Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Laboratorium 13.

Cel:

- Doskonalenie umiejętności analizy wydajności programów równoległych.

Uwaga 1: **badanie wydajności programów ma sens tylko wtedy, kiedy sprzęt i konfiguracja oprogramowania pozwalają na uzyskiwanie skrócenia czasu wykonania**. Do badania przyspieszenia obliczeń maszyna powinna mieć co najmniej 4 rdzenie fizyczne. Pierwszy badany program (*calka_omp.c*) powinien uzyskiwać z dużym przybliżeniem tylokrotne skrócenie czasu w stosunku do wersji sekwencyjnej ile jest wątków (rdzeni) użytych do wykonania. Podane warunki powinien spełniać serwer Honorata - w przypadku nie dysponowania odpowiednią własną maszyną, obliczenia powinno się zrealizować na serwerze.

(*uwaga do uwagi: w przypadku każdego z programów może się zdarzyć, że skrócenie czasu nie jest proporcjonalne do liczby rdzeni. Dzieje się tak z powodu rozmaitych narzutów obliczeń równoległych. Narzutami mogą być czynniki związane z programami: występowanie fragmentów niedających się zrównoleglić, brak zrównoważenia obciążenia rdzeni, dodatkowe obliczenia w wersji równoległej, częste odwołania do czasochłonnych procedur systemowych (np. komunikacji, synchronizacji, alokacji pamięci itp.). Na nieoptymalny czas wykonania mogą także wpływać czynniki zewnętrzne (np. korzystanie z komputera przez innych użytkowników, inne procesy zużywające zasoby maszyny, zmiana częstotliwości taktowania procesora w zależności od liczby aktualnie pracujących rdzeni - patrz uwagi w zadaniach dodatkowych). Na serwerze Honorata w trakcie zajęć, kiedy korzysta z niego wielu użytkowników, warunki pomiaru czasu mogą być dalekie od optymalnych. Powoduje to, że w celu uzyskania wiarygodnych parametrów dla programu (po eliminacji zaburzeń zewnętrznych) może być wymagane co najmniej kilkukrotne uruchomienie programu, a nawet uruchomienie go po zajęciach, przy niskim obciążeniu maszyny (sprawdzanym np. poprzez uruchomienie top lub htop))*

Uwaga 2: eksperymenty pomiaru czasu muszą być przeprowadzone zgodnie 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 czas wykonania powinien przekraczać 0.001s** (dotyczy to także wersji równoległych, co oznacza, że wersje sekwencyjne (na jednym rdzeniu) do tworzenia wykresów przyspieszenia obliczeń i skalowalności powinny liczyć się w czasie ok. 0.1-1 s)
- **wszelkie eksperymenty należy przeprowadzać z liczbami wątków/procesów nie większych 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 (procesorów logicznych) 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.

Uwaga 3: 2022 - w przypadku kiedy zajęcia odbywają się przed właściwym wykładem omawiającym analizę wydajności obliczeń równoległych (z powodu kuriozalnych przesunięć w terminach zajęć wtorkowych i czwartkowych) należy przy realizacji ćwiczeń wspomagać się informacjami na slajdach do wykładu (**W13, 20.01.2022: Wydajnosc**) . Przy tworzeniu tabel i wykresów pomocne są slajdy 6 i 8. Odpowiedzi na pytania zamieszczone poniżej w tematach kolejnych zadań najlepiej udzielać po wysłuchaniu wykładu (na żywo lub z nagrania).

Kroki:

1. Utworzenie katalogu roboczego (np. lab13) i podkatalogu (*np. calka*)
2. Skopiowanie ze strony przedmiotu pliku *calka_omp.c,* kompilacja (np. *gcc -O3 -fopenmp calka_omp.c*) i uruchomienie
3. Przeprowadzenie serii eksperymentów wydajnościowych dla programu obliczania całki:
   1. Uruchomienie kodu i uzyskanie wyników dla 1, 2, 4, 8 (dla serwera Honorata także 16) wątków (maksymalna liczba wątków nie powinna przekraczać liczby rdzeni/wątków sprzętowych, patrz wyjaśnienie powyżej)
      - liczbę wątków najlepiej ustalać przez nadanie odpowiedniej wartości zmiennej środowiskowej, np. *export OMP_NUM_THREADS=2*
   2. Wpisanie wyników do tabeli i naniesienie na wykresy zależności od liczby wątków p dla:
      - czasu wykonania
      - przyspieszenia obliczeń
      - efektywności zrównoleglenia

   W sprawozdaniu można umieścić analizę wyników - czy uzyskana skalowalność w sensie silnym jest dobra czy nie, jakie są tego przyczyny? Badając kod źródłowy można ustalić czy są jakieś potencjalne źródła narzutu spowalniającego obliczenia równoległe. Czy program w testowanych konfiguracjach wyczerpywał dla pewnej liczby wątków zasoby komputera (co uniemożliwiałoby dalszy wzrost wydajności)?
4. Utworzenie podkatalogu (*np. mat_vec*), skopiowanie paczki ***mat_vec_row_MPI_perf.tgz***, rozpakowanie, uruchomienie kodu *(przez make*, a potem *make run*)
5. Przeprowadzenie serii eksperymentów wydajnościowych dla programu obliczania iloczynu macierz-wektor w środowisku MPI
   1. Uruchomienie kodu (przez *mpiexec*) i uzyskanie wyników dla 1, 2, 4, 8 (dla serwera Honorata także 16) procesów (maksymalna liczba procesów nie powinna przekraczać liczby rdzeni/wątków sprzętowych, patrz wyjaśnienie powyżej)
      - Zmianę liczby procesów należy realizować w typowy dla MPI sposób poprzez opcję *-np* dla *mpiexec*
   2. Wpisanie wyników do tabeli i naniesienie na wykresy zależności od liczby procesów p dla:
      - czasu wykonania
      - przyspieszenia obliczeń
      - efektywności zrównoleglenia

   W sprawozdaniu można umieścić analizę wyników - czy uzyskana skalowalność w sensie silnym jest dobra czy nie, jakie są tego przyczyny? Badając kod źródłowy można ustalić czy są jakieś potencjalne źródła narzutu spowalniającego obliczenia równoległe. Czy program w testowanych konfiguracjach wyczerpywał dla pewnej liczby wątków zasoby komputera (co uniemożliwiałoby dalszy wzrost wydajności)?

---------- 3.0 ----------

5. Przeprowadzenie testów skalowalności (w sensie słabym – stały rozmiar zadania dla każdego wątku) dla mnożenia macierz-wektor:
   1. Uruchomienie kodu (przez *make*, a potem *make run*) i uzyskanie wyników dla 1, 2, 4, 8 wątków (maksymalna liczba wątków nie powinna przekraczać liczby rdzeni/wątków sprzętowych, patrz wyjaśnienie powyżej) oraz dla rozmiaru zadania (ilości operacji arytmetycznych) proporcjonalniej do liczby wątków (rozmiar zadania także można dobierać do możliwości sprzętu, dostępnej pamięci itp., pamiętając by czasy wykonania nie schodziły nigdy poniżej 0.001s)
      - Uwaga: liczba operacji to ROZMIAR tablicy (WYMIAR to pierwiastek z ROZMIAR)!
      - 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)
        - w pliku *moj_program.c* znajduje się przykładowa sekwencja wymiarów tablicy odpowiednia do badania skalowalności w sensie słabym

   uwaga*: zmiana parametru WYMIAR na maszynach w laboratorium z przesuniętym czasem w stosunku do serwera plików może wymagać wywołania* make clean *przed rekompilacją kodu!*

2. Wpisanie wyników do tabeli (wykład 13, slajd 8 - wystarczy wypełnić pierwszy wiersz i przekątną główną) i naniesienie na wykresy zależności od liczby procesów p 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)
    2. **przyspieszenia przeskalowanego (*scaled speed-up*),** S_s(p) = T(p*W_0, 1) / T(p*W_0, p), (oznacza to dla każdej liczby rdzeni/procesorów dwa pomiary czasu z tabeli!)
6. Przeprowadzenie badania skalowalności w sensie silnym (zgodnie z punktem 5, dla wymiaru macierzy 12144), ale dla programu niezoptymalizowanego
    1. dokonanie zmiany poziomu optymalizacji z -O3 na -O0
    2. rekompilacja kodu (*make clean; make*)
    3. przeprowadzenie pomiarów czasu, wypełnienie tabel, utworzenie wykresów
7. Porównanie wyników dla wersji zoptymalizowanej i niezoptymalizowanej
    - dla wielu maszyn i programów pojawić się może częste zjawisko lepszej skalowalności programu niezoptymalizowanego (gorzej napisany program, mniej optymalnie skompilowany nadrabia braki przez wykorzystanie rosnącej liczby wątków/procesów)
    - mimo tego wersja niezoptymalizowana może mieć dla każdej liczby wątków/procesów czasy wykonania znacznie gorsze od wersji zoptymalizowanej

   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 iloczynu macierz-wektor jej gorsza skalowalność po przekroczeniu pewnej liczby procesów wynika z tego, że szybciej wykorzystuje dostępną przepustowość magistrali i nie jest w stanie dalej skracać czasu wykonania ze względu na ograniczenia sprzętowe)

---------- 4.0 ----------

Dalsze kroki:

1. Obliczenie wydajności systemu w GFlops dla każdego z przypadków (w tym celu należy dla każdego algorytmu obliczyć liczbę wykonywanych operacji zmiennoprzecinkowych i podzielić ją przez czas wykonania kodu). Porównanie osiągniętej wydajności z maksymalną teoretyczną wydajnością systemu (czyli sumą wydajności teoretycznych wszystkich użytych rdzeni).
    - aspektem, który powinien zostać uwzględniony przy dokładnych pomiarach wydajności jest występowanie w większości współczesnych mikroprocesorów wielordzeniowych mechanizmu przetaktowywania – dostosowania częstotliwości pracy do aktualnych warunków. W szczególności **przetaktowywanie może oznaczać wyższą częstotliwość pracy, kiedy używany jest tylko jeden rdzeń oraz niższą kiedy wykorzystywane jest wiele rdzeni**. Występowanie przetaktowywania zaburza obraz skalowalności algorytmów, do już istniejących narzutów wykonania równoległego dodaje zmniejszenie częstotliwości pracy rdzeni. Jedną z możliwości neutralizacji przetaktowywania jest wyłączenie go (o ile istnieje taka możliwość – najczęściej wymaga uprawnień administratora). Jeśli nie daje się wyłączyć przetaktowania można próbować uwzględnić je w analizie. Za pomocą odpowiednich narzędzi (np. poleceniem:
      $ *watch -n 1 cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq*)
      można ustalić częstotliwość pracy dla danej liczby procesorów i przeskalować czas wykonania zakładając częstotliwość bazową. Analiza przyspieszenia obliczeń może uwzględniać oba przypadki – pierwszy kiedy włącza się narzut wynikający z przetaktowania i stosuje po prostu zmierzony czas, drugi kiedy niweluje się efekt przeskalowania i stosuje odpowiednio przeskalowany czas wykonania
        - dla serwera Honorata częstotliwość bazowa to 2.2 GHz. W trakcie wykonania programu z p. 2 laboratorium na 1 rdzeniu, częstotliwość rdzenia osiąga 3 GHz. Dla wykonania w zmierzonym czasie t_boost – przeskalowany czas dla częstotliwości bazowej wynosi t_base = (3/2.2)*t_boost.
        - uwzględnianie przeskalowania jest oczywiście zadaniem dodatkowym, jednak wiedza o przetaktowywaniu jest istotna dla wyciągnięcia właściwych wniosków z laboratorium

2. Obliczenie wydajności przesyłania danych pomiędzy pamięcią RAM i procesorem dla każdego z badanych programów(w tym celu należy dla każdego algorytmu obliczyć ilość bajtów pobieranych z pamięci i zapisywanych do pamięci, a nastepnie podzielić ją przez czas wykonania kodu). Porównanie z maksymalną teoretyczną wydajnością używanego systemu.
3. 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 z paczki *mat_vec_row_MPI_OpenMP.tgz* należy zmodyfikować odkomentowując dyrektywy OpenMP
    - 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 1-5.
2.Oddanie sprawozdania z opisem zadania, wynikami, tabelami, wykresami i wnioskami – zgodnie z regulaminem laboratoriów.