Optymalizacja Kodu Na Różne Architektury

Zadanie 1

How To Optimize Gemm

Przemysław Roman

14.04.2023

1 Procesor

1.1 Parametry

Parametr	Wartość
Producent	Intel
Model	i7-4810MQ
Mikroarchitektura	Haswell
Rdzenie	4
Wątki	8
Częstotliwość bazowa	2.80 GHz
Częstotliwość turbo	3.80 GHz
Cache	6144 KB
GFLOPS	179.2
GFLOPS/rdzeń	44.8

1.2 Wyznaczenie GFLOPS/rdzeń

1.2.1 Korzystając z wartości tablicowej

$$\frac{GFLOPS}{Rdzenie} = \frac{179.2}{4} = 44.8$$

Wartości GFLOPS dla procesorów firmy Intel można znaleźć pod tym linkiem.

1.2.2 Korzystając ze wzoru podanym w PlotAll.m

$$nflops\_per\_cycle * nprocessors * GHz\_of\_processor = 16 * 1 * 2.8 = 44.8$$

Najpierw należy sprawdzić mikroarchitekturę naszego procesora:

```
cat /sys/devices/cpu/caps/pmu_name
```

Następnie sprawdzamy wartość $FP64 \equiv nflops\_per\_cycle$ na Wikipedii.

1.2.3 Podsumowanie

Oba sposoby obliczania doprowadziły nas do tego samego wyniku - 44.8.

2 Optymalizacje

2.1 Opisane w "How To Optimize Gemm"

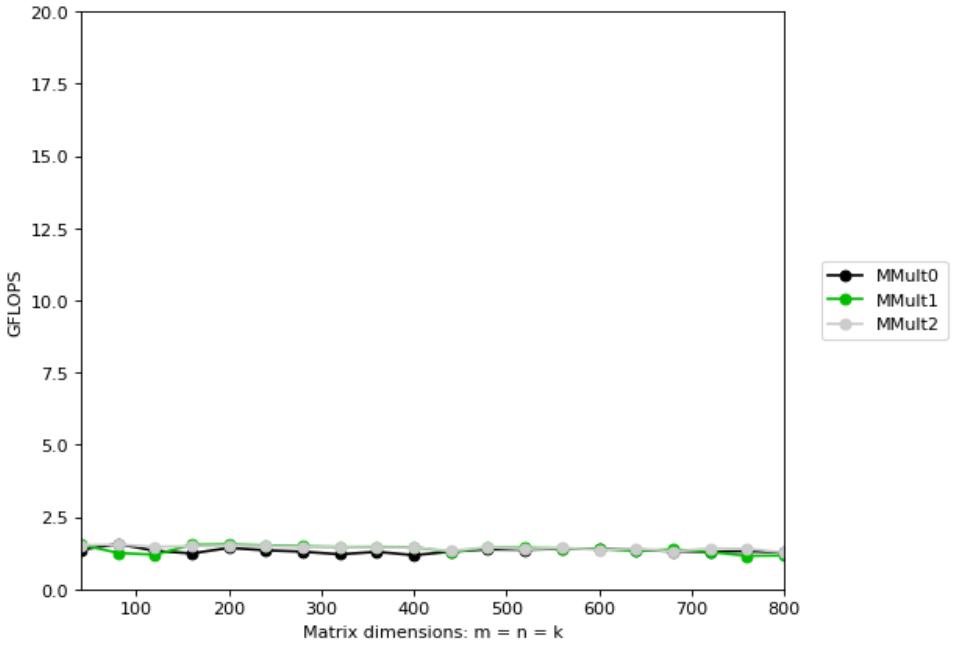
1. MMult1 - Dodanie makra X oraz funkcji AddDot
2. MMult2 - Krok co 4 dla j
3. MMult_Ax4_3 - Przeniesienie wywołań AddDot do funkcji AddDotAx4
4. MMult_Ax4_4 - Rozwinięcie AddDot w miejscach wywoływania
5. MMult_Ax4_5 - Jedna pętla dla rozwinięć z poprzedniej optymalizacji
6. MMult_Ax4_6 - Rejestry dla A i C
7. MMult_Ax4_7 - Wskaźniki do B

8. MMult_Ax4_8
   (a) A = 1 - Zmiana kroku pętli z optymalizacji 5. na 4
   (b) A = 4 - Rejestry dla B
9. MMult_Ax4_9
   (a) A = 1 - Zmiana kroku wskaźników do B na 4
   (b) A = 4 - Zmiana kolejności wykonywanych operacji (grupujemy rzędy po 2 a następnie w grupach przechodzimy kolumnami, dotychczas przechodziliśmy rzędami bez żadnego grupowania)
10. MMult_4x4_10 - Wektory __m128d
11. MMult_4x4_11 - Podział na bloki, dodanie funkcji InnerKernel
12. MMult_4x4_12 - Dodanie funkcji PackMatrixA
13. MMult_4x4_13 - Uproszczenie odwołania do elementów z A
14. MMult_4x4_14 - Dodanie funkcji PackMatrixB, zmiana kroku na 4 w PackMatrixA
15. MMult_4x4_15 - Warunkowe wykonanie PackMatrixB

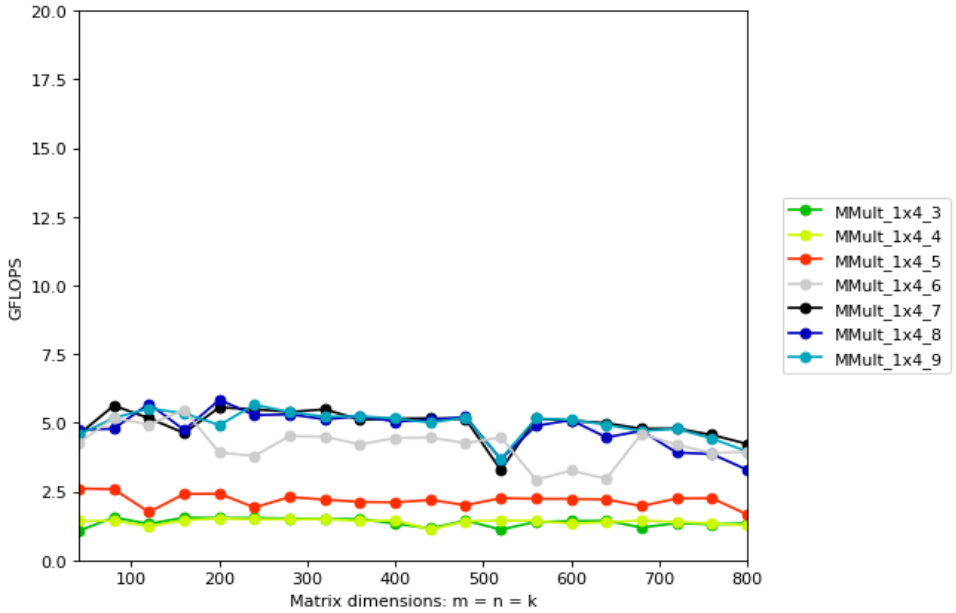
2.2 Dostosowane do procesora

1. Uruchomienie z flagą optymalizującą dla danej mikroarchitektury: $-march = haswell$

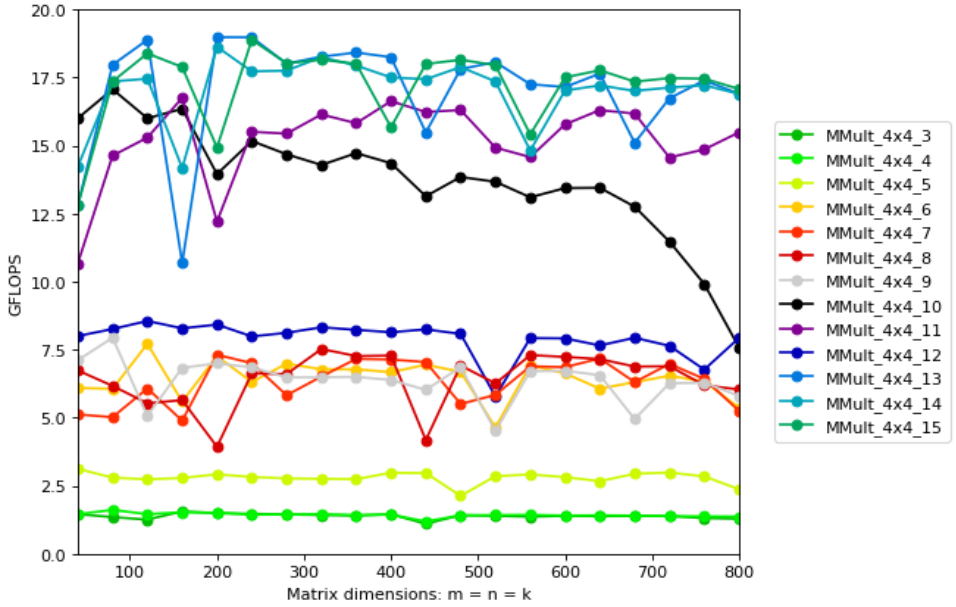
3 Wyniki



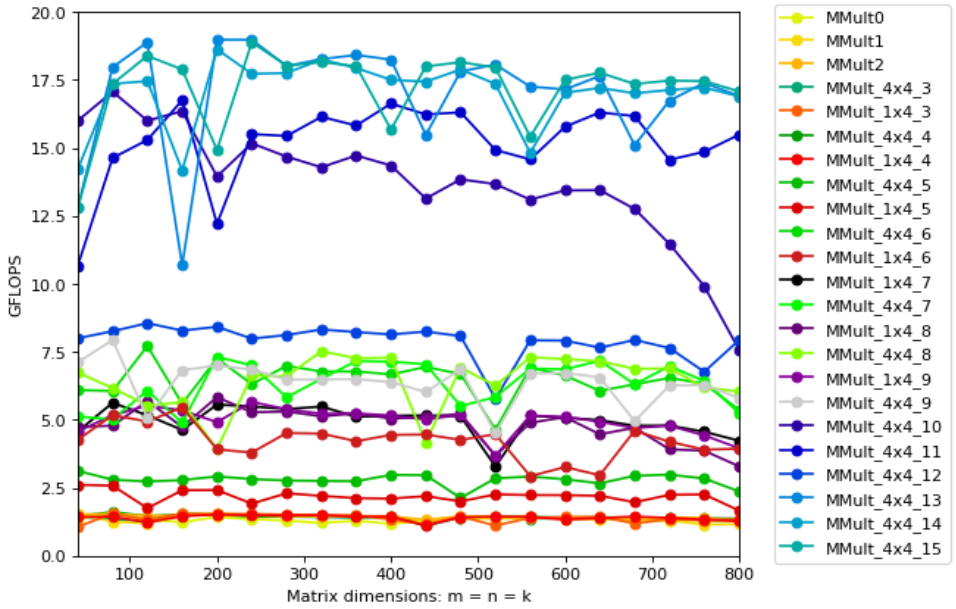
Rysunek 1: Wspólne optymalizacje



Rysunek 2: Optymalizacje 1x4



Rysunek 3: Optymalizacje 4x4



Rysunek 4: Wszystkie optymalizacje

4 Podsumowanie

1. Najwydajniejszy wynik 18.4 GFLOPS jest kiepski w porównaniu z teoretycznym 44.8 GFLOPS (41% maksymalnej wydajności)
2. Najwydajniejsze wersje programu:
   - MMult_4x4_13
   - MMult_4x4_15
   - MMult_4x4_14
3. Największe zyski wydajności wprowadziły:
   - MMult_1x4_6
   - MMult_4x4_10
   - MMult_4x4_13
4. Największe spadki wydajności:
   - MMult_4x4_10 dla macierzy o dużych rozmiarach
   - MMult_4x4_13-14 dla n = 160
   - MMult_4x4_11,15 dla n = 200