Przetwarzanie współbieżne – laboratorium 15.

Cel:

- Doskonalenie umiejętności analizy wydajności programów równoległych.

Uwaga: eksperymenty pomiaru czasu muszą być przeprowadzone zgodne z pewnymi elementarnymi zasadami:

- eliminuje się wpływ czynników nieistotnych (inne obliczenia, korzystanie z sieci przez inne osoby, fragmenty kodu nieistotne ze względu na cel pomiaru)
- odrzuca się wyniki znacznie odbiegające od pozostałych (należy przeprowadzić co najmniej kilka pomiarów)
- jako wynik przyjmuje się średnią z wyselekcjonowanych rezultatów lub rezultat minimalny
- **jako rozmiar zadania należy przyjmować wartości, które gwarantują odpowiednią dokładność pomiaru czasu wykonania – każdorazowo powinno to być ponad 0.001s (dotyczy to także wersji równoległych, co oznacza, że wersje sekwencyjne (na jednym rdzeniu powinny liczyć się w czasie ok. 0.1-1 s)**
- **wszelkie eksperymenty należy przeprowadzać z liczbami wątków/procesów mniejszych od liczby rdzeni (ewentualnie procesorów logicznych dla rdzeni z hyperthreadingiem). Intuicyjnie można oczekiwać, że przy rosnącej liczbie rdzeni czas obliczeń powinien się skracać (chyba że w programie są jakieś istotne problemy z efektywnym zrównolegleniem), dla korzystania z dwóch wątków na jednym rdzeniu z hyperthreadingiem skrócenie czasu zazwyczaj nie przekracza 30%. W przypadku przekroczenia przez liczbę wątków/procesów liczby rdzeni czas wykonania dla zdecydowanej większości programów rośnie. Opisy w sprawozdaniu nie powinny być sprzeczne z tymi intuicyjnie oczywistymi stwierdzeniami. Jeśli otrzymane wyniki są inne należy to dokładnie zweryfikować i podać jakie są tego przyczyny.**

Kroki:

1. Utworzenie katalogu roboczego (np. lab15) i podkatalogu (*np. calka*)
2. Skopiowanie ze strony przedmiotu pliku *MPI_calka.tgz*, kompilacja i uruchomienie (*make* – wykonuje jednocześnie uruchomienie)
3. Przeprowadzenie serii eksperymentów wydajnościowych dla programu obliczania całki:
    1. Uruchomienie kodu i uzyskanie wyników dla 1, 2, 4, 8, 16, ... procesów (maksymalna liczba procesów nie powinna przekraczać liczby rdzeni/wątków sprzętowych, patrz wyjaśnienie powyżej)
    2. Naniesienie wyników na wykresy zależności od liczby procesów dla:
        1. czasu wykonania
        2. przyspieszenia obliczeń
        3. efektywności zrównoleglenia

Jest wiele przyczyn, które mogą powodować, że przyspieszenie obliczeń odbiega od idealnego. Nawet dla tak prostego i z pozoru nie wymagającego dużego narzutu w postaci synchronizacji, komunikacji, dodatkowych operacji, programu jak obliczanie całki, mogą pojawić się niespodziewane przeszkody, jak np. zmiana częstotliwości procesora przy przejściu z obliczeń jednordzeniowych na wielordzeniowe

4. Przeprowadzenie testów skalowalności (w sensie słabym – stały rozmiar zadania dla każdego wątku):
    1. Uruchomienie kodu i uzyskanie wyników dla 1, 2, 4, 8, 16, ... procesów (maksymalna liczba procesów nie powinna przekraczać liczby rdzeni/wątków sprzętowych, patrz wyjaśnienie powyżej) oraz rozmiaru zadania (ilości operacji arytmetycznych – liczby iteracji w obliczeniach całki) proporcjonalniej do ilości procesów (rozmiar zadania także można dobierać do możliwości sprzętu, dostępnej pamięci itp., pamiętając by czasy wykonania nie schodziły nigdy poniżej 0.001s)
    2. Naniesienie wyników na wykresy zależności od liczby procesów (wątków) dla:
        1. czasu wykonania T(p*W_0, p) – czas rozwiązania zadania p-razy większego na p rdzeniach/procesorach (W_0 jest założonym rozmiarem początkowym zadania – należy tak go dobrać, żeby zadanie p-razy większe dało się efektywnie rozwiązać
5. Utworzenie podkatalogu (*np. mat_vec*)
6. Przeprowadzenie serii eksperymentów wydajnościowych dla programu obliczania iloczynu macierz-wektor w środowisku OpenMP (program własny z lab 9 lub rozpakowany i uruchomiony program z paczki *mat_vec_row_omp.tgz* ze strony przedmiotu)
    1. Uruchomienie kodu i uzyskanie wyników dla 1, 2, 4, 8, 16, ... wątków (maksymalna liczba wątków nie powinna przekraczać liczby rdzeni/wątków sprzętowych, patrz wyjaśnienie powyżej)
        1. Uzyskanie odpowiedniej liczby wątków można uzyskać przez ustawianie zmiennej systemowej OMP_NUM_THREADS – dzięki temu nie jest potrzebna rekompilacja kodu
        2. iloczyn obliczany jest dwukrotnie:
            1. w sposób maksymalnie zoptymalizowany w pliku mat_vec.c (kompilowanym z opcją -O3)

2. jako procedura sprawdzająca w pliku moj_program.c (kompilowanym z opcją -g)
3. program każdorazowo wyświetla i czas obliczeń zoptymalizowanych, i bez optymalizacji - obserwacja wyników pozwala stwierdzić jak skaluje się program silnie zoptymalizowany, a jak program mniej zoptymalizowany

2. Naniesienie wyników (dla obu wersji programu) na wykresy zależności od liczby wątków dla:
   1. czasu wykonania
   2. przyspieszenia obliczeń
   3. efektywności zrównoleglenia

   Czy wysokie wartości przyspieszenia obliczeń zawsze oznaczają optymalny czas wykonania równoległego? (odpowiedź w sprawozdaniu uzasadnij wynikami dla wersji zoptymalizowanej i nieoptymalizowanej)

   (W przypadku wersji zoptymalizowanej jej gorsza skalowalność wynika z tego, że szybciej wykorzystuje dostępną przepustowość magistrali i nie jest w stanie dalej skracać czasu wykonania ze względu na ograniczenia sprzętowe)

7. Przeprowadzenie testów skalowalności (w sensie słabym – stały rozmiar zadania dla każdego wątku):
   1. Uruchomienie kodu i uzyskanie wyników dla 1, 2, 4, 8, 16, ... wątków (maksymalna liczba wątków nie powinna przekraczać liczby rdzeni/wątków sprzętowych, patrz wyjaśnienie powyżej) oraz dla rozmiaru zadania (ilości operacji arytmetycznych) proporcjonalniej do ilości wątków (rozmiar zadania także można dobierać do możliwości sprzętu, dostępnej pamięci itp., pamiętając by czasy wykonania nie schodziły nigdy poniżej 0.001s)
      1. Uwaga: liczba operacji to ROZMIAR tablicy (WYMIAR to pierwiastek z ROZMIAR)!
      2. sekwencja powinna wynosić: WYMIAR, sqrt(2)*WYMIAR, 2*WYMIAR, sqrt(8)*WYMIAR, itd. (gdzie WYMIAR jest odpowiednio dobrany dla 1 wątku – czas wykonania ok. 1 sekunda, przy czym zadanie dla sqrt(p)*WYMIAR dla maksymalnej liczby wątków powinno także dać się efektywnie rozwiązać, co może wymagać redukcji wartości WYMIAR)
   2. Naniesienie wyników na wykresy zależności od liczby wątków dla:
      1. czasu wykonania T(p*W_0, p) – czas rozwiązania zadania p-razy większego na p rdzeniach/procesorach (W_0 jest założonym rozmiarem początkowym zadania dla wartości WYMIAR)

Dalsze kroki:

1. Obliczenie wartości i naniesienie na odpowiednie wykresy **przyspieszenia przeskalowanego (*scaled speed-up*)** dla obu programów (całka i iloczyn macierz-wektor)
   1. obliczanie przyspieszenia przeskalowanego i sprawdzenie jak bardzo odbiega od liniowego przyspieszenia idealnego jest alternatywną formą badania skalowalności w sensie słabym
   2. należy pamiętać, że wzór na przyspieszenie przeskalowane to S_s(p) = T(p*W_0, 1) / T(p*W_0, p), co oznacza dla każdej liczby rdzeni/procesorów dwa pomiary!
   3. wartości T(p*W_0, p) – czas rozwiązania zadania p-razy większego na p rdzeniach/procesorach należy wziąć z pp. 4 i 7, czasy rozwiązania zadania p-razy większego na 1 rdzeniu należy dodatkowo uzyskać (W_0 jest założonym rozmiarem początkowym zadania – należy tak go dobrać, żeby zadanie p-razy większe dało się efektywnie rozwiązać)
2. Obliczenie wydajności systemu w GFlops dla każdego z przypadków. Porównanie osiągniętej wydajności z maksymalną teoretyczną wydajnością systemu (czyli sumą wydajności teoretycznych wszystkich użytych rdzeni).
3. Obliczenie wydajności przesyłania danych pomiędzy pamięcią RAM i procesorem. Porównanie z maksymalną teoretyczną wydajnością używanego systemu.
4. Przeprowadzenie serii eksperymentów wydajnościowych (skalowalność w sensie silnym i słabym – tak jak dla tematów z części podstawowej) dla programu obliczania iloczynu macierz wektor zawierającego równoległość i na poziomie procesów (MPI), i na poziomie wątków (OpenMP)
   - program znajduje się w paczce `mat_vec_row_MPI_OpenMP.tgz`
   - liczbą procesów sterować można poprzez parametr *mpiexec (lub mpirun)*, liczbą wątków dla każdego procesu poprzez zmienną środowiskową OMP_NUM_THREADS
   - ciekawe wyniki uzyskuje się przeprowadzając eksperymenty na klastrze:
     - czysty MPI – kilka procesów na każdym węźle (OMP_NUM_THREADS=1)
     - MPI/OpenMP – np. jeden proces MPI z wieloma wątkami OpenMP na każdym węźle

Warunki zaliczenia:

1.Obecność na zajęciach i wykonanie kroków 1-7.
2.Oddanie sprawozdania z opisem zadania, kodem źródłowym programów, wynikami i wnioskami.