Analiza i modelowanie wydajności obliczeń

Lab 4. Parametry pamięci podręcznej

Wstęp:

1. Celem laboratorium jest przeprowadzenie szeregu eksperymentów obliczeniowych, z wykorzystaniem odpowiednio skonstruowanych mikro-benchmarków, których celem jest określenie podstawowych parametrów pamięci podręcznych różnych poziomów, takich jak rozmiar pamięci, rozmiar pojedynczej linii pamięci czy drożność pamięci. Eksperymenty, szczegółowo opisane przy realizacji kolejnych kroków laboratorium, opierają się na znajomości zasad funkcjonowania sekcyjno-skojarzeniowych (*set associative*) pamięci podręcznych, omawianych na wykładzie.

Kroki realizowane w ramach laboratorium:

1. **Zadanie 1 (obowiązkowe). Badanie rozmiaru pamięci podręcznej**
2. Pobierz ze strony przedmiotu paczkę *cache_discovery.tgz*, rozpakuj w nowym katalogu (np. *lab_04*), wejdź do podkatalogu *01_cache_size*
3. Przeglądnij plik *cache_size_array_increment.c* . Istotą kodu jest przeprowadzenie mikrobenchmarku, w którym przeglądana jest (wraz ze zwiększaniem wartości elementów o 1) tablica *array*, zmiennych typu *double*, o odpowiednio dużej do celów eksperymentu wielkości. Dwoma standardowymi zmiennymi parametrami benchmarku są: rozmiar przeglądanego obszaru tablicy i skok pomiędzy dwoma kolejnymi odwiedzanymi wyrazami. Rozmiar przeglądanego obszaru tablicy zmienia się od wartości *working_set_size_min* (w przesłanym kodzie równej 512 czyli 4kB) do wartości *working_set_size_max* (w przesłanym kodzie równej 33554432 czyli 256MB), w każdej iteracji rosnąc dwukrotnie. Dla celów eksperymentu wykrywania rozmiarów pamięci podręcznej różnych poziomów, wartość skoku pomiędzy wyrazami przyjęta jest jako równa 1 (8B). (Przeglądanie tablicy odbywa się zawsze *repeat=100* razy, w celu uzyskania istotnie większych od zera czasów pomiaru, a dodatkowo obie pętle (powtórzenia *repeat* razy i przeglądanie obszaru tablicy o rozmiarze *working_set_size* ze skokiem *stride=1*) są powtarzane tyle razy, żeby czas eksperymentu dla pojedynczego rozmiaru tablicy wyniósł ok. *SINGLE_RUN_TIME*=0.4 s.
4. Dokonaj kompilacji posługując się dołączonym plikiem *Makefile***.** W wersji załączonej w paczce *Makefile* wymaga podania poprawnej lokalizacji biblioteki PAPI (w katalogu symbolicznie oznaczanym jako PAPI_HOME) oraz istnienia lokalnie, w katalogu roboczym dwóch bibliotek: *libpapi_driver.a* i *libutl_time.a*. Dwie ostatnie biblioteki można skopiować do katalogu roboczego z odpowiednich katalogów związanych z poprzednimi laboratoriami lub dopisać właściwe lokalizacje do opcji LIB (*-L/ścieżka_papi_driver -L/ścieżka_utl_time*). *Makefile* umożliwia utworzenie dwóch plików wykonywalnych: *cache_size_array_increment.exe* i *cache_size_array_increment_papi.exe* oraz uruchomienie obu z nich. Domyślna opcja *all* w pliku *Makefile* powoduje po wywołaniu *make* utworzenie obu plików wykonywalnych i uruchomienie obu programów.
5. Przeprowadź eksperyment (w kodzie znajduje się odpowiednia rozgrzewka dla uniknięcia zaburzeń pomiarów wydajności przy początkowych operacjach programu, jednak dla pełnej wiarygodności wyników dobrze jest powtórzyć eksperyment kilkakrotnie i jak zwykle wybrać najkrótsze czasy dostępu).).
6. Wyniki eksperymentu zilustruj za pomocą wykresów. W pierwszej kolejności, dla zmiennego parametru rozmiaru tablicy (**należy użyć na osi *x* skali logarytmicznej**) na osi y należy umieścić obliczony w programie *cache_size_array_increment.exe* średni czas dostępu do pamięci (operacji modyfikacji wartości elementu tablicy, a więc operacji odczytu

i zapisu). Wykres powinien pokazać kilka zakresów rozmiaru tablicy, wewnątrz których czas dostępu jest zbliżony. Zakresy te odpowiadają sytuacji kiedy cała tablica mieści się w pamięci podręcznej określonego poziomu. W momencie, kiedy tablica przestaje się mieścić w pamięci podręcznej danego poziomu, zaczynają występować chybienia pojemnościowe i konieczność podmiany linii z pamięci wyższego poziomu, co powoduje że średni czas dostępu zaczyna coraz bardziej odpowiadać czasowi dostępu do większej pamięci wyższego poziomu.
[wyniki wydajnościowe mogą być zaburzane przez rozmaite techniki ukrywania opóźnienia stosowane przez procesor i układ pamięci – należy je dodatkowo zweryfikować, patrz p. 7]

7. Na podstawie utworzonego wykresu wyciągnij wnioski o rozmiarze pamięci podręcznej każdego poziomu.
[uwaga: pamięć L2 może być słabo widoczna na wykresach – jest tylko kilka razy większa od L1, a w przeprowadzonym eksperymencie może uzyskiwać zbliżone czasy dostępu co L1]
8. Dla potwierdzenia obserwacji utwórz wykres na podstawie wyników zwracanych przez program *cache_size_array_increment_papi.exe*. Program wykorzystuje definicje monitorowanych zdarzeń zawarte w pliku *papi_set_user_events.c* . W konkretnym przypadku omawianego eksperymentu zliczane są trzy rodzaje zdarzeń związanych z pamięcią podręczną, będące zdarzeniami chybienia w pamięci konkretnego poziomu przy operacji pobrania z pamięci:: MEM_LOAD_UOPS_RETIRED.L1_MISS, MEM_LOAD_UOPS_RETIRED.L2_MISS, MEM_LOAD_UOPS_RETIRED.L3_MISS. Na osi *x* należy umieścić tak samo jak poprzednio rozmiar tablicy, na osi *y* procent chybień w pamięci podręcznej konkretnego poziomu, odniesiony do wartości MEM_UOPS_RETIRED.ALL_LOADS określającej liczbę wszystkich wykonanych operacji pobrania z pamięci (wykres powinien zawierać trzy krzywe).
[wartości bezwzględne liczby operacji i chybień mogą nie być oczywiste (np. kiedy kompilator użyje operacji wektorowych do odczytu (pobranie dwóch liczb w jednej operacji) lub stosuje inne techniki ukrywania opóźnienia, mogą też zawierać niedokładności, mimo, że eksperyment jest zaprojektowany tak, żeby wyeliminować zaburzenia pomiaru; niemniej wartość zmiany procentu chybień w stosunku do wszystkich operacji powinna pozostawać miarodajną wskazówką określającą użycie pamięci podręcznej]
9. W sprawozdaniu umieść opis wykonanego zadania (na czym polegał benchmark), utworzone wykresy wraz z tabelami zawierającymi dane do wykresów i odpowiednim opisem wyników oraz wnioski z eksperymentu.

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

10. Zadanie 2 (3,5). Badanie rozmiaru linii pamięci podręcznej.

11. Przejdź do podkatalogu *02_cache_line_size*.
12. Przeanalizuj zawartość pliku *cache_line_size_array_increment.c* . Istotą zawartego kodu jest przeprowadzenie podobnego eksperymentu jak dla wykrywania rozmiaru pamięci podręcznych, ale tym razem z innymi parametrami, dobranymi w celu wykrycia rozmiaru pojedynczej linii pamięci L1. Parametr *working_set_size_min* jest równy parametrowi *working_set_size_max*, ustalonym jako równy rozmiarowi tablicy (w przesłanej paczce ustawionemu na 33554432, czyli 256MB). Dla takiego stałego rozmiaru tablicy przeprowadzana jest seria eksperymentów ze zmiennym skokiem (*stride*) pomiędzy dwoma kolejno odwiedzanymi elementami tablicy, od wartości 1 (8B) do wartości 1024 (8192B). Wielokrotne powtarzanie przeglądania tablicy w celu uzyskania wiarygodnych pomiarów wydajności jest zrealizowane na podobnej zasadzie jak w poprzednim eksperymencie wykrywania rozmiaru pamięci podręcznych.
13. Dokonaj kompilacji posługując się dołączonym plikiem *Makefile*, dostosowując go odpowiednio. *Makefile* jest domyślnie napisany tak, że dokonuje kompilacji kodu *cache_line_size_array_increment.exe* (bez wywołania funkcji biblioteki PAPI) i uruchamia program.

14. Przeprowadź eksperyment (w kodzie znajduje się odpowiednia rozgrzewka dla uniknięcia zaburzeń pomiarów wydajności przy początkowych operacjach programu, jednak dla pełnej wiarygodności wyników dobrze jest powtórzyć eksperyment kilkakrotnie i jak zwykle wybrać najkrótsze czasy dostępu).
15. Wyniki eksperymentu zapisz w tabeli o kolumnach: skok (w bajtach), czas dostępu, stosunek czasu dostępu do czasu dostępu dla skoku dwa razy mniejszego (tej wartości oczywiście nie ma dla skoku najmniejszego)
16. Wyniki eksperymentu zilustruj za pomocą wykresu. Na osi poziomej powinien znaleźć się teraz skok pomiędzy dwoma kolejnymi wyrazami, a na osi pionowej uzyskany średni czas dostępu do pojedynczego wyrazu tablicy, obliczany uwzględniając liczbę elementów efektywnie wykorzystanych w algorytmie (algorytm modyfikuje tylko elementy oddalone od siebie o odstęp *stride*). [(uwaga: na osi x nie logarytm skoku, ale skok – inaczej charakterystyka nie będzie czytelna, zakres skoków na wykresie można ograniczyć do 2^8 = 256B; **ewentualnie zastosować można skale logarytmiczne na obu osiach – logarytm skoku na osi x i logarytm czasu dostępu na osi y**)] Rozumowanie związane z uzyskiwaniem określonej wydajności (mierzonej średnim czasem dostępu do elementu tablicy) w zależności od tego ile elementów jest efektywnie wykorzystywanych w algorytmie, w stosunku do liczby elementów pobranych z pamięci, prowadzi do następujących wniosków:
    - dla skoku równego 1 pobierane z pamięci są wszystkie elementy tablicy i wszystkie są także wykorzystywane,
    - dla skoku równego 2, pobierane są nadal wszystkie elementy tablicy (odwiedzany jest każdy blok pamięci o rozmiarze równym linii pamięci podręcznej), jednak wykorzystywany jest co drugi, co powinno prowadzić do dwukrotnego spadku wydajności, czyli dwukrotnego wzrostu średniego czasu dostępu,
    - podobnie dla dalszych wartości skoku (zmienna *stride*), aż do momentu kiedy skok przekracza rozmiar linii pamięci podręcznej, co powoduje, że nie wszystkie bloki pamięci związane z tablicą są pobierane z pamięci DRAM, a więc średni czas dostępu przestaje rosnąć równomiernie zgodnie z początkową zależnością.
17. Na podstawie tabeli (w szczególności ostatniej kolumny odpowiadającej ewentualnej liniowości przyrostu czasu dostępu) i utworzonego wykresu wyciągnij wnioski o rozmiarze linii pamięci podręcznej L1.
18. W sprawozdaniu umieść opis wykonanego zadania (na czym polegał benchmark), utworzony wykres wraz z tabelą zawierającą dane do wykresu i odpowiednim opisem wyników oraz wnioski z eksperymentu.

---------- 3.5 ----------

19. **Zadanie 3 (4,0) Badanie drożności pamięci podręcznej**
20. Przejdź do katalogu *03_associativity*.
21. Przeanalizuj kod w pliku *cache_associativity_discover.c* . Podstawowa pętla (w której wykonywana jest taka sama operacja zwiększenia elementu tablicy o 1 jak w poprzednich eksperymentach) dotyczy teraz USED_CACHE_LINES elementów oddzielonych odstępem OFFSET bloków, z których każdy odpowiada jednej linii pamięci podręcznej. W podstawowej wersji w paczce, parametr OFFSET ustawiony jest na wartość 512 (tyle ile linii 64B zawiera pamięć L1 – jeśli poprzednie eksperymenty wykazały inną wartość rozmiaru L1 niż 512*64 B, należy odpowiednio zmodyfikować kod (parametry CACHE_LINE_SIZE i OFFSET). Parametr USED_CACHE_LINES wczytywany jest z klawiatury. W pętli odwiedzany jest tylko jeden element w każdej linii pamięci podręcznej (dokładnie w bloku przyporządkowanym pojedynczej linii pamięci podręcznej), ale wystarcza to, żeby wymusić załadowanie całej linii – stąd liczba odwiedzanych elementów jest jednocześnie liczbą odwiedzanych bloków pamięci i linii L1. Jeśli USED_CACHE_LINES jest równe jeden w eksperymencie następuje wielokrotne odwiedzenie jednego tylko elementu (jednej linii). Jeśli USED_CACHE_LINES jest równe

2, odwiedzane są dwa elementy i dwa bloki, oddzielone rozmiarem pamięci podręcznej, a więc na pewno trafiające do tego samego zbioru linii pamięci L1, w ramach mechanizmu kojarzenia bloków pamięci DRAM z liniami pamięci podręcznej (bardziej szczegółowe wyjaśnienie na wykładach i w skrypcie). Jeśli pamięć L1 jest bezpośrednio odwzorowana, każdy dostęp do elementu wymusza podmianę zawartości linii (oba bloki odpowiadają temu samemu jednoelementowemu zbiorowi linii). Jeśli pamięć jest wielodrożna, oba bloki trafią do różnych linii w tym samym zbiorze linii. Wielokrotny dostęp do obu linii nie generuje dodatkowych chybień w L1. Dalsze zwiększanie parametru USED_CACHE_LINES nadal nie generuje przyrostu liczby chybień w L1, do momentu kiedy liczba odwiedzanych elementów (bloków pamięci DRAM) nie przekroczy liczby linii w pojedynczym zbiorze linii.

22. Dokonaj kompilacji posługując się dołączonym plikiem *Makefile*, dostosowując go odpowiednio. *Makefile* jest domyślnie napisany tak, że dokonuje kompilacji kodu *cache_associativity_discover_papi.exe* i uruchamia program.
23. Przeprowadź eksperyment wykonując wielokrotnie pętle dla różnych zadanych z klawiatury wartości parametru USED_CACHE_LINES. Zaobserwuj zwracaną wartość licznika dla zdarzenia chybienia w L1, MEM_LOAD_UOPS_RETIRED.L1_MISS – jako PAPI event 3 (w czwartej linii, wartość przeciętna *average*).
24. Wyniki eksperymentu zilustruj za pomocą wykresu. Na osi poziomej umieść wartość parametru USED_CACHE_LINES, a na osi pionowej procent chybień w L1 (obliczany jako iloraz MEM_LOAD_UOPS_RETIRED.L1_MISS przez MEM_UOPS_RETIRED.ALL_LOADS).
25. Na podstawie utworzonego wykresu wyciągnij wnioski o rozmiarze zbioru linii pamięci L1, do którego przyporządkowany jest każdy z bloków pamięci DRAM – jaką drożność pamięci L1 wskazuje eksperyment?
26. W sprawozdaniu umieść opis wykonanego zadania (na czym polegał benchmark), utworzony wykres wraz z tabelą zawierającą dane do wykresu i odpowiednim opisem wyników oraz wnioski z eksperymentu.

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

27. (4.5) Zilustruj funkcjonowanie benchmarku schematem, na którym znajdą się: pamięć podręczna złożona z kilku (co najmniej 2) zbiorów składających się z kilku (co najmniej 2) linii oraz pamięć DRAM z zaznaczonymi blokami o rozmiarze 1 linii pamięci podręcznej i strzałkami określającymi możliwe przyporządkowanie bloków do linii. Posługując się schematem wyjaśnij jakie byłyby dla niego parametry CACHE_LINE_SIZE i OFFSET oraz jak wyglądałoby działanie benchmarku dla różnych wartości USED_CACHE_LINES

---------- 4.5 ----------

Dalsze kroki (5.0):

1. W ostatnim eksperymencie (*associativity*) odkomentuj wczytywanie parametru OFFSET i potwierdź rozmiar pamięci L1: dla wartości USED_CACHE_LINES o jeden większej od uzyskanej drożności L1 (generującej duży procent chybień w L1), zmniejszaj kolejno wartość OFFSET od 512, za każdym razem dzieląc ją na pół (256, 128 itd.), aż do momentu kiedy liczba chybień w L1 znów znacznie zmaleje. Ostatnia uzyskana wartość OFFSET z dużym procentem chybień, pomnożona razy drożność L1 powinna dać rozmiar L1 w liniach.
    1. posługując się tym samym schematem co w p. 27 opisz działanie benchmarku dla przyjętych tam parametrów
    2. wyniki procentu chybień w L1 dla różnych wartości parametru OFFSET zapisz w tabelce

Zawartość sprawozdania (elementy wymagane, najlepiej wplecione w opis wykonanych kroków):

1. Badanie rozmiaru pamięci podręcznych na podstawie wyników wydajnościowych
    1. kod najważniejszej pętli mikro-benchmarku
    2. opis działania pętli dla parametrów dobranych do badania rozmiaru pamięci podręcznych
    3. wyniki wydajnościowe w postaci wykresu (i najlepiej także tabeli z danymi)
    4. analiza i wnioski
2. Badanie rozmiaru pamięci podręcznych na podstawie zliczania zdarzeń chybienia w pamięci podręcznej danego poziomu
    1. opis zliczanych zdarzeń i kodu z dodanymi wywołaniami funkcji interfejsu z PAPI
    2. wyniki uzyskane w teście w postaci wykresu (i najlepiej także tabelki z danymi)
    3. analiza i wnioski
3. Badanie rozmiaru pojedynczej linii pamięci podręcznej
    1. kod najważniejszej pętli mikro-benchmarku
    2. opis działania pętli dla parametrów dobranych do badania rozmiaru pojedynczej linii pamięci podręcznej (z uwzględnieniem analizy liczby pobieranych danych z pamięci i liczby efektywnie wykorzystywanych danych w kodzie – tylko ta ostatnia służy do raportowania wydajności w GB/s)
    3. wyniki wydajnościowe w postaci wykresu (i najlepiej także tabeli z danymi)
    4. analiza i wnioski
4. Badanie drożności pamięci podręcznej (liczby linii w pojedynczym zbiorze linii do których może być odwzorowany pojedynczy blok pamięci DRAM o rozmiarze jednej linii)
    1. kod najważniejszej pętli mikro-benchmarku
    2. schemat pamięci podręcznej i odwzorowania bloków pamięci DRAM w bloki linii dla wybranych parametrów
    3. opis działania pętli dla parametrów przykładowych na schemacie – uogólnienie na przypadek dowolnych wartości parametrów
    4. wyniki uzyskane w teście w postaci wykresu (i najlepiej także tabelki z danymi)
    5. analiza i wnioski
5. **Porównanie uzyskanych wyników z parametrami odczytanymi z dokumentacji mikroprocesora (ewentualnie z internetu lub z odpowiedniej tabelki na slajdach z wykładu :) - serwery Honorata i ESTERA to 2 mikroprocesory o architekturze rdzenia Broadwell**
6. Dowolny format dla kroków dodatkowych