**Sprawozdanie**

**Porównanie wydajności metod przesyłania struktur danych w MPI**

**Wiktor Zmiendak**

1. **Cel zadanie:**

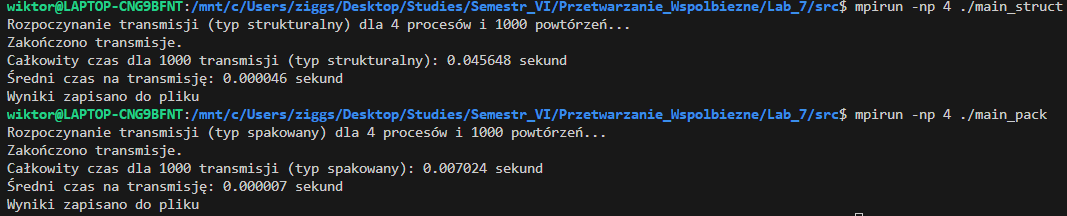
Celem zadania było zapoznanie się z metodami przesyłania złożonych struktur danych w środowisku obliczeń równoległych MPI oraz porównanie wydajności dwóch podejść:

* Użycie MPI\_Type\_create\_struct
* Użycie MPI\_Pack oraz MPI\_Unpack

1. **Implementacja:**

Zrealizowano dwa programy w języku C, oba implementujące komunikację w topologii pierścienia, gdzie każdy proces wysyła dane do swojego następnika i odbiera dane od swojego poprzednika. Komunikacja była powtarzana N\_TRANSMISSIONS razy w pętli, a całkowity czas transmisji był mierzony za pomocą MPI\_Wtime.

1. **Wyniki:**

****

Zalety metody MPI\_Type\_create\_struct:

* Wygoda programowania (mniej kodu do pakowania/rozpakowywania),
* Potencjalnie wyższa wydajność,
* Operacja zdefiniowania typu odbywa się raz, a koszt jest amortyzowany przez wiele transmisji.

Wady metody MPI\_Type\_create\_struct:

* Początkowy narzut związany z definicją i zatwierdzeniem typu,
* Konieczność poprawnego obliczenia offsetów.

Zalety metody MPI\_Pack/MPI\_Unpack:

* Pełna kontrola nad formatem danych w buforze,
* Potencjalnie mniejszy rozmiar przesyłanych danych,
* Serializacja do uniwersalnego formatu.

Wady metody MPI\_Pack/MPI\_Unpack:

* Znacznie więcej kodu do napisania i utrzymania,
* Narzut procesora związany z operacjami pakowania i rozpakowywania *przy każdej transmisji* w pętli,
* Konieczność manualnego zarządzania,
* Ryzyko błędów przy pakowaniu/rozpakowywaniu

1. **Wnioski:**

Obie metody pozwalają na efektywne przesyłanie złożonych struktur danych:

* Dla statycznych struktur przesyłanych wielokrotnie, MPI\_Type\_create\_struct jest często preferowanym podejściem ze względu na potencjalnie lepszą wydajność i mniejszą ilość kodu manualnego,
* MPI\_Pack/MPI\_Unpack zapewnia większą elastyczność i kontrolę, co może być przydatne w bardziej skomplikowanych scenariuszach lub przy dynamicznie zmieniających się danych, ale kosztem dłuższego kodu i potencjalnie niższą wydajnością w prostszych przypadkach.