Analiza i modelowanie wydajności obliczeń

Lab 7_x. Fałszywe współdzielenie i rozpychanie tablic

Cel: Zbadanie zjawiska "fałszywego współdzielenia" (*false sharing*) przy współbieżnym wielowątkowym dostępie do tablic oraz techniki "rozpychania tablic" (*array padding*) mającej na celu unikanie opóźnień przy dostępie do tablic (w szczególności dwu i więcej wymiarowych)

Kroki realizowane w ramach laboratorium

1. **Zadanie 1. Badanie zjawiska fałszywego współdzielenia (*false sharing)***
    1. Pobierz ze strony przedmiotu paczkę *false_sharing.tgz* i rozpakuj w katalogu roboczym (np. *lab_07_x*)
    2. Przejdź do katalogu *false_sharing.* W pliku *false_sharing_test_papi.c* znajduje się pętla, mająca sekwencyjną postać:

```
for(i=0;i<ARRAY_SIZE;i++) {
  suma += j*c[i]*c[i];
}
```

Po zrównolegleniu pętli za pomocą wątków *pthreads* i zastosowaniu dekompozycji blokowej (każdy wątek o identyfikatorze **thread_id** wykonuje **ARRAY_SIZE/NO_TH** iteracji, gdzie **NO_TH** jest liczbą wątków) kod przybiera postać:

```
for(i=thread_id*ARRAY_SIZE/NO_TH;i<(thread_id+1)*ARRAY_SIZE/NO_TH;i++) {
     suma += j*c[i]*c[i];
}
```

W pętli dokonywane są obliczenia mające charakter redukcji – wszystkie wątki mają wyprodukować pojedynczą liczbę. Pozostawienie kodu w powyższej formie prowadzi do wyścigu (*data race*) i niedeterministycznego działania (zmienna **suma** musi być zmienna wspólną wątków).

[W ramach specyfikacji *pthreads* nie ma standardowych wzorców przeprowadzania redukcji, jak np. w OpenMP gdzie dzięki klauzuli *reduction* można skorzystać z wzorca, w którym wątki przeprowadzają obliczenia na bezpiecznych zmiennych lokalnych, które **następnie, po zakończeniu pętli, w sekcji krytycznej** sumowane są do pojedynczej zmiennej wspólnej.]

Dla wątków *pthreads* można zrealizować taki schemat korzystając z *mutexów*. Alternatywą dla takiego podejścia może być użycie globalnej tablicy, do której wątki wpisują własne wyniki, przy czym każdy wątek ma sobie przypisany element tablicy dzięki czemu nie dochodzi do wyścigu :

```
for(i=thread_id*ARRAY_SIZE/NO_TH;i<(thread_id+1)*ARRAY_SIZE/NO_TH;i++) {
     a[STRIDE*thread_id] += j*c[i]*c[i];
}
```

(parametr **STRIDE** w kodzie jest wykorzystywany później, w początkowej fazie eksperymentów wartość **STRIDE** powinna pozostać domyślnie jako 1),

Na zakończenie ostateczny wynik można uzyskać np. sumując wyrazy tablicy **a** w wątku głównym .

[obliczenia w powyższej pętli powtarzane są w kodzie wielokrotnie dla uzyskania odpowiednio długiego czasu wykonania, umożliwiającego dokładne pomiary czasu i obserwacje wpływu zjawiska *false sharing* na wydajność - w celu uniknięcia eliminacji powtórzeń pętli przez kompilator przeprowadzający optymalizację *dead code removal* , w pętli znajduje się jeszcze jedna linijka z wywołaniem funkcji *funkcja_fake*, która jednak, na skutek odpowiedniego doboru wartości w tablicy *c*, nigdy nie jest wykonywana]

Rozwiązanie powyższe, mimo że poprawne (prowadzące zawsze do właściwego wyniku), może grozić obniżeniem wydajności z powodu zjawiska fałszywego współdzielenia (*false sharing*). Wprawdzie wątki posiadają własne wyrazy tablicy do sumowania lokalnie obliczonych wartości, ale wyrazy dla różnych wątków mogą znajdować się w tej samej linii pamięci podręcznej. Prowadzi to, przy modyfikacji wartości w tablicy, do nieustannego wzajemnego unieważniania zawartości linii pamięci podręcznej i konieczności ciągłego przeładowywania zawartości pamięci.

3. Przeprowadź obliczenia dla dostarczonego kodu, starając się precyzyjnie wykryć zjawisko *false sharing.* W tym celu dokonaj kompilacji kodu dla dwóch wątków (jest to domyślne ustawienie w pliku źródłowym *false_sharing_test_papi.c*).

   [Ustaw lub tylko sprawdź poprawną lokalizację biblioteki PAPI w pliku Makefile (symbol PAPI_HOME), a następnie utwórz wymaganą bibliotekę *libpapi_driver_mth.a* (poleceniem ***make recreate_papi_lib_mth***). Wywołanie *make* powoduje kompilację i uruchomienie kodu.]

   1. Do potwierdzenia zjawiska *false sharing* kod wykorzystuje liczniki sprzętowe z przypisanymi zdarzeniami:
      - L2_LINES_IN.ALL (L2_LINES) - liczba podmian linii pamięci podręcznej L2
      - L2_TRANS.RFO (*Read For Ownership*) (L2_RFO)– zdarzenie odnoszące się do chęci zapisu do linii w pamięci L2 o statusie *invalid,* co wymaga najpierw pobrania linii (jako realizacja strategii *write allocate*)
      - MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_HITM (L3_MISS) - odnosi się do sytuacji chybienia w lokalną dla rdzenia pamięć L3 i odnalezienia danych w pamięci innego mikroprocesora

      największe znaczenie dla zjawiska fałszywego współdzielenia ma zdarzenie L2_RFO
      - zaobserwuj dużą liczbę wystąpienia zjawiska przy wykonaniu kodu

4. Wyniki pomiarów umieść w pierwszej linii tabelki (jako czas wykonania i wydajność zapisz dane zwracane przez wersje bez wykorzystania liczników sprzętowych *false_sharing_test.exe* , uruchamianą przez *make* bezpośrednio po wersji z licznikami *false_sharing_test_papi.exe*):

STRIDE	Czas wykonania [s]	Wydajność [GB/s]	L2_LINES	L2_RFO	L3_MISS
1					

5. W celu eliminacji zjawiska *false sharing* można w prosty sposób zmodyfikować kod zmieniając wartość parametru **`STRIDE`** . W kolejnych linijkach tabeli należy wpisać wyniki eksperymentu dla rosnących wartości **`STRIDE`** (np. 1, 2, 4, 8). Dla której wartości zdarzenie L2_RFO praktycznie przestaje występować (jego proporcja do L2_LINES staje się pomijalnie mała)?

   [Jeśli wartość parametru **`STRIDE`** jest równa 1 – różne wątki korzystają z tej samej linii pamięci podręcznej (dzięki wyrównaniu na granicy 64-bajtowej przy alokacji tablicy za pomocą *posix_memalign*), dla odpowiednio dużej wartości **`STRIDE`** – różne wątki wpisują dane do elementów znajdujących w różnych liniach pamięci podręcznej.]

   [W ostatecznej wersji wystarczy podać tabelę dla pierwszej wartości **`STRIDE`** dla której znika fałszywe współdzielenie – jeśli wartość ta została uzyskana nie eksperymentalnie, ale na podstawie analizy zapisów do tablicy **`a`** i rozmiaru linii pamięci podręcznej, analiza taka powinna znaleźć się w sprawozdaniu]

6. Wyciągnij wnioski z eksperymentów i umieść je w sprawozdaniu: jak często podmieniane są linie pamięci podręcznej dla różnych wartości **`STRIDE`**?, jak często występuje zdarzenie *read for ownership*, wynikające z zastosowanego protokołu zgodności pamięci podręcznej?, jak często raportowane jest zjawisko konieczności pobrania danych z pamięci innego mikroprocesora?

2. Zadanie 2. Badanie wpływu zjawiska fałszywego współdzielenia na wydajność obliczeń

1. Uruchom program z zadania 1 w wersji bez zliczania zdarzeń sprzętowych dla co najmniej 8 wątków i wartości **`STRIDE=1`** . Zanotuj uzyskane wyniki wydajnościowe (czas obliczeń, efektywna przepustowość pamięci).

   [dla uzyskania lepszej powtarzalności wyników wydajnościowych można jak zwykle uruchomić kod pod kontrolą *numactl*]

2. Powtórz obliczenia dla wartości **`STRIDE`** dla której nie występuje *false sharing.* Jaki ilościowy wpływ na wydajność ma występowanie lub nie fałszywego współdzielenia?

3. Zadanie 3. Badanie wpływu sposobu przechowywania tablic dwuwymiarowych (macierzy) na wydajność.

1. Pobierz ze strony przedmiotu paczkę *array_padding.tgz* , rozpakuj w katalogu roboczym
   - standardowo utwórz bibliotekę *papi_driver_mth*
2. Skompiluj prosty program obliczania sumy wyrazów w tablicy dwuwymiarowej (macierzy)
   - program oblicza sumę wyrazów dla sekwencji macierzy o wybranych rozmiarach
   - z założenia macierz jest przechowywana wierszami w tablicy jednowymiarowej (wyraz a_{ij} w *i*-tym wierszu i *j*-tej kolumnie znajduje się w położeniu a[i*WYMIAR+j], gdzie WYMIAR jest długością wiersza)
   - obliczenie przeprowadzone jest dwukrotnie, za każdym razem w podwójnej pętli po wierszach i kolumnach, raz z pętlą po wierszach jako pętlą zewnętrzną, a raz z pętlą po kolumnach jako zewnętrzną
3. Na podstawie wyników skonstruuj wykres zależności wydajności dostępów do pamięci w zależności od rozmiaru macierzy i sposobu dostępu (wykres może być wykresem słupkowym, po dwa słupki na jeden rozmiar macierzy)

   [jako dane do wykresu można wziąć przepustowość pamięci w GB/s, czas dostępu do pojedynczego elementu tablicy w ns, ewentualnie, w celu uniknięcia zaburzeń wprowadzanych przez zmienną częstotliwość pracy procesora, przepustowość w B/takt lub czas dostępu, ale wyrażany w taktach]

   [wydruki generowane przez program z paczki zawierają: rozmiar tablicy, czas sumowania, częstotliwość pracy rdzenia, czas dostępu do elementu tablicy w ns, przepustowość w liczbie dostępów na takt]

   [kod wydruków innych wybranych wielkości należy stworzyć samodzielnie na podstawie rozmiaru tablicy (zmiennych double), czasu wykonania sumowania oraz częstotliwości obliczanej na podstawie liczników taktów PAPI]

4. Zanalizuj wzorzec dostępu do elementów macierzy dla każdego wariantu przeglądania i jego wpływ na wydajność
5. Wyciągnij wnioski co do powyższego wpływu, uwzględniając dodatkowo znaczenie specjalnych rozmiarów tablicy

---------- *3.0* ----------

4. Zadanie 4. Zastosowanie techniki rozpychania tablic (*array_padding*) w celu uniknięcia niekorzystnego wpływu specjalnych rozmiarów macierzy na wydajność przeglądania

1. W pliku *array_padding_test_papi.c* odkomentuj linię modyfikującą oryginalny rozmiar macierzy (parametr WYMIAR, jako długość wiersza), kiedy jest on podzielny przez 8

2. Modyfikacja kodu polega na alokacji tablicy o wydłużonym o 1 wierszu. Dodatkowy element w wierszu może być zainicjowany dowolną wartością, nie uczestniczy on w obliczeniach, jego istnienie służy tylko zmianie wzorca dostępu do pamięci przy przeglądaniu tablicy (w szczególności kolumnami)

   [więcej o technice *array padding* w skrypcie na stronach 10-13]

3. Skompiluj i ruchom zmodyfikowany kod. Zaobserwuj jak zmienia się, w stosunku do przypadku bez rozpychania tablic, wydajność dostępów do pamięci przy przeglądaniu macierzy wierszami i kolumnami (rozpychanie jest zastosowane dla wymiarów podzielnych przez 8).
4. Stwórz wykresy podobne jak w p. 3.3, umieszczając wielkości najlepiej ilustrujące wpływ techniki *array padding* na wydajność.

---------- *3.5* ----------

Dalsze kroki:

1. Uruchom obliczenia programu badającego zjawisko *false sharing* z dwoma wątkami i wartością parametru **`STRIDE=1`**, przypisując wątki do różnych rdzeni procesora. Szczególnie ciekawe mogą być przypadki, kiedy wątki trafią do pojedynczego rdzenia wykorzystującego SMT (Honorata – np. *numactl -C 0,24*), do różnych rdzeni tego samego mikroprocesora (gniazda) (Honorata – np. *numactl -C 0,1*), do różnych rdzeni różnych mikroprocesorów (gniazd) (Honorata – np. *numactl -C 0,12*) – to ostatnie oczywiście tylko dla maszyn wieloprocesorowych. Uzupełnij tabelkę o wyniki dla zbadanych przypadków. Zaobserwuj różnice między nimi w czasie wykonania i w wartościach liczników sprzętowych. Wyciągnij wnioski.

STRIDE=1 numactl -C	Czas wykonania [s]	Wydajność [GB/s]	L2_LINES	L2_RFO	L3_MISS
0,1					
0,12					
0,24					
...					

---------- *4.0* ----------

2. Przeprowadź badania takie jak w p. 3 i 4, tym razem dla macierzy przechowywanych kolumnami (wyraz a_{ij} w *i*-tym wierszu i *j*-tej kolumnie macierzy znajduje się w położeniu a[i+j*WYMIAR], gdzie WYMIAR jest tym razem długością (wysokością) kolumny)
   - dodatkowym celem ćwiczenia jest nabycie wprawy w manipulowaniu sposobem przechowywania i organizacją obliczeń dla macierzy przechowywanych w tablicach jednowymiarowych

Warunki zaliczenia:

1. Obecność na zajęciach i wykonanie kroków obowiązkowych
2. Oddanie sprawozdanie o formie i treści zgodnej z regulaminem laboratoriów