Wektoryzacja metod Rungego-Kutty i testowanie na równaniu Maltuzjańskim

1. Opis problemu

Przedmiotem projektu była implementacja metod Rungego-Kutty w języku C++ w wersji zoptymalizowanej pod koprocesory wektorowe SSE (ang. Streaming SIMD Extension) oraz przetestowanie jej na równaniu Maltuzjańskim[1]. Równanie to jest uproszczonym modelem dynamiki zmian populacji.

$$\frac{dP}{dt} = kP(t) = (u - z)P(t)$$

gdzie:

- t – czas,
- P(t) – wielkość populacji w chwili t,
- u – współczynnik urodzeń,
- z – współczynnik zgonów.

Implementacje metod RK z zajęć przepisano z wykorzystaniem operacji na wektorach i wykorzystano jako metody klasy wektora stanu, które uaktualniają go zgodnie z danym krokiem czasowym. W optymalizacji przyjęto dwa podejścia:

- optymalizacja klasy wektora,
- optymalizacja klasy wektora stanu.

2. Wyniki

Do skompilowania programów wykorzystano kompilator g++. Poniżej znajduje się porównanie działania programu, w którym równania Rungego-Kutty operują na skalarach i wersji zwektoryzowanch z opcjami optymalizacyjnymi -O2 oraz -funroll-all-loops.

Na początku jednak warto poruszyć kwestię dokładności wyników skalarnych i zwektoryzowanych. W [2] pokazano konwencje wywołań funkcji we współczesnych 64-bitowych systemach operacyjnych bazujących na architekturze x86. Liczby zmiennopozycyjne są tam przekazywane przez rejestry jednostki SSE. ABI (Application Binary Interface) traktuje więc skalary jak wektory z jednym uzupełnionym polem i przeprowadza na nich takie same operacje.

Tabela poniżej przedstawia porównanie czasów wykonywania algorytmów Rungego-Kutty w wersjach skalarnej i zoptymalizowanej na oba sposoby. Zostały one przetestowane na równaniu Maltuzjańskim dla 10000000 kroków symulacji.

Tabela 1. Porównanie czasów wykonywania algorytmów Rungego-Kutty dla równania Maltuzjańskiego

algorytm	w. skalarna	w. wektorowa - wektor	w. wektorowa - algorytm
RK1	real 3m37,523s user 3m12,477s sys 0m19,457s	real 3m37,036s user 3m9,977s sys 0m19,677s	real 3m38,119s user 3m12,927s sys 0m19,343s
RK2	real 3m42,393s user 3m14,273s sys 0m20,933s	real 3m30,634s user 3m3,967s sys 0m19,633s	real 3m35,181s user 3m9,433s sys 0m19,747s
RK4	real 3m39,257s user 3m14,533s sys 0m19,783s	real 3m28,998s user 3m7,130s sys 0m18,357s	real 3m28,525s user 3m4,330s sys 0m19,140s

3. Wnioski

Zebrane wyniki pokazują, że skalarne wersje algorytmów są wolniejsze niż zoptymalizowane. Co więcej, oba podejścia do optymalizacji dają porównywalne w czasie wyniki. Wynika to z faktu, że zaimplementowany problem opierał się na rozwiązywaniu równania różniczkowego pierwszego rzędu w przestrzeni jednowymiarowej, co daje jednowymiarowy wektor stanu.

4. Odniesienia

[1] K. Matyjasek, „Wybrane zastosowania równań różniczkowych zwyczajnych w biologii”

[2] https://en.wikipedia.org/wiki/X86_calling_conventions#List_of_x86_calling_conventions