Spis treści

Wstęp

1. Drzewa decyzyjne

1.1. Węzły i gałęzie

1.2. Kryteria podziału

1.3. Algorytmy budowania drzew decyzyjnych

1.3.1. Algorytm ID3

1.3.2. Algorytm C4.5

1.3.3. Algorytm CART

1.3.4. Algorytm CHAID

1.3.5. Algorytm MARS

2. Programowanie równoległe

Rozdział ma na celu uporządkowanie istotnych pojęć związanych z tematem programowania równoległego. Opisane zostały podstawowe zagadnienia, różnice między programowaniem współbieżnym i równoległym, rodzaje dekompozycji zadań, a także wzorce, stosowane w równoległych implementacjach algorytmów.

2.1. Podstawowe pojęcia

Przetwarzanie równoległe jest tematem omawianym w ramach dziedziny systemów operacyjnych. Celem pracy nie jest skupianie się wokół zagadnień specyficznych dla systemów operacyjnych. Rozumienie podstawowych pojęć jest jednak istotne podczas tworzenia równoległych implementacji algorytmów, dlatego zostały one poniżej przedstawione.

Procesor to jednostka sprzętowa, która pobiera dane z pamięci operacyjnej, interpretuje je i wykonuje. Pojęcie procesor używane jest na dwa sposoby. Pierwszy, zgodny z podaną definicją, używany jest głównie w elektronice. Drugie znaczenie procesora wykorzystywane jest częściej w programowaniu. Wówczas pojęcie procesor jest synonimem pojęcia rdzeń.

Rdzeniem fizycznym (ang. core) określany jest fizyczny element procesora, który pozwala na wykonywanie obliczeń. W uproszczeniu, im więcej rdzeni posiada procesor, tym szybciej wykonywane mogą być obliczenia. Liczba wykonywanych obliczeń w określonej jednostce czasu nazywana jest mocą obliczeniową. Jest to jeden z czynników branych pod uwagę przy ocenie wydajności procesora. Obecnie używa się głównie procesorów wielordzeniowych.

Rdzeń logiczny (ang. logical core, thread) przygotowuje dane wykorzystywane podczas obliczeń wykonywanych przez rdzeń fizyczny. Do niedawna na jeden rdzeń fizyczny przypadał jeden rdzeń logiczny. Obecnie najczęściej każdemu rdzeniowi fizycznemu przypisuje się dwa rdzenie logiczne. Rdzenie logiczne uzależnione są od rdzenia fizycznego, do którego są przypisane, natomiast nie są zależnie od operacji rdzeni logicznych przypisanych do odrębnego rdzenia fizycznego.

Rozkazem nazywane jest pojedyncze polecenie, które zapisane jest w postaci liczb binarnych i które wykonywane jest przez procesor.

Instrukcja definiowana jest jako bardziej złożone zadanie, które składa się ze zbioru rozkazów. Instrukcje mogą być niskopoziomowe (napisane w np. asemblerze) lub wysokopoziomowe (napisane w np. C, Java). Instrukcje wysokopoziomowe tłumaczone są na kilka instrukcji niskopoziomowych, natomiast instrukcje niskopoziomowe tłumaczone są na zbiór rozkazów. Zbiór rozkazów może być dzielony na podzbiory w celu uruchomienia każdego

podzbioru na innym procesorze.

Program jest zbiorem instrukcji, który pozwala na rozwiązanie pewnego problemu obliczeniowego.

Proces najprościej definiowany jest jako program, który jest w trakcie wykonywania. Pod pojęciem procesu zawiera się jednak wiele mechanizmów, przy pomocy których system operacyjny zarządza wykonywaniem programu. System operacyjny przydziela każdemu nowemu procesowi zasoby m.in. odrębny obszar pamięci operacyjnej, nadaje unikatowy numer PID (process identifier), kontroluje stan procesu oraz zarządza plikami, z których korzysta proces.

Wątek (ang. thread) jest częścią procesu. Jest to niezależny strumień instrukcji, który uruchamiany jest przez system operacyjny. Na jeden proces najczęściej składa się wiele strumieni instrukcji. Instrukcje składające się na wątek są wykonywane sekwencyjnie. Wszystkie wątki istniejące w ramach jednego procesu współdzielą przestrzeń adresową – mają dostęp do pamięci wspólnej. Z tego powodu komunikacja między wątkami należącymi do jednego procesu jest łatwa i niewymaga wsparcia systemu operacyjnego. Przekazanie danych polega na podaniu jedynie wskaźnika do miejsca w pamięci. Programiści często definiują wątek nieco inaczej. Wątek traktowany jest jako nieblokująca metoda, która wykonywana jest niezależnie od procesu, który ją uruchomił [1]. Określenie wątek jest również używane wymienne z określeniem rdzeń logiczny. Wątek w znaczeniu rdzenia logicznego nie jest jednak równoznaczny przedstawionej definicji wątku.

2.2. Rodzaje procesów

2.2.1. Proces sekwencyjny

Proces sekwencyjny charakteryzuje się tym, że każda kolejna instrukcja wykonywana jest dopiero wtedy, gdy zakończy się wykonywanie poprzedniej. Kolejność wykonywania instrukcji jest jednoznacznie określona, dlatego proces sekwencyjny określany jest jako pojedynczy ciąg instrukcji [2]. W przypadku procesu sekwencyjnego wątek nie jest częścią procesu, natomiast jest z nim utożsamiany [3].

2.2.2. Proces współbieżny

Procesy sekwencyjne, które zachodzą na siebie w czasie, określane są jako proces współbieżny. Innymi słowami proces współbieżny to proces, który składa się z wielu strumieni instrukcji. Instrukcje należące do jednego wątku, wykonywane są, zanim ukończone zostanie wykonywanie wszystkich instrukcji tworzących drugi, wcześniej uruchomiony wątek.

Dane są dwa procesy sekwencyjne P_1 i P_2, instrukcje $i_{1,1}, i_{1,2} \in P_1$ oraz $i_{2,1}, i_{2,2} \in P_2$. Rysunek [1] przedstawia dwie z wielu możliwych realizacji procesu P_1 oraz P_2.

2.2.3. Proces wykonywany metodą przeplotu

Proces wykonywany w przeplocie jest rodzajem procesu współbieżnego, w którym wątki uruchamiane są na przemiennie. Gdy uruchomiony jest wątek W_1 to wątek W_2 jest wstrzymywany. Gdy przerywane jest działanie wątku W_1 to na pewien czas uruchamiany jest wątek W_2. Metoda przeplotu pozwala na zastosowanie współbieżności w procesorach jednordzeniowych.

2.2.4. Proces równoległy

Proces równoległy jest szczególnym rodzajem procesu współbieżnego, w którym wątki uruchamiane są jednocześnie. Równoległe uruchamianie wątków jest możliwe, tylko gdy wykorzystany jest specjalny sprzęt. Wątki rozdzielane mogą być między rdzenie procesora – wtedy potrzeby jest komputer posiadający procesor kilkurdzeniowy. Inną możliwością jest zastosowanie architektury rozproszonej. Wówczas wątki dzielone są między zbiór komputerów, które połączone są ze sobą siecią. Proces równoległy nazywany jest wtedy rozproszonym.

2.3. Rodzaje dekompozycji problemów obliczeniowych

W celu równoległego wykonywania programu istotne jest zaprojektowanie podziału zadań obliczeniowych. Podział problemu na zadania nazywany jest dekompozycją. Wyróżniane są cztery rodzaje dekompozycji [2].

2.3.1. Dekompozycja danych

Dekompozycja danych to jeden z najczęściej wykorzystywanych rodzajów dekompozycji. Swoje zastosowanie znajduje szczególnie w przypadkach, gdzie przetwarzane są bardzo duże ilości danych. Dekompozycja danych dzieli się dekompozycję danych wejściowych i wyjściowych. Pierwsza z nich polega na podziale danych wejściowych na względnie równe części, które przetwarzane są w ramach osobnych zadań. Najczęściej zadania polegają na wykonaniu dokładnie takiego samego rodzaju obliczeń. Taki rodzaj dekompozycji charakteryzuje się tym, że po zakończeniu zadań, konieczne jest ich zsumowanie.

Dekompozycja danych wyjściowych jest możliwa, gdy elementy danych wyjściowych mogą zostać wyznaczone niezależnie od siebie. Wówczas każdemu zadaniu przydzielone zostają te

dane wejściowe, które konieczne są do otrzymania poszczególnych elementów danych wyjściowych. Wadą takiego podejścia jest stosunkowo niski stopień współbieżności [2].

Model, w którym zrównoleglanie osiągane jest poprzez zastosowanie dekompozycji danych, to model określany jest pojęciem **równoległość danych** (równoległość danych).

2.3.2. Dekompozycja funkcjonalna

Dekompozycja funkcjonalna polega na wyodrębnieniu obliczeń, których wykonanie konieczne jest do rozwiązania problemu. Obliczenia dzielone są na grupy, które formowane są w funkcje. Zadania funkcji różnią się od siebie i najczęściej przetwarzają różne rodzaje danych [2].

Zastosowanie dekompozycji funkcjonalnej oznacza Wykorzystanie (równoległości zadań) (task parallelism).

2.3.3. Dekompozycja rekursywna

Dekompozycja rekursywna stosowana jest przy rozwiązywaniu problemów metodą „dziel i zwyciężaj". Problem dzielony jest na mniejsze podproblemy, które są od siebie niezależne. Każdy podproblem jest mniejszym przypadkiem pierwotnego problemu. Podział wykonywany jest tak długo, aż podproblemy stają się trywialne do rozwiązania. Następnie wszystkie rozwiązania scalane są jedno, które jest ostatecznym rozwiązaniem [2].

2.3.4. Dekompozycja eksploracyjna

Dekompozycja eksploracyjna używana jest wtedy, gdy zadanie obliczeniowe polega na przeszukiwaniu przestrzeni rozwiązań. Przestrzeń dzielona jest na części, które eksplorowane są równolegle przez odrębne zadania. Jeśli rozwiązanie zostanie znalezione w którejś części przestrzeni, wówczas wykonywanie pozostałych zadań jest przerywane [2].

2.4. Wzorce programowania równoległego

Istnieje wiele wzorców programowania równoległego. Podczas implementacji algorytmu równoległego ciężko jest dobrać jeden, najlepiej pasujący wzorzec. Z tego powodu wzorce traktowane są raczej jako ogólne wskazówki, które mogą być przydatne podczas projektowania algorytmu. Najczęściej programy tworzone są zgodnie z kilkoma wzorcami, które łączone są ze sobą w celu stworzenia implementacji dopasowanej do konkretnego przypadku. W tym rozdziale opisane zostały wybrane wzorce programowania równoległego.

2.4.1. Wzorzec Master-Slave

Wzorzec inaczej nazywany jest wzorcem Manager-Worker. Wątek, który nazywany jest zarządcą (Master/Manager) definiuje zadania i rozdziela je pomiędzy wykonawców (Worker/Slave). Gdy wykonawca zrealizuje zadanie, przesyła otrzymane wyniki zarządcy, którego zadaniem jest scalić wszystkie zgromadzone wyniki w jedno rozwiązanie problemu. Wzorzec nie sprawdza się w problemach o dużym rozdrobnieniu zadań. Zarządca nie jest w stanie odpowiednio szybko generować i przydzielać zadań, ponieważ wykonawcy bardzo szybko kończą realizację zadań [2].

2.4.2. Wzorzec Fork-Join

Wzorzec Fork-Join to jeden z najczęściej stosowanych wzorców. Wykonywanie programu rozpoczyna się w ramach jeden wątku głównego. W momencie, gdy w kodzie programu pojawia się instrukcja, która wymaga równoległego przetwarzania, tworzone są dodatkowe wątki, które wykonywane są równolegle. Dopóki wszystkie wątki nie zakończą pracy i nie zostaną zniszczone, wątek główny nie może wznowić wykonywania sekwencyjnej części kodu. Etap „fork" polega na ustawieniu argumentów, które następnie otrzymuje każdy wątek. Etap „join" łączy wyniki po zakończeniu pracy wszystkich wątków równoległych. Etapy „fork" i „join" mogę wykonywane być dowolną ilość razy. [4].

2.4.3. Wzorzec Map-Reduce

Wzorzec Map-Reduce jest podobny do wzorca Fork-Join. Dane wejściowe są przetwarzane równolegle przez wiele wątków. Następnie wszystkie uzyskane wyniki są łączone, aż do momentu uzyskania jednego rozwiązania. W działaniu obydwa wzorce są prawie identycznie, jednak wywodzą się z różnych pomysłów. Idea mapowania pochodzi z technik stosowanych w funkcjonalnych językach programowania. Kilka mapowań może być połączonych w łańcuchy składające się na większe funkcje. Etapy „map" i „reduce" są od siebie bardziej niezależne niż etapy „fork" i „join". Mapowanie może występować bez redukcji, a redukcja bez mapowania [4].

2.4.4. Wzorzec Work Pool

Wzorzec puli zadań (Work Pool) wykorzystywany jest w algorytmach, w których zadania generowane są dynamicznie lub gdy istotnie różnią się złożonością. Zadania przechowywane są w strukturze danych, która dostępna jest w pamięci współdzielonej. Najczęściej wykorzy-

stywane struktury to lista, kolejka priorytetowe czy tablica z haszowaniem. W momencie, gdy wątek zakończył obliczanie zadania, dostarczane jest mu kolejne zadanie przechowywane w strukturze [2].

2.4.5. Wzorzec Pipeline

Wzorzec Pipeline polega na potokowym przetwarzaniu danych. Nazywany jest również metodą producenta i konsumenta. Strumień danych przekazywany jest do kolejnych wątków, które modyfikują oryginalny strumień. Każdy z wątków wykonuje inny rodzaj obliczeń, które tworzą etapy potoku. Dane wyjściowe jednego etapu stają się danymi wejściowymi kolejnego etapu. Stosowanie wzorca jest efektywne, jeśli czasy realizacji poszczególnych etapów są podobne [2].

3. Przegląd literatury

Rozdział trzeci zawiera przegląd publikacji naukowych. Tematem omawianych prac są różne metody programowania równoległego stosowane w algorytmach uczenia maszynowego.

3.1. Równoległe konstrukcje algorytmów drzew decyzyjnych

W artykule [5] zostało przedstawionych kilka strategii konstrukcji algorytmów drzew decyzyjnych, które oparte są o techniki takie jak: równoległość zadań, równoległość danych oraz równoległość hybrydowa. W pracy zaprezentowana została autorska implementacja równoległej konstrukcji drzewa decyzyjnego algorytmem C4.5. Na zakończenie autorzy przedstawili wyniki działania algorytmu i postawili wstępne wnioski dotyczące jego działania.

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 związanych 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 kontynuowania 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 wystąpienie 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 części 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 wspominane 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. Następnie komunikuje się z pozostałymi procesorami, w celu ustalenia jednego, najlepszego podziału. Po podziale (utworzeniu 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 rezultatów 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.

3.2. Równoległe uczenie zespołowe sieci neuronowych i lokalnych wzorców binarnych

W pracy [6] 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 ekstrakcji 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 następnie klasyfikowania zestawów danych treningowych. 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 zestawów 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 przeprowadzono 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 treningowego ORL wyniosła około 98%. Po przeprowadzeniu 2800 treningów dokładność zbioru testowego zwiększyła 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 dokładność sieci neuronowej jest znacznie zmniejszona, gdy do klasyfikacji nie wykorzystuje się schematu uczenia zespołowego.

3.3. Równoległe implementacje algorytmu genetycznego

Autorzy artykułu [7] zaprezentowali 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 pomiędzy 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 modyfikacjom 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 przypadku 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 przeprowadzone 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 następnie 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 się 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 zależność 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.

3.4. Zestawienie artykułów

W literaturze można znaleźć wiele artykułów na temat metod programowania równoległego wykorzystywanego do optymalizacji pracy algorytmów uczenia maszynowego. Metody jednak mogą znacznie się od siebie różnić w zależności od rodzaju algorytmu, dla którego są przeznaczone. W dalszej części pracy uwaga zostanie poświęcona już tylko podejściom stosowanym w algorytmach do budowy drzew decyzyjnych.

Tabela 1 zawiera odnośniki do tytułów artykułów, słowa kluczowe oraz krótki opis przybliżający tematykę poruszaną w każdym z artykułów.

Artykuł	**Słowa kluczowe**	**Opis**
[8]	równoległość wewnątrzwęzłowa; równoległość międzywęzłowa	Autorzy przedstawili oraz porównali wydajność czterech metod do równoległej implementacji algorytmu C4.5. W analizie uwzględniony został rodzaj danych wykorzystywanych do konstrukcji drzewa, który okazał się mieć duży wpływ na wyniki wydajności porównywanych metod. Wyszczególnione zostały trzy rodzaje techniki wewnątrzwęzłowe, które wykorzystują zrównoleglanie przetwarzania danych. Równolegle przetwarzane mogą być rekordy, atrybuty lub ich kombinacja (podejście hybrydowe). W technice międzywęzłowej równolegle przetwarzane są całe węzły – wszystkie operacje, które muszą zostać przeprowadzone do stworzenia węzła, wykonywane są przez jeden wątek.
[9]	węzły Hoeffdinga; architektura Master-Slave;	Autorzy artykułu zdecydowali się na zrównoleglanie tylko fragmentów algorytmu tj. szukania najlepszego atrybuty do podziału węzła. Zadania przydzielane przez główny procesor (master) innym procesorom (slave) polegają na obliczenia zysku informacyjnego dla określonego atrybutu. Zastosowana architektura oraz zrównoleglanie tylko wyszukiwania atrybutów skutkuje jednak w niskiej skalowalności. Pomimo tego testy wykazały, że przyspieszenie działania algorytmu jest dość efektywne. Dzięki zastoso-

		waniu nierówności Hoeffdinga, do podziału drzewa nie muszą być używane wszystkie dane. Ma to dodatkowy wpływ na szybkość działania algorytmu.
[10]	równoległość w węźle; duże zbiory danych	W artykule przedstawiono algorytm ParDTLT. Algorytm jest równoległą wersją algorytmu DTLT (Decision Trees from Large Training sets), który nie wymaga ładowania w całości wszystkich danych treningowych do pamięci komputera. ParDTLT oparty jest na idei sekwencyjnej budowy struktury drzewa oraz równoległym przetwarzaniu danych w każdym węźle. W danym czasie wszystkie procesory dostępne są dla jednego węzła. W węźle uprzywilejowanym tworzona jest kolejka atrybutów, dla których obliczany jest zrównoważony zysk informacji. Analiza kolejnych atrybutów przydzielane są procesorom do momentu, aż kolejka będzie pusta. Pozostałe węzły drzewa czekają, aż staną się węzłem uprzywilejowanym. Autorzy artykuły przeprowadzili testy algorytmu, które wykazały, że ParDTLT jest szybszy od algorytmów jak np. DTLT czy C4.5.
[11]	wielowątkowość; pamięć współdzielona; algorytm ID3	Modyfikacjom został poddany algorytm ID3. Przedstawiono dwie różne implementacje wykorzystujące równoległość. Pierwsza polega na stworzeniu tylu wątków, ile istnieje atrybutów, dla których obliczony musi zostać przyrost informacji. Gdy atrybut dzielący zostanie odnaleziony, tworzony jest węzeł. Następnie ponownie tworzone są kolejne wątki, które przetwarzają atrybuty nowo powstałych węzłów. Dopiero gdy wszystkie wątki zakończą pracę, z dostarczonych wyników składane jest drzewo. Różnica w drugiej implementacji polega na wykorzystaniu pamięci współdzielonej, dzięki czemu algorytm jest bardziej wydajnym pamięciowo. Węzeł główny jest współdzielony, dlatego każdy nowy węzeł może być tworzony, gdy tylko atrybut dzielący został odnaleziony.

Tabela 1: Zestawienie artykułów poruszających tematykę równoległości w algorytmach drzew decyzyjnych

Literatura

[1] Andrzej Karbowski, Ewa Niewiadomska-Szynkiewicz. Programowanie równoległe i rozproszone. Oficyna Wydawnicza Politechniki Warszawskiej. Warszawa, 2009.

[2] Czech Zbigniew J. Wprowadzenie do obliczeń równoległych. Wydawnictwo Naukowe PWN. Wyd. 2, 2013.

[3] Krzysztof Banaś, Skrypt Programowanie równoległe i rozproszone. Wydział Fizyki, Matematyki i Informatyki Politechniki Krakowskiej. Kraków, 2011.

[4] OpenCSF Project [Online]. Available: `https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/ParallelDesign.html`. Dostęp: 12.12.2022

[5] Amado, N., Gama, J., & Silva, F. (2001). Parallel Implementation of Decision Tree Learning Algorithms. Lecture Notes in Computer Science, 6–13. doi:10.1007/3-540-45329-6_4

[6] Tang, J., Su, Q., Su, B., Fong, S., Cao, W., & Gong, X. (2020). Parallel Ensemble Learning of Convolutional Neural Networks and Local Binary Patterns for Face Recognition. Computer Methods and Programs in Biomedicine, 105622. doi:10.1016/j.cmpb.2020.105622

[7] Skorpil, V., Oujezsky, V., & Tuleja, M. (2020). Testing of Python Models of Parallelized Genetic Algorithms. 2020 43rd International Conference on Telecommunications and Signal Processing (TSP). doi:10.1109/tsp49548.2020.9163475

[8] Kubota, K., Nakase, A., Sakai, H., & Oyanagi, S. (2000). Parallelization of decision tree algorithm and its performance evaluation. Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region. doi:10.1109/hpc.2000.843500

[9] Cal, P., & Woźniak, M. (2013). Parallel Hoeffding Decision Tree for Streaming Data. Advances in Intelligent Systems and Computing, 27–35. doi:10.1007/978-3-319-00551-5_4

[10] Rcega, A. F.-A., Suarez-Cansino, J., & Flores-Flores, L. G. (2013). A parallel algorithm to induce decision trees for large datasets. 2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT). doi:10.1109/icat.2013.6684045

[11] Maheshwari, S., Jatav, VK., & Meena, YK. (2011). Improved ID3 Decision Tree Generation using Shared-Memory and Multi-Threading Approach. 2011 International Conference on Education Technology and Computer (ICETC 2011). doi:10.13140/2.1.3216.2247