Αριστοτελειο Πανεπιστημιο Θεσσαλονικης

Διπλωματικη Εργασια

Βελτιστοποίηση Απόδοσης του αλγορίθμου SG-tSNE-Π με την χρήση Μονάδων Επεξεργασίας Γραφικών

Συγγραφέας:
Ιακωβίδης Ιωάννης

Επιβλέπων:
κ. Νικόλαος Πιτσιάνης,
κ. Δημήτριος Φλώρος

Κατεύθυνση: Ηλεκτρονικής και Υπολογιστών

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Ιούνιος 2,2021

ARISTOTLE UNIVERSITY OF THESSALONIKI

DIPLOMA THESIS

Performance Optimization of the SG-tSNE-Π algorithm using the Graphics Processing Unit

Author:
Iakovidis Ioannis

Supervisor:
Mr. Nikolaos Pitsianis,
Mr. Dimitrios Floros

Specialization: Electronic and Computer Engineering

School of Electrical and Computer Engineering
Department of Electronics and Computer Engineering

June 2,2021

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κ. Νικόλαο Πιτσιάνη και τον υποψήφιο διδάκτορα Δημήτριο Φλώρο για την βοήθεια και την καθοδήγηση που μου παρείχαν μέσα από τις συμβουλές τους. Θα ήθελα επίσης να τους ευχαριστήσω για το χρόνο που αφιέρωσαν στις συζητήσεις μας για την υλοποίηση της εργασίας και για την καλύτερη εμβάθυνση στο θέμα. Θα ήθελα, ακόμη, να τους εκφράσω την ευγνωμοσύνη μου για όλες αυτές τις συζητήσεις που οδήγησαν στην διεύρυνση του τρόπου σκέψης μου και στην περαιτέρω εξέλιξη μου στον τομέα.

Τέλος, ευχαριστώ πάρα πολύ την οικογένεια μου, τους γονείς μου και την αδερφή μου που ήταν και είναι πάντα δίπλα μου στηρίζοντας τις αποφάσεις μου και κάνοντας το καλύτερο για μένα με όλα τα μέσα.

Περίληψη

Ο t-distributed Stochastic Neighborhood Embedding (t-SNE) είναι ένας ευρέως διαδεδομένος αλγόριθμος μείωσης της διαστασιμότητας, ο οποίος χρησιμοποιείται κυρίως για την οπτικοποίηση πολυδιάστατων δεδομένων. Μέσα από αυτήν την διπλωματική θα παρουσιάσουμε μία επιτάχυνση του t-SNE, υλοποιημένη σε CUDA, με την χρήση Μονάδων Επεξεργασίας Γραφικών. Βασίζουμε την υλοποίησή μας στην εργασία "Spaceland Embedding of Sparse Stochastic Graphs, HPEC 2019". Για την απόκτηση μίας απεικόνισης απαιτούνται ουσιαστικά δύο βήματα, ένας πυκνός και ένας αραιός υπολογισμός. Μία από τις κύριες προκλήσεις της χρήσης μονάδων επεξεργασίας γραφικών είναι ότι οι αραιοί υπολογισμοί συνήθως δεν επεκτείνονται ικανοποιητικά καθώς απαιτούν απρόβλεπτες προσπελάσεις μνήμης. Για να ξεπεράσουμε αυτό το πρόβλημα χρησιμοποιούμε μία κατάλληλη δομή δεδομένων για την αποθήκευση του αραιού πίνακα, που οδηγεί σε έναν τοπικά πυκνό υπολογισμό, και αξιοποιεί καλύτερα την μονάδα επεξεργασίας γραφικών. Ο πυκνός υπολογισμός υλοποιείται με τη χρήση μιας μεθόδου παρεμβολής η οποία επιταχύνεται με την βοήθεια του Γρήγορου Μετασχηματισμού Φουριέ. Τελειώνοντας, για συστήματα υψηλής απόδοσης τα οποία διαθέτουν μεγάλο αριθμό πυρήνων παρουσιάζουμε μία Υβριδική υλοποίηση που χρησιμοποιεί ταυτόχρονα την Κεντρική Μονάδα Επεξεργασίας για την εκτέλεση του αραιού υπολογισμού και την Μονάδα Επεξεργασίας Γραφικών για την εκτέλεση του πυκνού υπολογισμού. Με αυτή την διαδικασία μπορούμε να κρύψουμε κάποιο από το χρονικό κόστος του υπολογισμού και έτσι να έχουμε επιπλέον επιτάχυνση.

Λέξεις- κλειδιά: Υπολογισμοί Υψηλής Απόδοσης, t-SNE, Μείωση Διαστασιμότητας, Οπτικοποίηση Δεδομένων, CUDA, Μονάδα Επεξεργασίας Γραφικών, Οπτικοποίηση Γράφων

Abstract

t-distributed Stochastic Neighborhood Embedding (t-SNE) is a widely used dimensionality reduction technique, that is particularly well suited for visualization of high-dimensional datasets. On this diploma thesis we introduce a high performance GPU-accelerated implementation of the t-SNE method in CUDA. We base our approach on the work "Spaceland Embedding of Sparse Stochastic Graphs, HPEC 2019". Obtaining an embedding essentially requires two steps, namely a dense and a sparse computation. One of the main bottlenecks is that usually sparse computation does not scale well in GPU's because of the irregularity of the memory accesses. To overcome this problem, we use a more suitable sparse matrix storage format that leads to locally dense data and is better suited for GPU processing. The dense computation is performed with an interpolation-based fast Fourier Transform accelerated method. Finally, for high performance multicore systems we introduce a Hybrid CPU-GPU implementation that executes the sparse computation on the CPU and the dense computation on the GPU in parallel, hiding some of the total temporal cost in the process.

Keywords: Performance Computing, t-SNE, Dimensionality Reduction, Data Visualization, CUDA, Graphical Processing Unit, Embedding, Graph Visualization

Περιεχόμενα

Λίστα εικόνων

Κεφάλαιο 1

Εισαγωγή

1.1 Περιγραφή του αλγορίθμου

Τις τελευταίες δεκαετίες, με την επέκταση της χρήσης του διαδικτύου, η απόκτηση δεδομένων γίνεται ολοένα και πιο λεπτομερής και με μεγαλύτερη ταχύτητα. Έτσι τα σημερινά σύνολα δεδομένων αποτελούνται από πάρα πολλά πεδία που συνθέτουν την περιγραφή των αντικειμένων. Αυτό κάνει την επεξεργασία τους όλο και πιο δύσκολη. Ως αποτέλεσμα η σχεδίαση μίας αποτελεσματικής μεθόδου οπτικής απεικόνισης των δεδομένων γίνεται είναι ολοένα και πιο σημαντική. Καθώς μία κατάλληλη απεικόνιση μπορεί να βοηθήσει στην μείωση του συνολικού χρόνου επεξεργασίας των δεδομένων και στην αναγνώριση μοτίβων σε αυτά. Ο t-Distributed Stochastic Neighbor Embedding (t-SNE) είναι ένας αλγόριθμος μείωσης της διαστασιμότητας που είναι ιδιαίτερα χρήσιμος για οπτική απεικόνιση πολυδιάστατων συνόλων δεδομένων [1].

Έστω $x_i \in \mathbb{R}^d, i = 1, \ldots, n$ στοιχεία δεδομένων στον πολυδιάστατο χώρο και $y_i \in \mathbb{R}^s, i = 1, \ldots, n$ στοιχεία δεδομένων σε έναν ολιγοδιάστατο χώρο, όπου το s είναι συνήθως 2 ή 3 για οπτική απεικόνιση. Ο στόχος του t-SNE είναι να διατηρήσει την τοπική δομή των δεδομένων, δηλαδή σημεία που είναι όμοια μεταξύ τους (έχουν μικρή απόσταση) στον πολυδιάστατο χώρο να απεικονίζονται σε όμοια σημεία (μικρή απόσταση) στον ολιγοδιάστατο χώρο. Αυτός ο στόχος επιτυγχάνεται με την ελαχιστοποίηση ενός μέτρου απόκλισης βαθμών ομοιότητας ανάμεσα στα στοιχεία, που έχουν οριστεί στον πολυδιάστατο και ολιγοδιάστατο χώρο.

Πιο συγκεκριμένα από τις αποστάσεις d_{ij} των $(x_i)_{i=1}^n$ ορίζουμε

$$p_{i|j} = \frac{e^{-d_{ij}^2/2\sigma_i^2}}{\sum_{l \neq i} e^{-d_{il}^2/2\sigma_i^2}} \quad \text{and} \quad p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

ως τους βαθμούς ομοιότητας ανάμεσα στα στοιχεία του πολυδιάστατου χώρου. Η τιμή $p_{j|i}$ μπορεί να ερμηνευθεί ως η κατανομή των υπόλοιπων στοιχείων ως προς το σημείο x_i ή η πιθανότητα το j-οστο σημείο να είναι γείτονας του i-οστου σημείου. Οι παράμετροι $(\sigma_i)_{i=1}^n$ επιλέγονται για ρύθμιση της μεθόδου από τον χρήστη (περισσότερες πληροφορίες στην παράγραφο 1.4.1). Για τον ολιγοδιάστατο χώρο ορίζουμε τα

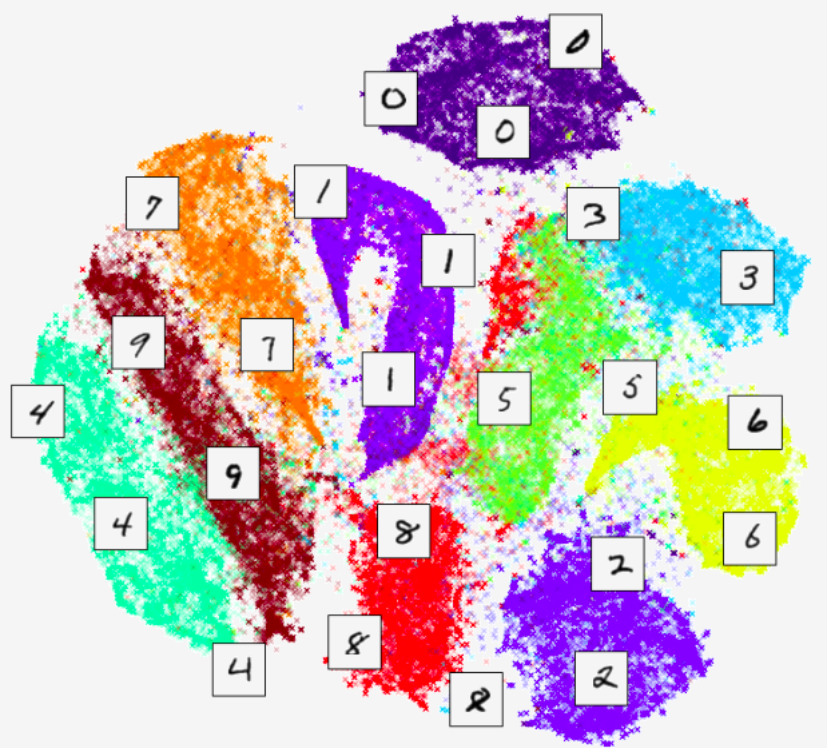
$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$

ως τους τους βαθμούς ομοιότητας των στοιχείων $(y_i)_{i=1}^n$ αλλά σε αυτήν την περίπτωση χρησιμοποιούμε την t-κατανομή. Χρησιμοποιούμε την t-κατανομή αντί της Γκαουσιανής για τον χώρο ενσωμάτωσης ώστε να γίνει αποφυγή του προβλήματος overcrowding [1]. Για διευκόλυνση του συμβολισμού έστω $P = \{p_{ij}\}$και $Q = \{q_{ij}\}$ $n \times n$ πίνακες και $X = [x_1, \ldots, x_n]$, $Y = [y_1, \ldots, y_n]$ $d \times n$ και $s \times n$ πίνακες αντίστοιχα.

Η απεικόνιση t-SNE παράγεται με την ελαχιστοποίηση της, $C(Y)$, Kullback–Leibler (KL) απόκλισης

$$C(Y) = KL(P \mid Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

Η ελαχιστοποίηση αυτή μπορεί να γίνει με την χρήση της μεθόδου ελάττωσης της παραγώγου (gradient descent). Αξίζει να σημειωθεί ότι η συγκεκριμένη συνάρτηση απόκλισης είναι μη κυρτή και έτσι ο αλγόριθμός μας μπορεί να τερματίσει σε ένα τοπικό ελάχιστο. Όμως η μέθοδος αυτή δουλεύει εξαιρετικά στην πράξη και διαθέτει ελκυστικές θεωρητικές ιδιότητες [2] [3], παρά το γεγονός ότι βασίζεται στην ελαχιστοποίηση ενός μη κυρτού προβλήματος. Στην παρακάτω φιγούρα βλέπουμε την απεικόνιση του συνόλου Mnist με τη χρήση αυτής της μεθόδου στον δισδιάστατο χώρο.



Εικόνα 1.1: Εφαρμογή του t-SNE στο σύνολο δεδομένων Mnist

1.2 Υπολογισμός της Παραγώγου

Σε αυτήν την παράγραφο θα υπολογίσουμε την έκφραση της παραγώγου που θα χρησιμοποιηθεί στην gradient descent. Αρχικά έχουμε:

$$\begin{aligned} C(Y) &= \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \\ &= \sum_{i \neq j} p_{ij} \log p_{ij} - \sum_{i \neq j} p_{ij} \log q_{ij} \\ &= \sum_{i \neq j} p_{ij} \log p_{ij} + \sum_{i \neq j} p_{ij} \log f_{ij} + \sum_{i \neq j} p_{ij} \log Z \\ &= \sum_{i \neq j} p_{ij} \log p_{ij} + \sum_{i \neq j} p_{ij} \log f_{ij} + \log Z. \end{aligned}$$

Όπου $f_{ij} = 1 + \|y_i - y_j\|^2$ και $Z = \sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}$.
Και έτσι έπειτα με λίγο υπολογισμό:

$$\begin{aligned} \frac{\partial \sum_{i \neq j} p_{ij} \log f_{ij}}{\partial y_m} &= \sum_{i \neq m} p_{im} \frac{\partial \log f_{im}}{\partial y_m} + \sum_{i \neq m} p_{mi} \frac{\partial \log f_{mi}}{\partial y_m} \\ &= 2 \sum_{i \neq m} p_{mi} \frac{\partial \log f_{mi}}{\partial y_m} \\ &= 2 \sum_{i \neq m} p_{mi} \frac{1}{f_{mi}} \frac{\partial f_{mi}}{\partial y_m} \\ &= 2 \sum_{i \neq m} p_{mi} \frac{2(y_m - y_i)}{1 + \|y_m - y_i\|^2} \\ &= 4 \sum_{i \neq m} p_{mi} \frac{Z(y_m - y_i)}{Z(1 + \|y_m - y_i\|^2)} \\ &= 4 \sum_{i \neq m} p_{mi} q_{mi} Z(y_m - y_i). \end{aligned}$$

$$\begin{aligned} \frac{\partial \log Z}{\partial y_m} &= \frac{\partial Z}{\partial y_m} \frac{1}{Z} \\ &= \frac{2}{Z} \sum_{i \neq m} \frac{\partial (1 + \|y_m - y_i\|^2)^{-1}}{\partial y_m} \\ &= -\frac{2}{Z} \sum_{i \neq m} \frac{(y_m - y_i)}{(1 + \|y_m - y_i\|^2)^2} \\ &= -\frac{2}{Z} \sum_{i \neq m} \frac{2(y_m - y_i)}{1 + \|y_m - y_i\|^2} \\ &= -4 \sum_{i \neq m} q_{mi}^2 Z(y_m - y_i). \end{aligned}$$

Άρα

$$\frac{\partial C(Y)}{\partial y_i} = 4 \sum_{j \neq i} p_{ij} q_{ij} Z(y_i - y_j) - 4 \sum_{j \neq i} q_{ij}^2 Z(y_i - y_j) \tag{1}$$

Μπορούμε επίσης να εκτελέσουμε τον ίδιο υπολογισμό και με έναν επιπλέον τρόπο στον οποίο μπορεί γίνει πιο εύκολη πιθανή αλλαγή της αντικειμενικής συνάρτησης ή των βαθμών ομοιότητας. Αρχικά παρατηρούμε τον τρόπο με τον οποίο η αντικειμενική συνάρτηση C εξαρτάται από τα ενσωματωμένα σημεία (embedded points). Έστω πιο γενικά ότι $q_{ij} = \frac{w_{ij}}{\sum_{kl} w_{kl}}$, όπου w_{ij} ο βαθμός ομοιότητας του y_i και y_j που εξαρτάται μόνο από την απόσταση τους, d_{ij}. Τότε όταν υπολογίζουμε την κλίση υπολογίζουμε:

1. Τις αποστάσεις d_{ij} που παράγονται από τις συντεταγμένες των,y_i, y_j.
2. Τους βαθμούς ομοιότητας w_{ij} που παράγονται σαν συνάρτηση των αποστάσεων, d_{ij}.
3. Τις πιθανότητες τις εξόδου, q_{ij}, που είναι κανονικοποιημένες εκφράσεις των βαθμών ομοιότητας, w_{ij}.
4. Και τέλος την συνάρτηση κόστους, C η οποία είναι συνήθως μία μορφή απόκλισης όπως η KL απόκλιση, δηλαδή μία έκφραση των πιθανοτήτων εξόδου, q_{ij}.

Από την παραπάνω πληροφορία θα εφαρμόσουμε τον κανόνα της αλυσίδας για μερικές παραγώγους. Η αλυσίδα των εξαρτήσεων των μεταβλητών είναι: $C \to q \to w \to d \to y$. Οπότε

$$\frac{\partial C}{\partial \mathbf{y_m}} = \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{kl} \frac{\partial q_{ij}}{\partial w_{kl}} \sum_{pq} \frac{\partial w_{kl}}{\partial d_{pq}} \frac{\partial d_{pq}}{\partial \mathbf{y_m}}$$

Η σχέση ανάμεσα στα w, και d είναι τέτοια ώστε οι εκατέρωθεν όροι να είναι 0, δηλαδή αν δεν ισχύει $k = p$ και $l = q$ τότε η παράγωγος είναι ίση με 0. Επίσης προφανώς ισχύει, είτε $k = m$ ή $l = m$, αλλιώς $\partial d_{kl} / \partial \mathbf{y_m} = 0$. Άρα

$$\begin{aligned}
\frac{\partial C}{\partial \mathbf{y_m}} &= \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{kl} \frac{\partial q_{ij}}{\partial w_{kl}} \sum_{pq} \frac{\partial w_{kl}}{\partial d_{pq}} \frac{\partial d_{pq}}{\partial \mathbf{y_m}} \\
&= \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{kl} \frac{\partial q_{ij}}{\partial w_{kl}} \frac{\partial w_{kl}}{\partial d_{kl}} \frac{\partial d_{kl}}{\partial \mathbf{y_m}} \\
&= \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{l} \frac{\partial q_{ij}}{\partial w_{ml}} \frac{\partial w_{ml}}{\partial d_{ml}} \frac{\partial d_{ml}}{\partial \mathbf{y_m}} + \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{k} \frac{\partial q_{ij}}{\partial w_{km}} \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}}
\end{aligned}$$

Έπειτα εναλλάσσονταν την σειρά των αθροισμάτων έχουμε

$$\begin{aligned}
\frac{\partial C}{\partial \mathbf{y_m}} &= \sum_{l} \Big(\sum_{ij} \frac{\partial C}{\partial q_{ij}} \frac{\partial q_{ij}}{\partial w_{ml}} \Big) \frac{\partial w_{ml}}{\partial d_{ml}} \frac{\partial d_{ml}}{\partial \mathbf{y_m}} + \sum_{k} \Big(\sum_{ij} \frac{\partial C}{\partial q_{ij}} \frac{\partial q_{ij}}{\partial w_{km}} \Big) \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}} \\
&= \sum_{l} f_{ml} \frac{\partial w_{ml}}{\partial d_{ml}} \frac{\partial d_{ml}}{\partial \mathbf{y_m}} + \sum_{k} f_{km} \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}} \\
&= \sum_{k} f_{mk} \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}} + \sum_{k} f_{km} \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}}, \text{Υποθέτουμε ότι τα w και d είναι συμμετρικά.} \\
&= \sum_{k} \Big(f_{mk} + f_{km} \Big) \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}}
\end{aligned}$$

Ισχύει

$$\frac{\partial d_{ij}}{\partial \mathbf{y_i}} = \frac{1}{d_{ij}} \left(\mathbf{y_i} - \mathbf{y_j}\right) \text{ για } d_{ij} = \left[\sum_{l}^{K} \left(y_{il} - y_{jl}\right)^2\right]^{1/2}$$

Επίσης

$$\frac{\partial w_{ij}}{\partial d_{ij}} = -\frac{2d_{ij}}{(1+d_{ij}^2)} \text{ for } w_{ij} = \frac{1}{(1+d_{ij}^2)^2}$$

Οπότε

$$\frac{\partial C}{\partial \mathbf{y_m}} = -2\sum_{k} \left(f_{mk} + f_{km}\right) \frac{(y_m - y_k)}{(1+{d_{km}}^2)^2}$$

Τώρα συγκεκριμένα για την KL απόκλιση έχουμε:

$$\frac{\partial C}{\partial q_{ij}} = -\frac{p_{ij}}{q_{ij}}$$

$$\frac{\partial q_{ij}}{\partial w_{km}} = -\frac{w_{ij}}{(\sum_{lt} w_{lt})^2} = -\frac{q_{ij}}{Z}$$

Επίσης

$$\frac{\partial q_{ij}}{\partial w_{ij}} = -\frac{w_{ij}}{(\sum_{lt} w_{lt})^2} + \frac{1}{\sum_{lt} w_{lt}} = -\frac{q_{ij}}{Z} + \frac{1}{Z}$$

Άρα

$$f_{km} = -\frac{p_{km}}{Z q_{km}} + \sum_{ij} \frac{p_{ij}}{Z}$$

Τέλος παίρνουμε ως αποτέλεσμα την ζητούμενη έκφραση

$$\frac{\partial C(Y)}{\partial y_i} = 4\sum_{j \neq i} p_{ij} q_{ij} Z(y_i - y_j) - 4\sum_{j \neq i} q_{ij}^2 Z(y_i - y_j)$$

1.3 Ερμηνεία της Παραγώγου

Ο υπολογισμός της κλίσης του t-SNE μπορεί να αναδιατυπωθεί ως μία προσομοίωση ενός προβλήματος N-σωμάτων όπου οι δυνάμεις που δρουν είναι:

$$F_{attr,i} = \sum_{j \neq i} p_{ij} q_{ij} Z(y_i - y_j)$$

$$F_{rep,i} = \sum_{j \neq i} q_{ij}^2 Z(y_i - y_j)$$

$$\frac{1}{4}\frac{\partial C(Y)}{\partial y_i} = F_{attr,i} - F_{rep,i}$$

Οι δυνάμεις μπορούνε επίσης να γραφτούν με πράξεις πίνακα-διάνυσμα με τους ακόλουθους τρόπους:

- $$F_{attr} = (P \odot Q)O \odot Y - (P \odot Q)Y$$
$$F_{rep} = (Q \odot Q)O \odot Y - (Q \odot Q)Y$$
Όπου Y ο $N \times 2$ πίνακας των ενσωματομένων σημείων και O ο $N \times 2$ πίνακας που κάθε στοιχείο του είναι ίσο με ένα.

- $$F_{attr} = (\text{diag}(\text{sum}((P \odot Q), 1)) - (P \odot Q)) * Y$$
$$F_{rep} = (\text{diag}(\text{sum}((Q \odot Q), 1)) - (Q \odot Q)) * Y$$
με την χρήση συμβολισμού matlab. Η συνάρτηση diag δημιουργεί έναν διαγώνιο πίνακα από ένα διάνυσμα εισόδου και η sum(,1) αθροίζει τα στοιχεία των γραμμών.

Αξίζει να σημειωθεί ότι οι ελκτικές δυνάμεις εξαρτώνται από της ομοιότητες της εισόδου, ενώ οι απωθητικές εξαρτώνται μόνο από τις σχετικές θέσεις των ενσωματωμένων σημείων. Από τα παραπάνω συμπεραίνουμε την ερμηνεία ότι ο ελκτικός όρος οδηγεί το y_i σε έναν σταθμισμένο μέσο όρο των υπολοίπων y_j με βάρη μεγαλύτερα για σημεία τα οποία είναι πιο κοντά στο x_i στον πολυδιάστατο χώρο. Ενώ ο απωθητικός διαχωρίζει τα σημεία όταν είναι κοντά στον ενσωματωμένο χώρο.

1.4 Υπερπάρμετροι του Αλγορίθμου

1.4.1 Perplexity

Η πιο σημαντική υπερπαράμετρος που χρησιμοποιεί ο αλγόριθμος t-SNE είναι η perplexity. Η Perplexity χρησιμοποιείται για των καθορισμό των σ_i στην έκφραση των βαθμών ομοιότητας του πολυδιάστατου χώρου. Απλοϊκά, καθορίζει πως να εξισορροπήσουμε την προσοχή μας ανάμεσα στα τοπικά και καθολικά χαρακτηριστικά των δεδομένων. Και μπορεί να ερμηνευτεί ως μία υπόθεση για τον αριθμό των κοντινών γειτόνων που διαθέτει κάθε σημείο. Για να εκμεταλλευτούμε την μέθοδο t-SNE πρέπει να αναλύσουμε πολλαπλές απεικονίσεις για διαφορετικές τιμές της perplexity. Οι συνήθειες τιμές της παραμέτρου κυμαίνονται από 5 ως 50.

1.4.2 Πρώιμη και Τελική Υπερβολή

Όπως αναφέραμε στην προηγούμενη παράγραφο τη κλίση μπορεί να ερμηνευτεί ως ένα άθροισμα ελκτικών και απωθητικών δυνάμεων που κινούν τα σημεία στον ολιγοδιάστατο χώρο. Μία από τις προκλήσεις που έχει η εφαρμογή αυτού του αλγόριθμου είναι ότι η σύγκλιση επιβραδύνει όταν ο αριθμός των σημείων αυξάνεται. Ένας τρόπος ώστε να βελτιωθεί η διαδικασία της βελτιστοποίησης είναι να πολλαπλασιαστεί ο ελκτικός όρος με μία υπερπαράμετρο a, που ονομάζεται παράμετρος πρώιμης υπερβολής (early exaggeration).

Αυτή η τεχνική συχνά χρησιμοποιείται μόνο για τις πρώτες εκατοντάδες επαναλήψεις. Με αυτό το κόλπο ο t-SNE αναγνωρίζει καλύτερα την συνολική δομή των δεδομένων, δημιουργώντας πιο συγκεντρωμένες συστάδες σημείων που μπορούν να κινηθούν στον ολιγοδιάστατο χώρο. Συνηθείς τιμές για την παράμετρο a κυμαίνονται ανάμεσα σε 4 και

12. Καθώς αυξάνουμε την τιμή της παραμέτρου παρατηρούμε ότι η τελική απεικόνιση γίνεται πιο μικρή σε έκταση και πιο κυκλική.

Αυτή η τεχνική μπορεί επίσης να χρησιμοποιηθεί στις τελευταίες επαναλήψεις ώστε να μικρύνει το μέγεθος των συστάδων και να τις κάνει πιο διακριτές. Τελειώνοντας αξίζει να σημειωθεί ότι η χρήση της παραμέτρου early exaggeration μπορεί να μελετηθεί αυστηρά και υπάρχει κατάλληλη επιλογή των παραμέτρων a και h (βήμα της gradient descent ή ρυθμός μάθησης) που οδηγεί σε εκθετική σύγκληση και ακριβή διαχωρισμό των συστάδων με ασθενείς υποθέσεις [3].

1.5 Σύνοψη κεφαλαίων

Η εργασία αυτή περιγράφει μια βελτιστοποίηση της απόδοσης του αλγορίθμου t-SNE με την χρήση Μονάδων Επεξεργασίας Γραφικών (GPU) βασισμένη στην εργασία [4]. Αρχικά στο 2ο κεφάλαιο θα αναλύσουμε διάφορες μεθόδους προσέγγισης της κλίσης του t-SNE που χρησιμοποιούνται για την επιτάχυνση της μεθόδου καθώς και χαρακτηριστικά των αποδοτικότερων ως σήμερα υλοποιήσεων. Έπειτα στο 3ο κεφάλαιο θα παρουσιάσουμε τα χαρακτηριστικά της υλοποίησης GPU που έχουμε αναπτύξει, επίσης θα περιγράψουμε και θα αιτιολογήσουμε την χρήση μίας υβριδικής υλοποίησης για αποδοτικά πολυπύρηνα συστήματα που αξιοποιεί την CPU και την GPU. Στο κεφάλαιο 4 θα παρουσιάσουμε αποτελέσματα σύγκρισης των υλοποιήσεών μας με τις πλέον διαδεδομένες υλοποιήσεις για μία μεγάλη γκάμα παραδειγμάτων. Τέλος, στα παραρτήματα παρουσιάζονται συνοπτικά τα τμήματα του κώδικα που είναι απαραίτητα για την κατανόηση της υλοποίησης και την επαναχρησιμοποίηση της σε περίπτωση που απαιτηθεί.

Κεφάλαιο 2

Προσέγγιση της Παραγώγου

2.1 Υπολογισμός της Πολυπλοκότητας

Ας μελετήσουμε την πολυπλοκότητα t-SNE. Πρώτα από όλα εκτελούμε gradient descent σε μία μη κυρτή συνάρτηση οπότε δεν μπορούμε να έχουμε ακριβή ανάλυση της σύγκλισης του αλγορίθμου. Παρόλα αυτά έχουν αποδειχτεί θεωρητικά αποτελέσματα τα οποία που υποστηρίζουν ότι ο t-SNE έχει καλές ιδιότητες σύγκλισης.

Για να εκτελέσουμε την gradient descent πρέπει να υπολογίζουμε την τιμή της παραγώγου σε κάθε επανάληψη. Από την έκφραση (1) συμπεραίνουμε ότι ο αφελής υπολογισμός της παραγώγου μπορεί να γίνει σε χρόνο $O(n^2)$. Όμως για μεγάλα σύνολα δεδομένων προβλήματα αυτής της πολυπλοκότητας είναι ανέφικτο να λυθούν. Στις επόμενες παραγράφους θα μειώσουμε αυτόν τον χρόνο χρησιμοποιώντας προσεγγίσεις της παραγώγου και αξιοποιώντας την δομή του προβλήματος.

2.2 Ελκτικές Δυνάμεις

Στην πράξη ο υπολογισμός των βαθμών (βαρών) ομοιότητας όλων των σημείων του πολυδιάστατου χώρου, δηλαδή ο υπολογισμός του πίνακα P, είναι πολύ χρονοβόρος. Αντί αυτού θα προσεγγίσουμε τον Ελκτικό υπολογισμό χρησιμοποιώντας για κάθε σημείο μόνο τα βάρη των k κοντινότερων γειτόνων του. Αυτή η προσέγγιση είναι δικαιολογημένη καθώς τα πραγματικά βάρη μειώνονται εκθετικά με το τετράγωνο της απόστασης. Οπότε μετά από μία συγκεκριμένη απόσταση οι τιμές των βαρών θα προσεγγίζουν το μηδέν. Έτσι η παράληψη τιμών αυτών δεν θα αλλάζει σημαντικά τον υπολογισμό της παραγώγου και ως αποτέλεσμα την απεικόνιση εξόδου. Με άλλα λόγια οι κοντινότεροι γείτονες αποτυπώνουν την τοπική δομή των δεδομένων.

Με την χρήση των κοντινότερων γειτόνων, αφού $n \gg k$, έχουμε σε μία αραιή προσέγγιση του P. Οπότε αφού ο νέος πίνακας P είναι αραιός μπορούμε να επιταχύνουμε τον υπολογισμό του Ελκτικού όρου χρησιμοποιώντας μόνο τα μη μηδενικά στοιχεία. Επιπλέον λόγο της φύσης της έκφρασης του Ελκτικού όρου (2) ο υπολογισμός του παρομοιάζει την ροή υπολογισμού ενός γινόμενου αραιού πίνακα με διάνυσμα (Sparse Matrix Vector Product ή αλλιώς SpMV). Ο υπολογισμός των κοντινότερων γειτόνων μπορεί να υλοποιηθεί με την χρήση k-d ή Vantage-point [5] δέντρων ή αλλιώς μπορεί να εκτελεστεί προσεγγιστικός υπολογισμός των κοντινότερων γειτόνων για ακόμα μεγαλύτερη ταχύτητα. Η τελική πολυπλοκότητα για τον υπολογισμό του ελκτικού όρου είναι $O(k \cdot n)$.

$$F_{attr,i} = \sum_{j \neq i} p_{ij} q_{ij} Z(y_i - y_j) \tag{2}$$

Πειραματικά διαπιστώνεται ότι αν $k \geq 3 \cdot u$, όπου u η καθορισμένη από το χρήστη perplexity, τότε ο αριθμός γειτόνων είναι αρκετός για μία έμπιστη προσέγγιση. Πιο συγκεκριμένα επανακαθορίζουμαι τις ανά δύο ομοιότητες ανάμεσα στα αντικείμενα εισόδου ως:

$$p_{j|i} = \begin{cases} \frac{\exp(-d(x_i,x_j)^2/2\sigma_i^2)}{\sum_{k \in N_i} \exp(-d(x_i,x_k)^2/2\sigma_i^2)}, j \in N_i \\ 0, \text{ αλλιώς} \end{cases} \quad \text{και } p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

Όπου N_i είναι το σύνολο των k κοντινότερων γειτόνων του x_i.

Οι υπολογισμοί γινομένου αραιού πίνακα διάνυσμα συχνά κυριαρχούν τον χρόνο εκτέλεσης πολλών επιστημονικών και μηχανικών εφαρμογών. Καθώς παρουσιάζουν μικρά ποσοστά βέλτιστης απόδοσης μηχανής (machine peak performance) λόγω της μη αποδοτικής χρήσης της κρυφής μνήμης και της δεσμευμένης στη μνήμη (memory-bound) φύσης τους. Η βελτίωση του χρόνου ενός SpMV περιλαμβάνει την επιλογή της κατάλληλης δομής δεδομένων και μοτίβου προσπέλασης για τον αραιό πίνακα. Συχνές δομές δεδομένων που χρησιμοποιούνται είναι Λίστα Συντεταγμένων (COO) όπου αποθηκεύονται μόνο τα μη μηδενικά στοιχεία και οι δείκτες γραμμής και στήλης σε τρία διανύσματα και Compressed sparse row (CSR) όπου επεκτείνοντας στην δομή COO, συμπυκνώνονται τα δεδομένα του διανύσματος γραμμής αποθηκεύοντας μόνο δείκτες που υποδεικνύουν την αρχή της κάθε γραμμής.

Διαφορετικά σύνολα δεδομένων έχουν διαφορετικές απαιτήσεις μνήμης και διαφορετικό μοτίβο προσπέλασης όταν υπολογίζουμε ένα SpMV. Η κύρια πρόκληση που συναντάμε όταν εκτελούμε υπολογισμούς με αραιούς πίνακες, ιδιαίτερά σε GPU, είναι ότι έχουμε οι μη ευθυγραμμισμένες ακανόνιστες προσπελάσεις στη μνήμη που μειώνουν την απόδοση.

Επιπλέον αξίζει να σημειωθεί ότι στην περίπτωση του t-SNE υπολογίζουμε τον Ελκτικό όρο σε κάθε επανάληψη μέχρι την σύγκλιση. Οπότε θα ήταν αποδοτικό να μετασχηματίσουμε την δομή των δεδομένων ώστε να έχουμε μία πιο κανονικά μοτίβα προσπέλασης μνήμης και παραλληλισμού. Αν το SpMV του καινούριου πίνακα έχει αρκετή επιτάχυνση ως προς το SpMV με τη χρήση των δομών COO και CSR τότε το κόστος αυτού του μετασχηματισμού θα εξαργυρωθεί με τον αριθμό των επαναλήψεων και θα υπάρχει επιτάχυνση στο συνολικό χρόνο εκτέλεσης.

2.3 Απωθητικές Δυνάμεις

Σε αυτήν την παράγραφο θα εξετάσουμε την μέθοδο προσέγγισης των απωθητικών δυνάμεων που χρησιμοποιείται από τις υλοποιήσεις FIt-SNE [6] και SG-t-SNE-Π [4]. Ο υπολογισμός του απωθητικού όρου είναι πολύ διαφορετικός από αυτόν του ελκτικού. Ο πίνακας $Q \odot Q$ δεν μπορεί να προσεγγιστεί από έναν αραιό πίνακα επειδή τα στοιχεία του είναι οι βαθμοί ομοιότητας των ενσωματωμένων σημείων τα οποία αλλάζουν κάθε επανάληψη. Επιπλέον η t-κατανομή διαθέτει μεγαλύτερή ουρά από την κανονική κατανομή και έτσι δεν μπορούμε να κάνουμε την ίδια προσέγγισή με παραπάνω. Άρα πρέπει να διαχειριστούμε τον υπολογισμό $Q \odot Q$ ως ένα πυκνό υπολογισμό,

ευτυχώς υπάρχουν μέθοδοι αριθμητικής ανάλυσης που μπορούν να επιταχύνουν αυτό το πρόβλημα.

Αρχικά ας γράψουμε την έκφρασή του απωθητικού όρου με τον συμβολισμό που θα χρησιμοποιήσουμε στις επόμενες παραγράφους. Έχουμε:

$$F_{rep,i} = \sum_{j=1 j\neq i}^{n} \frac{y_i - y_j}{(1 + \|y_j - y_i\|^2)^2} \cdot \frac{1}{\sum_{k\neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$

Μπορούμε να υπολογίσουμε την έκφραση αυτή υπολογίζοντας αθροίσματα της μορφής:

$$u_i = \sum_{j=1}^{N} G(y_i, y_j) v_j$$

όπου G είναι είτε $G_1(y_i, y_j) = \frac{1}{1+\|y_i - y_j\|^2}$ ή $G_2(y_i, y_j) = \frac{1}{(1+\|y_i - y_j\|^2)^2}$. Πιο συγκεκριμένα για δισδιάστατη απεικόνιση υπολογίζουμε την F_{rep} από τα:

$$h_{1,i} = \sum_{\substack{j=1 \\ j\neq i}}^{N} G_1(y_i, y_j)$$

$$h_{2,i} = \sum_{\substack{j=1 \\ j\neq i}}^{N} G_2(y_i, y_j) y_j(1)$$

$$h_{3,i} = \sum_{\substack{j=1 \\ j\neq i}}^{N} G_2(y_i, y_j) y_j(2)$$

$$h_{4,i} = \sum_{\substack{j=1 \\ j\neq i}}^{N} G_2(y_i, y_j)$$

Ως

$$F_{rep,i}(1) = (h_{2,i} - y_i(1) h_{4,i}) / Z$$
$$F_{rep,i}(2) = (h_{3,i} - y_i(2) h_{4,i}) / Z$$

όπου

$$Z = \sum_{j=1}^{N} h_{1,i}.$$

2.3.1 Fast Multipole Methods

Ουσιαστικά ένα υποπρόβλημα του υπολογισμού της κλίσης είναι ο υπολογισμός αθροισμάτων της μορφής:

$$u_i = \sum_{j=1}^{N} G(y_i, y_j) v_j$$

στη δική μας περίπτωση για τον υπολογισμό των απωθητικών δυνάμεων $G(y_i, y_j) = \frac{1}{1+\|y_i - y_j\|^2}$. Πιο γενικά υπάρχουν μέθοδοι, fast multipole methods, που έχουν κατασκευαστεί με σκοπό να υπολογίσουν αθροίσματα της μορφής:

$$u_i = \sum_{j=1}^{N} G(t_i, y_j) v_j$$

τα σημεία $\{t_i\}_1^M$ ονομάζονται στόχοι ενώ τα σημεία $\{y_i\}_1^N$ ονομάζονται πηγές, και τα δύο έχουν διάσταση s. Ο πυρήνας $G(t_i, y_j)$ είναι μία συνάρτηση που εξαρτάται μόνο από την απόσταση των t_i και y_j. Μπορούμε αν παρατηρήσουμε ότι ο αφελής υπολογισμός αυτών των αθροισμάτων παίρνει $O(NM)$ (δευτεροβάθμιο) χρόνο.

Τοποθετώντας όλες τις τιμές του πυρήνα σε έναν πίνακα $A : A_{ij} = G(t_i, y_j)$ παρατηρούμε ότι $u = Av$. Είναι γνωστό ότι αν το πεδίο των $\{t_i\}_1^M$ είναι διαφορετικό από αυτό των $\{y_i\}_1^N$ τότε μπορούμε να διαχωρίσουμε τις μεταβλητές και να έχουμε $A \approx B(X)C(Y)$, όπου B ένας $M \times P$ πίνακας και C ένας $P \times N$ πίνακας. Αυτό ισχύει καθώς αν έχουμε διαφορετικά πεδία τότε ο πυρήνας μπορεί να προσεγγιστεί από έναν πίνακα μικρής τάξης. Οπότε για αυτήν την περίπτωση έχουμε:

$$\begin{aligned} u_i &= \sum_{j=1}^{N} G(t_i, y_j) v_j \\ &= \sum_{j=1}^{N} \sum_{k=1}^{P} B_{ik} C_{kj} v_j \\ &= \sum_{k=1}^{P} B_{ik} \Big(\sum_{j=1}^{N} C_{kj} v_j \Big) \\ &= \sum_{k=1}^{P} B_{ik} m_k \end{aligned}$$

Αξίζει να σημειωθεί ότι τα m_k υπολογίζονται μόνο μία φορά για όλα τα i,που θα πει ότι τα αθροίσματα μπορούν να υπολογιστούν με $P \cdot (N + M)$ πράξεις που αποτελεί τεράστια βελτίωση από το αρχικό $M \cdot N$.

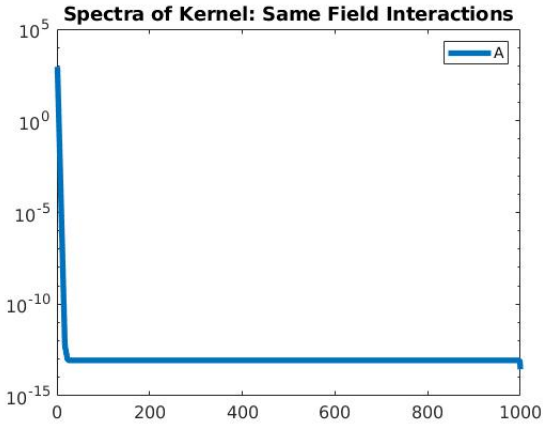
2.3.2 Διαχωρισμός Μεταβλητών

Σε αυτήν την παράγραφο θα εξετάσουμε έναν τρόπο ώστε να επιτύχουμε έναν διαχωρισμό μεταβλητών όπως τον παραπάνω. Θα παρουσιάσουμε τον βέλτιστο διαχωρισμό, βέλτιστο ως προς την ακρίβεια. Ας υποθέσουμε την ανάλυση του A σε ιδιάζουσες τιμές (Singular Value Decomposition ή αλλιώς SVD), $A \approx U\Sigma V^T$ χρησιμοποιώντας τις p μεγαλύτερες ιδιάζουσες τιμές. Έστω $\{u_i\}_1^p, \{v_i\}_1^p$ οι στήλες των U και V αντίστοιχα ενώ $\{\sigma_i\}_1^p$ οι ιδιάζουσες τιμές. Τότε

$$A \approx U\Sigma V^T = \sum_{k=1}^{p} \sigma_k u_k {v_k}^T$$

$$\implies A_{ij} = \sum_{k=1}^{p} \sigma_k u_k(i) {v_k}^T(j)$$

Αυτός είναι ένας διαχωρισμός μεταβλητών παρόμοιος με αυτόν της προηγούμενης παραγράφου. Ποιες είναι οι ιδιότητές του ως προς την ακρίβεια; Η SVD για τις p υψηλότερες ιδιάζουσες τιμές είναι η βέλτιστη προσέγγιση του πίνακα ως προς την L_2 νόρμα (θεώρημα Eckart-Young-Minsky). Το σφάλμα αυτής της προσέγγισης δίνεται ως έκφραση των ιδιαζουσών τιμών που δεν λάβαμε υπόψη μας. Στην δική μας περίπτωση ο πυρήνας είναι αρκετά μικρής τάξης και έτσι η προσέγγιση αυτή είναι εξαιρετικά ακριβής.



Εικόνα 2.1: Ιδιάζουσες τιμές πυρήνα 1000 τυχαίων σημείων

Επίσης αξίζει να σημειωθεί ότι επειδή στην περίπτωση μας το σύνολο των στόχων είναι το ίδιο με το σύνολο των πηγών έχουμε $U = V$ και άρα

$$A = V\Sigma V^T$$

2.3.3 Διαχωρισμός Μεταβλητών με τη χρήση Παρεμβολής

Όμως η εύρεση της SVD ενός πίνακα ακόμα και προσεγγιστικά είναι αρκετά χρονοβόρα, για τις p μεγαλύτερες ιδιάζουσες τιμές έχουμε χρόνο $O(N \cdot M \cdot p)$. Οπότε χρειαζόμαστε έναν άλλο τρόπο για να διαχωρίσουμε τις μεταβλητές. Για αυτό τον σκοπό θα χρησιμοποιήσουμε μία τεχνική παρεμβολής. Αρχίζοντας θα εργαστούμε για $s = 1$ (μονοδιάστατο χώρο απεικόνισης για πιο εύκολη κατανόηση) και έπειτα θα γενικεύσουμε σε περισσότερες διαστάσεις. Αντί την χρήση του πυρήνα θα χρησιμοποιήσουμε το Lagrange πολυώνυμο παρεμβολής του πυρήνα που ορίζεται από διαδοχικά ισαπέχοντα σημεία των πεδίων στόχου και πηγών.

Έστω R_y και R_z τα διαστήματα που καθορίζουν τα εύρη των σημείων πηγών και στόχων, δηλαδή $y_i \in R_y, \forall i = 1, \ldots, N$ και $z_i \in R_z, \forall i = 1, \ldots, M$. Και ας υποθέσουμε ότι $\tilde{z_1}, \ldots, \tilde{z_p}$ είναι διαδοχικά ισαπέχοντα σημεία στο R_z και $\tilde{y_1}, \ldots, \tilde{y_p}$ διαδοχικά ισαπέχοντα σημεία στο R_y. Ο αριθμός p των σημείων θα ελέγχει το σφάλμα της προσέγγισης της μεθόδου. Η χρησιμότητα της απαίτησης τα σημεία να είναι διαδοχικά ισαπέχοντα θα εξεταστεί στην επόμενη παράγραφο.

Έστω $L_{l,\tilde{y}}$ και $L_{l,\tilde{z}}$ να είναι τα Lagrange πολυώνυμα :

$$L_{l,\tilde{y}}(y) = \frac{\prod_{j\neq l}^{p}(y - \tilde{y_j})}{\prod_{j\neq l}^{p}(\tilde{y_l} - \tilde{y_j})} \text{ και } L_{l,\tilde{z}}(z) = \frac{\prod_{j\neq l}^{p}(z - \tilde{z_j})}{\prod_{j\neq l}^{p}(\tilde{z_l} - \tilde{z_j})}$$

Με αυτά τα πολυώνυμα μπορούμε να δημιουργήσουμε ένα πολυώνυμο παρεμβολής G_p για τον πυρήνα.

$$G_p(z,y) = \sum_{l=1}^{p}\sum_{j=1}^{p} G(\tilde{z_l},\tilde{y_j})L_{l,\tilde{z}}(z)L_{j,\tilde{y}}(y)$$

Έπειτα χρησιμοποιούμε αυτό το πολυώνυμο για προσέγγιση του G με και έτσι μπορούμε να προσεγγίσουμε τα

$$u_i = \sum_{j=1}^{N} G(t_i,y_j)v_j$$

από

$$\tilde{u_i} = \sum_{j=1}^{N} G_p(t_i,y_j)v_j.$$

Οπότε

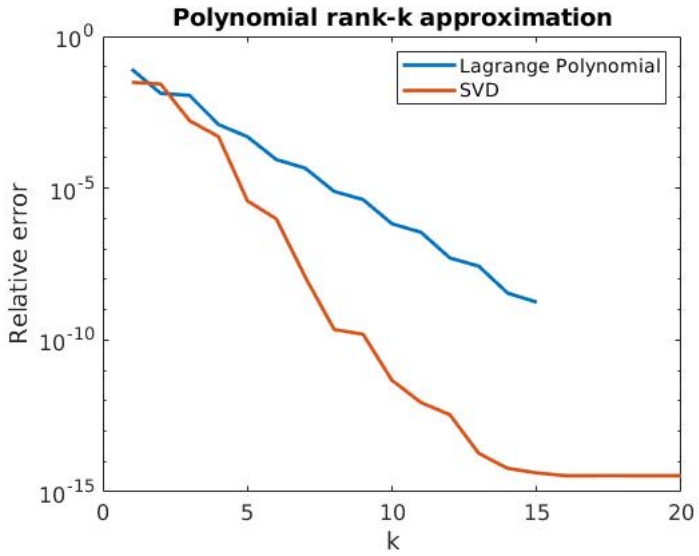
$$\begin{aligned}
\tilde{u_i} &= \sum_{j=1}^{N} G_p(t_i,y_j)v_j \\
&= \sum_{j=1}^{N}\sum_{l=1}^{p}\sum_{m=1}^{p} G(\tilde{z_l},\tilde{y_m})L_{l,\tilde{z}}(z_i)L_{m,\tilde{y}}(y_j)v_j \\
&= \sum_{l=1}^{p} L_{l,\tilde{z}}(z_i)\Big(\sum_{m=1}^{p} G(\tilde{z_l},\tilde{y_m})\Big(\sum_{j=1}^{N} L_{m,\tilde{y}}(y_j)v_j\Big)\Big).
\end{aligned}$$

Από αυτόν τον διαχωρισμό μεταβλητών μπορούμε να υπολογίσουμε τις τιμές με την χρήση τριών συνελήξεων σε χρόνο $O((M+N)p+p^2)$. Αλλά ποίες είναι οι ιδιότητες ακρίβειας αυτής της μεθόδου; Ισχύει ότι το σφάλμα γράφεται ως

$$\begin{aligned}
|u_i - \tilde{u_i}| &= |\sum_{j=1}^{N}(G(z_i,y_j) - G_p(z_i,y_j))\cdot v_j| \\
&\leq \sum_{j=1}^{N}|(G(z_i,y_j) - G_p(z_i,y_j))|\cdot|v_j| \\
&\leq \epsilon\sum_{1}^{N}|v_j|.
\end{aligned}$$

Δεδομένου ότι το πολυώνυμο προσεγγίζει ομοιόμορφα τον πυρήνα στα πεδία R_y και R_z. Δηλαδή

$$\exists\epsilon : \sup|G(z,y) - G_p(z,y)| \leq \epsilon$$



Εικόνα 2.2: Σχετικό σφάλμα SVD και Lagrange προσεγγίσεων.

2.3.4 Χρήση του γρήγορου μετασχηματισμού Φουριέ

Για το υπόλοιπο κείμενο θα υποθέσουμε ότι οι στόχοι και πηγές είναι το ίδιο σύνολο. Ο υπολογισμός της ζητούμενης συνέλιξης $\sum_{m=1}^{p} G(\tilde{y}_l, \tilde{y_m}) w_m$ μπορεί να εκφραστεί ως ένα γινόμενο πίνακα διάνυσμα $K \cdot w$, όπου K ο πίνακας των τιμών του πυρήνα και w το διάνυσμα των τιμών $\{w_m\}_1^p$. Επειδή τα σημεία $\tilde{y}_l$ είναι διαδοχικά ισαπέχοντα, ο πίνακας K είναι Toeplitz και άρα το γινόμενο μπορεί να υπολογιστεί αποτελεσματικά με την βοήθεια του γρήγορου μετασχηματισμού Φουριέ (FFT) σε χρόνο $O(p \log p)$. Αξίζει να σημειωθεί ότι ένα γινόμενο πίνακα με διάνυσμα παίρνει χρόνο $O(p^2)$ αλλά χρησιμοποιούμε τον FFT για να εκμεταλλευτούμε την δομή του πίνακα Toeplitz. Ο FFT μπορεί να χρησιμοποιηθεί με τον ακόλουθο τρόπο. Αρχικά έστω

$$K = \begin{bmatrix} t_0 & t_1 & \dots & t_{p-1} \\ t_1 & t_0 & \dots & \dots \\ \dots & \dots & \dots & t_1 \\ t_{p-1} & \dots & t_1 & t_0 \end{bmatrix}$$

κατασκευάζουμε τον πίνακα

$$C = \begin{bmatrix} K & B \\ B & K \end{bmatrix}$$

όπου

$$B = \begin{bmatrix} 0 & t_{p-1} & \dots & t_1 \\ t_{p-1} & 0 & \dots & \dots \\ \dots & \dots & \dots & t_{p-1} \\ t_1 & \dots & t_{p-1} & 0 \end{bmatrix}.$$

Ο πίνακας C είναι κυκλικός οπότε μπορούμε να υπολογίζουμε τα γινόμενα πίνακα διάνυσμα αυτού με την χρήση του FFT. Τέλος αν γεμίζουμε το διάνυσμα εισόδου με μηδενικά (zero padding) θα πάρουμε την επιθυμητή απάντηση.

$$C \cdot \begin{bmatrix} w \\ 0 \end{bmatrix} = \begin{bmatrix} Kw \\ Bw \end{bmatrix}$$

2.3.5 Αποφυγή του Γεμίσματος Μηδενικών (zero-padding)

Σε αυτή την παράγραφο θα μελετήσουμε πως να αποφύγουμε το zero-padding των διανυσμάτων και έτσι να μειώσουμε τις απαιτήσεις μνήμης του υπολογισμού FFT. Αυτή η μέθοδος εισήχθηκε πρώτα με την εργασία [4] και μείωσε κατά πολύ τις απαιτήσεις μνήμης κάνοντας δυνατή την απεικόνιση σε μεγαλύτερο αριθμό διαστάσεων. Το πρόβλημα του zero-padding γίνετε εμφανές όταν χρησιμοποιούμε περισσότερες διαστάσεις καθώς αυξάνονται οι απαιτήσεις μνήμης του αλγορίθμου κατά 2^d φορές, d η διάσταση του χώρου ενσωμάτωσης. Αρχικά θα δούμε πως μπορούμε να λύσουμε αυτό το πρόβλημα σε μία διάσταση όπου οι οπτικοποίηση είναι πιο εύκολη. Ανακεφαλαιώνοντας θέλουμε να εκτελέσουμε ένα Toeplitz γινόμενο πίνακα-διάνυσμα. Στην προηγούμενή παράγραφο είδαμε ότι αυτό μπορεί να υπολογιστεί από:

$$F^{-1}\Big(F(\begin{bmatrix}t_0 & \dots & t_{p-1} & 0 & t_{p-1} & \dots & t_1\end{bmatrix}) \odot F(\begin{bmatrix}w_0 & \dots & w_{p-1} & 0 & \dots & 0\end{bmatrix})\Big)$$

Όπου οι τιμές $t_0, \dots, t_{p-1}$ ορίζουν τον πίνακα Toeplitz και οι συναρτήσεις F και F^{-1} εκτελούν τον διακριτό μετασχηματισμό Φουριέ (Discrete Fourier transform ή DFT) και τον αντίστροφο DFT αντίστοιχα. Θα αποδείξουμε ότι αυτό μπορεί να υπολογιστεί από το b όπου:

$$b = \frac{b_1 + b_2}{2}$$

$$b_1 = F^{-1}\Big(F(\begin{bmatrix}t_0 & \dots & t_{p-1}\end{bmatrix}) + \begin{bmatrix}0 & t_{p-1} & \dots & t_1\end{bmatrix}) \odot F(\begin{bmatrix}w_0 & \dots & w_{p-1}\end{bmatrix})\Big)$$

$$b_2 = \bar{c} \odot F^{-1}\Big(c \odot (F(\begin{bmatrix}t_0 & \dots & t_{p-1}\end{bmatrix}) - \begin{bmatrix}0 & t_{p-1} & \dots & t_1\end{bmatrix})) \odot F(\begin{bmatrix}w_0 & \dots & w_{p-1}\end{bmatrix})\Big)$$

$$c = \begin{bmatrix}e^{\frac{2\pi i \cdot 0}{2n}} & \dots e^{\frac{2\pi i \cdot (p-1)}{2n}}\end{bmatrix} \text{ και } \bar{c} \text{ ο συζυγής του.}$$

Έστω $T_1 = \begin{bmatrix}t_0 & \dots & t_{p-1}\end{bmatrix}$ και $T_2 = \begin{bmatrix}0 & t_{p-1} & \dots & t_1\end{bmatrix}$.
Τώρα παίρνοντας τον DFT του zero-padded Toeplitz διανύσματος $T = \begin{bmatrix}t_0 & \dots & t_{p-1} & 0 & t_{p-1} & \dots & t_1\end{bmatrix}$ βλέπουμε ότι:

$$F(T)[k] = \sum_{j=0}^{2N-1} T[j] w^{jk}, w^{jk} = e^{\frac{-2\pi i}{2n} jk}$$

$$= \sum_{j=0}^{N-1} T_1[j] w^{jk} + (\sum_{j=0}^{N-1} T_2[j] w^{jk}) \cdot w^{kn}$$

$$= \begin{cases} F(T_1)[k] + F(T_2)[k], \text{ για k άρτιο} \\ F(c \odot T_1)[k] - F(c \odot T_2)[k], \text{ για k περιττό.} \end{cases}$$

Επειδή ισχύει $w^{2kj} = 1$ και $w^{(2k+1)j} = w^j$. Άρα η εκτέλεση του γινόμενου Hadamard με το διάνυσμα εισόδου μας δίνει:

$$(F(T) \cdot F(V))[k] = \begin{cases} (F(T_1 + T_2)[k]) \cdot F(v)[k], \text{ για k άρτιο} \\ F(c \odot (T_1 - T_2))[k] \cdot F(c \odot v)[k], \text{ για k περιττό.} \end{cases}$$

όπου v το μη zero-padded διάνυσμα εισόδου. Τώρα έχουμε τα πάντα για να υπολογίσουμε το γινόμενο πίνακα διάνυσμα. Άρα για μία οποιαδήποτε ακολουθία P έχουμε:

$$\begin{aligned} F^{-1}(P)[k] &= \sum_{j=0}^{2n-1} P(j) w^{-jk} \\ &= \begin{cases} (F^{-1}(P)[k]), \text{ για k άρτιο} \\ F(c \odot (T_1 - T_2))[k] \cdot F(c \odot v)[k], \text{ για k περιττό.} \end{cases} \end{aligned}$$

Με μία απλή διευθέτηση των όρων παίρνουμε την επιθυμητή έκφραση.

2.3.6 Βελτιώνοντας την Ακρίβεια

Η μέθοδος που περιγράφηκε παραπάνω δεν είναι αριθμητικά ευσταθής, και για μεγάλο αριθμό διαδοχικά ισαπέχοντων σημείων παρεμβολής, θα υποφέρει από το φαινόμενο του Runge. Το φαινόμενο του Runge δηλώνει ότι η αύξηση των σημείων παρεμβολής δεν οδηγεί πάντα σε καλύτερη ακρίβεια προσέγγισης καθώς ένα πολυώνυμο μεγάλου βαθμού έχει μεγάλες διακυμάνσεις ανάμεσα στα σημεία παρεμβολής. Οπότε αντί να χρησιμοποιήσουμε p σημεία παρεμβολής για όλο το διάνυσμα, μπορούμε να διαχωρίσουμε το διάστημα σε N_{int} υποδιαστήματα, με p σημεία παρεμβολής το καθένα. Συγκεκριμένα μπορούμε να υποδιαιρέσουμε ένα διάστημα σε N_{int} διαστήματα ίσου με μήκους $I_j, j = 1, \ldots, N_{int}$. Έστω $\tilde{y_{j,l}}$ να δηλώνει το j σημείο στο διάστημα I_l.

$$\tilde{y_{j,l}} = \frac{h}{2} + ((j-1) + (l-1) \cdot p) \cdot h$$

όπου $h = 1/(N_{int} \cdot p)$ το μήκος του κάθε διαστήματος. Ο αλγόριθμος για τον υπολογισμό του αθροίσματος είναι ο ίδιος με την παράγραφο 2.3.3 άλλα για την προσέγγιση του πυρήνα χρησιμοποιούμε τμηματικές πολυωνυμικούς παρεμβολείς για τα αντίστοιχα διαστήματα. Παρεμβάλλουμε το κάθε σημείο χρησιμοποιώντας τα σημεία παρεμβολής που βρίσκονται μέσα στο ίδιο διάστημα. Συνοψίζοντας:

Βήμα 1ο Για κάθε διάστημα $I_l, l = 1, \ldots, N_{int}$ υπολογίζουμε το του συντελεστές $w_{m,l}$ που ορίζονται από τον τύπο:

$$w_{m,l} = \sum_{y_j \in I_l} L_{m,l}(y_j) v_j$$

Βήμα 2ο Υπολογίζουμε τις τιμές $u_{m,n}$ στα διαδοχικά ισαπέχοντα σημεία $\tilde{y_{m,n}}$ από τον τύπο:

$$u_{m,n} = \sum_{j=1}^{N_{int}} \sum_{j=1}^{p} G(\tilde{y_{m,n}}, \tilde{y_{l,j}}) w_{l,j}$$

Βήμα 3ο Για κάθε σημείο y_i υπολογίζουμε την έξοδο από το διάστημα που ανήκει άρα δηλαδή αν $y_i \in I_l$ τότε:

$$f(y_i) = \sum_{j=1}^{p} L_{j,l}(y_i) u_{j,l}$$

Πρέπει αν σημειωθεί ότι αφού όλα τα σημεία παρεμβολής είναι διαδοχικά ισαπέχοντα το 2ο βήμα μπορεί να υπολογιστεί με τη χρήση του FFT όπως περιγράφηκε στις προηγούμενες παραγράφους.

2.3.7 Γενικεύοντας σε περισσότερες διαστάσεις

Σε αυτήν την παράγραφο θα περιγράψουμε πως μπορούμε να χρησιμοποιήσουμε τις μεθόδους που έχουμε αναπτύξει για τον υπολογισμό του απωθητικού όρου όταν έχουμε να κάνουμε απεικόνιση σε 2 διαστάσεις επιπλέον γενίκευση μετά σε περισσότερες είναι προφανής. Για να χρησιμοποιήσουμε αυτήν την μέθοδο παρεμβολής για δισδιάστατο χώρο ενσωμάτωσης θα χρησιμοποιήσουμε ένα πλέγμα σημείων παρεμβολής. Έστω ότι έχουμε p σημεία για κάθε γραμμή του πλέγματος άρα p^2 συνολικά. Παίρνουμε την προσέγγιση:

$$\begin{aligned}\tilde{u_i} &= \sum_{j=1}^{N} G_p(y_i, y_j) v_j \\ &= \sum_{j=1}^{N} \sum_{l=1}^{p^2} \sum_{m=1}^{p^2} G(\tilde{y_l}, \tilde{y_m}) L_{l,\tilde{y}}(y_i) L_{m,\tilde{y}}(y_j) v_j \\ &= \sum_{l=1}^{p^2} L_{l,\tilde{y}}(y_i) \Big(\sum_{m=1}^{p^2} G(\tilde{y_l}, \tilde{y_m}) \Big(\sum_{j=1}^{N} L_{m,\tilde{y}}(y_j) v_j \Big) \Big).\end{aligned}$$

Όπου τα u_i και v_j είναι δισδιάστατα διανύσματα. Και το L_m είναι το Lagrange πολυώνυμο για το m-οστό σημείο παρεμβολής.

$$L_m = L_{xm} L_{ym}$$

Όπου L_{xm}, L_{ym} το σημεία παρεμβολής για την κατεύθυνση x και y αντίστοιχα.

Θα εκτελέσουμε τον ίδιο αλγόριθμο που περιγράφηκε στην παράγραφο 2.3.3 με την ίδια σειρά των παρενθέσεων. Αλλά στην περίπτωσή μας έχουμε τον πίνακα S, $S_{ij} = G(\tilde{y}_i, \tilde{y}_j)$ ο οποίος δεν είναι πίνακας Toeplitz όπως στην μονοδιάστατη περίπτωση και έτσι δεν μπορούμε να χρησιμοποιήσουμε την ίδια μέθοδο για να εκμεταλλευτούμε τον FFT.

Αντί αυτού μπορούμε να παρατηρήσουμε ότι ο πίνακας των τιμών του πυρήνα είναι ένας πίνακας Toeplitz υποπινάκων με Toeplitz υποπίνακες (BTBT). Παραδείγματος χάρη όταν το πλέγμα μας περιέχει τέσσερα σημεία έχουμε:

$$S = \begin{bmatrix} s_0 & s_1 & s_1 & s_2 \\ s_1 & s_0 & s_2 & s_1 \\ s_1 & s_2 & s_0 & s_1 \\ s_2 & s_1 & s_1 & s_0 \end{bmatrix} = \begin{bmatrix} S_0 & S_1 \\ S_1 & S_0 \end{bmatrix}$$

Άρα χρειαζόμαστε ένα γρήγορο γινόμενο πίνακα διάνυσμα για πίνακες αυτής της δομής. Μπορούμε να υπολογίσουμε αυτό το γινόμενο με την βοήθεια του FFT καθώς τα στοιχεία εξόδου μπορούν αν εκφραστούν ως μία συνέλιξη των μη-περιττών σημείων του S και του διανύσματος zero-padded [7].

Αλλά η γραφή και η επεξεργασία του πλήρη BTBT πίνακα είναι αρκετά μη αποδοτική (ιδίως στη μνήμη). Ένας άλλος τρόπος για να εκτελέσουμε αυτό το γινόμενο με χρήση του FFT είναι να κατασκευάσουμε τον πίνακα της μορφής:

$$S' = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & s_2 & s_1 & s_2 \\ 0 & s_1 & s_0 & s_1 \\ 0 & s_2 & s_1 & s_2 \end{bmatrix}$$

ή γενικά:

$$S' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & s_{0,(p^2-1)} & \cdots & s_{0,(p\cdot(p-1)+1)} & s_{0,p\cdot(p-1)} & s_{0,(p\cdot(p-1)+1)} & \cdots & s_{0,(p^2-1)} \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & s_{0,(p-1)} & \cdots & s_{0,1} & s_{0,0} & s_{0,1} & \cdots & s_{0,(p-1)} \\ 0 & s_{0,(2p-1)} & \cdots & s_{0,(p+1)} & s_{0,p} & s_{0,(p+1)} & \cdots & s_{0,(2p-1)} \\ 0 & s_{0,(p-1)} & \cdots & s_{0,1} & s_{0,0} & s_{0,1} & \cdots & s_{0,(p-1)} \\ 0 & s_{0,(2p-1)} & \cdots & s_{0,(p+1)} & s_{0,p} & s_{0,(p+1)} & \cdots & s_{0,(2p-1)} \\ 0 & s_{0,(p^2-1)} & \cdots & s_{0,(p\cdot(p-1)+1)} & s_{0,p\cdot(p-1)} & s_{0,(p\cdot(p-1)+1)} & \cdots & s_{0,(p^2-1)} \end{bmatrix}$$

όπου $s_{0,i}$ η τιμή του πυρήνα για το 0-οστο και i-οστο σημείο του πλέγματος παρεμβολής (αυτές είναι μόνο μη-περιττές τιμές του S). Για να εκτελέσουμε τον πολλαπλασιασμό $S \cdot v$ με τη χρήση του S' και του FFT υπολογίζουμε την έκφραση:

$$R = \text{ifft2}\Big(\text{fft2}(S') \odot \text{fft2}(V)\Big)$$

όπου

$$V = \left[\begin{array}{c|c} O_{p\times p} & O_{p\times p} \\ \hline O_{p\times p} & \begin{bmatrix} v(1:p)^T \\ v(p+1:2p)^T \\ \dots \\ v(p(p-1)+1:p^2)^T \end{bmatrix} \end{array}\right]$$

Τα τελικά αποτελέσματα μας θα είναι ο πίνακας $R(1:p, 1:p)$ σε σειρά γραμμών. Αλλά γιατί αυτή η έκφραση είναι σωστή; Ας πάρουμε την ίδια μέθοδο μονοδιάστατη περίπτωση για τον BTBT πίνακα S. Έστω

$$C = \begin{bmatrix} B & S \\ S & B \end{bmatrix} \text{ where } B = \begin{bmatrix} 0 & S_1 \\ S_1 & 0 \end{bmatrix}$$

Ο πίνακας C είναι ένας πίνακας με κυκλικούς υποπίνακες και άρα μπορούμε να χρησιμοποιήσουμε τον FFT του κάθε υποπίνακα για επιτάχυνση του υπολογισμού του γινομένου.

$$C \cdot \begin{bmatrix} 0_{2\times 2} \\ w \end{bmatrix}$$

Ουσιαστικά ο παραπάνω FFT είναι ο μετασχηματισμός στηλών που γίνεται κατά τον δισδιάστατο FFT. Ας σημειώσουμε ότι ένας δισδιάστατος DFT μπορεί να γραφεί ως ένας μονοδιάστατος DFT στηλών και ένας γραμμών όπως ακολουθεί:

$$\text{fft2}(X) = \text{fft}(\text{fft}(X, [], 2), [], 1) = W^\star(xW^\star)$$

Αυτό εξηγεί τη δομή των στηλών του S'. Και ο μετασχηματισμός γραμμών υπολογίζει τα γινόμενα πίνακα διανύσματος για κάθε υποπίνακα στο γινόμενο. Αυτή η διαδικασία δεν αλλάζει όταν διαμερίσουμε το πλέγμα μας σε κουτιά καθώς τα σημεία όλων των κουτιών δημιουργούν ένα πλέγμα ισαπέχοντων σημείων. Ας γενικεύσουμε αυτή η τεχνική σε περισσότερες διαστάσεις με το παρακάτω θεώρημα.

Θεώρημα (Υπολογισμός-FFT). Έστω S ένας k-επιπέδων $BTTB$ πίνακας και v διάνυσμα. Τότε μπορούμε να υπολογίσουμε το γινόμενο πίνακα διάνυσμα με την ακόλουθη έκφραση:

$$S \cdot v = iFFTk\ (FFTk\ (S') \odot FFTk\ (V))$$

Όπου S' και V υπολογίζονται από τα S και v όπως έχει περιγραφεί παραπάνω.

Απόδειξη: Θα κάνουμε μαθηματική επαγωγή στο k.
Αρχικά ισχύει ότι στην βασική κατάσταση ο πίνακας S είναι ένας $n \times n$ Toeplitz και το v ένα $n \times 1$ διάνυσμα.
Έστω ο πίνακας

$$C = \begin{bmatrix} B & S \\ S & B \end{bmatrix} \text{ όπου } B = \text{Toeplitz}(S(1, n : -1 : 2))$$

Τότε

$$C \cdot \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} S \cdot v \\ B \cdot v \end{bmatrix}$$

Αλλά ο C είναι κυκλικός πίνακας και έτσι διαγωνοποιείται από τους πίνακες DFT, F και F^{-1}. Άρα

$$C \cdot \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} S \cdot v \\ B \cdot v \end{bmatrix} = F^{-1} \cdot \text{diag}(\text{DFT}(\text{C}(1,:))) \cdot F \cdot V = \text{ifft}(\text{fft}(S') \odot \text{fft}(V))$$

Τώρα έστω η πρόταση για $k+1$ με $l \times l$ το μέγεθος των κουτιών στο $k+1$-επίπεδο. Έστω

$$C = \begin{bmatrix} B & S \\ S & B \end{bmatrix} \text{ και } B = \text{Toepltz}(S(1 : l, n : -1 : 2)$$

Ο C είναι πίνακας με κυκλικούς υποπίνακες οπότε αν χρησιμοποιήσουμε τους πίνακες που αποτελούνται από υποπινάκες DFT και αντίστροφου DFT , F_b και F_b^{-1}, θα έχουμε:

$$C \cdot \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} S \cdot v \\ B \cdot v \end{bmatrix} = F_b^{-1}\text{Diag}(DFT(C(1 : l, :)))F_b \cdot V$$

Τώρα μπορούμε να χρησιμοποιήσουμε το επαγωγικό βήμα γιατί ο blockDiag$(DFT(C(1 : l, :))) \cdot V$ είναι ένα γινόμενο k-επίπεδων BTTB πινάκων και ενός διανύσματος στήλης V. Άρα

$$\begin{aligned} C \cdot \begin{bmatrix} 0 \\ v \end{bmatrix} &= F_b^{-1} \cdot \text{iFFTk}\ (\text{FFTk}\ (\text{Diag}(DFT(C(1 : l, :)))') \odot \text{FFTk}\ ((F_b \cdot V)') \\ &= \text{iFFT(k+1)}\ (\text{FFT(k+1)}\ (S') \odot \text{FFT(k+1)}\ (V')) \end{aligned}$$

2.4 Αποδοτικότερες Υλοποιήσεις

Σε αυτή την παράγραφο θα περιγράψουμε τα χαρακτηριστικά των πλέον αποδοτικότερων υλοποιήσεων t-SNE.

2.4.1 FIt-SNE

Η FFT-accelerated Interpolation-based t-SNE (FIt-SNE) είναι μία πολύ αποδοτική υλοποίηση t-SNE που εισήχθηκε με την εργασία [6]. Η υλοποίηση αυτή πρώτα εισήγαγε την μέθοδο υπολογισμού των απωθητικών δυνάμεων με τη χρήση του FFT και μείωσε σημαντικά τον χρόνο που χρειαζόταν για να δημιουργηθεί μία απεικόνιση t-SNE.

Αρχικά σε αυτήν την υλοποίηση ο ελκτικός υπολογίζεται με την αραιή προσέγγιση και με την αφελή χρήση της δομής CSR. Ο υπολογισμός αυτός δεν είναι ιδιαίτερα αποδοτικός καθώς ο πίνακας των βαθμών ομοιότητας P δεν αλλάζει κατά την διάρκεια των επαναλήψεων και πρέπει να εξετάσουμε πιθανές αλλαγές στη δομή του που μπορούν να οδηγήσουν σε μεγαλύτερη απόδοση. Επίσης το αφελής γινόμενο αραιού πίνακα διάνυσμα με την δομή CSR δεν παρουσιάζει καλά χαρακτηριστικά απόδοσης και ο υπολογισμός του ελκτικού όρου κυριαρχεί τον συνολικό χρόνο της εφαρμογής.

Ο απωθητικός όρος υπολογίζεται με την χρήση της μεθόδου παρεμβολής με τη βοήθεια του FFT για επιτάχυνση όπως περιγράφηκε στην παράγραφο 2.3.6. Όμως δεν χρησιμοποιείται η μέθοδος αποφυγής του γεμίσματος με μηδενικά και έτσι παρουσιάζονται προβλήματα μνήμης όταν απεικονίζουμε σε μεγάλες διαστάσεις (τρεις διαστάσεις). Τέλος αυτή η μέθοδος χρησιμοποιεί την βιβλιοθήκη Annoy για τον υπολογισμό των κοντινότερων γειτόνων.

2.4.2 SG-tSNE-Π

Η υλοποίηση SG-tSNE-Π [4] επιταχύνει σημαντικά την προηγούμενη μέθοδο βελτιώνοντας την απόδοση του υπολογισμού και του ελκτικού και απωθητικού όρου. Επίσης μειώνονται και οι απαιτήσεις μνήμης του συστήματος κάνοντας την εφικτή και αποδοτική την απεικόνιση σε μεγαλύτερες διαστάσεις.

Αρχικά για τον υπολογισμό του ελκτικού όρου χρησιμοποιείται η δομή δεδομένων CSB (Compressed Sparse Blocks [8]). Μία δομή που αποθηκεύει υποπίνακες του πίνακα και οδηγεί σε καλή χρήση της κρυφής μνήμης και εξαιρετική απόδοση. Συνήθως η χρήση δομών αποθήκευσης αραιών πινάκων που αποθηκεύουν υποπίνακες οδηγούν σε μεγάλο αριθμό περιττού γεμίσματος μηδενικών. Για την μείωση αυτού του γεμίσματος γίνονται μεταθέσεις του πίνακα με μία τεχνική διαμερισμού γράφων που ονομάζεται Nested Disection[9]. Η προετοιμασία του πίνακα για την μετατροπή του σε αυτή τη μορφή είναι χρονοβόρα αλλά ο χρόνος αυτός εξαργυρώνεται στον αριθμό επαναλήψεων που υπολογίζονται σε μία gradient descent και στον αριθμό εκτελέσεων της υλοποίησης στα ίδια δεδομένα. Όπως αναφέραμε στην παράγραφο 1.4.1 για να εκμεταλλευτούμε την μέθοδο t-SNE πρέπει να παράγουμε πολλαπλές απεικονίσεις για διαφορετικές τιμές της παραμέτρου perplexity.

Όσων αφορά τον υπολογισμό του απωθητικού όρου χρησιμοποιείται η ίδια τεχνική παρεμβολής με την προηγούμενη υλοποίηση μόνο που αυτή την φορά χρησιμοποιείται η τεχνική της παραγράφου 2.3.5 για την αποφυγή του γεμίσματος με μηδενικά. Έτσι μειώνονται οι απαιτήσεις της μνήμης της μεθόδου και είναι εφικτή η χρήσης μεγάλων

πλεγμάτων παρεμβολής ακόμα και σε τρεις διαστάσεις. Επίσης για την παρεμβολή του κάθε σημείου δεν χρησιμοποιούνται απαραίτητα τα σημεία παρεμβολής κουτιών που καθορίζονται στο πλέγμα αλλά τα κοντινότερα σημεία παρεμβολής στο σημείο. Αυτό κάνει την προσέγγιση πιο ακριβή ακόμα και με την χρήση μικρότερου αριθμού σημείων παρεμβολής. Τέλος για καλύτερη χρήση της κρυφής μνήμης όταν χρησιμοποιούνται μεγάλα πλέγματα γίνονται μεταθέσεις στα ενσωματωμένα σημεία με βάση τον δείκτη της μεγαλύτερης διάστασης του κουτιού που βρίσκονται [10]. Σε αυτές τις περιπτώσεις χρησιμοποιείται διαφορετική τεχνική παραλληλισμού που να εκμεταλλεύεται αυτές τις μεταθέσεις.

Επίσης γίνεται γενίκευση της μεθόδου t-SNE στην απεικόνιση αραιών γράφων. Κατά αυτή τη γενίκευση έχουμε ως είσοδο τον αραιό πίνακα γειτνίασης του γράφου με βάρη ομοιότητας για κάθε ακμή που καθορίζουν την ομοιότητα των κόμβων και συνιστούν κατανομή (αθροίζουν στο 1). Ο παραπάνω πίνακας χρησιμοποιείται ως ο πίνακας των πολυδιάστατων βαρών P και εκτελείται ο υπόλοιπος αλγόριθμος. Η παραπάνω τεχνική παράγει εντυπωσιακά αποτελέσματα απεικονίσεων που εμφανίζουν τοπικότητα όμοιων κόμβων [11].

2.4.3 t-SNE-CUDA

Η υλοποίηση t-SNE-CUDA [12] είναι η πλέον ταχύτερη υλοποίηση GPU. Η τελευταία έκδοση της χρησιμοποιεί την ίδια μέθοδο προσέγγισης της κλίσεις η υλοποίηση FIt-SNE.

Για τον υπολογισμού των ελκτικών δυνάμεων χρησιμοποιείται η δομή δεδομένων COO. Αυτή η δομή δεδομένων παρουσιάζει πάρα πολύ καλά αποτελέσματα στην εκτέλεση γινομένου πίνακα-διανύσματος με την GPU γιατί έχει πολύ φυσική μέθοδο παραλληλοποίησης και χρειάζεται ελάχιστη προεπεξεργασία δεδομένων, όμως όπως θα δούμε στο επόμενο κεφάλαιο δεν είναι η συνήθως βέλτιστη.

Για τον υπολογισμό του απωθητικού όρου χρησιμοποιείται η ίδια μέθοδος με την υλοποίηση FIt-SNE αλλά υλοποιείται σε CUDA με τη χρήση του cuFFT [13]. Όμως καθώς η κατασκευή σχεδίου εκτέλεσης του FFT από τον cuFFT είναι χρονοβόρα. Το πλέγμα παρεμβολής δεν αλλάζει ανάλογα με της ανάγκες ακρίβειας και είναι προκαθορισμένο για να γίνεται χρήση ενός μοναδικού σχεδίου cuFFT. Αυτό όμως μπορεί να οδηγήσει σε στην χρήση επιπλέον ή λιγότερων σημείων παρεμβολής σε σχέση με αυτά χρειάζονται.

Κεφάλαιο 3

Επιταχύνοντας τον t-SNE

3.1 Υλοποίηση σε GPU

Σε αυτήν την παράγραφο θα περιγράψουμε την υλοποιήσεις CUDA και τις μεθόδους επιτάχυνσης που αναπτύξαμε κατά τη διάρκεια της διπλωματικής εργασίας. Η υλοποίησή μας βασίζεται την μέθοδο SG-tSNE-Π.

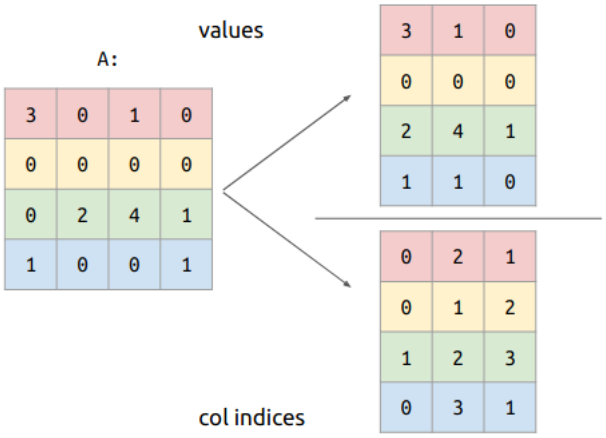
3.1.1 Επιτάχυνση Ελκτικού Υπολογισμού

Ισχύει ότι ο υπολογισμός του ελκτικού όρου, όπως θα δούμε στη συνέχεια, κυριαρχεί τον συνολικό χρόνο της εφαρμογής, ιδίως για μεγάλα σύνολα δεδομένων. Άρα η επιτάχυνση του είναι πολύ σημαντική για την μείωση του συνολικού χρόνου της εφαρμογής.

Η επιτάχυνση ενός γινομένου πίνακα διάνυσμα συχνά περιλαμβάνει την επιλογή της κατάλληλης δομής δεδομένων για την δομή του πίνακα. Ισχύει ότι δεν υπάρχει δομή δεδομένων η οποία να είναι η προτιμότερη σε όλες τις περιπτώσεις αλλά υπάρχουν δομές που να δουλεύουν καλύτερα σε μεγαλύτερο εύρος πινάκων. Επίσης καθώς ο υπολογισμός του ελκτικού όρου εκτελείται πολλαπλές φορές μπορούμε να χρησιμοποιήσουμε μία δομή αραιού πίνακα η οποία χρειάζεται μεγάλο χρόνο δημιουργίας καθώς ο χρόνος αυτός θα εξαργυρωθεί, αν είναι σημαντικά πιο γρήγορη από τις COO και CSR δομές (οι οποίες χρειάζονται ελάχιστη επεξεργασία για την δημιουργία τους).

Στην υλοποίησή μας επιλέγουμε την Υβριδική [14] (HYB) δομή δεδομένων για την αποθήκευση και επεξεργασία του αραιού πίνακα. Η δομή HYB παρατηρείται ότι είναι η συνήθως πιο αποδοτική δομή αραιού πίνακα για την εκτέλεση γινομένου αραιού πίνακα-διανύσματος με την χρήση GPU στην αρχιτεκτονική CUDA. Αυτή η δομή αποτελεί ένα συνδυασμό των δομών ELL (ELLPACK [15]) και COO.

Κατά την δομή ELL ένας $M \times N$ αραιός πίνακας με το πολύ K μη-μηδενικά στοιχεία ανά γραμμή, αποθηκεύεται σε έναν πυκνό πίνακα $M \times K$ που αποθηκεύει τα μη-μηδενικά δεδομένα του πίνακα και έναν $M \times K$ πυκνό πίνακα δεικτών που αποθηκεύει τους δείκτες στηλών των μη-μηδενικών. Όλες οι γραμμές γεμίζουν με μηδενικά έτσι ώστε να έχουν μήκος K. Στην Εικόνα 3.1 βλέπουμε ένα απλοϊκό παράδειγμα της δομής. Η ELL είναι πιο αποδοτική όταν ο μέγιστος αριθμός των μη-μηδενικών ανά γραμμή δεν διαφέρει αρκετά από τον μέσο, όπου αυτή είναι συχνά η περίπτωση για τυχαίους πίνακες και πίνακες που παράγονται από ημι-δομημένα πλέγματα.



Εικόνα 3.1: Παράδειγμα χρήσης της ELL δομής.

Τώρα όσον αφορά την δομή HYB, ενώ η δομή ELL είναι αποδοτική σε SIMD αρχιτεκτονικές, η απόδοσή της μειώνεται γρήγορα καθώς ο αριθμός των μη-μηδενικών του πίνακα κυμαίνεται. Σε αντίθεση με την απόδοση της COO δομής που είναι ανεξάρτητη από την κατανομή των μηδενικών. Για να έχουμε τα πλεονεκτήματα και των δύο δομών τις συνδυάζουμε σε μία υβριδική, HYB, δομή της ELL και COO. Συγκεκριμένα αποθηκεύουμε έναν προκαθορισμένο αριθμό μη-μηδενικών σε μία δομή ELL, με επιπλέον γέμισμα μηδενικών όπου χρειάζεται, και τα υπόλοιπα μη μηδενικά στοιχεία αποθηκεύονται σε μία COO δομή. Στην περίπτωσή μας αυτός ο προκαθορισμένος αριθμός καθορίζεται ως ο βέλτιστος, λαμβάνοντας υπόψη την διαφορά ταχύτητας εκτέλεσης υπολογισμού του γινομένου πίνακα-διάνυσμα των δομών ELL και COO, μέσω της βιβλιοθήκης Cusp [16]. Από πειραματικά αποτελέσματα ισχύει ότι μία πλήρως κατειλημμένη ELL δομή είναι περίπου τρεις φορές πιο γρήγορη από την αντίστοιχη COO για τον υπολογισμό γινομένου πίνακα διανυσματος, εκτός όταν ο αριθμός των γραμμών είναι μικρότερος από τέσσερις χιλιάδες [14]. Οπότε η χρήση της HYB δομής μπορεί να είναι αρκετά πιο αποδοτική από τη χρήση της COO δομής, που χρησιμοποιείται από την υλοποίηση t-SNE-CUDA, χωρίς ρίσκο αφού πέρα του προκαθορισμένου αριθμού τα υπόλοιπα στοιχεία του πίνακα αποθηκεύονται σε COO.

3.1.2 Επιτάχυνση Απωθητικού Υπολογισμού

Η υλοποίηση του απωθητικού υπολογισμού βασίζεται στη μέθοδο παρεμβολής με χρήση του FFT για επιτάχυνση της υλοποίησης SG-t-SNE-Π. Η μέθοδος αυτή διαχωρίζει τον υπολογισμού του απωθητικού όρου σε τέσσερα βήματα, τα S2G, G2G και G2S όπως στην παράγραφο 2.3.6 και Phi2Force για τη μετατροπή των αθροισμάτων σε δυνάμεις από την παράγραφο 2.3. Για τον υπολογισμό του απωθητικού όρου χρησιμοποιείται επίσης και η βιβλιοθήκη Thrust [17][18] για να εκτελεστούν συχνές διεργασίες όπως αθροίσματα και μέγιστα-ελάχιστα με μεθόδους reduction.

S2G

Αρχικά στο βήμα S2G, για κάθε σημείο του πλέγματος, m, υπολογίζουμε το άθροισμα που ορίζεται από τον ακόλουθο τύπο:

$$V_m = \sum_{y_j \in I_m} L_{m,j}(y_j) v_j$$

Όπου $L_{m,j}$ το πολυώνυμο Lagrange του σημείου m που ορίζεται από την παρεμβολή του σημείου y_j στα p κοντινότερα σημεία του πλέγματος. Και I_m το σύνολο των σημείων y_j που έχουν το m ως ένα από τα p κοντινότερα σημεία στο πλέγμα. Χρησιμοποιούνται σταθερά τα $p = 4$ κοντινότερα σημεία διότι παρουσιάζουν καλές ιδιότητες ακρίβειας και έχει γίνει ο προϋπολογισμός της έκφρασης του πολυωνύμου ώστε να έχουμε πλήρη βελτιστοποίηση.

Για αυτό το βήμα υπάρχει ένας προφανής παραλληλισμός ο οποίος χρησιμοποιείται επίσης και από την υλοποίηση t-SNE-CUDA. Και είναι για κάθε σημείο παράλληλα βρίσκουμε τα κοντινότερα σημεία του πλέγματος που πρέπει να ανανεώσουμε και ανανεώνουμε ατομικά αθροίζοντας το αντίστοιχο αποτέλεσμα στην θέση του πίνακα V. Ισχύει ότι οι ατομικές αθροίσεις στην CUDA έχουν πολύ καλή απόδοση. Επίσης λόγο του φυσικού παραλληλισμού της μεθόδου οι προσπελάσεις μνήμης που απαιτούνται είναι ευθυγραμμισμένες. Αυτά τα χαρακτηριστικά κάνουν την μέθοδό μας να έχει μεγάλη απόδοση. Παραθέτουμε ένα συνοπτικό απόσπασμα που περιγράφει την υλοποίηση της μεθόδου στο παράρτημα A.1.1.

G2G

Στο βήμα G2G για τα σημεία του πλέγματος $\tilde{y_k}$ υπολογίζουμε τις τιμές u_m από τον ακόλουθο τύπο:

$$u_m = \sum_{j=1}^{Ngrid} G(\tilde{y_m}, \tilde{y_j}) V_j$$

Όπως αναφέρθηκε προηγουμένως στην παράγραφο 2.3.7. Αυτή η συνέλιξη μπορεί να γίνει πολύ αποδοτικά με την χρήση του FFT. Για την υλοποίηση αυτού του βήματος χρησιμοποιούμε την βιβλιοθήκη cuFFT της Nvidia και την μέθοδο της παραγράφου 2.3.5 για την αποφυγή του γεμίσματος με επιπλέον μηδενικά ώστε να μπορούμε να κάνουμε απεικόνιση σε περισσότερες διαστάσεις με πολύ καλή ακρίβεια. Λόγο της αποδοτικότητάς του cuFFT και καθώς ο αριθμός των σημείων του πλέγματος που χρειάζονται για ικανοποιητική ακρίβεια με αυτήν την μέθοδο δεν είναι ιδιαίτερα μεγάλος (όπως θα δούμε στο επόμενο κεφάλαιο) ο χρόνος του βήματος G2G είναι σχεδόν αμελητέος όταν έχουμε μεγάλο αριθμό σημείων εισόδου.

G2S

Σε αυτό το βήμα για κάθε ενσωματωμένο σημείο y_i υπολογίζουμε την έξοδο από το διάστημα κοντινότερων σημείων του πλέγματος που ανήκε,ι άρα δηλαδή αν $y_i \in I_l$ τότε:

$$f(y_i) = \sum_{j=1}^{p} L_{j,l}(y_i) u_{j,l}$$

Όπου $L_{l,j}$ το πολυώνυμο Lagrange του σημείου l που ορίζεται από την παρεμβολή του σημείου y_j στα p κοντινότερα σημεία του πλέγματος. Αυτό το βήμα μπορεί να υλοποιηθεί σε GPU πολύ εύκολα υπολογίζοντας το άθροισμα για κάθε σημείο παράλληλα. Σε αντίθεση με το βήμα S2G σε αυτήν την περίπτωση δεν χρειαζόμαστε ατομική ανανέωση της εξόδου και επίσης έχουμε ευθυγραμμισμένες προσπελάσεις μνήμης, χαρακτηριστικά που κάνουν αυτό το βήμα ιδιαίτερα γρήγορο. Παραθέτουμε ένα συνοπτικό απόσπασμα που περιγράφει την υλοποίηση της μεθόδου στο παράρτημα A.1.2.

Phi2Force

Στη μέθοδο Phi2Force γίνεται μετατροπή της εξόδου της εξόδου της G2S στο αποτέλεσμα του Απωθητικού όρου όπως περιγράφεται στην παράγραφο 2.3. Αυτό το βήμα μπορεί να υλοποιηθεί σε GPU πολύ εύκολα και απαιτεί μόνο ευθυγραμμισμένες προσπελάσεις μνήμης, ως αποτέλεσμα είναι ιδιαίτερα γρήγορο.

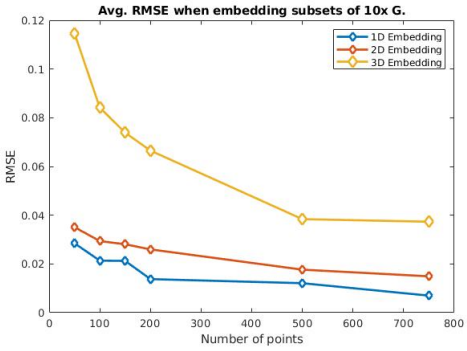
3.2 Επιλογή Μεγέθους σταθερού πλέγματος

Ισχύει ότι στην αρχιτεκτονική CUDA οι αναθέσεις, απελευθερώσεις μνήμης και η δημιουργία του σχεδίου για εφαρμογή του cuFFT είναι κλήσεις φραγής, δηλαδή η GPU και CPU πρέπει να συγχρονιστούν κάθε φορά πριν την εκτέλεση τους. Αυτό κάνει τη εκτέλεση της μεθόδου πιο αργή, ιδίως όταν ο αριθμός των σημείων εισόδου δεν είναι ιδιαίτερα μεγάλος. Για αυτό το λόγο στην εκτέλεσή μας διαθέτουμε την επιλογή στο χρήστη να χρησιμοποιήσει ένα σταθερό πλέγμα, με αριθμό σημείων της επιλογής του. Παρόλα αυτά οφείλουμε να παρέχουμε προκαθορισμένες, με ορισμένα κριτήρια βέλτιστες, τιμές πλέγματος στο χρήστη ώστε να διαθέτει μία εύρωστη εφαρμογή χωρίς να χρειάζεται εσωτερική γνώση των λεπτομερειών υλοποίησης. Σε αυτήν την παράγραφο θα καθορίσουμε αυτές τις τιμές. Οι τιμές θα χρησιμοποιηθούν στις επόμενες παραγράφους για να παραγωγή των αποτελεσμάτων αποδοτικότητας της υλοποίησης. Επίσης τα αποτελέσματα αυτής της παραγράφου μπορούν να αποτελέσουν ένα οδηγό με τον οποίο ο χρήστης να μπορεί να επιλέξει το μέγεθος του πλέγματος ανάλογα με τις απαιτήσεις του.

Η επιλογή του αριθμού των σημείων του πλέγματος πρέπει να είναι μία συνάρτηση της ακρίβειας και του χρόνου που χρειάζεται ο υπολογισμός. Ακόμα πρέπει να επιλεγεί ανεξάρτητα για διαφορετικές τιμές της διάστασης ενσωμάτωσης αφού τα χαρακτηριστικά ακρίβειας και χρόνου είναι διαφέρουν όταν αλλάζουμε τον αριθμό των διαστάσεων. Για να δούμε πως το σφάλμα αλλάζει με τον αριθμό των σημείων εισόδου και των αριθμό των σημείων του πλέγματος θα εκτελέσουμε τα πειράματα μας σε διάφορα υποσύνολα ενός μεγάλου πραγματικού συνόλου δεδομένων. Συγκεκριμένα θα χρησιμοποιήσουμε τα 50k, 100k, 150k, 250k, 500k, 750k και 1.3M δειγματοληπτημένα υποσύνολα του συνόλου δεδομένων από την 10X Genomics που περιέχει 1.3 εκατομμύρια δεδομένα κυττάρων εγκεφάλου ποντικιών.

Το σφάλμα εξαρτάται άμεσα από την έκταση της περιοχής ενσωμάτωσης. Για να αναπαραστήσουμε το σφάλμα μίας ολόκληρης απεικόνισης με έναν μοναδικό αριθμό χρησιμοποιούμε ως στατιστικό σφάλματος το μέσο, χωρίς την περίοδο της πρώιμης υπερβολής, Απωθητικό RMSE (Root Mean Square Error $RSME = \frac{\|exact-aprox\|}{\|exact\|}$). Μετρημένο σε μία από κάθε 50 επαναλήψεις της gradient descent.

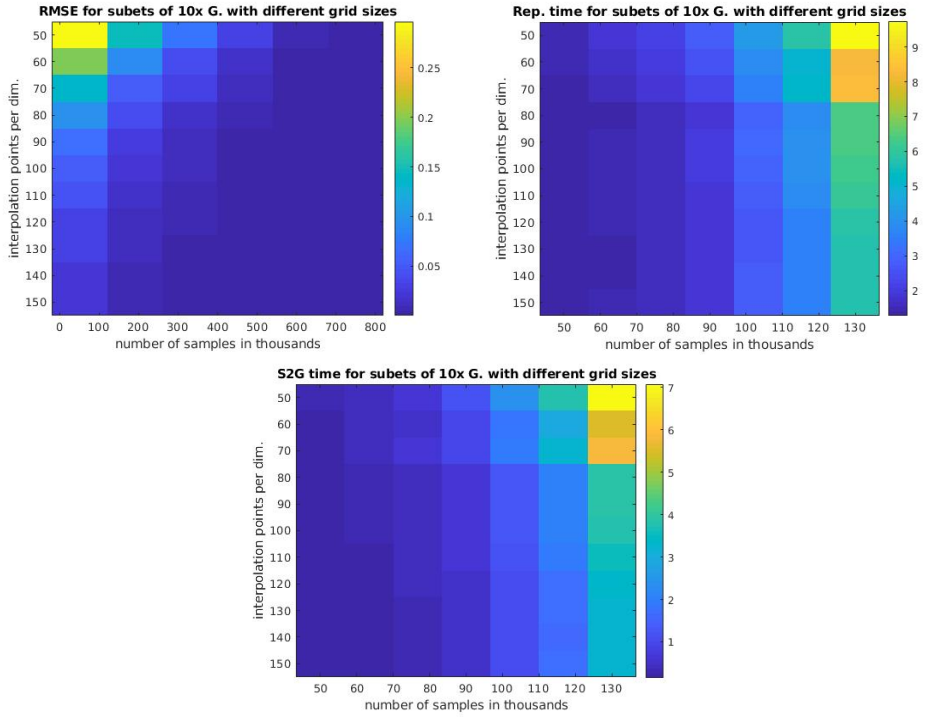
Πρώτα από όλα όμως θα αναπαραστήσουμε την ακρίβεια της SG-tSNE-Π, μίας εύρωστης υλοποίησης χρησιμοποιεί μεταβλητό πλέγμα, με το στατιστικό σφάλματος που θα χρησιμοποιήσουμε. Έτσι θα έχουμε μία αναφορά για τις τιμές ακρίβειας που θα συναντήσουμε παρακάτω.



Εικόνα 3.2: Μέσο απωθητικό RMSE της Υλοποίησης SG-tSNE-Π κατά την απεικόνιση υποσυνόλων του 10x Genomics

3.2.1 Μονοδιάστατες Απεικονίσεις

Παρακάτω βλέπουμε δύο χάρτες θερμότητας (heatmaps) έναν του Μέσου απωθητικού RMSE και έναν του χρόνου του απωθητικού υπολογισμού. Ο κατακόρυφος άξονας αναπαριστά τον χρόνο τον αριθμό των σημείων παρεμβολής ανά διάσταση που χρησιμοποιήθηκε, συγκεκριμένα χρησιμοποιήθηκαν $10 \cdot i, 50 \leq i \leq 150$ σημεία παρεμβολής ανά διάσταση για να παραχθούν αυτά τα διαγράμματα.

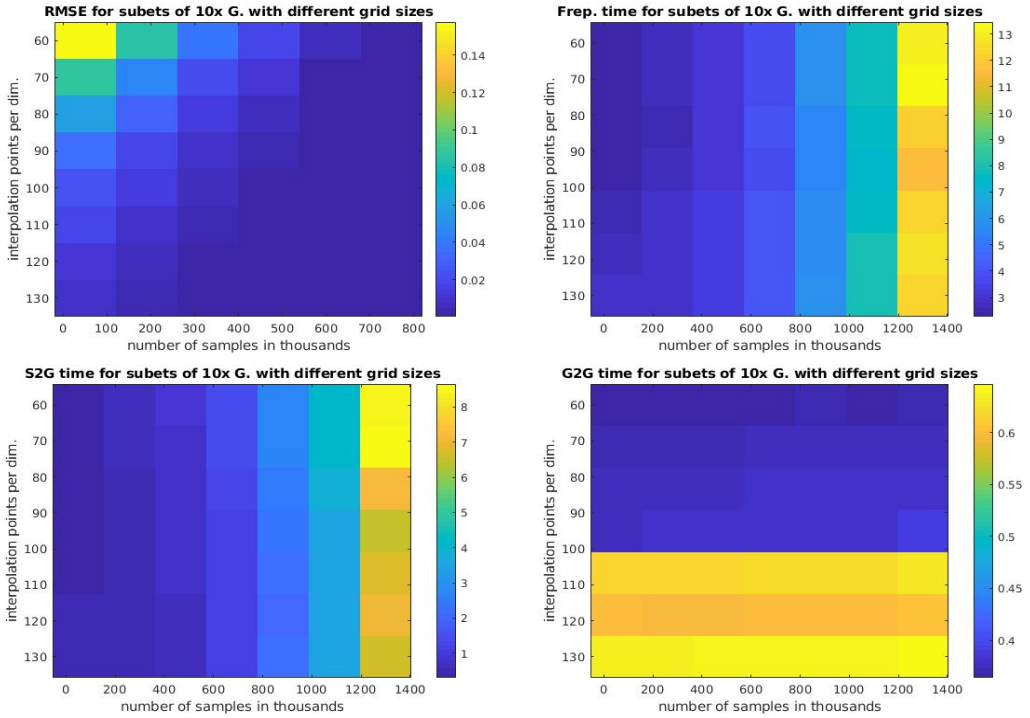


Εικόνα 3.3: Ανάλυση χρόνου και σφάλματος μονοδιάστατων απεικονίσεων για σταθερό πλέγμα.

Από της παραπάνω φιγούρες παρατηρούμε ότι το μεγαλύτερο ποσοστό χρόνου του Απωθητικού υπολογισμού χρησιμοποιείται για τον S2G μετασχηματισμό. Ο χρόνος άλλων λειτουργιών του Απωθητικού υπολογισμού είναι αμελητέος. Επειδή ο S2G υλοποιείται με ατομικές πράξεις παρατηρούμε ότι ο απαιτούμενος χρόνος μειώνεται όταν μεγαλώνουμε τον αριθμό των σημείων παρεμβολής, αφού έχουμε μικρότερο αριθμό συγκρούσεων. Από τα παραπάνω μπορούμε να επιλέξουμε ως αριθμό σημείων παρεμβολής το 150 ώστε να έχουμε μέγιστη ακρίβεια.

3.2.2 Δισδιάστατες Απεικονίσεις

Παρακάτω ακολουθούν τέσσερις heatmaps έναν για το Απωθητικό RMSE, έναν για τον χρόνο που χρειάστηκε από τον απωθητικό υπολογισμό. Και για να έχουμε μία πιο αναλυτική εικόνα αναπαριστούμε και τον χρόνο που χρησιμοποιήθηκε για τους S2G και G2G μετασχηματισμούς. Ο κατακόρυφος άξονας αναπαριστά τον αριθμό των σημείων παρεμβολής ανά διάσταση που χρησιμοποιήθηκε, συγκεκριμένα χρησιμοποιήθηκαν $10 \cdot i, 60 \leq i \leq 130$ σημεία παρεμβολής ανά διάσταση για να παραχθούν αυτά τα διαγράμματα.



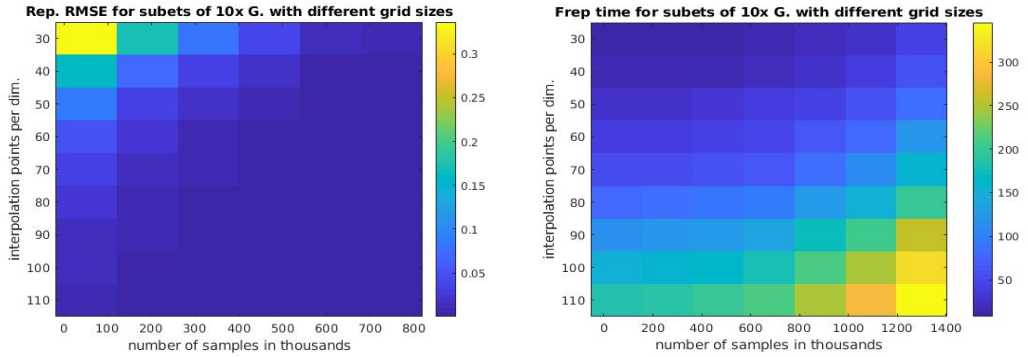
Εικόνα 3.4: Ανάλυση χρόνου και σφάλματος δισδιάστατων απεικονίσεων για σταθερό πλέγμα.

Από αυτά τα διαγράμματα μπορούμε να παρατηρήσουμε ότι ενώ ο χρόνος του S2G μετασχηματισμού μειώνεται όταν ο αριθμός των σημείων παρεμβολής αυξάνεται ο χρόνος του G2G μετασχηματισμού αυξάνεται. Επιπλέων μπορούμε να διαπιστώσουμε ότι αυτό το φαινόμενο εξισορροπείται όταν ο αριθμός των σημείων παρεμβολής ανά διάσταση είναι 90.

Έτσι από τα παραπάνω έχοντας υπόψη τα αντίστοιχα αποτελέσματα της υλοποίησης SG-tSNE-Π μπορούμε να επιλέξουμε ως αριθμό σημείων παρεμβολής για δισδιάστατη απεικόνιση 90 σημεία ανά διάσταση. Καθώς για αυτό τον αριθμό παρατηρούμε ικανοποιητικό χρόνο υλοποίησης και ακρίβεια.

3.2.3 Τρισδιάστατες Απεικονίσεις

Στην Εικόνα 3.5 βλέπουμε δύο heatmaps έναν που αναπαριστά το σφάλμα RMSE και έναν τον χρόνο που χρειάστηκε για τον υπολογισμό του Απωθητικού όρου. Ο κατακόρυφος άξονας αναπαριστά τον αριθμό των σημείων παρεμβολής ανά διάσταση που χρησιμοποιήθηκε, συγκεκριμένα χρησιμοποιήθηκαν $10 \cdot i, 30 \leq i \leq 110$ σημεία παρεμβολής ανά διάσταση για να παραχθούν αυτά τα διαγράμματα. Σε αυτήν την περίπτωση χρησιμοποιούμε μικροτέρα μεγέθη πλέγματος από της προηγούμενες καθώς ο χρόνος του απωθητικού υπολογισμού αυξάνεται πιο γρήγορα από τις προηγούμενες περιπτώσεις καθώς αυξάνουμε το μέγεθος του πλέγματος.



Εικόνα 3.5: Ανάλυση χρόνου και σφάλματος τρισδιάστατων απεικονίσεων για σταθερό πλέγμα.

Από αυτά τα διαγράμματα μπορούμε να διαπιστώσουμε ότι η επιλογή 50 σημείων παρεμβολής ανά διάστασης μας δίνει ικανοποιητική ακρίβεια, παρόμοια της υλοποίησης SG-tSNE-Π ,και εισόδους μεγάλου μέγεθος ακόμα μεγαλύτερη. Άρα επιλέγουμε 50 σημεία παρεμβολής ανά διάσταση ως την προκαθορισμένη τιμή για την υλοποίηση του τρισδιάστατου απωθητικού υπολογισμού.

3.3 Υβριδική CPU-GPU υλοποίηση

Όπως θα παρατηρήσουμε στα πειράματα της επόμενης παραγράφου ισχύει ότι ο συνολικός χρόνος της εφαρμογής κυριαρχείται από τον χρόνο υπολογισμού του Ελκτικού όρου. Επίσης ισχύει ότι σε ένα σύστημα που διαθέτει μεγάλο αριθμό πυρήνων ο χρόνος που απαιτεί η μέθοδος SG-tSNE-Π για τον υπολογισμό του ελκτικού όρου είναι συνήθως πιο μικρός από τον χρόνο που απαιτεί η GPU υλοποίησή μας. Αλλά ο χρόνος που απαιτείται για τον υπολογισμό του Απωθητικού όρου από την SG-tSNE-Π είναι πολύ πιο μεγάλος από αυτόν της υλοποίησής μας ανεξαρτήτως του αριθμού πυρήνων.

Λόγο αυτών των παρατηρήσεων έχουμε αναπτύξει επίσης και μία Υβριδική CPU-GPU υλοποίηση. Συγκεκριμένα σε αυτήν την υλοποίηση εκτελούμε τον υπολογισμό

του Ελκτικού όρου στην CPU με τις ρουτίνες της υλοποίησης SG-tSNE-Π και τον υπολογισμό του Απωθητικού όρου την GPU με την βοήθεια ενός CPU νήματος. Αφού αυτοί οι υπολογισμοί είναι εντελώς ανεξάρτητοι και χρησιμοποιούν διαφορετικούς πόρους του συστήματος εκτελούνται παράλληλα.

Αφού ο ελκτικός υπολογισμός της Υβριδικής υλοποίησης είναι πολύ πιο χρονοβόρος από τον απωθητικό, ιδίως όταν απεικονίζουμε μεγάλα σύνολα δεδομένων. Επομένως το χρονικό κόστος του απωθητικού υπολογισμού εξαφανίζεται λόγο της παραλληλίας. Με αυτόν τον τρόπο αξιοποιούμε πλήρως τις δυνατότητες του συστήματός. Επίσης αφού το χρονικό κόστος του απωθητικού όρου κρύβεται χρησιμοποιούμε διπλή ακρίβεια για τον απωθητικό υπολογισμό χωρίς να έχουμε επιπλέον επιβράδυνση. Αξίζει να σημειωθεί ότι αυτή η υλοποίησή είναι προτιμότερη μόνο όταν διαθέτουμε ένα πολύ αποδοτικό πολυπύρηνο σύστημα αλλιώς έχουμε χειρότερη απόδοση από την GPU υλοποίηση.

Κεφάλαιο 4

Πειραματικά Αποτελέσματα

4.1 Πειραματική Διάταξη

Όλα τα πειράματα πραγματοποιήθηκαν σε σύστημα με CPU AMD Ryzen Threadripper1900X (8 πυρήνων και 16 νημάτων με Υπερνημάτωση), 64GB RAM και GPU GeForce GTX 1060 6GB. Για όλες τις υλοποιήσεις χρησιμοποιήθηκαν οι ίδιες παράμετροι t-SNE και βελτιστοποίησης, συγκεκριμένα:

Τιμές των παραμέτρων κατά την εκτέλεση των πειραμάτων						
Παράμετρος	u	k	η	m	a	N
Τιμή	30	90	200	0.5	12	1000

Συμβολίζουμε με u την παράμετρο Perplexity, με k των αριθμό των κοντινότερων γειτόνων, με η το βήμα της gradient descent, με m την ορμή (momentum) της gradient descent, a την παράμετρο πρώιμης υπερβολής και N τον αριθμό των επαναλήψεων. Αυτός ο συμβολισμός χρησιμοποιείται συνήθως από όλες τις υλοποιήσεις t-SNE. Μετά από τις πρώτες 250 επαναλήψεις η παράμετρος πρώιμης υπερβολής και η ορμή θέτεται στο 1 και 0.8 αντίστοιχα. Αυτή η επιλογή των παραμέτρων εμφανίζεται συχνά κατά την εφαρμογή της μεθόδου.

4.2 Αποδοτικότητα

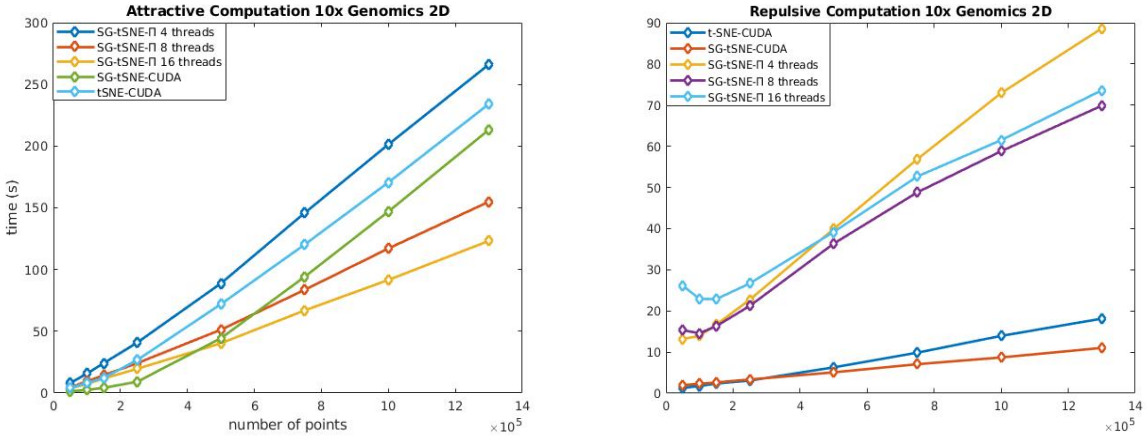
Σε αυτό το κεφάλαιο θα εξετάσουμε τα χαρακτηριστικά επίδοσης των υλοποιήσεων που αναπτύξαμε κατά τη διάρκεια της διπλωματικής σε σχέση με τις πλέον αποδοτικότερες υλοποιήσεις. Ως κύριο χαρακτηριστικό επίδοσης θα λαμβάνουμε την επίδοση για μεγάλα σύνολα δεδομένων καθώς όλες οι υλοποιήσεις που αναφέραμε έχουν καλή απόδοση για μικρά σύνολα δεδομένων. Θα αναφερόμαστε στις υλοποιήσεις μας ως SG-tSNE-CUDA για την GPU υλοποίηση και SG-tSNE-HYB για την υβριδική υλοποίηση για ευκολία διατύπωσης. Ως συνολικό χρόνο θα μετρήσουμε τον χρόνο βελτιστοποίησης χωρίς την προ-επεξεργασία που χρειάζεται για να αρχικοποιήθουν τα δεδομένα. Αυτό γίνεται καθώς με μία εκτέλεση προ-επεξεργασίας μπορούμε να τρέξουμε πολλαπλές βελτιστοποιήσεις για διαφορετικές τιμές των παραμέτρων.

4.2.1 Σύνολο δεδομένων 10x Genomics

Στις ακόλουθες παραγράφους θα παρουσιάσουμε πειραματικά αποτελέσματα για τις απεικονίσεις του συνόλου δεδομένων κυττάρων ποντικιών της 10x Genomics. Συγκεκριμένα θα εκτελέσουμε πειράματα για τυχαία δειγματοληπτημένα υποσύνολα μεγέθους 100k, 250k, 500k, 750k, 1.3m στοιχείων. Με αυτόν τον τρόπο μπορούμε να δούμε πως ο χρόνος των διαφόρων υλοποιήσεων αλλάζει με τον αριθμό των σημείων εισόδου.

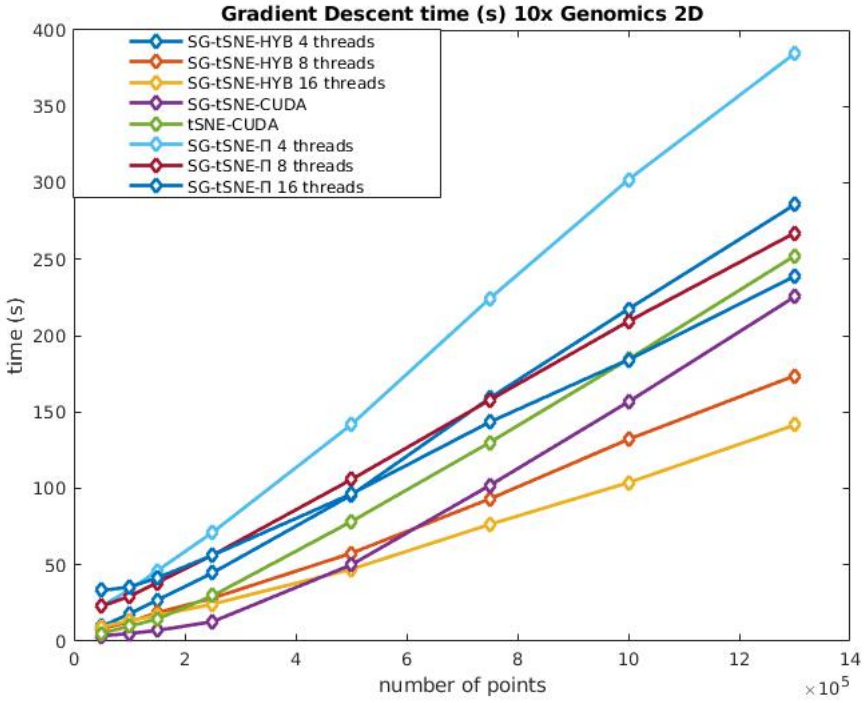
Δισδιάστατες απεικονίσεις

Στην Εικόνα 4.1 βλέπουμε τις χρονικές απαιτήσεις του Ελκτικού και Απωθητικού υπολογισμού για τις διάφορες υλοποιήσεις. Βλέπουμε ότι για αυτά τα σύνολα δεδομένων η υλοποίηση SG-tSNE-Π, με την χρήση 8 και 16 νημάτων, είναι αρκετά γρηγορότερη στον υπολογισμό του ελκτικού όρου από τις υπόλοιπες. Ενώ για την υπολογισμό του απωθητικού όρου οι GPU υλοποιήσεις είναι πολύ πιο γρήγορες.



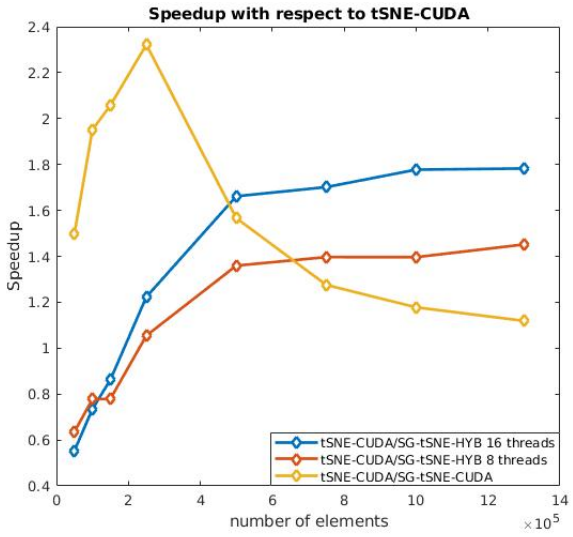
Εικόνα 4.1: Χρόνος 2D Ελκτικού και Απωθητικού υπολογισμού σε υποσύνολα του 10x Genomics

Τα παραπάνω έχουν ως αποτέλεσμα (Εικόνα 4.2) ότι για αυτό το σύνολο δεδομένων η Υβριδική μας υλοποίηση, που χρησιμοποιεί για την μέθοδο της SG-tSNE-Π για τον υπολογισμό του ελκτικού όρου και την GPU για τον υπολογισμό του απωθητικού να είναι αρκετά πιο γρήγορη από τις υπόλοιπες ιδίως για μεγάλο αριθμό νημάτων, 8 και 16. Επίσης, ισχύει ότι για μεγάλο αριθμό σημείων το χρονικό κόστος του απωθητικού υπολογισμού εξαφανίζεται από το συνολικό χρόνο της Υβριδικής υλοποίησης λόγω της παραλληλίας. Αυτό μπορεί να διαπιστωθεί από την ομοιότητα του ελκτικού χρόνου της SG-tSNE-Π και του συνολικού χρόνου της υβριδικής υλοποίησης.



Εικόνα 4.2: Συνολικός Χρόνος 2D Απεικόνισης σε υποσύνολα του 10x Genomics

Παρακάτω βλέπουμε την επιτάχυνση των υλοποιήσεων μας ως προς την t-SNE-CUDA. Μπορούμε να παρατηρήσουμε ότι η SG-tSNE-HYB με τη χρήση 16 νημάτων έχει παραπάνω από 1.6 επιτάχυνση ως προς την t-SNE-CUDA για μεγάλα σύνολα δεδομένων και με την χρήση 8 νημάτων έχει περίπου παραπάνω από 1.4 επιτάχυνση. Ενώ η υλοποίηση SG-tSNE-CUDA παρουσιάζει μεγάλη επιτάχυνση για μικρά και μεσαία σύνολα δεδομένων.

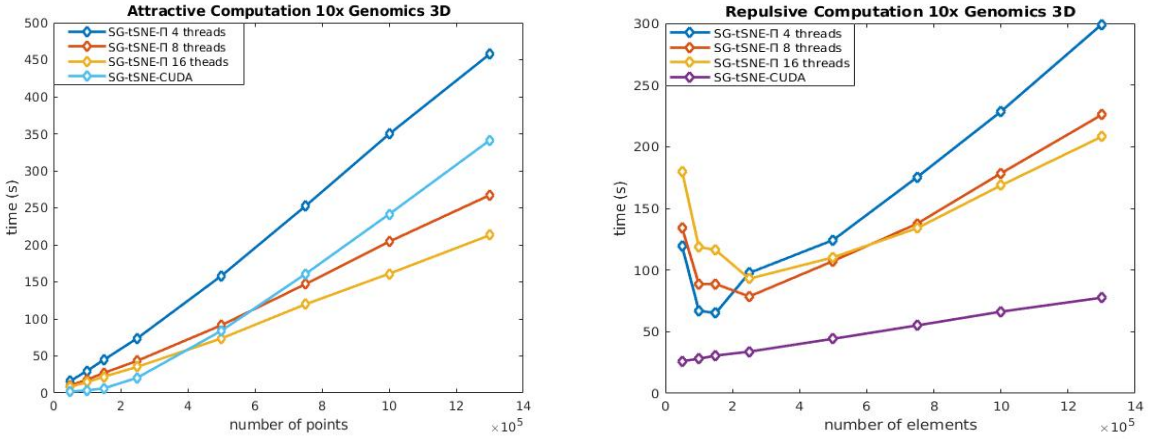


Εικόνα 4.3: Επιτάχυνση των υλοποιήσεων SG-tSNE-CUDA και SG-tSNE-HYB ως προς την t-SNE-CUDA

Τρισδιάστατες απεικονίσεις

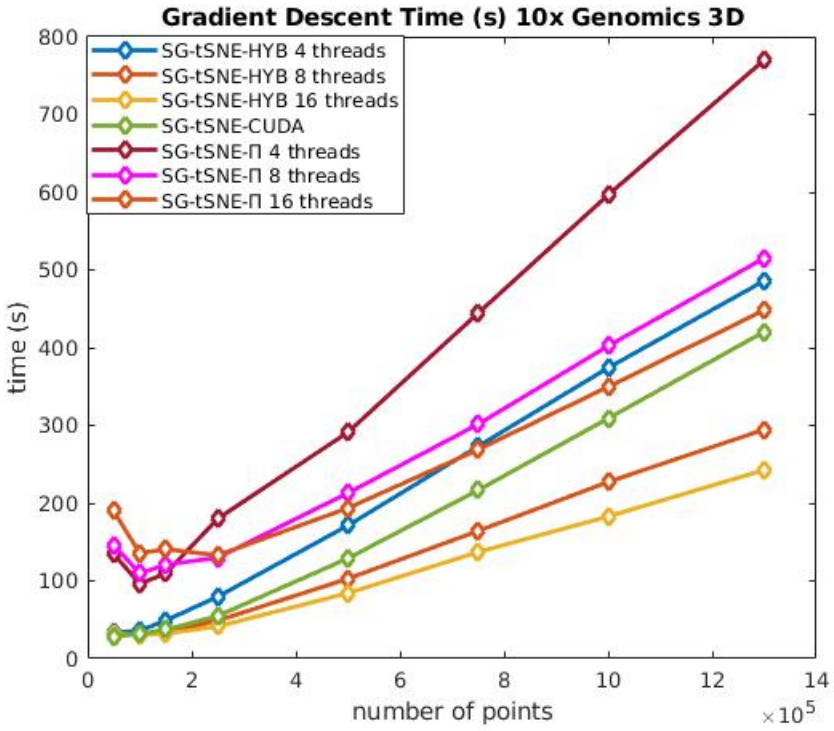
Σε αυτή την παράγραφο θα εκτελέσουμε τα ίδια πειράματα αλλά για απεικόνιση σε τρεις διαστάσεις. Ισχύει ότι η υλοποίηση t-SNE-CUDA δεν υποστηρίζει τρισδιάστατες απεικονίσεις οπότε η σύγκρισή μας θα γίνει ανάμεσα στις υλοποιήσεις SG-tSNE-Π, SG-tSNE-CUDA και SG-tSNE-HYB.

Στην ακόλουθη φιγούρα βλέπουμε τον απαιτούμενο χρόνο του Ελκτικού και Απωθητικού υπολογισμού για τις διάφορες υλοποιήσεις. Μπορούμε να διαπιστώσουμε ότι και σε αυτήν την περίπτωση η υλοποίηση SG-tSNE-Π είναι αρκετά πιο γρήγορη για τον υπολογισμό του Ελκτικού όρου από τις υπόλοιπες αλλά όχι για τον υπολογισμό του απωθητικού.



Εικόνα 4.4: Χρόνος 3D Ελκτικού και Απωθητικού υπολογισμού σε υποσύνολα του 10x Genomics

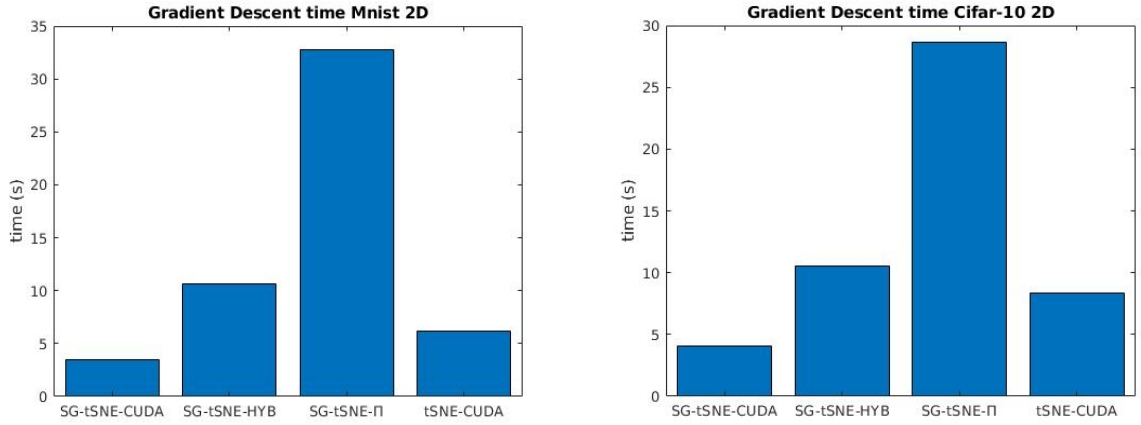
Τα παραπάνω έχουν ως αποτέλεσμα η Υβριδική υλοποίηση να είναι αρκετά πιο γρήγορη από της υπόλοιπες. Επίσης μπορούμε να παρατηρήσουμε ότι σε αυτήν την περίπτωση η επιλογή της υβριδικής υλοποίησης είναι ακόμα πιο ιδανική καθώς ο χρόνος υπολογισμού του τρισδιάστατου απωθητικού όρου, που κρύβεται λόγω της παραλληλίας, είναι μεγαλύτερος.



Εικόνα 4.5: Συνολικός Χρόνος 3D Απεικόνισης σε υποσύνολα του 10x Genomics

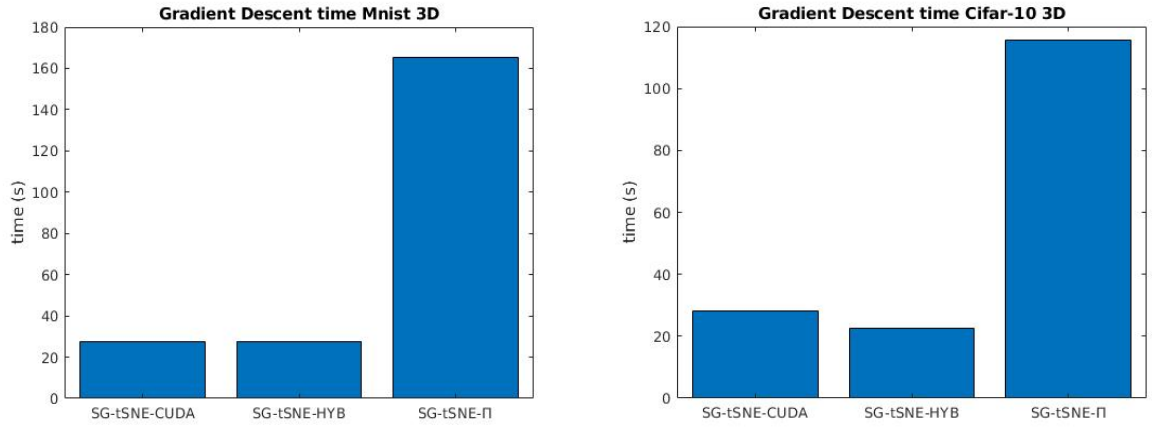
4.2.2 Μικρότερα σύνολα δεδομένων

Στις ακόλουθες εικόνες βλέπουμε τους συνολικούς χρόνους απεικόνισης των συνόλων δεδομένων Mnist και Cifar-10. Αυτά τα σύνολα περιέχουν 60 χιλιάδες στοιχεία το καθένα, που αναπαριστούν εικόνες. Το Mnist περιέχει 28 επί 28 ασπρόμαυρες εικόνες χειρόγραφων ψηφίων και το Cifar-10 32 επί 32 έγχρωμες εικόνες αντικειμένων. Αυτά τα σύνολα χρησιμοποιούνται συχνά για την σύγκριση αποδοτικότητας υλοποιήσεων t-SNE καθώς δίνουν χαρακτηριστικές απεικονίσεις. Παρακάτω βλέπουμε τον χρόνο που απαιτείται για την παραγωγή δισδιάστατης απεικόνισης αυτών των συνόλων. Παρατηρούμε ότι για την απεικόνιση του Mnist η υλοποίηση SG-tSNE-CUDA είναι η ταχύτερη και έχει μάλιστα επιτάχυνση 1.78 ως προς την t-SNE-CUDA. Για την απεικόνιση του Cifar-10 βλέπουμε ότι η SG-tSNE-CUDA είναι πάλι η ταχύτερη με επιτάχυνση 2.06 ως προς την t-SNE-CUDA. Η υβριδική υλοποίηση SG-tSNE-HYB σε αυτήν την περίπτωση δεν είναι η ταχύτερη καθώς έχουμε μικρό αριθμό στοιχείων και έτσι η μεταφορά δεδομένων που χρειάζεται από την CPU στην GPU και αντίστροφα μαζί με τον απωθητικό υπολογισμό δεν καλύπτονται από τον ελκτικό υπολογισμό.



Εικόνα 4.6: Χρόνος δισδιάστατης απεικόνισης Mnist και Cifar-10 από τις διάφορες υλοποιήσεις.

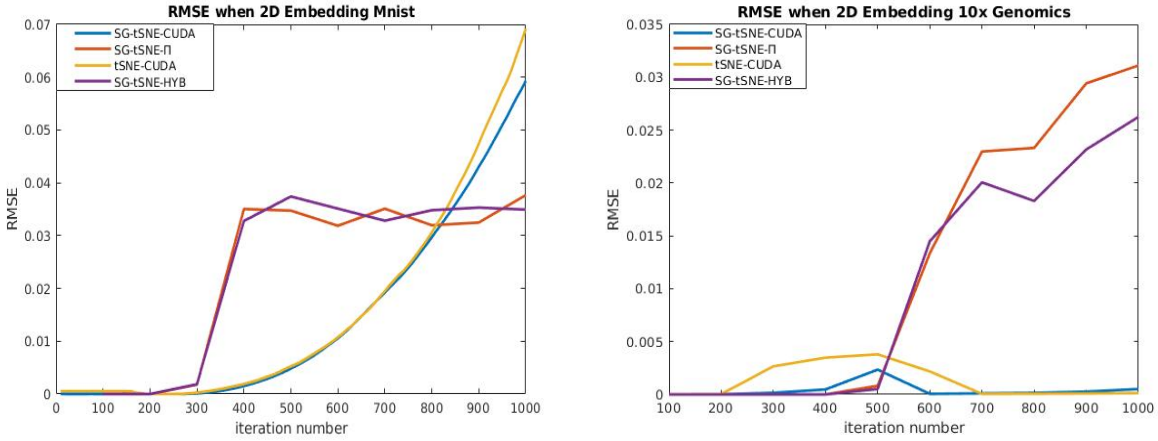
Για τρισδιάστατες απεικονίσεις διαπιστώνουμε ότι η SG-tSNE-CUDA και η SG-tSNE-HYB είναι οι ταχύτερες. Η αλλαγή από την προηγούμενη περίπτωση είναι ότι οι χρόνοι δεν είναι τόσο μικροί και έτσι οι μεταφορές δεδομένων καλύπτονται. Επίσης η SG-tSNE-HYB χρησιμοποιεί μεταβλητό αριθμό σημείων παρεμβολής ενώ η SG-tSNE-CUDA σταθερό. Ως αποτέλεσμα για μεγάλο αριθμό επαναλήψεων η SG-tSNE-CUDA χρησιμοποιεί μεγαλύτερο αριθμό σημείων από την SG-tSNE-HYB αφού ο προκαθορισμένος είναι μεγαλύτερος από τις ανάγκες της υλοποίησης.



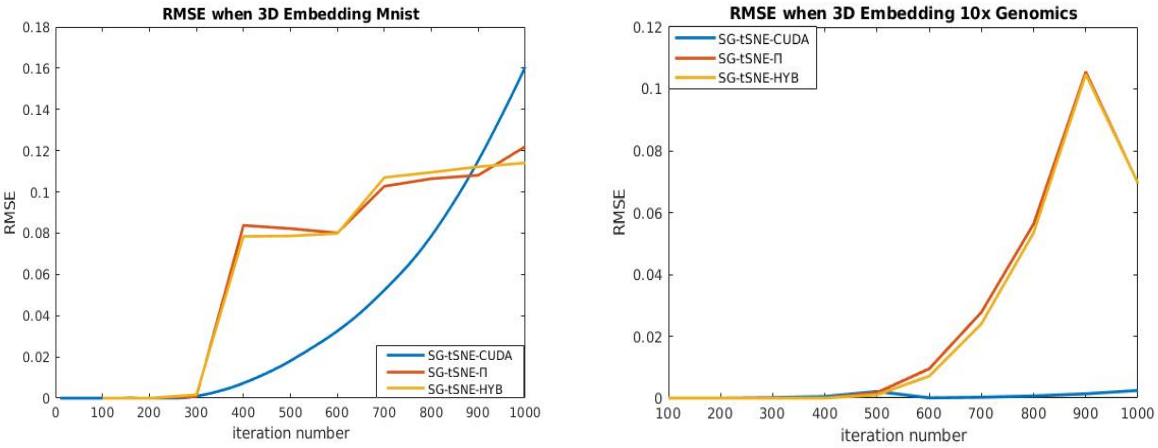
Εικόνα 4.7: Χρόνος τρισδιάστατης απεικόνισης Mnist και Cifar-10 από τις διάφορες υλοποιήσεις.

4.3 Ακρίβεια

Όπως αναφέραμε στην παράγραφο 3.2 η ακρίβεια της προσέγγισης, αριθμός των σημείων παρεμβολής, έχει πολύ μεγάλη σημασία στην τελική απεικόνιση. Στην ακόλουθη εικόνα βλέπουμε το Απωθητικό σφάλμα RMSE ανά επανάληψη, των διαφόρων υλοποιήσεων (με υποδειγματοληψία καθώς ο ακριβείς υπολογισμός του απωθητικού όρου απαιτεί χρόνο $O(n^2)$). Τα πειράματα μας γίνονται σε ένα μικρό σύνολο δεδομένων, Mnist, και ένα μεγάλο, το 10x Genomics.



Εικόνα 4.8: Απωθητικό σφάλμα ανά επανάληψη κατά την 2D απεικόνιση του Mnist και του 10x Genomics



Εικόνα 4.9: Απωθητικό σφάλμα ανά επανάληψη κατά την 3D απεικόνιση του Mnist και 10x Genomics.

Από τις παραπάνω φιγούρες βλέπουμε ότι η υλοποίηση SG-tSNE-HYB έχει την ίδια συμπεριφορά ακριβείας με την SG-tSNE-Π αυτό συμβαίνει διότι οι δύο υλοποιήσεις χρησιμοποιούν τον ίδιο νόμο για την ανανέωση των σημείων ανάλογα με τις απαιτήσεις ακριβείας. Επίσης παρατηρούμε ότι οι υλοποιήσεις SG-tSNE-CUDA και tSNE-CUDA που έχουν σταθερό αριθμό σημείων παρεμβολής έχουν καλύτερη συμπεριφορά ακρίβειας για το μεγάλο σύνολο δεδομένων. Αυτό συμβαίνει διότι η ακρίβεια είναι άμεσα εξαρτημένη με την έκταση της απεικόνισης. Παρατηρείται πειραματικά ότι στις απεικονίσεις t-SNE όταν ο αριθμός σημείων, σε έναν συγκεκριμένο όγκο, μεγαλώνει συνήθως η έκταση της απεικόνισης μικραίνει. Και έτσι οι απεικονίσεις μεγάλου αριθμού σημείων εισόδου είναι μικρές σε έκταση. Οι υλοποιήσεις μεταβλητού αριθμού σημείων παρεμβολής όπως η SG-tSNE-Π αυξάνουν τον αριθμό των σημείων του πλέγματος ανάλογα με την έκταση της απεικόνισης. Όμως οι απεικονίσεις σταθερού αριθμού σημείων παρεμβολής, που χρησιμοποιούν έναν προκαθορισμένο αριθμό σημείων, έχουν μεγάλο μέγεθος πλέγματος για μικρές εκτάσεις απεικονίσεων. Παρατηρείται επίσης ότι η υλοποίηση SG-tSNE-CUDA έχει καλύτερη ακρίβεια από την t-SNE-CUDA. Όπως

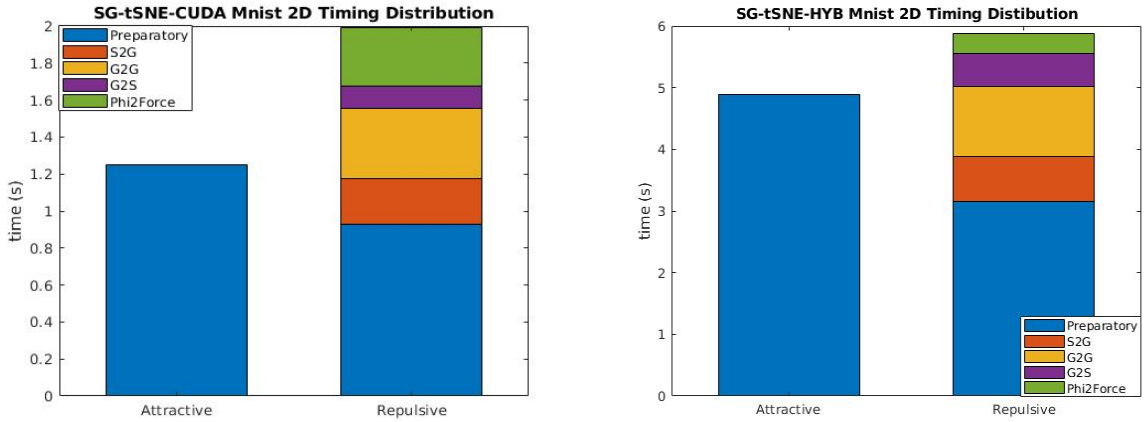
θα δούμε στη συνέχεια αυτές οι τιμές ακρίβειας είναι ικανές για την αντιπροσωπευτική απεικόνιση t-SNE αυτών των συνόλων δεδομένων.

4.3.1 Κατανομή του Χρόνου

Σε αυτή την παράγραφο θα δούμε πως ο συνολικός χρόνος απεικόνισης κατανέμεται στα διάφορα βήματα της μεθόδου. Όπως θα δούμε η κατανομή του χρόνου αλλάζει με τον αριθμό σημείων εισόδου για αυτό θα εξετάσουμε δύο παραδείγματα ένα του συνόλου Mnist ,60 χιλιάδων στοιχείων, και ένα του συνόλου 10x Genomics ,1.3 εκατομμυρίων σημείων. Ο χρόνος του απωθητικού υπολογισμού διαμερίζεται σε 5 διεργασίες. Αρχικά τις προ-διεργασίες (Preparatory processes), αυτές στην SG-tSNE-CUDA θα αποτελούν την διάφορές διεργασίες για την δημιουργία του πλέγματος ενώ στην SG-tSNE-HYB θα περιέχουν επίσης τις αναθέσεις μνήμης, την δημιουργία του σχεδίου cuFFT και τις μεταφορές δεδομένων από CPU σε GPU και αντίστροφα. Οι υπόλοιπες διαμερίσεις του απωθητικού χρόνου αποτελούν τους υπολογισμούς που περιγράφονται στην 3.1.2 (S2G, G2G, G2S και Phi2Force). Στις παρακάτω περιπτώσεις για την υλοποίηση SG-tSNE-HYB απεικονίζουμε τον χρόνο που απαιτείται από τον ελκτικό και τον απωθητικό υπολογισμό άλλα τονίζουμε ότι το άθροισμα τους δεν αποτελεί τον συνολικό χρόνο καθώς αφού εκτελούνται παράλληλα και υπάρχει επικάλυψη. Επίσης χρησιμοποιούνται 16 νήματα κατά την εκτέλεση του SG-tSNE-HYB στην παραγωγή των παρακάτω αποτελεσμάτων.

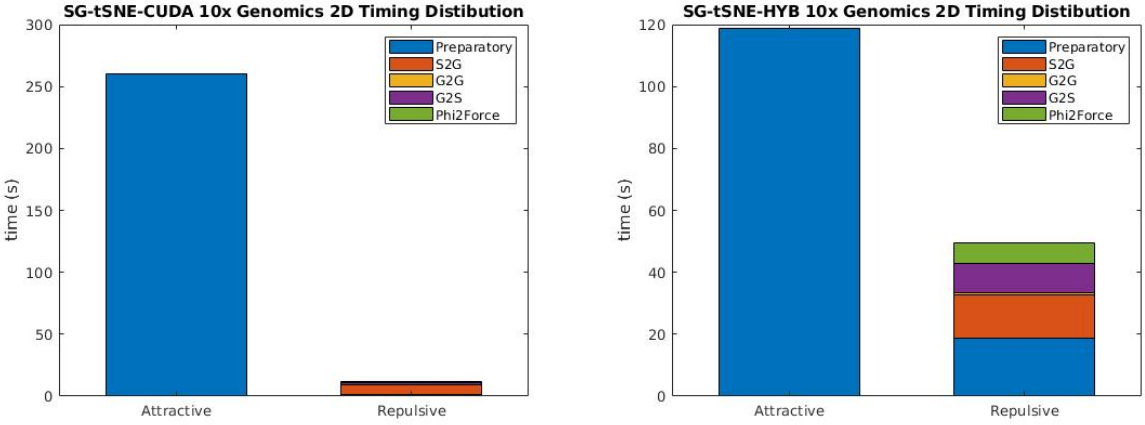
4.4 Δισδιάστατες Απεικονίσεις

Αρχικά βλέπουμε την κατανομή χρόνου της SG-tSNE-CUDA και SG-tSNE-HYB όταν παράγουμε δισδιάστατη απεικόνιση του Mnist. Σε αυτήν την περίπτωση όπως είδαμε και παραπάνω η χρήση της SG-tSNE-CUDA είναι προτιμότερη. Αυτό ισχύει διότι αρχικά ο αραιός πίνακας που παράγεται από το Mnist είναι πιο κατάλληλος για την χρήση της HYB δομής. Και επίσης το απωθητικό κόστος της SG-tSNE-HYB δεν μπορεί ακόμα να καλυφθεί από το ελκτικό λόγο του μικρού αριθμού σημείων εισόδου. Ακόμα μπορούμε να παρατηρήσουμε ότι οι προ-διεργασίες της SG-tSNE-HYB δαπανούν πολύ μεγάλο χρόνο καθώς οι μεταφορές δεδομένων CPU-GPU είναι χρονοβόρες.



Εικόνα 4.10: Κατανομή χρόνου των υλοποιήσεων κατά την 2D απεικόνιση του Mnist.

Παρακάτω βλέπουμε τα ίδια πειράματα για την δισδιάστατη απεικόνιση του συνόλου 10x Genomics. Ισχύει ότι σε αυτό το παράδειγμα ο υπολογισμός του ελκτικού όρου με την CPU και την CSB δομή είναι πολύ πιο αποδοτικός. Ακόμα βλέπουμε ο χρόνος των υπολογισμών S2G και G2S είναι πιο εμφανείς καθώς επηρεάζονται από τον αριθμό των σημείων εισόδου ενώ ο G2G όχι. Επίσης σε αυτήν την περίπτωση έχουμε πλήρη κάλυψη του απωθητικού χρόνου από τον ελκτικό. Έτσι ο συνολικός χρόνος της SG-tSNE-HYB είναι αρκετά μικρότερος και περίπου ίσος με τον ελκτικό.

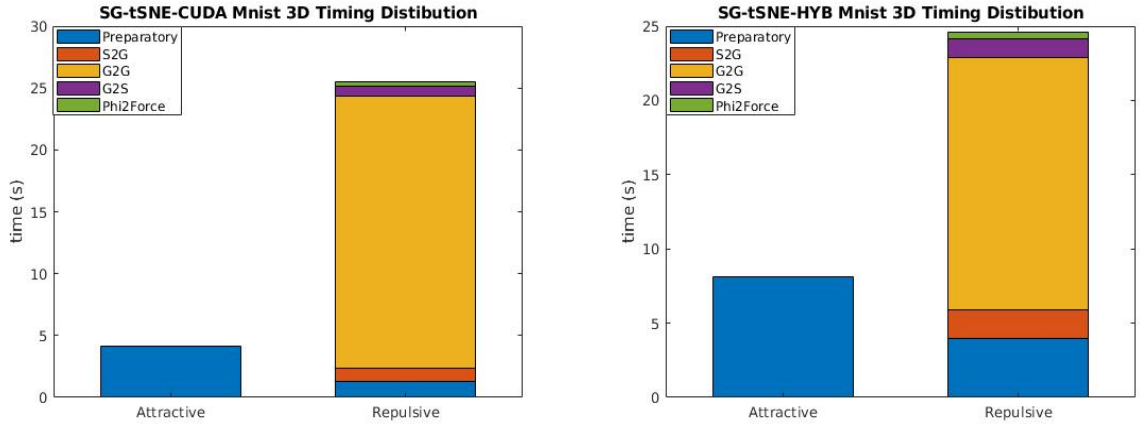


Εικόνα 4.11: Κατανομή χρόνου των υλοποιήσεων κατά την 2D απεικόνιση του 10x Genomics.

4.5 Τρισδιάστατες Απεικονίσεις

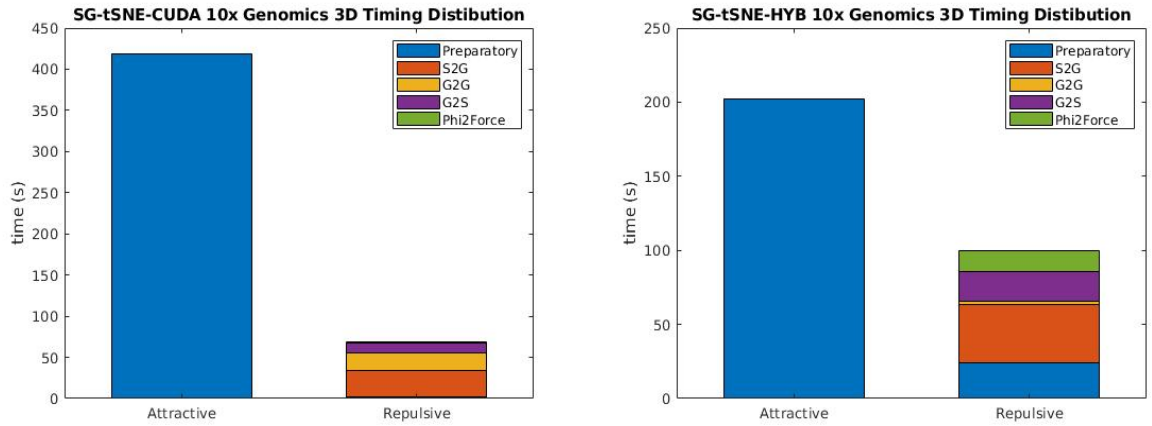
Σε αυτήν την παράγραφο θα παρουσιάσουμε τα ίδια πειράματα για απεικόνιση σε 3 διαστάσεις. Αρχικά για την απεικόνιση του Mnist βλέπουμε ότι ο χρόνος που χρειάζεται ο μετασχηματισμός G2G είναι πολύ πιο μεγάλος αφού χρησιμοποιούνται περισσότερα σημεία παρεμβολής. Ακόμα για την SG-tSNE-HYB βλέπουμε ότι το κόστος του ελκτικού χρόνου κρύβεται διότι εκτελείται παράλληλα με τον απωθητικό υπολογισμό (το άθροισμά τους είναι 35.6sec ενώ ο συνολικός χρόνος είναι 27sec). Τα

παραπάνω έχουν ως αποτέλεσμα η υλοποίησης SG-tSNE-HYB να είναι η ταχύτερη για αυτό το παράδειγμα παρότι ο ελκτικός της χρόνος είναι μεγαλύτερος από τον αντίστοιχο της SG-tSNE-CUDA (Εικόνα 4.7).



Εικόνα 4.12: Κατανομή χρόνου των υλοποιήσεων κατά την 3D απεικόνιση του Mnist.

Τέλος εξετάζουμε την ίδια περίπτωση για το 10x Genomics. Εδώ παρατηρούμε ότι αφού ο αριθμός των σημείων είναι μεγαλύτερος, η επιφάνεια απεικόνισης είναι μικρότερη και ως αποτέλεσμα ο χρόνος των μετασχηματισμού S2G και G2G κυριαρχεί τον απωθητικό χρόνο. Επίσης ομοίως με την δισδιάστατη περίπτωση βλέπουμε ότι λόγω της παραλληλίας στην SG-tSNE-HYB το κόστος του απωθητικού υπολογισμού κρύβεται από ελκτικό υπολογισμό. Ως αποτέλεσμα η SG-tSNE-HYB είναι αρκετά πιο αποδοτική από την SG-tSNE-CUDA.

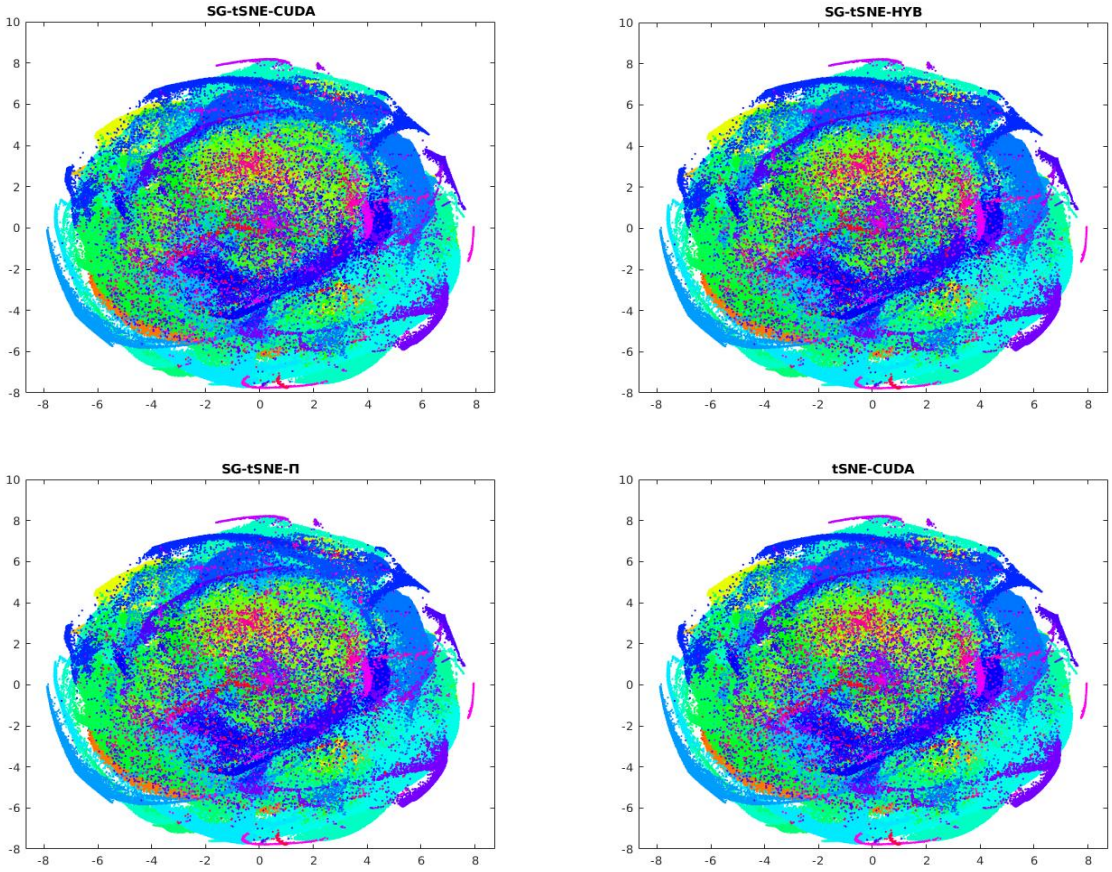


Εικόνα 4.13: Κατανομή χρόνου των υλοποιήσεων κατά την 2D απεικόνιση του 10x Genomics.

4.5.1 Παραδείγματα Απεικονίσεων

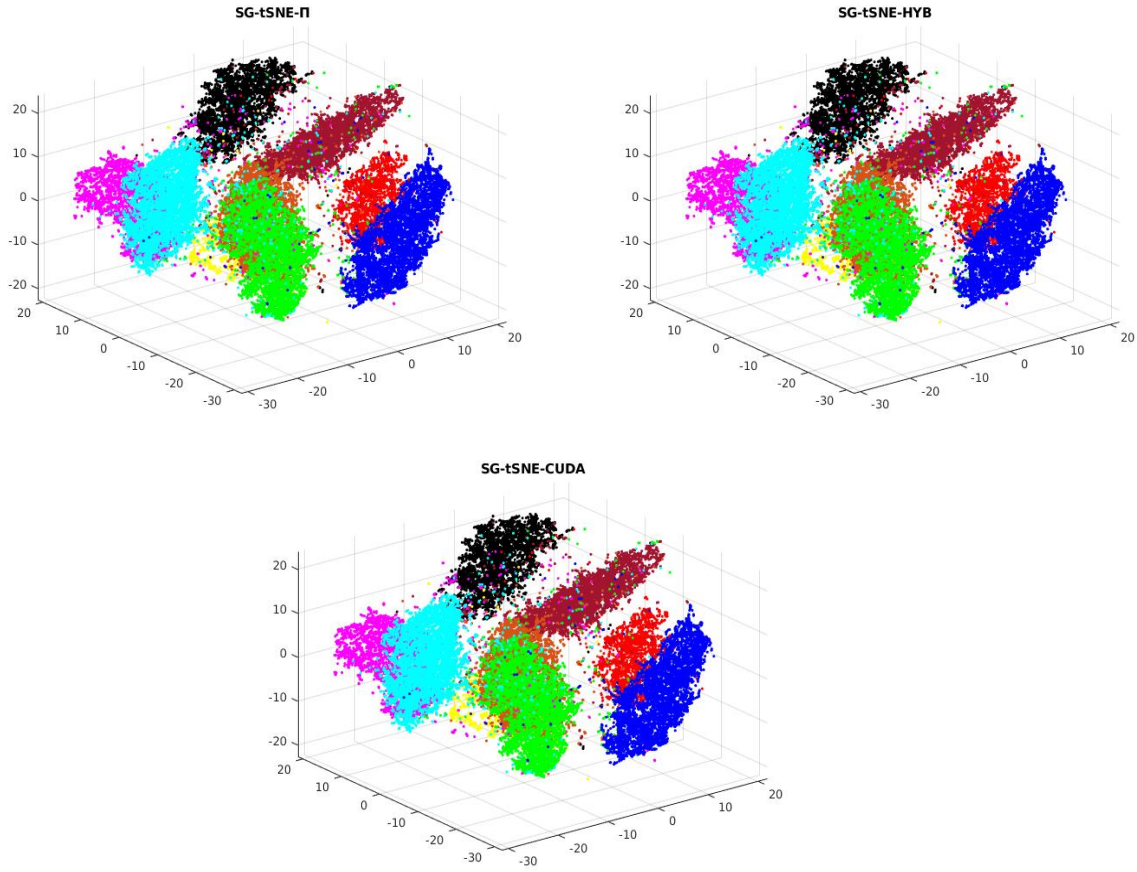
Τελειώνοντας την ανάλυση των πειραματικών αποτελεσμάτων βλέπουμε τις απεικονίσεις εξόδου για το πλήρη 1.3m σύνολο δεδομένων 10x Genomics από κάθε υλοποίηση,

χρησιμοποιώντας το ίδιο σημείο αρχικοποίησης για την gradient descent και τους ίδιους ελάχιστους γείτονες. Μπορούμε να πούμε ότι δεν υπάρχει οπτική διαφορά ανάμεσα στις απεικονίσεις.



Εικόνα 4.14: Δισδιάστατη απεικόνιση του συνόλου δεδομένων 10x Genomics

Παρακάτω βλέπουμε την τρισδιάστατη απεικόνιση του συνόλου Mnist από τις υλοποιήσεις SG-tSNE-CUDA, SG-tSNE-HYB και SG-tSNE-Π.



Εικόνα 4.15: Τρισδιάστατη απεικόνιση του συνόλου δεδομένων Mnist

Αφού βλέπουμε ότι δεν υπάρχει διαφορά στις παραπάνω απεικονίσεις όλων των υλοποιήσεων (το ίδιο διαπιστώσουμε και για άλλα παραδείγματα) μπορούμε να διαπιστώσουμε ότι οι τιμές ακρίβειας των υλοποιήσεων είναι ικανές για έμπιστες απεικονίσεις.

4.6 Συμπεράσματα

Από τα πειραματικά αποτελέσματα συμπεραίνουμε ότι οι υλοποιήσεις που αναπτύξαμε κατά την διάρκεια της διπλωματικής επιταχύνουν τις υπάρχοντες. Η χρήση της υβριδικής υλοποίησης συνίσταται όταν έχουμε στην διάθεση μας ένα πολύ αποδοτικό σύστημα και θέλουμε να απεικονίσουμε μεγάλα σύνολα δεδομένων. Αλλά για μέτριο ή μικρό αριθμό σημείων και νημάτων μπορεί να μην είναι η αποδοτικότερη, σε αυτές τις περιπτώσεις συνιστούμε την χρήση της GPU υλοποίησης μας SG-tSNE-CUDA. Ένα σημαντικό πλεονέκτημα της υβριδικής υλοποίησης είναι ότι αφού το χρονικό κόστος του απωθητικού υπολογισμού κρύβεται μπορούμε να χρησιμοποιήσουμε περισσότερα σημεία παρεμβολής. Όσον αφορά μελλοντικές βελτιώσεις, αφού το χρονικό κόστος του απωθητικού υπολογισμού μπορεί να κρυφτεί με την χρήση μίας Υβριδικής CPU-GPU υλοποίησης, για βελτίωση του χρόνου πρέπει να επιλεγεί ένας καταλληλότερος αλγόριθμος/δομή για την εκτέλεση του ελκτικού υπολογισμού. Με την ανάλυσή μας

ελέγξαμε κάποιες από της πιο αποδοτικές υλοποιήσεις ελεύθερου κώδικα γινομένου πίνακα-διανύσματος.

Appendix A

Περιγραφή Κώδικα

A.1 Περιγραφή Πυρήνων Απωθητικού Όρου

Παρακάτω περιγράφονται οι βασικότερες πυρήνες (kernels) CUDA που χρησιμοποιήθηκαν για την επιτάχυνση της υλοποίησης. Δίνονται μόνο οι πυρήνες που υλοποιούν την μονοδιάστατη απεικόνιση. Η υλοποίηση των υπόλοιπων πυρίνων είναι προφανής.

A.1.1 S2G

Για αυτό το βήμα εκτελείται παράλληλος υπολογισμός για κάθε ενσωματωμένο σημείο. Και χρησιμοποιείται χρήση ατομικής άθροισης για την αποθήκευση της εξόδου. Επίσης χρησιμοποιείται η μέθοδος διαμέρισης φορτίου Grid-Striding για μία εύρωστη υλοποίηση στην αλλαγή του αριθμού των σημείων. Ακόμα σε αυτόν τον πυρήνα έχουμε ευθυγραμμισμένη προσπέλαση των σημείων κάνει τον πυρήνα πολύ αποδοτικό. Τέλος οι ατομικές αθροίσεις στη CUDA έχουν μεγάλη απόδοσή και αυτό δικαιολογεί τη συχνή χρήση τους στις εφαρμογές CUDA.

```
1  //s2g1d<<<Blocks, Threads>>>(VGrid, y, VScat, nGridDim, n, d, m);
       Kernel Launch
2  template <class dataPoint>
3  __global__ void s2g1d(dataPoint *__restrict__ V, const dataPoint *
       const y, const dataPoint *const q, const uint32_t ng, const
       uint32_t nPts, const uint32_t nDim,  const uint32_t nVec) {
4    dataPoint v1[4];
5    register uint32_t f1;
6    register dataPoint d;
7  //Grid Striding Execution
8    for (register int TID = threadIdx.x + blockIdx.x * blockDim.x;
       TID < nPts;
9          TID += gridDim.x * blockDim.x) {
10     f1 = (uint32_t)floor(y[TID]);
11     d = y[TID] - (dataPoint)f1;
12     v1[0] = g2(1 + d);
13     v1[1] = g1(d);
14     v1[2] = g1(1 - d);
15     v1[3] = g2(2 - d);
16     for (int j = 0; j < nVec; j++) {
17       dataPoint qv = q[nPts * j + TID];
18       for (int idx1 = 0; idx1 < 4; idx1++) {
19         atomicAdd(&V[f1 + idx1 + j * ng], qv * v1[idx1]);
20       }}}}
```

A.1.2 G2S

Για αυτό το βήμα γίνεται παράλληλος υπολογισμός για κάθε ενσωματωμένο σημείο. Σε αυτόν τον πυρήνα έχουμε επίσης ευθυγραμμισμένη προσπέλαση των σημείων κάνει τον πυρήνα πολύ αποδοτικό.

```
//g2s2d<<<Blocks, Threads>>>(PhiScat, PhiGrid, y, nGridDim, n, d, m) Kernel Launch
template <class dataPoint>
__global__ void g2s1d(volatile dataPoint *__restrict__ Phi,
                      const dataPoint *const V, const dataPoint *const y,
                      const uint32_t ng, const uint32_t nPts,
                      const uint32_t nDim, const uint32_t nVec) {
  dataPoint v1[4];
  uint32_t f1;
  dataPoint d;
//Grid Striding Execution
  for (register int TID = threadIdx.x + blockIdx.x * blockDim.x;
   TID < nPts;
       TID += gridDim.x * blockDim.x) {
    f1 = (uint32_t)floor(y[TID]);
    d = y[TID] - (dataPoint)f1;
    v1[0] = g2(1 + d);
    v1[1] = g1(d);
    v1[2] = g1(1 - d);
    v1[3] = g2(2 - d);
    for (uint32_t j = 0; j < nVec; j++) {
      dataPoint accum = 0;
      for (uint32_t idx1 = 0; idx1 < 4; idx1++) {
        accum += V[f1 + idx1 + j * ng] * v1[idx1];
      }
      Phi[TID + j * nPts] = accum;
    }}}
```

Βιβλιογραφία

[1] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[2] S. Arora, W. Hu, and P. K. Kothari, "An Analysis of the t-SNE Algorithm for Data Visualization," *CoRR*, vol. abs/1803.01768, 2018. arXiv: 1803.01768. [Online]. Available: http://arxiv.org/abs/1803.01768.

[3] G. C. Linderman and S. Steinerberger, "Clustering with t-SNE, provably," *CoRR*, vol. abs/1706.02582, 2017. arXiv: 1706.02582. [Online]. Available: http://arxiv.org/abs/1706.02582.

[4] N. Pitsianis, A.-S. Iliopoulos, D. Floros, and X. Sun, "Spaceland Embedding of Sparse Stochastic Graphs," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–8. DOI: 10.1109/HPEC.2019.8916505.

[5] P. N. Yianilos, "Excluded Middle Vantage Point Forests for Nearest Neighbor Search," in *In DIMACS Implementation Challenge, ALENEX'99*, 1999.

[6] G. C. Linderman, M. Rachh, J. G. Hoskins, S. Steinerberger, and Y. Kluger, "Efficient Algorithms for t-distributed Stochastic Neighborhood Embedding," *CoRR*, vol. abs/1712.09005, 2017. arXiv: 1712.09005. [Online]. Available: http://arxiv.org/abs/1712.09005.

[7] D. Lee, "Fast multiplication of a recursive block toeplitz matrix by a vector and its application," *J. Complex.*, vol. 2, no. 4, 295–305, Dec. 1986, ISSN: 0885-064X. DOI: 10.1016/0885-064X(86)90007-5. [Online]. Available: https://doi.org/10.1016/0885-064X(86)90007-5.

[8] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, "Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication Using Compressed Sparse Blocks," in *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '09, Calgary, AB, Canada: Association for Computing Machinery, 2009, 233–244, ISBN: 9781605586069. DOI: 10.1145/1583991.1584053. [Online]. Available: https://doi.org/10.1145/1583991.1584053.

[9] G. Karypis and V. Kumar, "Analysis of Multilevel Graph Partitioning," in *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '95, San Diego, California, USA: Association for Computing Machinery, 1995, 29–es, ISBN: 0897918169. DOI: 10.1145/224170.224229. [Online]. Available: https://doi.org/10.1145/224170.224229.

[10] J. Mellor-Crummey, D. Whalley, and K. Kennedy, "Improving Memory Hierarchy Performance for Irregular Applications Using Data and Computation Reorderings," *Int. J. Parallel Program.*, vol. 29, no. 3, 217–247, Jun. 2001, ISSN: 0885-7458. DOI: 10.1023/A:1011119519789. [Online]. Available: https://doi.org/10.1023/A:1011119519789.

[11] N. Pitsianis, D. Floros, and X. Sun, "Literature Graph of Scholarly Articles Relevant to COVID-19 Study," 2020. [Online]. Available: https://lg-covid-19-hotp.cs.duke.edu/.

[12] D. M. Chan, R. Rao, F. Huang, and J. F. Canny, *t-SNE-CUDA: GPU-Accelerated t-SNE and its Applications to Modern Data*, 2018. arXiv: 1807.11824 [cs.LG].

[13] Nvidia, "cuFFT, CUDA Fast Fourier Transform library, Documentation," 2021. [Online]. Available: https://docs.nvidia.com/cuda/cufft/index.html.

[14] N. Bell and M. Garland, "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09, Portland, Oregon: Association for Computing Machinery, 2009, ISBN: 9781605587448. DOI: 10.1145/1654059.1654078. [Online]. Available: https://doi.org/10.1145/1654059.1654078.

[15] D. R. Kincaid, T. C. Oppe, J. R. Respes, and D. M. Young, "ITPACK Solution Modules," in *Solving Elliptic Problems Using ELLPACK*. New York, NY: Springer New York, 1985, pp. 237–258, ISBN: 978-1-4612-5018-0. DOI: 10.1007/978-1-4612-5018-0_7. [Online]. Available: https://doi.org/10.1007/978-1-4612-5018-0_7.

[16] S. Dalton, N. Bell, L. Olson, and M. Garland, *Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations*, Version 0.5.0, 2014. [Online]. Available: http://cusplibrary.github.io/.

[17] J. Hoberock and N. Bell, *Thrust: a C++ template library for CUDA primatives*. [Online]. Available: https://github.com/NVIDIA/thrust.

[18] N. Bell and J. Hoberock, "Chapter 26 - Thrust: A Productivity-Oriented Library for CUDA," in *GPU Computing Gems Jade Edition*, ser. Applications of GPU Computing Series, W. mei W. Hwu, Ed., Boston: Morgan Kaufmann, 2012, pp. 359–371, ISBN: 978-0-12-385963-1. DOI: https://doi.org/10.1016/B978-0-12-385963-1.00026-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123859631000265.