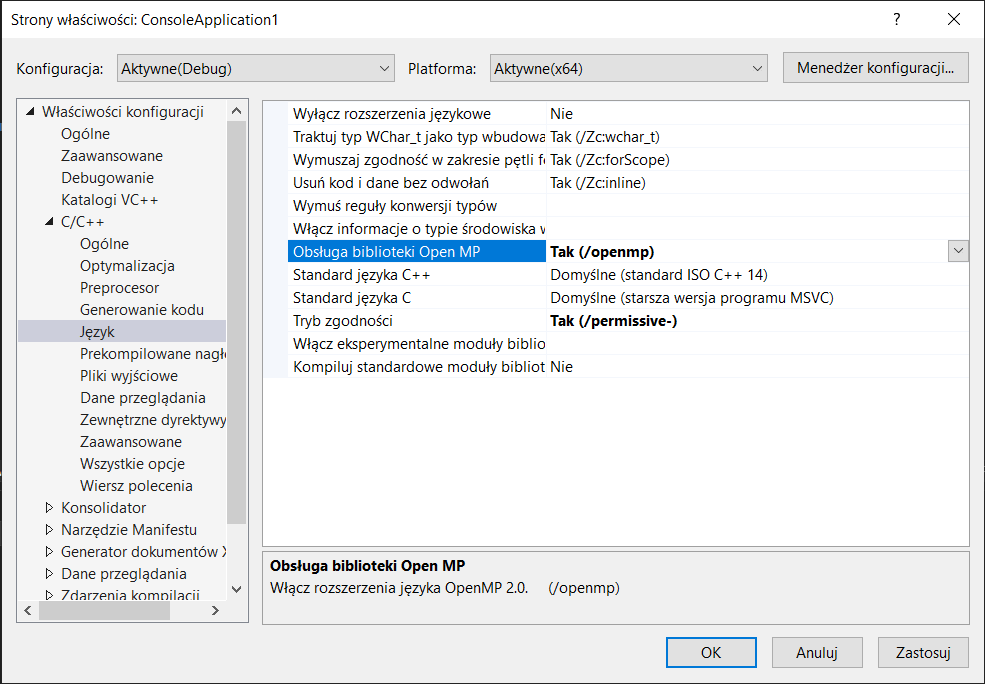
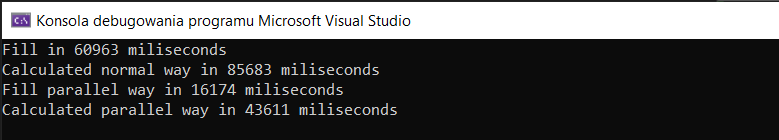
|  |  |
| --- | --- |
| Akademia Nauk Stosowanych w Nowym Sączu  Wydział Nauk Inżynieryjnych | |
| Imię i nazwisko: | Michał Bernardy |
| Grupa: | P1 |
| Ocena: |  |

**Wstęp**

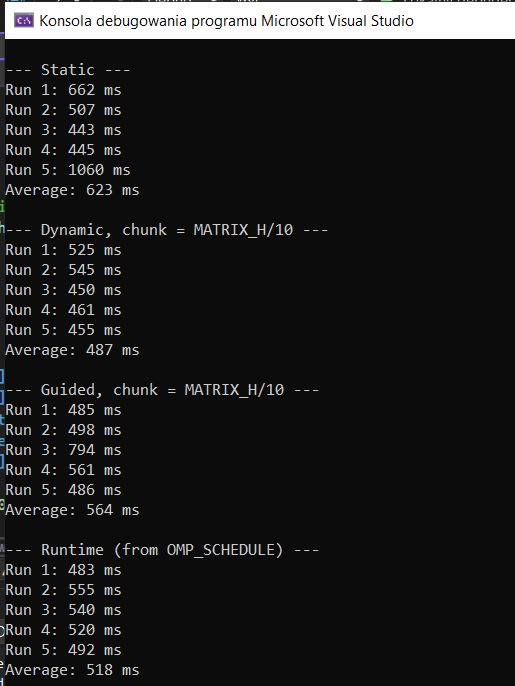
W testach sprawdzono wydajność operacji na dużych zbiorach danych z wykorzystaniem OpenMP, porównując czas wykonania obliczeń sekwencyjnych i równoległych. Zbadano również wpływ różnych strategii planowania zadań (static, dynamic, guided, runtime) oraz narzut związany z równoległym przetwarzaniem na wydajność obliczeń.****

**KOD 00**



Ten kod demonstruje mnożenie dużej macierzy przez wektor z wykorzystaniem OpenMP do równoległego przetwarzania danych. Najpierw dane są losowo generowane i wypełniane sekwencyjnie, a następnie wykonywane są obliczenia również sekwencyjnie i równolegle, co pozwala porównać wydajność obu podejść. Zmienna default(shared) w dyrektywie OpenMP oznacza, że wszystkie zmienne w danym bloku są współdzielone między wątkami, chyba że zadeklarowano inaczej. Alternatywnie, default(none) wymusza jawne określenie dla każdej zmiennej, czy jest współdzielona (shared) czy prywatna (private), np. default(none) shared(matrix, vector, result) private(i, k). Dzięki temu unika się przypadkowych błędów wynikających z nieokreślonego dostępu do danych. Na podstawie otrzymanych wyników widać wyraźnie, że zastosowanie przetwarzania równoległego znacząco przyspieszyło zarówno proces wypełniania danych (z 60963 ms do 16174 ms), jak i obliczeń (z 85683 ms do 43611 ms). Oznacza to skrócenie czasu wykonania odpowiednio o około 73% i 49%, co pokazuje skuteczność wykorzystania wielowątkowości w operacjach na dużych zbiorach danych.

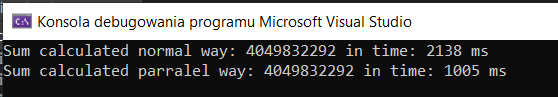
**KOD 01**



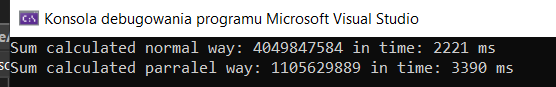
Kod testuje wydajność różnych strategii planowania równoległego w OpenMP (static, dynamic, guided, runtime) podczas mnożenia macierzy przez wektor. Wyniki pokazują, że średni czas wykonania dla strategii static wynosi 623 ms, dynamic 487 ms, guided 564 ms, a runtime 518 ms. Najszybsza okazała się strategia dynamic z podziałem na porcje (chunks) o rozmiarze MATRIX\_H/10, co sugeruje, że lepiej radzi sobie z nierównomiernym obciążeniem obliczeniowym. Strategia static była najwolniejsza, prawdopodobnie ze względu na sztywny podział pracy. Kod wykorzystuje OpenMP do równoległego przetwarzania i mierzy czas wykonania dla każdej strategii, przeprowadzając testy wielokrotnie dla wiarygodności wyników.

**KOD 02**

Wyniki bez zmian:



Wyniki po zmianach:

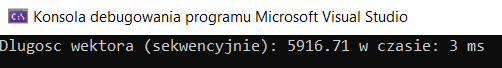


Wyniki pokazują, że po usunięciu dyrektywy reduction w wersji równoległej suma obliczona jest błędna (1105629889 zamiast ~4049832292), a czas wykonania wzrósł (3390 ms zamiast 1005 ms). Dyrektywa reduction zapewniała poprawne sumowanie wyników cząstkowych z różnych wątków, unikając wyścigów (race conditions). Bez niej, wiele wątków jednocześnie modyfikuje zmienną sum, co prowadzi do utraty danych i niepoprawnego wyniku. Wersja sekwencyjna działa poprawnie, ponieważ nie ma rywalizacji o dostęp do zmiennej. Wzrost czasu w wersji równoległej wynika z narzutu związanego z synchronizacją wątków i błędnymi operacjami.

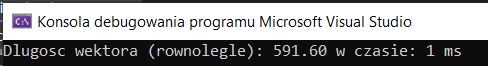
2. bez zrównoleglenia: #define VECTOR\_SIZE 10000



#define VECTOR\_SIZE 1000000



Ze zrównolegleniem: #define VECTOR\_SIZE 10000



#define VECTOR\_SIZE 1000000



Wyniki pokazują, że wersja sekwencyjna osiąga lepszą wydajność niż równoległa w obu testowanych przypadkach - zarówno dla wektora o rozmiarze 10 000 elementów, jak i 1 000 000 elementów. Dla mniejszego wektora różnica jest szczególnie widoczna - 0 ms w wersji sekwencyjnej versus 1 ms w równoległej. W przypadku większego wektora różnica wynosi 3 ms do 20 ms na korzyść rozwiązania sekwencyjnego. Różnice w obliczonych wartościach długości wektora (np. 589.60 vs 591.60) są minimalne i wynikają z losowego charakteru danych wejściowych, co potwierdza poprawność obu implementacji.

Gorsza wydajność wersji równoległej wynika głównie z narzutu związanego z zarządzaniem wątkami, który przewyższa korzyści z równoległego przetwarzania w przypadku tak prostych obliczeń. Operacje na pojedynczych bajtach są na tyle szybkie, że dodatkowe koszty organizacji przetwarzania równoległego nie są rekompensowane wzrostem wydajności. Ponadto, sekwencyjny dostęp do pamięci w tym przypadku lepiej wykorzystuje pamięć podręczną procesora.

Te wyniki pokazują ważną zasadę optymalizacji - równoległość nie zawsze oznacza przyspieszenie. W przypadku prostych operacji na małych lub średnich zbiorach danych, tradycyjne podejście sekwencyjne często okazuje się bardziej efektywne. OpenMP i inne techniki równoległe warto stosować dopiero przy bardziej złożonych obliczeniach lub naprawdę dużych zbiorach danych, gdzie korzyści z podziału pracy przewyższą narzut związany z organizacją przetwarzania równoległego.

**Wnioski**

Wyniki testów pokazują, że równoległość poprawia czas wykonania przy dużych zbiorach danych, jednak w przypadku prostych operacji na małych danych równoległe przetwarzanie może być mniej efektywne. Dla mniejszych wektorów wersja sekwencyjna była szybsza, z minimalnymi różnicami w obliczeniach. Równoległe przetwarzanie jest skuteczne w przypadku bardziej złożonych obliczeń lub dużych danych, gdzie zyski z równoległości przewyższają narzut związany z synchronizacją i zarządzaniem wątkami.