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**

[**Rozdział 1 Wstęp** 4](#_Toc405673613)

[1.1 Cele Projektu 5](#_Toc405673614)

[1.2 Zarys problemu 5](#_Toc405673615)

[**Rozdział 2 Wprowadzenie teoretyczne** 6](#_Toc405673616)

[2.1 Zasady działania algorytmu 8](#_Toc405673617)

[2.1.1 Random Search 9](#_Toc405673618)

[2.1.2 Kiefer – Wolfowitz 10](#_Toc405673619)

[2.2 Przegląd innych algorytmów optymalizacji lokalnej 14](#_Toc405673620)

[2.3 Przegląd innych metod optymalizacji globalnej 14](#_Toc405673621)

[2.4 Charakterystyki użytych języków 15](#_Toc405673622)

[2.4.1 Język R 15](#_Toc405673623)

[2.4.2 Język Haskell 15](#_Toc405673624)

[2.5 Programowanie równoległe 16](#_Toc405673625)

[2.5.1 Działanie programu zrównoleglonego na CPU 16](#_Toc405673626)

[2.5.2 Działanie programu zrównoleglonego na GPU 17](#_Toc405673627)

[**Rozdział 3 Środowiska.** 18](#_Toc405673628)

[3.1 Opis stanowiska oraz środowiska 18](#_Toc405673629)

[3.2 Środowisko Microsoft Azure 18](#_Toc405673630)

[**Rozdział 4 Badania** 20](#_Toc405673631)

[4.1 Przebieg oraz badanie implementacji w języku R 21](#_Toc405673632)

[4.2 Przebieg oraz badanie implementacji w języku Haskell 22](#_Toc405673633)

[4.3 Badanie implementacji w języku Haskell zrównoleglone na CPU 23](#_Toc405673634)

[4.3.1 Algorytm na maszynie Azure 24](#_Toc405673635)

[**Rozdział 5 Podsumowanie** 25](#_Toc405673636)

[5.1 Dodatkowe uwagi oraz plany 25](#_Toc405673637)

[**Bibliografia** 26](#_Toc405673638)

[**Spis rysunków** 27](#_Toc405673639)

*Chciałbym podziękować mojemu promotorowi za prowadzenie tej pracy oraz wszystkim innym którzy wspierali mnie przy pisaniu pracy inżynierskiej, dodatkowo dziękuje użytkownikom kanału IRC #haskell-beginners za udzielone odpowiedzi podczas pisania programu.*

# **Rozdział 1 Wstęp**

Żyjemy w czasach w których generujemy przeogromne ilości danych, głównym czynnikiem jest powszechny dostęp do Internetu. Dodatkowo coraz bardziej wszelakie instytucje udostępniają swoje wielkie zbiory danych. 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 (1) oraz poszukać hrabstwa w którym jest najmniejsze zużycie prądu. W taki oto sposób narodziła się nowa dziedzina w świecie programowania która jest 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 przeciągu miesiąca lub wpływ mediów społecznościowych na zebrane pieniądze przez fundacje charytatywne. 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 rozwiązywaniu takich modelów matematycznych opartych na „big data” jest na pewno optymalizacja.

Optymalizacja towarzyszyła człowiekowi od bardzo dawna, prawie zawsze człowiek chciał dany problem zminimalizować lub zmaksymalizować. 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 codziennym zwiększaniu mocy obliczeniowej która jest pomocna przy rozwiązywaniu bardzo złożonych problemów.

Od kilku lat promowany jest taki byt który się nazywa „cloud computing” jest on pomocnym narzędziem przy tematyce optymalizacji oraz wynajmowania wysokiej klasy sprzętu komputerowego, są to wielkie centra obliczeniowe w których za pomocą wirtualizacji systemów operacyjnych jesteśmy w stanie wynająć sobie wirtualną maszynę o znacznej mocy obliczeniowej na czas rozwiązywania problemu, takie rozwiązanie jest o wiele tańsze niż inwestowanie potężnych środków pieniężnych na budowanie swoich rozwiązań.

## 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 opisany w pracy Sid-a Yakowitz-a (2). Dodatkowo algorytm ten zostanie uruchomiony na wirtualnej maszynie z szesnastoma procesorami w usłudze Microsoft Azure.

## 1.2 Zarys problemu

W idealnym świecie prawdopodobnie zawsze posiadalibyśmy informacje o danym przedmiocie, prawdopodobnie wszystko było by ciągłe i różniczkowalne, jednak w rzeczywistym świecie nie zawsze możemy posiadać tyle informacji o tak dobrej jakości, często odczytane dane występują z różnego rodzaju zakłóceniami, deformacjami (szumy), wtedy to komplikuje wygląd danych jak i późniejsze rozwiązanie danego problemu.

Załóżmy ze chcemy znaleźć globalne minimum funkcji dla której posiadamy pomiary funkcji w pewnej dziedzinie , która jest podzbiorem . Pierwszym problemem jest to że powyższy problem może nie posiadać rozwiązania. Po pierwsze funkcja może nie posiadać minimum w (rozważmy ), a jeśli minimum istnieje może nie być unikalne (rozważmy ), a nawet jeśli posiada unikalne minimum to może być prawie nie możliwe znalezienie dokładnej wartości, jednakże mamy nadzieje zaproksymować tę wartość. (3).

Dzięki badanią Profesora Sid Yakowitz (3), (2) jesteśmy w stanie rozwiązać problem, w dość szybki sposób. Metoda która została opisana (2) jest o wiele mocniejsza niż przeszukiwania deterministyczne, oznacza to że jest to najlepsze metoda dla najgorszych przypadków (nieciągłość, wielo-wymiarowość oraz multimodalność) ale najgorsza z możliwych przy najlepszych przypadkach (ciągłość, unimodularność).

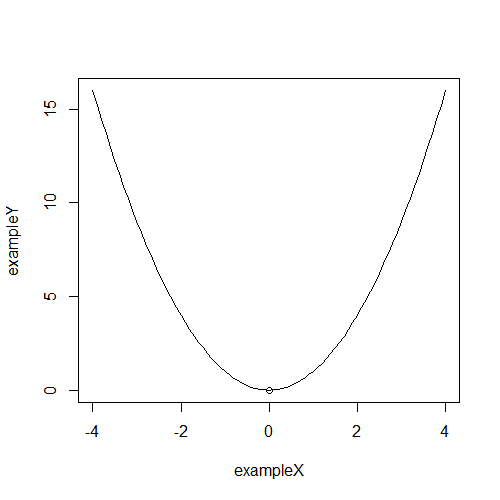
# **Rozdział 2 Wprowadzenie teoretyczne**

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

Problem ten można opisać w następujący sposób.

|  |  |  |
| --- | --- | --- |
|  |  | (2.1) |

Należy znaleźć taki że dla każdego zachodzi następująca nierówność . 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. (4)

Dodatkowo optymalizacje możemy podzielić na dwie zasadnicze grupy, programowanie liniowe oraz programowanie nieliniowe (4). 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**

|  |  |
| --- | --- |
| – funkcja celu  - przestrzeń globalnej optymalizacji  – podzbiór w czasie  – losowy punkt generowany na etapie RS z całej  – optymalny punkt w czasie  – kandydat  – ilość iteracji  (n – numer iteracji)  dla | (2.2) |

**Inicjalizacja**

|  |  |
| --- | --- |
| (n – numer iteracji)  dla | (2.3) |

**Procedura**

1. Jeśli , pobierz nowy losowy punkt
   1. Wykonaj K-W dla oraz i wynik przypisz do
   2. Jeśli , wtedy
2. Jeśli zwróć
3. Zakończ

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

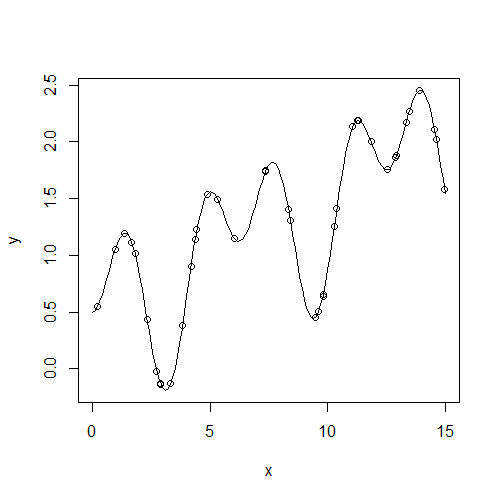
### 2.1.1 Random Search

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 pozycje w którą możemy próbkować jako .

Niech będzie funkcją celu którą chcemy zoptymalizować. Niech wtedy RS możemy opisać następującym pseudo-kodem.

* Wygeneruj z dziedziny w której chcemy zoptymalizować
* Powtarzaj poniższe kroki dopóki nie będzie spełniony warunek zakończenia poszukiwania, może to być limit iteracji lub błąd bezwzględny.
  + , (3)
* Teraz jest naszym rozwiązaniem



Rysunek 2 Losowe punkty wygenerowane dla RS w których będzie przeszukiwane minimum lokalne

### 2.1.2 Kiefer – Wolfowitz

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

Niech będzie funkcją która posiada minimum w punkcie , zakładamy że jest nieznane, jednak z pewniej obserwacji możemy obliczyć ekstremum.

Struktura algorytmu jest podobna do algorytmów gradientowych. Ogólny opis algorytmu możemy zapisać jako następujący ciąg.

|  |  |
| --- | --- |
|  | (2.4) |
|  | (2.5) |

Aktualna implementacja algorytmu w projekcie inżynierskim nieco się różni od ogólnego zapisu, zmieniliśmy parametry .

**Dane**

|  |  |
| --- | --- |
|  | (2.6) |

**Inicjalizacja**

|  |  |
| --- | --- |
|  | (2.7) |

**Procedura**

1. Jeśli , wykonuj
   1. Jeśli , wtedy
2. Jeśli , zwróć

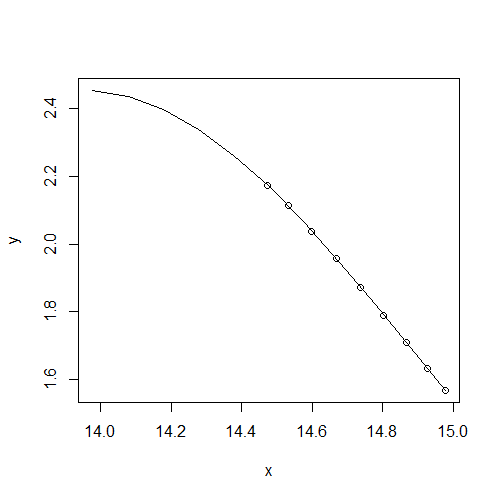
Jak już wcześniej zostało wspomniane parametry różnią się od ogólnego zapisu, w tym przypadku mamy dowolność i sami możemy sobie dobierać te parametry w zależności od wyglądu funkcji.

Parametry te muszą spełniać dwa następujące warunki

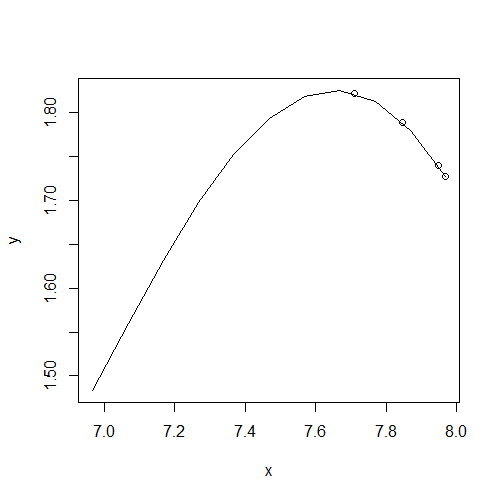
|  |  |
| --- | --- |
|  | (2.8) |

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

|  |  |
| --- | --- |
|  | (2.9) |



Rysunek 3 Prawidłowe znalezienie minimum lokalnego przez algorytm KW



Rysunek 4 Błędne znalezienie minimum lokalnego przez algorytm KW

## 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 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

## 2.3 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. (5)

* 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. (5)

## 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 wydanym 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 rodzimy języków funkcyjnych, sam haskell czysto funkcyjny. 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 nie mamy możliwości zmieniać stanów istniejących zmiennych musimy wynik zwrócić jako nowy obiekt.

Ponadto kompilator GHC wspiera pisanie programów równolegle używając pamięci dzielonej oraz współbieżne (6), na potrzeby mojego projektu inżynierskiego algorytm będzie napisany równolegle.

## 2.5 Programowanie równoległe

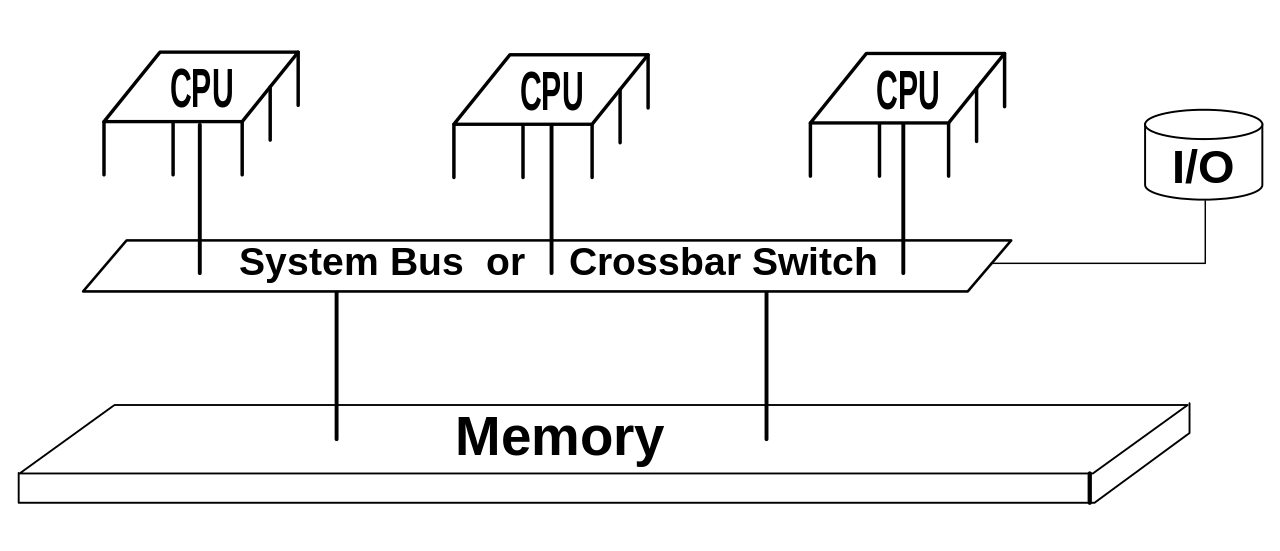
Jak zostało wspomniane wcześniej Haskell wspiera programowanie równoległe trzeba nadmienić że jest możliwość programowanie równoległego dla dwóch typów procesorów

* 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

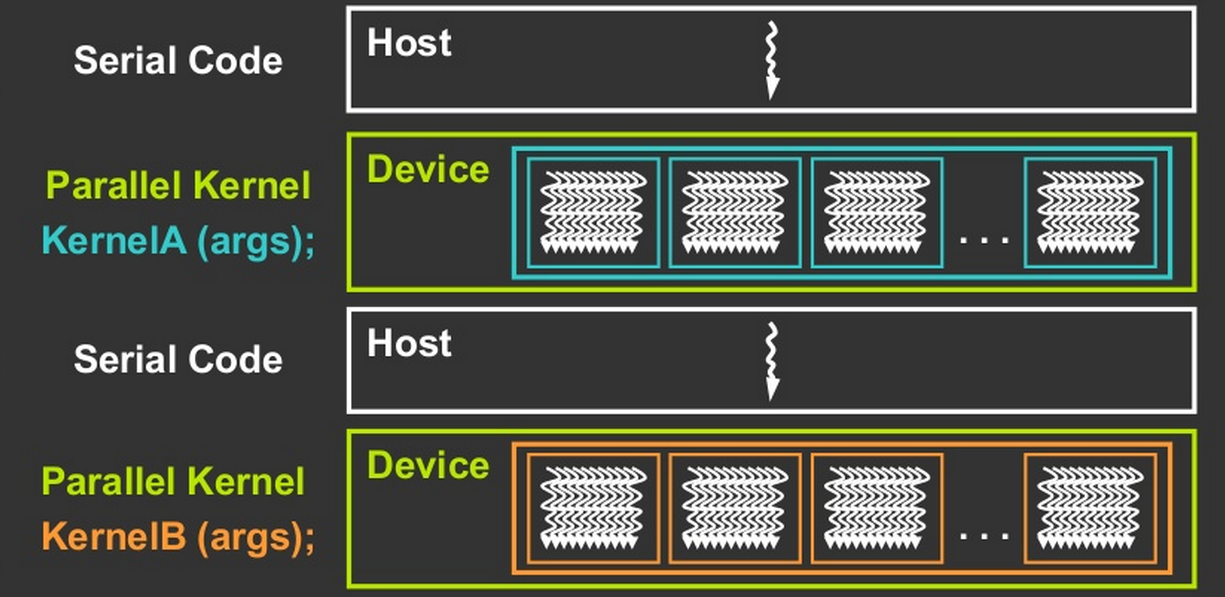
Zazwyczaj pamięć dzielona jest to pamięć do której jest możliwość jednoczesnego korzystania przez wiele procesorów, gdy mówi się o pamięci dzielonej najczęściej ma się namyśli ją jako duży blok danych zaalokowany w pamięci RAM, do którego dostęp może posiadać wiele procesorów tak jak jest to ukazane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

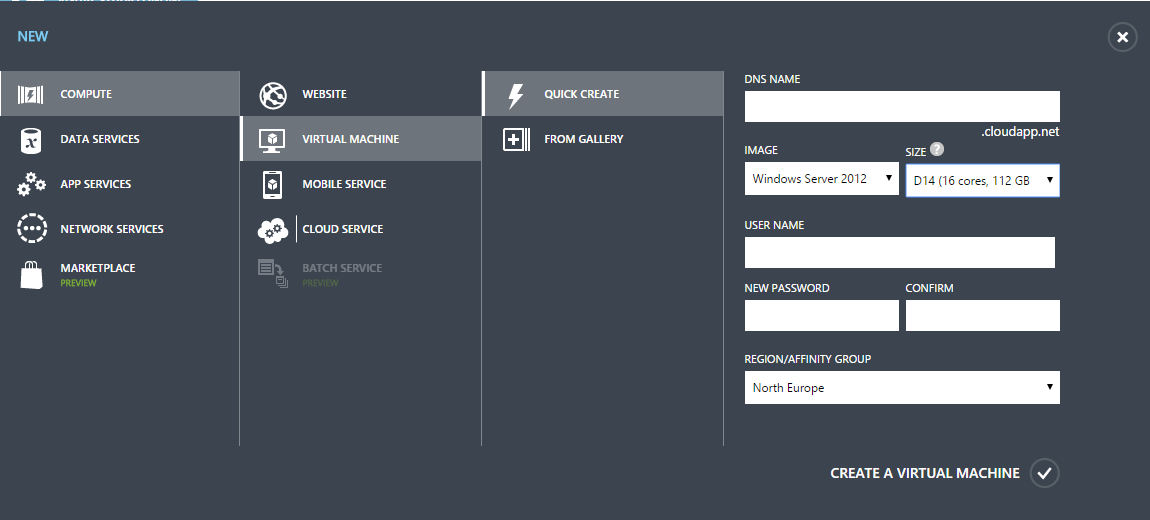
## 3.2 Środowisko Microsoft Azure

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 zadecydować 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.



Rysunek 7 Proces tworzenia wirtualnej maszyny - Microsoft Azure

Proces tworzenia wirtualnej maszyny jest bardzo prosty, wystarczy przejść do panelu zażądania 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 konfiguracje 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.

|  |  |
| --- | --- |
|  | (4.1) |
|  | (4.2) |
|  | (4.3) |
|  | (4.4) |
|  | (4.5) |
|  | (4.6) |

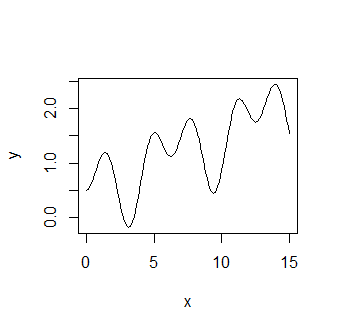
RandomPoints – jest to zmienna która określa nam ile losowych punktów będzie wygenerowanych dla poszukiwania losowego.

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.



Rysunek 8 Wizualizacja badanej funkcji

## 4.1 Przebieg oraz badanie implementacji w języku R

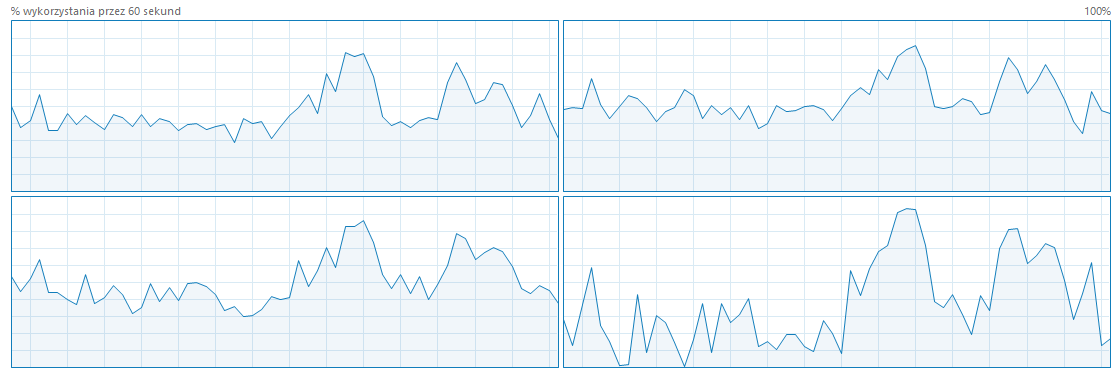
W pierwszej kolejności algorytm został napisany w języku R, ponieważ bardzo szybko możną przenieść idee z papieru na komputer w tym języku, łatwo się prototypuje.

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

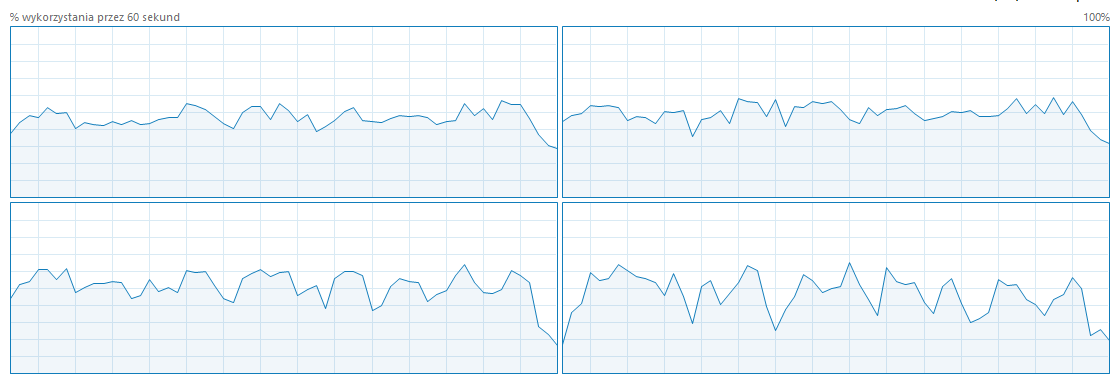
Jak można zauważyć nasze procesory nie są równomiernie wykorzystywane dodatkowo tylko jeden procesor pracuje w tym czasie.

## 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.

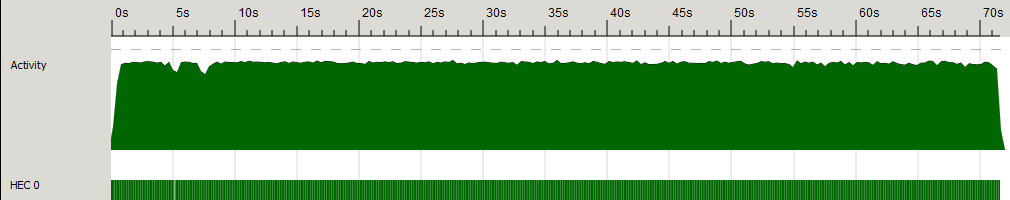
Po zakończeniu przepisywania algorytmu zostały przeprowadzone badania nad efektywnością implementacji, tak samo jak miało miejsce przy poprzednim punkcie.



Rysunek 10 Wykorzystanie zasobów procesorów - Haskell, bez zrównoleglenia

Czas wykonywania 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 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”

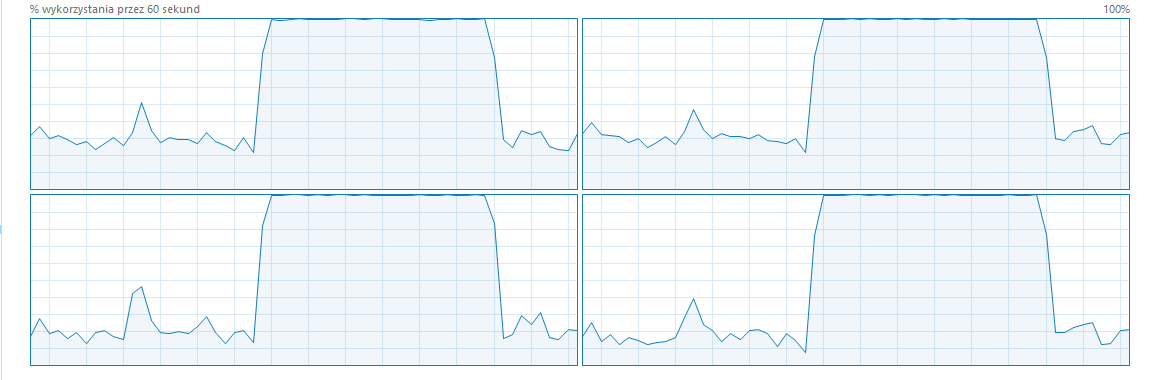
Pierwszym zadaniem przy zrównoleglaniu programy to załadowanie modułu „Control.Parallel” oraz „Control.Parallel.Strategies” są one częścią modułu „Parallel”.

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

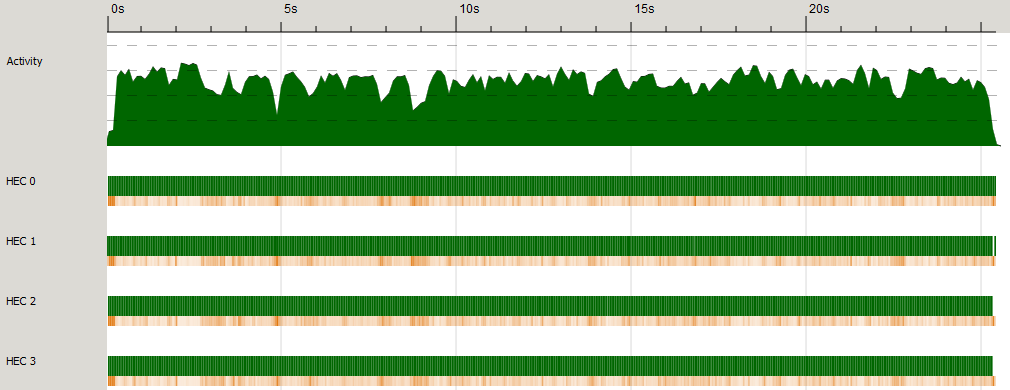
Naszym celem jest równoległe obliczanie minimum lokalnego w wygenerowanych losowo punktach.

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 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 osiągane 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 (2) 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 (7) 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 prace.

# **Bibliografia**

1. **Change Energy and Climate.** *data.gov.uk.* [Online] 11 Grudzień 2011. http://data.gov.uk/dataset/energy\_consumption\_in\_the\_uk.

2. *A globally convergent stochastic approximation.* **Yakowitz, Sid.** 1, January 1993, SIAM J. Control and Optimization, Vol. 31, pp. 30-40.

3. **Devroye Luc i Krzyzak Adam.** Random Search Under Additive Noise. *Modeling Uncertainty.* 2005, 19.

4. **Wikipedia. *Optymalizacja.* [Online] http://pl.wikipedia.org/wiki/Optymalizacja\_(matematyka).**

**5. *Evolutionary Computation and Global Optimization.* Niewiadomska-Szynkiewicz Ewa. Potok Złoty : brak nazwiska, 1999. Przegląd Metod Optymalizacji Globalnej.**

**6. haskell.org. [Online] https://downloads.haskell.org/~ghc/7.0-latest/docs/html/users\_guide/lang-parallel.html.**

**7. Random Search. *Wikipedia.* [Online] http://en.wikipedia.org/wiki/Random\_search.**

# **Spis rysunków**

[Rysunek 1 Przykładowa funkcja z minimum w punkcie 0 6](#_Toc405673380)

[Rysunek 2 Losowe punkty wygenerowane dla RS w których będzie przeszukiwane minimum lokalne 10](#_Toc405673381)

[Rysunek 3 Prawidłowe znalezienie minimum lokalnego przez algorytm KW 12](#_Toc405673382)

[Rysunek 4 Błędne znalezienie minimum lokalnego przez algorytm KW 13](#_Toc405673383)

[Rysunek 5 Współdzielony dostęp do pamięci przez trzy procesory 16](#_Toc405673384)

[Rysunek 6 Zasada przetwarzania danych przez GPU 17](#_Toc405673385)

[Rysunek 7 Proces tworzenia wirtualnej maszyny - Microsoft Azure 19](#_Toc405673386)

[Rysunek 8 Wizualizacja badanej funkcji 20](#_Toc405673387)

[Rysunek 9 Wykorzystanie procesora podczas uruchomionego programu napisanego w R 21](#_Toc405673388)

[Rysunek 10 Wykorzystanie zasobów procesorów - Haskell, bez zrównoleglenia 22](#_Toc405673389)

[Rysunek 11 Wykorzystanie zasobów procesora – threadscope 22](#_Toc405673390)

[Rysunek 12 Wykorzystanie zasobów procesorów przy zrównolegleniu 23](#_Toc405673391)

[Rysunek 13 Wykorzystanie zasobów procesora - threadscope 24](#_Toc405673392)

[Rysunek 14 Zużycie zasobów procesorów podczas wykonywania algorytmu- 16 procesorów 24](#_Toc405673393)