

# Programowanie równoległe

## Laboratorium 11

# Sprawozdanie

## Zadanie 1:

1. Utworzenie katalogu roboczego.
2. Opracowanie programu obliczającego liczbę  $\pi$  z szeregu Leibniza. Proces o randze 0 powinien pobrać informację o liczbie sumowanych składników (podaną jako parametr przy uruchomieniu programu, z klawiatury itp.). Liczba składników szeregu powinna zostać równo rozdzielona między procesy liczące sumy częściowe (należy rozwiązać problem w przypadku niepodzielności liczby składników przez liczbę procesów liczących)
3. Testowanie opracowanego programu (sprawdzenie poprawności otrzymanego wyniku).

Kod źródłowy rozwiązania:

```
#include <iostream>
#include <mpi.h>

int main(int argc, char *argv[]) {
    int npes;
    int myrank;
    double start, end;

    int root = 0, n = 0;

    MPI_Init(&argc, &argv);

    start = MPI_Wtime();
    MPI_Comm_size(MPI_COMM_WORLD, &npes);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    double localSum = 0;
    double globalSum = 0;

    if(!myrank) {
        std::cout << "n: ";
        std::cin >> n;
    }

    MPI_Bcast(&n, 1, MPI_INT, root, MPI_COMM_WORLD);
```

```

if(myrank) {
    for(int denominator = myrank * 2 - 1; denominator < n *
2; denominator += (npes - 1) * 2) {
        localSum += 4. / denominator * (myrank % 2 ? 1 :
-1);
    }
}

MPI_Reduce(&localSum, &globalSum, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
end = MPI_Wtime();

if(!myrank) {
    std::cout << "Calculated PI is " << globalSum <<
std::endl;
    std::cout << "End of process " << end - start <<
std::endl;
}

MPI_Finalize();
return 0;
}

```

Wydruk programu dla n = 10:

```

n: 10
Calculated PI is 3.04184
End of process 1.06784

```

Wydruk programu dla n = 100:

```

n: 100
Calculated PI is 3.13159
End of process 1.08084

```

Wydruk programu dla n = 1000:

```

n: 1000
Calculated PI is 3.14059
End of process 1.5941

```

Wydruk programu dla n = 10000:

```

n: 10000

```

```
Calculated PI is 3.14149
End of process 1.52072
```

Wnioski:

- Szereg Leibniza może być wykorzystany do obliczenia liczby  $\pi$ .
- Dokładność wyniku rośnie ze zwiększeniem liczby elementów sumy n.
- Czas obliczenia wyniku też rośnie.

## Zadanie 2:

1. Stałą gamma Eulera definiuje się jako granicę ciągu  $g_n$ . Napisz program współbieżny w MPI, który oblicza  $g_n$ , przy następujących założeniach:
  - -n -liczba elementów sumy jest wczytywana z klawiatury
  - -p -liczba procesów, które będą liczyć sumę jest stałą programu
  - -każdy z p procesów liczy pewien fragment sumy tzn. przykładowo dla n=100, p=10, pierwszy proces liczy sumę od 1 do 10, drugi od 11 do 20, itd., na zakończenie sumy częściowe są dodawane i odejmowany jest logarytm, uwaga: n nie musi być podzielne przez p
  - -wynik jest wyświetlany na ekranie

Kod źródłowy rozwiązania:

```
#include <iostream>
#include <mpi.h>
#include <math.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int npes;
    int myrank;
    double start, end;

    int root = 0, n = 0;

    MPI_Init(&argc, &argv);

    start = MPI_Wtime();
    MPI_Comm_size(MPI_COMM_WORLD, &npes);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    double localSum = 0;
```

```

double globalSum = 0;

if(!myrank) {
    std::cout << "n: ";
    std::cin >> n;
}

MPI_Bcast(&n, 1, MPI_INT, root, MPI_COMM_WORLD);

if(myrank) {
    int portion = n / (npes - 1);
    int extra = n % (npes - 1);
    int start = (myrank - 1) * portion + std::min(myrank - 1,
extra);

        for(int i = start; i < start + portion + (myrank <= extra
? 1 : 0); ++i) {
            localSum += 1. / (i + 1);
        }
}

MPI_Reduce(&localSum, &globalSum, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
globalSum -= log(n);

end = MPI_Wtime();

if(!myrank) {
    std::cout << "Calculated gamma is " << globalSum <<
std::endl;
    std::cout << "End of process " << end - start <<
std::endl;
}

MPI_Finalize();
return 0;
}

```

Wydruk programu dla n = 10:

```

n: 10
Calculated gamma is 0.626383
End of process 1.11893

```

Wydruk programu dla n = 100:

```
n: 100
Calculated gamma is 0.582207
End of process 1.25086
```

Wydruk programu dla n = 1000:

```
n: 1000
Calculated gamma is 0.577716
End of process 1.46196
```

Wydruk programu dla n = 10000:

```
n: 10000
Calculated gamma is 0.577266
End of process 1.59661
```

Wnioski:

- Szereg  $g_n$  może być wykorzystany do obliczenia Eulera-Mascheroniego.
- Dokładność wyniku rośnie ze zwiększeniem liczby elementów sumy n.
- Czas obliczenia wyniku też rośnie.