Analiza i modelowanie wydajności obliczeń

Lab 2. Wpływ skoków i przewidywania skoków na czas wykonania programu

Kroki:

1. Pobierz ze strony przedmiotu paczkę *branch_prediction.tgz*, rozpakuj w nowym katalogu *lab_02* (np. za pomocą: *tar xvzf branch_prediction.tgz)*, wejdź do podkatalogu *branch_prediction*
2. Sprawdź definicje w pliku *Makefile* (szczególnie wybór kompilatora, konsolidatora i ich opcji, położenie odpowiednich katalogów z plikami nagłówkowymi i bibliotekami, m.in. położenie biblioteki PAPI – najbezpieczniejszymi są zawsze ścieżki absolutne plików i katalogów)
   Na serwerze Honorata biblioteka PAPI jest zainstalowana w udostępnionym katalogu:
   PAPI_HOME = /home/students_wo/common_files/papi-6.0.0/src
   1. Plik *Makefile* domyślnie tworzy dwa pliki wykonywalne – jeden standardowy, *branch_prediction* (tylko z pomiarem czasu), drugi, *branch_prediction_papi,* z pomiarem czasu i ze zliczaniem zdarzeń za pomocą liczników sprzętowych oraz aplikacji PAPI (Performance Application Programming Interface)
      1. PAPI umożliwia pobieranie danych o liczbie zdarzeń sprzętowych dla wybranych fragmentów kodu (polecenie systemowe *perf stat* udostępnia statystyki dla całości wykonania programu)
      2. PAPI będzie podstawowym interfejsem wykorzystywanym w trakcie laboratoriów do uzyskiwania informacji o zdarzeniach sprzętowych
      3. PAPI powinno być zainstalowane w systemie na komputerach w laboratorium (*/usr/lib64/libpapi.so, /usr/include/papi.h)*
      4. W przypadku problemów z korzystaniem z biblioteki dostarczanej w ramach dystrybucji Linuksa lub jej nieaktualnością, można zainstalować własną wersję (korzystając ze strony *https://icl.utk.edu/papi/*)
      5. Pierwszym krokiem do uruchomienia testu przewidywania rozgałęzień jest utworzenie biblioteki pomiaru czasu *libutl_time.a* w katalogu *utd_time_unix* poleceniem: ***make -f Makefile_no_papi_lib recreate_time_lib*** (polecenie wydawane jest w katalogu *branch_prediction*, działa także: *make recreate_time_lib*) - utworzona biblioteka będzie wykorzystywana w kolejnych laboratoriach
      6. Następnie można skompilować kod bez zliczania zdarzeń sprzętowych *PAPI*: ***make -f Makefile_no_papi_lib***
      7. Utworzony program *branch_prediction* można uruchomić wraz z zliczaniem zdarzeń sprzętowych przez narzędzie *perf*: ***perf stat ./branch_prediction*** (zliczanie dokonywane jest dla całego programu i tutaj ma znaczenie dla porównania niektórych wyników z wersją *PAPI*)
      8. Biblioteka wspomagająca zbieranie danych z liczników sprzętowych *libpapi_driver.a (*w katalogu *utd_papi_driver)* jest tworzona poleceniem ***make recreate_papi_lib*** (wydawanym w katalogu *branch_prediction*) – także ta biblioteka będzie wykorzystywana w kolejnych laboratoriach
      9. Uruchomienie kodu z biblioteką *PAPI* (utworzonego standardowym ***make****):* *./branch_prediction_papi*
3. Zgodnie z instrukcjami powyżej skompiluj i uruchom program z pliku źródłowego *branch_prediction.c*
4. Przeanalizuj kod źródłowy programu, w szczególności pętle z:
   1. odczytywaniem z tablicy wygenerowanych liczb losowych
   2. wykonywaniem operacji arytmetycznych
   3. odczytywaniem z tablicy wygenerowanych liczb losowych, wykonywaniem operacji arytmetycznych i wykonywaniem instrukcji warunkowych zależnych od wartości wylosowanych liczb
5. Wyciągnij z kodu źródłowego wniosek ile skoków zostanie wykonanych, ile nie wykonanych i czy jest możliwe przewidzenie wykonania lub nie skoku
6. Przeanalizuj plik źródłowy *papi_driver.c* w katalogu *../utd_papi_driver*, z kodem stanowiącym adapter do interfejsu *PAPI* – funkcje z pliku *papi_driver.c* są uniwersalne i będą wykorzystywane (jako biblioteka) w kolejnych laboratoriach

7. Przeanalizuj plik źródłowy *papi_set_user_events.c* w katalogu *branch_prediction* – funkcja *papi_set_user_events* wypełnia struktury danych wykorzystywane w funkcjach z *papi_driver.c* danymi specyficznymi dla konkretnego zadania - w tym przypadku dla przewidywania rozgałęzień
    1. korzystając z plików biblioteki *PAPI* i informacji w internecie znajdź znaczenie wykorzystywanych w programie zdarzeń: PAPI_BR_UCN , PAPI_BR_CN, PAPI_BR_TKN, PAPI_BR_NTK, PAPI_BR_MSP, PAPI_BR_PRC, PAPI_BR_INS
8. Uruchom program *branch_prediction_papi* i zanalizuj liczby powyższych zdarzeń w odpowiednim fragmencie kodu źródłowego (wnioski zawrzyj w sprawozdaniu z laboratorium)
9. Porównaj czasy wykonania kolejnych pętli w programie – powiąż czasy z liczbą skoków, w tym błędnie przewidzianych przez układ przewidywania rozgałęzień, wyciągnij wnioski co do narzutu związanego z błędami przewidywania pętli
   [Przed wyciąganiem wniosków przeczytaj raz jeszcze zasady dokonywania pomiarów czasu wykonania , warto np. zwrócić uwagę na częstotliwość pracy zegara uzyskiwaną przy uruchomieniu kodu z narzędziem *perf*]
10. Posługując się zmienną *percent_oper* zmień procent spełnianych warunków w instrukcji *if* i liczbę wykonywanych operacji arytmetycznych (np. *percent_oper = 0.0, 0.25, 0.5, 0.75, 1.0*)
    1. zanalizuj liczbę prawidłowo przewidywanych skoków w zależności od wartości zmiennej *percent_oper*, zwłaszcza dla wartości 0.0, 0.5 i 1.0
    2. wyciągnij dalsze wnioski dotyczące działania układu przewidywania skoków i wpływu źle przewidzianych skoków na czas wykonania (a więc na wydajność kodu) – wnioski zawrzyj w sprawozdaniu z laboratorium

---------- ewentualny ciąg dalszy – zależnie od planu zajęć ----------

11. Utwórz nowy podkatalog *arithmetic_operations* i skopiuj do niego plik *branch_prediction.c* zamieniając nazwę na *arithmetic_operations.c* . Skopiuj także pliki *Makefile i Makefile_no_papi_lib* oraz *papi_set_user_events.c*.
12. Podmień w plikach *make* nazwę *branch_prediction* na *arithmetic_operations* .
13. W pliku *arithmetic_operations.c* usuń wszystkie pętle poza pętlą wykonywania operacji arytmetycznych (bez warunku), obejmując tę pętlę wywołaniami odpowiednich funkcji adaptera PAPI.
14. Skompiluj i uruchom obie wersje kodu: *arithmetic_operations i arithmetic_operations_papi* .
    1. sprawdź liczbę zdarzeń związanych z przewidywaniem skoków dla tej wersji programu – wnioski umieść w sprawozdaniu
15. Zaobserwuj występowanie zależności w pętli obliczeniowej – umieść w sprawozdaniu informacje ogólne co to są zależności danych (w szczególności zależności przenoszone w pętli), na czym polega zależność w badanym kodzie, jak zależność danych wpływa na wydajność przetwarzania potokowego, jaki jest związek zależności danych z przyjętą definicją opóźnienia przy wykonywaniu operacji przez procesor
16. Dokonaj kompilacji pliku za pomocą polecenia:
    gcc -c -O3 -S arithmetic_operations.c -I../utd_time_unix
    *(*optymalizacja *-O3* jest użyta, m.in. w celu eliminacji zbędnych do uzyskania wyniku, potrzebnych tylko w celu debugowania, dostępów do pamięci) otrzymując w wyniku plik asemblera *arithmetic_operations.s*
17. Przeanalizuj kod z pliku asemblera, znajdź pętlę odpowiadającą pętli obliczeniowej z kodu źródłowego (powinna zaczynać się od etykiety, np. .L3: , zawierać tylko kilka rozkazów i kończyć skokiem do etykiety, np. jne .L3). **Zaobserwuj jak zależność danych z kodu źródłowego przenosi się do kodu asemblera. Odpowiedni fragment kodu asemblera i wnioski dotyczące zależności umieść w sprawozdaniu.**

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

18. Oblicz samodzielnie parametry CPI i IPC (patrz wykłady i skrypt) dla operacji zmiennoprzecinkowych wykonywanych w pętli (kod asemblera wskazuje, że wykonywane są dwie operacje w każdej iteracji pętli – mnożenie i dodawanie) – umieść w sprawozdaniu liczby taktów dla całej pętli, liczby operacji i wartości CPI i IPC
    1. liczbę operacji, jako łączną liczbę dodawań i mnożeń, wyznacz z kodu źródłowego i kodu asemblera – zauważ, że operacje zmiennoprzecinkowe nie mogą być wykonywane współbieżnie ze względu na zależność (można założyć, że pozostałe operacje, np. operacje

na liczbach całkowitych i porównania, wykonywane są w tle – współbieżnie, w innych potokach przetwarzania niż operacje zmiennoprzecinkowe)

2. przybliżoną liczbę taktów procesora można uzyskać uruchamiając kod (bez korzystania z biblioteki PAPI) pod nadzorem programu *perf* (*perf stat ./arithmetic_operations*), ewentualnie używając interfejsu PAPI do zliczania liczby taktów procesora w trakcie wykonania samej pętli (patrz punkt 1 w Dalszych krokach)
3. **obliczone parametry IPC i CPI dotyczą tak naprawdę uśrednionego parametru dla pary rozkazów (mnożenia i dodawania), sprzęt może optymalizować ich wykonanie, dlatego opóźnienie wykonania każdej pary niekoniecznie musi odpowiadać sumie opóźnień każdej z operacji (powinno jednak być do niej zbliżone)**
4. wynik sprawdź porównując z danymi producenta procesora, np. w "Intel® 64 and IA-32 Architectures Optimization Reference Manual", Appendix C (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-optimization-reference-manual) – ze względu na występowanie zależności w kodzie wyniki odpowiadają wartości *latency* dla badanych operacji

---------- 4.*0* ----------

Dalsze kroki:

1. Zbadaj wartości innych, wybranych przez siebie, liczników sprzętowych dla badanej pętli w *arithmetic_operations.c* – opisz motywację do wyboru i wnioski z pomiaru liczby odpowiednich zdarzeń
    1. jako pierwsze zadanie można wybrać zliczanie zdarzeń raportowanych także przez *perf* i porównanie liczb uzyskanych za pomocą obu sposobów – w sprawozdaniu powinno znaleźć się uzasadnienie dla ewentualnych różnic (jedną z możliwych przyczyn jest to, że *perf* bada cały program, a nie wybraną pętlę)
    2. w pliku *papi_set_user_events.c* w katalogu *arithmetic_operations*, w miejsce zdarzeń dotyczących przewidywania skoków (ustawianych w linijkach ok. 145-170) należy wstawić nazwy zdarzeń dotyczących całkowitej liczby taktów (PAPI_TOT_CYC), całkowitej liczby wykonanych rozkazów (PAPI_TOT_INS) i wybranych innych – spośród wymienionych w pliku *papi_set_user_events.c* w liniach 25-125
2. Dla nabycia wprawy w posługiwaniu się plikami asemblera, powtórz analizę asemblera dla kompilatora *icc* (zaobserwuj zaletę kodu produkowanego przez *icc* zawierającego dla każdego rozkazu procesora numer linii w pliku kodu źródłowego)
   (w aktualnej wersji kompilatora konieczne może być usunięcie polskich liter z napisów i włączenie opcji *-no-multibyte-chars*)

   Na serwerze Honorata w celu korzystania z kompilatora icc należy wykonać polecenie:
   ```
   source /opt/intel/oneapi/setvars.sh intel64
   [ Na potrzeby kolejnych laboratoriów najlepiej umieścić powyższą linijkę w pliku ~/.bashrc ]
   ```

Sprawozdanie :

1. Zrealizowane kroki, wnioski, fragmenty kodu dodanego do pobranych procedur lub samodzielnie zmodyfikowanego - zgodnie z regulaminem laboratoriów
    1. w sprawozdaniu powinny znaleźć się wszystkie pętle badanego kodu, z możliwie szczegółową analizą wykonywanego kodu – w szczególności liczbą wykonanych skoków i możliwościami ich przewidzenia oraz ewentualnymi zależnościami danych przenoszonymi w pętlach