

Porównanie MPI, OpenMP oraz podejścia hybrydowego w programowaniu równoległym

Piotr Gębalski, Adrian Jabłoński

25 maja 2025

Spis treści

1	Wstęp	2
1.1	Cel sprawozdania	2
1.2	Podstawy teoretyczne	2
1.2.1	Open Multi-Processing (OpenMP)	2
1.2.2	Message Passing Interface (MPI)	2
1.2.3	Podejście hybrydowe (MPI + OpenMP)	3
1.2.4	Architektura sprzętowa a wybór technologii	3
2	OpenMP	4
2.1	Efektywność równoległego przetwarzania	4
2.2	Przyśpieszenie obliczeń	5
2.3	Czas wykonania	5
3	MPI	7
3.1	Efektywność równoległego przetwarzania	7
3.2	Przyśpieszenie obliczeń	8
3.3	Czas wykonania	8
4	Hybryda	10
4.1	Efektywność równoległego przetwarzania	10
4.2	Przyśpieszenie obliczeń	11
4.3	Czas wykonania	12
5	Porównanie	13
5.1	Analiza efektywności	13
5.2	Analiza przyśpieszenia	13
5.3	Analiza czasu wykonania	13
5.4	Wnioski	14

Rozdział 1

Wstęp

1.1 Cel sprawozdania

Niniejsze sprawozdanie ma na celu przeprowadzenie szczegółowej analizy porównawczej trzech kluczowych podejść w programowaniu równoległym: Message Passing Interface (MPI), Open Multi-Processing (OpenMP) oraz rozwiązań hybrydowych łączących obydwa paradygmaty. Głównym celem jest zbadanie wydajności zastosowania każdego z tych podejść w różnych scenariuszach obliczeniowych.

Dokument prezentuje teoretyczne podstawy każdej z technologii, a następnie porównuje ich efektywność na podstawie implementacji programu do badania czy dany punkt należy do wykresu.

1.2 Podstawy teoretyczne

1.2.1 Open Multi-Processing (OpenMP)

OpenMP to interfejs programowania aplikacji (API) dla programowania równoległego z pamięcią współdzieloną. Wykorzystuje model fork-join, w którym program rozpoczyna wykonanie jako pojedynczy wątek (master thread), a następnie tworzy zespół wątków roboczych do wykonywania równoległych sekcji kodu.

Główne cechy OpenMP:

- **Model pamięci współdzielonej** – wszystkie wątki mają dostęp do tej samej przestrzeni adresowej
- **Dyrektywy kompilatora** – równoległość wprowadzana poprzez pragma w kodzie źródłowym
- **Przyrostowe równoleglenie** – możliwość stopniowego dodawania równoległości do istniejącego kodu sekwencyjnego
- **Automatyczne zarządzanie wątkami** – środowisko wykonawcze automatycznie zarządza tworzeniem i niszczeniem wątków

1.2.2 Message Passing Interface (MPI)

MPI stanowi standard komunikacji dla programowania równoległego w środowiskach rozproszonych. Opiera się na paradygmacie przekazywania komunikatów między niezależ-

nymi procesami, które nie dzielą wspólnej pamięci. Każdy proces MPI posiada własną, prywatną przestrzeń adresową, a komunikacja odbywa się poprzez jawne wysyłanie i odbieranie wiadomości.

Kluczowe charakterystyki MPI obejmują:

- **Model SPMD** (Single Program, Multiple Data) – ten sam program wykonuje się na różnych procesorach z różnymi danymi
- **Jawną komunikację** – programista musi explicite określić kiedy i jakie dane są przesyłane między procesami
- **Skalowalność** – możliwość efektywnego działania na tysiącach procesorów w klastrach obliczeniowych
- **Przenośność** – standard działa na różnych architekturach od wielordzeniowych stacji roboczych po superkomputery

1.2.3 Podejście hybrydowe (MPI + OpenMP)

Programowanie hybrydowe łączy zalety obu paradygmatów, wykorzystując MPI do komunikacji między węzłami obliczeniowymi a OpenMP do równoległości wewnątrz każdego węzła. To podejście jest szczególnie efektywne w nowoczesnych architekturach, gdzie klastry składają się z węzłów wielordzeniowych.

Zalety podejścia hybrydowego:

- **Hierarchiczna równoległość** – wykorzystanie struktury sprzętowej (węzły + rdzenie)
- **Redukcja komunikacji** – mniej procesów MPI oznacza mniej komunikatów między węzłami
- **Efektywność pamięciowa** – współdzielenie pamięci wewnątrz węzła przy użyciu OpenMP
- **Lepsze dopasowanie do architektury** – naturalne mapowanie na klastry wielordzeniowe

1.2.4 Architektura sprzętowa a wybór technologii

Wybór odpowiedniej technologii programowania równoległego zależy w znacznej mierze od dostępnej architektury sprzętowej:

Systemy z pamięcią współdzieloną (SMP) – wielordzeniowe procesory w jednym węźle, gdzie OpenMP często stanowi optymalne rozwiązanie ze względu na niski koszt komunikacji i prostotę implementacji.

Klastry obliczeniowe – systemy rozproszone składające się z wielu niezależnych węzłów połączonych siecią, gdzie MPI jest naturalnym wyborem ze względu na brak wspólnej pamięci między węzłami.

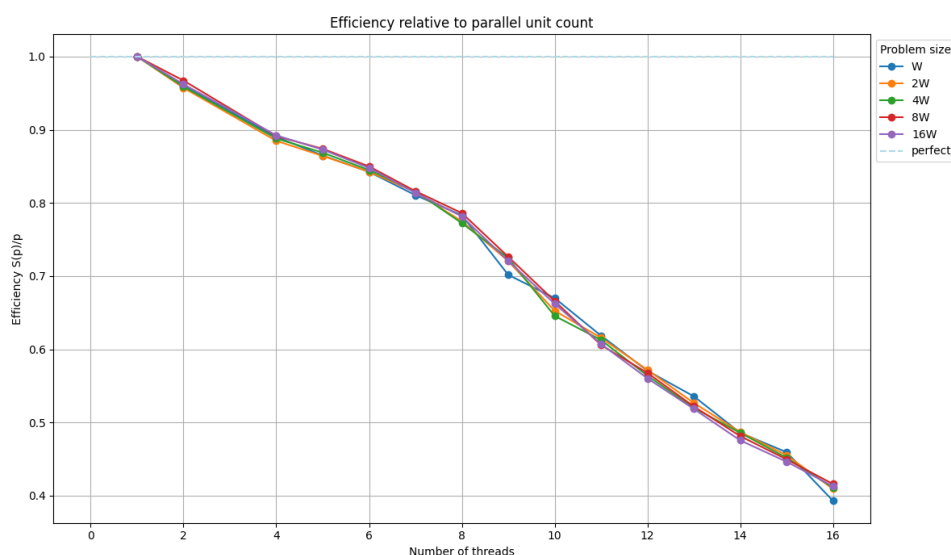
Klastry wielordzeniowe – nowoczesne systemy łączące zalety obu architektur, idealne dla podejścia hybrydowego, pozwalającego na wykorzystanie współdzielonej pamięci wewnątrz węzłów i komunikacji sieciowej między nimi.

Rozdział 2

OpenMP

2.1 Efektywność równoległego przetwarzania

Analiza wykresu efektywności (Rysunek 2.1) pokazuje charakterystyczny spadek efektywności wraz ze wzrostem liczby wątków. Wszystkie testowane rozmiary problemów wykazują podobne zachowanie:



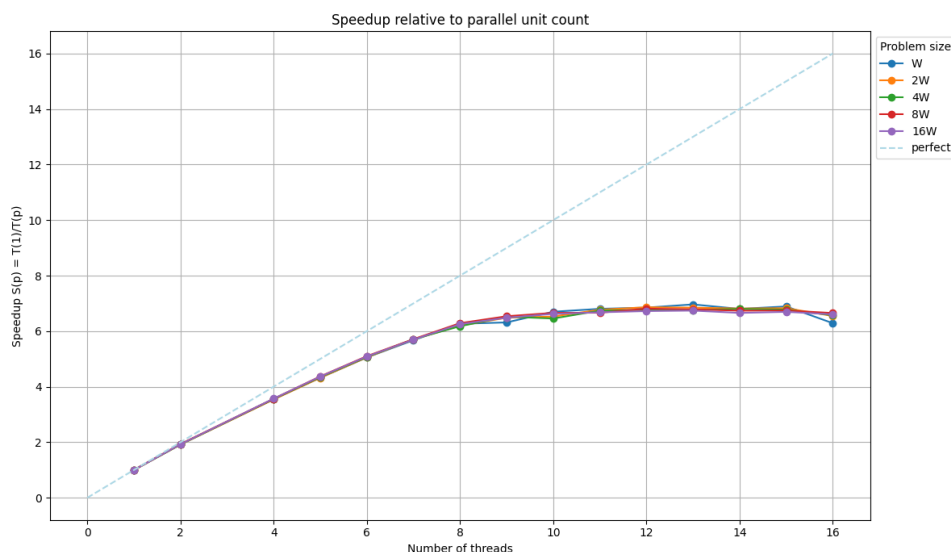
Rysunek 2.1: Efektywność OpenMP

- Efektywność rozpoczyna się od wartości bliskiej 1.0 (100%) dla małej liczby wątków
- Następuje stopniowy spadek efektywności wraz ze wzrostem liczby wątków
- Dla 16 wątków efektywność spada do około 40-45% niezależnie od rozmiaru problemu
- Krzywe dla różnych rozmiarów problemów są bardzo zbliżone, co wskazuje na podobne właściwości skalowania

Obserwowany spadek efektywności wynika z rosnących kosztów synchronizacji między wątkami oraz narzutu związanego z zarządzaniem większą liczbą jednostek wykonawczych. Zjawisko to jest typowe dla programowania równoległego i zgodne z prawem Amdahla.

2.2 Przyspieszenie obliczeń

Wykres przyspieszenia (Rysunek 2.2) ujawnia ograniczenia skalowalności OpenMP w testowanym środowisku:



Rysunek 2.2: Przyspieszenie OpenMP

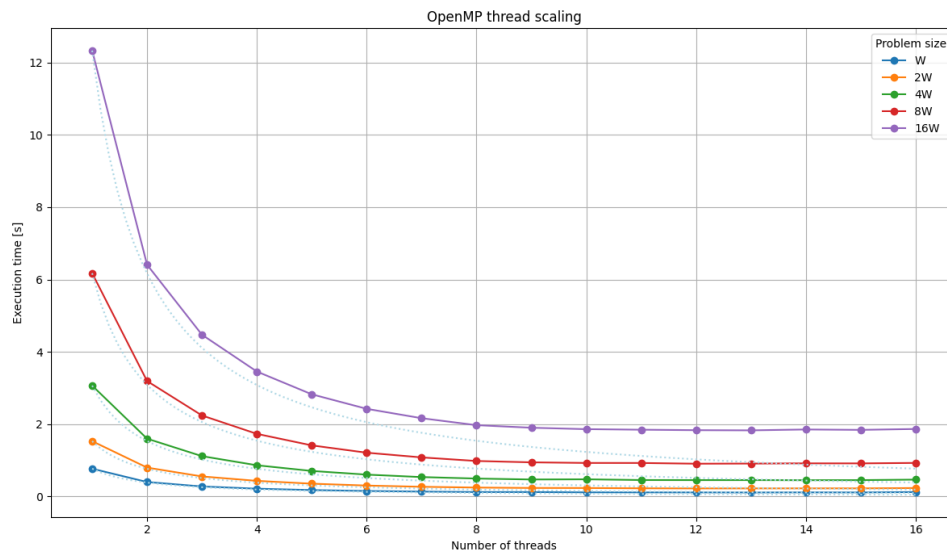
- Maksymalne osiągnięte przyspieszenie wynosi około 6.5-7x dla większości rozmiarów problemów
- Przyspieszenie stabilizuje się po osiągnięciu 8-10 wątków
- Dalsze zwiększanie liczby wątków powyżej 10 nie przynosi istotnych korzyści
- Linia reprezentująca idealne przyspieszenie (perfect scaling) pokazuje znaczną różnicę względem osiągniętych wyników

Plateau przyspieszenia wskazuje na osiągnięcie granicy efektywności równoległego przetwarzania w danej architekturze sprzętowej. Prawdopodobnie wynika to z ograniczeń przepustowości pamięci oraz zwiększonej rywalizacji o zasoby systemowe.

2.3 Czas wykonania

Wykres czasu wykonania (Rysunek 2.3) dostarcza szczegółowego obrazu wydajności dla poszczególnych rozmiarów problemów:

- **Problem 16W:** Najdłuższy czas wykonania (około 12.3s dla 1 wątku), największy potencjał do równoleglenia
- **Problem 8W:** Średni czas wykonania (około 6.1s dla 1 wątku), dobra skalowalność
- **Problem 4W:** Krótszy czas (około 3.1s dla 1 wątku), zauważalny spadek czasu z liczbą wątków



Rysunek 2.3: Przyspieszenie OpenMP

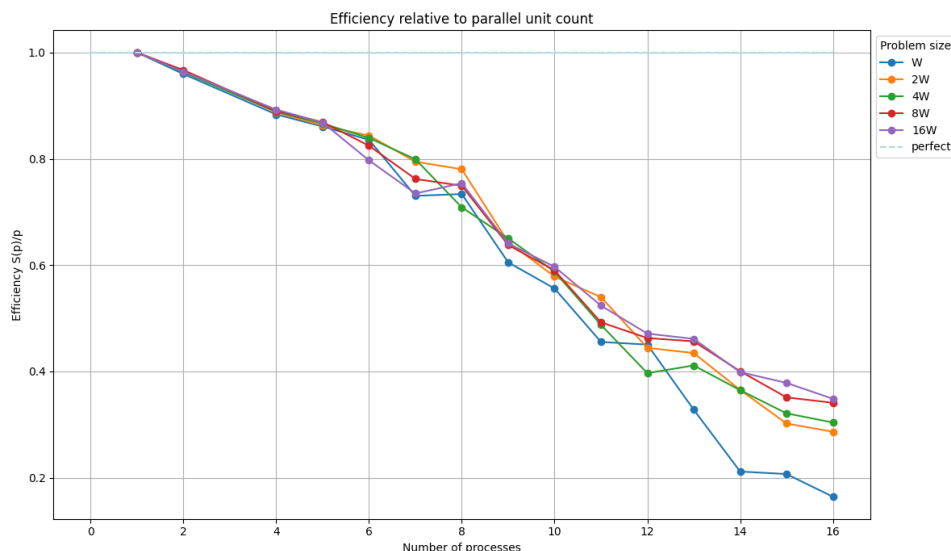
- **Problem 2W:** Szybkie wykonanie (około 1.6s dla 1 wątku), ograniczone korzyści z równoleglenia
- **Problem W:** Najkrótszy czas (około 0.8s dla 1 wątku), minimalne korzyści z dodatkowych wątków

Rozdział 3

MPI

3.1 Efektywność równoległego przetwarzania

Analiza wykresu efektywności (Rysunek 3.1) pokazuje znacznie bardziej dramatyczny spadek efektywności w porównaniu do OpenMP:



Rysunek 3.1: Efektywność MPI

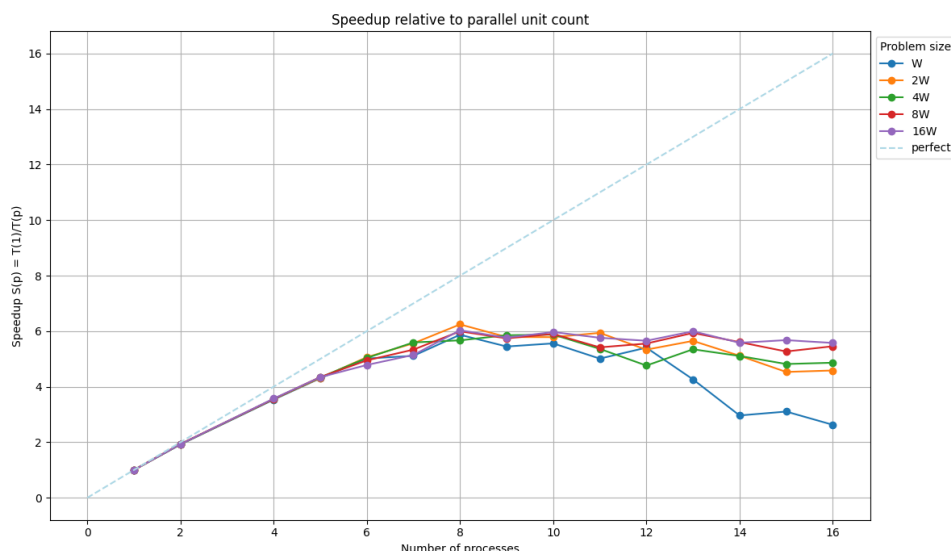
- Efektywność rozpoczyna się od wartości bliskiej 1.0 (100%) dla 1-2 procesów
- Następuje bardzo gwałtowny spadek efektywności już po przekroczeniu 4 procesów
- Dla 16 procesów efektywność spada dramatycznie do około 15-30
- Problem o rozmiarze W wykazuje najgorsze właściwości skalowania, osiągając zaledwie około 16% efektywności przy 16 procesach
- Większe problemy (16W) zachowują nieco lepszą efektywność, ale nadal znacznie gorszą niż OpenMP

Obserwowany drastyczny spadek efektywności wynika z wysokich kosztów komunikacji między procesami MPI oraz narzutu związanego z przesyłaniem danych przez sieć. W

przeciwieństwie do OpenMP, gdzie wątki dzielą wspólną pamięć, procesy MPI muszą jawnie komunikować się poprzez przesyłanie komunikatów.

3.2 Przyspieszenie obliczeń

Wykres przyspieszenia (Rysunek 3.2) ujawnia poważne ograniczenia skalowalności MPI:



Rysunek 3.2: Przyspieszenie MPI

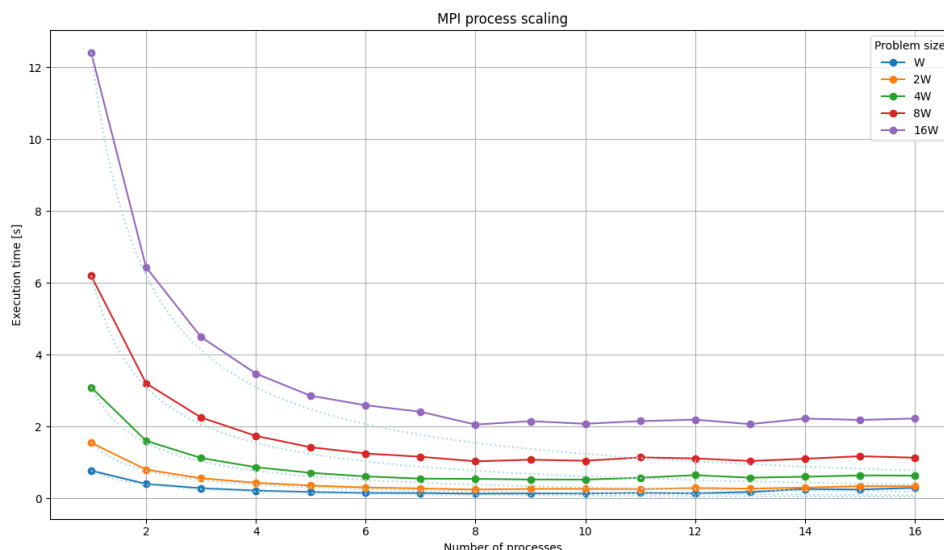
- Maksymalne osiągnięte przyspieszenie wynosi około 5.5-6x dla większych problemów (8W, 16W)
- Przyspieszenie osiąga maksimum już przy 6-8 procesach, po czym następuje stagnacja lub nawet spadek
- Problem o rozmiarze W wykazuje najgorsze przyspieszenie, osiągając maksimum około 2.5-3x
- Dalsze zwiększanie liczby procesów powyżej 8-10 skutkuje pogorszeniem wydajności
- Różnica względem idealnego przyspieszenia (perfect scaling) jest znacznie większa niż w przypadku OpenMP

Plateau przyspieszenia występuje znacznie wcześniej niż w OpenMP, co wskazuje na dominujący wpływ kosztów komunikacji. Po przekroczeniu optymalnej liczby procesów, koszty komunikacji przewyższają korzyści z równoległego przetwarzania.

3.3 Czas wykonania

Wykres czasu wykonania (Rysunek 3.3) pokazuje różnice w wydajności dla poszczególnych rozmiarów problemów:

- **Problem 16W:** Najdłuższy czas wykonania (około 12.3s dla 1 procesu), największy spadek czasu do około 2.2s przy optymalnej liczbie procesów



Rysunek 3.3: Czas wykonania MPI

- **Problem 8W:** Średni czas wykonania (około 6.1s dla 1 procesu), spadek do około 1.1s przy 6-8 procesach
- **Problem 4W:** Krótszy czas (około 3.1s dla 1 procesu), spadek do około 0.8s przy małej liczbie procesów
- **Problem 2W:** Szybkie wykonanie (około 1.6s dla 1 procesu), spadek do około 0.4s, ale ograniczone korzyści z większej liczby procesów
- **Problem W:** Najkrótszy czas (około 0.8s dla 1 procesu), minimalne korzyści z równoległości, czas stabilizuje się na poziomie około 0.3s

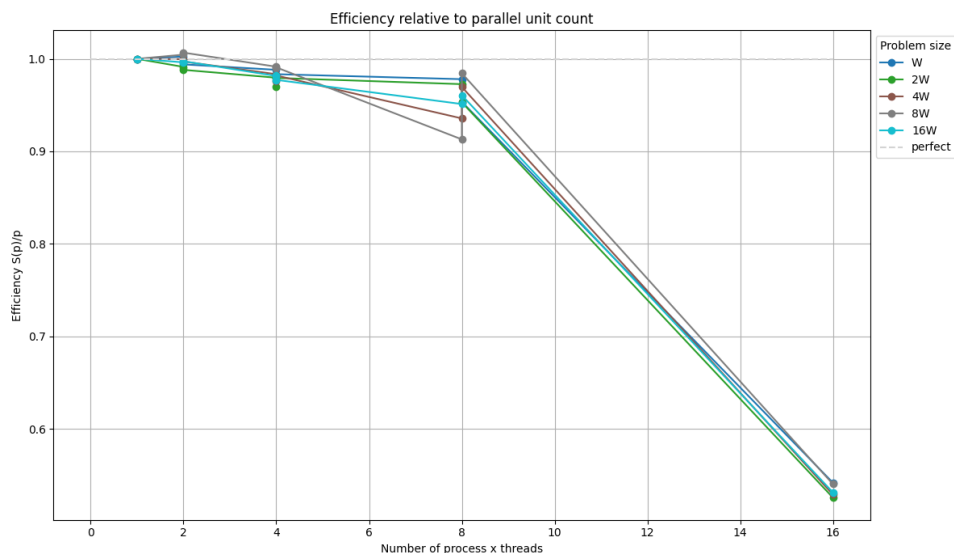
Charakterystyczną cechą MPI jest wyraźnie widoczne optimum w liczbie procesów dla każdego rozmiaru problemu. Po przekroczeniu tej optymalnej wartości, czas wykonania przestaje spadać lub nawet wzrasta, co jest efektem przewagi kosztów komunikacji nad korzyściami z równoległego przetwarzania.

Rozdział 4

Hybryda

4.1 Efektywność równoległego przetwarzania

Analiza wykresu efektywności (Rysunek 4.1) pokazuje znacznie lepsze właściwości skalowania modelu hybrydowego w porównaniu do czystego MPI:



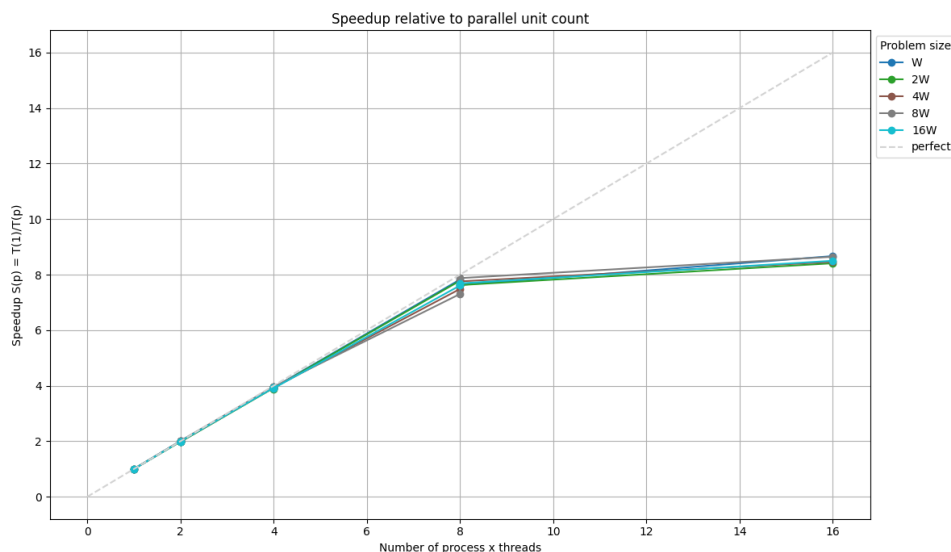
Rysunek 4.1: Efektywność Hybrid MPI+OpenMP

- Efektywność rozpoczyna się od wartości bliskiej 1.0 (100%) dla małej liczby jednostek równoległych
- Wszystkie rozmiary problemów wykazują bardzo podobne zachowanie skalowania
- Do 8 jednostek równoległych efektywność utrzymuje się na poziomie około 95-97%
- Znaczący spadek efektywności następuje dopiero po przekroczeniu 8 jednostek równoległych
- Dla 16 jednostek równoległych efektywność spada do około 53-54% dla wszystkich rozmiarów problemów
- Krzywe dla różnych rozmiarów problemów są praktycznie identyczne, co wskazuje na bardzo dobrą skalowalność niezależną od rozmiaru problemu

Model hybrydowy wykazuje znacznie lepszą efektywność niż czysty MPI dzięki kombinacji komunikacji między procesami (MPI) z równoległością wątków w obrębie procesu (OpenMP). Pozwala to na zmniejszenie liczby kosztownych komunikacji międzyprocesowych.

4.2 Przyspieszenie obliczeń

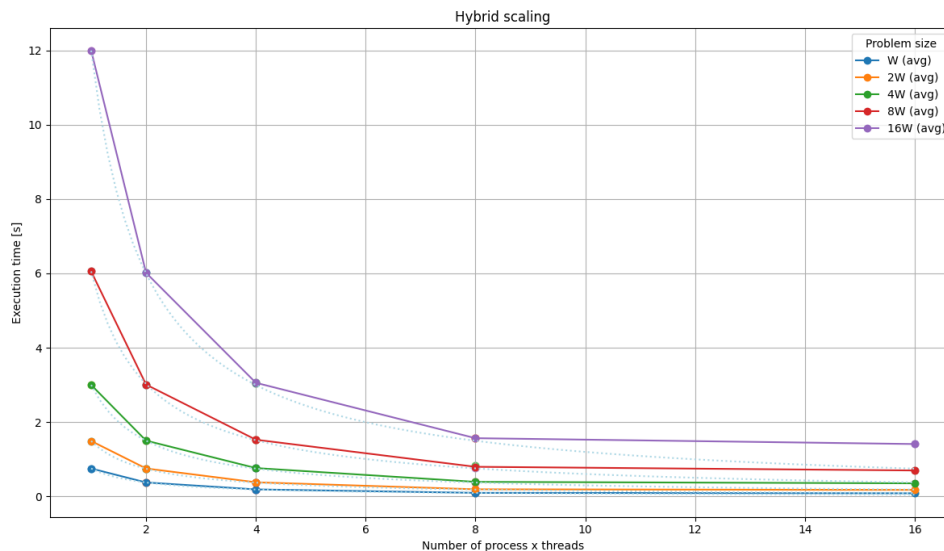
Wykres przyspieszenia (Rysunek 4.2) demonstruje doskonałe właściwości skalowania modelu hybrydowego:



Rysunek 4.2: Przyspieszenie Hybrid MPI+OpenMP

- Wszystkie rozmiary problemów wykazują niemal identyczne krzywe przyspieszenia
- Przyspieszenie rośnie liniowo do około 8 jednostek równoległych, osiągając wartość około 7.8x
- Dalszy wzrost jest wolniejszy, ale kontynuowany - dla 16 jednostek osiąga się około 8.5x przyspieszenia
- Brak wyraźnego plateau czy spadku wydajności, charakterystycznego dla czystego MPI
- Krzywa przyspieszenia jest znacznie bliższa idealnej linii skalowania niż w przypadku OpenMP czy MPI
- Niezależność od rozmiaru problemu wskazuje na doskonałą skalowalność algorytmu

Model hybrydowy osiąga najlepsze przyspieszenie spośród wszystkich testowanych podejść, łącząc zalety obu technologii i minimalizując ich wady.



Rysunek 4.3: Czas wykonania Hybrid MPI+OpenMP

4.3 Czas wykonania

Wykres czasu wykonania (Rysunek 4.3) potwierdza skuteczność modelu hybrydowego dla wszystkich rozmiarów problemów:

- **Problem 16W:** Najdłuższy czas wykonania (około 12.0s dla 1 jednostki), konsekwentny spadek do około 1.4s przy 16 jednostkach równoległych
- **Problem 8W:** Średni czas wykonania (około 6.0s dla 1 jednostki), spadek do około 0.7s przy maksymalnej równoległości
- **Problem 4W:** Krótszy czas (około 3.0s dla 1 jednostki), spadek do około 0.35s
- **Problem 2W:** Szybkie wykonanie (około 1.5s dla 1 jednostki), spadek do około 0.18s
- **Problem W:** Najkrótszy czas (około 0.75s dla 1 jednostki), spadek do około 0.09s przy pełnej równoległości

Charakterystyczną cechą modelu hybrydowego jest brak wyraźnego optimum - czas wykonania konsekwentnie spada wraz ze wzrostem liczby jednostek równoległych. Wszystkie rozmiary problemów wykazują podobne względne przyśpieszenie, co potwierdza doskonałą skalowalność tej architektury.

Linie reprezentujące czas wykonania mają gładki, monotoniczny charakter spadkowy, bez załamań czy plateau charakterystycznych dla czystego MPI, co wskazuje na optymalną kombinację obu technologii równoległego przetwarzania.

Rozdział 5

Porównanie

Bezpośrednie porównanie trzech technologii równoległego przetwarzania dla największego testowanego problemu (16W) ujawnia wyraźne różnice w charakterystykach wydajnościowych każdego podejścia.

5.1 Analiza efektywności

Wykres porównania efektywności (Rysunek 5.1) pokazuje hierarchię wydajności analizowanych technologii:



Rysunek 5.1: Porównanie efektywności dla problemu 16W

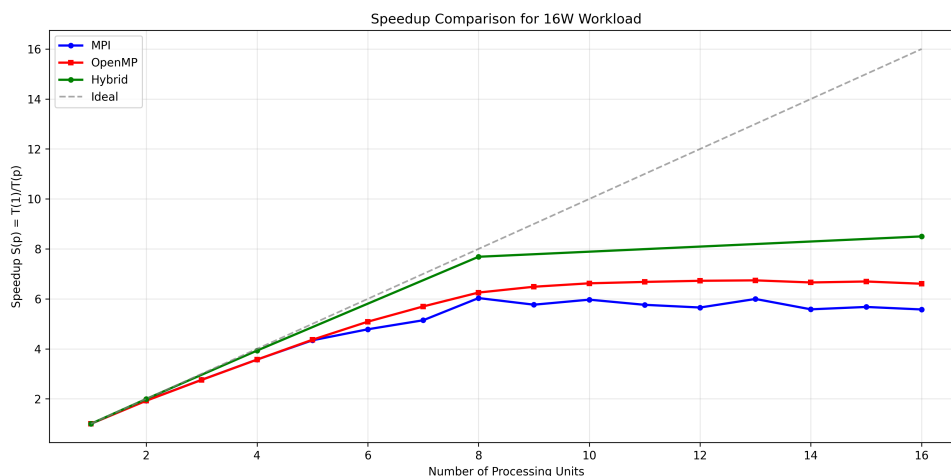
Model hybrydowy wykazuje zdecydowanie najlepszą efektywność w całym zakresie testowanych jednostek przetwarzania. Utrzymuje efektywność powyżej 95% do 8 jednostek, a nawet przy 16 jednostkach osiąga około 53% efektywności.

OpenMP zajmuje pozycję pośrednią, rozpoczynając od doskonałej efektywności, ale wykazując stopniowy spadek. Przy 8 jednostkach osiąga około 78% efektywności, a przy 16 jednostkach spada do około 41%.

MPI wykazuje najgorsze właściwości skalowania z dramatycznym spadkiem efektywności już po 4 procesach. Przy 8 procesach efektywność wynosi tylko około 75%, a przy 16 procesach spada do zaledwie 35%.

5.2 Analiza przyspieszenia

Porównanie przyspieszenia (Rysunek 5.2) potwierdza przewagę modelu hybrydowego:



Rysunek 5.2: Porównanie przyspieszenia dla problemu 16W

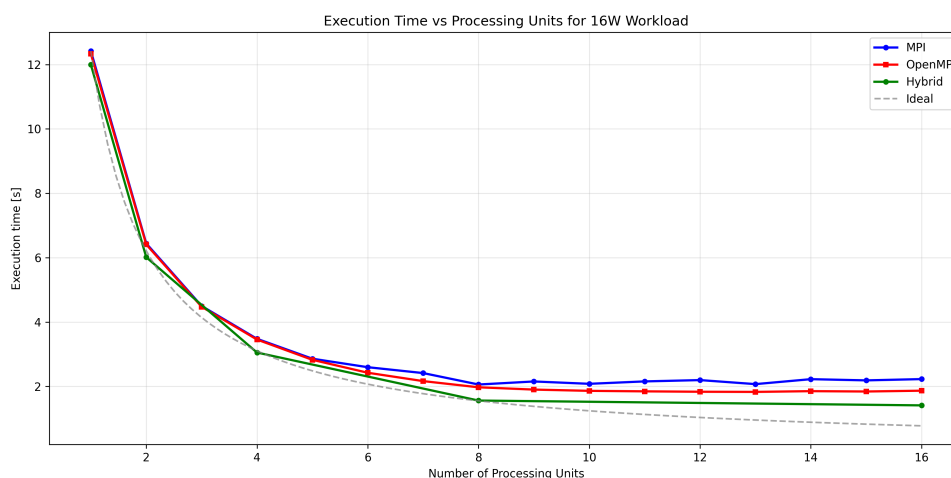
Model hybrydowy osiąga najlepsze wyniki z konsekwentnie rosnącym przyspieszeniem, które osiąga wartość około 8.5x przy 16 jednostkach. Krzywa wykazuje najbardziej liniowy charakter, zbliżony do idealnej skalowalności.

OpenMP osiąga maksymalne przyspieszenie około 6.5x przy 8-12 wątkach, po czym stabilizuje się na tym poziomie. Plateau wskazuje na osiągnięcie granicy wydajności architektury.

MPI wykazuje najgorsze przyspieszenie z wyraźnym maksimum około 6x przy 8 procesach, po czym następuje stagnacja lub nawet lekki spadek wydajności.

5.3 Analiza czasu wykonania

Wykres czasu wykonania (Rysunek 5.3) przedstawia praktyczne implikacje różnic w wydajności:



Rysunek 5.3: Porównanie czasu wykonania dla problemu 16W

Wszystkie technologie rozpoczynają od podobnego czasu wykonania sekwencyjnego (około 12.3s), ale różnią się znacząco w zakresie redukcji czasu:

Model hybrydowy osiąga najkrótszy czas wykonania przy maksymalnej równoległości (około 1.4s przy 16 jednostkach), wykazując konsekwentny spadek bez plateau.

OpenMP osiąga czas około 1.8s przy optymalnej konfiguracji (8-12 wątków), z niewielkimi korzyściami z dalszego zwiększania liczby wątków.

MPI stabilizuje się na poziomie około 2.2s przy 8-10 procesach, wykazując wyraźne optimum i brak korzyści z większej liczby procesów.

5.4 Wnioski

Analiza porównawcza jednoznacznie wskazuje na przewagę **modelu hybrydowego MPI+OpenMP**, który łączy zalety obu technologii:

- Najlepsza efektywność i skalowalność w całym zakresie jednostek przetwarzania
- Brak wyraźnego plateau wydajności charakterystycznego dla innych podejść
- Konsekwentne skracanie czasu wykonania wraz ze wzrostem równoległości

OpenMP stanowi dobre rozwiązanie dla systemów wielordzeniowych z współdzieloną pamięcią, oferując prostotę implementacji przy zadowalającej wydajności.

MPI wykazuje największe ograniczenia skalowalności ze względu na koszty komunikacji, ale pozostaje niezbędny dla systemów rozproszonych i klastrów obliczeniowych.