1 Przegląd literatury

Rodział pierwszy zawiera przegląd publikacji naukowych, których tematem są różne metody programowania równoległego stosowanego w algorytmach uczenia maszynowego.

1.1 „Parallel Implementation of Decision Tree Learning Algorithms"

Artykuł o tytule „Parallel Implementation of Decision Tree Learning Algorithms" został napisany przez Nuno Amado, João Gama oraz Fernando Silva. W pracy zostało przedstawionych klika strategii konstrukcji algorytmów drzew decyzyjnych, które oparate są o techniki takie jak: równoległość zadań, równoległość danych oraz rownoległość hybrydowa. Następnie w artykule zaprezentowa została autorska implementacja równoległej konstrukcji drzewa decyzyjnego algorytmem C4.5. Na zakończenie autorzy przedstawili wyniki działania algorytmu i postawili wstępne wnioski dotyczące działania algorytmu.

Artykuł rozpoczyna się od zaprezentowania trudności, które pokazują jak złożonym zadaniem jest implementacja równoległych algorytmów do budowy drzew decyzyjnych. Wymienione zostają m.in. problemy z zastosowaniem statycznego jak i dynamicznego przydzielania procesorów. Nieregularny kształt drzewa, który określany jest dopiero w momencie działania programu, jest dużą przeszkodą do stosowania statycznej alokacji. Takie podejście prowadzi najczęściej do nierównomiernego rozłożenia obciążenia. W przeciwnej sytuacji, gdy dane przetwarzane są przez dynamicznie przydzielane procesory, problemem staje się konieczność zaimplementowania przekazywania danych. Współdzielenie danych jest wymagane, ponieważ część danych związnych z rodzicami musi dostępna być również dla potomków.

Autorzy szczegółowo opisują różnice pomiędzy równoległością zadań, równoległością danych oraz równoległością hybrydową. Równoległość zadań określana jest jako dynamiczne rozdzielanie węzłów decyzyjnych między procesory, w celu kontunuowania ich rozbudowy. Wadą takiego podejścia jest konieczność replikacji całego zbioru treningowego lub, alternatywnie, zapewnienie dużej ilości komunikacji pomiędzy procesorami. Równoległość danych przedstawiona jest jako wykonywanie tego samego zbioru instrukcji algorytmu przez wszystkie zaangażowane procesory. Zbiór treningowy zostaje podzielony (pionowo lub poziomo) pomiędzy procesory tak, że każdy z nich

odpowiedzialny jest za inny zestaw przykładów ze zbioru. Autorzy zwracają uwagę, że przetwarzanie z pionowym podziałem danych narażone jest na wystepienie nierównowagi obciążenia. Równoległość hybrydowa scharakteryzowana jest jako połączenie równoległości zadań oraz danych. Dla węzłów, które muszą przetworzyć dużą liczbę przykładów, wykorzystywana jest równoległość danych. W ten sposób unika się problemów związanych z nierównomiernym obciążeniem. W przypadku węzłów z przypisaną mniejszą ilością przykładów, czas potrzebny do komunikacji może być większy niż czas potrzebny do przetwarzania przykładów. Zastosowanie równoległości zadań w takie sytuacji pozwala uniknąć dysproporcji.

W kolejnej cześci artykułu przestawiona została implementacja równoległej konstrukcji drzewa decyzyjnego. Program został stworzony do wykonywania w środowisku pamięci rozproszonej, w której każdy z procesorów ma własną pamięć prywatną. Autorzy zaproponowali takie podejście, ponieważ ma ono rozwiązywać dwa problemy wyspominane na początku pracy: równoważenie obciążenia oraz konieczność przekazywania danych. Każdy z procesorów ma za zadanie tworzyć własne listy atrybutów i klas na podstawie przydzielonych podzbiorów przykładów. Wykorzystanie obydwu list jest kluczem do osiągnięcia efektywnego paralelizmu. Wpisy w liście klas zawierają etykietę klasy, indeks globalny przykładu w zbiorze treningowym oraz wskaźnik do węzła w drzewie, do którego należy dany przykład. Listy atrybutów również zawierają wpis dla każdego przykładu z atrybutem, jak również indeks wskazujący na odpowiadający wpis w liście klas. Każdy procesor znajduje własne, najlepsze podziały lokalnego zbioru dla każdego atrybutu. Nastepnie komunikuje się z pozostałymi procesorami, w celu ustalenia jednego, najlepszego podziału. Po podziale (utowrzeniu węzła), następuje aktualizacja list atrybutów przez każdy procesor, dokonana poprzez rozdzielenie atrybutów w zależności od wartości wybranego atrybutu dzielącego.

Zaprezentowane przez autorów wyniki określone są jako wstępne i wymagające udoskonaleń. Autorzy zdecydowali się jednak na wykorzystanie ich do przewidzenia oczekiwanego zachowania algorytmu. Implementacja wykorzystuje takie same kryteria oceny jak stosowane w algorytmie C4.5, dlatego autorzy skupili się głównie na analizie czasu potrzebnego do zbudowania drzewa. Do wszystkich testów wykorzystany był zestaw danych syntetycznych Agrawal, w którym każdy przykład ma dziewięć atrybutów (pięć ciągłych i trzy dyskretne).

Z przedstawionych wyników testów wynika, że algorytm wykazał dobre wyniki przyspieszania. Twórcy artykułu przeprowadzili również testy mające

na celu sprawdzenie skalowalności. Jak w pierwszym przypadku, testy wykazały, że algorytm osiąga dobre wyniki skalowania.

1.2 „Parallel ensemble learning of convolutional neural networks and local binary patterns for face recognition"

W artykule pt. „Parallel ensemble learning of convolutional neural networks and local binary patterns for face recognition" autorstwa Jialin Tanga, Qinglang Sub, Binghua Sua, Simon Fongc, Wei Caoa oraz Xueyuan Gonga został zaprezentowany sposób pozwalający na rozpoznawanie twarzy. Podejście oparte jest o równoległe uczenie zespołowe lokalnych wzorców binarnych (LBP) oraz konwolucyjnych sieci neuronowych (CNN). Metoda LBP zastosowana została do ekstracji cech tekstury twarzy, które posłużyły jako dane treningowe sieci CNN.

Paralelizm w omawianym przykładzie został uzyskany poprzez zastosowanie równoległego uczenia zespołowego. Kilka konwolucyjnych sieci neuronowych, opartych o różne struktury LBP, zostało wykorzystanych do uczenia się, a nastepnie klasyfikowania zestawów danych treniengowych. Każda sieć kończyła trening podając wynik klasyfikacji. Ostateczny wynik uzyskiwany był na podstawie głosowania większościowego. Wyjściowy wynik sieci o największej liczbie głosów przyjmowany był jako finalny klasyfikator obrazu twarzy.

W celu sprawdzenia skuteczności przedstawionego rozwiązania, autorzy zdecydowali się na wybór dwóch zestwów danych: Yale-B i ORL. Zbiór Yale-B składa się 576 obrazów 38 twarzy o różnych wyrazach, które wykonane zostały w zmiennych warunkach oświetlenia. Zbiór ORL składa się z 40 obrazów 4 twarzy, po 10 zdjęć na osobę. Współczynnik rozpoznawania zdefiniowany został jako stosunek liczby skutecznie rozpoznanych obrazów do całkowitej ilości obrazów w zbiorze testowym. W pracy zintegrowanych zostało 10 sieci neuronowych. Sprzęt na którym przeprowadzonono testy posiadał następujące parametry: CPU Intel(R) Core(TM) i5-.6300HQ 2.3GHZ, pamięć RAM 8G DDR4, karta graficzna NVIDIA GeForce GTX 960M, system operacyjny Windows 10 64 bity oraz środowisko programistyczne Python 3.5 Tensorflow-gpu 1.10.0.

Dokładność zintegrowanych sieci dla zbioru trenigowego ORL wyniosła około 98%. Po przeprowadzeniu 2800 treningów dokładność zbioru testowe-

go zwiekszyła się aż 100%, co daje znacznie lepszy wynik niż w przypadku zastosowania pojedynczej sieci konwolucyjnej. Dowodzi to wysokiej odporności zintegrowanych sieci na zmiany postawy ciała czy oświetlenia. Dla zbioru Yale-B dokładność pojedynczej sieci zbliżona była do 85%, natomiast zintegrowanej sieci wyniosła około 97,5%. Na podstawie wyników przeprowadzonych testów autorzy stwierdzili, że w przypadku, gdy sieć neuronowa nie wykorzystuje schematu uczenia zespołowego do klasyfikacji, wówczas dokładność jest znacznie zmniejszona.

1.3 „Parallel Genetic Algorithms' Implementation Using a Scalable Concurrent Operation in Python"

W artykule o tytule „Parallel Genetic Algorithms' Implementation Using a Scalable Concurrent Operation in Python" napisanym przez Vladislav Skorpil oraz Vaclav Oujezsky zostały zaprezentowane trzy różne implementacje równoległych algorytmów gentycznych(GA): Master-Slave, Coarse-Grained oraz Fine-Grained. Modele zostały zaimplementowane przy użyciu języka programowania Python, brokera wiadomości RabbitMQ oraz modułu „Scalable Concurrent Operations in Python" (SCOOP) umożliwiającego programowanie równoległe.

W pierwszej części artykułu autorzy przedstawiają różnice pomiędzy równoległymi implementacjami a tradycyjną implementacją sekwencyjną. Model Master-Slave w wersji synchronicznej działa prawie tak jak model sekwencyjny. Różni się tylko w przetwarzaniu funkcji Fitness, które rozdzielone jest pomiędzy różne procesory. Model Master-Slave może zwiększyć szybkość GA poprzez równomierne rozłożenie obciążenia między procesorami, mimo konieczności komunikacji pomiedzy nimi. W modelu Fine-Grained tworzona jest jedna globalna populacja rozproszona przestrzennie na węzły (procesory). W ten sposób zostaje utworzona topologia z sąsiedztwami. Sąsiedztwa tworzą przestrzeń, w której odbywa się równoległa i tylko lokalna selekcja (osobnik uczestniczy w selekcji tylko w obrębie sąsiedztwa). Z powodu izolacji sąsiedztwa, najlepsze osobniki rozprzestrzeniają się wolnej niż w innych rodzajach GA, co zwiększa różnorodność populacji. Dodatkowo tylko jeden element centralny w obrębie jednego sąsiedztwa jest poddawany modifikacjom przez krzyżowanie i mutacje. Do zalet tego modelu autorzy zaliczają dużą wydajność. Model Coarse-Grained różni się od modelu Fine-Grained głównie tym, że pracuje on z mniej rozdrobnioną globalną populacją (w przy-

padku Fine-Grained jest to po jeden lub dwa osobniki na węzeł). Autorzy przyjmują zasadę, że model Coarse-Grained występuje wtedy, gdy liczba węzłów jest mniejsza niż liczba osobników w jednym z nich.

Paralelizacja GA została otrzymana poprzez użycie funkcji z modułu SCOOP, natomiast do komunikacji między procesami wybrany został serwer RabbitMQ. Testy pomiędzy poszczególnymi implementacjami a implementacją szeregową zostały przeprowadzaone na następującym sprzęcie: Linux z systemem operacyjnym Fedora wersja 25, CPU Intel(R) Xeon(R) L5408 2.13 GHz z czterema procesorami, pamięć RAM 32 GB z wirtualizacją trzech innych stacji roboczych z systemem operacyjnym Ubuntu 17.10. Rozmiar populacji rozpoczynał się od 64 osobników a nastepnie zwiększany był dla kolejnych testów.

Zgodnie z oczekiwaniami autorów, najmniejsze zapotrzebowanie na pamięć operacyjną wykazał model sekwencyjny. Model Coarse-Grained charakteryzował się natomiast największym zużyciem pamięci. Pomiędzy modelami równoległymi niższym zużyciem CPU wyróżnił sie model Master-Slave. Efektywność samego algorytmu oceniana jest w artykule poprzez porównanie liczby iteracji potrzebnych do znalezienia najlepszego rozwiązania. We wszystkich równoległych modelach GA można było zaobserwować zależnosć pomiędzy liczbą iteracji, wielkością populacji oraz zużyciem pamięci. Wraz ze wzrostem ilości osobników liczba iteracji malała, natomiast wzrastało zużycie pamięci. Najszybszym modelem okazał się model Fine-Grained, który uzyskiwał najlepszy wynik prawie 27-krotnie szybciej niż model sekwencyjny. Testy przeprowadzone przez autorów potwierdziły korzyści płynące z wykorzystania równoległych implementacji algorytmu genetycznego. Wszystkie trzy modele równoległe osiągnęły istotne przyspieszenie i lepszą wydajność w porównaniu z modelem sekwencyjnym. W szczególności modele Fine-Grained i Coarse-Grained były bardziej wydajne, ponieważ liczba wymaganych iteracji była znacznie mniejsza niż w modelu sekwencyjnym.