Przykłady - lab. 3

- 3_1 Ping-Pong
  - 3_2 Ring
- 3_3 Porównanie MPI_Bcast z MPI_Send i MPI_Recv
- 3_4 Podział komunikatora
- 3_5 Obliczanie średniej liczb z tablicy za pomocą MPI_Reduce
  - 3_6 Obliczanie odchylenia standardowego liczb z tablicy
  - 3_7 Obliczanie średniej liczb z tablicy za pomocą MPI_Scatter i MPI_Gather
- 3_8 Pliki

3_1 Ping-Pong

- Procesy używają MPI_Send i MPI_Recv do ciągłego odbijania wiadomości od siebie, dopóki nie zdecydują się zatrzymać.
- Program ma być wykonany na dwóch procesach.
- Procesy najpierw określają swojego partnera za pomocą prostej arytmetyki.
- Licznik jest inicjowany do zera i jest zwiększany w każdym kroku Ping-Ponga.
- W miarę zwiększania liczby odbić, procesy na zmianę są nadawcą i odbiorcą.
- Po osiągnięciu limitu (10), procesy przestają wysyłać i odbierać.

p3_2 Ring

- Program przekazuje wartość przez wszystkie procesy w sposób podobny do pierścienia.
- Program inicjalizuje wartość (-1) dla procesu 0 i wartość ta jest przekazywana przez każdy proces.
- Program kończy się, gdy proces 0 otrzyma wartość z ostatniego procesu.
- Uwaga: Zwróć uwagę na to, aby nie doszło do zakleszczenia. Proces 0 upewnia się, że zakończył swoje pierwsze wysłanie, zanim spróbuje odebrać wartość z ostatniego procesu. Wszystkie inne procesy po prostu wywołują MPI_Recv (odbieranie od sąsiedniego procesu niższego), a następnie MPI_Send (wysyłanie wartości do sąsiedniego procesu wyższego), aby przekazać wartość wzdłuż pierścienia. MPI_Send i MPI_Recv będą blokować do momentu przesłania wiadomości.

p3_3 Porównanie MPI_Bcast z MPI_Send i MPI_Recv

- Funkcja o nazwie my_bcast jest prostym opakowaniem dla MPI_Send i MPI_Recv. Proces root wysyła dane do wszystkich innych, podczas gdy inni otrzymują od procesu root.
- Skrypt uruchamiania wykonuje kod przy użyciu n=2, 4, 8, 16 procesów, num_elements = 10000 liczb całkowitych (int* data = (int*)malloc(sizeof(int) * num_elements);) dla 10 powtórzeń (tyle eksperymentów należy wykonać).
- Śledzimy skumulowany czas (pamiętamy o barierze) obu funkcji my_bcast i MPI_Bcast.

my_bcast

```
void my_bcast(void* data, int count, MPI_Datatype datatype, int root,
              MPI_Comm communicator) {
  int world_rank;
  MPI_Comm_rank(communicator, &world_rank);
  int world_size;
  MPI_Comm_size(communicator, &world_size);

  if (world_rank == root) {
    // If we are the root process, send our data to everyone
    int i;
    for (i = 0; i < world_size; i++) {
      if (i != world_rank) {
        MPI_Send(data, count, datatype, i, 0, communicator);
      }
    }
  } else {
    // If we are a receiver process, receive the data from the root
    MPI_Recv(data, count, datatype, root, 0, communicator,
MPI_STATUS_IGNORE);
  }
}
```

p3_4 Podział komunikatora

- Podzielić jeden globalny komunikator na zestaw mniejszych komunikatorów (MPI_Comm_split).
- Przykładowo dzielimy komunikator składający się z 16 procesów na 4 komunikatory po 4 procesy każdy.
- Wyświetlić informacje o procesach komunikatora pierwotnego i o procesach utworzonych komunikatorów.

p3_5 Obliczanie średniej liczb z tablicy za pomocą MPI_Reduce

- Każdy proces tworzy liczby losowe (create_rand_nums) i dokonuje obliczenia local_sum.

- Tworzenie losowej tablicy elementów we wszystkich procesach:
  srand(time(NULL)*world_rank);
  float *rand_nums = NULL;
  rand_nums = create_rand_nums(num_elements_per_proc);

- local_sum jest redukowana do procesu głównego (MPI_SUM).

- Średnia globalna to:
  global_sum/(world_size * num_elements_per_proc).

Algorytm 3_5

- Utwórz losową tablicę elementów we wszystkich procesach.
- Sumuj liczby lokalnie.
- Wydrukuj losowe liczby w każdym procesie.
- Zredukuj wszystkie lokalne sumy do sumy globalnej.

create_rand_nums

```
float *create_rand_nums(int num_elements) {
  float *rand_nums = (float *)malloc(sizeof(float) * num_elements);
  assert(rand_nums != NULL);
  int i;
  for (i = 0; i < num_elements; i++) {
    rand_nums[i] = (rand() / (float)RAND_MAX);
  }
  return rand_nums;
}
```

p3_6 Obliczanie odchylenia standardowego liczb z tablicy

- Aby znaleźć odchylenie standardowe, należy najpierw obliczyć średnią arytmetyczną wszystkich liczb. Każdy proces oblicza local_sum elementów i sumuje je (MPI_Allreduce).
- Po obliczeniu średniej obliczana jest wariancja, czyli średnia arytmetyczna kwadratów odchyleń od ich średniej arytmetycznej.

Algorytm 3_6

- Sumuj liczby lokalnie.
- Zmniejsz wszystkie lokalne sumy do sumy globalnej, aby obliczyć średnią (MPI_Allreduce).
- Oblicz lokalną średnią kwadratów odchyleń.
- Utwórz globalną średnią kwadratów odchyleń w procesie 0 (MPI_Reduce).
- Odchylenie standardowe to pierwiastek kwadratowy ze średniej kwadratów odchyleń.

p3_7 Obliczanie średniej liczb z tablicy za pomocą MPI_Scatter i MPI_Gather

- Wygeneruj losową tablicę liczb w procesie głównym (proces 0).
- Przesłać liczby na wszystkie procesy (MPI_Scatter), nadając każdemu procesowi równą liczbę liczb (elements_per_proc).
- Każdy proces oblicza średnią swojego podzbioru liczb.
- Zbierz wszystkie średnie do procesu 0.
- Następnie proces główny oblicza średnią z tych liczb, aby uzyskać ostateczną średnią.

p3_8

- http://lisboa.lip.pt/computing/publications/2/wednesday/MPI_and_Parallel_IO-handout.pdf