

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

KIERUNEK: Elektronika i Telekomunikacja	1
PROJEKT INŻYNIERSKI	1
dr hab. inż. Przemysław Śliwiński	1
WROCŁAW 2014	1
Spis treści	2
Rozdział 1 Wstęp	4
1.1 Cele Projektu	5
Rozdział 2 Wprowadzenie teoretyczne	6
2.1 Zasady działania algorytmu	8
2.1.1 Random Search	9
2.1.2 Kiefer – Wolfowitz	11
2.2 Przegląd innych algorytmów optymalizacji lokalnej	14
2.3 Przegląd innych metod optymalizacji globalnej	14
2.4 Charakterystyki użytych języków	15
2.4.1 Język R	15
2.4.2 Język Haskell	15
2.5 Programowanie równoległe	16
2.5.1 Działanie programu zrównoleglonego na CPU	16
2.5.2 Działanie programu zrównoleglonego na GPU	17
Rozdział 3 Środowiska.	18
3.1 Opis stanowiska oraz środowiska	18
3.2 Środowisko Microsoft Azure	18
Rozdział 4 Badania	20
4.1 Przebieg oraz badanie implementacji w języku R	21
4.2 Przebieg oraz badanie implementacji w języku Haskell	22
4.3 Badanie implementacji w języku Haskell zrównoleglone na CPU	23
4.3.1 Algorytm na maszynie Azure	24
Rozdział 5 Podsumowanie	25
5.1 Dodatkowe uwagi oraz plany	25
Bibliografia	26
Spis rysunków	27

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 produkcję 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 implementacja równoległa algorytmu optymalizacji globalnej, metodą stochastyczną. Następnie zbadanie wydajności kilku implementacji takiego algorytmu oraz wyciągnięcie wniosków na temat zrównoleglania takich algorytmów. Algorytm będzie napisany w dwóch językach R oraz Haskell. Wybrany algorytmem jest algorytm (1). Dodatkowo algorytm ten zostanie uruchomiony na wirtualnej maszynie z szesnastoma procesorami w usłudze Microsoft Azure.

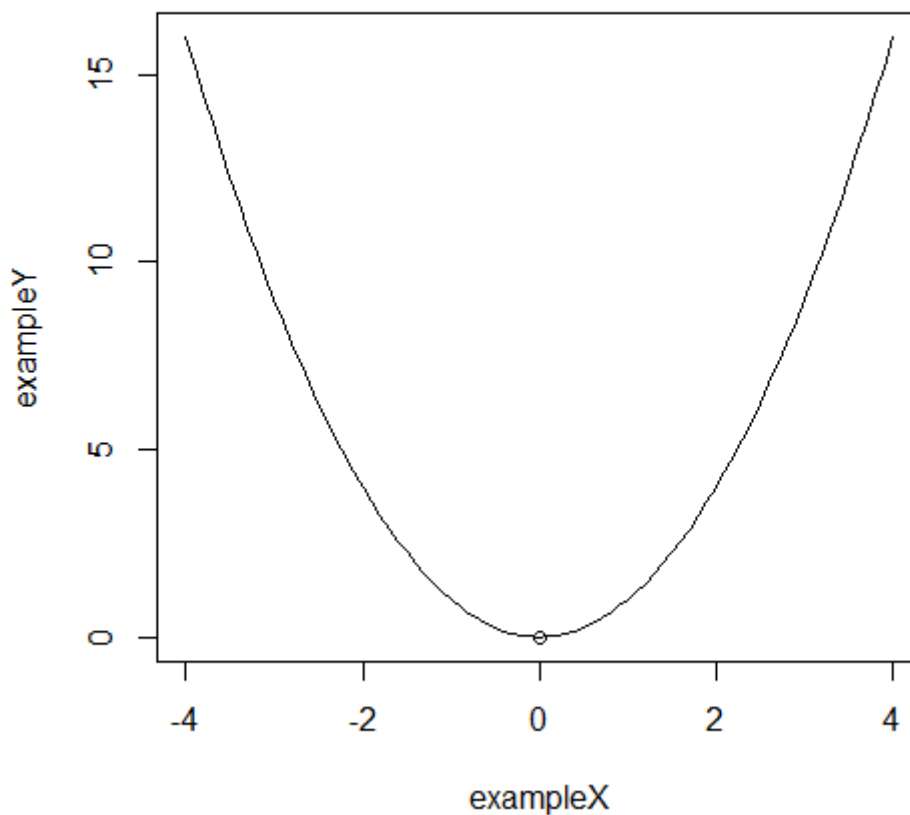
Rozdział 2 Wprowadzenie teoretyczne

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

Problem ten można opisać w następujący sposób.

$$\begin{aligned} f : A &\mapsto \mathbb{R} \\ A &\subset \mathbb{R}^n \end{aligned} \tag{2.1}$$

Należy znaleźć taki $x_o \in A$ że dla każdego $x \in A \setminus \{x_o\}$ 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 programowania 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.

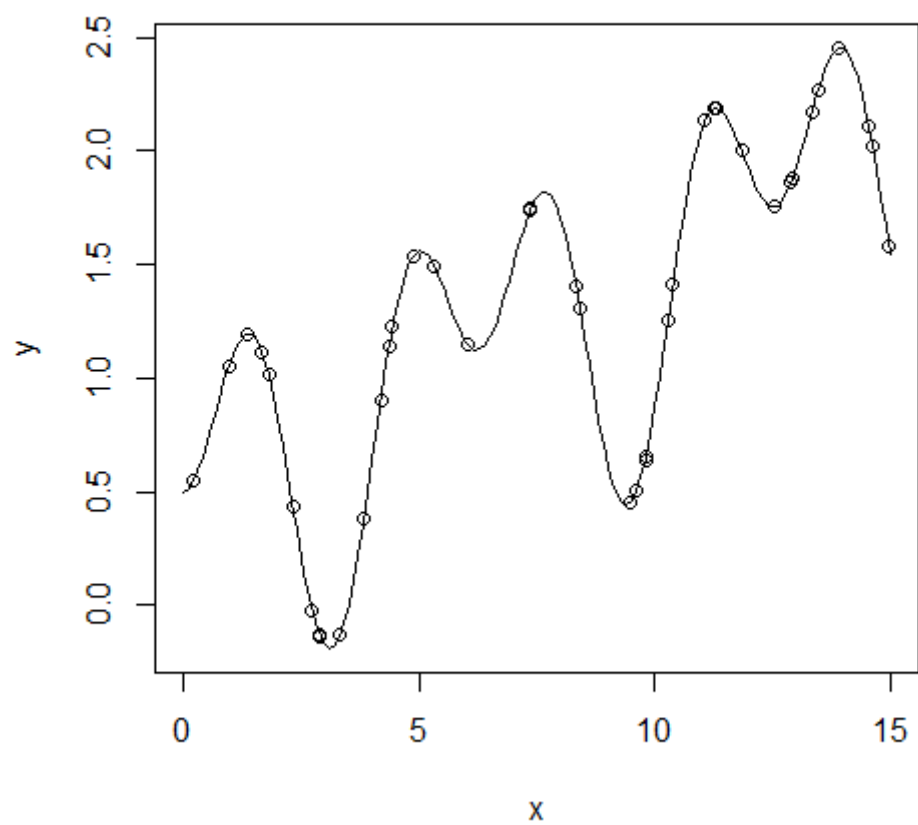
2.1.1 Random Search

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 pozycję 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

2.1.2 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.

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

$$\text{gdzie : } c_n = \frac{1}{n}, \quad a_n = n^{-1/3} \quad (2.5)$$

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

Dane

$$c_n = \frac{1}{n+1} \quad (2.6)$$

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

N – ilość kroków $K - W$

Inicjalizacja

$$n = 1 \quad (2.7)$$

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

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

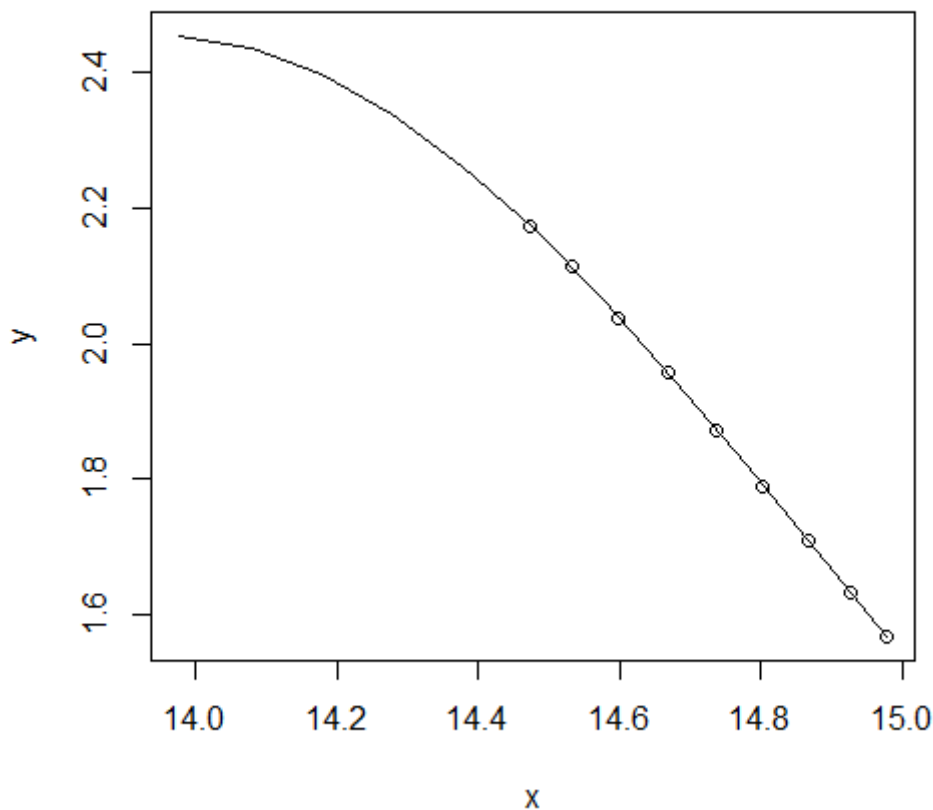
$$c_n \rightarrow 0, a_n \rightarrow 0 \text{ gdy } n \rightarrow \infty \quad (2.8)$$

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

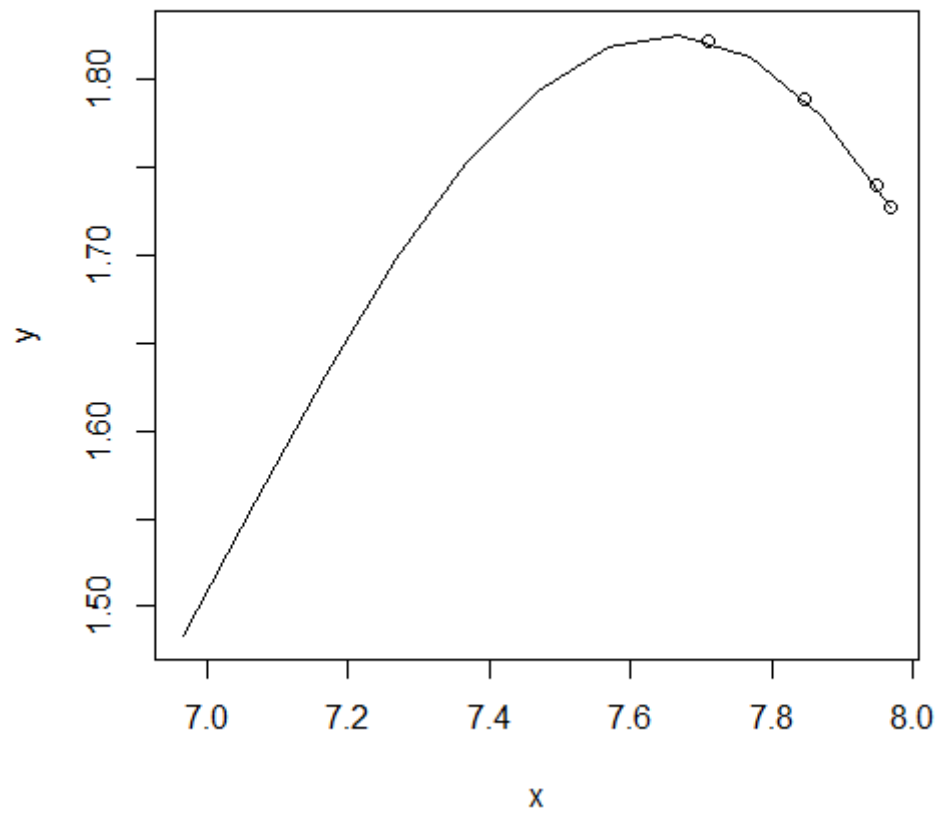
W bardzo łatwy sposób możemy zbadać czy nasze parametry zbiegają do zera, z pomocą przychodzi nam twierdzenie o granicy ciągu.

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

Zdjęcia przedstawiające prawidłowe obliczenie minimum lokalnego oraz błędne.



Rysunek 3 Prawidłowe działanie algorytmu



Rysunek 4 Błędne działanie algorytmu

2.2 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 modułarny 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

2.3 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. (2)

- 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 stosowanie programowania równoległego. (2)

2.4 Charakterystyki użytych języków

2.4.1 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 wydany na licencji GNU PL, dlatego też jest to jeden z czynników dla czego ten język został wybrany. Jest to narzędzie używane głównie przez analityków oraz statystyków.

2.4.2 Język Haskell

Haskell jest językiem programowania z rodziny języków funkcyjnych, sam haskell czysto funkcyjny. Język ten jest intensywnie rozwijany przy Uniwersytecie Glasgow najbardziej popularnym kompilatorem jest GHC (Glasgow Haskell Compiler).

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 przyjmujemy w ciele funkcji, kiedy w języku funkcyjnym nasza funkcja nie może zmieniać swojego stanu w trakcie jej wykonywania, dodatkowo zmienne są „niezmienne” (ang. immutable) np. kiedy będziemy chcieli każdy element listy pomnożyć przez dwa, wtedy wynik zawsze będzie reprezentowany przez nową listę ponieważ nie mamy możliwości zmieniać stanów istniejących zmiennych musimy wynik zwrócić jako nowy obiekt.

Ponadto kompilator GHC wspiera pisanie programów równoległe używając pamięci dzielonej oraz współbieżne (3), na potrzeby mojego projektu inżynierskiego algorytm będzie napisany równoległe.

2.5 Programowanie równoległe

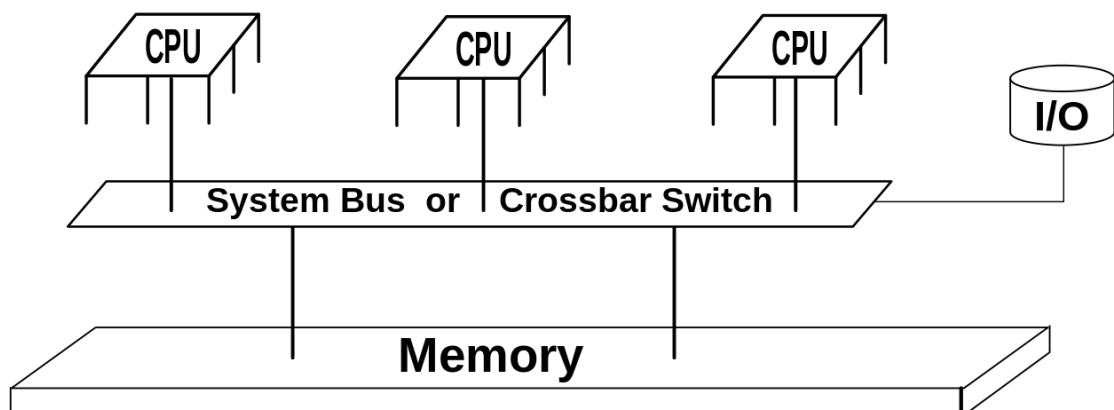
Jak zostało wspomniane wcześniej Haskell wspiera programowanie równoległe trzeba nadmienić że jest możliwość programowanie równoległego dla dwóch typów procesorów

- CPU
- GPU

Programowanie równoległe można traktować jako kontrargument na aktualne limity w częstotliwości taktowania procesora, oraz brak postępu w tej dziedzinie. Zapewnia nam przyspieszenie obliczeń sekwencyjnych oraz w pełni wykorzystanie zasobów komputera, od kilku lat standardem produkcyjnym są procesory kilku rdzeniowe.

2.5.1 Działanie programu zrównoleglonego na CPU

Przy programowaniu równoległym na CPU, gdzie jedyną jednostką obliczeniową jest CPU, należy wspomnieć o „Shared Memory Multiprocessors” (SMP), współdzielonej pamięci wykorzystywanej przez wiele procesorów



Rysunek 5 Współdzielony dostęp do pamięci przez trzy procesory

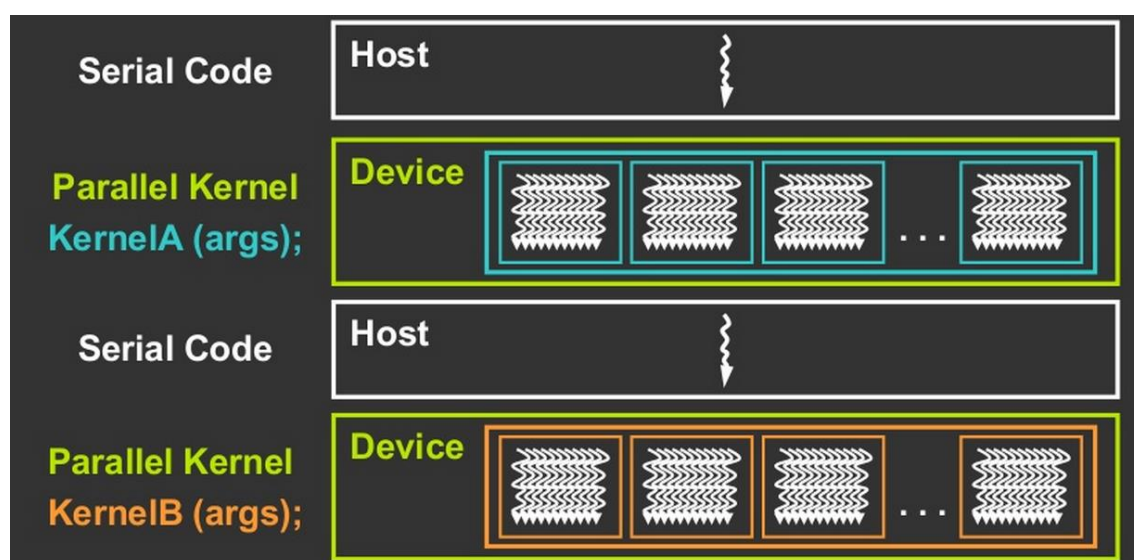
Zazwyczaj pamięć dzielona jest to pamięć do której jest możliwość jednoczesnego korzystania przez wiele procesorów, gdy mówi się o pamięci dzielonej najczęściej ma się na myśli ją jako duży blok danych zaalokowany w pamięci RAM, do którego dostęp może posiadać wiele procesorów tak jak jest to ukazane na obrazku powyżej.

Dodatkową zaletą jest to iż komunikacja pomiędzy procesorami może być szybka tak jak dostęp do pamięci w tej samej lokalizacji.

Zastosowanie pamięci dzielonej jest o wiele tańszym rozwiązaniem oraz prostszym w porównaniu do pamięci rozproszonej, jednak ogranicza nas to do kilkudziesięciu procesorów, gdy w przypadku pamięci rozproszonej tych procesorów może być nawet kilka tysięcy.

2.5.2 Działanie programu zrównoleglonego na GPU

W tym przypadku w obliczeniach biorą udział dwa rodzaje jednostek obliczeniowych, GPU oraz CPU.



Rysunek 6 Zasada przetwarzania danych przez GPU

Działanie jest bardzo proste najpierw musimy zaalokować pamięć na CPU a następnie częściami przesyłamy dane do pamięci karty graficznej, następnie procesory karty graficznej wykonują obliczenia i zwracają wynik który jest wysyłany na CPU.

Rozdział 3 Środowiska.

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 implementację algorytmu oraz pomaga przy zarządzaniu 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 narzędzie o nazwie „ThreadScope”, który pozwala obejrzeć ilość tworzonych wątków, czasy procesora, czasy Garbage Collector (GC) jest to dość zaawansowane narzędzie.

3.2 Środowisko Microsoft Azure

Korzystając z okazji, algorytm został uruchomiony także na maszynie wirtualnej w usłudze Microsoft Azure.

Temat wirtualizacji znacząco odbiega od tematu projektu inżynierskiego także zostanie ukazany tylko proces tworzenia takiej maszyny oraz komentarz do tego.

Ogólnie rzecz ujmując firma Microsoft umożliwia nam w dość przystępny sposób stworzenie maszyny wirtualnej typu PaaS (ang. Platform as a service) oznacza to iż sami nie musimy się martwić o infrastrukturę, możemy jedynie zdecydować jaki system operacyjny oraz jakie programy będą dostępne na naszej maszynie, jest to dość wygodne rozwiązanie z uwagi na to że w przeciągu 10 minut możemy stworzyć środowisko posiadające 16 a nawet 32 procesory oraz przeogromne zasoby pamięci RAM.

Na cele projektowe została stworzona maszyna wirtualna o konfiguracji 16 procesorów logicznych oraz 112GB pamięci RAM, posiada system operacyjny typu Windows.

The screenshot displays the 'NEW' section of the Microsoft Azure portal. On the left, there is a navigation menu with categories: COMPUTE, WEBSITE, and QUICK CREATE. Under COMPUTE, options include DATA SERVICES, APP SERVICES, NETWORK SERVICES, and MARKETPLACE (marked as PREVIEW). Under WEBSITE, options include VIRTUAL MACHINE (highlighted), MOBILE SERVICE, CLOUD SERVICE, and BATCH SERVICE (marked as PREVIEW). The QUICK CREATE section has a 'FROM GALLERY' option. The main area on the right contains a form for creating a virtual machine. It includes fields for DNS NAME, IMAGE (set to Windows Server 2012), SIZE (set to D14 (16 cores, 112 GB)), USER NAME, NEW PASSWORD, and CONFIRM. A dropdown for REGION/AFFINITY GROUP is set to North Europe. At the bottom right, there is a 'CREATE A VIRTUAL MACHINE' button with a checkmark icon.

Rysunek 7 Proces tworzenia wirtualnej maszyny - Microsoft Azure

Proces tworzenia wirtualnej maszyny jest bardzo prosty, wystarczy przejść do panelu zarządzania swoim kontem w usłudze Azure a następnie kliknąć przycisk „New” który nam wyświetli powyższe okienko, następnie zaznaczamy to samo co jest zaznaczona na obrazku powyżej i wypełniamy puste pola oraz wybieramy z dostępnej listy system operacyjny oraz konfigurację maszyny.

Na zakończenie tego podpunktu należy wspomnieć także o innych usługodawcach, jest to Amazon Web Services (AWS), Octawave.

Rozdział 4 Badania

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

$$\text{RandomPoints} = 100000 \quad (4.1)$$

$$\text{SubsetWidth} = 0.5 \quad (4.2)$$

$$\text{lGlobalDomain} = 0 \quad (4.3)$$

$$\text{rGlobalDomain} = 15 \quad (4.4)$$

$$\text{kwSteps} = 100 \quad (4.5)$$

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

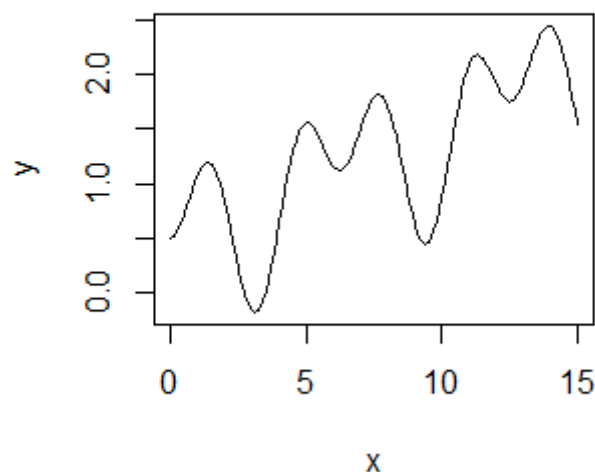
RandomPoints – jest to zmienna która określa nam ile losowych punktów będzie wygenerowanych dla poszukiwania losowego.

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

lGlobalDomain 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.



Rysunek 8 Wizualizacja badanej funkcji

4.1 Przebieg oraz badanie implementacji w języku R

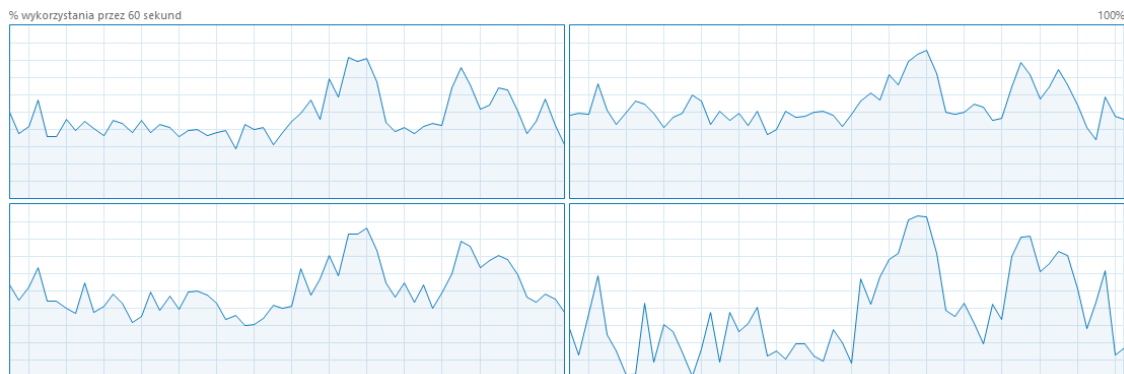
W pierwszej kolejności algorytm został napisany w języku R, ponieważ bardzo szybko można przenieść idee z papieru na komputer w tym języku, łatwo się prototypuje.

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 9 Wykorzystanie procesora podczas uruchomionego programu napisanego w R

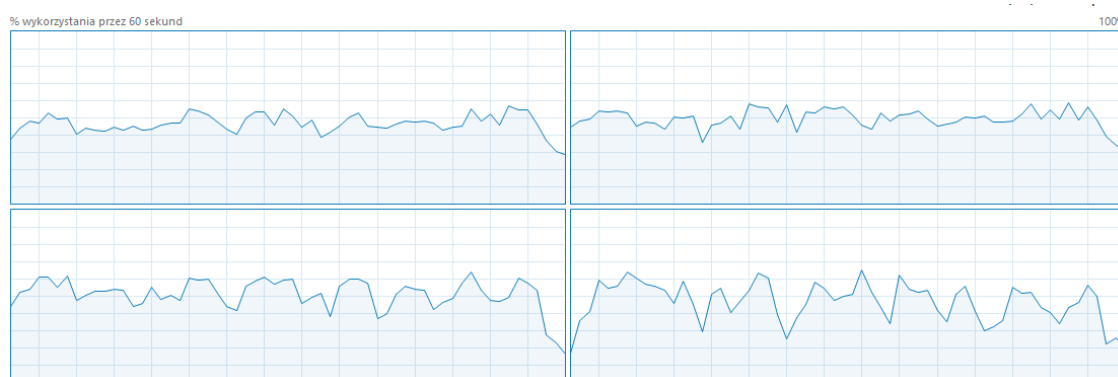
Jak można zauważyć nasze procesory nie są równomiernie wykorzystywane dodatkowo tylko jeden procesor pracuje w tym czasie.

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.

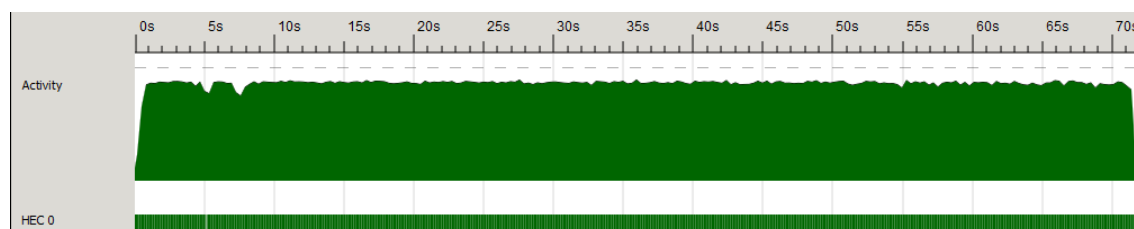
Po zakończeniu przepisywania algorytmu zostały przeprowadzone badania nad efektywnością implementacji, tak samo jak miało miejsce przy poprzednim punkcie.



Rysunek 10 Wykorzystanie zasobów procesorów - Haskell, bez zrównoleglenia

Czas wykonywania programu 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 11 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”

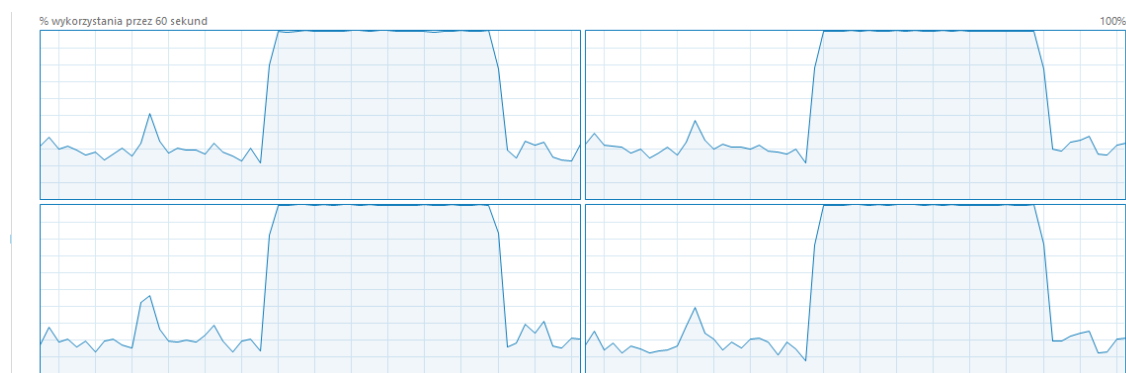
Pierwszym zadaniem przy zrównoleglaniu programy to załadowanie modułu „Control.Parallel” oraz „Control.Parallel.Strategies” są one częścią modułu „Parallel”.

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

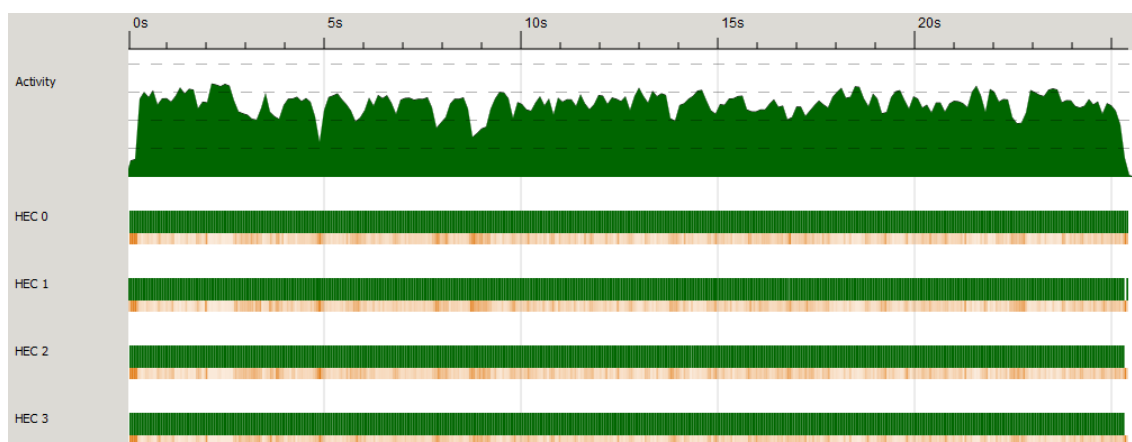
Naszym celem jest równoległe obliczanie minimum lokalnego w wygenerowanych losowo punktach.

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 12 Wykorzystanie zasobów procesorów przy zrównolegleniu



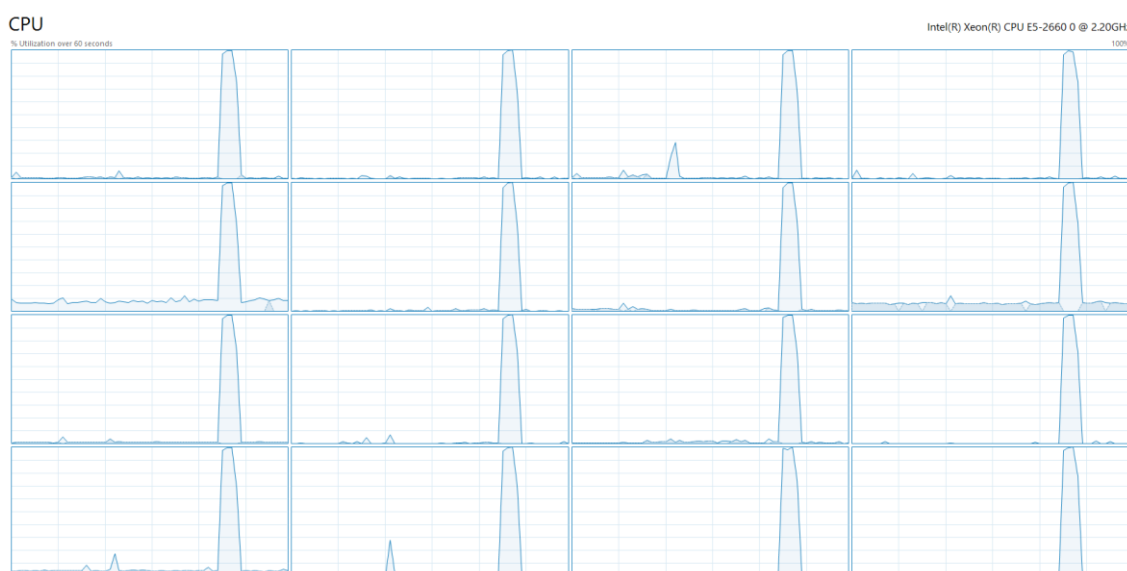
Rysunek 13 Wykorzystanie zasobów procesora - threadscope

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

4.3.1 Algorytm na maszynie Azure

Tak jak zakładaliśmy w celach tej pracy, program został uruchomiony na maszynie posiadającej szesnaście procesorów uruchomionej w usłudze Microsoft Azure.

Pierwsze co można było zaobserwować to to że algorytm wykonał się poniżej 3 sekund a dokładnie 2,69 sekund.



Rysunek 14 Zużycie zasobów procesorów podczas wykonywania algorytmu- 16 procesorów

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 procesory. Dla szesnastu procesorów nasz problem wydaje się być z byt trywialny ponieważ ledwo jest osiągnane maksimum użycia procesorów, jako ciekawostkę można tutaj napisać ze dopiero zwiększenie problemu o jeden rząd (zwiększenie losowych punktów w RS) ukazuje pracę procesorów na pełnym wykorzystaniu.

Najlepsze efekty mogły by być uzyskane tylko i wyłącznie jeśli byśmy zrównoleglili nasz algorytm tak aby wykorzystywał zasoby GPU prawdopodobnie przyspieszyłby o kilkadziesiąt razy.

5.1 Dodatkowe uwagi oraz plany

Pierwszy etap został zakończony czyli został zbudowany algorytm dla funkcji jedno wymiarowej, następnym etapem jest dostosowanie algorytmu do funkcji 2 oraz 3 wymiarowych a na końcu do 9 wymiarów, jak wspomina Sid Yakowitz (1) algorytm który opisał jest efektywny dla funkcji maksymalnie dziewięciu wymiarów.

Następnym etapem ulepszania algorytmu mogło by być automatyczne dobieranie szerokości przedziału w którym będzie poszukiwane minimum lokalne opisane (4) jako „Adaptive Step Size Random Search”

Na samym końcu jak już algorytm zostanie maksymalnie ulepszony, trzeba przepisać go tak aby wykorzystywał zasoby GPU a następnie można stworzyć portal który by udostępniał tę funkcjonalność dzięki któremu moglibyśmy spieniężyć naszą pracę.

Bibliografia

1. *A globally convergent stochastic approximation*. **Yakowitz, Sid.** 1, January 1993, SIAM J. Control and Optimization, Vol. 31, pp. 30-40.
2. *Evolutionary Computation and Global Optimization*. **Niewiadomska-Szynkiewicz Ewa.** Potok Złoty : brak nazwiska, 1999. Przegląd Metod Optymalizacji Globalnej.
3. **haskell.org.** [Online] https://downloads.haskell.org/~ghc/7.0-latest/docs/html/users_guide/lang-parallel.html.
4. **Random Search.** *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Random_search.

Spis rysunków

Rysunek 1 Przykładowa funkcja z minimum w punkcie 0	6
Rysunek 2 Przykładowe działanie RS	10
Rysunek 3 Prawidłowe działanie algorytmu	13
Rysunek 4 Błędne działanie algorytmu	13
Rysunek 5 Współdzielony dostęp do pamięci przez trzy procesory	16
Rysunek 6 Zasada przetwarzania danych przez GPU	17
Rysunek 7 Proces tworzenia wirtualnej maszyny - Microsoft Azure.....	19
Rysunek 8 Wizualizacja badanej funkcji.....	20
Rysunek 9 Wykorzystanie procesora podczas uruchomionego programu napisanego w R.....	21
Rysunek 10 Wykorzystanie zasobów procesorów - Haskell, bez zrównoleglenia.....	22
Rysunek 11 Wykorzystanie zasobów procesora – threadscope.....	22
Rysunek 12 Wykorzystanie zasobów procesorów przy zrównolegleniu.....	23
Rysunek 13 Wykorzystanie zasobów procesora - threadscope	24
Rysunek 14 Zużycie zasobów procesorów podczas wykonywania algorytmu- 16 procesorów	24