

Zrównoległone całkowanie metodą Monte Carlo

21 listopada 2024

1 Wstęp

Obliczanie całek jest kluczowym zagadnieniem w matematyce, wykorzystywanym w wielu dziedzinach nauki, inżynierii i finansów. Tradycyjne metody numeryczne, takie jak reguła trapezów czy metoda Simpsona, są skuteczne w wielu przypadkach, ale nie zawsze są w stanie sprostać złożonym problemom, szczególnie w przestrzeniach o wyższych wymiarach. W takich przypadkach przydatne stają się metody probabilistyczne, a jedną z najbardziej popularnych technik jest metoda Monte Carlo.

Metoda Monte Carlo opiera się na wykorzystywaniu losowych prób do przybliżania wartości całek. Chociaż jest to technika prosta w implementacji, jej efektywność obliczeniowa w dużych zbiorach danych może być problematyczna. Aby poprawić wydajność obliczeń, można wykorzystać równoległe przetwarzanie danych, co jest szczególnie efektywne w przypadku dużych problemów numerycznych. W tym kontekście język Java oferuje narzędzia takie jak `Stream` oraz `ParallelStream`, które umożliwiają efektywne i równoległe przetwarzanie danych na wielu rdzeniach procesora.

Celem niniejszego raportu jest przedstawienie procesu implementacji metody Monte Carlo do obliczania całek w języku Java z wykorzystaniem `Stream` oraz `ParallelStream`. W szczególności raport skupi się na porównaniu wydajności obu podejść, badając, w jakim stopniu równoległe przetwarzanie może przyspieszyć obliczenia w porównaniu do tradycyjnych metod sekwencyjnych. Przedstawione będą również przykłady implementacyjne oraz wyniki eksperymentów, które ilustrują zalety i wady każdej z metod.

2 Opis Programu

W celu realizacji projektu utworzono programy `MonteCarloIntegralEstimator` i `MonteCarloParallelIntegralEstimator`. Oba programy przyjmują cztery parametry: ścieżkę do badanej funkcji, początek i koniec zakresu całkowania (dla uproszczenia przyjęto ten sam przedział dla każdej zmiennej), oraz liczbę iteracji metody. Przykładowe wywołanie:

```
java .\MonteCarloIntegralEstimator.java .\Examples\ComplexFunction.java 0 1 100000
```

3 Obliczanie Całek

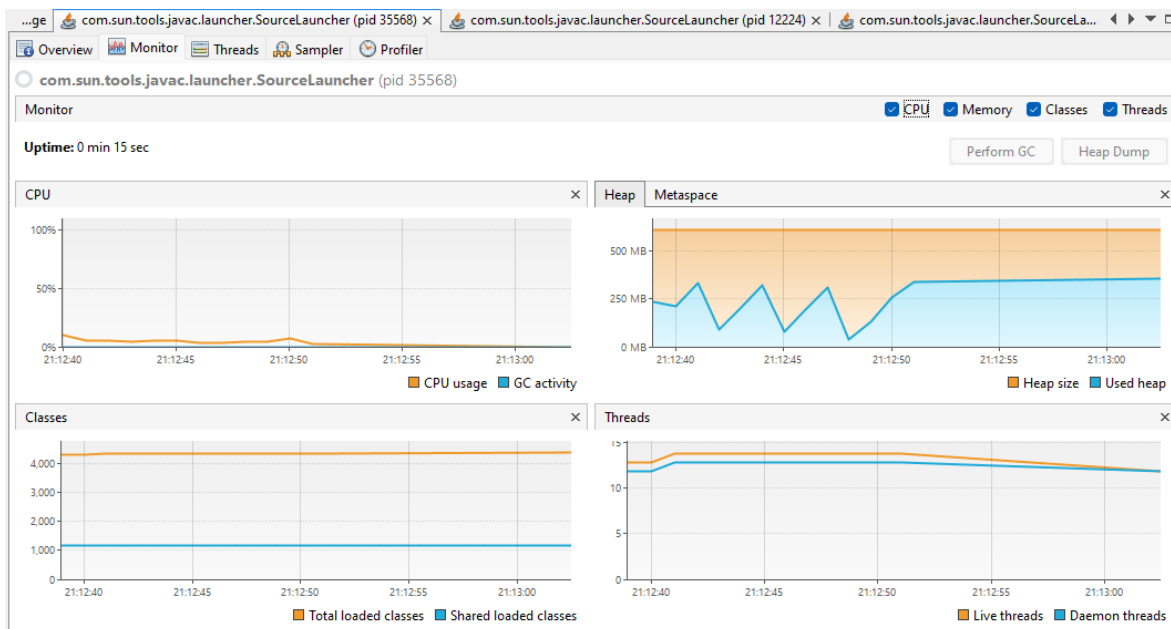
Całki są przybliżane za pomocą wzoru:

$$I = \frac{(b-a)^z}{N} \sum_i^N f(x_1, x_2, \dots, x_z) \quad (1)$$

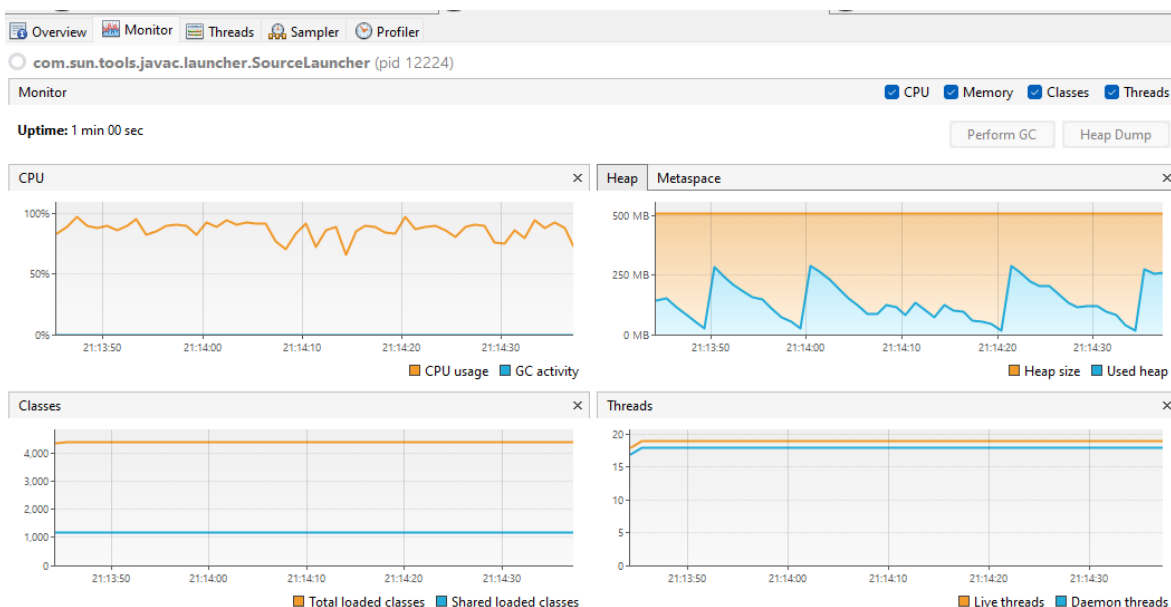
4 Testy rozwiązania

W pierwszej wersji programu postanowiliśmy wykorzystać refleksję do wyliczania wartości funkcji, co okazało się problematyczne przy wykorzystaniu `ParallelStream`. Okazało się, że przy przyjętej architekturze (wykorzystaniu statycznych funkcji wczytywanych przez refleksję) zrównoleglone wątki blokują się nawzajem, spowalniając działanie programu.

Badania profilera wykazały ponad czterokrotną przewagę `Stream` nad `ParallelStream`. Jest to spowodowane niemożliwością zrównoleglenia wywołań statycznych metod, co po czasie wydaje się być dość oczywiste.



Rysunek 1: Wyniki testów bez zrównoleglenia



Rysunek 2: Wyniki testów ze zrównolegleniem

5 Wnioski

Przy przyjętej architekturze zrównoleglenie nie wpłynęło pozytywnie na czas wykonywania programu. Przeprowadzona analiza wskazuje, że próba wykorzystania `ParallelStream` do zrównoleglenia obliczeń w tym przypadku nie przyniosła oczekiwanych korzyści, a wręcz spowodowała pewne spowolnienia w porównaniu z podejściem sekwencyjnym. Zasadniczym powodem tego zjawiska było wykorzystanie refleksji do wywoływania statycznych metod, co jest procesem kosztownym i trudnym do zrównoleglenia w sposób efektywny.

Po przeanalizowaniu wyników z profilerem, stało się jasne, że w kontekście naszej architektury programowej, zrównoleglone wątki blokują się nawzajem, oczekując na dostęp do statycznych metod. Takie blokady, szczególnie w przypadku obliczeń, które są wykonywane na dużych zbiorach danych, mogą

przewodzą do spadku wydajności. W praktyce oznacza to, że wątki nie mogą równocześnie wykonywać obliczeń, ponieważ każda próba wywołania statycznej metody wymaga synchronizacji, co zmniejsza efektywność równoległości.

Z kolei w przypadku standardowego Stream, bez równoległego przetwarzania, obliczenia wykonywane są sekwencyjnie, co zapewnia lepszą kontrolę nad kolejnością i synchronizacją, a także unika problemów związanych z blokowaniem wątków. Choć w przypadku dużych danych równoległe przetwarzanie powinno przynieść korzyści, to jednak w naszej konkretnej implementacji, tradycyjny Stream okazał się bardziej wydajny.

Wnioski płynące z tego eksperymentu sugerują, że nie każda aplikacja, nawet jeżeli wykorzystuje równoległe przetwarzanie danych, będzie zyskiwała na wydajności. W szczególności, zastosowanie refleksji w kontekście równoległych wątków może prowadzić do nieoczekiwanych problemów z wydajnością.

Podsumowując, nasze doświadczenia pokazują, że równoleglenie obliczeń może być skuteczne tylko wtedy, gdy odpowiednia architektura programu wspiera efektywne zarządzanie wątkami i minimalizowanie problemów związanych z synchronizacją. W przypadku bardziej zaawansowanych aplikacji numerycznych, zaleca się ostrożność przy stosowaniu refleksji i zwrócenie uwagi na aspekty takie jak współbieżność i zarządzanie zasobami w systemie wielordzeniowym.