POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Elektronika i Telekomunikacja
SPECJALNOŚĆ: Zastosowanie inżynierii komputerowej w technice

PROJEKT INŻYNIERSKI

Implementacja równoległa wybranego algorytmu globalnej optymalizacji stochastycznej

Parallel implementation of some global stochastic optimization algorithm

AUTOR:
Paweł Sawicz

PROWADZĄCY PRACĘ:
dr hab. inż. Przemysław Śliwiński

OCENA PRACY:

WROCŁAW 2014

Spis treści

Podziękowania…TBC

Rozdział 1 Wstęp

Żyjemy w czasach w których generujemy niezmierne ilości danych, głównym czynnikiem jest powszechny dostęp do Internetu. Dodatkowo coraz bardziej wszelakie instytucje udostępniają swoje zbiory danych do zastosowania publicznego. Niegdyś akwizycje i przetwarzanie danych przeprowadzano w dużych przedsiębiorstwach lub na uczelniach, dzisiaj każdy może pobrać dowolne dane z Internetu chociażby zużycie prądu w Wielkiej Brytanii oraz poszukać hrabstwa w którym jest najmniejsze zużycie prądu. Tak narodziła się nowa nauka w świecie programowania która jest dumnie nazywana „Big Data".

Zespoły „Big Data" zazwyczaj są budowane przez analityków oraz ludzi specjalizujących się w statystyce oraz optymalizacji. Zadaniem takiego zespołu jest dostarczenie odpowiedzi biznesowych na podstawie posiadanych danych.

Dane mogą przedstawiać różne informacje np. wartość wpłaconych pieniędzy dla fundacji charytatywnej w danym czasie, ile użytkownik zebrał pieniędzy od swoich przyjaciół którzy używają portalu Facebook.com. Gdy jesteśmy w posiadaniu tych wielkich zasobów danych może spróbować zamodelować matematyczny model dla danego zachowania się rynku, wtedy posiadamy dużo zmiennych które tworzą nam następne wymiary naszej funkcji i nasz problem staje się coraz bardziej trudniejszy do rozwiązania.

Jedną z pomocnych nauk przy pracowaniu i modelowaniu funkcji opartych na „big data" jest na pewno optymalizacja.

Optymalizacja towarzyszyła człowiekowi od bardzo dawna, prawie zawsze człowiek chciał dany problem zminimalizować lub zmaksymalizować jakąś produkcje danego dobra. Optymalizacja znajdzie zastosowanie w każdej dziedzinie życia poczynając od medycyny, przetwarzania sygnałów, transportu a kończąc na produkcji ciężkiego przemysłu, możemy zoptymalizować wszystko co można opisać jako model matematyczny.

Trzeba także wspomnieć o szybko rozwijającym się przemyśle komputerowym oraz codzienne zwiększanie mocy obliczeniowych które będą nam pomocne przy rozwiązywaniu bardzo złożonych problemów.

Od kilku lat promowany jest taki by który się nazywa „cloud computing" który jest pomocnym narzędziem przy tematyce optymalizacji, są to wielkie centra

obliczeniowe w których za pomocą wirtualizacji systemów operacyjnych jesteśmy w stanie wynająć sobie wirtualną maszynę o znacznej mocy obliczeniowej.

1.1 Cele Projektu

Celem projektu jest implementacje równoległa algorytmu optymalizacji globalnej, metodą stochastyczna. Następnie zbadanie wydajności kilku implementacji takiego algorytmu oraz wyciągniecie wniosków na temat zrównoleglania takich algorytmów. Algorytm będzie napisany w dwóch językach R oraz Haskell. Wybranym algorytmem jest algorytm Sid Yakowitz. Dodatkowo algorytm ten zostanie uruchomiony na wirtualnej maszynie z szesnastoma procesorami w usłudze Microsoft Azure.

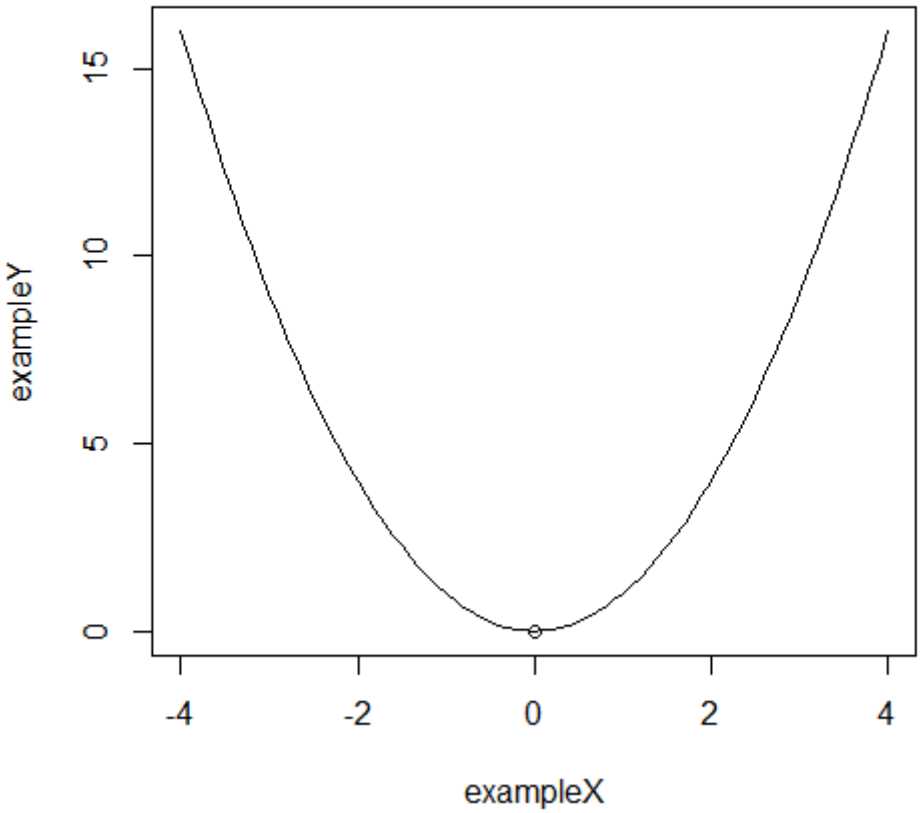
Rozdział 2 Wprowadzenie teoretyczne

Optymalizacja odnosi się do problemu znalezienia maksimum lub minimum zadanej funkcji celu.

Problem ten można opisać w następujący sposób.

$$f : A \mapsto \mathbb{R} \qquad (2.1)$$
$$A \subset \mathbb{R}^n$$

Należy znaleźć taki $x_o \in A$ że dla każdego $x \in A$ zachodzi następująca nierówność $f(x_o) < f(x)$. W przypadku poszukiwania maksimum zmienia się tylko znak nierówności przy funkcji.



Rysunek 1 Przykładowa funkcja z minimum w punkcie 0

Poszukiwanie ekstremum może odbywać się na pewnej przestrzeni która posiada jedno ekstremum wtedy mówimy o optymalizacji lokalnej. Jednak możemy także poszukiwać ekstremum na funkcji posiadającej wiele ekstremów lokalnych wtedy mówi się o optymalizacji globalnej.

Dodatkowo optymalizacje możemy podzielić na dwie zasadnicze grupy, programowanie liniowe oraz programowanie nieliniowe. W przypadku programowania liniowego funkcja celu jak i funkcje ograniczające są sformułowane w postaci liniowej, wtedy rozwiązanie lokalne jest także optimum globalnym. Natomiast w przypadku programowani nieliniowego funkcja celu jak i funkcje ograniczające nie muszą być funkcjami liniowymi, różniczkowalnymi lub nawet ciągłymi, wtedy nasz funkcja celu posiada wiele ekstremów lokalnych.

2.1 Zasady działania algorytmu

Dane (?)

f – funkcja celu (2.2)

$S \subset \mathbb{R}^n$- przestrzeń globalnej optymalizacji

$R_d \subset S$ – podzbiór w czasie n

x_{RS} – losowy punkt generowany na etapie RS z całej S

x_0 – optymalny punkt w czasie n

x_c – kandydat

N – ilość iteracji

$n = 1$ (n – numer iteracji)

$x_0 = 0$ dla $n = 0$

Inicjalizacja

$n = 1$ (n – numer iteracji) (2.3)

$x_0 = 0$ dla $n = 0$

Procedura

1. Jeśli $n \leq N$, pobierz nowy losowy punkt x_{RS}
   1.1. Wykonaj K-W dla x_{RS} oraz R_d i wynik przypisz do x_c
   1.2. Jeśli $f(x_c) < f(x_0)$, wtedy $x_0 = x_c$
2. Jeśli $n > N$, zwróć x_0
3. Zakończ

W następnych punktach są opisane szczegółowo Random Search oraz Kiefer – Wolfowitz.

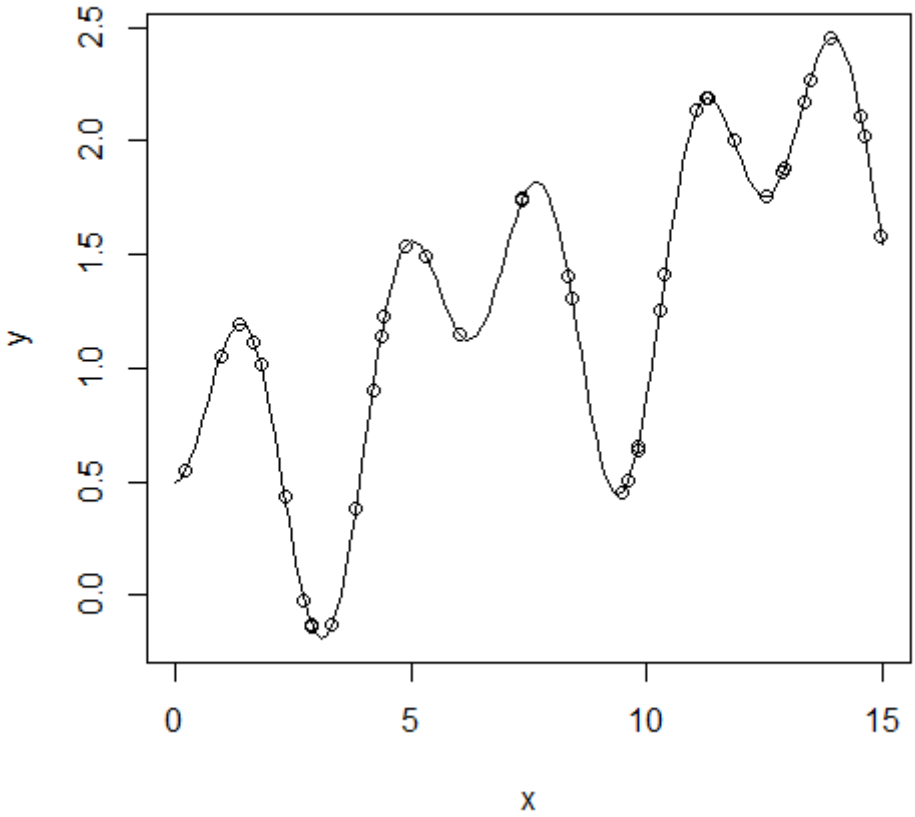
2.2 Random Search

Random Search (dalej RS) – Jest to metoda optymalizacji która nie wymaga gradientu do optymalizacji, dodatkowo RS można zastosować przy funkcjach które nie są ciągłe oraz różniczkowalne.

RS działa na zasadzie iteracyjnego posuwania się na lepszą pozycje w S którą możemy próbkować jako $R_d \subset S$.

Niech f będzie funkcją celu którą chcemy zoptymalizować. Niech $x \in \mathbb{R}^n$ wtedy RS możemy opisać następującym pseudo-kodem.

- Wygeneruj x z dziedziny w której chcemy zoptymalizować f
- Powtarzaj poniższe kroki dopóki nie będzie spełniony warunek zakończenia poszukiwania, może to być limit iteracji lub błąd bezwzględny.
    - Wygeneruj nowy losowy punkt x
    - Jeśli $f(y) < f(x)$ wtedy $x = y$
- Teraz x jest naszym rozwiązaniem



Rysunek 2 Przykładowe działanie RS

2.3 Kiefer – Wolfowitz

Kiefer-Wolfowitz (dalej K-W) – Jest to algorytm z rodziny stochastycznej optymalizacji która wyszukuje ekstrema funkcji których nie można obliczyć bez pośrednio, a jedynie wyestymować poprzez obserwacje szumu.

Niech $M(x)$ będzie funkcją która posiada minimum w punkcie θ, zakładamy że $M(x)$ jest nieznane, jednak z pewniej obserwacji $N(x)$ możemy obliczyć ekstremum. Struktura algorytmu jest podobna do gradientowych algorytmów. Ogólny opis algorytmu możemy zapisać jako następujący ciąg.

$$x_{n+1} = x_n + a_n\left(\frac{N(x_n + c_n) - N(x_n - c_n)}{c_n}\right) \tag{2.4}$$

$$gdzie:\ c_n = \frac{1}{n}, \qquad a_n = n^{-1/3} \tag{2.5}$$

Aktualna implementacja algorytmu w projekcie inżynierskim nieco się różni od ogólnego zapisu, zmieniliśmy parametry $\{c_n\}\ oraz\ \{a_n\}$.

Dane

$$c_n = \frac{1}{n+1} \tag{2.6}$$

$$a_n = \frac{2}{n^{\frac{1}{6}}}$$

$$N - ilość\ kroków\ K - W$$

Inicjalizacja

$$n = 1 \tag{2.7}$$

$$x_1 \in \mathbb{R}$$

Procedura

1. Jeśli $n \leq N$, wykonuj

   1.1. $x_{n+1} = x_n + a_n\left(\frac{N(x_n+c_n)-N(x_n-c_n)}{c_n}\right)$

   1.2. Jeśli $x_{n+1} > R_d \ \vee\ x_{n+1} < R_d$, wtedy $x_{n+1} = R_d$ (jak oznaczyć lewa / prawą stronę zbioru ?)

2. Jeśli $n > N$, zwróć x_N

Jak już wcześniej zostało wspomniane parametry $\{c_n\}\, oraz\, \{a_n\}$ różnią się między od ogólnego zapisu, w tym przypadku mamy dowolność i sami możemy sobie dobierać te parametry w zależności od wyglądu funkcji.

Parametry te muszą spełniać dwa następujące warunki

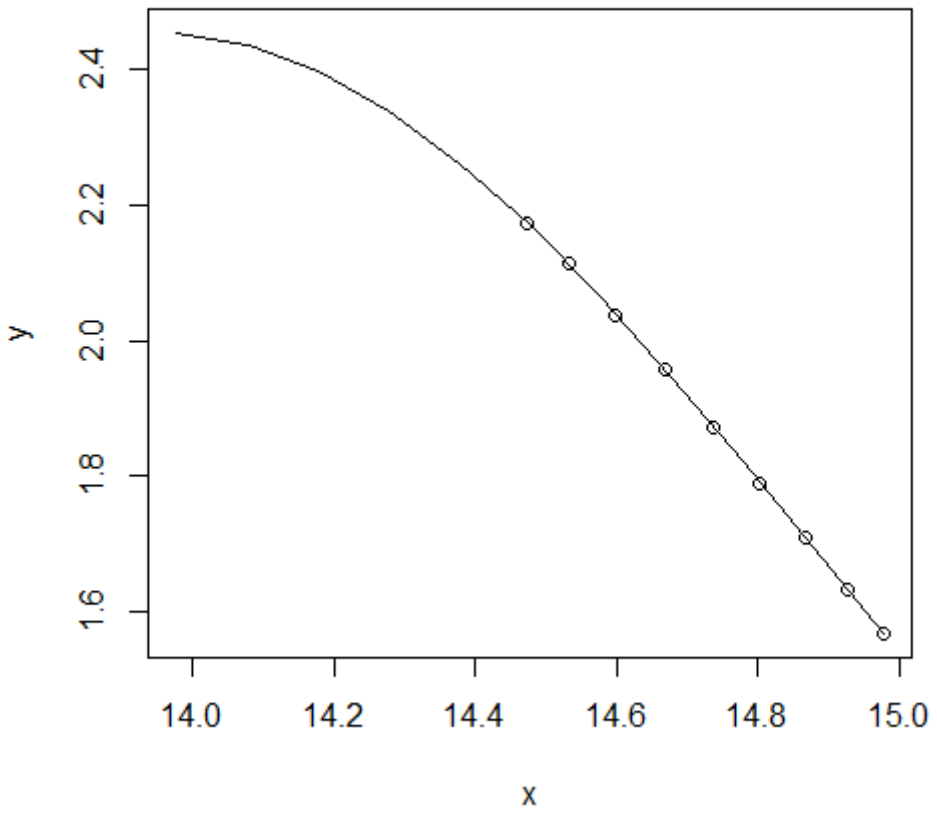
$$c_n \to 0, a_n \to 0\ gdy\ n \to \infty \tag{2.8}$$

$$\sum_{n=0}^{\infty} a_n = \infty, \sum_{n=0}^{\infty} \frac{a_n^2}{c_n^2} < \infty$$

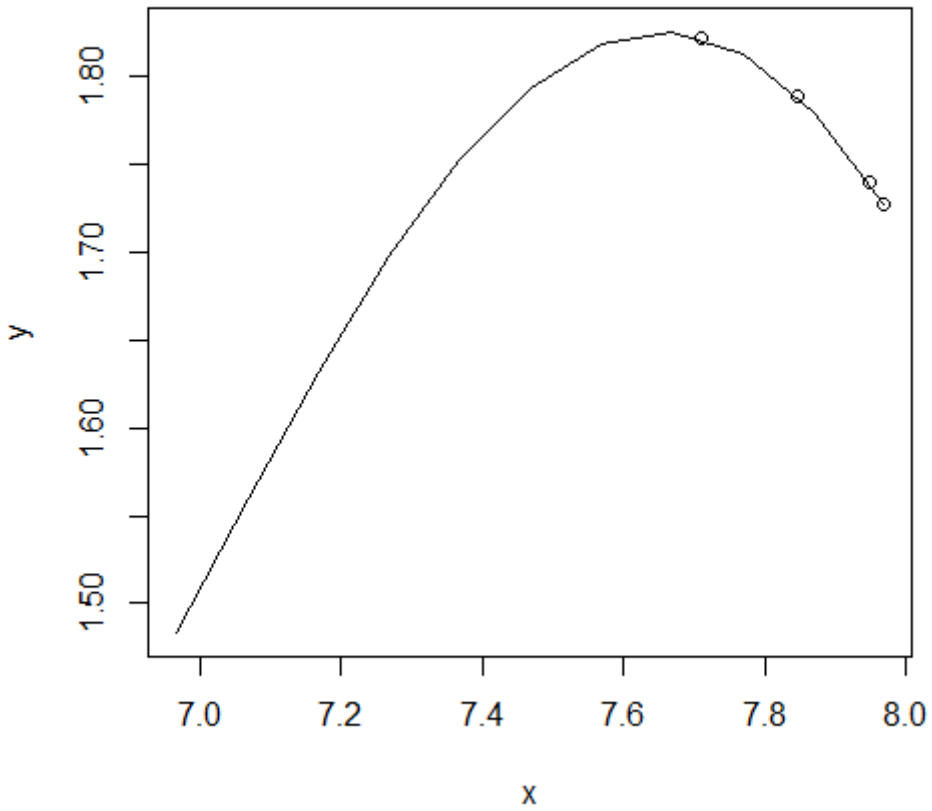
W bardzo łatwy sposób możemy zbadać czy nasze parametry zbiegają do zera, z pomocą przychodzi nam twierdzenie o granicy ciągu.

$$\lim_{n\to\infty}\left(\frac{1}{n+1}\right) = 0\ oraz\ \lim_{n\to\infty}\left(\frac{2}{\frac{1}{n^6}}\right) = 0 \tag{2.9}$$

Zdjęcia przedstawiające prawidłowe obliczenie minimum lokalnego oraz błędne.



Rysunek 3 Prawidłowe działanie algorytmu



Rysunek 4 Błędne działanie algorytmu

2.4 Przegląd innych algorytmów optymalizacji lokalnej

Jak już wcześniej zostało wspomniane nasz algorytm składa się w zasadzie z dwóch innych algorytmów, czyli Random Search oraz Kiefer-Wolfowitz, w takim przypadku nasz algorytm jest dość mocno modularny i możemy dowolnie zmieniać metodę poszukiwania minimum lokalnego.

W tym miejscu należy wymienić pozostałe algorytmy optymalizacji lokalnej.

- Algorytmy optymalizacji w kierunku
- Algorytmy optymalizacji bez ograniczeń
- Algorytmy optymalizacji z ograniczeniami

2.5 Przegląd innych metod optymalizacji globalnej

Należy wspomnieć także o innych metodach optymalizacji globalnej oraz powiedzieć kilka zdań na ich temat. Metody te można podzielić na dwie grupy, metody deterministyczne oraz niedeterministyczne, gdzie operacje losowe są istotnym elementem.

- Metody deterministyczne
    - siatki
    - trajektorii cząstki
    - kary
- Metody niedeterministyczne
    - poszukiwań losowych
    - wykorzystujące grupowanie
    - z wykorzystaniem stochastycznego modelu funkcji celu
    - algorytmy ewolucyjne

Z dotychczasowych badań, zostało wykazane iż metody niedeterministyczne są o wiele szybsze od deterministycznych, jednak wymagają dużych zasobów komputera dlatego zalecane jest stosować programowanie równoległe.

2.6 Charakterystyki użytych języków

2.6.1 Język R

Język ten jest dość mocno abstrakcyjny dzięki czemu można w bardzo szybki oraz przystępny sposób stworzyć prototyp algorytmu. Środowisko posiada bardzo wielką bazę zewnętrznych pakietów (np. sieci neuronowe), jest to pomocne ponieważ można się skupić na implementacji naszego algorytmu.

„R" jest darmowym językiem wydanym na licencji GNU PL, dlatego też jest to jeden z czynników dla czego ten język został wybrany.

2.6.2 Język Haskell

Haskell jest z rodzimy języków funkcyjnych, sam haskell jest językiem czysto funkcyjnym. Język ten jest intensywnie rozwijany przy Uniwersytecie Glasgow najbardziej popularnym kompilatorem jest GHC (Glasgow Haskell Compilator).

Najbardziej charakterystyczne cechy tego języka to :

- Leniwe wartościowanie
- Monady
- Statyczny polimorfizm
- Klasy typów (ang. Typeclass)
- Strażnicy (ang. Guards)
- Rozwijanie funkcji (ang. currying) oraz częściowe funkcje (ang. partial functions)

Główną różnicą pomiędzy językiem funkcyjnym a imperatywnym jest formowanie problemu oraz zapis. W języku imperatywnym nasza funkcja posiada kilka kroków do wykonania i może zmieniać swój stan w zależności jakie zmienne przyjmiemy w ciele funkcji, kiedy w języku funkcyjnym nasza funkcja nie może zmieniać swojego stanu w trakcie jej wykonywania, dodatkowo zmienne są „niezmienne" (ang. unmutable) np. kiedy będziemy chcieli każdy element listy pomnożyć przez dwa, wtedy wynik zawsze będzie reprezentowany przez nową listę ponieważ mamy możliwości zmieniać stanów istniejących zmiennych.

Ponadto kompilator GHC wspiera pisanie programów równolegle oraz współbieżne, na potrzeby mojego projektu inżynierskiego algorytm będzie tylko zrównoleglony bez zarządzania wątkami jak to się ma przy programowaniu współbieżnym. (do sprawdzenia!!!!)

2.7 Programowanie równoległe

Haskell wspiera programowanie równoległe dla dwóch typów procesorów

- CPU
- GPU

Temat programowania równoległego można traktować jako kontrargument na aktualne limity w częstotliwości taktowania procesora, oraz brak postępu w tym temacie.

//Równoległe – oznacza to że nasz program będzie pracował na wielu procesorach jeśli takowe komputer posiada, zazwyczaj zrównoleglanie odbywa się „nie widocznie" oraz bez większych zmian implementacyjnych.

Współbieżne – cos

Rozdział 3 Implementacja algorytmu.

3.1 Opis stanowiska oraz środowiska

Cała praca oraz badania zostały wykonane na osobistym laptopie o następującej konfiguracji procesor - AMD A8-3500M, pamięć ram - 6 GB 1333 MHz, dysk TOSHIBA MK6459GSXP SATA, system operacyjny Windows 8.

Podczas pisania programu w języku R było używane środowisko o nazwie „RStudio" które pozwoliło na dość szybką implementacje algorytmu oraz pomaga przy zarzadzaniu różnymi bibliotekami.

Przy pisaniu algorytmu w języku Haskell był używany kompilator GHC oraz program „Cabal" do zarządzania zewnętrznymi bibliotekami. Dodatkowo do diagnostyki oraz pomiaru pracy procesorów trybie zrównoleglonym zostało użyte narzędzia o nazwie „ThreadScope", który pozwala obejrzeć ilość tworzonych wątków, czasy procesora, czasy Grabage Collctor (GC) jest to dość zaawansowanie narzędzie.

Rozdział 4 Badania

Przy badaniu wydajności algorytmu każda z implementacji została uruchomiona dziesięć razy przy następujących konfiguracjach.

$$RandomPoints = 100000 \tag{4.1}$$

$$SubsetWidth = 0.5 \tag{4.2}$$

$$lGlobalDomain = 0 \tag{4.3}$$

$$rGlobalDomain = 15 \tag{4.4}$$

$$kwSteps = 100 \tag{4.5}$$

$$functionToSolve = f(x) = \frac{2 * \sin(x)^2 + \cos(x)}{2} + x * 0.1 \tag{4.6}$$

RandomPoints – jest to zmienna która określa nam ile losowych punktów będzie wygenerowanych dla poszukiwania losowego.

SubsetWidth – wartość która określa nam szerokość podprzedziału w którym będziesz poszukiwanie minimum lokalne.

lGlobalDoman oraz rGlobalDomain – zakres przedziału do przeszukiwania minimum globalnego.

kwSteps – ilość kroków dla algorytmu Kiefer – Wolfowitz, czyli optymalizacja lokalna.

functionToSolve – wzorcowa funkcja w której poszukujemy minimum globalne.

(obrazek wzorcowej funkcj)

4.1 Przebieg oraz badanie implementacji w języku R

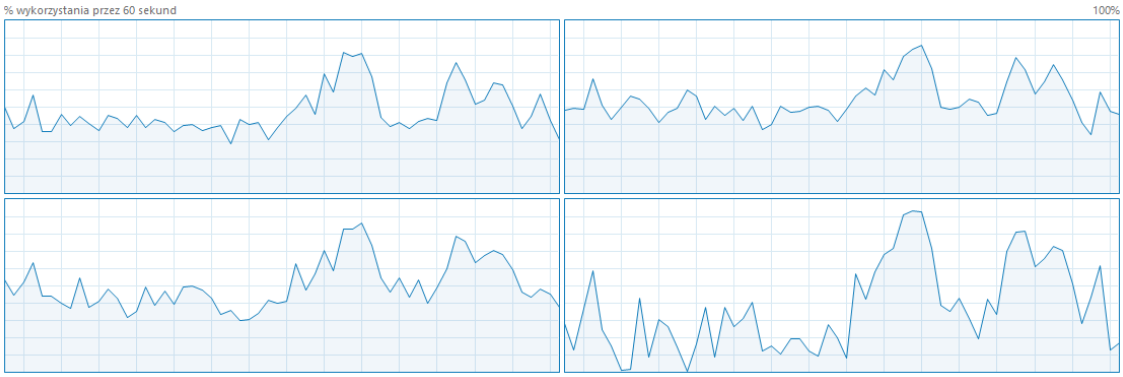
W pierwszej kolejności algorytm został napisany w języku R, ponieważ bardzo szybko można przenieść idee z papieru na komputer w tym języku, łatwo się prototypuje.

Na początek została zaimplementowana metoda optymalizacji globalnej przeszukiwania losowego bez algorytmu Kiefer-Wolfowitza.

Gdy metoda RS została poprawnie zaimplementowana zaczęła się praca nad implementacją algorytmu do poszukiwania minimum lokalnego (KW). Na tym etapie prac nie zostały napotkane żadne problemy. Dodatkowo trzeba tutaj nadmienić iż algorytm na razie działa dla funkcji jednej zmiennej.

Kiedy pisanie programu zostało ukończone, zostały przeprowadzone badania nad wydajnością algorytmu oraz samego języka i platformy. Została użyta do tego wbudowana funkcja system.time(FUNC) która jako argument przyjmuje funkcje.

Dla danych podanych na początku rozdziału, czas działania algorytmu to 222,98 sekund



Rysunek 5 Wykorzystanie procesora podczas uruchomionego programu napisanego w R

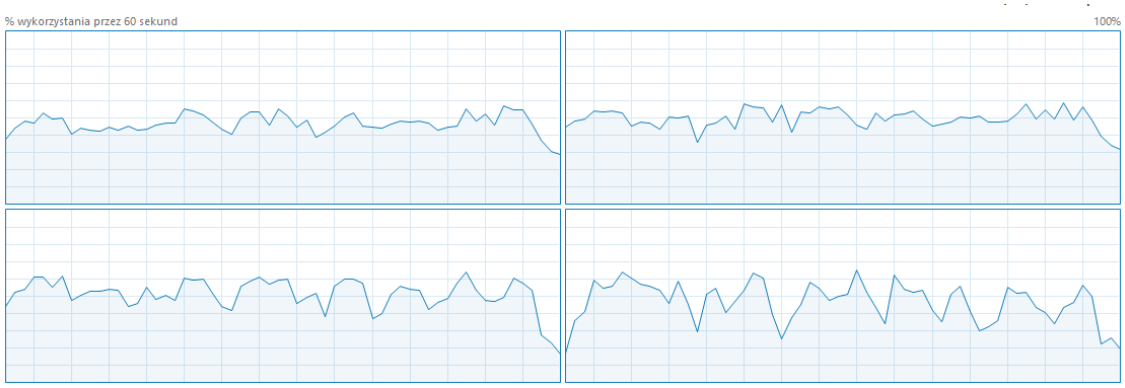
Jak można zauważyć nasze procesory nie są równomiernie wykorzystywane dodatkowo tylko jeden procesor pracuje w tym czasie.

4.2 Przebieg oraz badanie implementacji w języku Haskell

Następnym etapem projektu inżynierskiego było przepisanie z prototypowanego algorytmu w R do Haskell. Zostały napotkane pewnie problemy głównie przez to iż programowanie funkcyjne bardzo mocno różni się od programowania imperatywnego, potrzebna jest zmiana myślenia oraz pojmowania problemu na temat rekurencji.

Pierwszym z widocznych rezultatów przepisywania wszystkich algorytmów numerycznych na języki funkcyjne jest prostota zapisu. Implementacja algorytmów w językach funkcyjnych jest bardziej oczywista z punktu widzenia matematycznego.

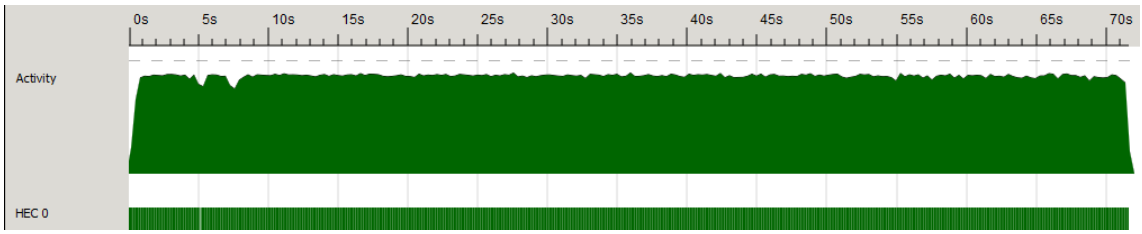
Po zakończeniu przepisywania algorytmu zostały przeprowadzone badania nad efektywnością implementacji, tak samo jak miało miejsce przy poprzednim punkcie.



Rysunek 6 Wykorzystanie zasobów procesorów - Haskell, bez zrównoleglenia

Czas wykonywania programy to 71,58 sekundy, czyli już samo przepisanie na język kompilowany to ogromne przyspieszenie przy bardzo złożonych obliczeniach, prawdopodobnie dla trywialnych problemów nie było by większego sensu przepisywanie algorytmu na Haskell.

Następnie wykres pochodzi z bardziej zaawansowanego narzędzia o nazwie „threadScope”.



Rysunek 7 Wykorzystanie zasobów procesora – threadscope

Aktywność (ang. activity) w tym przypadku możemy zobaczyć aktywność naszych procesorów, które są oddzielone pomiędzy sobą przerywaną szarą linią, w naszym przypadku wykorzystujemy około 80% zasobów jednego procesora.

„HEC” możemy traktować jako jeden procesor, w późniejszych przykładach tych pozycji będzie więcej o tyle o ile zostało uruchomionych procesorów.

4.3 Badanie implementacji w języku Haskell zrównoleglone na CPU

Oby zrównoleglić algorytm, został użyty moduł „Parallel”, który został zainstalowany poprzez program „cabal”.

Kompilator pozwala nam na zrównoleglenie naszego programu, dzięki zastosowaniu architektury symetrycznych procesorów, czyli wykorzystanie dwóch lub więcej procesorów do jednoczesnego wykonywania zadań, wtedy procesory są równo obciążone pracą.

Trzeba przepisać nasz algorytm tak aby można było pewne elementy przetwarzać równolegle oraz na etapie kompilacji trzeba zaznaczyć kompilatorowi że chcemy skompilować nasz program w trybie zrównoleglonym, w tym przypadku musimy skompilować program wraz z flagą „-threaded”

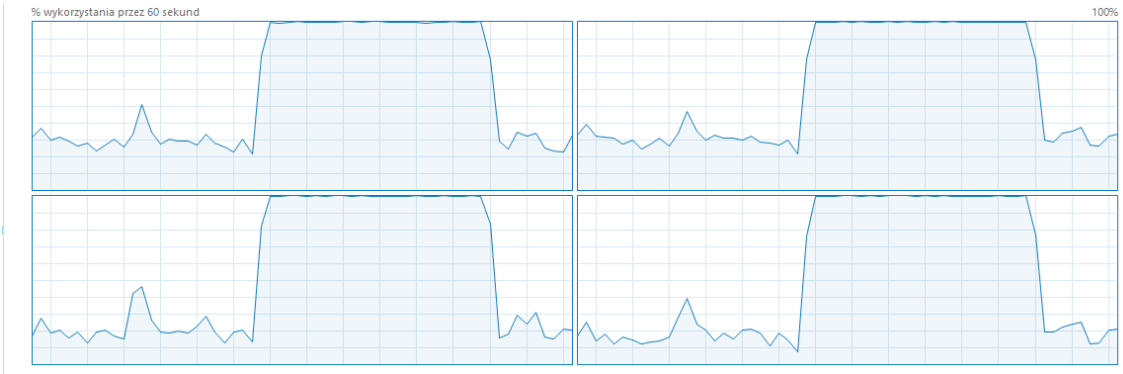
Pierwszym zadaniem przy zrównoleglaniu programy to załadowanie modułu „Control.Parallel” oraz „Control.Parallel.Strategies” są one częścią modułu „Parallel”.

Następnym etapem jest stworzenie funkcji która będzie aplikowała inną funkcje do każdego elementu w liście, ale metodą „kawałków” czyli określamy po ile elementów z będziemy mapować w jednej sekwencji.

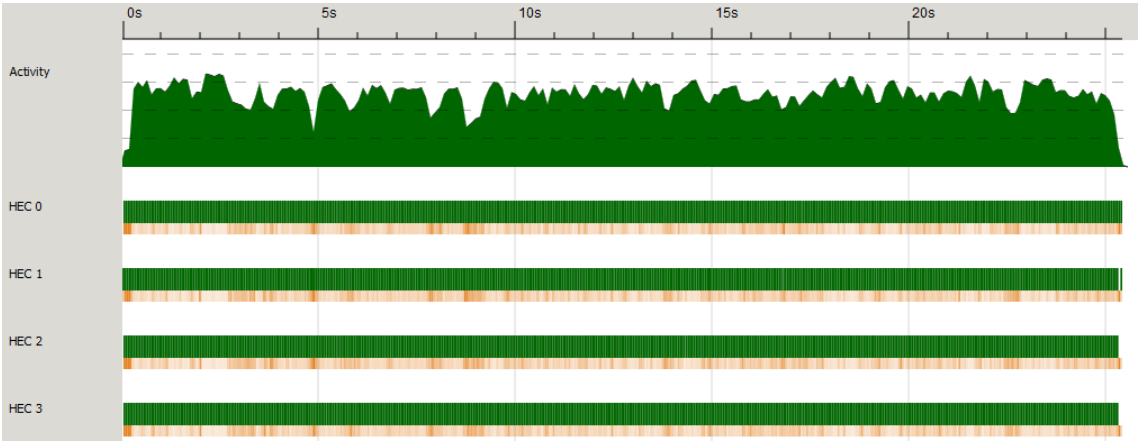
Naszym celem jest równoległe obliczanie minimum lokalnego w wygenerowanych losowo punktach.

Po zastosowaniu tych zmian w implementacji, zrównolegliliśmy obliczanie minimum lokalnego na wygenerowanych punktach w przeszukiwaniu losowym.

Następnie została zbadana szybkość wykonywania się programu oraz praca procesorów, co widać na załączanym poniżej obrazku.



Rysunek 8 Wykorzystanie zasobów procesorów przy zrównolegleniu



Rysunek 9 Wykorzystanie zasobów procesora - threadscope

Czas wykonania programu to 25,41sekund, można zauważyć że efektywność wykorzystania wszystkich czterech procesorów wynosi około 80%.

Rozdział 5 Podsumowanie

W pracy inżynierskiej zostały wykonane wszystkie zadania które zostały założone.

Program napisany w R rozwiązuje problem optymalizacji najwolniej, najprawdopodobniej dlatego iż jest to język skryptowy bez kompilatora, jak można zauważyć już samo przepisanie algorytmu w na język Haskell bardzo mocno przyspiesza nasz program. Jednak dopiero program zrównoleglony daje najlepsze rezultaty, i w pełni wykorzystuje zasoby komputera, w tym przypadku wszystkie. (Opisać jako ciekawostkę także test na 16 procesorach)

5.1 Dodatkowe uwagi oraz plany

TBC – pomysły oraz plany co robić dalej + propozycje przy ulepszeniu algorytmu

Bibliografia

TBC

Spis rysunków