Imię i Nazwisko	Kierunek	Rok studiów i grupa
Patryk Twardosz	Informatyka Techniczna	I rok, Gr. 9
Data	**Numer i temat sprawozdania**	
18.01.2025r.	Lab 11, 12 i 13 – MPI i pomiary wydajności	

Lab11:

Cel:

- Opanowanie podstaw programowania z przesyłaniem komunikatów MPI.

Zadanie:

1. Przygotowanie projektu
2. Uzupełnienie programu o przesyłanie adresu internetowego węzła nadawcy.

```
if(size>1){

  if( rank != 0 ){ dest=0; tag=0;
    MPI_Send( hostname, HOSTNAME_MAX, MPI_CHAR, dest, tag,
              MPI_COMM_WORLD );
  } else {
    char recvhostname[HOSTNAME_MAX];

    for( i=1; i<size; i++ ) {
      MPI_Recv( recvhostname, HOSTNAME_MAX, MPI_CHAR,
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status );
      printf("Proces %d wysłał hostname: %s\n",
             status.MPI_SOURCE, recvhostname);
    }
  }
}
else {
  printf("Pojedynczy proces o randze: %d (brak komunikatów)\n", rank);
}
```

Wszystkie procesy potomne otrzymują nazwę hosta nadrzędnego

```
twarug@Twarug:/mnt/a/School/AGH/Sem_5/Paral
mpicc -c -g -DDEBUG MPI_simple.c
mpicc -g -DDEBUG MPI_simple.o -o out -lm
mpiexec  -np 4 ./out
Proces 2 wysłał hostname: Twarug
Proces 3 wysłał hostname: Twarug
Proces 1 wysłał hostname: Twarug
```

3. Opracowanie programu propagującego komunikaty w konwencji pierścienia

```c
  if (rank == 0) {
    value = 1;
    printf("Proces %d inicjuje sztafetę z liczbą %d\n", rank, value);

    MPI_Send(&value, 1, MPI_INT, next, 0, MPI_COMM_WORLD);
  }
  else {
    MPI_Recv(&value, 1, MPI_INT, prev, 0, MPI_COMM_WORLD, &status);

    printf("Proces %d odebrał liczbę %d od procesu %d\n", rank, value,
status.MPI_SOURCE);
    value++;

    if (rank != size - 1)
      MPI_Send(&value, 1, MPI_INT, next, 0, MPI_COMM_WORLD);
  }
```

```
mpicc -c -g -DDEBUG main.c
mpicc -g -DDEBUG main.o -o out -lm
mpiexec  -np 4 ./out
Proces 3 odebrał liczbę 3 od procesu 2
Proces 0 inicjuje sztafetę z liczbą 1
Proces 1 odebrał liczbę 1 od procesu 0
Proces 2 odebrał liczbę 2 od procesu 1
```

4. Przesyłanie struktur między procesami

```
typedef struct {
    char name[50];
    float pi;
    int num;
} Data;


// Pakowanie danych

 Data data = {"Patryk", 21.37, 8080};
        void* buffer = malloc(BUFFER_SIZE);
        int position = 0;

        MPI_Pack(data.name, 50, MPI_CHAR, buffer, BUFFER_SIZE, &position,
MPI_COMM_WORLD);
        MPI_Pack(&data.pi, 1, MPI_FLOAT, buffer, BUFFER_SIZE, &position,
MPI_COMM_WORLD);
        MPI_Pack(&data.num, 1, MPI_INT, buffer, BUFFER_SIZE, &position,
MPI_COMM_WORLD);

        MPI_Send(buffer, position, MPI_PACKED, next, tag, MPI_COMM_WORLD);
        free(buffer);


// Rozpakowanie danych
 void *buffer = malloc(BUFFER_SIZE);
        Data recData;
        int position = 0;

        MPI_Recv(buffer, BUFFER_SIZE, MPI_PACKED, prev, tag,
MPI_COMM_WORLD, &status);
        MPI_Unpack(buffer, BUFFER_SIZE, &position, recData.name, 50,
MPI_CHAR, MPI_COMM_WORLD);
        MPI_Unpack(buffer, BUFFER_SIZE, &position, &recData.pi, 1,
MPI_FLOAT, MPI_COMM_WORLD);
        MPI_Unpack(buffer, BUFFER_SIZE, &position, &recData.num, 1,
MPI_INT, MPI_COMM_WORLD);

        printf("Proces %d otrzymał dane: %s, %d, %.2f\n", rank,
recData.name, recData.num, recData.pi);
        free(buffer);
```

5. Wykonanie procesowania potokowaego struktury

```
        void *buffer = malloc(BUFFER_SIZE);
        int position = 0;
        MPI_Pack(dataset[i].text, 50, MPI_CHAR, buffer, BUFFER_SIZE,
&position, MPI_COMM_WORLD);
        MPI_Pack(&dataset[i].a_count, 1, MPI_INT, buffer, BUFFER_SIZE,
&position, MPI_COMM_WORLD);
        MPI_Pack(&dataset[i].length, 1, MPI_INT, buffer, BUFFER_SIZE,
&position, MPI_COMM_WORLD);
        MPI_Pack(&dataset[i].is_done, 1, MPI_INT, buffer, BUFFER_SIZE,
&position, MPI_COMM_WORLD);
        MPI_Send(buffer, position, MPI_PACKED, next, tag, MPI_COMM_WORLD);
        free(buffer);

        buffer = malloc(BUFFER_SIZE);
        position = 0;
        MPI_Recv(buffer, BUFFER_SIZE, MPI_PACKED, prev, tag,
MPI_COMM_WORLD, &status);
        MPI_Unpack(buffer, BUFFER_SIZE, &position, dataset[i].text, 50,
MPI_CHAR, MPI_COMM_WORLD);
        MPI_Unpack(buffer, BUFFER_SIZE, &position, &dataset[i].a_count, 1,
MPI_INT, MPI_COMM_WORLD);
        MPI_Unpack(buffer, BUFFER_SIZE, &position, &dataset[i].length, 1,
MPI_INT, MPI_COMM_WORLD);
        MPI_Unpack(buffer, BUFFER_SIZE, &position, &dataset[i].is_done, 1,
MPI_INT, MPI_COMM_WORLD);
        free(buffer);
```

```
if(rank == size - 1) {
          break;
        } else {
          void *buffer = malloc(BUFFER_SIZE);
          int position = 0;
          MPI_Pack(data.text, 50, MPI_CHAR, buffer, BUFFER_SIZE,
&position, MPI_COMM_WORLD);
          MPI_Pack(&data.a_count, 1, MPI_INT, buffer, BUFFER_SIZE,
&position, MPI_COMM_WORLD);
          MPI_Pack(&data.length, 1, MPI_INT, buffer, BUFFER_SIZE,
&position, MPI_COMM_WORLD);
          MPI_Pack(&data.is_done, 1, MPI_INT, buffer, BUFFER_SIZE,
&position, MPI_COMM_WORLD);

          MPI_Send(buffer, position, MPI_PACKED, next, tag,
MPI_COMM_WORLD);

          break;
        }
      }

      if (rank == 1) {
        count_a_in_text(&data);
      } else if (rank == 2) {
        add_a(&data);
      } else if (rank == 3) {
        calc_len(&data);
      }

      buffer = malloc(BUFFER_SIZE);
      position = 0;

      MPI_Pack(data.text, 50, MPI_CHAR, buffer, BUFFER_SIZE, &position,
MPI_COMM_WORLD);
```

```
mpicc -g -DDEBUG main.o -o out -lm
mpiexec  -np 4 ./out
Proces 0: "Hello Worlda" - 'a': 0, Długość: 12, Zakończone: 0
Proces 0: "Tesowanie testamia" - 'a': 2, Długość: 18, Zakończone: 0
Proces 0: "Patryk Twardosza" - 'a': 2, Długość: 16, Zakończone: 0
Proces 0: "Hello Worldaa" - 'a': 1, Długość: 13, Zakończone: 0
Proces 0: "Tesowanie testamiaa" - 'a': 3, Długość: 19, Zakończone: 0
Proces 0: "Patryk Twardoszaa" - 'a': 3, Długość: 17, Zakończone: 0
Proces 0: "Hello Worldaaa" - 'a': 2, Długość: 14, Zakończone: 0
Proces 0: "Tesowanie testamiaaa" - 'a': 4, Długość: 20, Zakończone: 0
Proces 0: "Patryk Twardoszaaa" - 'a': 4, Długość: 18, Zakończone: 0
Proces 0: "Hello Worldaaaa" - 'a': 3, Długość: 15, Zakończone: 0
Proces 0: "Tesowanie testamiaaaa" - 'a': 5, Długość: 21, Zakończone: 1
Proces 0: "Patryk Twardoszaaaa" - 'a': 5, Długość: 19, Zakończone: 1
Proces 0: "Hello Worldaaaaa" - 'a': 4, Długość: 16, Zakończone: 0
Proces 0: "Hello Worldaaaaaa" - 'a': 5, Długość: 17, Zakończone: 1
Proces 3 zakończył pracę
Proces 0 zakończył pracę
Proces 1 zakończył pracę
Proces 2 zakończył pracę
```

Wnioski:

- **Opanowanie podstaw MPI**
  Przeprowadzono ćwiczenia umożliwiające poznanie podstawowych operacji z przesyłaniem komunikatów w środowisku MPI, takich jak inicjalizacja, przesyłanie danych między procesami oraz finalizacja.
- **Realizacja modelu sztafety w MPI**
  Opracowanie programu propagującego komunikaty w konwencji pierścienia pozwoliło na zrozumienie mechanizmów komunikacji między procesami. Szczególnie istotne było wyznaczenie ról poszczególnych procesów (poprzednika i następcy) oraz modyfikacja danych przesyłanych w pierścieniu.
- **Praktyczne zastosowanie struktur danych w MPI**
  Stworzenie "bogatej" struktury danych w języku C, a następnie jej przesyłanie za pomocą typu MPI_PACKED, umożliwiło pogłębienie umiejętności związanych z zaawansowaną obsługą danych w środowisku MPI.
- **Zastosowanie przetwarzania potokowego**
  Rozwinięcie programu do realizacji przetwarzania potokowego pozwoliło na wdrożenie schematu, w którym wiele danych jest przetwarzanych jednocześnie w sposób równoległy. To ćwiczenie pokazało potencjalne przyspieszenie wynikające z równoległości.
- **Efektywność i testowanie działania**
  Programy były uruchamiane i testowane na różnych liczbach procesów, co umożliwiło ocenę poprawności działania oraz porównanie wyników w zależności od parametrów uruchomienia.
- **Nowy typ danych w MPI**
  Użycie funkcji MPI_Type_create_struct do utworzenia nowego typu danych dla struktury w języku C podkreśliło możliwości dostosowywania komunikacji w MPI do specyficznych wymagań aplikacji.

Lab12:

Cel:

- Doskonalenie podstaw programowania z przesyłaniem komunikatów MPI

Zadanie:

1. Przygotowanie projektu
2. Analiza sekwencyjnego obliczania liczby PI
3. Zrównoleglenie obliczeń liczby PI

```
if (rank == 0) {
    printf("Podaj maksymalną liczbę wyrazów do obliczenia przybliżenia PI\n");
    scanf("%d", &max_liczba_wyrazow);
  }
  // Rozsyłanie liczby iteracji do wszystkich procesów
  MPI_Bcast(&max_liczba_wyrazow, 1, MPI_INT, 0, MPI_COMM_WORLD);

  // Obliczanie zakresów iteracji dla każdego procesu
  int my_start = rank * (max_liczba_wyrazow / size);
  int my_end = (rank == size - 1) ? max_liczba_wyrazow : my_start +
(max_liczba_wyrazow / size);
  for (int i = my_start; i < my_end; i++) {
    int j = 1 + 4 * i;
    local_sum_plus += 1.0 / j;
    local_sum_minus += 1.0 / (j + 2.0);
  }
  // Redukcja wyników lokalnych do procesu 0
  MPI_Reduce(&local_sum_plus, &global_sum_plus, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
  MPI_Reduce(&local_sum_minus, &global_sum_minus, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);

  if (rank == 0) {
    SCALAR pi_approx = 4 * (global_sum_plus - global_sum_minus);
    printf("PI obliczone: \t\t\t%20.15lf\n", pi_approx);
    printf("PI z biblioteki matematycznej: \t%20.15lf\n", M_PI);
  }
```

```
twarug@Twarug:/mnt/a/School/AGH/Sem_5/Paralle1/lab12/MPI_pi$ mak
mpicc -g -DDEBUG MPI_pi.o -o out -lm
mpiexec  -np 4 ./out
Podaj maksymalną liczbę wyrazów do obliczenia przybliżenia PI
10000
PI obliczone:                   3.141542653589825
PI z biblioteki matematycznej:  3.141592653589793
```

4. Zmniejszenie ilości wymaganych do równoległych obliczeń iloczynu macierzy i wektora, użycie Bcast i Scatter

```
   MPI_Scatter( a, WYMIAR*n_wier, MPI_DOUBLE, a_local,
WYMIAR*n_wier, MPI_DOUBLE, 0, MPI_COMM_WORLD );

   MPI_Bcast(x, WYMIAR, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Około 50 linii kodu zostało zastąpione 2. Pokazuje to rozbudowanie biblioteki MPI jednocześnie dając możliwość na szczegółową implementacje przy jego użyciu.

Wnioski:

- **Obliczanie liczby π z szeregu Leibniza**
  Zrównoleglenie obliczeń liczby π umożliwiło efektywne podziałanie pracy między procesy. Procesy indywidualnie obliczały swoje części sumy, a proces o randze 0 zbierał i sumował wyniki, co pokazało możliwości redukcji komunikacyjnej w MPI.
- **Równoległe mnożenie macierzy przez wektor**
  Implementacja algorytmu mnożenia macierz-wektor pokazała znaczenie poprawnej dekompozycji danych i synchronizacji procesów. Testowanie wyników z użyciem komunikacji punkt-punkt oraz grupowej umożliwiło analizę różnic wydajności i czytelności kodu.
- **Optymalizacja przez komunikację grupową**
  Zastąpienie wymiany punkt-punkt (MPI_Send/MPI_Recv) funkcjami grupowymi (MPI_Bcast, MPI_Gather, MPI_Scatter itp.) znacząco uprościło kod i przyczyniło się do poprawy jego czytelności i wydajności, szczególnie przy większej liczbie procesów.

Lab13:

Cel:

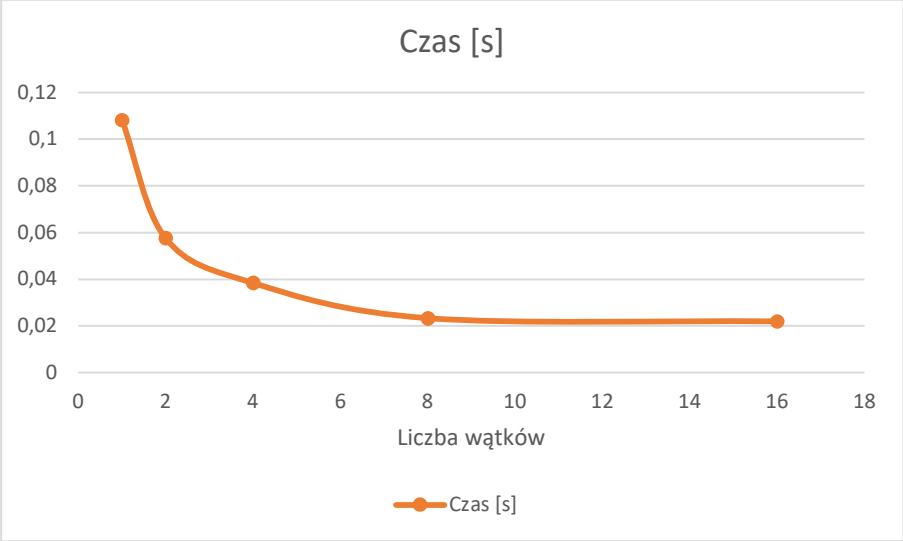
- Doskonalenie umiejętności analizy wydajności programów równoległych

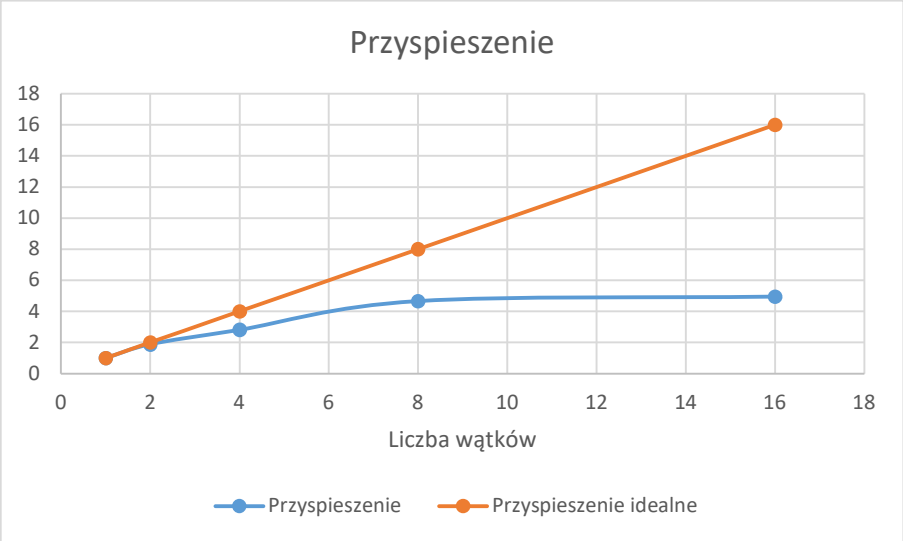
Zadanie:

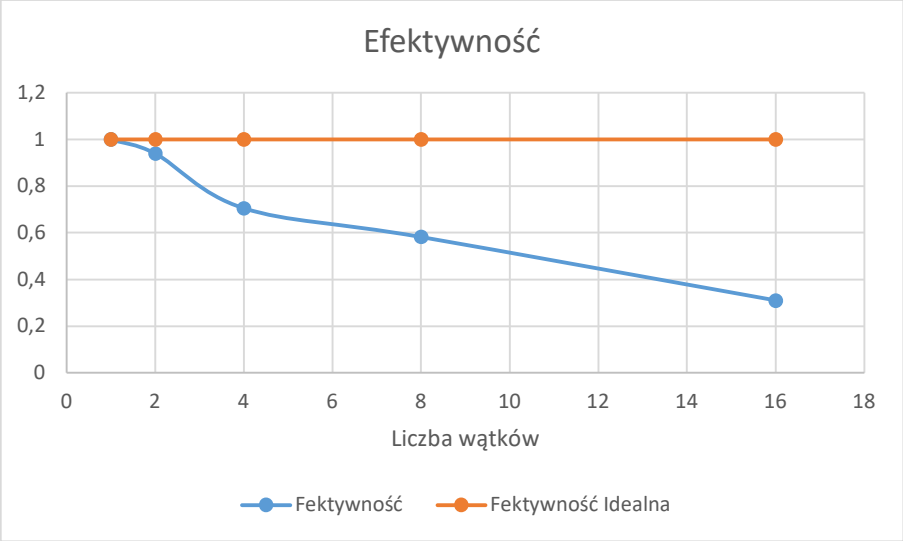
1. Przygotowanie projektu
2. Przeprowadzenie testów dla różnych ilości wątków.
3. Zebranie danych w arkuszu.
4. Wykonanie wykresów.

Całka (OpenMP):

Thread Count	Czas [s]	Przyspieszenie	Fektywność
1	0,108229	1	1
2	0,057564	1,880150789	0,940075
4	0,038389	2,819271145	0,704818
8	0,023229	4,65921908	0,582402
16	0,02184	4,955540293	0,309721

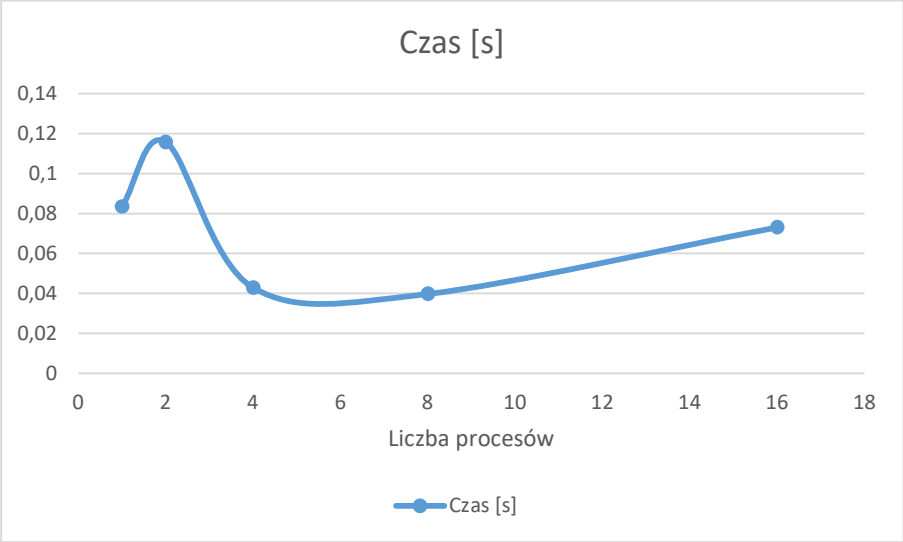


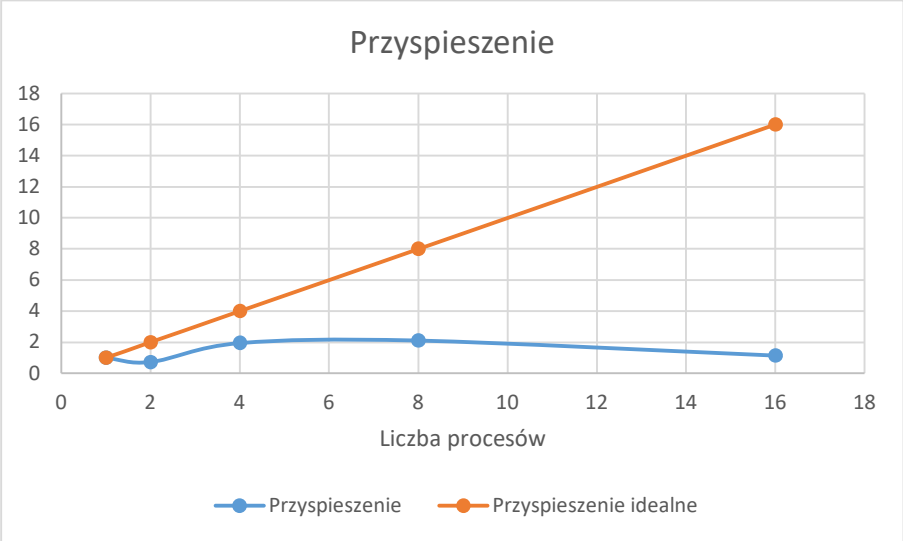

Przyspieszenie
18
16
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
16
18
Liczba wątków
Przyspieszenie
Przyspieszenie idealne


Efektywność
1,2
1
0,8
0,6
0,4
0,2
0
0
2
4
6
8
10
12
14
16
18
Liczba wątków
Fektywność
Fektywność Idealna

Mnożenie Macierz – Wektor (MPI)

Liczba procesów	Czas [s]	Przyspieszenie	Fektywność
1	0,083597	1	1
2	0,115793	0,721952104	0,360976
4	0,042919	1,947785363	0,486946
8	0,039835	2,098581649	0,262323
16	0,073133	1,143081783	0,071443





Wnioski:

- **Całka (OpenMP):**
  - Zależność czasu wykonania od liczby wątków pokazuje poprawę wydajności do pewnego momentu, ale osiągnięcie idealnego przyspieszenia (linearnego wzrostu) nie jest możliwe.
  - Przy większej liczbie wątków narzut związany z synchronizacją oraz zarządzaniem wątkami zaczyna dominować, zmniejszając efektywność obliczeń.
  - Efektywność spada istotnie szybciej niż w przypadku mat_vec, co wskazuje na potencjalne problemy z równomiernym podziałem pracy między wątki.
- **Mat_vec (MPI):**
  - Wydajność zmniejsza się wraz ze wzrostem liczby procesów. Początkowy wzrost szybkości jest ograniczony narzutem komunikacyjnym i zarządzaniem procesami.
  - Dla niewielkiej liczby wątków (1–4), wzrost wydajności jest zauważalny, ale znacząco poniżej idealnego przyspieszenia. Wynika to z rosnącej trudności w efektywnym wykorzystaniu większej liczby rdzeni.
  - Wysoka liczba wątków prowadzi do efektu przeciążenia (overhead), co wpływa na spadek efektywności.
  - Skalowalność algorytmu jest ograniczona przez jego charakter (możliwe wąskie gardła w dostępie do pamięci lub synchronizacji).