Projekt na przedmiot: Programowanie Równoległe i Rozproszone
Autor: Mateusz Tutaj
Temat: Wpływ zrównoleglenia obliczeń na liczenie średniej ze zbioru cyfr.

Najprostsze podejście do przedstawionego problemu opiera się na zsumowaniu wszystkich branych pod uwagę cyfr w zbiorze i podzielenie ich przez ich ilość.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}(x_1 + \cdots + x_n)$$

Problematyka: Dla naprawdę dużego zbioru danych, operacja ta sekwencyjnie może zająć bardzo długi okres czasu, zależnie od taktowania procesora oraz źródła pochodzenia danych.

Ograniczenia: Oczywiście sama operacja sumowania nie jest wymagająca dla procesora, o ile wartości nie spowodują przepełnienia(overflow) zmiennej do której, spisujemy dodane wartości. Zależy to oczywiście od wykorzystanego języka programowania. W przypadku c++ dla zmiennych typu int najczęściej może wystąpić zawinięcie lub wynik ujemny, w przypadku typów zmiennoprzecinkowych tzn. float i double natomiast wartość zmieni się w inf lub NaN. Innym problemem, który można napotkać w przypadku tego projektu jest sama ilość danych wymagana aby przedstawić w zauważalny sposób przyspieszenie wprowadzone przez wykorzystanie rozwiązań paralelnych.

Wykorzystane języki programowania:
Python - wygenerowanie zbioru danych i walidacja średniej
C++ - implementacja programu z wykorzystaniem OpenMP i MPI

Biblioteki :
OpenMP(Open Multi-Processing): pozwala na zrównoleglenie programu poprzez rozdzielenie na osobne wątki(threads), oprócz tego operuje na wspólnej pamięci, dzięki czemu nie wprowadza opóźnienia związanego z przesyłaniem danych między wątkami. Jest bardzo intuicyjny do wprowadzenia w przypadku prostych operacji wewnątrz pętli.

MPI(Message Passing Interface): ułatwia skalowalność równoległą, dzięki możliwości osobnego uruchomienia programu na wielu węzłach klastra komputerów, lub lokalnie między procesami. Jedną z cech odróżniających go od OMP, jest korzystanie z rozproszonego modelu pamięci, co za tym idzie, dane muszą zostać początkowo

rozdzielone i przesłane do procesu a następnie zwrócone aby otrzymać wynik. Wiele metod zawartych w bibliotece pozwala na kontrolowanie wykonywania programu MPI_Barrier() wymuszający synchronizacje między procesami oraz MPI_Bcast(), MPI_Scatter(),MPI_Scatterv() i MPI_Reduce() do powiadamiania, przesyłania danych oraz odbierania.

Egzekucja: Dane wykorzystane do obliczeń generowane są w programie *generate_rand.py* przekazać można parametry takie jak ceiling i floor oraz count odpowiedzialnie za granice w których, zostaną one wygenerowane oraz ich ilość. Dzięki wykorzystaniu funkcji generującej wartości z rozkładem normalnych, możliwe jest wybranie wartości float z zakresu -x do +x, dzięki temu przy większym podziale suma wartości które dzielimy na osobne procesy dzięki MPI nie powodują lokalnego przesycenia podczas operacji sumowania ich wartości, zmienne typu float zostały wybrane aby utrudnić zadanie obliczeń poprzez większą ilość zajmowanej pamięci i wykonywanych pod spodem operacji. Z powodów ograniczeń pamięci ram, graniczną wartością zostało wybrane 300.000.000 wartości z zakresu -100000.0 ~ 100000.0.

```
> head -n 10 data.txt
76724.97775349137
1713.5327887414896
-83288.06046516875
-88940.90530759266
-4607.354183162606
16796.105801904356
46763.223687487276
-63930.23960464368
13024.500394671617
-29534.948270163237
```

```
5.22 GiB data.txt
```

Następnie sekwencyjny program verify_average.py, oblicza ich średnią.

```
> python verify_average.py
Verified average from 'data.txt': -0.244641
Total values processed: 300000000
```

Na tej podstawie możliwe było późniejsze wykluczenie sytuacji gdzie dane lub sama wartość średniej zwrócona przez program zrównoleglony jest nieprawidłowa. Przez wcześniej wspomniane wystąpienie overflow zmiennej lub błędy w sychronizacji.

Każdy z programów: avg_mpi i avg_omp zostały uruchomione dla ilości procesów/wątków z przedziału 1~8. Następnie wartość przyspieszenia została wyliczona względem wyniku czasowego dla jednej jednostki obliczającej.

Ilość Processów/Wątków	Przyspieszenie MPI	Przyspieszenie OMP
1	1	1
2	1.630	1.943
3	1.954	2.930
4	2.221	3.841
5	2.358	4.748
6	2.580	5.641
7	2.643	6.383
8	2.733	6.937

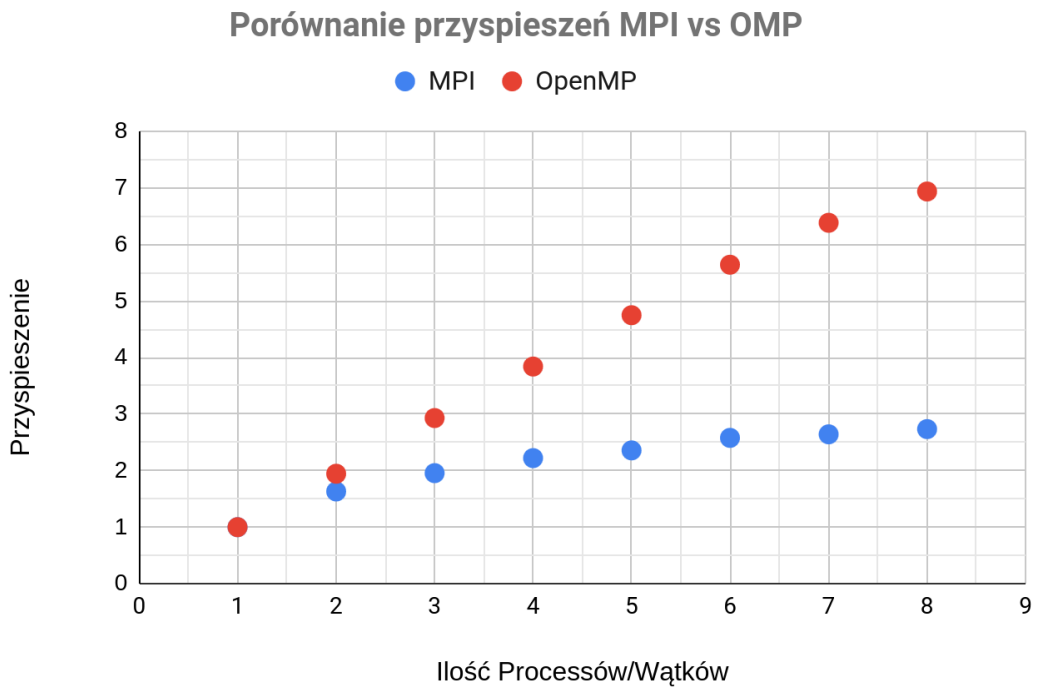

Fig. 1 Wykres zależności przyspieszenia do ilości procesów/wątków.

Na podstawie rozkładu punktów w Fig.1 oraz wartości w tabeli możemy uznać, że powyżej 4 procesów MPI przynosi niskie zyski, najprawdobniej przez narzut komunikacyjny i konieczność synchronizacji stosunkowo dużej ilości danych. OpenMP natomiast osiąga praktycznie liniowe skalowanie do 7 wątków, co wskazuje że jest to lepsze rozwiązanie dla rozpatrywanego problemu w danej skali. Dla znacznie większej ilości danych i bardziej skomplikowanych obliczeń zalety współdzielonej pamięci straciłyby na wartości.

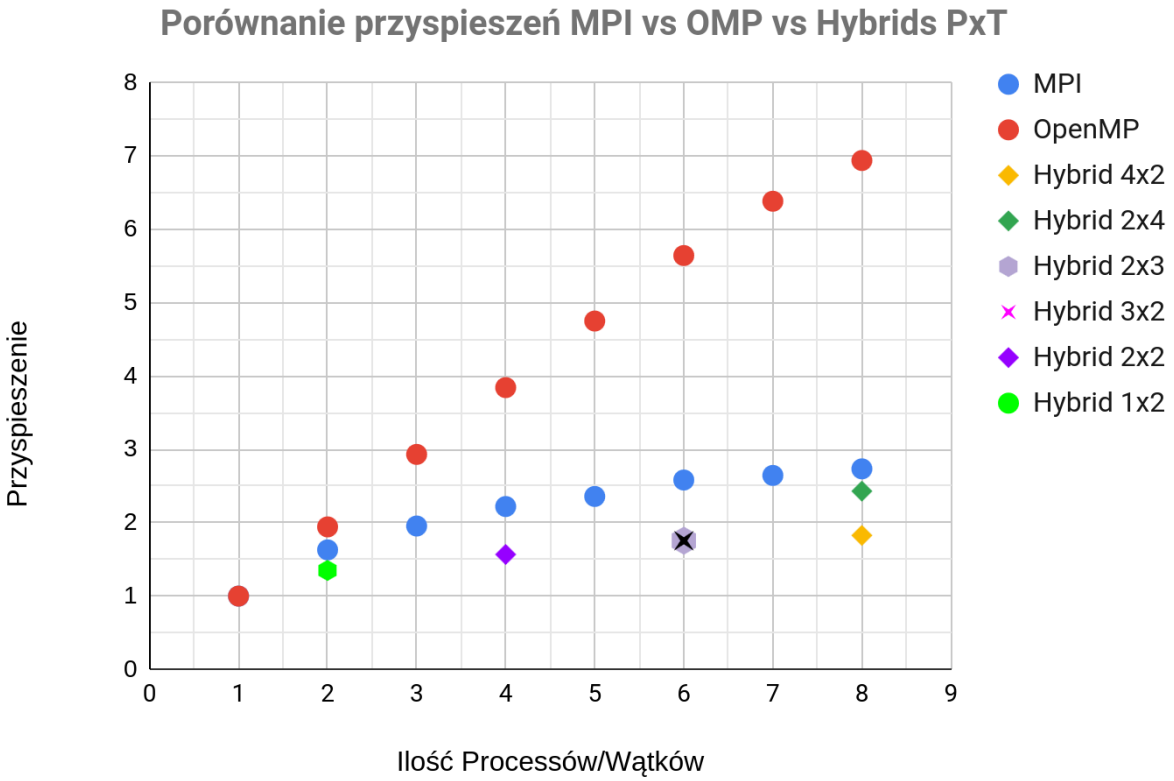

Fig. 2 Wykres zależności przyspieszenia do ilości procesów/wątków wzbogacony o wyniki programu hybrydowego.

Wyniki implementacji hybrydowej dały negatywne skutki w porównaniu do zachowania się programu przy podejściu skupiającym się na jednej metodologii.Dodatkowo ograniczona ilość rdzeni uniemożliwiła dalsze pomiary wyników a ograniczenie pamięci uniemożliwiło wyższy stopień skomplikowania problemu Ostatecznie komunikacja międzyprocesowa i zarządzaniem wątkami wydłużyło czas operacji, co przewyższa zyski z równoległości. Pomijając możliwe wady implementacji ważne jest wzięcie pod uwagę prostoty problemu, która w danej skali utrudnia podkreślenie możliwości biblioteki MPI.