Analiza i modelowanie wydajności obliczeń

Lab 11. Model wydajności "roofline"

Cel: Analiza wydajności wybranych algorytmów korzystając z modelu wydajności "roofline".

Kroki:

1. **Na podstawie wykładu i wyników poprzednich laboratoriów skonstruuj diagram *roofline* platformy, na której dokonywane są obliczenia (dla obliczeń jednowątkowych). Na diagramie zaznacz linie odpowiadające teoretycznym maksimom lub wynikom uzyskanym w testach (wydajności przetwarzania w Gflop/s – lab 3 oraz przepustowości w GB/s – lab 5; w sprawozdaniu podaj z jakich testów korzystasz).**
2. Rozpakuj paczkę *mat_vec_optimization_papi.tgz* w nowym katalogu np. *lab_11,* ewentualnie wykorzystaj pliki z laboratorium 8 (w dalszym opisie założone jest korzystanie z dostarczonej paczki – własne modyfikacje można wprowadzać do plików z paczki)
3. Zmodyfikuj plik *Makefile* wybierając jeden z kompilatorów i uruchom program w wersji bez *register blocking* i *cache blocking* (w pliku *mat_vec.c)*
   - *pierwszym krokiem może być jak zwykle:* **`make recreate_papi_lib_mth`**
4. Po przetestowaniu poprawności działania wykomentuj fragment realizujący sprawdzenie wyniku w pliku *mat_vec_test.c* (linie 65-86, cały kod jest specjalnie uproszczony, żeby ułatwić późniejsze pomiary)
   - uruchomienie poprzez *make* wyświetla szereg parametrów wykonania analizowanych później
   - w laboratorium pomijany jest wpływ zmian częstotliwości pracy procesora
5. Oblicz liczbę operacji oraz liczbę dostępów do pamięci **w algorytmie mnożenia macierz-wektor w pliku *mat_vec.c*** (wzorując się także na wynikach laboratorium 8):
   - oblicz osobno liczbę dostępów dla pliku *mat_vec_test.c* i osobno dla *mat_vec.c*
   - liczba dostępów w całym programie (jako suma powyższych) składa się z liczby zapisów i odczytów, liczba zapisów jest jednoznacznie określona przez inicjowanie tablicy a i wektora x (w pliku *mat_vec_test.c)* oraz jednokrotny zapis do wektora y (w pliku *mat_vec.c)*, natomiast liczba odczytów jest jednoznacznie określona dla *a*, natomiast dla *x* zależy od sposobu traktowania *x* (od tego jaka jest liczba transferów elementów wektora *x* pomiędzy pamięcią DRAM i różnymi poziomami pamięci podręcznej w trakcie wykonania algorytmu)

   Wskazówka – obliczenie liczby operacji i dostępów do pamięci w pliku *mat_vec_test.c* (po wykomentowaniu sprawdzenia poprawności wyniku):
   - linia 19 – ROZMIAR = WYMIAR*WYMIAR zapisów do *a* (*data write operation*)
   - linia 20 – WYMIAR zapisów do *x* (*data write operation*)
6. Na podstawie obliczonych liczb zapisów i odczytów, oblicz **intensywność arytmetyczną algorytmu mnożenia macierz-wektor (plik *mat_vec.c)*** posługując się założeniami dotyczącymi traktowania *x*
   - **w pierwszej fazie intensywność arytmetyczna dotyczy tylko dostępów do pamięci DRAM**
   - w sprawozdaniu opisz poczynione założenia i ich wpływ na obliczoną intensywność arytmetyczną – można posługiwać się wartością minimalną intensywności (*x* przez cały czas w L3) i maksymalną (dla każdej linii *a* od nowa pobierany cały wektor *x*)
7. Uruchom program pod kontrolą narzędzia *valgrind* (uwaga: *valgrind* jest rodzajem symulatora, czas wykonania jest wielokrotnie dłuższy niż standardowy): *valgrind --tool=cachegrind ./mat_vec_test_opt.exe*
   - zaobserwuj zwracane wyniki w kategoriach dotyczących danych (dostępy związane z kodem – rozkazami, *instructions, I* – w tak krótkim programie są bez znaczenia) :
     - *D refs (data references)* – liczba rozkazów dostępów do pamięci w asemblerze (uwaga na liczbę danych w pojedynczym rozkazie – patrz poniżej)

- *D1 misses* – liczba chybień w pamięci L1
- *LLd misses* – liczba chybień w pamięci L3 (*last level cache*)

8. Zweryfikuj obliczenia liczby dostępów do pamięci w algorytmie **mnożenia macierz-wektor (plik *mat_vec.c)*** za pomocą wyników z uruchomienia za pomocą *valgrind* oraz za pomocą *make,* które zwraca wyniki z interfejsu PAPI. Uwzględniając następujące fakty:
   - porównaj wyniki raportowane przez *valgrind* i przez zliczanie zdarzeń sprzętowych przez PAPI
     - dla uzyskania liczby dostępów do pamięci DRAM poprzez interfejs PAPI stosowane są dwa zdarzenia: predefiniowane zdarzenie PAPI_L3_TCM (*total cache misses)* oraz zdarzenie OFFCORE_RESPONSE_0:ANY_DATA:ANY_RFO:L3_MISS_LOCAL:L3_MISS_REMOTE:SNP_ANY Wyniki mogą się różnić, także w stosunku do wyników z *valgrind*. W analizie można rozważyć wartości maksymalne i minimalne tranferów
   - symulator *valgrind* podaje wyniki dla całego kodu – wyniki dla funkcji mnożenia macierz-wektor należy uzyskać odejmując od wartości dla całego kodu wartości dla funkcji *main* w pliku *mat_vec_test.c*
   - dostępy do pamięci (*data references)* mogą być realizowane przez operacje wektorowe – jako pierwszy etap należy określić ilu dostępom do wartości skalarnych odpowiada jeden dostęp raportowany przez *valgrind*, można do tego użyć np. kod asemblera dla badanej wersji
     (*gcc -fopenmp -S -O3 mat_vec.c* lub *icc -qopenmp -S -O3 mat_vec.c*)
     [np. *movsd* jest rozkazem dotyczącym pobrania jednej wartości 64-bitowej do rejestru *xmm*, natomiast *movapd, movaps, movups, movupd mogą* pobierać 2, 4 lub 8 wartości – należy sprawdzać adresy i rejestry, których dotyczą]
   - chybienie w pamięci oznacza pobranie całej linii (najczęściej 64 bajty), a więc kilku elementów tablicy (standardowo byłoby to 8 elementów double)
   - w algorytmie mnożenia macierz-wektor (plik *mat_vec.c)* występują odczyty i zapisy - te ostatnie tylko dla wektora y,
   - w pozostałej części programu (plik *mat_vec_test.c)* występują wyłącznie zapisy (inicjowanie *a i x*) – wyniki raportowane przez *valgrind* dla odczytów bezpośrednio dotyczą więc algorytmu mnożenia macierz-wektor, natomiast w przypadku zapisów należy od liczb raportowanych przez *valgrind* odjąć wartości obliczone dla *mat_vec_test.c*
   - obliczona liczba dostępów do pamięci DRAM powinna być równa liczbie chybień w pamięci L3 (LLC – *last level cache*)
     - ile chybień pojawiło się ze względu na odczyty *x* ? (jako różnica całkowitej liczby chybień i liczby chybień ze względu na *a* – z powodu braku lokalności czasowej i pełnej lokalności przestrzennej chybienie następuje dla (standardowo) co ósmego wyrazu *a)*
     - ile razy wektor *x* był pobierany z DRAM do L3?
   - dla bardziej dokładnego obliczenia wydajności algorytmu ważna jest także liczba chybień w pamięci L1
     - różnica między liczbą chybień w L1 i L3 to liczba trafień w L2 lub w L3
     - ile chybień pojawiło się ze względu na odczyty *x* ? (jako różnica całkowitej liczby chybień i liczby chybień ze względu na *a* – z powodu braku lokalności czasowej i pełnej lokalności przestrzennej chybienie następuje dla (standardowo) co ósmego wyrazu *a)*
     - ile razy wektor *x* był pobierany z L3 (lub L2) do L1?
9. Ustal ostateczną wartość intensywności arytmetycznej dla badanego kodu
   - oblicz intensywność arytmetyczną w liczbie operacji na jedną pobraną zmienną (s_{pma} = nr_oper/nr_access)
   - **oblicz intensywność arytmetyczna w liczbie operacji na jeden pobrany bajt, ($s_{pmB}=s_{pma}/8$ – dla liczb podwójnej precyzji)**
10. Na diagramie *roofline* obliczeń jednowątkowych zaznacz pionową linię odpowiadającą intensywności arytmetycznej algorytmu mnożenia macierz-wektor.
11. Nanieś na diagram *roofline* wynik wydajnościowy uzyskany przez program.*(uwaga: wydruk programu podaje czas wykonania wyłącznie funkcji mnożenia macierz-wektor)*

12. Dokonaj zrównoleglenia kodu poprzez odkomentowanie dyrektyw OpenMP w pliku *mat_vec.c* (pamiętaj o modyfikacji odpowiednich opcji w pliku *Makefile*, np. **-***fopenmp* dla *gcc*, **-***qopenmp* dla *icc*, efekt wykonania wielowątkowego sprawdź np. za pomocą *htop*)
    - zrównoleglenie może dotyczyć dowolnej poprawnej wersji kodu
13. Przeprowadź obliczenia dla optymalnej liczby wątków równej liczbie rdzeni stosując zmienną środowiskową OMP_NUM_THREADS
    (na serwerze Honorata: **`export OMP_NUM_THREADS=24`**).
14. Odnotuj czas wykonania algorytmu mnożenia macierz wektor, oblicz wydajność w GFLOP/s, a także w GB/s, na podstawie przyjętej wartości intensywności arytmetycznej (obliczenia należy powtórzyć kilka razy i wybrać najkrótszy czas, można także eksperymentować z przypisaniem wątków do rdzeni za pomocą **`numactl`**):
15. **Na podstawie wyników poprzednich laboratoriów skonstruuj diagram *roofline* dla całej platformy, na której dokonywane są obliczenia – tym razem dla obliczeń wielowątkowych, pokazujący pełny potencjał sprzętu. Na diagramie zaznacz linie odpowiadające teoretycznym maksimom i wynikom uzyskanym w testach (w sprawozdaniu podaj z jakich testów korzystasz).**
16. Zaznacz na diagramie pionową linię odpowiadającą intensywności arytmetycznej algorytmu mnożenia macierz-wektor.
17. Nanieś na diagram *roofline* uzyskany wynik obliczeń wielowątkowych
    - jak daleko od punktu *machine balance* znajduje się intensywność arytmetyczna mnożenia macierz-wektor dla obliczeń jednowątkowych i obliczeń wielowątkowych?
    - które z obliczeń lepiej wykorzystują potencjał potoków przetwarzania maszyny?

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

18. Przeprowadź analizę podobną do powyższej dla kodu obliczeń iloczynu macierz-macierz badanego w poprzednim laboratorium
    1. skopiuj wszystkie pliki do katalogu obecnego laboratorium
    2. wykomentuj zliczanie zdarzenia L2_LINES_IN.ALL w pliku *papi_set_user_events_mth.c*
    3. zmień rozmiar tablic na 3996
    4. powtarzaj kroki opisane dla mnożenia macierz-wektor, uwzględniając wiedzę nabytą w poprzednich laboratoriach na temat mnożenia macierz-macierz
        - intensywność arytmetyczna powinna być obliczana na podstawie transferów z DRAM uzyskiwanych na podstawie eksperymentów z użyciem *valgrind* i interfejsu PAPI (wyniki zwracane przez *perf* wydają się być zaniżone)
            - uwaga: czas obliczeń *valgrind* może być bardzo długi
    5. w pierwszej kolejności uwzględnij algorytm wersji 2, następnie algorytm z *cache blocking* – w dowolnej wersji 3, 4, 5
        - **dla każdej z wersji intensywność arytmetyczna może być inna**

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

19. Dokonaj badania dla wersji równoległej mnożenia macierz-macierz
    1. należy odkomentować dyrektywy OpenMP w pliku *mat_mul_par_papi.c*

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

20. Uwzględnij w badaniu algorytmu mnożenia macierz-macierz transfery danych pomiędzy rejestrami i pamięcią podręczną L1
    1. objętość transferu oblicz tak jak w poprzednim laboratorium i na jej podstawie uzyskaj intensywność arytmetyczną dla pamięci L1
    2. nanieś prostą związaną z przepustowością L1 na diagram *roofline* (w wersji diagramu z obiema osiami ze skalą logarytmiczną będzie ona ponad linią przepustowości DRAM)
    3. wstaw linię intensywności arytmetycznej mnożenia macierz-macierz na diagramie i zaznacz punkt wydajności uzyskanej eksperymentalnie

---------- 5.0 ----------

Dalsze kroki:

21. Jeśli wyniki wydajnościowe dla najlepszej dotychczasowej wersji kodu odbiegają znacząco od "linii dachu" na diagramie *roofline* zastanów się nad możliwymi przyczynami tego zjawiska oraz możliwymi dalszymi optymalizacjami kodu, które pozwoliłyby na zwiększenie wydajności. Spróbuj przeprowadzić te optymalizacje, zanotuj otrzymane wyniki i wyciągnij odpowiednie wnioski.

Sprawozdanie:

1. Zrealizowane kroki, najważniejsze fragmenty modyfikowanego kodu (a także ewentualnie uzyskiwanego asemblera), spostrzeżenia z analizy kodu (i ewentualnie odpowiadającego kodu asemblera), tabele, wykresy, opisy, wnioski – zgodnie z regulaminem laboratoriów