Optymalizacja Kodu Na Różne Architektury
Faktoryzacja Cholesky'ego
raport

Iwo Szczepaniak

20 maja 2024

1 Procesor użyty do wykonania zadania

1.1 Parametry procesora

Platform: Desktop Laptop	**Product Family:** AMD Ryzen™ Processors	**Product Line:** AMD Ryzen™ 7 Mobile Processors with Radeon™ Graphics
# of CPU Cores: 8	**# of Threads:** 16	**Max. Boost Clock:** Up to 4.2GHz
Base Clock: 2.9GHz	**L2 Cache:** 4MB	**L3 Cache:** 8MB
Default TDP: 45W	**AMD Configurable TDP (cTDP):** 35-54W	**Processor Technology for CPU Cores:** TSMC 7nm FinFET
CPU Socket: FP6	**Max. Operating Temperature (Tjmax):** 105°C	**Launch Date:** 1/6/2020

Rysunek 1: Parametry procesora

1.2 Liczba GFLOPS

Informacje o procesorze znajdują się w pliku /proc/cpuinfo oraz w specyfikacji procesora [1]. Dodatkowym źródłem informacji o naszym procesorze mogą być również inne materiały - na przykład artykuł z wikipedii pozwalający ustalić liczbę FLOPs [2].

W przypadku CPU wykorzystanego do obliczeń, istotne parametry to:
FLOPs_per_cycle = 8
GHz_clock_rate = 4,2

Tak więc teoretyczne GFLOPS to:

$$GFLOPS = FLOPs\_per\_cycle * n\_processors * GHz\_clock\_rate =$$

$$= 8 * 1 * 4,2 = 33,6$$

W celu optymalizacji zastosowano flagi: -O2 -msse3 -march=znver2 [3].

2 Opis poszczególnych optymalizacji

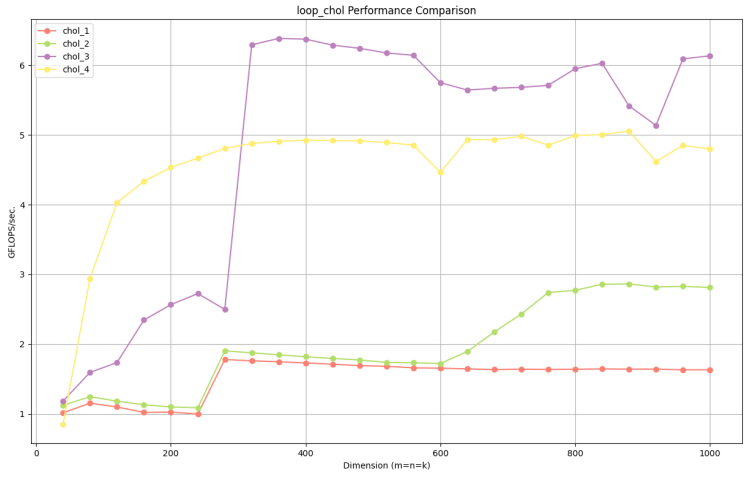
Rozwijanie pętli:

- chol_1: Podstawowa wersja faktoryzacji Cholesky'ego
- chol_2: Iteratory oraz tymczasowa zmienna w rejestrach
- chol_3: Rozwinięcie pętli do bloku 8 wykonań na raz
- chol_4: Rozwinięcie pętli do bloku 16 wykonań na raz

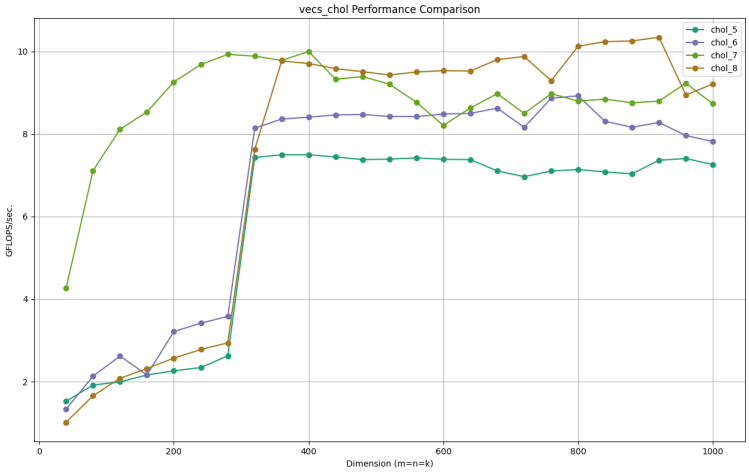
Użycie operacji wektorowych:

- chol_5: Rozwinięcie pętli do bloku 8 wykonań na raz oraz użycie operacji wektorowych 128 bitowych
- chol_6: Rozwinięcie pętli do bloku 16 wykonań na raz oraz użycie operacji wektorowych 128 bitowych
- chol_7: Rozwinięcie pętli do bloku 8 wykonań na raz oraz użycie operacji wektorowych 256 bitowych.
- chol_8: Rozwinięcie pętli do bloku 16 wykonań na raz oraz użycie operacji wektorowych 256 bitowych.

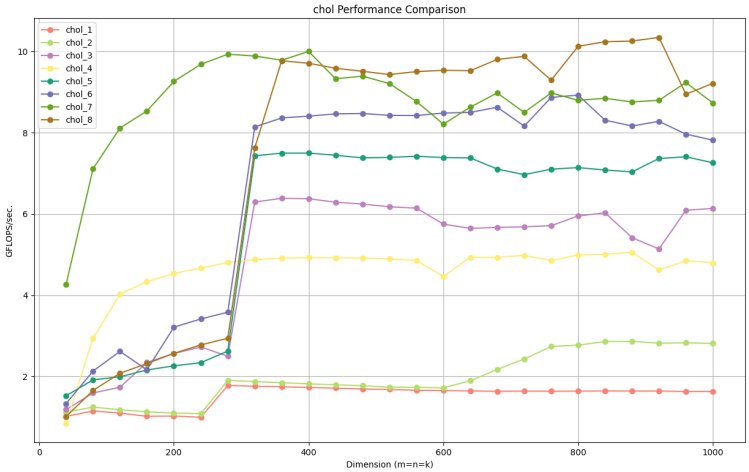
3 Wyniki optymalizacji



Rysunek 2: Wykres porównujący wyniki rozwinięcia pętli



Rysunek 3: Wykres porównujący wyniki zastosowania operacji wektorowych



Rysunek 4: Zbiorczy wykres porównujący wyniki wszystkich optymalizacji

4 Podsumowanie

Pierwsza optymalizacja, polegająca na zastosowaniu słowa kluczowego register, okazała się mało efektywna - dała naprawdę nieznaczny wzrost GFLOPS i to jedynie dla dużych macierzy. Następne 2 optymalizacje - rozwinięcia pętli do 8 i 16 - przyniosły znaczny wzrost wydajności(około 5 krotny). Spory przeskok wydajności widać dla użycia operacji wektorowych, tutaj wynik zbliża sie do 10 GFLOPS, co stanowi około dwukrotnie lepszy wynik niż samo rozwinięcie pętli.

Najlepszy wynik FLOPS wynoszący 10.34238 jest znacznie niższy od przewidywanego teoretycznie 33,6. To znaczy, że uzyskaliśmy 30,78% wydajności trybu turbo procesora. Co zdumiewające, efekty dla optymalizacji nie spadają dla większych rozmiarów tablic, tak jak było to w poprzednim ćwiczeniu. Jednak podobnie jak wtedy, dla małych rozmiarów tablic korzyści wynikające z modyfikacji pozostają parokrotnie mniejsze niż dla rozmiarów większych niż 300.

Literatura

[1] Specyfikacja procesora AMD Ryzen 7 4800H, `https://www.amd.com/en/support/apu/amd-ryzen-processors/amd-ryzen-7-mobile-processors-radeon-graphics/amd-ryzen-7-4800h`

[2] Artykuł na Wikipedii o FLOPS, `https://en.wikipedia.org/wiki/FLOPS#Floating-point_operations_per_clock_cycle_for_various_processors`

[3] Strona Gentoo Wiki o procesorach AMD, `https://wiki.gentoo.org/wiki/Ryzen`