    }
    return 0;
}


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

    MPI_Init(NULL, NULL);

    int world_rank, mode;
    mode = atoi(argv[1]); // 0 - standard, 1 - sync, 2 - buff
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    MPI_Buffer_attach(malloc(BUFF_SIZE), BUFF_SIZE);

    for (int size = 1; size < MAX_MSG_SIZE; size *= 2) { // in bytes
        double time = ping_pong(world_rank, size, mode); // in seconds
      if (world_rank == 0) {
            if (size == 1) {
                printf("TIME: %.9f\n", time); // in seconds
            }
          double x = MSG_COUNT * 8 * (size / 10E6); // Mbit
            printf("%d %d %.9f\n", mode, size, x / (time / 2)); // Mbit/s
        }
    }
    MPI_Finalize();
}
```

Skrypt testujący

Plik *run_vnode_comm.sh.*

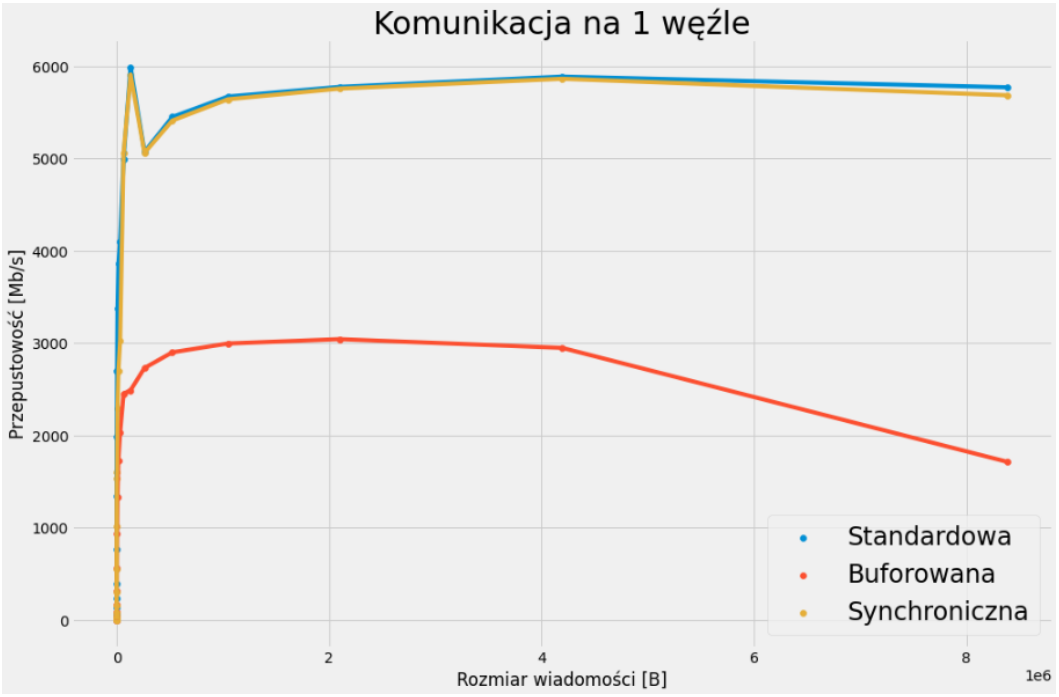
```
#!/bin/bash

mpicc -std=c99 -o ping_pong ping_pong.c

for mode in 0 1 2
do
   mpiexec -machinefile ./onenode -np 2 ./ping_pong $mode >> pp1_out.txt
   mpiexec -machinefile ./twonodes -np 2 ./ping_pong $mode >> pp2_out.txt
done
```

Wyniki

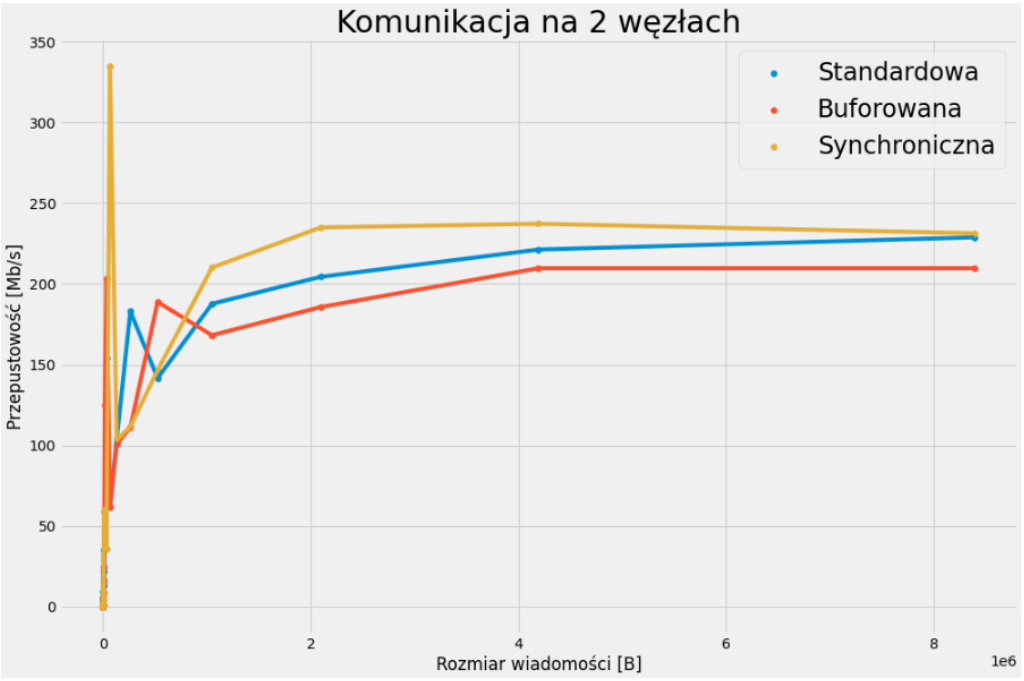
Na podstawie przeprowadzonych eksperymentów i uzyskanych z nich danych, przygotowałam wykresy przedstawiające przepustowość oraz opóźnienie dla każdego rodzaju komunikacji. Dane znajdują się w pliku *vcluster_comm.txt.*

Przepustowość

Poniższy wykres przedstawia zależność przepustowości od liczby procesorów dla komunikacji standardowej, synchronicznej i buforowanej na 1 węźle.

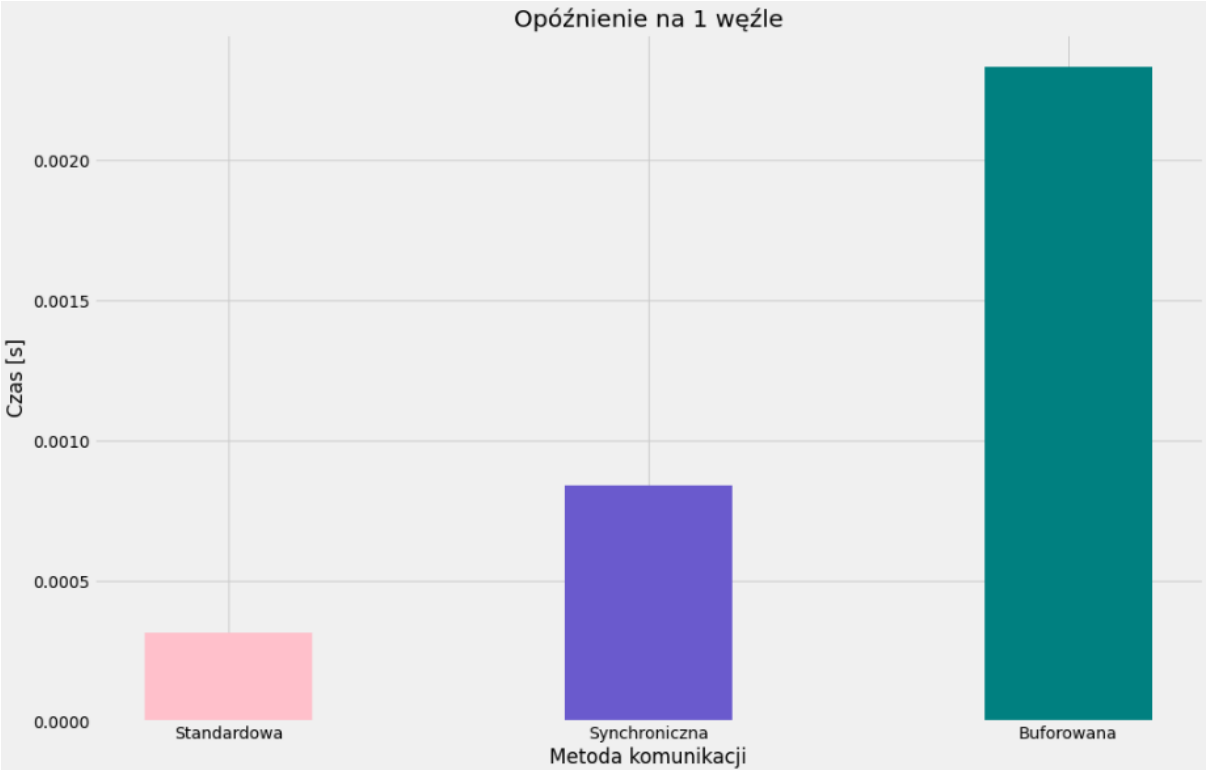


Kolejny wykres przedstawia tę samą zależność, dla komunikacji procesów znajdujących się na 2 różnych węzłach.



Opóźnienie

Poniższy wykres prezentuje zmierzone opóźnienie na pojedynczym węźle dla testowanych sposobów komunikacji.



Kolejny wykres przedstawia opóźnienie zmierzone dla komunikacji między 2 różnymi węzłami.

Wnioski

- ➔ Gdy oba procesy znajdowały się na 1 węźle, komunikacja standardowa i synchroniczna radziły sobie najlepiej (ok. 6000 Mb/s), podczas gdy przepustowość komunikacji buforowanej okazała się o połowę mniejsza do pozostałych.
- ➔ Gdy procesy znajdowały się na osobnych węzłach, przepustowość była zbliżona dla każdego rodzaju komunikacji (ok 250 Mb/s). We wszystkich trybach komunikacji dla dużych wiadomości (powyżej 2MB) przepustowość przestała rosnąć, ze względu na ograniczenie przepustowości sieci.
- ➔ Największe opóźnienie na 1 węźle zostało zmierzone dla komunikacji buforowanej.
- ➔ Największe opóźnienie na 2 węzłach zostało zmierzone dla komunikacji synchronicznej.
- ➔ Zarówno w przypadku komunikacji na tym samym węźle jak i między węzłami, najmniejsze opóźnienie dała komunikacji standardowa.

Badanie efektywności programu równoległego

Celem eksperymentów było zbadanie efektywności zrównoleglonego programu obliczającego wartość liczby π, bazującego na metodzie Monte Carlo.

Program testujący

Dla obu środowisk program testujący był jednakowy. Plik *pi.c*.

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


double get_coord() {
    double range = 1;
    double div = RAND_MAX / range;
    return (rand() / div);
}


int is_inside(double x, double y) {
    return pow(x, 2) + pow(y, 2) < 1;
}


unsigned long long calc_inside(unsigned long long n) {
    unsigned long long inside = 0;
    double x, y;

    for (unsigned long long i=0; i<n; i++) {
        x = get_coord();
        y = get_coord();

        if (is_inside(x, y)) {
            inside++;
        }
    }
    return inside;
}

double calc_pi(unsigned long long n, unsigned long long inside) {
    return 4.0 * inside / n;
}
```

```
void run_experiment(int type, unsigned long long points_per_proc, unsigned long long
total_points, int rank) {
    MPI_Barrier(MPI_COMM_WORLD);
    double start = MPI_Wtime();

    unsigned long long points_inside = calc_inside(points_per_proc);
    unsigned long long total_inside;

    MPI_Reduce(&points_inside , &total_inside, 1, MPI_UNSIGNED_LONG_LONG, MPI_SUM, 0,
MPI_COMM_WORLD);

    if (rank == 0) {
      double pi = calc_pi(total_points, total_inside);
      double end = MPI_Wtime();
        printf("%i %llu %.8f %.8f\n", type, total_points, pi, end - start);
    }
}

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

    MPI_Init(NULL, NULL);
    int rank, size, split_points;
    unsigned long long points_num, points_per_proc, total_points;

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    points_num = strtoull(argv[1], NULL, 0);
    split_points = atoi(argv[2]);

    if (split_points) {
        points_per_proc = points_num / size;
      total_points = points_num;
    } else {
        points_per_proc = points_num;
      total_points = points_num * size;
    }

    srand(time(NULL));
    run_experiment(split_points, points_per_proc, total_points, rank);

    MPI_Finalize();
    return 0;
}
```

Środowisko vCluster

Konfiguracja eksperymentu

Każdy pomiar został przeprowadzony:

- ➔ dla liczby procesorów od 1 do 12
- ➔ dla wariantu skalowania silnego i słabego
- ➔ dla wejściowego rozmiaru zadania `1000000000`

Skrypt testujący

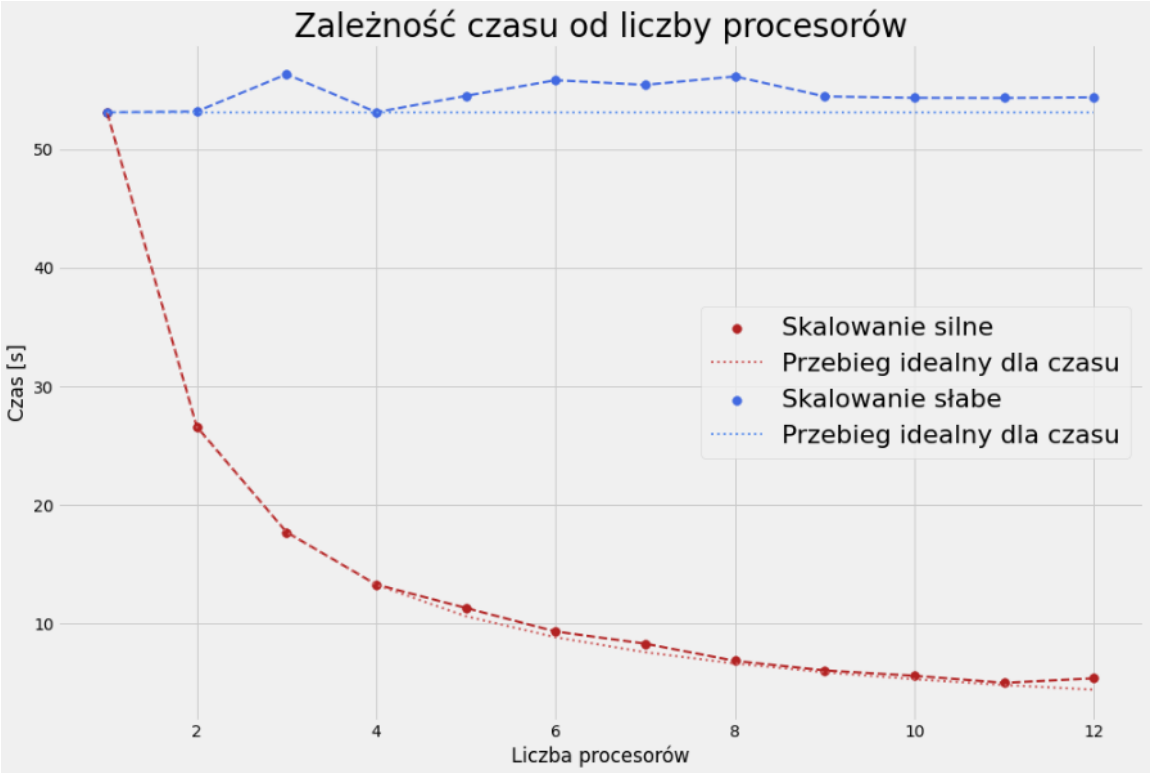
Plik *run_vnode.sh.*

```
#!/bin/bash

mpicc -std=c99 -o pi pi.c

for type in 0 1
do
     for proc in $(seq 1 12);
     do
        mpiexec -machinefile ./allnodes -np $proc ./pi 1000000000 $type >> vnode_out.txt
     done
done
```

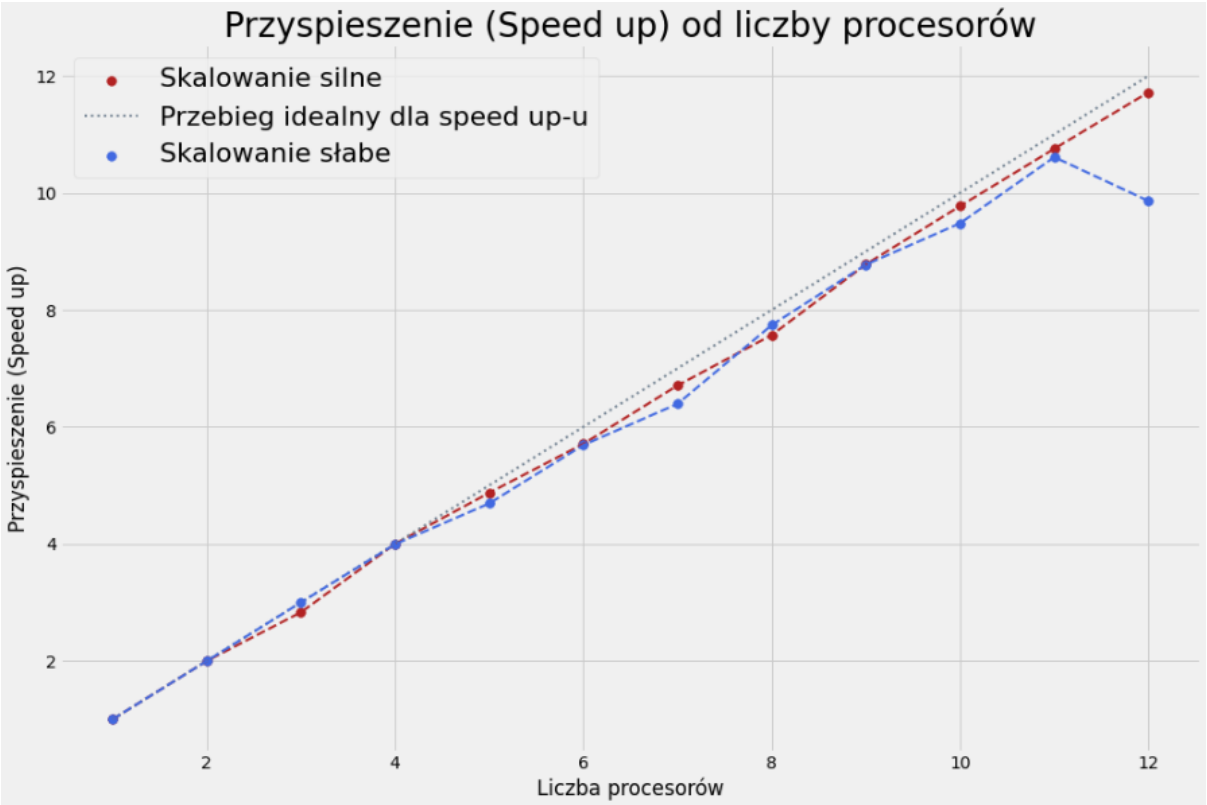
Wyniki

Na podstawie zebranych danych, przygotowałam wykresy zależności liczby procesorów od: czasu, przyspieszenia, efektywności oraz części sekwencyjnej. Dla każdej metryki dodałam także linię idealnego teoretycznego przebiegu (linia kropkowana). Wyniki znajdują się w pliku *vcluster_pi.txt.*

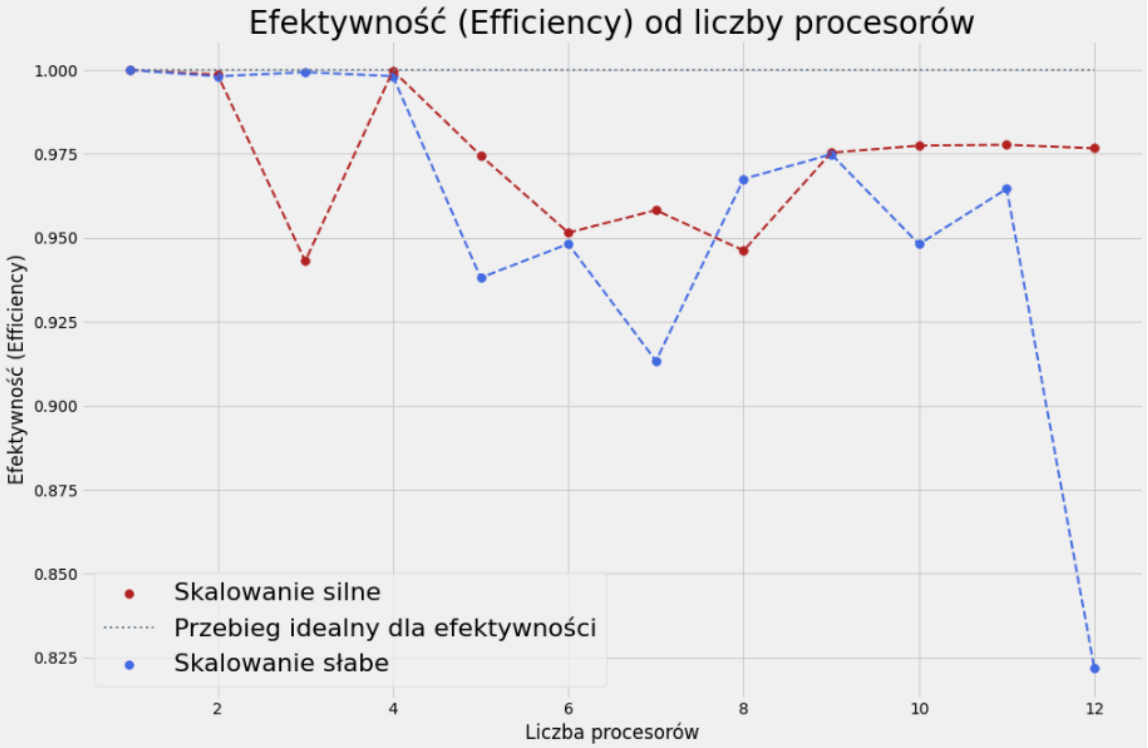
Poniższy wykres przedstawia zależność czasu obliczeń od liczby procesorów, dla skalowania słabego oraz silnego, w środowisku vCluster.



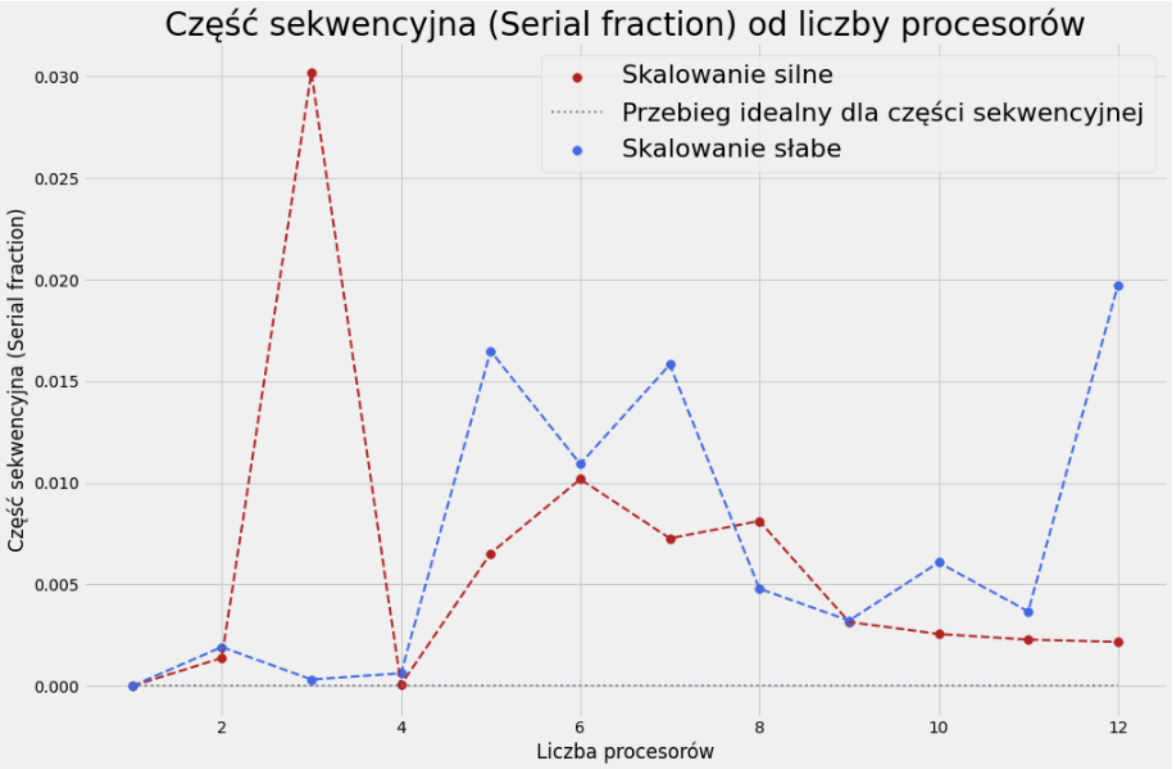
Kolejny wykres przedstawia zależność przyspieszenia (speed up) od liczby procesorów, dla obu wariantów skalowania w środowisku vCluster.



Wykres przedstawia zależność efektywności programu od liczby procesorów, dla obu wariantów skalowania w środowisku vCluster.



Ostatni wykres przedstawia zależność rozmiaru części sekwencyjnej programu od liczby procesorów, dla obu wariantów skalowania w środowisku vCluster.



Wnioski

- ➔ W obu wariantach: skalowaniu silnym oraz skalowaniu słabym zależność czasu od liczby procesorów jest zbliżona do idealnego teoretycznego przebiegu.
- ➔ W przypadku przyspieszenia, zależność także była zbliżona do idealnego przebiegu dla obu wariantów skalowania.
- ➔ Efektywność w obu wariantach utrzymywała się na poziomie wyższym niż 0.9, z wyjątkiem pojedynczego pomiaru w wariancie skalowania słabego.
- ➔ Część sekwencyjna nie przekracza 0.031 w przypadku skalowania silnego, a dla skalowania słabego jest mniejsza niż 0.02.
- ➔ W przypadku skalowania silnego możemy zaobserwować punkt pomiarowy, w których wartość znacząco odbiega od reszty serii (dla 3 procesorów), a w przypadku skalowania słabego dla 12 procesów - może to wynikać z dodatkowego obciążenia node'a w danym momencie, np. przez inny program.

Środowisko Ares

Konfiguracja eksperymentu

Każdy pomiar został przeprowadzony:

- ➔ dla liczby procesorów od 1 do 12
- ➔ dla wariantu skalowania silnego i słabego
- ➔ dla 3 rozmiarów problemu: `100`, `2000000` oraz `30000000000`
- ➔ dla ustalonych parametrów pomiar został przeprowadzony 10 razy

Skrypt testujący

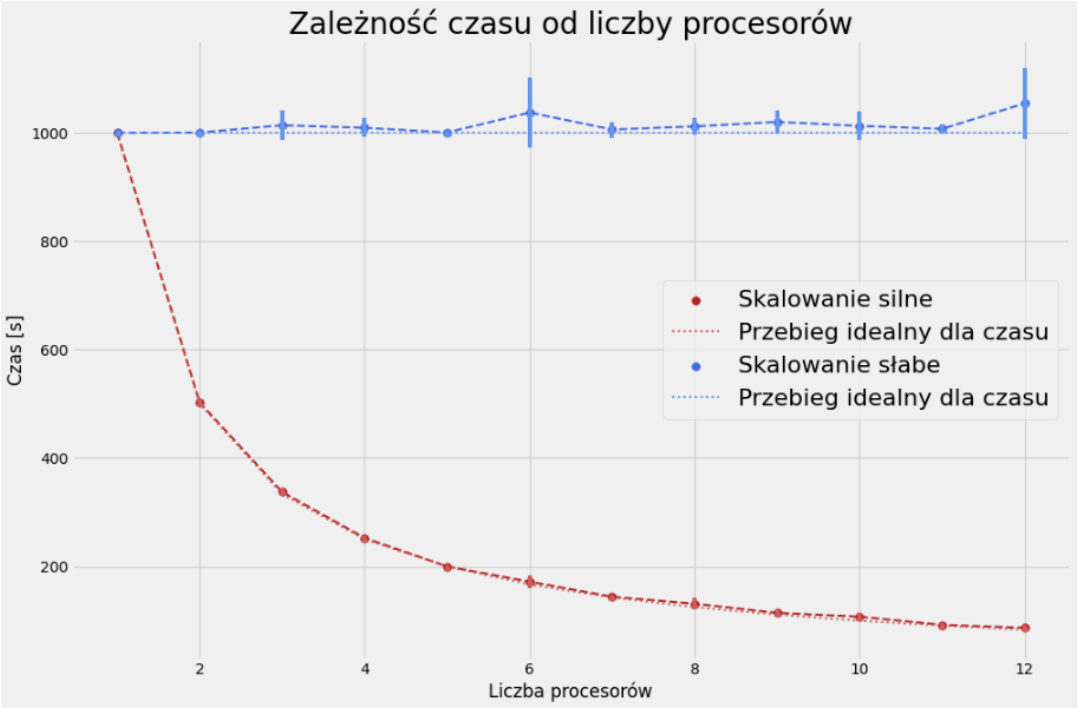
Plik *run_ares.sh*.

```
#!/bin/bash -l
#SBATCH --nodes 1
#SBATCH --ntasks 12
#SBATCH --time=05:00:00
#SBATCH --partition=plgrid
#SBATCH --account=plgmpr23-cpu

module add .plgrid plgrid/tools/openmpi
mpicc -std=c99 -o pi pi.c

for type in 0 1
do
    for proc in $(seq 1 12);
    do
        mpiexec -np $proc ./pi $1 $type
    done
done
```

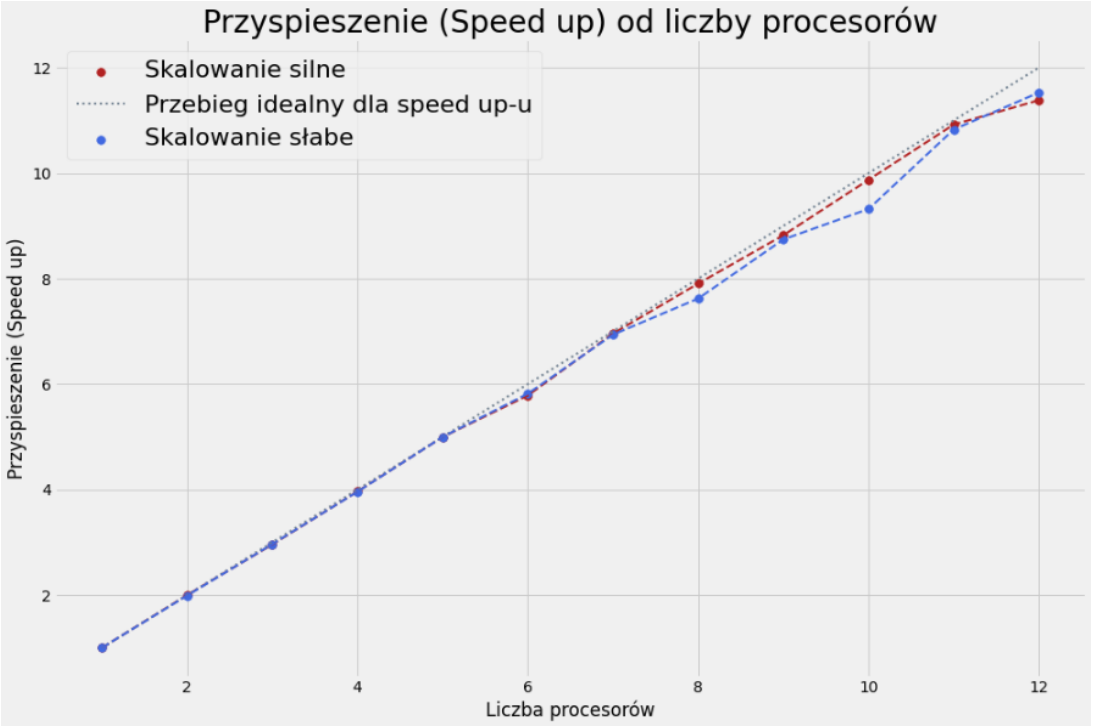
Wyniki

➔ **dla dużego problemu**

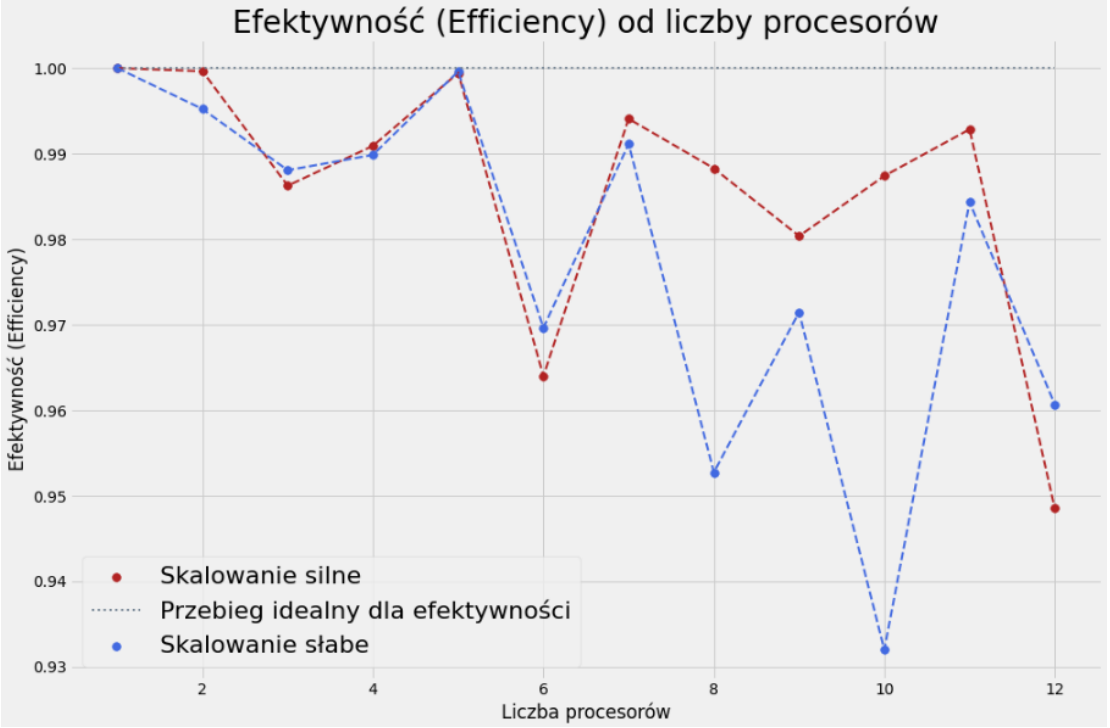
Poniższy wykres przedstawia zależność czasu obliczeń od liczby procesorów, dla skalowania słabego oraz silnego, w środowisku Ares.



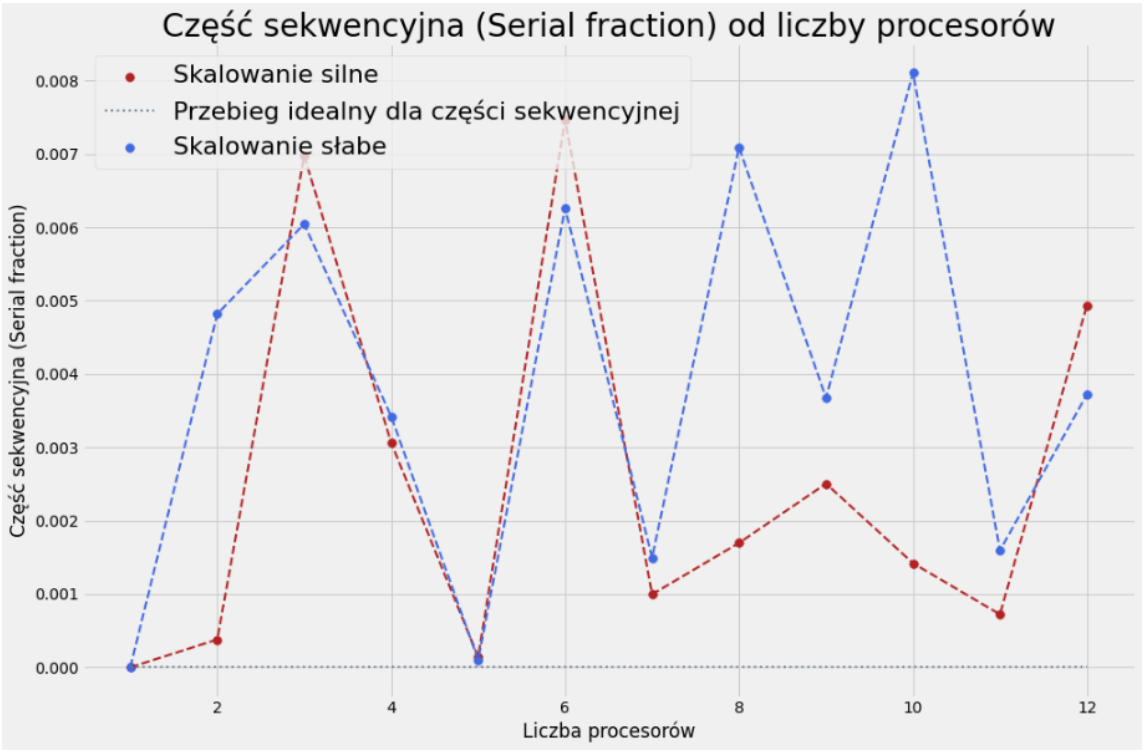
Kolejny wykres przedstawia zależność przyspieszenia (speed up) od liczby procesorów, dla obu wariantów skalowania w środowisku Ares.



Wykres przedstawia zależność efektywności programu od liczby procesorów, dla obu wariantów skalowania w środowisku Ares.

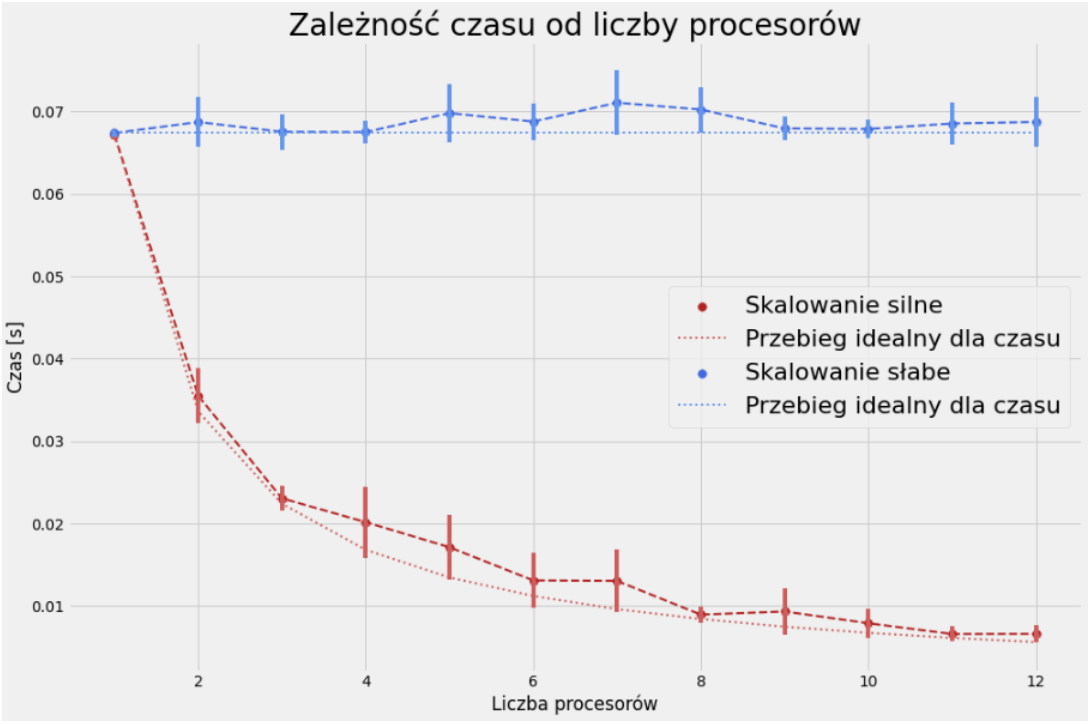


Ostatni wykres przedstawia zależność rozmiaru części sekwencyjnej programu od liczby procesorów, dla obu wariantów skalowania w środowisku Ares.

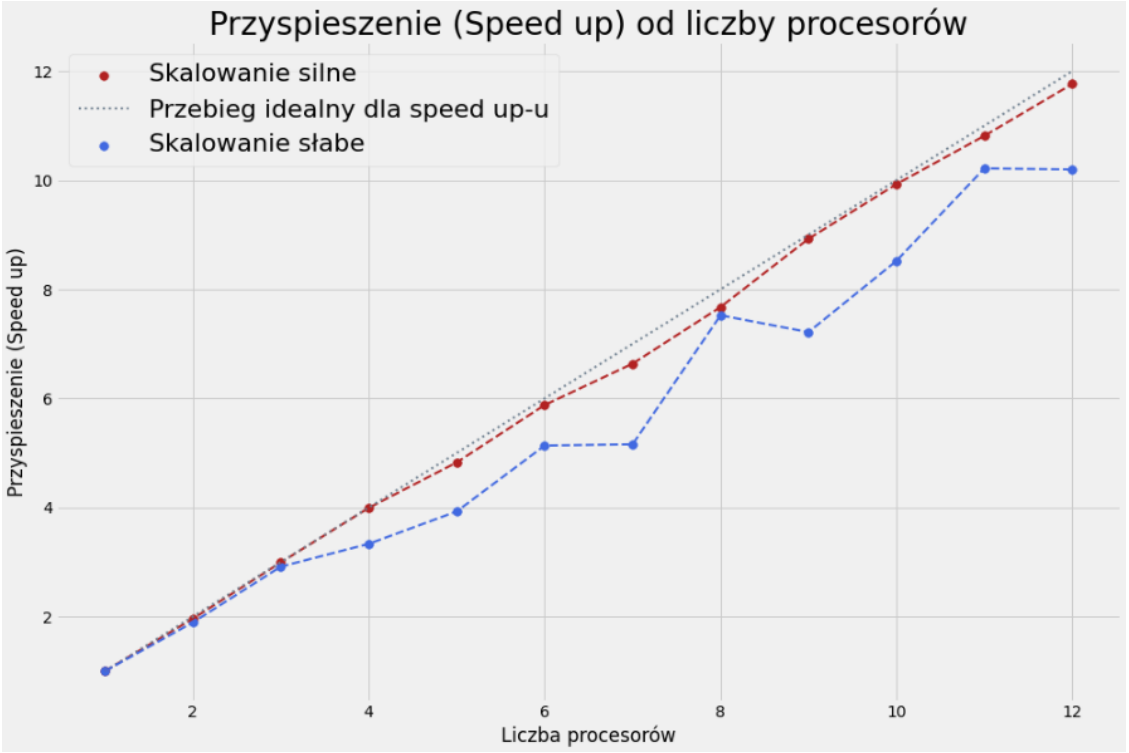


➔ **dla średniego problemu**

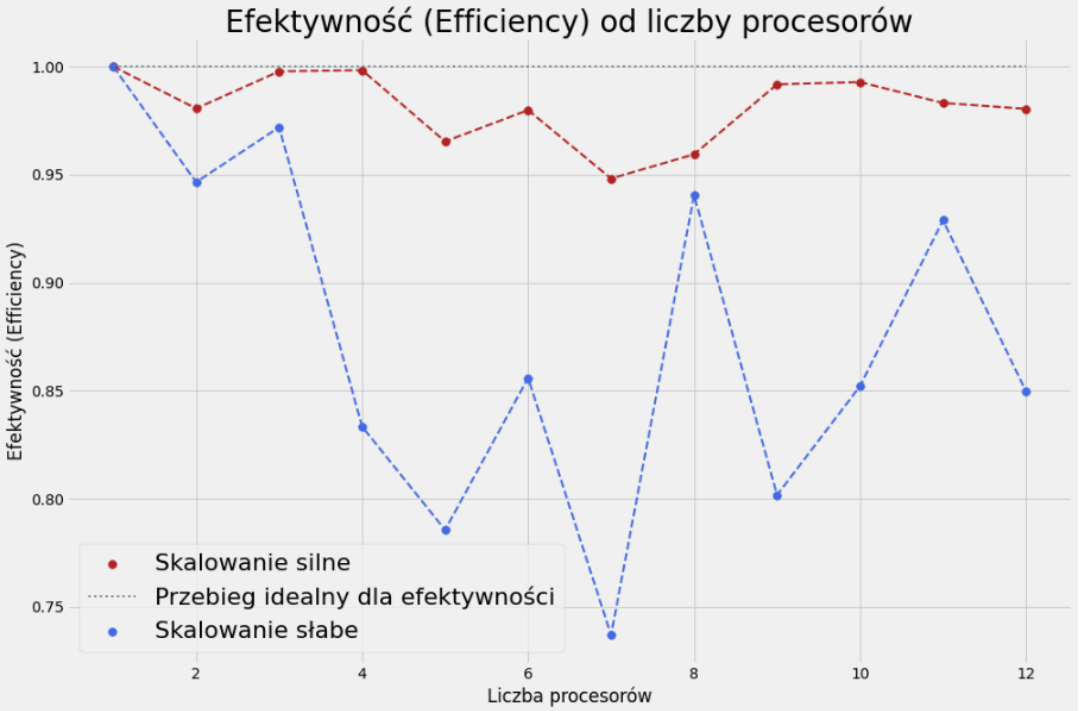
Poniższy wykres przedstawia zależność czasu obliczeń od liczby procesorów, dla skalowania słabego oraz silnego, w środowisku Ares.



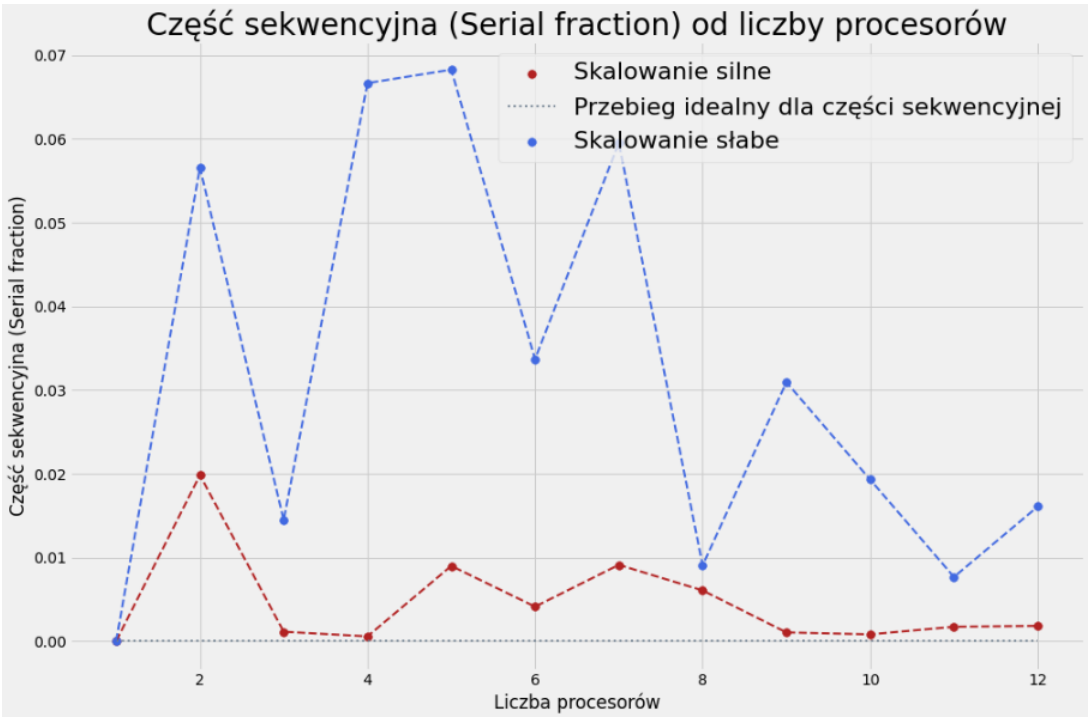
Kolejny wykres przedstawia zależność przyspieszenia (speed up) od liczby procesorów, dla obu wariantów skalowania w środowisku Ares.



Wykres przedstawia zależność efektywności programu od liczby procesorów, dla obu wariantów skalowania w środowisku Ares.

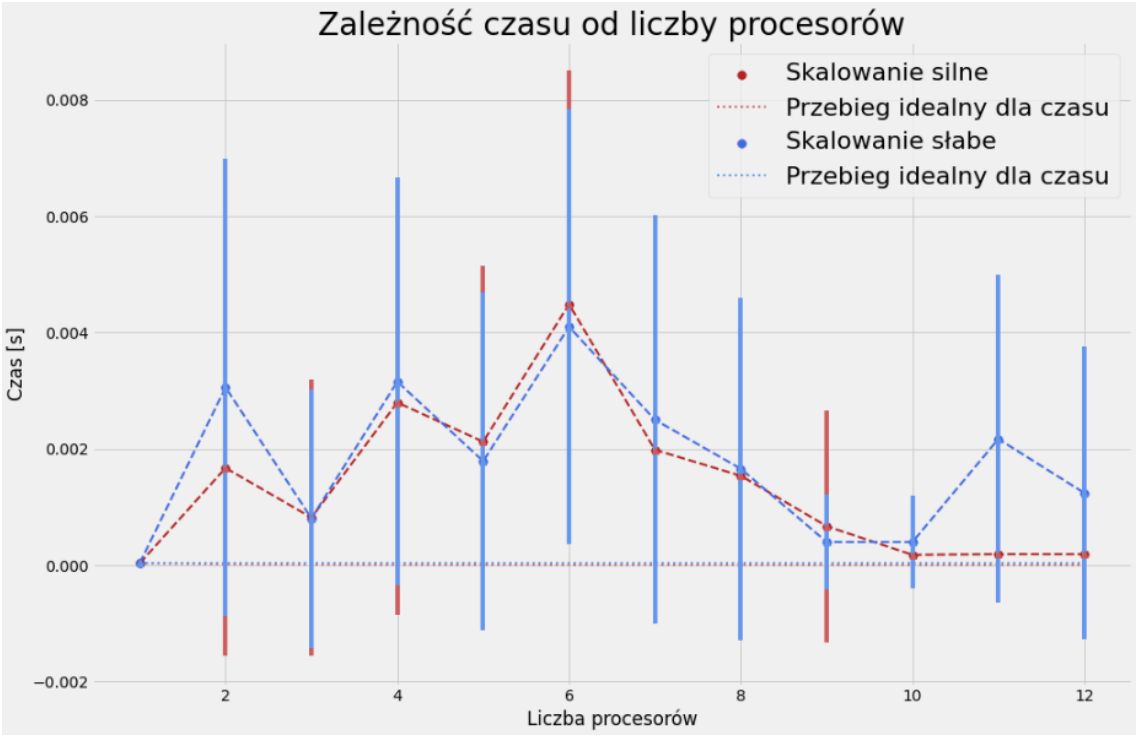


Ostatni wykres przedstawia zależność rozmiaru części sekwencyjnej programu od liczby procesorów, dla obu wariantów skalowania w środowisku Ares.

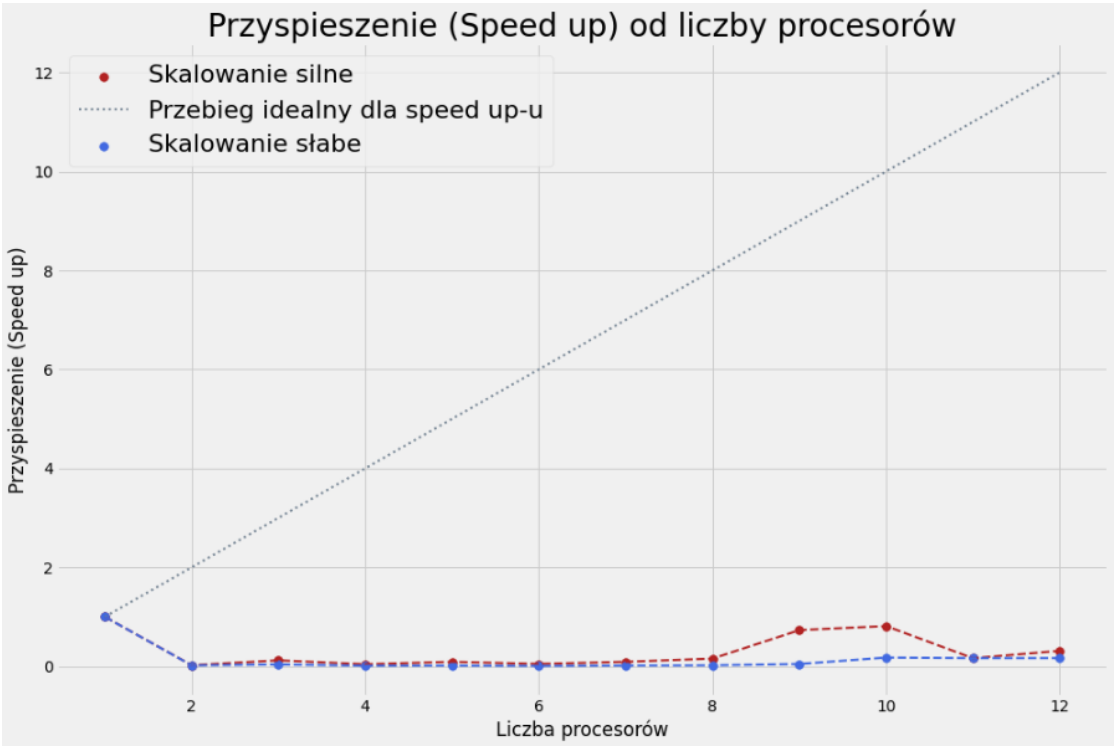


➔ **dla małego problemu**

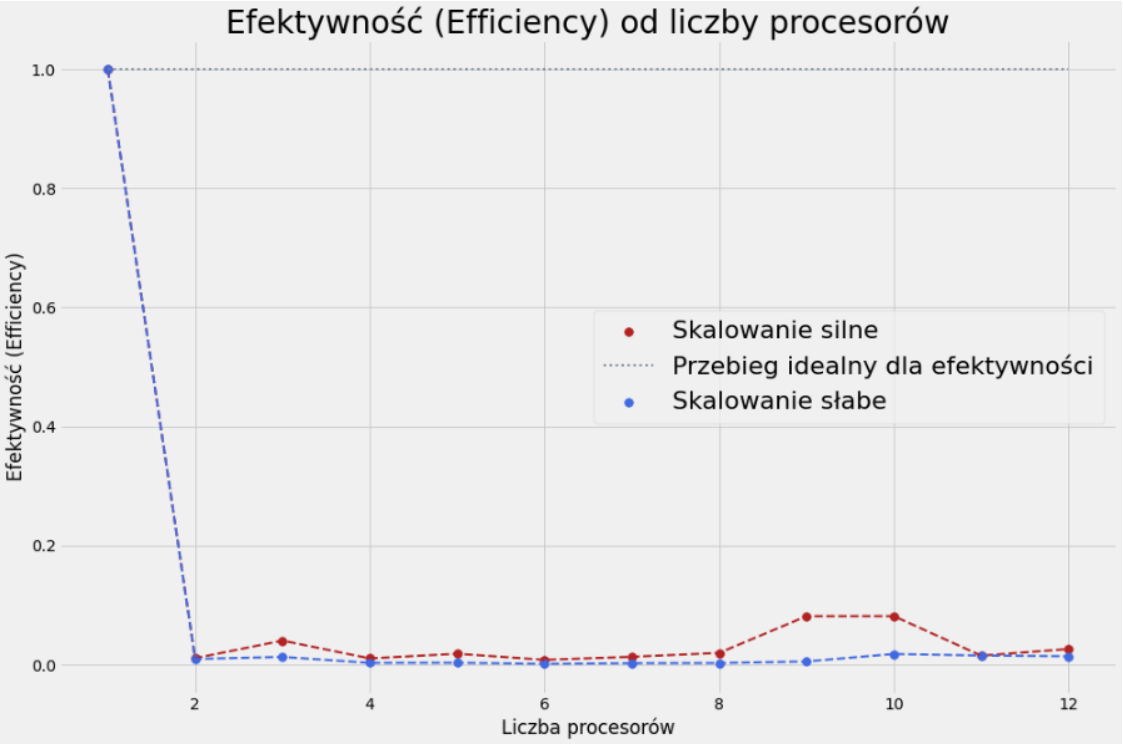
Poniższy wykres przedstawia zależność czasu obliczeń od liczby procesorów, dla skalowania słabego oraz silnego, w środowisku Ares.

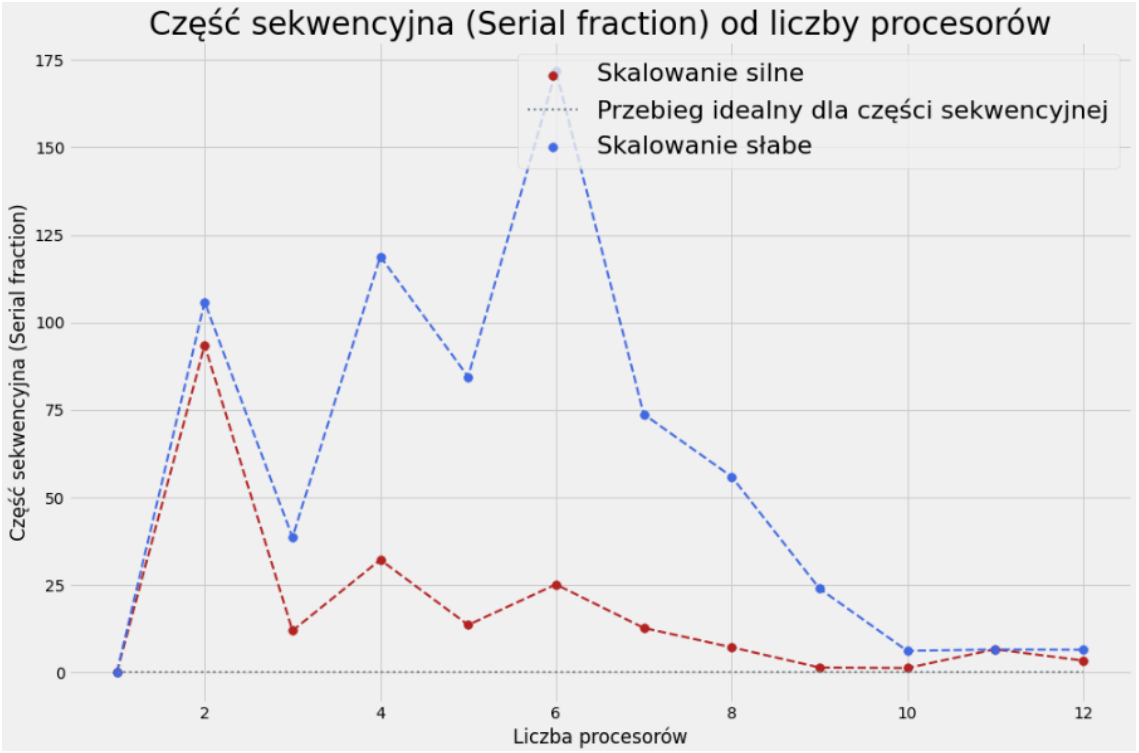


Kolejny wykres przedstawia zależność przyspieszenia (speed up) od liczby procesorów, dla obu wariantów skalowania w środowisku Ares.



Wykresy poniżej przedstawiają zależność efektywności oraz części sekwencyjnej programu od liczby procesorów, dla obu wariantów skalowania w środowisku Ares.





Wyniki znajdują się w plikach *ares_small.txt*, *ares_medium.txt* oraz *ares_large.txt.*

➔ **Wartość π**

Poniższa tabela przedstawia uzyskane wartości przybliżenia liczby π.

	Obliczone PI	STD	Błąd
vCluster	3.141595	0.000088	-0.000002
Ares Mały	3.160303	0.241430	-0.018710
Ares Średni	3.141696	0.001752	-0.000103
Ares Duży	3.141594	0.000015	-0.000001

Wnioski

- ➔ Dla problemu dużego oraz średniego, w obu wariantach (skalowaniu silnym oraz skalowaniu słabym) zależność czasu oraz przyspieszenia od liczby procesorów jest zbliżona do idealnego teoretycznego przebiegu.
- ➔ Dla problemu dużego, oba skalowania utrzymały efektywność powyżej 0.93.
- ➔ Dla problemu dużego, część sekwencyjna obu skalowań była zbliżona i nie przekroczyła 0.0082.
- ➔ Dla problemu średniego, skalowanie silne dało lepsze wyniki niż skalowanie słabe.
- ➔ Dla problemu średniego, dla skalowania silnego efektywność utrzymuje się na poziomie wyższym niż 0.95. Natomiast w skalowaniu słabym, efektywność jest gorsza i dużo bardziej zmienna, utrzymuje się na poziomie ponad 0.7.
- ➔ Dla problemu średniego, część sekwencyjna obu skalowań była zbliżona i nie przekroczyła 0.07.
- ➔ Dla problemu małego, rozmiar zadania okazał się zbyt mały i zrównoleglenie go nie przyniosło żadnego zysku.
- ➔ Im większy był rozmiar problemu tym mniejsze było odchylenie standardowe zmierzonego czasu.
- ➔ Najlepsze przybliżenie wartości π otrzymałam z eksperymentu na Aresie z dużym rozmiarem problemu. Najgorszy wynik dał eksperyment z małym rozmiarem problemu na Aresie.