Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki, Elektroniki i Telekomunikacji AGH
Katedra Informatyki

Autoreferat rozprawy doktorskiej

Metody poprawiania efektywności algorytmu skalowania wielowymiarowego w kontekście wizualizacji dużych zbiorów danych

mgr inż. Piotr Pawliczek

Promotor:
Prof. dr hab. inż. Witold Dzwinel

Kraków, 2012

1 Wstęp

Szybki rozwój techniki komputerowej umożliwia coraz łatwiejsze i tańsze gromadzenie wielkich ilości danych. Z jednej strony pozwala to na pozyskiwanie większej ilości informacji o badanych zjawiskach, z drugiej prowadzi do coraz większych problemów z ich przetwarzaniem i analizą. Często z powodu zbyt dużej ilości danych trudno jest pozyskać z nich poszukiwane informacje.

Elementy przetwarzanych zbiorów danych często są reprezentowane przez wektory cech. Wprowadzenie odpowiedniej metryki do takiej przestrzeni pozwala na obliczenie odległości pomiędzy wektorami cech, a tym samym określenie podobieństwa pomiędzy nimi. Jednym z istotnych aspektów jest potrzeba wizualizacji tego typu danych, co pozwoliłoby wprawnemu operatorowi na szybkie wychwycenie niektórych ich własności. W ciągu kilkudziesięciu lat rozwoju narzędzi przetwarzania danych opracowano szereg różnego typu metod służących do wizualizacji zbiorów wielowymiarowych wektorów. Jedną z najważniejszych jest ekstrakcja nieliniowa reprezentowana przez popularny algorytm (grupę algorytmów) skalowania wielowymiarowego (and. Multidimensional Scaling), oznaczany również skrótem MDS.

1.1 Definicja skalowania wielowymiarowego

Pod pojęciem skalowania wielowymiarowego kryje się grupa algorytmów, których celem jest znalezienie lokalizacji zadanej ilości punktów w podanej przestrzeni. Pozycje punktów powinny być wyznaczone w taki sposób, aby odległości pomiędzy nimi jak najlepiej odpowiadały zadanym na wejściu wartościom.

Danymi wejściowymi dla algorytmu MDS jest kwadratowa macierz odległości $\mathbf{D} = [d_{ij}]$ o wymiarach $n \times n$. Celem skalowania wielowymiarowego jest znalezienie n punktów $\overline{y_1}, \overline{y_2}, \ldots, \overline{y_n}$ w h-wymiarowej przestrzeni docelowej, tak aby odległości pomiędzy nimi jak najlepiej pasowały do odpowiednich elementów macierzy $\mathbf{D}$, tj.:

$$d_{ij} \sim \|\overline{y_i} - \overline{y_j}\| \tag{1}$$

Poprzez $\mathbf{Y} = [y_{ij}]$ oznaczana będzie macierz o wymiarach $n \times h$, której kolejne wiersze odpowiadają szukanym punktom w przestrzeni docelowej:

$$\begin{aligned} \overline{y_1} &= (y_{11}, y_{12}, \ldots, y_{1h}) \\ \overline{y_2} &= (y_{21}, y_{22}, \ldots, y_{2h}) \\ &\ldots \\ \overline{y_n} &= (y_{n1}, y_{n2}, \ldots, y_{nh}) \end{aligned} \tag{2}$$

Przestrzeń docelowa jest zazwyczaj przestrzenią euklidesową.

Jak łatwo zauważyć, dla każdej konfiguracji $\mathbf{Y}$ istnieje nieskończenie wiele innych konfiguracji, będących równoważnym rozwiązaniem dla zadanej macierzy odległości $\mathbf{D}$. Wynika to z faktu, że odległości pomiędzy punktami wyznaczające jakość rozwiązania są takie same dla każdej konfiguracji $\mathbf{Y}'$, otrzymanej poprzez translację, rotację lub odbicie symetryczne konfiguracji $\mathbf{Y}$.

1.2 Funkcja kosztu

Jak łatwo wywnioskować z powyższej definicji, za wyjątkiem szczególnych przypadków, wynik skalowania wielowymiarowego będzie obarczony pewnym błędem. W większości metod skalowania wielowymiarowego jakość znalezionej konfiguracji $\mathbf{Y}$ mierzona jest specjalną

funkcją kosztu o ogólnej postaci:

$$E : (\mathbf{Y}) \to \mathbb{R}^+ \cup \{0\} \tag{3}$$

Im większa wartość funkcji E, tym większy błąd wyznaczonego wyniku. Zakładamy, że wartość równa zero oznacza idealne dopasowanie odległości.

Trzy z najbardziej znanych funkcji kosztu są zdefiniowane następująco:

$$STRESS_A(\mathbf{Y}) = \sqrt{\frac{2}{n(n-1)} \cdot \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij}(\|\overline{y_i} - \overline{y_j}\| - d_{ij})^2} \tag{4}$$

$$STRESS_B(\mathbf{Y}) = \sqrt{\frac{2}{n(n-1)} \cdot \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} \frac{(\|\overline{y_i} - \overline{y_j}\| - d_{ij})^2}{d_{ij}^2}} \tag{5}$$

$$STRAIN(\mathbf{Y}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (< \overline{x_i}, \overline{x_j} > - b_{ij})^2 \tag{6}$$

gdzie $< \cdot >$ oznacza iloczyn skalarny, a b_{ij} dane jest wzorem:

$$b_{ij} = -\frac{1}{2} \left(d_{ij}^2 - \frac{1}{n} \sum_{i=1}^{n} d_{ij}^2 - \frac{1}{n} \sum_{j=1}^{n} d_{ij}^2 + \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij}^2 \right) \tag{7}$$

Funkcja kosztu $STRESS_A$ odpowiada średniej kwadratowej z błędów bezwzględnych policzonych dla odwzorowanych odległości. Z kolei funkcja kosztu $STRESS_B$ odpowiada analogicznej średniej kwadratowej z błędów względnych. $STRAIN$ jest funkcją kosztu, której minimum globalne wyznacza klasyczny algorytm MDS, opisany pierwszy raz w pracy [9].

1.3 Metoda cząstek

Zastosowanie metody cząstek (ang. Molecular Dynamics - MD) do skalowania wielowymiarowego zostało opisane pracy [2]. Główna idea tego podejścia polega na "zakodowaniu" minimalizowanej funkcji kosztu za pomocą sił działających na n cząstek. Cząstki te poruszają się zgodnie z prawami dynamiki Newtona. Początkowe położenia cząstek oraz ich prędkości są losowane. Po uruchomieniu symulacji cząstki swobodnie przemieszczają się pod wpływem działających na nie sił. Energia jest stopniowo odprowadzana z układu poprzez siłę tarcia, działającą na każdą poruszającą się cząstkę i proporcjonalną do jej prędkości. Po pewnej liczbie iteracji cząstki wytracają całą swoją prędkość i zastygają w położeniu, które jest tożsame z wynikowym ułożeniem punktów w przestrzeni docelowej.

Każda para cząstek związana jest z pewnym potencjałem, który powoduje ich przyciąganie lub odpychanie, zależnie od odległości pomiędzy nimi. Wartość tego potencjału jest równa składnikowi funkcji kosztu dla danej pary punktów. Na przykład, dla funkcji kosztu:

$$E(\mathbf{Y}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (d_{ij} - \|\overline{y_i} - \overline{y_j}\|)^2 \tag{8}$$

wartość potencjału dla pary punktów o indeksach i oraz j jest równa:

$$E_{ij} = (d_{ij} - \|\overline{y_i} - \overline{y_j}\|)^2 \tag{9}$$

Wypadkowa siłą działająca na cząsteczkę jest sumą $(n-1)$ cząstkowych sił wynikających z oddziaływań pomiędzy nią i pozostałymi cząsteczkami. Po uwzględnieniu siły oporu proporcjonalnej do prędkości, ogólny wzór na wypadkową siłę działającą na cząsteczkę o indeksie i można zapisać następująco:

$$\overline{f}_i = m\frac{d\overline{v}_i}{dt} = -K\sum_{j=1}^{n}\nabla E_{ij} - \lambda\overline{v_i} \tag{10}$$

Wartości stałych K, m oraz λ są ustalone z góry, $\overline{v_i}$ oznacza prędkość cząsteczki.

Na całkowitą energię układu składa się suma potencjałów wszystkich par cząstek oraz suma ich energii kinetycznych. Podczas symulacji energia ta stopniowo maleje, co jest spowodowane działaniem siły tarcia. Na końcu symulacji, gdy cząstki osiągają prędkość bliską zeru, ich pozycja określa poszukiwane rozwiązanie.

1.4 Cel i zakres pracy

Dużym ograniczeniem większości metod skalowania wielowymiarowego jest ich złożoność obliczeniowa i pamięciowa (rzędu $\Omega(n^2)$), co uniemożliwia ich zastosowanie dla dużych zbiorów danych. Celem niniejszej pracy jest opracowanie metody skalowania wielowymiarowego pozwalającej na przetwarzanie dużych zbiorów danych ($n > 10^5$). Umożliwi to wizualizację zbiorów złożonych z kilkuset tysięcy wektorów o dużej ilości współrzędnych.

W rozprawie postawiono i wykazano następujące tezy:

1. ***Istnieje równoległy algorytm MDS, który pozwala na mapowanie dużych zbiorów danych ($n > 10^5$) poprzez rozproszenie obliczeń w środowisku klastra komputerów. Obliczenia w obrębie pojedynczego węzła mogą być efektywnie przyśpieszone poprzez ich rozdzielenie na wiele procesorów (w architekturze SMP) lub wykorzystanie współczesnych procesorów graficznych.***

2. ***Odpowiedni wybór elementów macierzy odległości D uwzględnianych podczas skalowania wielowymiarowego powoduje, że ze wzrostem ilości wektorów n w zbiorze danych, ilość elementów macierzy odległości potrzebnych do uzyskania zadanego błędu względnego rośnie znacznie wolniej niż n^2. Pozwala to na poprawienie złożoności obliczeniowej algorytmu MDS.***

2 Zrównoleglenie algorytmu MDS bazującego na metodzie cząstek

Sposobem przyśpieszenie algorytmu MDS może być zrównoleglenie obliczeń, czyli ich rozproszenie pomiędzy wątki lub procesy. Podział obliczeń pomiędzy wątki umożliwia efektywne wykorzystanie więcej niż jednego rdzenia procesora. Z kolei rozdzielenie ich pomiędzy procesy pozwala na wykorzystanie klastra komputerów. Alternatywnym rozwiązaniem jest wykorzystanie kart GPU (ang. Graphics Processing Unit), które pozwalają na równoległe wykonywanie tej samej operacji na wielu zmiennych jednocześnie.

2.1 Podział obliczeń pomiędzy wątki

Aby efektywnie rozproszyć obliczenia pomiędzy wątki trzeba odpowiednio zmodyfikować wykorzystany algorytm symulacji cząstek. Celem jest oczywiście zrównoleglenie pętli obliczającej siły działające na cząstki (wykonującej $\frac{(n-1)n}{2}$ iteracji). Dlatego wszystkie przetwarzane pary cząstek zostały podzielone pomiędzy użyte do obliczeń wątki - każdy wątek

	Ilość wątków			
	2	4	12	20
Intel Itanium 2 („baribal")	1,95	3,99	11,74	18,56
Dual Core AMD Opteron 270	2,00	3,97	—	—
Intel Core i5 M 430	2,00	2,19	—	—
Intel Xeon X5670	2,00	3,92	11,18	—
Intel Core2 Duo E8400	2,00	—	—	—

Tablica 1: Przyśpieszenie obliczeń uzyskane przez wielowątkowy algorytm MDS, zmierzone dla różnych procesorów.

oblicza częściowe siły będące wynikiem oddziaływań w obrębie przydzielonych mu par cząstek.

Bazujący na tym założeniu wielowątkowy algorytm MDS został zaimplementowany i przetestowany na różnych platformach: serwerze HP SL390 wyposażonym w dwa procesory Intel Xeon, superkomputerze "baribal" z ACK Cyfronet AGH, laptopie z procesorem Intel Core i5 oraz dwóch stacjach roboczych. Do podziału obliczeń pomiędzy wątki wykorzystano technologię OpenMP. Dla każdej z testowych platform wyznaczono przyśpieszenie uzyskane w wyniku zrównoleglenia obliczeń. Wszystkie pomiary zostały wykonane na danych wejściowych składających się z 30720 wektorów. Obliczenia podczas symulacji były wykonywane na zmiennych typu *float*.

Wyznaczone wartości współczynnika *speedup* zostały zebrane w tabeli 1. Osiągnięte wyniki pokazują, że testowany algorytm jest w stanie efektywnie rozdzielić obliczenia pomiędzy wątki. Jedynym wyjątkiem jest wynik otrzymany dla czterech wątków uruchomionych na procesorze Intel Core i5 M 430. Technologia Hyper-Threading nie była w stanie zastąpić dodatkowych rdzeni i osiągnięte dzięki niej przyśpieszenie jest stosunkowo niewielkie. We wszystkich pozostałych testowanych konfiguracjach osiągnięto *efficiency* większe od 90%.

2.2 Algorytm MDS dla GPU

Jednym ze sposobów przyśpieszenia obliczeń jest wykorzystanie, coraz popularniejszej w ostatnich latach, technologii GPGPU (ang *General-Purpose computing on Graphics Processing Unit*). Pojęcie to oznacza wykorzystanie do obliczeń układów zaprojektowanych pierwotnie do wydajnego renderowania grafiki 3D. W ramach pracy zaimplementowano algorytm MDS w środowisku CUDA, umożliwiającym obliczenia na kartach GPU wyprodukowanych przez firmę NVIDIA.

Do wydajnej implementacji algorytmu MDS dla GPU kluczowe jest zminimalizowanie operacji odczytu i zapisu do pamięci globalnej. Ponadto struktura danych musi być tak dobrana, aby te same operacje były wykonywane równolegle na ułożonych w szeregu zmiennych. Zaprojektowany w ramach tej pracy algorytm jest nieco podobny do algorytmu symulacji ruchu ciał niebieskich opisanego w [3] i bazuje na następujących założeniach:

- Cała macierz odległości jest na samym początku wgrywana do pamięci karty GPU i pozostaje tam niezmieniona aż do końca obliczeń.
- Do obliczeń wykorzystywana jest cała macierz odległości (n^2 elementów), a nie tylko jej połowa, jak to ma miejsce w algorytmie dla CPU.
- Bufory przechowujące położenie oraz prędkość cząstek podzielone są na 32-elementowe wektory.

Nazwa karty GPU	*Speedup* względem serwera 2 x Intel Xeon X5670			
	arytmetyka **fast**		arytmetyka **ieee**	
	względem 2 procesorów (12 wątków)	względem 1 rdzenia (1 wątek)	względem 2 procesorów (12 wątków)	względem 1 rdzenia (1 wątek)
GeForce 8800 Ultra	2,9	32,8	—	—
GeForce GT 330M	1,2	13,4	—	—
GeForce GTX 260	5,0	57,1	—	—
GeForce GTX 460	5,1	58,0	2,1	24,3
GeForce GTX 480	8,3	94,6	3,9	43,7
Tesla M2050	7,1	80,1	3,6	40,6

Tablica 2: *Speedup* uzyskany na kartach GPU w obliczeniach na zmiennych typu *float* w porównaniu do serwera wyposażonego w dwa procesory Intel Xeon X5670.

- Jeżeli cząsteczki leżą bardzo blisko siebie, to ich wzajemne oddziaływanie jest ignorowane.

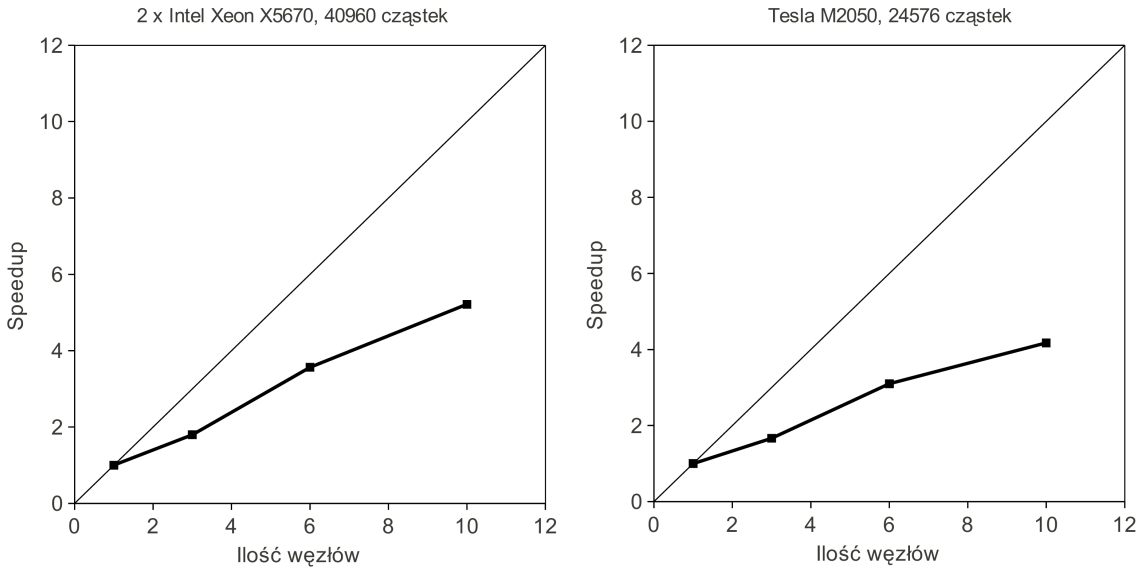
Opracowany program wykonujący obliczenia na karcie GPU ma postać pojedynczego kernela. Jest on uruchamiany dokładnie jeden raz w trakcie każdej iteracji.

Testy opracowanego algorytmu zostały przeprowadzone z wykorzystaniem uproszczonej arytmetyki zmiennoprzecinkowej (wersja **fast**). Dodatkowo dla kart GPU o zdolności obliczeniowej równej 2.0 lub 2.1 przeprowadzono testy z wykorzystaniem arytmetyki zgodnej ze standardem IEEE-754 (wersja **ieee**). W tabeli 2 zestawiono osiągnięte przez testowane karty GPU wartości *speedup*, osobno dla obliczeń z wykorzystaniem arytmetyki **fast** oraz **ieee**. Wartości parametru *speedup* zostały obliczone zarówno względem pojedynczego rdzenia procesora Intel Xeon X5670, jak i względem w pełni obciążonej pary takich procesorów, posiadającej łącznie 12 rdzeni. Otrzymane przyśpieszenie zostało obliczone na podstawie pomiarów dokonanych dla największego z testowych zbiorów danych, jaki udało się wgrać na daną kartę GPU.

Wszystkie karty GPU wymienione w tabeli 2 wykonały obliczenia szybciej niż dwuprocesorowy serwer z 12 rdzeniami. Czasy osiągnięte przez wielowątkowy algorytm, wykorzystujący wszystkie rdzenie dwóch procesorów Intel Xeon X5670, są podobne do czasów osiągniętych przez algorytm dla GPU uruchomiony na stosunkowo słabej karcie GeForce GT 330M, zamontowanej w średniej klasy laptopie. Mocniejsze karty graficzne do komputerów PC, takie jak GeForce GTX 260 lub GeForce GTX 460, pozwalają na osiągnięcie pięciokrotnego przyśpieszenia w porównaniu do wyżej wymienionego, dwuprocesorowego serwera. Przewaga kart GPU wyraźnie maleje, gdy w obliczeniach wymagana jest pełna arytmetyka zmiennoprzecinkowa (zgodna z normą IEEE-754). Wyniki przeprowadzonych dla tych kart testów pokazują, że zmiana wykorzystywanego zbioru instrukcji na **ieee** powoduje około dwukrotny wzrost czasu obliczeń.

2.3 Algorytm MDS dla klastra komputerowego

Jednym ze sposobów przyśpieszenia obliczeń jest rozdzielenie ich pomiędzy kilka komputerów. Dedykowana do tego celu sieć komputerów nazywana jest klastrem komputerowym, a należące do niej komputery węzłami. Obecnie najbardziej popularnym standardem, z którym zgodna jest większość istniejących bibliotek umożliwiających zrównoleglenie obliczeń na tego typu architekturze jest MPI (ang. Message Passing Interface). W wielu pracach



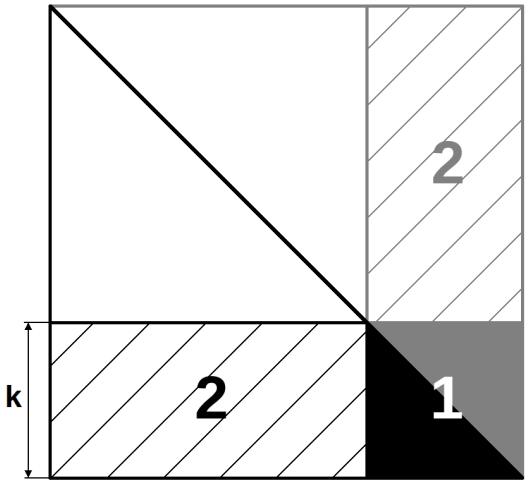
Rysunek 1: Przyśpieszenie symulacji na zmiennych typu *float* uzyskane przy pomocy klastra komputerów. Pomiary zostały wykonane niezależnie dla pary procesorów Intel Xeon X5670 oraz dla karty Tesla M2050.

naukowych, które pojawiły się w ciągu ostatnich dwóch dekad, można znaleźć przykłady zastosowania tego środowiska do przyśpieszenia różnego typu obliczeń (np.: [6]). W testach opisanych w ramach pracy użyto środowiska MPICH2, zgodnego ze standardem MPI2.

Przedstawiony w niniejszym rozdziale algorytm podziału obliczeń pomiędzy komputery klastra został opisany w pracy [5]. Bazuje on na algorytmie przedstawionym w pracy [8], a dokładniej jego modyfikacji opisanej w artykule [7]. W ramach niniejszej pracy zaimplementowano i przetestowano dwie wersje równoległego algorytmu MDS wykorzystującego środowisko MPI. W jednej wersji algorytmu obliczenia cząstkowych sił w obrębie węzła wykonywane są na dwóch procesorach Intel Xeon X5670, z wykorzystaniem wszystkich 12 dostępnych rdzeni. W drugiej wersji algorytmu do tych obliczeń została wykorzystana karta Tesla M2050 (jedna na każdym węźle).

Na wykresach przedstawionych na rysunku 1 przedstawiono wartości parametru *speedup* osiągnięte przez obie wersje równoległego algorytmu MDS. Wersja wykorzystująca procesory do obliczeń częściowych sił została przetestowana na zbiorze składającym się z 40960 cząstek. Natomiast wersja korzystająca z karty GPU została przetestowana na zbiorze zawierającym 24576 cząstek. Wszystkie obliczenia wykonywane były na zmiennych typu *float*. Jako pomiar referencyjny (dla jednego węzła) przyjęto czasy uzyskane odpowiednio przez wielowątkowy algorytm MDS (sekcja 2.1) oraz algorytm MDS dla GPU (sekcja 2.2).

Zmierzone przyśpieszenie dla 3, 6 i 10 węzłów daje *efficiency* w granicach 40-50%. Nieco gorsze wyniki otrzymane dla implementacji korzystającej z GPU są spowodowane mniejszą ilością symulowanych cząstek i wyraźnie krótszymi czasami otrzymanymi podczas pomiaru referencyjnego. Przez to w pomiarach otrzymanych dla 3, 6 i 10 węzłów czas zużyty na nawiązanie komunikacji pomiędzy nimi ma większy wpływ na osiągnięte przyśpieszenie.



Rysunek 2: Fragmenty macierzy odległości używane przy obliczeniach w algorytmie LANDMARKS.

3 Redukcja ilości odległości

Algorytmy MDS bazują na macierzy odległości pomiędzy mapowanymi punktami. Ilość jej elementów rośnie proporcjonalnie do kwadratu ilości mapowanych punktów, co implikuje co najmniej kwadratową złożoność czasową i pamięciową standardowych algorytmów MDS. Jednym ze sposobów na przyśpieszenie algorytmu MDS jest uwzględnienie podczas obliczeń tylko wybranych elementów macierzy odległości $\mathbf{D}$.

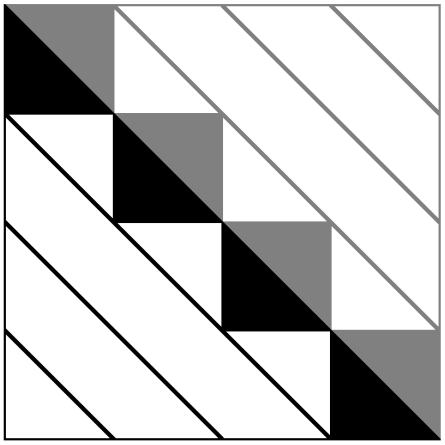
3.1 Algorytm LANDMARKS

Algorytm ten jest zainspirowany metodą Landmark MDS opisaną w pracy [1]. Polega ona na wybraniu z mapowanego zbioru k punktów referencyjnych i zrzutowaniu ich poprzez skalowanie wielowymiarowe do przestrzeni docelowej. Pozycje pozostałych punktów w przestrzeni docelowej są wyznaczane na podstawie ich odległości do ustalonych już k punktów referencyjnych.

W proponowanym tutaj podejściu do skalowania wielowymiarowego wykorzystywana jest metoda cząstek. Pozwala to na użycie funkcji kosztu $STRESS_A$ lub $STRESS_B$. Oryginalna metoda Landmark MDS bazuje na klasycznym algorytmie MDS, więc pozwala tylko na skalowanie wielowymiarowe z funkcją kosztu $STRAIN$.

Na rysunku 2 przedstawiono fragmenty macierzy odległości $\mathbf{D}$ wykorzystywane podczas obliczeń. Obszar oznaczony cyfrą 1 oznacza odległości użyte do obliczeń w pierwszym etapie algorytmu, a cyfrą 2 odległości wykorzystane w drugim etapie. Ilość odległości wykorzystanych podczas obliczeń jest równa $\frac{k\cdot(k+1)}{2}+k\cdot(n-k)$, co bezpośrednio przekłada się na mniejszą złożoność obliczeniową całego algorytmu. Do implementacji powyższej metody można wykorzystać opracowany w ramach tej pracy algorytm MDS dla GPU.

Przeprowadzone w ramach pracy testy pokazały, że zastosowanie algorytmu LANDMARKS powoduje znaczne zniekształcenie przetwarzanych zbiorów danych. Dużo lepsze wyniki dały opisane dalej algorytmy RANDOM i SUBSETS.



Rysunek 3: Fragmenty macierzy odległości używane przy obliczeniach w algorytmie SUBSETS.

3.2 Algorytm RANDOM

Najprostszym sposobem wyboru reprezentatywnego podzbioru elementów z macierzy odległości **D** jest ich wylosowanie. Algorytm ten został już opisany w pracy [4], gdzie zbadano również zależność pomiędzy jakością wyniku a ilością wykorzystanych do obliczeń elementów macierz **D**. Proces losowania odległości jest przeprowadzany raz na początku obliczeń. Skalowanie wielowymiarowe jest przeprowadzane na podstawie wyznaczonego losowo ciągu par indeksów punktów (i, j) i obliczonych dla nich odległości d_{ij}.

Ciąg elementów o takiej postaci może być danymi wejściowymi do odpowiednio zmodyfikowanej metody cząstek. Problemem jest tutaj przygotowanie wydajnej implementacji takiego algorytmu dla kart GPU. Uwzględnione w obliczeniach odległości są losowo porozrzucane po całej macierzy **D**, co uniemożliwia ich efektywne przetwarzanie przez GPU.

3.3 Algorytm SUBSETS

Pomysł tego algorytmu opiera się na podziale mapowanego zbioru punktów na p podzbiorów, wewnątrz których uwzględniane są wszystkie odległości. Podzbiory te łączone są pomiędzy sobą, tworząc strukturę reprezentującą cały mapowany zbiór. Połączenie dwóch podzbiorów odbywa się poprzez dodanie do zbioru przetwarzanych odległości odpowiednio wybranych krawędzi, które łączą punkty z jednego i drugiego podzbioru. Wartość parametru p (ilość podzbiorów) jest wyznaczana zgodnie z następującym wzorem:

$$p = \lfloor \sqrt{\frac{n}{3}} + 0,5 \rfloor \tag{11}$$

Złożoność obliczeniowa pojedynczej iteracji algorytmu SUBSETS wynosi $\Theta(hn\sqrt{n})$.

Na rysunku 3 przedstawiono schemat macierzy odległości z zaznaczonymi elementami, które są przetwarzane w każdej iteracji. Cztery wypełnione kwadraty leżące na przekątnej macierzy odpowiadają czterem podzbiorom, w obrębie których uwzględniane są wszystkie odległości. Ukośne linie odpowiadają elementom macierzy, które "zszywają" razem te podzbiory. Dzięki temu, że wykorzystywane do obliczeń elementy macierzy odległości ułożone są według dokładnie zdefiniowanego wzorca, możliwe było przygotowanie efektywnej implementacji algorytmu SUBSETS dla kart GPU.

4 Podsumowanie

4.1 Wnioski końcowe

W ramach niniejszej pracy dążono do udowodnienia tez przedstawionych w sekcji 1.4. Problem postawiony w pierwszej tezie został rozwiązany poprzez opracowanie równoległej wersji algorytmu skalowania wielowymiarowego, pozwalającej na wykorzystanie do obliczeń klastra komputerów wyposażonych w karty GPU Tesla M2050. Przy wykorzystaniu pełnej macierzy odległości D, przygotowana w ramach pracy implementacja pozwala na przetworzenie zbioru danych o wymaganej w tezie wielkości w ciągu kilkudziesięciu minut.

Druga teza została udowodniona poprzez zastosowanie odpowiedniego algorytmu wyboru odległości używanych podczas skalowania wielowymiarowego. Rezultaty przeprowadzonych testów pokazują, że opisane w pracy algorytmy o nazwach RANDOM i SUBSETS kosztem niewielkiego, stałego błędu względnego pozwalają na ograniczenie ilości używanych elementów macierzy odległości do $n^{\frac{3}{2}}$. Przekłada się to bezpośrednio na redukcję złożoności obliczeniowej całego algorytmu.

Opracowanie w pełni zoptymalizowanych dwóch wersji tego samego algorytmu równoległego, jednego dla kart GPU, a drugiego dla wielordzeniowych i wieloprocesorowych komputerów pozwoliło na porównanie wydajności tych dwóch architektur. Zamieszczone w pracy wyniki pokazują, że jedna karta GPU może wykonywać obliczenia kilka razy szybciej niż dedykowany do obliczeń serwer posiadający kilkanaście rdzeni. Jednak na niekorzyść kart GPU przemawia stosunkowo niewielka ilość pamięci. Mała przepustowość pomiędzy pamięcią globalną karty GPU oraz pamięcią operacyjną komputera nie pozwala na efektywne wykorzystanie tej drugiej podczas obliczeń.

4.2 Oryginalne elementy pracy

W ramach niniejszej pracy zostało zaprojektowanych kilka własnych algorytmów:

- Równoległa wersja algorytmu skalowania wielowymiarowego, pozwalająca na efektywny podział obliczeń pomiędzy wiele rdzeni procesorów.
- Algorytm skalowania wielowymiarowego pozwalający na wykorzystanie do obliczeń kart GPU firmy NVIDIA.
- Metoda skalowania wielowymiarowego dużych zbiorów danych w środowisku klastra komputerów.
- Wersja algorytmu Landmark MDS bazująca na funkcji kosztu $STRESS_A$ i $STRESS_B$.
- Algorytm wyboru odległości do obliczeń o nazwie SUBSETS, pozwalający na redukcję złożoności obliczeniowej skalowania wielowymiarowego do $n^{\frac{3}{2}}$ przy zachowaniu dobrej jakości otrzymywanych wyników.

Wszystkie te algorytmy zostały zaimplementowane i przetestowane.

4.3 Sugerowane kierunki dalszych badań

W kontekście otrzymanych wyników celowym wydaje się dalsze prowadzenie badań nad sposobem wyboru elementów z macierzy odległości. Pomimo tego, że istnieją algorytmy pozwalające na skalowanie wielowymiarowe dużych zbiorów danych, żaden z nich nie jest w stanie zapewnić wyników porównywalnych z rezultatami otrzymanymi dla pełnej macierzy

odległości. Osiągnięta w ramach niniejszej pracy złożoność obliczeniowa rzędu $n^{\frac{3}{2}}$ nadal nie pozwala w czasie rzeczywistym na dokładną wizualizację zbiorów danych o zawierających więcej niż 10^5 wektorów.

Literatura

[1] De Silva V., Tenenbaum J.B. Global versus local methods in nonlinear dimensionality reduction. *Advances in Neural Information Processing Systems*, 15, 705–712, 2003.

[2] Dzwinel W., Błasiak J. Method of particles in visual clustering of multi-dimensional and large data sets. *Future Generation Computer Systems*, 15, 365–379, 1999.

[3] Nyland L., Harris M., Prins J. Fast N-Body Simulation with CUDA. H. Nguyen (redaktor), *GPU Gems 3*, 677–695. Addison-Wesley Professional, 2007.

[4] Pawliczek P., Dzwinel W. Visual analysis of multidimensional data using fast MDS algorithm. R.S. Romaniuk (redaktor), *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2007*, *Proceedings of the SPIE*, tom 6937, 69372N–69372N–11. SPIE, 2008.

[5] Pawliczek P., Dzwinel W. Parallel Implementation of Multidimensional Scaling Algorithm Based on Particle Dynamics. R. Wyrzykowski, J. Dongarra, K. Karczewski, J. Wasniewski (redaktorzy), *Parallel Processing and Applied Mathematics*, *Lecture Notes in Computer Science*, tom 6067, 312–321. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[6] Pawliczek P., Romanowska-Pawliczek A., Soltys Z. Parallel deconvolution of large 3D images obtained by confocal laser scanning microscopy. *Microscopy Research and Technique*, 73, 187–194, 2010.

[7] Shu J.W., Wang B., Chen M., Zhao Wang J., Min Zheng W. Optimization techniques for parallel force-decomposition algorithm in molecular dynamic simulations. *Computer Physics Communications*, 154, 121–130, 2003.

[8] Taylor V.E., Stevens R.L., Arnold K.E. Parallel molecular dynamics: communication requirements for massively parallel machines. *Frontiers of Massively Parallel Computation, 1995. Proceedings.*, 156–163. IEEE, McLean, VA , USA, 1995.

[9] Torgerson W.S. Multidimensional scaling: I. Theory and method. *Psychometrika*, 17, 401–419, 1952.