Obliczenia inżynierskie w chmurze

KWADRATURA NUMERYCZNA FUNKCJI JEDNEJ ZMIENNEJ

25 stycznia 2019

Michał Rosa
Numer albumu: 276394
Politechnika Warszawska
Wydział Mechaniczny Energetyki i Lotnictwa

1 Wstęp

Tematem projektu jest program służący do obliczania całki oznaczonej dowolnej funkcji rzeczywistej jednej zmiennej w zadanych przedziałach. Projekt ten jest podsumowaniem serii wykładów poświęconych tematyce obliczeń inżynierskich w chmurze.

2 Założenia projektu

2.1 Funkcjonalność

Głównym wymaganiem stawianym przed programem jest możliwość przeprowadzenia obliczeń z wykorzystaniem mechanizmów wielowątkowości. Program będzie uruchamiany na wirtualnej maszynie obliczeniowej, której usługe dostarcza przedsiębiorstwo *Amazon*.

Funkcja, dla której mają zostać przeprowadzone obliczenia, będzie wprowadzana przez użytkownika bezpośrednio poprzez terminal. Funkcja musi mieć postać funkcji rzeczywistej jednej zmiennej. Użytkownik decyduje również o kroku metody całkowania oraz wprowadza dolną i górną granicę.

Ważną funkcjonalnością jest również możliwość wprowadzenia liczby wątków, które mają zostać utworzone w celu przeprowadzenia obliczeń. Zbadana zostanie zależność szybkości obliczeń od liczby utworzonych wątków.

2.2 Środowisko

Ze względu na obliczeniowy charakter zagadnienia, implemetacja projektu zostanie przeprowadzona w środowisku Python. Modułami, które zostaną wykorzystane podczas pracy, będą moduły *math* oraz *multiprocessing*.

Moduł *math* składa się głównie z niewielkich funkcji opakowujących funkcje ze standardowej biblioteki matematycznej języka C na używanej platformie. W standardzie języka C zachowanie niektórych funkcji nie zostało szczegółowo opisane, w związku z czym sposób informowania o pojawiających się błędach w funkcjach matematycznych języka Python pochodzi z implementacji w języku C.

Biblioteka *multiprocessing* dostarcza mechanizmów tworzenia procesów wykorzystując API podobnego do tego używanego w module *thread*. Oferuje ona funkcjonalność przetwarzania współbierznego zarówno lokalnie jak i zdalnie, pozwalając na obchodzenie niektórych blokad dzięki wykorzystaniu podprocesów zamiast oddzielnych wątkó. Z uwagi na to, moduł pozwala programiście na pełniejszą kontrole i wykorzystanie dostępnych zasobów komputera. Moduł jest kompatybilny zarówno z systemami Linux jak i Windows.

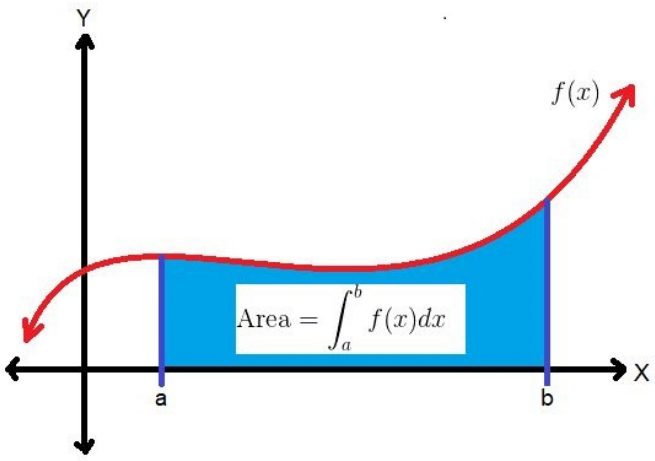
Moduł *multiprocessing* dodatkowo zawiera funkcjonalności nie występujące w klasycznym module *thread*. Najważniejszą z nich są obiekty typu **Pool**, które dostarczają wygodnych metod obliczeń równoległych, pomagając rozdzielać dostępne zasoby.

2.3 GUI

Użytkownik będzie komunikował się z programem za pomocą terminalu. Zostanie poproszony o wprowadzenie funkcji jednej zmiennej, przedziałów i kroku całkowania oraz liczbę watków, które mają zostać utworzone podczas obliczeń. Po wprowadzeniu wszystkich danych wynik zostanie wydrukowany do terminalu.

3 Projekt obliczeniowy

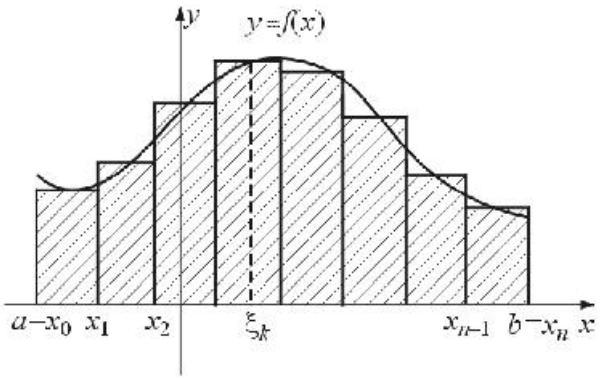
3.1 Obliczanie całki oznaczonej



Rysunek 1: Pole pod wykresem funkcji f(x).

Załóżmy, że chcemy obliczyć całkę z funkcji $f(x)$ w przedziale $< x_p; x_k >$.

Definicja całki oznaczonej Riemana, mówi nam, że wartość całki równa jest sumie pól obszarów pod wykresem krzywej w zadanym przedziale całkowania. Sumę taką możemy obliczyć w przybliżeniu dzieląc obszar całkowania na n równych części. Dla każdej takiej części możemy wyznaczyć prostokąt, który w przybliżeniu będzie odpowiadał polu obszaru pod wykresem krzywej. Jak widać na schemacie poniżej, dla funkcji rosnącej wartości tych przybliżeń będą większe niż w rzeczywistości - nadmiar powoduje część prostokąta znajdująca się ponad wykresem krzywej - dwa pierwsze prostokąty na schemacie. Natomiast dla funkcji malejącej wartości przybliżeń będą mniejsze niż rzeczywiste pole pod wkresem - niedomiar powoduje część pola znajdująca się nad wyznaczonym prostokątem - ostatni prostokąt na schemacie.



Rysunek 2: Niedokładności metody prostokątów.

Jak już wpomnieliśmy przedział całkowania $< x_p; x_k >$ podzielimy na n równych części. Szerokość każdej z nich wynosić będzie zatem:

$$dx = \frac{x_k - x_p}{n} \tag{1}$$

Taka też będzie szerokość każdego prostokąta przybliżającego nam wartość całki w zadanym przedziale. Wysokość każdego z prostokątów wynosić będzie:

$$f(x_i) \quad dla \quad i = 1, 2, ..., n, \quad gdzie \quad x_i = x_p + i * dx \tag{2}$$

Całkę w zadanym przedziale uzyskamy dodając do siebie pola wszystkich tych prostokątów,

wynosić będzie ona zatem:

$$dx * f(x_1) + dx * f(x_2) + ... + dx * f(x_n) = dx * (f(x_1) + f(x_2) + ... + f(x_n)) \quad (3)$$

Warto zauważyć, iż im większa liczba przedziałów n z tym większą dokładnością wyznaczymy interesującą nas całkę. W szzególności, przy $n \to \infty$ otrzymamy dokładną wartość całki.

3.2 Implementacja kodu

3.2.1 Zmienne środowiskowe

- **foo** - jest zmienną typu *string* przechowującą funkcję wprowadzoną przez użytkownika;
- **skok** - jest zmienną typu *float*, która przechowuje wprowadzoną przez użytkownika wartość kroku całkowania;
- **a** - jest zmienna typu *float* przechowującą wartość dolnej granicy całkowania. Wprowadzana przez użytkownika;
- **b** - analogiczna postać zmiennej co zmiennej **a**, przechowuje jednak górną granice całkowania;
- **processes** - jest zmienna typu *int* przechowującą informacje o liczbie wątków tworzonych w czasie obliczeń równoległych. Jak wszystkie wymienione wyżej, ta zmienna również jest wprowadzana przez użytkownika
- **results** - jest zmienna tablicową, która wykorzystywana jest do przechowywania wyników obliczeń.

3.2.2 Funkcje

- **eval_fun** - jest funkcją przetwarzającą funkcje wpisaną przez użytkownika. Określa ona matematyczną postać zgodną z semantyką języka i oblicza jej wartość w określonym punkcie. Jako argumenty przyjmuje zmienną *string* przechowującą postać funkcji oraz wartość *float*, dla której funkcja ma być obliczona.
- **Integrate** - jest główną funkcją obliczeniową programu. Zgodnie z metodą prostokątów przybliżania całki oznaczonej wyznacza kolejne pola pod wykresem funkcji, na

końcu sumując i zwracając wynik. Jako argumenty przyjmuje funkcje, której całka jest obliczana, dolną i górną granice obliczeń oraz krok obliczeń.

- **frange** - jest pomocniczą funkcją, wykorzystywaną przy tworzeniu list zmiennych typu float.

3.2.3 Klasy

- **Pool** - dostarcza wygodnego podejścia do prostych zadań obliczeń równoległych. W skład klasy wchodzą 4 interesujące metody:

  1. Pool.apply
  2. Pool.map
  3. Pool.apply_async
  4. Pool.map_async

  Metody Pool.map oraz Pool.apply blokują główny program do czasu aż wszystkie procesy zostaną zakończone, co jest przydatne, kiedy chcemy uzyskać wyniki w konkretnej kolejności od poszczególnych aplikacji.

  Przeciwną funkcjonalność prezentuje metoda Pool.apply_async, która tworzy procesy w jednej chwili i zwraca wyniki jak tylko zostaną ukończone. Dodatkowo nie ma potrzeby korzystania z metody *get* po wywołaniu metody *xapply_async* w celu uzyskania wyniku z zakończonych procesów. To własnie ta metoda będzie wykorzystywana w projekcie obliczeniowym

Program dzieli przedział obliczeń wynikający z zadanych wartości dolnej i górnej granicy całkowania na mniejsze podprzedziały, których liczba jest równa liczba stworzonych wątków obliczeniowych. Następnie dla każdego z podprzedziałów wywoływana jest metoda pool.apply_async, która wykorzustuje funkcje **Integrate**.

Wyniki są przekazywane do tablicy **results**, która po zakończeniu całkowania jest sumowana, a tak otrzymany wynik jest drukowany na standardowe wyjście.

4 Testowanie

Podczas etapu testów, przygotowanych zostało kilka funkcji, dla których policzono całki oznaczone w granicach $< 0; 10 >$ przy użyciu platformy *Wolframalpha*. Następnie te same funkcje zostały wprowadzone do programu i przeprowadzono te same obliczenia. Wyniki końcowe zgadzały się z wynikami uzyskanymi na platformie *Wolframalpha*. Wraz ze zmniejszaniem kroku całkowania, wydłużał się czas potrzebny na obliczenia. Wynikało to z konieczności wykonania więcej operacji dodawania oraz mnożenia.

Z uwagi na problemy z autoryzacją konta na platformie Amazon, obliczenia na platformie AWS przeprowadzono przy użyciu konta innego studenta. Za pomocą programu *PuTTY* udało połączyć się z maszyną wirtualną. Program został przerzucony przy pomocy programu *WinSCP* do folderu znajdującego sie na platformie AWS.

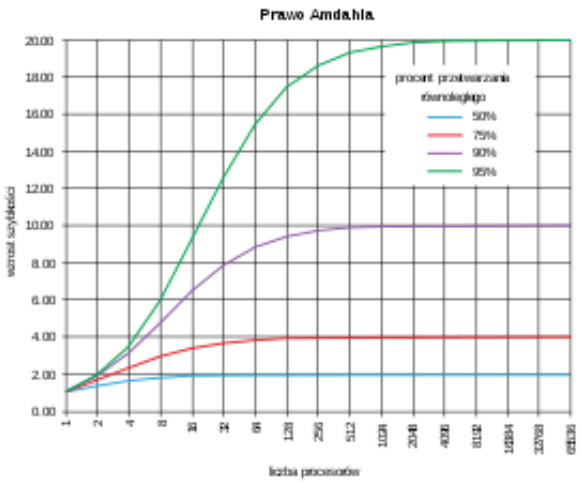
Obliczenia na niej również przebiegły pomyślnie. Mimo zmiany liczby wątków tworzonych w trakcie obliczeń, nie zaobserwowano znaczącego zwiększenia ich wydajności.

5 Podsumowanie

Mimo wykonywania obliczeń na maszynie wirtualnej i wykorzystania modułów pozwalających na przeprowadzaniu równoległych operacji nie zauważono znaczącej poprawy wydajności. Obliczanie całki oznaczonej metodą prostokątów jest nieskomplikowaną operacją, dlatego możliwe, że przeważały takie zjawiska jak:

- Koszty komunikacji procesorów (rosną wraz ze wzrostem ich liczby)
- Niezrównoważenie obciążenia (ang. load imbalance)
- Potrzeba duplikacji obliczeń na różnych procesorach

Według prawa Amdahla zwiększenie szybkości wykonywania się programu przy użyciu wielu procesorów w obliczeniach równoległych jest ograniczane przez czas potrzebny do sekwencyjnego dzielenia programu. Możliwe, że czas podziału zdominował ogólny czas potrzebny na obliczenia całki.



Rysunek 3: Wykres zależności wzrostu wydajności od użytych procesów.