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

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

„vires in numeris”

Spis treści

Rozdział 1 Wstęp

Żyjemy w czasach w których generujemy przeogromne ilości danych, głównym czynnikiem jest powszechny dostęp do Internetu oraz coraz mocniejsze przenikanie technologii w codzienne rzycie człowieka. Ponadto coraz chętniej wszelakie instytucje udostępniają swoje wielkie zbiory danych, ostatnim czasy CERN opublikował zbiory danych które otrzymali podczas poszuiwań nad bozonem Higgsa (1). Niegdyś akwizycje i przetwarzanie danych przeprowadzano w dużych przedsiębiorstwach lub na uczelniach, dzisiaj każdy może pobrać dowolne dane z Internetu na przykład zużycie prądu w Wielkiej Brytanii (2) 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 model matematyczny 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 a co za tym idzie, potrzebujemy coraz to większych zasobów do obliczeń.

Jedną z pomocnych nauk przy rozwiązywaniu takich problemów opartych na „big data" jest na pewno optymalizacja która posiada różnorakie algorytmy optymalizacyjne.

Optymalizacja towarzyszyła człowiekowi od bardzo dawna, 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 i fizyce kwantowej, 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 ponieważ dla bardzo złożonych modeli będziemy potrzebowali bardzo

dużej mocy obliczeniowej aby rozwiązać dany problem w rozsądnym czasie. Dodatkowo pozwala na wynajmowanie wysokiej klasy sprzętu komputerowego, 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 należy zbadać wydajność kilku implementacji algorytmu oraz wyciągnąć wnioski na temat zrównoleglania algorytmów posiadających multi-start. Algorytm zostanie zaprogramowany w dwóch językach R oraz Haskell. Został wybrany algorytm opisany przez Sid Yakowitz (3). Implementacja wykonana będzie tylko dla funkcji jedno wymiarowej. Ponadto algorytm ten zostanie uruchomiony na wirtualnej maszynie z szesnastoma procesorami w usłudze Microsoft Azure aby ukazać potęgę programowania równoległego.

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 problemu.

Załóżmy ze chcemy znaleźć globalne minimum funkcji dla której posiadamy pomiary funkcji Q w pewnej dziedzinie X, która jest podzbiorem $\mathbb{R}^d$. Po pierwsze funkcja Q może nie posiadać minimum w X (rozważmy $X = (0,1)\ i\ Q(x) = x$), a jeśli minimum istnieje może nie być unikalne (rozważmy $Q(x) = \sin(x)$), a nawet jeśli posiada unikalne minimum to może być prawie nie możliwe znalezienie dokładnej wartości, jednakże mamy nadzieje zaproksymować tę wartość z pewną dokładnością. (4).

Dzięki badaniom Profesora Sid Yakowitz (4), (3) jesteśmy w stanie rozwiązać problem, w dość szybki sposób. Metoda która została opisana (3) 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, unimodalność).

.

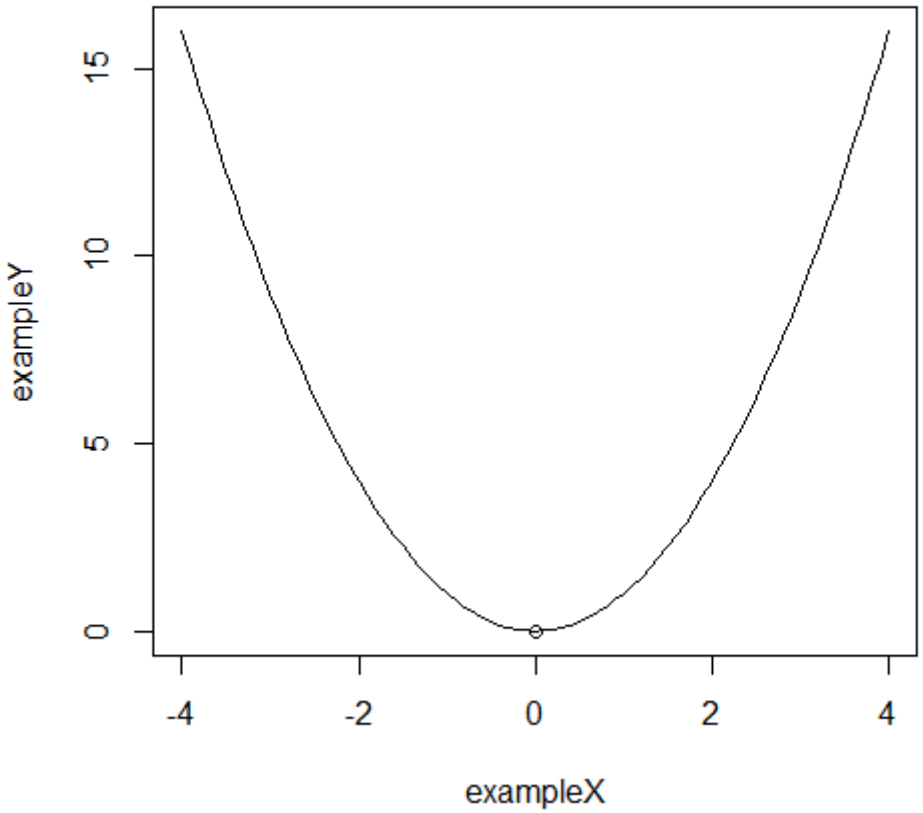
Rozdział 2 Wprowadzenie teoretyczne

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 $x \in A \setminus \{x_0\}$ 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. (5)

Ponadto optymalizacje możemy podzielić na dwie zasadnicze grupy, programowanie liniowe oraz programowanie nieliniowe (5). 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 funkcja celu posiada wiele ekstremów lokalnych.

2.1 Zasady działania algorytmu

Dane

f – funkcja celu

$S \subset \mathbb{R}^n$- 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 $n = 0$ (2.2)

Inicjalizacja

$n = 1$ (n – numer iteracji)

$x_0 = 0$ dla $n = 0$ (2.3)

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

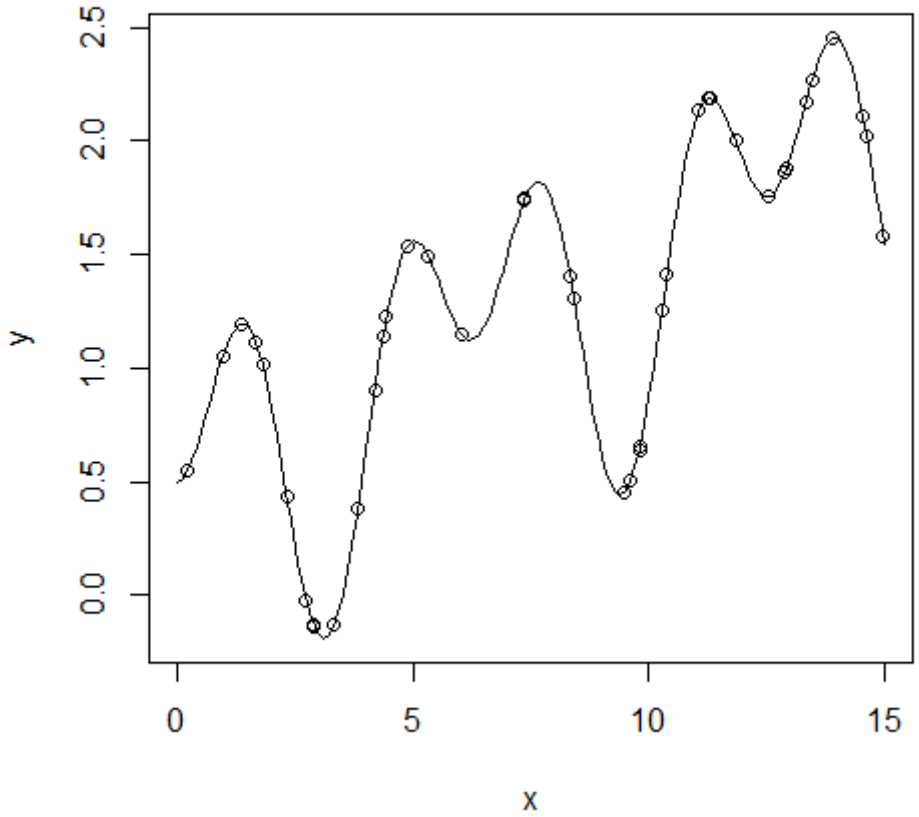
W następnych punktach są opisane szczegółowo algorytmy Random Search oraz Kiefer – Wolfowitz.

2.1.1 Random Search

Random Search (dalej RS) – Jest to metoda optymalizacji która nie wymaga gradientu, dodatkowo RS można zastosować przy funkcjach które nie są ciągłe oraz różniczkowalne. RS działa na zasadzie iteracyjnego posuwania się na lepszą pozycje w przestrzeni 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 zmienną losową X_0 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.
    - $X_{n+1} = \begin{Bmatrix} X_{n+1} & Jeśli\ f(X_{n+1}) < f(X_n) \\ X_n & inaczej \end{Bmatrix}$, (4)
- X_n 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 $M(x)$ 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 algorytmów gradientowych. 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

$$c_n = \frac{1}{n+1}$$
$$a_n = \frac{2}{n^{\frac{1}{6}}}$$
$$N - ilość\ kroków\ K - W \tag{2.6}$$

Inicjalizacja

$$n = 1$$
$$x_1 \in \mathbb{R} \tag{2.7}$$

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 $x_{n+1} < \min R_d = \min R_d \ \vee\ x_{n+1} > \max R_d = \max R_d$

2. Jeśli $n > N$, zwróć x_N

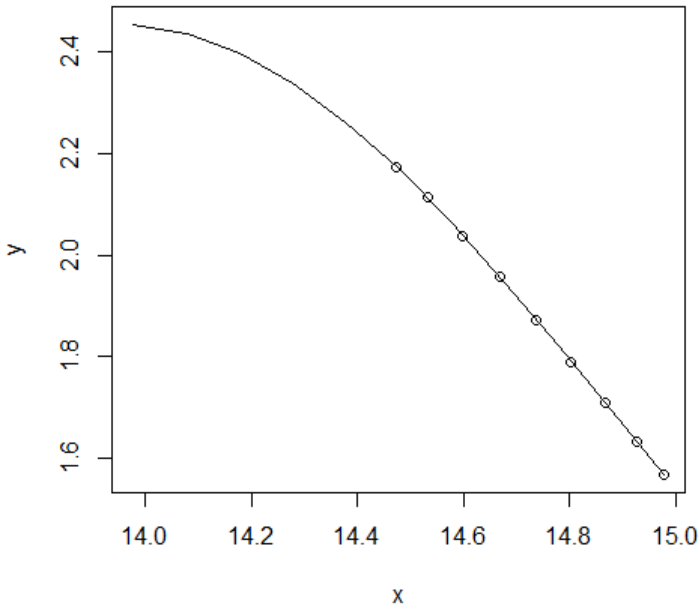
Jak już wcześniej zostało wspomniane parametry $\{c_n\}\ oraz\ \{a_n\}$ różnią się od ogólnego zapisu, w tym przypadku mamy dowolność i sami możemy sobie dobierać te parametry w zależności od wyglądu funkcji.

Parametry te muszą spełniać dwa następujące warunki

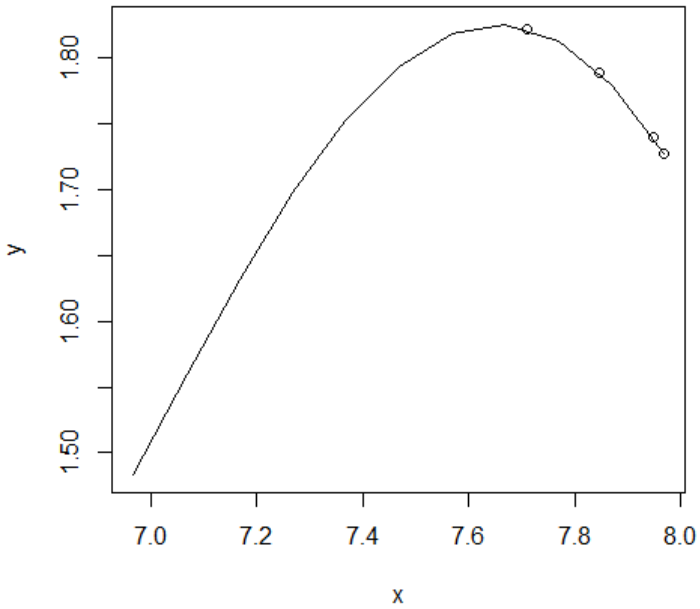
$$c_n \to 0, a_n \to 0\ gdy\ n \to \infty$$
$$\sum_{n=0}^{\infty} a_n = \infty, \sum_{n=0}^{\infty} \frac{a_n^2}{c_n^2} < \infty \tag{2.8}$$

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

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



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

Nasz algorytm składa się w zasadzie z dwóch innych algorytmów, Random Search oraz Kiefer-Wolfowitz, w takim przypadku algorytm jest dość mocno modularny i możemy dowolnie zmieniać metodę poszukiwania minimum lokalnego.

W tym miejscu należy wymienić pozostałe rodzimy algorytmów optymalizacji lokalnej każda z rodziny posiada co najmniej kilka algorytmów także jest tutaj bardzo duże pole do ewentualnych badań. (6)

- 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. Metody te można podzielić zasadniczo na dwie grupy, metody deterministyczne oraz niedeterministyczne, gdzie operacje losowe są istotnym elementem. (7)

- 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. (7)

2.4 Charakterystyka 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. 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 równolegle używając pamięci dzielonej oraz współbieżne (8), na potrzeby mojego projektu inżynierskiego algorytm będzie napisany równolegle.

2.5 Programowanie równoległe

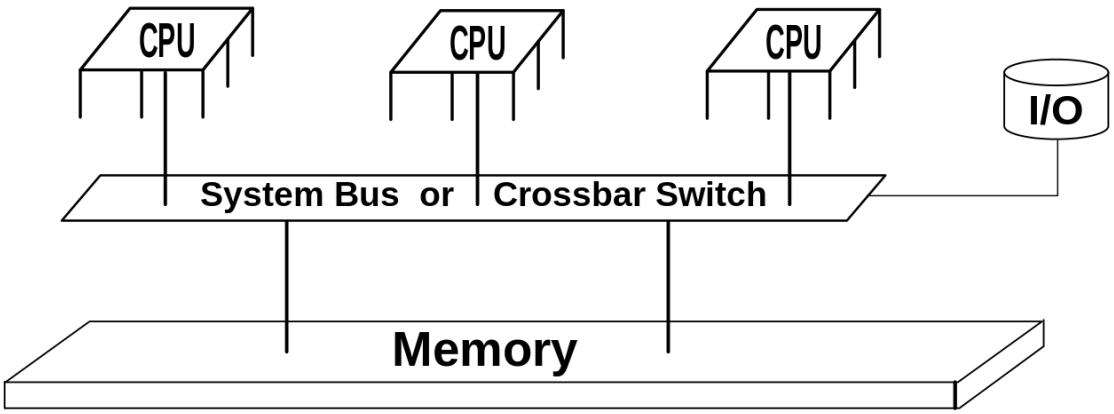
Haskell wspiera programowanie równoległe na dwóch typach procesorów

- CPU
- GPU

Programowanie równoległe można traktować jako kontrargument na aktualne limity w częstotliwości taktowania procesora, oraz brak postępu w tej dziedzinie. Programowanie równoległe zapewnia nam przyspieszenie obliczeń sekwencyjnych oraz w pełni wykorzystanie zasobów komputera, od kilku lat standardem produkcyjnym są komputery posiadające wiele procesorów z wieloma rdzeniami. Jednak programowanie równoległe jest bardzo kosztownym przedsięwzięciem dlatego bardzo mała ilość firm decyduje się na taki krok, w naszym przypadku jest to bardziej wymóg aniżeli przywilej.

2.5.1 Charakterystyka programu zrównoleglonego na CPU

Przy programowaniu równoległym na CPU, gdzie jedyną jednostką obliczeniową jest procesor komputerowy, należy wspomnieć o architekturze „Shared Memory Multiprocessors" (SMP), jest to współdzielona pamięć wykorzystywana przez wiele procesorów



Rysunek 5 Współdzielony dostęp do pamięci przez trzy procesory

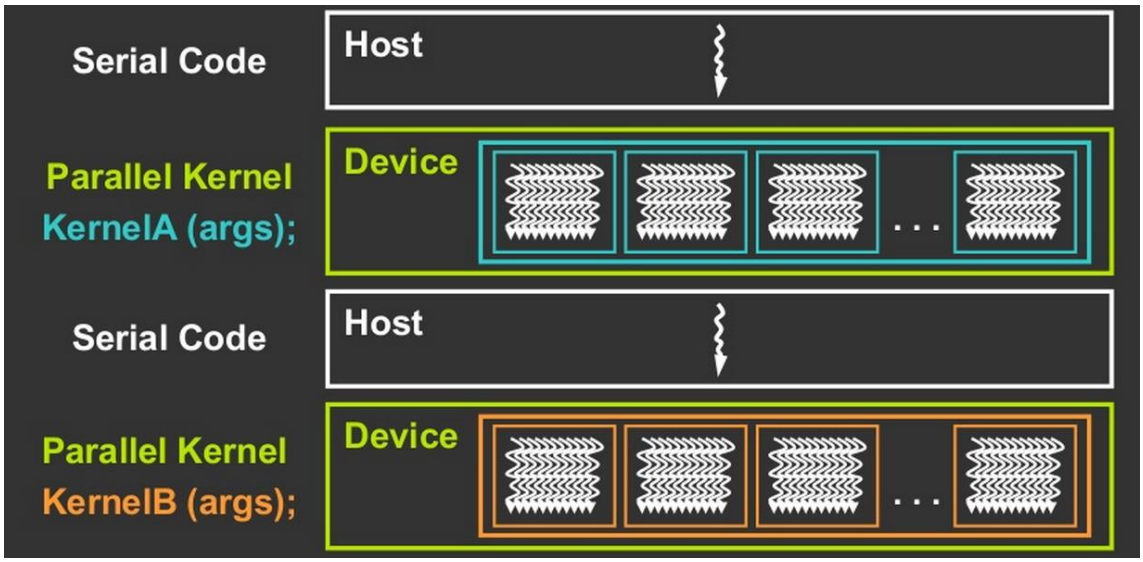
Pamięć dzielona jest to pamięć do której jest możliwość jednoczesnego odwoływania się przez wiele procesorów, gdy mówi się o pamięci dzielonej najczęściej

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 zastosowanie pamięci dzielonej ogranicza nas to do kilkudziesięciu procesorów, gdy w przypadku pamięci rozproszonej tych procesorów może być nawet kilka tysięcy.

2.5.2 Charakterystyka 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.

Użycie karty graficznej do obliczeń idealnie pasują kiedy mam bardzo duże ilości danych na których chcemy wykonać jakieś obliczenia, na przykład chcemy wykonać projekcje na milionie elementów w tablicy.

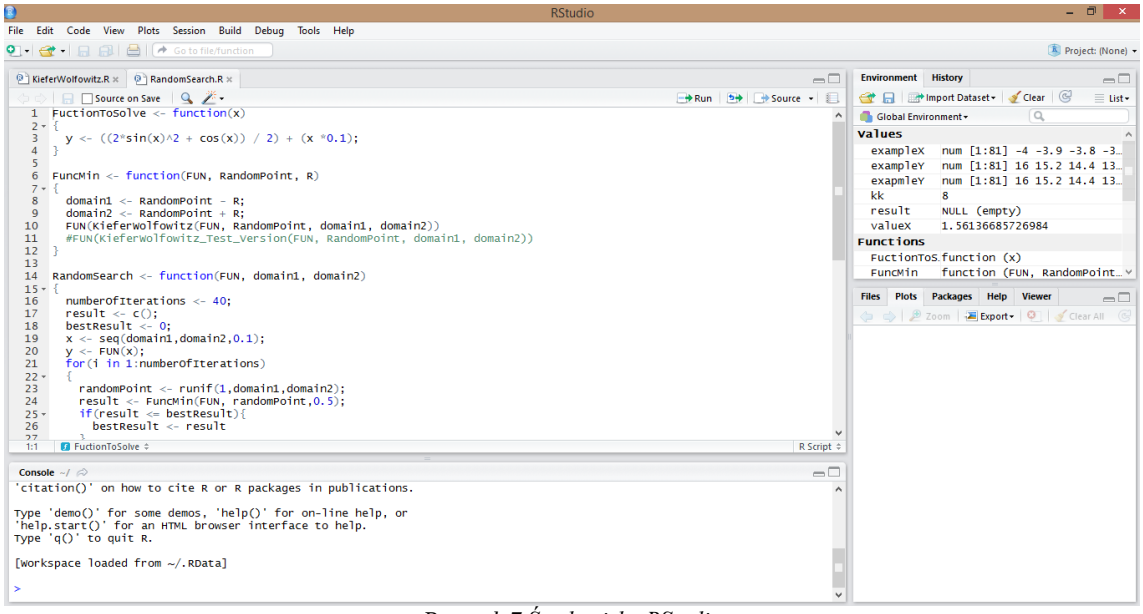
Rozdział 3 Środowiska badawcze

3.1 Opis stanowiska

Cała praca oraz badania zostały wykonane na osobistym laptopie o następującej konfiguracji :

- Procesor - AMD A8-3500M
- Pamięć RAM - 6 GB 1333 MHz
- Dysk twardy - TOSHIBA MK6459GSXP SATA
- System operacyjny - Windows 8

Podczas pisania programu w języku R zostało użyte środowisko o nazwie „RStudio” które pozwoliło na dość szybką implementacje algorytmu oraz pomaga przy zarzadzaniu różnymi bibliotekami tego języka.



Rysunek 7 Środowisko RStudio

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.

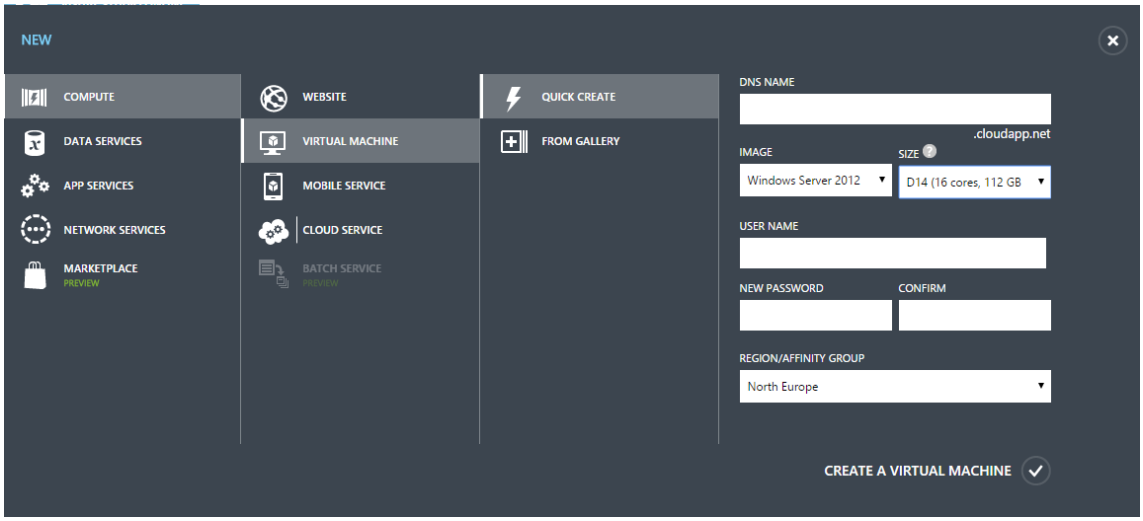
Rysunek 8Program threadscope

3.2 Microsoft Azure

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 nie musimy się martwić o infrastrukturę, możemy jedynie zadecydować jaki system operacyjny oraz jakie programy będą zainstalowane 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 następującej konfiguracji :

- 16 procesorów logicznych
- 112GB pamięci RAM,
- System operacyjny Windows Server 2012.



Rysunek 9 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 „Virtual Machines" i wypełniamy puste pola oraz wybieramy z dostępnej listy system operacyjny oraz konfiguracje maszyny. Na zakończenie tego podpunktu należy wspomnieć także o innych usługodawcach, Amazon Web Services (AWS) oraz Octawave. Microsoft Azure został wybrany ze względu na wcześniejsze doświadczenie z tą usługą, aczkolwiek da się uzyskać identyczne efekty używając innych usług.

Rozdział 4 Badania

Podczas badania wydajności algorytmu, eksperyment był przeprowadzany przy następująco skonfigurowanych parametrach.

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

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

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

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

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

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

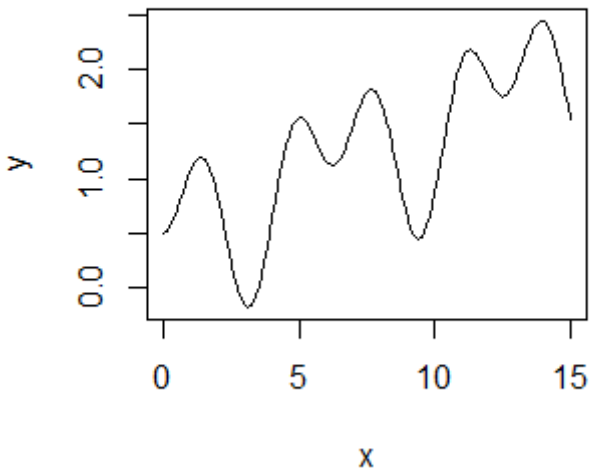
RandomPoints –zmienna która określa nam ile losowych punktów będzie wygenerowanych dla poszukiwania losowego.

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

lGlobalDoman oraz rGlobalDomain – zakres dziedziny dla poszukiwania minimum globalnego.

kwSteps – ilość kroków dla algorytmu Kiefer – Wolfowitz.

functionToSolve – wzorcowa funkcja w której poszukujemy minimum globalne.



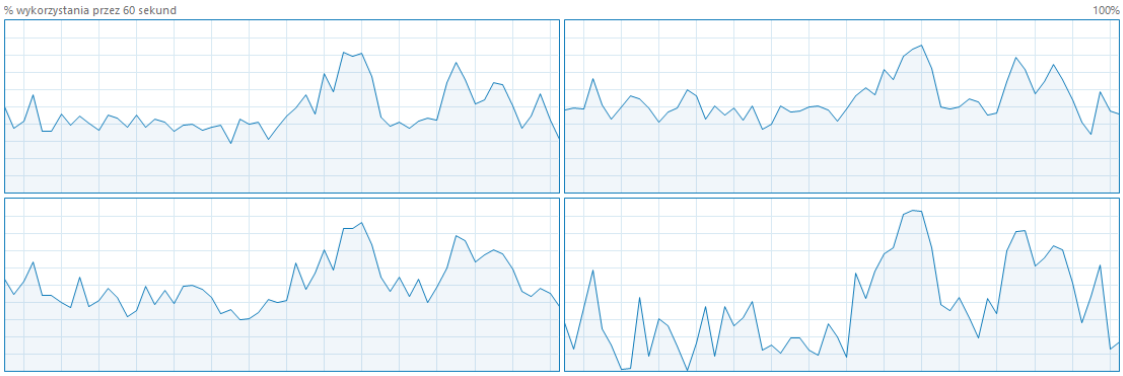
Rysunek 10 Wizualizacja badanej funkcji

4.1 Przebieg badania programu napisanego 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. Kiedy algorytm został ukończony, zostały przeprowadzone badania nad wydajnością algorytmu. Aby zmierzyć czas wykonywania funkcji, została użyta do tego wbudowana funkcja system.time(FUNC) która jako argument przyjmuje inną funkcje.

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



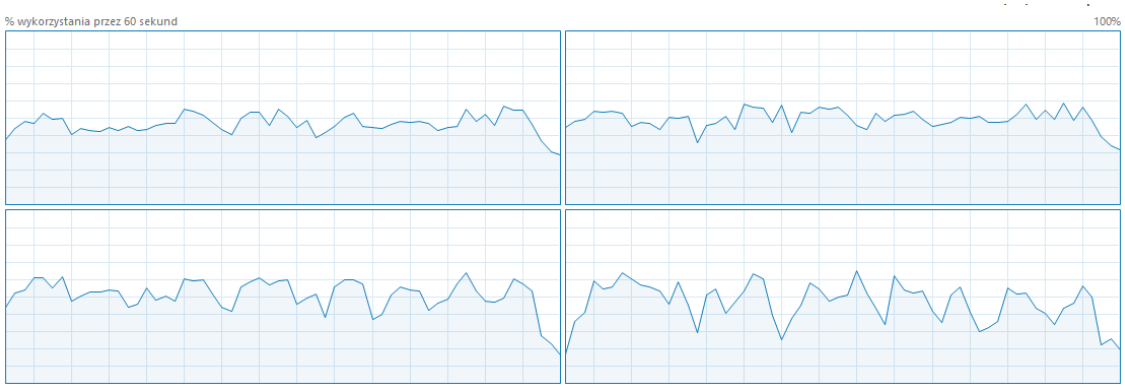
Rysunek 11 Wykorzystanie procesora podczas uruchomionego programu napisanego w R

Jak można zauważyć nasze procesory nie są równomiernie wykorzystywane dodatkowo tylko jeden procesor najmocniej jest obciążony.

4.2 Przebieg badania programu napisanego w języku Haskell

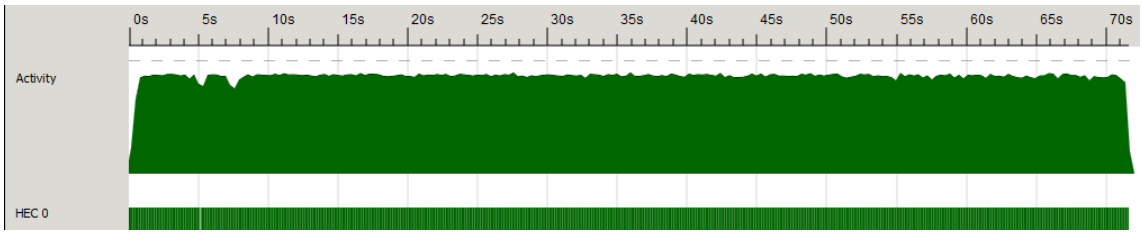
Kiedy już został napisany algorytm w języku R, następnym etapem było przepisanie z prototypowanego algorytmu na język Haskell. Zostały napotkane pewnie problemy głównie przez to iż programowanie funkcyjne bardzo mocno różni się od programowania imperatywnego, potrzebna jest zmiana myślenia oraz pojmowania problemu na temat rekurencji. Kiedy pojmie się już zasady programowania w języku funkcyjnym pierwszym z widocznych rezultatów pisania 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 12 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. Następniy wykres pochodzi z narzędzia „threadScope".



Rysunek 13 Wykorzystanie zasobów procesora – threadscope

Aktywność (ang. activity) oznacza 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 Przebieg badanie programu napisanego w języku Haskell oraz zrównoleglone na CPU

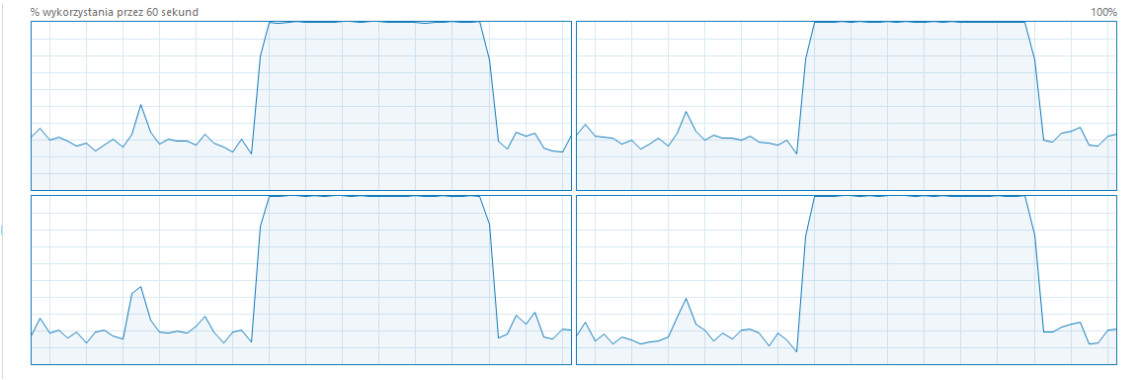
Oby zrównoleglić algorytm, został użyty moduł „Parallel”, który został zainstalowany poprzez program „cabal”. Kompilator pozwala nam na zrównoleglenie naszego programu, dzięki zastosowaniu architektury „Shared Memory Multiprocessors”, czyli wykorzystanie dwóch lub więcej procesorów do jednoczesnego wykonywania zadań na dzielonej pamięci, wtedy procesory są równo obciążone pracą a czas dostępu do danych jest taki sam.

Trzeba przepisać 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”

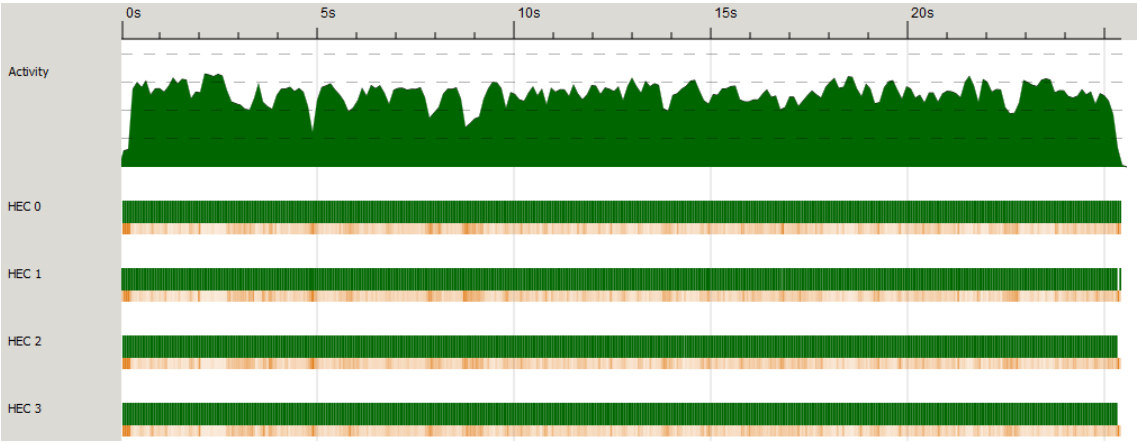
Pierwszym zadaniem przy zrównoleglaniu programu to załadowanie modułu „Control.Parallel” oraz „Control.Parallel.Strategies” są one częścią modułu „Parallel”. Następnym etapem jest stworzenie funkcji która będzie aplikowała inną funkcje do każdego elementu w liście (w programowaniu nazywa się to mapowaniem), 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. Tutaj także się pojawia pierwsza zmiana względem poprzednich programów, pierwszą rzeczą jaką robimy to generujemy sobie listę N losowych punktów w zadanej przestrzeni a następnie dla każdego z nich uruchamiamy optymalizacje lokalną.

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 14 Wykorzystanie zasobów procesorów przy zrównolegleniu



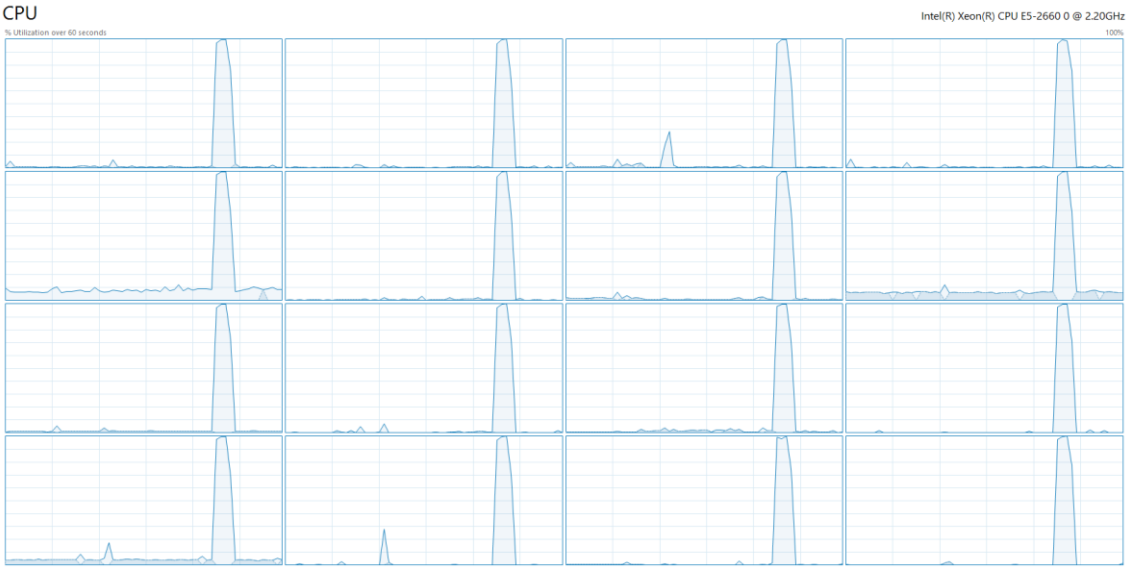
Rysunek 15 Wykorzystanie zasobów procesora - threadscope

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

4.3.1 Przebieg badanie algorytm na wirtualnej maszynie

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 16 Zużycie zasobów procesorów podczas wykonywania algorytmu- 16 procesorów

4.4 Podsumowanie

Tabela 1 Zestawienie wyników badań nad czasem wykonywania się algorytmu

Konfiguracja (Język + ilość rdzeni)	Czas - [s]	Zrównoleglenie	Wartość zmiennej RandomPoints
R + 1 CPU	222,98 s	Nie	$10e^5$
Haskell + 1CPU	71,58 s	Nie	$10e^5$
Haskell + 4CPU	24,41 s	Tak	$10e^5$
Haskell + 16 CPU	2.69 s	Tak	$10e^5$

Z wyników badan wynika że nasz program został prawidłowo zrównoleglony, można było by się zastanowić czy dało by się w jakiś sposób zrównoleglić algorytm Kiefer-Wolfowitz.

Program napisany w R rozwiązuje problem optymalizacji najwolniej, najprawdopodobniej dlatego iż jest to język skryptowy bez kompilatora jednak zakładaliśmy taki wynik na początku więc cel został spełniony ponieważ algorytm został bardzo szybko napisany w przeciągu kilku dni bez martwienia się o szybkość.

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, dopiero zwiększenie problemu o jeden rząd (zwiększenie losowych punktów w RS) ukazuje pracę procesorów na pełnym wykorzystaniu.

Rozdział 5 Podsumowanie

Celem niniejszej pracy była implementacja algorytmu globalnej optymalizacji stochastycznej, metodą opisaną przez Sid Yakowitz (3). W wyniku pracy inżynierskiej zostały wykonane wszystkie założone zadania. Implementacje algorytmu wykonano w oparciu o następujące pracę (4), (3), (9).

Tak jak założyliśmy pierwotnie algorytm został za prototypowany w „R" tak aby jak najszybciej zobaczyć w pełni działający algorytm bez martwienia się o szybkość działania naszej implementacji dla bardziej złożonych problemów. Program ten zrealizowaliśmy jako moduł języka R, co umożliwia pobieranie algorytmu przez innych użytkowników, dodatkowo kod źródłowy jest otwarty i dostępny. (10)

Następnym etapem było przepisanie algorytmu na język kompilowany w naszym przypadku jest to język Haskell, język ten został wybrany ponieważ w językach funkcyjnych o wiele łatwiej programuje się algorytmy czysto matematyczne, tak jak w przypadku poprzedniej implementacji ta też została zrealizowana jako moduł, i jest dostępna dla innych użytkowników oraz kod źródłowy jest otwarty. (11)

Stworzony system został poddany serii testów, i wyniki w większości wypadków pokrywały się z poprzednimi, jednak najlepsze efekty mogą być uzyskane tylko i wyłącznie jeśli przepiszemy nasz algorytm równolegle na GPGPU tak aby wykorzystywał pełne zasoby GPU, prawdopodobnie algorytm przyspieszyłby swoje działanie kilkadziesiąt razy.

5.1 Dodatkowe uwagi oraz dalsze plany rozwoju.

Pierwszy etap większej pracy został zakończony, został zbudowany algorytm optymalizacji stochastycznej dla funkcji jedno wymiarowej, następnym etapem będzie dostosowanie algorytmu do funkcji 2 oraz 3 wymiarowych a na końcu do 9 wymiarów, tak jak zostało założone w pracy Sid Yakowitz (3), algorytm który opisał jest efektywny dla funkcji maksymalnie dziewięciu zmiennych. Następnym etapem była by zmiana implementacji opisanej przez Sid Yakowitza, a dokładniej przejście z czystego „Random Search" do metody która by automatycznie dobierała szerokość przedziału w którym będzie poszukiwane minimum lokalne opisane (12) jako „Adaptive Step Size Random Search". Trzeba także pomyśleć nad możliwością przepisania całego algorytmu tak aby wykonywał się równolegle na procesorach karty graficznej w rezultacie moglibyśmy rozwiązywać o wiele bardziej skomplikowane problemy w dość szybkim czasie.

Bibliografia

1. **CERN.** *opendata CERN.* [Online] http://opendata.cern.ch/.
2. **Change Energy and Climate.** *data.gov.uk.* [Online] 11 Grudzień 2011. http://data.gov.uk/dataset/energy_consumption_in_the_uk.
3. *A globally convergent stochastic approximation.* **Yakowitz, Sid.** 1, January 1993, SIAM J. Control and Optimization, Vol. 31, pp. 30-40.
4. **Devroye Luc i Krzyzak Adam.** Random Search Under Additive Noise. *Modeling Uncertainty.* 2005, 19.
5. **Wikipedia.** ***Optymalizacja.*** **[Online] http://pl.wikipedia.org/wiki/Optymalizacja_(matematyka).**
6. Szlachcic Ewa. ***Instytutu Informatyki, Automatyki i Robotyki.*** **[Online] http://staff.iiar.pwr.wroc.pl/ewa.szlachcic/materialy%20dydaktyczne/eit_studia_2_stopnia/7w_mo_optymalizacja_lokalna.pdf.**
7. ***Evolutionary Computation and Global Optimization.*** **Niewiadomska-Szynkiewicz Ewa. Potok Złoty : brak nazwiska, 1999. Przegląd Metod Optymalizacji Globalnej.**
8. haskell.org. [Online] https://downloads.haskell.org/~ghc/7.0-latest/docs/html/users_guide/lang-parallel.html.
9. Brownlee Jason. ***Clever Algorithms: Nature-Inspired Programming Recipes.*** **2012. strony 30-33.**
10. Sawicz Paweł. ***github.com.*** **[Online] https://github.com/pawelsawicz/Thesis/tree/master/R.Prototype.**
11. —. ***github.com.*** **[Online] https://github.com/pawelsawicz/Thesis/tree/master/Haskell.Implementation.**
12. Random Search. ***Wikipedia.*** **[Online] http://en.wikipedia.org/wiki/Random_search.**

Spis rysunków