Analiza i modelowanie wydajności obliczeń

Lab 8. Ręczna optymalizacja kodu

Cel: Przetrenowanie stosowania technik ręcznej optymalizacji kodu, porównanie wyników z efektami pracy optymalizującego kompilatora.

Kroki:

1. Rozpakuj paczkę *mat_vec_optimization.tgz* w nowym katalogu np. *lab_08*
2. Prosty program zawarty w paczce testuje realizację iloczynu macierz-wektor dla standardowej macierzy przechowywanej wierszami. Kod procedury będącej przedmiotem optymalizacji znajduje się w pliku *mat_vec.c*
3. Pierwszym krokiem ćwiczenia jest wybór kompilatora (*gcc* lub *icc)*. Kod produkowany przez każdy kompilator jest inny, inne są też czasy wykonania. Ćwiczenie obejmuje analizę kodu asemblera – kompilator *gcc* produkuje prostszy kod asemblera, kompilator *icc* umieszcza w asemblerze więcej informacji, np. numer linii kodu źródłowego, której odpowiada rozkaz asemblera, ale produkuje kilka wersji dla tego samego fragmentu kodu źródłowego – zawsze należy wybierać wersję najbardziej zoptymalizowaną, mającą najmniej operacji i dostępów do pamięci). Część obowiązkowa ćwiczenia dotyczy jednego wybranego kompilatora. Badania dla drugiego stanowią treść kroków dodatkowych.
4. Uruchom program (poleceniem *make*). Zaobserwuj różnice w czasie wykonania dla dwóch wersji kodu – pierwszej, w której zastosowana jest opcja kompilacji -O0 (wymuszony brak optymalizacji) i drugiej, w której ustawiona jest opcja -O3, oznaczająca relatywnie wysoką, agresywną optymalizację (dokładny opis np. w *man gcc*)
5. Przeanalizuj kod assemblera uzyskany dla każdej z opcji (kompilacja np. `icc -S -O3 mat_vec.c -o mat_vec_opt.s` , `gcc -S -O0 mat_vec.c -o mat_vec_no_opt.s` , itp. itd.). Najważniejsze jest znalezienie bloków podstawowych w asemblerze realizujących kod wewnętrznej pętli algorytmu i rozszyfrowanie jakim operacjom kodu źródłowego odpowiadają kolejne rozkazy w asemblerze. *(Uwaga: założeniem ćwiczenia jest stosowanie wyłącznie kodu skalarnego, bez wektoryzacji, jeśli kompilator stosuje wektoryzację należy uniemożliwić mu to za pomocą stosownych opcji, np. -no-vec dla icc)*
   1. zamieść w sprawozdaniu kod asemblera dla najbardziej wewnętrznej pętli w funkcji *mat_vec*

      [należy zwrócić uwagę na możliwą optymalizację kompilatora rozwinięcia najbardziej wewnętrznej pętli – wtedy blok podstawowy będzie zawierał kod dla kilku iteracji pętli]
   2. zaobserwuj ile dostępów do pamięci znajduje się w kodzie zoptymalizowanym, a ile w kodzie niezoptymalizowanym. Powiąż obserwację z postacią kodu źródłowego, dla którego, w dosłownym tłumaczeniu na język asemblera, każdemu pojawieniu się zmiennej (także każdego elementu tablicy) w instrukcji kodu powinien towarzyszyć dostęp do pamięci – odczyt lub zapis

      [operacja lea *nie jest związana z dostępem do pamięci, polega tylko na wykonaniu operacji arytmetycznych typowych dla obliczania adresu w złożonych trybach adresowania]*
6. Utwórz tabele według poniższego wzoru (przed wypełnianiem przeczytaj pp.7 i 8):

Opcje i zastosowane techniki optymalizacji	Czas działania [s]	Wydajność minimalna [GB/s]	% wydajności teoretycznej	Wydajność [GFLOP/s]	% wydajności teoretycznej
kod oryginalny, O0					

kod oryginalny, O3					
klasyczne optymalizacje (CSE, LICM, IVS), O0					
klasyczne optymalizacje (CSE, LICM, IVS), O3					
jw. + loop unrolling, O0					
jw. + loop unrolling, O3					
jw. + register blocking, O0					
jw. + register blocking, O3					
full optimization (jw + własne techniki), O0					
full optimization (jw + własne techniki), O3					

7. Dokonaj sukcesywnej ręcznej optymalizacji kodu, zgodnie z treścią kolejnych linii w tabelce.

   [zadanie optymalizacji *cache blocking* może być zadaniem dodatkowym w ramach technik własnych]

   Wypełnij kolejne wiersze tabeli dla kolejno przeprowadzanych optymalizacji. Standardowe optymalizacje oznaczają: CSE – *common subexpression elimination*, LICM – *loop invariant code motion*, IVS – *induction variable simplification* . Można zastosować także inne (np. zamiana *for* na *while*, odliczanie indeksu pętli do tyłu, żeby porównywać go z zerem itd. itp.). Nie należy tylko przedwcześnie umieszczać opcji zawartych w znajdujących się poniżej wierszach tabeli. W ostatnich dwóch wierszach należy umieścić najlepszy uzyskany przez siebie wariant (dokładnie opisany w sprawozdaniu) – wariant z najkrótszym czasem wykonania (jeśli warianty dla O0 i O3 są różne dokładny opis ich obu powinien znaleźć się w sprawozdaniu).

   Do obliczenia wydajności w GFLOP/s przyjmij liczbę operacji zmiennoprzecinkowych w sposób oczywisty wynikającą z kodu źródłowego: 2*WYMIAR*WYMIAR=2*10^8 . Do obliczenia minimalnej wydajności w GB/s (**przepustowości pamięci DRAM**) załóż, że do obliczeń pobierana jest macierz a i wektor x – jednokrotnie, a także jednokrotnie zapisywany wektor y. Dla takiego założenia liczba dostępów wynosi WYMIAR*(WYMIAR+2), co po pomnożeniu przez rozmiar danych daje liczbę bajtów przesłanych z i do pamięci DRAM podczas wykonania funkcji. Obliczenia prowadzone są w podwójnej precyzji, więc rozmiar zmiennej to 8 B.

   % wydajności teoretycznej to ważna miara, która mówi jak daleko od maksimum dostępnego na danym sprzęcie znajduje się wydajność wykonywanych obliczeń. Jeśli celem optymalizacji jest pełne wykorzystanie możliwości sprzętu, to uzyskanie wydajności obliczeń w granicach 80-90% wydajności teoretycznej (lub wydajności benchmarku, co do którego zakładamy, że dobrze wskazuje maksymalne wydajności sprzętu) jest bardzo dobrym wynikiem i znakiem, że dalsze optymalizacje nie przyniosą już znaczącego skrócenia czasu obliczeń

   - z drugiej strony **uzyskanie wydajności powyżej 100% wydajności teoretycznej (maksymalnej) oznacza błąd –** albo błąd pomiaru aktualnej wydajności (lub przyjęcie założeń, które nie w pełni odpowiadają rzeczywistej sytuacji), albo błąd oszacowania wydajności teoretycznie maksymalnej (lub błąd doboru benchmarku do określania wydajności maksymalnej)
   - % wydajności teoretycznej dla wydajności w GFLOP/s oblicz porównując ją z teoretyczną wydajnością **obliczeń jednowątkowych** (ewentualnie z najlepszym

wynikiem uzyskanym w ramach poprzednich laboratoriów (np. w *lab_03* kodem *latency_throughput_vector_flops*).

- % wydajności teoretycznej dla **wydajności pamięci DRAM** w GB/s oblicz porównując ją z najlepszym wynikiem uzyskanym w ramach poprzednich laboratoriów (np. *lab_05* kodem *multiple_arrays_vector* lub *multiple_arrays_scalar*).
- podaj w sprawozdaniu jakie wartości zostały użyte do obliczenia procentu wydajności maksymalnej

[Oszacowanie wydajności w GB/s jest tylko pewną wskazówką dotyczącą rzeczywistej przepustowości. Dosłowne tłumaczenie oryginalnej wersji kodu prowadzi do liczby dostępów 4*WYMIAR*WYMIAR (dwa dostępy do elementów y i po jednym do elementów a i x w każdej iteracji). Rozmaite optymalizacje (lub ich wymuszony brak) powodują z jednej strony zmianę liczby dostępów do pamięci w kodzie asemblera, a z drugiej strony zmianę organizacji dostępów do pamięci, co prowadzi do modyfikacji korzystania z pamięci podręcznych. Oszacowanie rzeczywistych transferów staje się zagadnieniem bardziej złożonym, wymagającym m.in. uwzględnienia rozmiaru i sposobu funkcjonowania pamięci podręcznych. Będzie to tematem kolejnego laboratorium.]

8. Dla każdego z otrzymanych w ramach optymalizacji wariantów funkcji *mat_vec* (łącznie z pierwotnym kodem), umieść w sprawozdaniu:
   - opis optymalizacji kodu
   - postać kodu źródłowego (funkcja *mat_vec*)
9. Wykonaj wykres (np. słupkowy) wydajności w GFLOP/s dla kolejnych wariantów optymalizacji (obie opcje O0 i O3 obok siebie)

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

10. Zadanie dodatkowe (4.0)

    Dla wybranych 2 przypadków – najlepszego i najgorszego – otrzymanych w ramach optymalizacji wariantów funkcji *mat_vec* (łącznie z pierwotnym kodem), umieść w sprawozdaniu postać kodu asemblera (dla wersji O0 i O3 – tylko najbardziej zoptymalizowany kod najbardziej wewnętrznej pętli) po optymalizacji. Zaobserwuj czy i jak optymalizacje wpływają na kod asemblera (dla wersji O0 i O3). Powiąż wydajność kodu z postacią asemblera. Swoje obserwacje umieść w sprawozdaniu.

Dalsze kroki (5.0):

11. Wykonaj wszystkie kroki od 4 do 10 dla drugiego z kompilatorów.

Sprawozdanie:

1. Zrealizowane kroki, spostrzeżenia z analizy kodu (a także ewentualnie odpowiadającego kodu asemblera), tabele, wykresy, opisy, wnioski – zgodnie ze wskazówkami w poszczególnych punktach instrukcji i regulaminem laboratoriów