Analiza i modelowanie wydajności obliczeń

Lab 5. Pomiary opóźnienia i przepustowości pamięci

Wstęp.

- *Czas poświęcony na pobieranie danych dla rozkazów wykonywanych przez potoki przetwarzania rdzeni stanowi często podstawowy składnik czasu wykonania programu. Każde odniesienie do pomięci w kodzie asemblera powoduje uruchomienie mechanizmów dostarczania danych. Dostarczanie nigdy nie odbywa się poprzez proste pobranie wartości pojedynczej zmiennej, lecz zawsze wykorzystywane są rozmaite techniki ukrywania opóźnienia: współbieżne działanie układów pamięci DRAM, przesyłanie danych szerokimi magistralami, zastosowanie kilku poziomów pamięci podręcznej, pobieranie z wyprzedzeniem (prefetching) i szereg innych.*
- *W ramach laboratorium badane są różne warianty kodu źródłowego, w którym transfery danych z i do pamięci stanowią podstawowy składnik czasu wykonania (założone jest, że pozostałe operacje wykonywane są współbieżnie, w tle, dzięki czemu czas ich wykonania można pominąć). Programy używają tablic zmiennych i pisane są tak, żeby dostępy do tablic w kodzie były tłumaczone przez kompilatory na dostępy do danych w asemblerze (tzn. wielokrotne dostępy do danych nie są zamieniane na dostępy do rejestrów z pominięciem operacji na pamięci). Na podstawie kodu można określić ile danych jest pobierane do potoków przetwarzania (odczyty wartości zmiennych), a ile zapisywane (operacje zapisu wartości zmiennych). Zliczając wszystkie dostępy do pamięci można uzyskać ich liczbę w całym badanym fragmencie programu, po czym mierząc czas wykonania fragmentu uzyskać średni czas dostępu do pojedynczej danej. Przeliczając dostępy do danych na liczbę transferowanych bajtów można uzyskać* ***efektywną przepustowość*** *układu pamięci na potrzeby programu.*
- *W celu uniezależnienia wyników od zmiennej częstotliwości pracy procesora programy umożliwiają także uzyskanie wyników nie tylko jako czasów dostępu do zmiennej w ns i przepustowości w GB/s, ale także jako opóźnienia (w dostępie do pojedynczej zmiennej) liczonego w taktach zegara i przepustowości w bajtach na takt (B/cycle).*
  - *W tym celu z liczników sprzętowych pobierane są: liczba taktów referencyjnych (z częstotliwością nominalną rdzenia) i liczba taktów rzeczywistych, przy realizacji określonego fragmentu kodu. Pozwala to obliczyć przyspieszenie taktowania rdzenia, a po przemnożeniu przez częstotliwość nominalną (będącą charakterystyką procesora, zawartą np. w pliku* /proc/*cpuinfo) uzyskać częstotliwość rzeczywistą pracy rdzenia, przy realizacji badanego fragmentu kodu. Znajomość czasu pojedynczego taktu, jako odwrotności częstotliwości pracy, umożliwia odpowiednie przeliczenia wyników.*
- *Badanie dotyczy pamięci DRAM i pamięci podręcznych różnych poziomów. Tablice w benchmarkach są odczytywane i zapisywane wielokrotnie, a ich zmieniający się rozmiar powoduje, że początkowo mieszczą się w całości w pamięci L1, potem w L2, następnie w L3, a ostatecznie przekraczają rozmiar L3. Jeśli rozmiar jest odpowiednio duży (tablica znacznie przekracza rozmiar pamięci danego poziomu) to przy kolejnym przeglądaniu tablicy i dostępie do konkretnego jej elementu wartość z dostępu przy poprzednim przeglądaniu jest już podmieniona przez inne elementy tablicy i należy ją ponownie pobrać (nie ma wystarczającej lokalności czasowej, żeby korzystać z pamięci danego poziomu i konieczne jest pobieranie z pamięci poziomu dalszego od rdzeni przetwarzania). Dzięki temu, dla pewnych rozmiarów tablic można uznać, że mierzone wydajności dostępu dotyczą pamięci konkretnego poziomu (a dokładnie sytuacji, kiedy po sprawdzeniu pamięci poziomów bliższych rdzeniowi nie znaleziono aktualnych danych, nastąpiło chybienie i pobranie linii z pamięci badanego poziomu).*

- *We wszystkich częściach laboratorium badanie wykonania nie jest ukierunkowane na uwzględnienie działania pamięci wirtualnej (choć w sprawozdaniu można także uwzględnić uwagi, kiedy spodziewamy się występowania błędów stron lub podmian linii w pamięci podręcznej TLB).*

Kolejne kroki realizowane w ramach zajęć:

1. **Zadanie 1 (obowiązkowe) Badanie czasu dostępu do danych w tablicach, przy odczycie realizowanym w pętli ze skokiem między kolejno odwiedzanymi elementami i z użyciem techniki *pointer chasing* – próba określenia opóźnienia w dostępie do pamięci konkretnego typu i poziomu (L1, L2, L3, DRAM)**

- *W pierwszej części laboratorium (eksperymenty* pointer chasing*) badane są takie warianty kodu, w których celem jest uzyskanie możliwie **najdłuższego czasu dostępu do pojedynczej zmiennej i najniższej przepustowości.** Innymi słowy, chodzi o to, żeby dla danej liczby dostępów do danych czas wykonania był jak najdłuższy. Takie badanie może być uznane za badanie **opóźnienia (latency)** w realizacji dostępów do pamięci. Celem jest napisanie kodu w taki sposób, żeby sprzęt realizował wymagane przez program dostępy do pamięci, ale żeby wszystkie stosowane przez sprzęt techniki ukrywania opóźnienia zawodziły. Należy znać te techniki i pisać kod tak, żeby przeciwdziałać ich wykorzystaniu (np. jeśli pobrana jest cała linia pamięci podręcznej, to kod stara się wykorzystać tylko jedną wartość zmiennej z całej linii). Dostępy są w związku z tym tak zorganizowane, żeby nie występowała lokalność przestrzenna (duże odstępy w pamięci między kolejnymi odczytywanymi danymi). Specyfika eksperymentów* pointer chasing *polega na tym, że miejsce kolejnego odczytu jest odczytywane z aktualnej pozycji w tablicy – tym samym powstaje zależność uniemożliwiająca wykorzystanie potokowości i szeregu innych optymalizacji.*

    1. **Pobierz ze strony przedmiotu paczkę *mem_lat.tgz* i rozpakuj w nowym katalogu np. *lab_05***
    2. **Przejdź do katalogu *01_latency/02_random_pointer_chasing/***
    3. **Zapoznaj się z kodem w pliku *random_pointer_chasing.c* .** Polega on na stosowaniu częstej w przypadku pomiaru opóźnienia pamięci strategii "ścigania wskaźnika" (*pointer chasing*). W liniach ok. 163-173 znajduje się właściwy kod mikrobenchmarku – polega on na odwiedzaniu wybranych elementów tablicy *array*, przy czym indeks następnego odwiedzanego wyrazu jest odczytywany jako zawartość aktualnego elementu tablicy (dzięki temu kompilator nie wie jaki jest to wyraz i nie może zastosować szeregu optymalizacji służących ukrywaniu opóźnienia, np. potokowości).
        1. W przykładowym kodzie pętla odwiedzania elementów tablicy znajduje się wewnątrz pętli realizującej wielokrotne powtórzenia tych samych operacji (w celu uzyskania czasu wykonania wystarczająco dużego do dokładnego pomiaru czasu) , a obie te pętle umieszczone są w podwójnej pętli po dwóch parametrach: rozmiarze tablicy *working_set_size* (w pętli wewnętrznej) oraz wartości zmiennej *stride* (w pętli zewnętrznej), która określa skok pomiędzy odwiedzanymi elementami w tablicy (w efekcie w kodzie dostęp uzyskiwany jest tylko do *working_set_size/stride* elementów tablicy)
            - rozmiar tablicy zmienia się od wartości minimalnej, określanej jako rozmiar strony (PAGE_SIZE), do wartości maksymalnej, ustalanej jako iloczyn parametrów PAGE_SIZE i MULT (oba parametry można samodzielnie modyfikować)
            - wartości *stride* w pętli zaczynają się od 64B – dzięki temu istnieje pewność, że żadne z dwóch odwiedzanych elementów tablicy nie znajdują się w tej samej linii pamięci podręcznej, a więc nie zachodzi lokalność przestrzenna

2. **Dodatkowo dane w tablicy *array* są poddane losowej permutacji elementów – tak żeby skoki pomiędzy kolejno odwiedzanymi elementami odbywały się losowo, do przodu i do tyłu, przeskakując losową liczbę elementów tablicy (dotyczy to zawsze tylko wybranych elementów rozmieszczonych z odstępem *stride* w tablicy)**
    - Inicjowanie tablicy *array* odbywa się w linii 106 i polega na wskazaniu dla każdego elementu, jako następnego odwiedzanego, elementu oddalonego o *stride* od niego. Następnie w liniach 116-130 dokonywane jest mieszanie elementów tablicy.

4. Uruchom program przez wpisanie *make* w bieżącym katalogu (w przypadku problemów sprawdź ścieżki, biblioteki, wybór kompilatora, opcje kompilacji itp.)

[na serwerze uzywanie kompilatora *icc* wymaga jak zwykle wywołania:

***source /opt/intel/oneapi/setvars.sh intel64;*]**

- zaobserwuj w jaki sposób uzyskiwane są parametry wyświetlane na wydrukach:
    - liczby dostępów obliczane z kodu źródłowego
    - czasy wykonania z odpowiednich funkcji
    - częstotliwość pracy z parametrów uzyskiwanych w specjalnej funkcji *papi_return_clock_ratio* z pliku *papi_set_user_events.c*

5. **Zanotuj wyniki pomiarów dla wybranych wartości skoku (w bajtach np. 64, 4096, 32768 – rozmiar linii L1, rozmiar strony pamięci wirtualnej, rozmiar L1) i kolejnych rozmiarów tablic.**
    1. Na podstawie wyników oblicz dla każdej kombinacji rozmiaru tablicy i skoku:
        - czas dostępu do pojedynczego elementu tablicy w nanosekundach
        - czas dostępu do pojedynczego elementu tablicy w taktach

    Zaobserwuj jak zmieniają się wartość czasu dostępu do pojedynczego elementu tablicy (*access time)* i opóźnienie (*latency*) w zależności od rozmiaru tablicy i wartości skoku (liczba dostępów na potrzeby wydruku jest liczona jako liczba elementów pobranych z pamięci i efektywnie wykorzystanych w kodzie, a nie cała pojemność pobranych linijek pamięci podręcznej lub dane pobierane w efekcie pobierania z wyprzedzeniem).

    Rozmiar tablicy decyduje o tym, jak wiele pamięci zajmuje tablica, czyli np. czy w całości mieści się w pamięci podręcznej konkretnego poziomu. Zaobserwuj różnice pomiędzy przypadkami, kiedy tablica w całości mieści się w L1, L2, L3 (wykorzystaj uzyskane w poprzednim laboratorium dane o rozmiarach pamięci podręcznych różnych poziomów) oraz kiedy jest większa od L3.

6. **Na podstawie danych skonstruuj wykres zależności opóźnienia (*latency*) od rozmiaru tablicy, dla wybranego przypadku skoku (w bajtach, np. 128 dzięki czemu wyniki obejmują szeroki zakres rozmiarów tablicy i każdy odwiedzany element jest w innej linii pamięci podręcznej) *(Krzywa na wykresie z rozmiarem tablicy na osi x i czasem dostępu na osi y (w taktach), osie powinny być w skali logarytmicznej)*.**
7. **Znając z poprzedniego laboratorium rozmiary pamięci L1, L2, L3 i dzięki temu wiedząc, dla których rozmiarów tablicy czasy dostępu są w praktyce czasami dostępu do pamięci określonego poziomu, wybierz wartości maksymalne dla każdej z pamięci L1, L2, L3, DRAM – wartości te mogą być uznane za przybliżenia opóźnienia w dostępie do każdej z pamięci**
    - **w przypadku różnic pomiędzy wartościami odczytanymi dla danych z wykresu oraz danych z p.5 zinterpretuj różnice i wybierz odpowiednie wartości z właściwym komentarzem**

2. **Zadanie 2 (obowiązkowe). Badanie maksymalnej możliwej do uzyskania w programie jednowątkowym przepustowości pamięci konkretnego typu i poziomu (L1, L2, L3, DRAM)**
    1. **Przejdź do katalogu *02_throughput/02_multiple_arrays/* .**
    2. **Przeanalizuj plik *multiple_arrays.c* .** Ideą zawartego w nim kodu jest realizacja pętli przetwarzania wyrazów w wielu tablicach, tak żeby umożliwić sprzętowi (także poprzez odpowiednią kompilację) wygenerowanie jak największej liczby niezależnych żądań dostępu do pamięci. W linii 115 wykonywana jest operacja (dwa mnożenia i dodawanie), w której biorą udział elementy pięciu tablic. Tablice i dostęp do nich są tak zorganizowane (różne rozmiary tablic, skok zawsze równy 1), aby uzyskać pełną lokalność przestrzenną (każdy pobrany bajt z pamięci jest wykorzystany w algorytmie). Tablice są zaalokowane w pamięci z wyrównaniem do granicy 64B dla optymalizacji wykorzystania pamięci podręcznej.
    3. **Uruchom program przez wpisanie *make* w bieżącym katalogu (w przypadku problemów sprawdź ścieżki, biblioteki, wybór kompilatora, opcje kompilacji itp.)**
    4. Zaobserwuj różnice w wydajności związane z rozmiarem tablic (w szczególności sumą rozmiarów wszystkich tablic) i jego proporcją w stosunku do rozmiaru kolejnych poziomów pamięci podręcznej, a także z wykonaniem skalarnym (dostępy 64-bitowe) i wektorowym (dla opcji *-march=core-avx2* rejestry i dostępy 256-bitowe).
    5. **Na podstawie danych skonstruuj wykres zależności przepustowości od rozmiaru tablicy. Na osi x umieść rozmiar tablicy w skali logarytmicznej, na osi y przepustowość w B/takt.**
    6. **Określ przepustowość charakteryzującą kolejne poziomy w hierarchii pamięci (L1, L2, L3, DRAM) dla jednowątkowego wykonania skalarnego i wektorowego.**

3. **Zadanie 3 (3.0). Porównanie minimalnej i maksymalnej wydajności pamięci poszczególnych poziomów, uzyskanej w eksperymentach**
    1. Na podstawie wyników otrzymanych w poprzednich punktach wypełnij tabelę:

	L1	L2	L3	DRAM
Czas dostępu [ns] ([takt]) - maksymalny				
Czas dostępu [ns] ([takt]) - minimalny				
Przepustowość [GB/s] (B/takt) - minimalna				
Przepustowość [GB/s] (B/takt) - maksymalna				
Przybliżony stosunek wartości maks do min				

    2. Wyciągnij wnioski w jakich granicach w konkretnych programach mogą mieścić się uzyskiwane przepustowości (czasy dostępu do zmiennej) dla różnych poziomów pamięci. Spróbuj oszacować jaki zysk może przynieść:
        1. zastosowanie podstawowej lokalności odniesień, a więc użycie zamiast pamięci DRAM pamięci podręcznych
        2. maksymalizacja lokalności odniesień (prowadząca w praktyce do używania pamięci podręcznej coraz bliższej rdzeniom, L1 zamiast L2 i L3, L2 zamiast L3 itd.)
        3. optymalna organizacja dostępów do pamięci (przejście od dostępów nieoptymalnych dla danego typu i poziomu pamięci – charakteryzujących się maksymalnymi czasami

dostępu, do dostępów optymalnych – charakteryzujących się minimalnymi czasami dostępu i maksymalną przepustowością

Co jest ważniejsze, zastosowanie lokalności odniesień czy poprawna organizacja dostępów (czy wystarczy używać pamięci podręcznej danego poziomu, czy trzeba to robić optymalnie)? Czy Pamięci różnych typów i rozmiarów (L1, L2, L3, DRAM) różnią się zakresem zysków i strat związanych z nieodpowiednią organizacją dostępów?

3. **Wnioski zapisz w sprawozdaniu, postaraj się posługiwać kategoriami względnymi – w stosunku do rozwiązania x zastosowanie rozwiązania y może przynieść z-krotny zysk, czyli z-krotne zmniejszenie czasu wykonania programu**
4. **Ważnym wnioskiem praktycznym jest oszacowanie jak wiele można stracić jeśli mając w kodzie określoną wymaganą liczbę dostępów do danych, zrealizuje się te dostępy w sposób najwolniejszy z możliwych, zamiast zrealizować je w sposób optymalny**

Dalsze kroki:

1. **(4.0) Poddaj bardziej szczegółowej analizie różnice miedzy optymalnym i nieoptymalnym dostępem do pamięci (w szczególności DRAM, ale także L1, L2, L3). Wypisz, np. w punktach, jak należy starać się pisać kod, aby uzyskać optymalne działanie pamięci.**
   - W analizie posłuż się zawartością plików asemblera i wiedzą na temat adresów generowanych w kolejnych rozkazach (tworzenie plików przez kompilacje z dodatkową opcją -S). Zwróć uwagę na różnice wersji skalarnej i wektorowej kodu.

2. **Badanie czasu dostępu do danych w tablicach, realizowanego z użyciem techniki *pointer chasing*, bez losowej permutacji elementów**
   1. **Przejdź do katalogu *01_latency/01_strided_pointer_chasing/***
   2. **Zapoznaj się z kodem w pliku strided*_pointer_chasing.c* .**
   3. Metoda pomiaru w pliku *strided_pointer_chasing.c* dla małych wartości skoku pozwala uzyskać pewną lokalność przestrzenną odniesień, a także umożliwia zastosowanie mechanizmu przewidywania miejsca kolejnego pobrania danych i *prefetchingu* (dla dużych wartości skoku mechanizm ten może przestać funkcjonować).
   4. **Uruchom program przez wpisanie *make* w katalogu *01_strided_pointer_chasing/* (w przypadku problemów sprawdź ścieżki, biblioteki, wybór kompilatora, opcje kompilacji itp.)**

      Skok pomiędzy elementami decyduje o tym jaki procent danych pobranych do linijki pamięci podręcznej zostaje efektywnie wykorzystany w algorytmie, a także które zbiory linii są wykorzystywane w asocjacyjnej pamięci podręcznej. W końcu, skok decyduje o tym jak często zmieniana jest strona w pamięci wirtualnej, a więc jak często można oczekiwać chybień w pamięci TLB. Zwróć uwagę, że dla najmniejszej wartości skoku, strategia *pointer chasing* (wprowadzająca zależność pomiędzy każdymi dwoma kolejnymi dostępami do pamięci) spowalnia dostępy w taki sposób, że znika różnica między pamięciami różnych poziomów (pamięciom wyższych poziomów i pamięci DRAM pomaga *prefetching*, optymalizujący pracę sprzętu). Dopiero dla wyższych wartości skoku widać, że niektóre poziomy pamięci radzą sobie z nimi lepiej, a inne gorzej.

      Dla każdej z pamięci zaobserwuj ewentualne różnice, kiedy skok przekracza np. rozmiar linii pamięci podręcznej, rozmiar strony pamięci wirtualnej czy rozmiar pamięci podręcznej innego poziomu.

5. **Na podstawie danych z eksperymentu skonstruuj wykres zależności opóźnienia od rozmiaru tablicy, dla wybranych wartości skoku, np. ponownie w bajtach: 4, 64, 4096, 32768 – rozmiar int, rozmiar linii L1, rozmiar strony pamięci wirtualnej, rozmiar L1. Na osi x umieść rozmiar tablicy w skali logarytmicznej, na osi y opóźnienie w taktach.**
   - **spróbuj zinterpretować różnice pojawiające się w tym eksperymencie w stosunku do badania random_pointer_chasing**

3. W katalogu *01_latency/03_strided_array_increment* znajduje się plik źródłowy programu mierzącego czas dostępu i przepustowość dla rozważanego w poprzednim laboratorium testu polegającego na zwiększaniu o 1 wartości wybranych elementów w tablicy. **Znaczenie tego testu polega na tym, że choć uzyskane wyniki nie są aż tak ekstremalne jak dla techniki *pointer chasing*, to postać pętli testowej jest bliższa rzeczywistym programom.** Parametrami algorytmu są rozmiar tablicy i skok pomiędzy kolejnymi odwiedzanymi elementami. Uruchomienie programu zwraca cały zestaw wyników, spośród których można wybrać specyficzne kombinacje wartości parametrów, zmodyfikować kod, tak żeby wykonywał obliczenia tylko dla jednego wybranego przypadku kombinacji parametrów i poddać tak wybrane specjalne przypadki dokładniejszej analizie.
   1. Za pomocą biblioteki PAPI, poprzez interfejs wykorzystywany w poprzednim laboratorium, można uzyskać informacje o rozmaitych zdarzeniach dotyczących pamięci – w pliku *papi_set_user_events.c* znajdują się przykładowe liczniki, dające ważne informacje o wykonaniu programu.
   2. Ciekawymi kombinacjami parametrów są takie, gdzie jako jedna z lub obie wartości występują np. rozmiar zmiennej double, rozmiar linii pamięci podręcznej (uzyskany w poprzednim laboratorium), rozmiar strony pamięci wirtualnej (4kB), rozmiary L1, L2, L3 (uzyskane w poprzednim laboratorium).

4. W katalogu *02_throughput/01_array_increment/* znajduje się plik *array_increment.c* z algorytmem takim jak powyżej, ale dostosowanym do pomiaru przepustowości (m.in. skok równy 1). Poza poddaniem go analizie takiej jak w punkcie poprzednim, można badać możliwości zwiększenia wydajności w algorytmie, poprzez zwiększenie liczby tablic przetwarzanych w kodzie (uwaga: zwiększenia liczby przetwarzanych tablic zmienia sposób obliczania całkowitej liczby dostępów do pamięci w zmiennej *nr_accesses*). Wspomaganiem badania wydajności może być przeglądanie utworzonych plików asemblera, dla wersji skalarnej (dostępy 64-bitowe) i wektorowej. Jakie wydajności daje się osiągnąć?

Sprawozdanie:

1. Zrealizowane kroki, spostrzeżenia z analizy kodu (a także ewentualnie odpowiadającego kodu asemblera i uzyskanych wartości liczników sprzętowych), wykresy, opis wykresów, wnioski – zgodnie z wskazówkami w odpowiednich punktach instrukcji i regulaminem laboratoriów