Πολυτεχνειο Κρητης

Τμημα Ηλεκτρονικων Μηχ.& Μηχ. Η/Υ

Επιτάχυνση του αλγορίθμου αναγνώρισης φωνής Sphinx 3 με χρήση πολυεπεξεργαστικής κάρτας γραφικών

Συγγραφέας:
Δημήτρης Τσιαμασιώτης

Ιανουάριος 2012

Περίληψη

Το τελευταίο καιρό, ο παράλληλος προγραμματισμός με χρήση πολυεπεξεργαστικών καρτών γραφικών βρίσκει όλο και περισσότερες εφαρμογές από την επιστημονική κοινότητα. Σε αυτό συνετέλεσε η παρουσίαση από την εταιρεία κατασκευής καρτών γραφικών Nvidia, μιας ολοκληρωμένης αρχιτεκτονικής παράλληλου προγραμματισμού για ανάπτυξη εφαρμογών γενικού σκοπού με χρήση των προϊόντων της. Η αρχιτεκτονική αυτή ονομάζεται CUDA. Στην παρούσα εργασία, μεταφέραμε στο μοντέλο του CUDA τον αλγόριθμο αναγνώρισης φωνής Sphinx 3, ο οποίος και αναπτύσσεται από το Carnegie Mellon University. Στόχος μας ήταν η επιτάχυνση του αλγορίθμου εκμεταλλευόμενοι την υψηλή υπολογιστική ισχύ μιας σύγχρονης κάρτας γραφικών. Η σύγκριση των αποτελεσμάτων από τις δικές μας υλοποιήσεις και την πρωταρχική υλοποίηση στη CPU, έδειξε ότι ο χρόνος εκτέλεσης του αλγορίθμου μπορεί να βελτιωθεί ακόμα και όταν παραλληλοποιείται ένα ποσοστό του κώδικα που δεν είναι συντριπτικό.

Περιεχόμενα

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένειά μου για την υποστήριξη και την υπομονή της κατά τη διάρκεια εκπόνησης αυτής της εργασίας καθώς και τον επιβλέποντα καθηγητή κ. Ιωάννη Παπαευσταθίου για την καθοδήγησή του.

Κεφάλαιο 1

Εισαγωγή

Η αναγνώριση φωνής είναι ένα επιστημονικό πεδίο που γίνεται όλο και πιο προσιτό στο ευρύ κοινό μέσω μιας πληθώρας εφαρμογών που το υποστηρίζουν. Ένας από τους πιο γνωστούς αλγόριθμους αναγνώρισης φωνής είναι ο Sphinx 3 που αναπτύσσεται από τους ερευνητές του Carnegie Mellon University. Η ανάπτυξή του είναι το αποτέλεσμα μακρόχρονης προσπάθειας. Σε αυτή τη διπλωματική προσπαθούμε να βελτιώσουμε ένα από τα αρνητικά χαρακτηριστικά του που είναι ο χρόνος εκτέλεσης. Αν και είναι ένας αλγόριθμος πραγματικού χρόνου, η επιτάχυνση του είναι δυνατή με τη χρήση παράλληλου προγραμματισμού με χρήση πολυεπεξεργαστικής κάρτας γραφικών. Ο συγκεκριμένος τρόπος προγραμματισμού απασχολεί αυτό τον καιρό ένα μεγάλο ποσοστό της επιστημονικής κοινότητας αλλά και απλούς προγραμματιστές, επειδή υπάρχει ολοκληρωμένη αρχιτεκτονική που το υποστηρίζει και βασίζεται σε μία από τις πιο διαδεδομένες γλώσσες προγραμματισμού, τη C. Η αρχιτεκτονική αυτή είναι το CUDA και παρέχεται από την εταιρεία Nvidia.

Στο δεύτερο κεφάλαιο, παρουσιάζουμε την έννοια της φωνής και πως αυτή δομείται από μικρότερες ηχητικές μονάδες.

Στο τρίτο κεφάλαιο, κάνουμε μία σύντομη εισαγωγή στο πρόβλημα της αναγνώρισης φωνής. Στη συνέχεια παρουσιάζουμε με ποιο τρόπο εισάγονται σε αυτή στατιστικά μοντέλα και συγκεκριμένα τα κρυμμένα μοντέλα Markov. Επίσης δίνουμε το θεωρητικό υπόβαθρο αυτών μέσα από την παρουσίαση των χαρακτηριστικών τους.

Στο κεφάλαιο τέσσερα, παρουσιάζεται λεπτομερώς ο αλγόριθμος Sphinx 3. Η έμφαση δίνεται στη λειτουργία του αποκωδικοποιητή καθώς η εκπαίδευση των ακουστικών μοντέλων αναφέρεται περιληπτικά για να συνδέσει καλύτερα ο αναγνώστης τα μέρη του αλγορίθμου. Ιδιαίτερη ανάλυση γίνεται και για το

κλάδεμα, ένα σημαντικό στοιχείο του κώδικα που έχει εισαχθεί για να βελτιώσει τις χρονικές επιδόσεις του αλγορίθμου.

Στο πέμπτο κεφάλαιο, εξηγούμε την έννοια του παράλληλου υπολογισμού. Επίσης περιγράφεται η αρχιτεκτονική του CUDA, την οποία και χρησιμοποιήσαμε στη συνέχεια για την επιτάχυνση του αποκωδικοποιητή.

Το κεφάλαιο 6 περιέχει το πρακτικό κομμάτι της παρούσας εργασίας. Στην αρχή γίνεται η χρονική ανάλυση του αλγορίθμου για να βρεθούν τα κομμάτια του κώδικα εκείνα που είναι χρονοβόρα και επομένως υποψήφια για να περαστούν στο CUDA. Εξηγείται επίσης γιατί δεν μπορεί να παραλληλοποιηθεί το σύνολο του αλγορίθμου και παρουσιάζουμε το κομμάτι το οποίο μπορεί να επιταχυνθεί με παράλληλο προγραμματισμό. Συνολικά για το κομμάτι αυτό έγιναν τέσσερις διαφορετικές προσεγγίσεις με διάφορες βελτιστοποιήσεις στην κάθε μία. Για κάθε υλοποίηση παρουσιάζονται οι χρόνοι εκτέλεσης και συγκρίνονται με τους αντίστοιχους από τη CPU αναφοράς. Επιπρόσθετα, έγινε παραλληλοποίηση μιας μικρότερης συνάρτησης, της mgau_precomp και μία υλοποίηση χωρίς λογαριθμικό πίνακα αναζήτησης, η χρήση του οποίου είναι παράμετρος του αποκωδικοποιητή. Τα συνολικά αποτελέσματα παρουσιάζονται στην τελευταία ενότητα του κεφαλαίου.

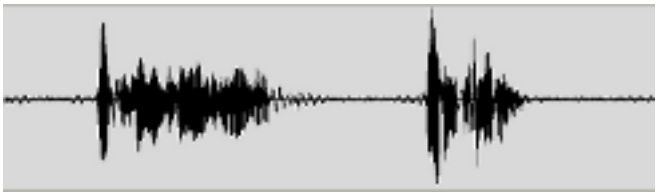
Τέλος, στο κεφάλαιο 7 παρουσιάζονται τα συμπεράσματα αυτής της εργασίας και δίνονται μερικές ιδέες σαν αντικείμενο μελλοντικής εργασίας.

Κεφάλαιο 2

Φωνή

2.1 Η έννοια της φωνής

Η φωνή είναι μία δυναμική διαδικασία και ένα, με μαθηματικούς όρους, μη στάσιμο σήμα. Όταν μιλάμε, το σύστημα άρθρωσης (χείλη, σαγόνια, γλώσσα) διαφοροποιεί την πίεση και τη ροή του αέρα ώστε να παραχθεί μία ακολουθία από ήχους που μπορεί να ακουστεί από το ανθρώπινο αυτί. Αν και ένα φωνητικό σήμα μπορεί να περιέχει πολύ υψηλές και πολύ χαμηλές συχνότητες, συνήθως δεν παρατηρούνται δραματικές αλλαγές περισσότερο από δέκα φορές το δευτερόλεπτο και η ζώνη συχνοτήτων που χρησιμοποιείται για να μεταδώσει τη φωνή κυμαίνεται περίπου από τα 300Hz έως τα 3000Hz[1]. Η ανθρώπινη φωνή ωστόσο μπορεί να φτάσει ακόμα πιο υψηλές συχνότητες αλλά υπό ειδικές συνθήκες, όπως το κλάμα και το ουρλιαχτό. Ένα δείγμα ηχογραφημένης φωνής σε πρόγραμμα επεξεργασίας ήχου φαίνεται παρακάτω.



Σχήμα 2.1: Ένα φωνητικό δείγμα ηχογραφημένο σε πρόγραμμα επεξεργασίας ήχου

2.2 Η δομή της φωνής

Σε ένα οποιοδήποτε φωνητικό δείγμα, μπορεί κανείς να παρατηρήσει ήχους που λίγο έως πολύ μοιάζουν μεταξύ τους. Τους ήχους αυτούς η γλωσσολογία τους ονομάζει φθόγγους και τους ορίζει ως την ελάχιστη ηχητική μονάδα στην οποία μπορεί να αναλυθεί μία λέξη. Αν και αντιλαμβανόμαστε τις λέξεις σαν ένα σύνολο φθόγγων, είναι λάθος να θεωρήσουμε ότι ένας συγκεκριμένος φθόγγος έχει πάντα τις ίδιες ακουστικές ιδιότητες και επομένως και μία συγκεκριμένη λέξη έχει πάντα τις ίδιες ακουστικές ιδιότητες. Οι ακουστικές ιδιότητες μιας κυματομορφής που αντιστοιχεί σε ένα φθόγγο ποικίλουν και εξαρτώνται από πολλούς παράγοντες όπως τα συμφραζόμενα, ο ομιλητής, το ύφος της φωνής, το μέσο μεταφοράς(π.χ. τηλέφωνο) κ.ά.[2]. Σημαντικό ρόλο σε αυτή τη διαφοροποίηση διαδραματίζει και το φαινόμενο της συνάρθρωσης. Συνάρθρωση καλείται το γενικευμένο φαινόμενο ένας ήχος ομιλίας να μην ανήκει αποκλειστικά σε ένα φθόγγο αλλά να προκύπτει από "συμψηφισμό" ή μετάβαση μεταξύ δύο φθόγγων. Η μετάβαση μεταξύ δύο φθόγγων μπορεί να δώσει περισσότερη πληροφορία από τις μη δυναμικές περιοχές ενός σήματος φωνής. Γι'αυτό το λόγο έχει αναπτυχθεί ο όρος δίφθογγος, ο οποίος περιγράφει την ηχογράφηση αυτής της μετάβασης.

Μερικές φορές οι φθόγγοι μελετώνται με βάση τους γειτονικούς τους φθόγγους. Έτσι προκύπτουν οι τρίφθογγοι και οι τετράφθογγοι. Όμως σε αντίθεση με την περίπτωση του δίφθογγου, καταλαμβάνουν σε μια κυματομορφή το ίδιο μήκος με ένα απλό φθόγγο. Διαφέρουν μόνο ως προς το όνομα. Αυτά τα αντικείμενα στην ορολογία του αλγορίθμου Sphinx ονομάζονται senones και η ανάλυσή τους θα γίνει στη συνέχεια.

Οι φθόγγοι σχηματίζουν υπομονάδες λέξεων όπως οι συλλαβές. Όταν η ομιλία γίνεται γρήγορη, οι φθόγγοι συχνά αλλάζουν αλλά οι συλλαβές παραμένουν ίδιες. Είναι πιο σταθερές σαν οντότητες και σχετίζονται με τον επιτονισμό που έχει γίνει στο εκάστοτε δείγμα φωνής. Υπομονάδες λέξεων μπορούν να σχηματιστούν και με άλλους τρόπους πέρα των συλλαβών, που εξαρτώνται από τη μορφολογική ποικιλία της κάθε γλώσσας.

Οι υπομονάδες λέξεων σχηματίζουν λέξεις. Οι λέξεις είναι πολύ σημαντικές στην αναγνώριση φωνής γιατί περιορίζουν αισθητά το πλήθος των συνδυασμών των φθόγγων. Αν υπάρχουν 40 φθόγγοι και κάθε λέξη περιέχει κατά μέσο όρο 7 φθόγγους, τότε μπορούν να παραχθούν 40^7 λέξεις. Αν και ο αριθμός αυτός είναι τεράστιος, ακόμα και ένας πολύ μορφωμένος άνθρωπος σπάνια χρησιμοποιεί περισσότερες από είκοσι χιλιάδες λέξεις, γεγονός που κάνει την αναγνώριση φωνής μία εφικτή διαδικασία.

Οι λέξεις μαζί με άλλους μη γλωσσικούς ήχους που ονομάζονται λέξεις πλή-

ρωσης(αναπνοή,βήχας,εεεε) σχηματίζουν εκφράσεις. Οι εκφράσεις είναι ηχητικά δείγματα ανάμεσα σε παύσεις. Δεν είναι απαραίτητο ότι αντιστοιχούν σε προτάσεις, οι οποίες έχουν περισσότερη σημασιολογική έννοια.

Κεφάλαιο 3

Αναγνώριση φωνής

3.1 Εισαγωγή

Η διαδικασία αναγνώρισης φωνής προσπαθεί να αντιστοιχίσει ένα σήμα φωνής στην ακολουθία λέξεων που αντιπροσωπεύει. Για να γίνει αυτό, παράγονται από το σήμα φωνής ένα σύνολο ακουστικών χαρακτηριστικών και στη συνέχεια χρησιμοποιούνται αλγόριθμοι αναγνώρισης προτύπων. Επομένως είναι πολύ σημαντική η επιλογή των ακουστικών χαρακτηριστικών για την απόδοση του συστήματος. Αν αυτά τα χαρακτηριστικά δεν αντιπροσωπεύουν το περιεχόμενο της φωνής ικανοποιητικά, τότε το σύστημα θα έχει μειωμένη απόδοση ασχέτως των αλγορίθμων που θα εφαρμοστούν.

Η επιλογή των χαρακατηριστικών δεν είναι κάτι εύκολο και είναι αντικείμενο μελέτης εδώ και δεκαετίες. Αυτό που δυσχεραίνει την επιλογή, είναι η ποικιλομορφία του σήματος φωνής. Για μία δεδομένη λέξη το αντίστοιχο σήμα φωνής μπορεί να διαφέρει από ομιλητή σε ομιλητή αλλά ακόμα και από πρόταση σε πρόταση όταν πρόκειται για τον ίδιο ομιλητή. Άλλοι παράγοντες όπως η φυσιολογία των οργάνων που παράγουν τη φωνή, το φύλο, το περιβάλλον και η έμφαση που μπορεί να δοθεί στο λόγο, διαφοροποιούν το φάσμα του σήματος που παράγεται με την ομιλία. Αν και ένας άνθρωπος μπορεί να αντιληφθεί αυτές τις διαφορές και να μην επηρεαστεί η αντίληψη του για το τι έχει ειπωθεί, τα συστήματα αναγνώρισης φωνής είναι ακόμα κατώτερα σε αυτό τον τομέα.

Με την πάροδο των ετών και την αύξηση της απόδοσης των συστημάτων αναγνώρισης φωνής, τα συστήματα αυτά βρήκαν εφαρμογή σε πολλούς τομείς. Τα πρώτα συστήματα βασίζονταν σε απομονωμένες λέξεις ή γράμματα από λεξικά περιορισμένου μεγέθους και εξαρτώνταν από τον ομιλητή. Στη συνέχεια αναπτύχθηκαν συστήματα που χρησιμοποιούσαν πιο μεγάλα λεξιλόγια της τά-

ξης των χιλίων λέξεων και ήταν για συνεχή ομιλία. Έπειτα έγινε χρήση γραπτής ομιλίας για τη δημιουργία μεγάλων λεξιλογίων που έφταναν και τις 65000 λέξεις. Ένα τέτοιο λεξιλόγιο παράγεται από το Wall Street Journal, όπου η πηγή των κειμένων είναι η εφημερίδα της Wall Street[3]. Πλέον, τα σύγχρονα συστήματα αναγνώρισης φωνής εφαρμόζονται για να αναγνωριστεί ομιλία που προκύπτει από συζήτηση ή είναι επιτηδευμένη ακόμα και σε θορυβώδη περιβάλλοντα.

3.2 Μοντελοποίηση φωνής

Για να προχωρήσουμε στην αναγνώριση φωνής πρέπει πρώτα να δηλώσουμε με ποιο τρόπο η φωνή μοντελοποιείται σαν ένα σύνολο στοχαστικών μοντέλων και συγκεκριμένα σαν κρυμμένα μοντέλα Markov. Τα HMM είναι αυτή τη στιγμή τα πιο δημοφιλή και επιτυχημένα ακουστικά μοντέλα στην αναγνώριση φωνής.

Καταρχάς, η φωνή στη μορφή του ακατέργαστου ηχητικού σήματος δεν είναι χρήσιμη στην αναγνώριση φωνής. Σε μία τέτοια μορφή που έχει καταγραφεί μόνο το πλάτος του σήματος είναι πολύ δύσκολο να αναγνωριστούν πρότυπα που μπορούν να σχετιστούν με αυτό που ειπώθηκε στην πραγματικότητα. Ως εκ τούτου πρέπει να μετασχήματισουμε τα δεδομένα αυτά στο χώρο της συχνότητας. Επίσης, για να εργαστούμε πιο αποδοτικά με στοχαστικά μοντέλα , πρέπει να χωρίσουμε τη φωνή σε μικρότερες ποσότητες δεδομένων, που θα ονομάζουμε πλαίσια. Το μέγεθος ενός πλαισίου ποικίλει από 10 έως 30ms και αυτά τα μικρά φωνητικά δείγματα έχουν χαρακτηριστικά μιας στάσιμης στοχαστικής διαδικασίας. Έτσι, στο φωνητικό σήμα εφαρμόζεται κάποιο παράθυρο το οποίο γενικά έχει την ιδιότητα να τείνει προς το μηδέν στα άκρα του ώστε να ελαχιστοποιηθεί η ασυνέχεια του σήματος εκτός του παραθύρου. Στη συνέχεια μία μέθοδος φασματικής ανάλυσης εφαρμόζεται σε κάθε ένα από τα παραχθέντα πλαίσια[4]. Το αποτέλεσμα αυτής της ενέργειας είναι η δημιουργία ενός διανύσματος αριθμών, του διανύσματος χαρακτηριστικών, που αναπαριστά τις φασματικές ιδιότητες της φωνητικής κυματομορφής. Η μέθοδος παραγωγής των διανυσμάτων χαρακτηριστικών είναι υπό μελέτη και ανάμεσα στις μεθόδους που έχουν προταθεί, είναι ο διακριτός μετασχηματισμός Fourier, μέθοδοι γραμμικής πρόβλεψης ελάχιστης φάσης όλων των πόλων και μοντέλα αυτόματης οπισθοδρόμησης/μεταβαλλόμενης μέσης τιμής[5]. Συνήθως αυτοί οι αριθμοί είναι παράγωγα του μετασχηματισμού Fourier.

3.2.1 Κρυμμένα μοντέλα Markov

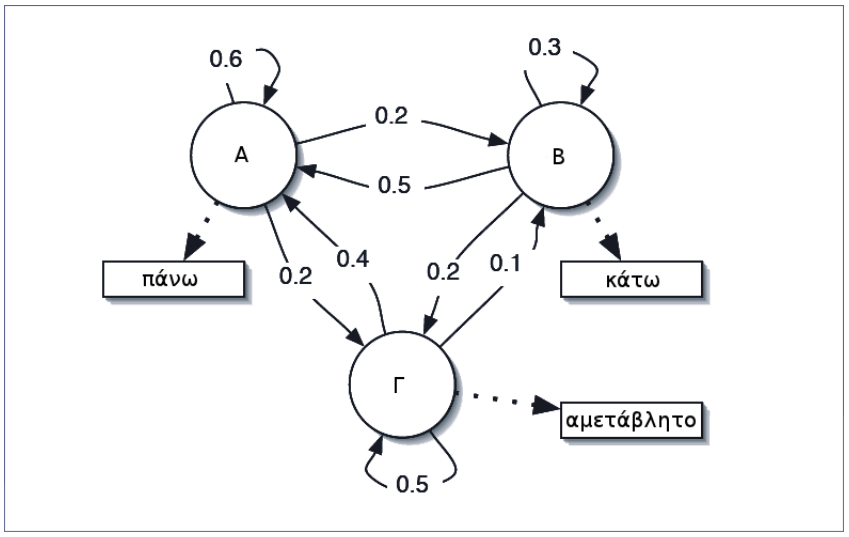
Τα κρυμμένα μοντέλα Markov (HMM) είναι ένα ισχυρό στατιστικό εργαλείο για τη μοντελοποίηση ενός συστήματος ή μιας ακολουθίας που μπορεί να χαρακτηριστεί ως διαδικασία Markov. Εφαρμόζονται σε πολλές περιπτώσεις αναγνώρισης προτύπων όπως η αναγνώριση φωνής εν προκειμένω αλλά και στην αναγνώριση γραφής, την αναγνώριση χειρονομιών τη βιοπληροφορική κ.ά. Η μαθηματική θεωρία της διαδικασίας Markov πήρε το όνομα της από τον Andrei Markov στις αρχές του 20ου αιώνα αλλά η θεωρία των κρυμμένων μοντέλων Markov αναπτύχθηκε από το Baum τη δεκαετία του 1960[6].

Ένα συνηθισμένο μοντέλο Markov αναπαριστάται με ένα αυτόματο πεπερασμένων καταστάσεων στο οποίο οι μεταβάσεις από τη μία κατάσταση στην άλλη είναι πιθανοτικές, δηλαδή γίνονται βάση μιας ορισμένης πιθανότητας. Σε αυτό το μοντέλο, οι καταστάσεις είναι άμεσα ορατές στον παρατηρητή και οι πιθανότητες μετάβασης είναι οι μόνες παράμετροι. Σε ένα κρυμμένο μοντέλο Markov, οι καταστάσεις δεν είναι άμεσα ορατές, αλλά είναι ορατές οι έξοδοι που παράγουν αυτές. Επομένως, η ακολουθία των παραχθέντων εξόδων μπορεί να δώσει κάποια πληροφορία για την ακολουθία των καταστάσεων που περνά το αυτόματο. Οι πιθανότητες μετάβασης δε, παραμένουν πάντα γνωστές.

Παράδειγμα μοντέλου Markov

Θα περιγράψουμε με μοντέλο Markov το δείκτη τιμών μιας χρηματιστηριακής αγοράς. Το μοντέλο έχει τρεις καταστάσεις, Α, Β και Γ. Η κατάσταση Α συμβολίζει την αισιοδοξία των χρηματιστών ότι ο δείκτης θα ανέβει, η Β την απαισιοδοξία ότι ο δείκτης θα πέσει και η κατάσταση Γ είναι μία ουδέτερη κατάσταση. Επίσης υπάρχουν τρεις παρατηρήσεις(έξοδοι καταστάσεων) για την κατάσταση του δείκτη. "κάτω" για την περίπτωση που είχαμε πτώση του δείκτη, "πάνω" για την περίπτωση που είχαμε άνοδο του δείκτη και "αμετάβλητο" για την περίπτωση που ο δείκτης δε σημείωσε κάποια μεταβολή. Το αυτόματο που αναπαριστά το μοντέλο φαίνεται στο σχήμα 3.1 .

Όπως φαίνεται στο αυτόματο, έχουν σημειωθεί με βελάκια οι μεταβάσεις από τη μία κατάσταση στην άλλη καθώς και οι αντίστοιχες πιθανότητες μετάβασης. Αν υποθέσουμε ότι έχουμε την ακολουθία παρατηρήσεων "πάνω"-"κάτω"-"κάτω" τότε μπορούμε εύκολα να επιβεβαιώσουμε ότι η ακολουθία καταστάσεων που παρήγαγε αυτές τις παρατηρήσεις ήταν η Α-Β-Β. Η δε πιθανότητα εμφάνισης αυτής της ακολουθίας, ισούται με το γινόμενο των πιθανοτήτων μετάβασης. Ειδικότερα θα είναι ίση με $0.2 \times 0.3 \times 0.3$.



Σχήμα 3.1: Παράδειγμα μοντέλου Markov[7]

Παράδειγμα κρυμμένου μοντέλου Markov

Το παραπάνω μοντέλο μπορεί να επεκταθεί σε HMM. Το νέο μοντέλο επιτρέπει όλες τις δυνατές παρατηρήσεις να παραχθούν από όλες τις καταστάσεις με βάση μια ορισμένη πιθανότητα. Γραφικά μπορεί να αναπαρασταθεί όπως στο σχήμα 3.2 .

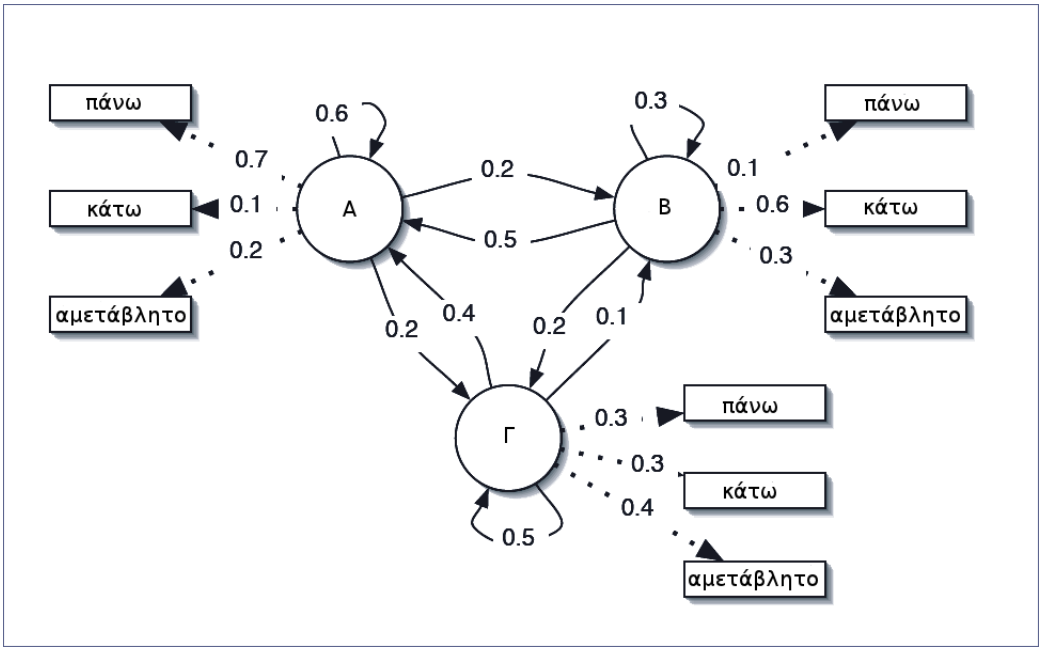
Η βασική διαφορά του νέου μοντέλου είναι ότι αν έχουμε την προαναφερθείσα ακολουθία παρατηρήσεων "πάνω"-"κάτω"-"κάτω", δεν μπορούμε να βρούμε ακριβώς την ακολουθία των καταστάσεων που την παρήγαγε και επομένως η ακολουθία καταστάσεων είναι "κρυμμένη". Μπορούμε όμως να υπολογίσουμε την πιθανότητα το μοντέλο να παρήγαγε την ακολουθία καθώς και το ποια ακολουθία καταστάσεων είναι πιο πιθανό να έχει παράγει τις συγκεκριμένες παρατηρήσεις.

Μαθηματικοί ορισμοί για τα HMMs

Ένα HMM ορίζεται ακολούθως:

$$\lambda = (A, B, \pi)$$

S είναι το σύνολο των καταστάσεων και V είναι το σύνολο των παρατηρήσεων:



Σχήμα 3.2: Παράδειγμα κρυμμένου μοντέλου Markov[7]

$$S = (s_1, s_2, ..., s_N)$$
$$V = (\upsilon_1, \upsilon_2, ..., \upsilon_M)$$

Ορίζουμε το Q σαν μία ακολουθία καταστάσεων μήκους T και τις παραχθείσες παρατηρήσεις ως O:

$$Q = q_1, q_2, ..., q_T \;\; O = o_1, o_2, ..., o_T$$

A είναι το διάνυσμα μετάβασης, το οποίο περιέχει την πιθανότητα να μεταβεί το αυτόματο από την κατάσταση i στην κατάσταση j. Οι πιθανότητες μετάβασης είναι ανεξάρτητες του χρόνου:

$$A = [\alpha_{ij}], \alpha_{ij} = P(q_t = s_j | q_{t-1} = s_i)$$

B είναι το διάνυσμα παρατηρήσεων που περιέχει την πιθανότητα μία παρατήρηση k να παραχθεί από μία κατάσταση j. Οι πιθανότητες αυτές είναι ανεξάρτητες του χρόνου t:

$$B = [b_i(k)], b_i(k) = P(x_t = \upsilon_k | q_t = s_i)$$

π είναι το αρχικό διάνυσμα πιθανοτήτων:

$$\pi = [\pi_i], \pi_i = P(q_1 = s_i)$$

Δύο υποθέσεις γίνονται από το μοντέλο. Σύμφωνα με την πρώτη, την υπόθεση Markov, η τωρινή κατάσταση εξαρτάται μόνο από την προηγούμενη κατάσταση.

$$P(q_t|q_1^{t-1}) = P(q_t|q_{t-1})$$

Σύμφωνα με τη δεύτερη υπόθεση, την υπόθεση της ανεξαρτησίας, η παραχθείσα παρατήρηση μια χρονική στιγμή t εξαρτάται μόνο από την τωρινή κατάσταση και είναι ανεξάρτητη από προηγούμενες παρατηρήσεις και καταστάσεις:

$$P(o_t|o_1^{t-1}, q_1^t) = P(o_t|q_t)$$

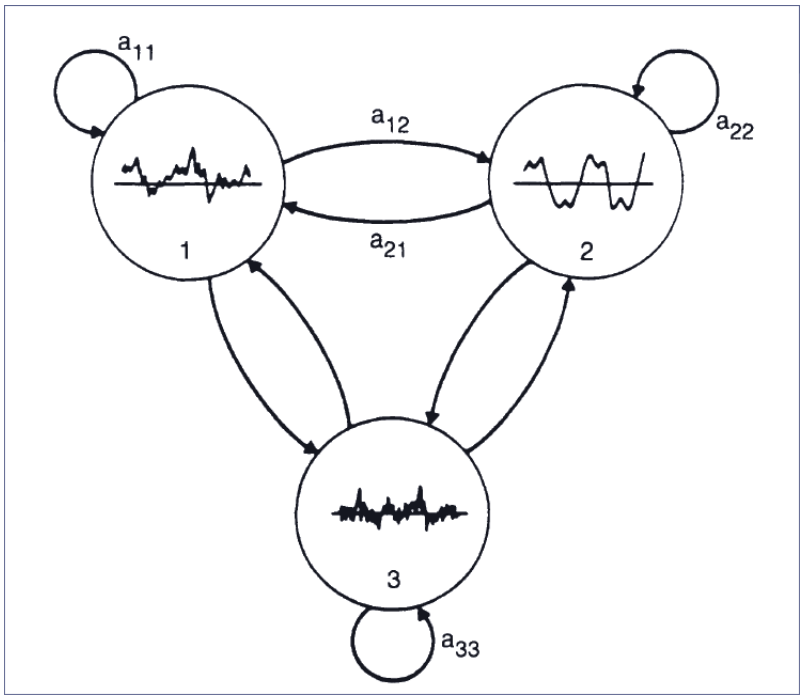
3.2.2 Κρυμμένα μοντέλα Markov στην αναγνώριση φωνής

Όπως αναφέρθηκε παραπάνω, τα HMM είναι τα πιο δημοφιλή στοχαστικά μοντέλα που χρησιμοποιούνται στην αναγνώριση φωνής. Για να αντιληφθεί κανείς πρακτικά πως γίνεται η χρήση τους, αρκεί να σκεφτεί ότι κάθε κατάσταση ενός HMM αντιστοιχεί στο κομμάτι ενός φθόγγου. Επομένως μία οποιαδήποτε κατάσταση αναπαριστά ένα κομμάτι της κυματομορφής του φωνητικού δείγματος που θέλουμε να αναγνωρίσουμε. Η γραφική αναπαράσταση αυτού φαίνεται στο σχήμα 3.3 .

Στο παραπάνω μοντέλο τριών καταστάσεων, δείχνεται σε κάθε κατάσταση το κομμάτι ενός φθόγγου που αναπαριστάται σαν κυματομορφή ενώ έχουν σημειωθεί και οι πιθανότητες μετάβασης. Για τη συσχέτιση ενός φθόγγου με τη μαθηματική έννοια των παρατηρήσεων σε ένα κρυμμένο μοντέλο Markov, πρέπει να εισάγουμε στο σημείο αυτό την έννοια του Μοντέλου Μείξης Γκαουσιανών Κατανομών(GMM).

Ένα μοντέλο Μείξης Γκαουσιανών Κατανομών είναι μία παραμετρική συνάρτηση πυκνότητας πιθανότητας που αναπαριστάται σαν ένα σταθμισμένο άθροισμα Γκαουσιανών κατανομών. Στην αναγνώριση φωνής ένα GMM μπορεί να αντιπροσωπεύσει το φάσμα συχνοτήτων ενός τυχαίου φωνητικού δείγματος. Η γενική μορφή ενός τέτοιου μοντέλου με M συνιστώσες(πλήθος κατανομών), βάρη συνιστωσών $P(\omega_m)$ και παραμέτρους θ_m είναι

$$p(x|\theta) = \sum_{m=1}^{M} P(\omega_m)p(x|\omega_m, \theta_m) \quad (3.1)$$



Σχήμα 3.3: Ένα κρυμμένο μοντέλο Markov τριών καταστάσεων

Εφόσον οι συνιστώσες είναι Γκαουσιανές κατανομές, μπορούν να περιγραφούν από τον τύπο

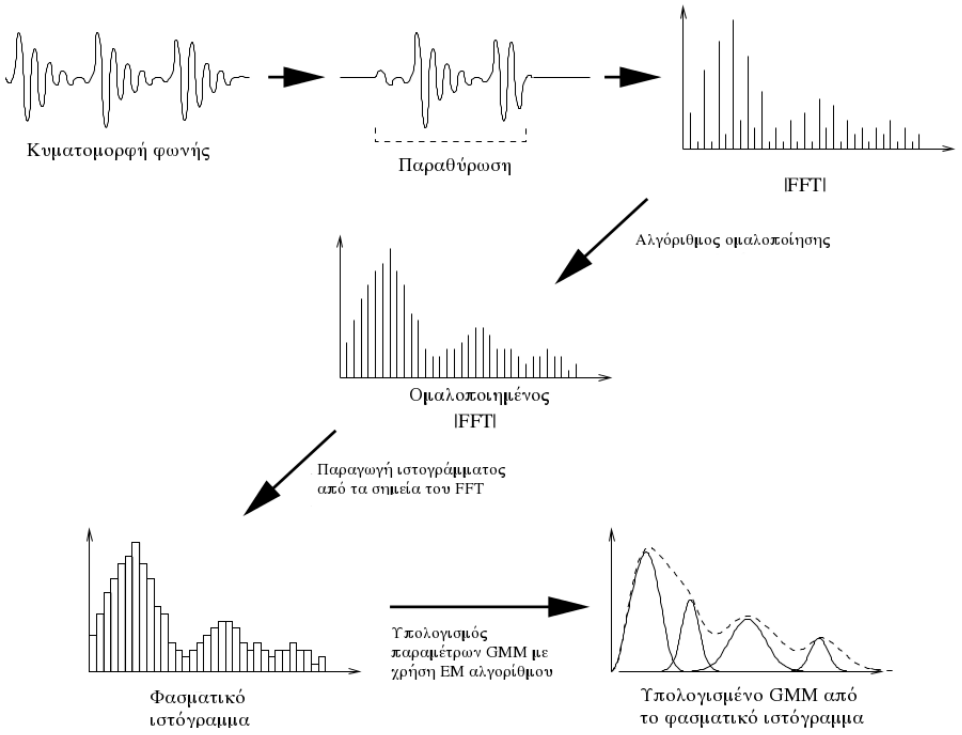
$$p(x|\omega_m, \theta_m) = \frac{1}{\sqrt{2\pi\sigma_m^2}} \exp\left[-\frac{(x-\mu_m)^2}{2\sigma_m^2}\right] \tag{3.2}$$

όπου μ_m είναι η μέση τιμή και σ_m η τυπική απόκλιση για τη συνιστώσα ω_m.

Η διαδικασία που ακολουθείται γενικά για τη δημιουργία ενός GMM με βάση ένα ορισμένο φωνητικό δείγμα είναι η εξής:

1. Στην κυματομορφή εφαρμόζεται ένα παράθυρο και αυτή χωρίζεται έτσι σε μικρότερα κομμάτια.
2. Σε κάθε κομμάτι εφαρμόζεται ο μετασχηματισμός Fourier και το προιόν που προκύπτει είναι το φάσμα του σήματος.
3. Στο φάσμα εφαρμόζεται κάποιος αλγόριθμος ομαλοποίησης. Σκοπός αυτού, είναι να αφαιρεθούν οι αιχμές του φάσματος. Αν δε γίνει αυτό και υπάρχουν

αιχμές που η απόσταση τους είναι μεγάλη, είναι πιθανόν να χρησιμοποιηθούν μόνο αυτές για την εύρεση των παραμέτρων των συνιστωσών και να αγνοηθούν έτσι οι γειτονικές αρμονικές του φάσματος.
4. Από το απότελεσμα του προηγούμενου βήματος παράγεται το φασματικό ιστόγραμμα.
5. Με χρήση ενός αλγορίθμου Expectation Maximization(EM algorithm) υπολογίζονται οι παράμετροι του GMM.
Η παραπάνω διαδικασία περιγράφεται στο ακόλουθο γράφημα



Σχήμα 3.4: Επισκόπηση της παραγωγής παραμέτρων GMM από ένα σήμα φωνής

Κεφάλαιο 4

Ο αλγόριθμος αναγνώρισης φωνής Sphinx-3

4.1 Εισαγωγή

Το Sphinx-3 είναι ένα ολοκληρωμένο σύστημα αναγνώρισης φωνής που αναπτύσσεται στο πανεπιστήμιο Carnegie Mellon και αποτελεί διάδοχο του Sphinx-II. Περιλαμβάνει έναν ακουστικό εκπαιδευτή και διάφορους αποκωδικοποιητές για αναγνώριση κειμένου, αναγνώριση φθόγγων, δημιουργία λίστας N-best κ.ά. Στην παρούσα διπλωματική εργασία, χρησιμοποιήθηκε ο αποκωδικοποιητής s3.X, όπου X είναι η εκάστοτε έκδοση. Στην περίπτωσή μας, X=6, που είναι η πιο πρόσφατη έκδοση.

Ο αποκωδικοποιητής s3.X είναι μια πρόσφατη υλοποίηση για αναγνώριση φωνής και ταυτοποίησή της σε κείμενο. Κύριος στόχος του είναι η βελτίωση σε ταχύτητα του αρχικού αποκωδικοποιητή Sphinx-3. Τρέχει περίπου 10 φορές πιο γρήγορα από τον τελευταίο όταν γίνεται χρήση μεγάλου λεξικού. Εν συντομία τα χαρακτηριστικά και οι περιορισμοί του είναι τα ακόλουθα:

- Χρόνος αναγνώρισης με μεγάλο λεξικό 5-10 φορές μικρότερος του πραγματικού χρόνου
- Περιορίζεται σε πλήρως συνεχή ακουστικά μοντέλα
- Περιορίζεται σε τοπολογίες HMM 3 ή 5 κατάστασεων
- Δίγραμμο ή τρίγραμμο γλωσσικό μοντέλο

- Ζωντανή λειτουργία ή επεξεργασία κατά ομάδες από προηχογραφημένα δείγματα φωνής

4.2 Επισκόπηση του αποκωδικοποιητή s3.X

Ο αποκωδικοποιητής s3.X βασίζεται στον αλγόριθμο αναζήτησης Viterbi και σε ευριστικές με δέσμη αναζήτησης. Χρησιμοποιεί μία δομή αναζήτησης που αναπαριστά το λεξικό σαν δέντρο και οι είσοδοί του προέρχονται από προηχογραφημένα δείγματα φωνής που βρίσκονται σε απλή PCM μορφή. Δεν έχουν υποστεί δηλαδή κάποια μετατροπή σε μορφές όπως το mp3 ή το wmv.

Είσοδοι στον αποκωδικοποιητή:

1. Το λεκτικό μοντέλο
2. Το ακουστικό μοντέλο
3. Το γλωσσικό μοντέλο
4. Τα φωνητικά δείγματα προς αναγνώριση

Λεκτικό μοντέλο

Το λεκτικό μοντέλο περιέχει τις προφορές όλων των λέξεων που χρησιμοποιεί ο αποκωδικοποιητής. Στο Sphinx, οι προφορές δηλώνονται σαν μια ακολουθία φωνημάτων. Έτσι, στο αρχείο που αποτελεί το λεξικό, υπάρχουν δύο στήλες. Στην αριστερή δηλώνονται οι λέξεις και στη δεξιά οι αντίστοιχες προφορές τους. Ένα παράδειγμα είναι το παρακάτω:

ZERO	Z IH R OW
ONE	W AH N
TWO	T UW
THREE	TH R IY
FOUR	F AO R
FIVE	F AY V
SIX	S IH K S
SEVEN	S EH V AX N
EIGHT	EY TD
NINE	N AY N

Μία λέξη μπορεί να έχει παραπάνω από μία προφορά. Αυτές δηλώνονται στο λεξικό σαν επιπλέον γραμμές όπως στο ακόλουθο παράδειγμα:

ACTUALLY	AE K CH AX W AX L IY
ACTUALLY(2nd)	AE K SH AX L IY
ACTUALLY(3rd)	AE K SH L IY

Ακουστικό μοντέλο

Το Sphinx-3 βασίζεται σε ακουστικά μοντέλα υποφθόγγων. Αρχικά, οι βασικοί ήχοι σε μία γλώσσα ταξινομούνται σε φθόγγους. Στην αγγλική γλώσσα για παράδειγμα, υπάρχουν περίπου 50 φθόγγοι.

Οι φθόγγοι στη συνέχεια ταξινομούνται σε τρίφθογγους εξαρτημένους από τα συμφραζόμενα, δηλαδή σε φθόγγους που προκύπτουν όταν είναι γνωστά τα δεξιά και τα αριστερά συμφραζόμενα. Ο λόγος που γίνεται αυτή η ταξινόμηση είναι ότι ο ίδιος φθόγγος όταν περιέχεται σε διαφορετικά συμφραζόμενα μπορεί να έχει ακουστικές αναπαραστάσεις που διαφέρουν μεταξύ τους κατά πολύ, απαιτώντας έτσι ξεχωριστά ακουστικά μοντέλα. Ο ίδιος φθόγγος σε μία λέξη θα μπορούσε υπάρχει δύο φορές, αλλά τη μία να είναι έρρινος και την άλλη όχι εξαιτίας διαφορετικών συμφραζομένων.

Σε αντίθεση με τους τρίφθογγους, ένας φθόγγος που θεωρείται ότι δεν έχει συμφραζόμενα ονομάζεται *ανεξάρτητος συμφραζομένων* ή *βασικός φθόγγος*. Αξίζει να σημειωθεί ότι η εξάρτηση από τα συμφραζόμενα, μπορεί να επεκταθεί και ανάμεσα σε διαδοχικές λέξεις. Τα αριστερά συμφραζόμενα του πιο αριστερού βασικού φθόγγου σε μία λέξη εξαρτώνται από την ακριβώς προηγούμενη λέξη.

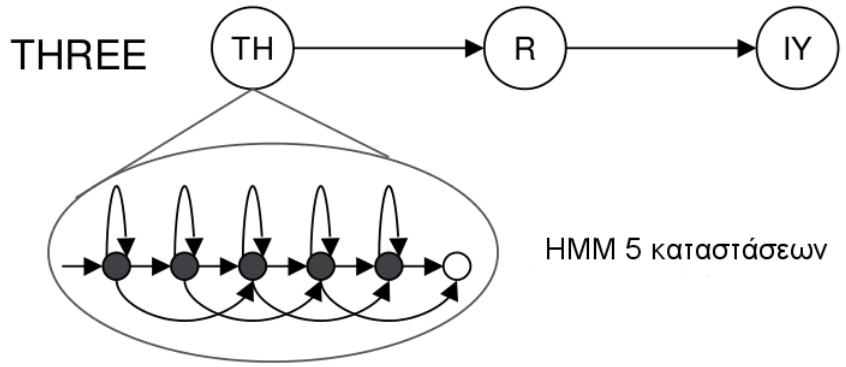
Οι φθόγγοι επίσης διαχωρίζονται ανάλογα τη θέση τους μέσα σε μία λέξη. Μπορεί να βρίσκονται στην αρχή, στο τέλος, ενδιάμεσα ή να είναι μόνοι τους αν η λέξη αυτή προφέρεται με ένα μόνο φθόγγο.

Τα ακουστικά μοντέλα που κατασκευάζονται για τους τρίφθογγους χαρακτηρίζονται από τις προαναφερθέντες δυνατές θέσεις ενός φθόγγου σε μία λέξη ώστε να υπάρχει μεγαλύτερη ακρίβεια σε σχέση με τη χρήση μόνο μοντέλων που προκύπτουν από τους βασικούς φθόγγους. Κάθε τρίφθογγος μοντελοποιείται με ένα κρυμμένο μοντέλο Markov το οποίο έχει 3 ή 5 καταστάσεις. Ωστόσο, σε μία γλώσσα όπως η Αγγλική που έχει 50 βασικούς φθόγγους, χρησιμοποιώντας 4 δυνατές θέσεις και HMM 3 καταστάσεων θα προέκυπταν $50^3 * 4 * 3$ διαφορετικές καταστάσεις. Ένα τέτοιο σύνολο μοντέλων είναι πολύ μεγάλο και πρακτικά αδύνατο να εκπαιδευτεί. Γι'αυτό το λόγο, οι καταστάσεις χωρίζονται σε μικρότερες ομάδες. Κάθε τέτοια ομάδα αποτελεί ένα senone και όλες οι καταστάσεις ενός senone μοιράζονται το ίδιο στατιστικό μοντέλο για την αναπαράστασή τους, που

δεν είναι άλλο από ένα μοντέλο μείξης Γκαουσιανών κατανομών. Η διαδικασία χωρισμού των καταστάσεων σε senones είναι πέραν των πλαισίων της παρούσας διπλωματικής και περιγράφεται στη διδακτορική διατριβή του Mei-Yuh Hwang.

Ακόμα, κάθε τρίφθογγος έχει ένα πίνακα με τις πιθανότητες μετάβασης από τη μία κατάσταση στην άλλη. Για τη διατήρηση των απαιτούμενων πόρων σε χαμηλά επίπεδα, υπάρχει ένας τέτοιος πίνακας για κάθε βασικό φθόγγο και όλοι οι τρίφθογγοι που προκύπτουν από τον ίδιο βασικό φθόγγο μοιράζονται από κοινού τον πίνακά του.

Χρησιμοποιώντας για παράδειγμα τη λέξη "three" και HMM 5 καταστάσεων θα μπορούσαμε γραφικά να αναπαραστήσουμε τις καταστάσεις που την περιγράφουν όπως παρακάτω



Σχήμα 4.1: Γραφική αναπαράσταση ενός HMM στο Sphinx-3

Όπως φαίνεται, κάθε τρίφθογγος στη λέξη "three" μοντελοποιείται με ένα HMM 5 καταστάσεων. Κάθε HMM χαρακτηρίζεται από ένα μοναδικό αριθμό, το State Sequence Id(SSID) και κάθε κατάσταση είναι ένα senone, δλδ ένα GMM. Στο σχήμα, εκτός των 5 καταστάσεων, υπάρχει και μια έκτη κατάσταση. Η κατάσταση αυτή ονομάζεται *κατάσταση εξόδου* και συμβολίζει το τέλος της Markov διαδικασίας. Δεν ορίζεται μετάβαση από την κατάσταση αυτή στην ίδια και επίσης δεν αναπαριστά κάποιο κομμάτι του τρίφθογγου.

- **Εξαγωγή ακουστικών χαρακτηριστικών**
  Όπως έχει ήδη αναφερθεί, τα φωνητικά δείγματα που χρησιμοποιούνται

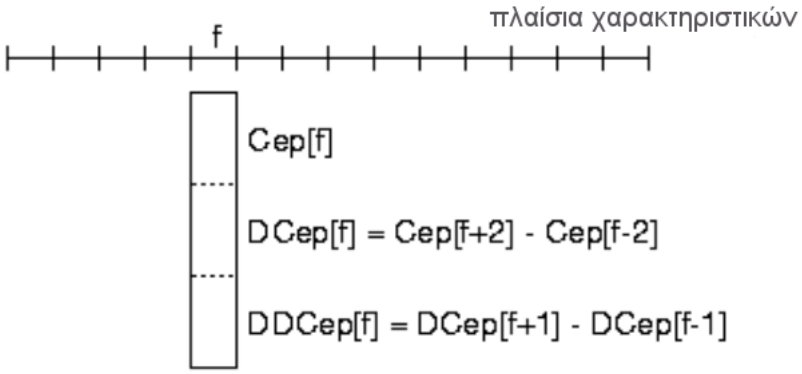
στην αναγνώριση φωνής, επεξεργάζονται με τέτοιο τρόπο ώστε να παράγουν διανύσματα χαρακτηριστικών. Η δημιουργία των ακουστικών μοντέλων γίνεται με βάση αυτά τα διανύσματα.

Στο Sphinx3 η δημιουργία των διανυσμάτων χαρακτηριστικών είναι μία διαδικασία δύο φάσεων. Στην πρώτη φάση το φωνητικό δείγμα μετατρέπεται στο πεδίο της συχνότητας και προκύπτει το φάσμα του, το οποίο στη συνέχεια μπορεί να περάσει σαν είσοδος στο λογισμικό του Sphinx. Η πρώτη φάση εκτελείται από εξωτερικό αλγόριθμο. Τα φωνητικά δείγματα είναι δειγματοληπτημένα με συχνότητα δειγματοληψίας 8 ή 16KHz, ανάλογα με το εύρος της ομιλίας και κβαντισμένα με 16bits. Εφαρμόζεται στον καθένα ένα παράθυρο με αποτέλεσμα την παραγωγή πλαισίων διάρκειας 25.625ms[8] . Ο αριθμός των δειγμάτων σε ένα πλαίσιο εξαρτάται από το ρυθμό δειγματοληψίας. Η έξοδος είναι μια ακολουθία από διανύσματα 13 διαστάσεων πραγματικών αριθμών που αναπαριστούν το φάσμα. Τα πλαίσια επικαλύπτονται και έτσι παράγονται 100 πλαίσια ανά δευτερόλεπτο και επομένως 100 διανύσματα ανά δευτερόλεπτο. Κάθε 10ms εμφανίζεται ένα καινούριο πλαίσιο.

Στη δεύτερη φάση, τα διανύσματα που αναπαριστούν το φάσμα μετατρέπονται σε διανύσματα χαρακτηριστικών. Αυτή η διαδικασία γίνεται σε 3 βήματα:

1. Προαιρετικά στο φάσμα γίνεται κανονικοποίηση μέσης τιμής και διασποράς.
2. Προαιρετικά γίνεται κανονικοποίηση της ισχύς των φασματικών διανυσμάτων
3. Το τελικό διάνυσμα χαρακτηριστικών δημιουργείται προσθέτοντας στο φασματικό διάνυσμα ένα ή περισσότερα χρονικά παράγωγα. Στο s3.X το διάνυσμα χαρακτηριστικών σε κάθε πλαίσιο υπολογίζεται εισάγοντας στο φασματικό διάνυσμα πρώτα και δεύτερα παράγωγα. Το αποτέλεσμα είναι ένα διάνυσμα 39 διαστάσεων.

- **Εκπαίδευση ακουστικών μοντέλων**
  Η διαδικασία υπολογισμού του στατιστικού μοντέλου για καθε senone περιγράφεται στα παρακάτω βήματα:



Σχήμα 4.2: Δημιουργία διανύσματος χαρακτηριστικών

1. Εξεύρεση της συλλογής με τα δεδομένα εκπαίδευσης. Αυτή μπορεί να περιέχει χιλιάδες προτάσεις που αποτελούνται από το κείμενο που έχει ειπωθεί και από το αντίστοιχο ηχητικό δείγμα.
2. Για κάθε πρόταση, το ηχητικό δείγμα μετατρέπεται σε διάνυσμα χαρακτηριστικών όπως περιγράφηκε παραπάνω.
3. Για κάθε πρόταση, το κείμενο που της αντιστοιχεί, μετατρέπεται σε μια γραμμική ακολουθία τρίφθογγων HMMs χρησιμοποιώντας το λεξικό.
4. Για κάθε πρόταση, βρίσκεται η καλύτερη ακολουθία καταστάσεων HMM με βάση το διάνυσμα χαρακτηριστικών της πρότασης. Καλύτερη ακολουθία καταστάσεων είναι αυτή που έχει τη μικρότερη απόκλιση από το διάνυσμα χαρακτηριστικών.
5. Για κάθε senone, συλλέγονται όλα τα πλαίσια από τη συλλογή δεδομένων εκπαίδευσης που ταίριαξαν με το συγκεκριμένο senone στο προηγούμενο βήμα και κατασκευάζεται ένα στατιστικό μοντέλο για την αντίστοιχη συλλογή από διανύσματα χαρακτηριστικών.

Όπως φαίνεται από τα βήματα, θέλουμε να εκπαιδεύσουμε τα senones αλλά παράλληλα τα χρειαζόμαστε για να βρουμε την καλύτερη ακολουθία καταστάσεων. Το πρόβλημα αυτό λύνεται με χρήση του επαναληπτικού αλγορίθμου Baum-Welch. Ο αλγόριθμος ξεκινά με ένα σύνολο μοντέλων, τα οποία μπορεί να μην είναι καθόλου παραμετροποιημένα, για τα senones. Στη συνέχεια επαναλαμβάνει τα τελευταία δύο βήματα πολλές φορές χρη-

σιμοποιώντας σε κάθε επανάληψη τα μοντέλα που προέκυψαν από την προηγούμενη.

Η εκπαίδευση των ακουστικών μοντέλων περιλαμβάνει και τον υπολογισμό των πιθανοτήτων μετάβασης από κατάσταση σε κατάσταση.

Γλωσσικό μοντέλο

Το γλωσσικό μοντέλο περιέχει τις πιθανότητες εμφάνισης των λέξεων μιας γλώσσας. Ο αποκωδικοποιητής s3.X χρησιμοποιεί δίγραμμο ή τρίγραμμο γλωσσικό μοντέλο το οποίο αποθηκεύεται σε δυαδική μορφή. Ειδικότερα ένα τρίγραμμο γλωσσικό μοντέλο αποτελείται από τα ακόλουθα[9]:

- Μονογράμματα: Το σύνολο των λέξεων σε αυτό το γλωσσικό μοντέλο και η πιθανότητα εμφάνισης κάθε μίας στη γλώσσα.
- Διγράμματα: Η πιθανότητα εμφάνισης μίας λέξης, έστω "λέξη2" εξαρτάται από την πιθανότητα εμφάνισης της αμέσως προηγούμενής της λέξης, έστω "λέξη1". Μαθηματικά η πιθανότητα αυτή είναι $P(\lambda\varepsilon\xi\eta 2|\lambda\varepsilon\xi\eta 1)$. Ένα γλωσσικό μοντέλο συνήθως περιέχει ένα υποσύνολο των πιθανών συνδυασμών δύο λέξεων και όχι το πλήρες σύνολο.
- Τριγράμματα: Σε αυτή την περίπτωση η πιθανότητα εμφάνισης μιας λέξης εξαρτάται από την πιθανότητα εμφάνισης της ακολουθίας των δύο προηγούμενων λέξεων. Με μαθηματικό συμβολισμό $P(\lambda\varepsilon\xi\eta 3|\lambda\varepsilon\xi\eta 1, \lambda\varepsilon\xi\eta 2)$. Όπως και με τα διγράμματα, δε χρειάζεται να καλυφθούν όλοι οι συνδυασμοί λέξεων από τα τριγράμματα.

  Η πιθανότητα μιας ολόκληρης πρότασης είναι το γινόμενο των πιθανοτήτων των λέξεων που την απαρτίζουν. Για παράδειγμα η πιθανότητα της πρότασης "HOW ARE YOU" είναι:

  P(HOW | <s>) * P(ARE | <s>,HOW) * P(YOU | HOW,ARE) *
  P(</s> | ARE,YOU)

  όπου <s> και </s> είναι ειδικοί χαρακτήρες για την αρχή και το τέλος μιας πρότασης αντίστοιχα και περιέχονται στα μονογράμματα.

Φωνητικά δείγματα

Ο αποκωδικοποιητής δίνει τη δυνατότητα αναγνώρισης ζωντανής ομιλίας, δλδ ομιλίας που παράγεται εκείνη τη στιγμή από το χρήστη και ηχογραφείται

με την κάρτα ήχου του και προηχογραφημένης ομιλίας. Στη δεύτερη περίπτωση οι προτάσεις για αναγνώριση περιγράφονται σε ένα αρχείο εισόδου και έχουν ήδη μετατραπεί στη συχνότητα. Κάθε πρόταση δεν μπορεί να διαρκεί περισσότερο από 300 δευτερόλεπτα και χωρίζεται σε μικρότερα κομμάτια των 10 δευτερολέπτων, αν χρειάζεται.

4.3 Λειτουργία του αποκωδικοποιητή s3.X

Αρχικοποίηση

Ο αποκωδικοποιητής παραμετροποιείται κατά τη διάρκεια της αρχικοποίησης και οι παράμετροι του παραμένουν σταθερές μέχρι το τέλος της εκτέλεσής του. Δεν μπορεί κάποιος να επαναρυθμίσει τα ακουστικά μοντέλα ώστε να προσαρμοστούν στις αλλαγές την εισόδου, δηλαδή από πρόταση σε πρόταση.

- **Αρχικοποιήση της βάσης του λογαρίθμου** Το Sphinx εκτελεί όλες τους υπολογισμούς των πιθανοτήτων στο λογαριθμικό τομέα. Η βάση του λογαρίθμου επιλέγεται με τέτοιο τρόπο ώστε οι πιθανότητες να διατηρούνται μέσα στο εύρος τιμών ακέραιων μεταβλητών 32-bit. Η προκαθορισμένη τιμή για τη βάση είναι 1.0003[8] αλλά μπορεί να αλλαχθεί από τη μεταβλητή εισόδου -logbase ώστε να αποφευχθεί τυχόν υπερχείλιση των 32-bit μεταβλητών. Η βάση μπορεί να αλλάξει σε μεγάλο εύρος τιμών χωρίς να επηρεαστεί η αναγνώριση των προτάσεων.

- **Αρχικοποιήση μοντέλων** Το λεκτικό, το ακουστικό και το λεκτικό μοντέλο που ορίζονται στις μεταβλητές εισόδου του αποκωδικοποιητή "φορτώνονται" κατά τη διάρκεια της αρχικοποίησης. Αυτό το σύνολο μοντέλων χρησιμοποιείται για την αναγνώριση όλων των προτάσεων.

- **Ενεργό λεξιλόγιο**

  Αφού τα μοντέλα έχουν "φορτωθεί", καθορίζεται το ενεργό λεξιλόγιο. Είναι το σύνολο των λέξεων που ο αποκωδικοποιητής μπορεί να αναγνωρίσει. Το ενεργό λεξιλόγιο παράγεται από το λεκτικό και το γλωσσικό μοντέλο ως εξής:

  1. Βρίσκεται η διατομή των λέξεων που ανήκουν στο γλωσσικό μοντέλο και στο λεξικό.

2. Προστίθενται χρησιμοποιώντας το κεντρικό λεξικό, όλες οι εναλλακτικές προφορές των λέξεων που βρέθηκαν στο πρώτο βήμα
3. Προστίθενται όλες οι λέξεις πλήρωσης από το αντίστοιχο λεξικό εκτός των <s> και </s>.

Το ενεργό λεξιλόγιο παραμένει ενεργό για όλη τη διάρκεια αναγνώρισης μιας ομάδας προτάσεων. Δεν μπορούν να προστεθούν ή να αφαιρεθούν από αυτό λέξεις δυναμικά.

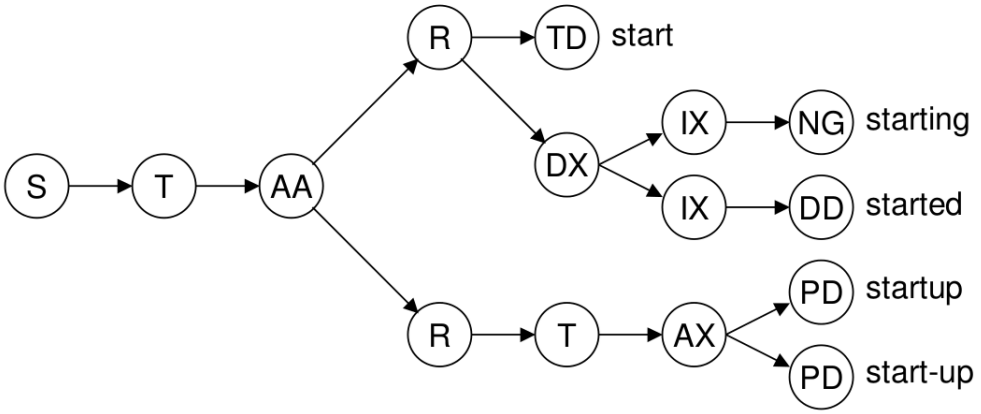
- **Κατασκευή Λεκτικών Δέντρων**

  Με βάση το ενεργό λεξιλόγιο, ο αποκωδικοποιητής κατασκευάζει μια δομή δέντρου για να αποθηκεύσει τις λέξεις. Κίνητρο για τη δημιουργία μιας τέτοιας δομής είναι το γεγονός ότι από τα ενεργά HMMs, δηλαδή από τα HMM που χρησιμοποιεί ο αλγόριθμος για την αναζήτηση που θα περιγραφεί παρακάτω, τα περισσότερα είναι μοντέλα που αντιστοιχούν στην αρχή μιας λέξης. Όμως από αυτά, πολύ λιγότερα διαφοροποιούνται. Πρακτικά αυτό δείχνει ότι σε ένα λεξιλόγιο, πολλές λέξεις ξεκινούν με τον ίδιο τρίφθογγο ή καλύτερα, πολλές λέξεις χρησιμοποιούν για πρώτο HMM ακριβώς το ίδιο. Αυτές οι λέξεις μπορούν να παρασταθούν με ένα δέντρο που η ρίζα του είναι το κοινό πρώτο HMM και οι ενδιάμεσοι κόμβοι είναι τα υπόλοιπα HMMs που συμβολίζουν τους εναπομείναντες τρίφθογγους. Οι κόμβοι μοιράζονται μεταξύ των λέξεων όταν οι τρίφθογγοι έχουν ακριβώς το ίδιο SSID.

  Στο σχήμα 4.3, φαίνεται το λεκτικό δέντρο που παράγεται για τις λέξεις start,starting,started,startup και start-up. Η ρίζα και οι δύο πρώτοι κόμβοι μοιράζονται από τις λέξεις ένω τα HMMs που διαφέρουν σχηματίζουν υποδέντρα. Ο τελευταίος κόμβος κάθε λέξης δε μοιράζεται ποτέ, όπως φαίνεται και στην περίπτωση των λέξεων startup και start-up.

Επεξεργασία φωνητικών δειγμάτων

Μετά την αρχικοποίηση, ο αποκωδικοποιητής επεξεργάζεται τις προτάσεις προς αναγνώριση μία μία. Κάθε πρόταση επεξεργάζεται σειριακά και χρονικά, δηλαδή όλα τα πλαίσια της πρότασης το ένα μετά το άλλο με βάση τη θέση τους στην πρόταση και από το παλιότερο στο νεότερο.



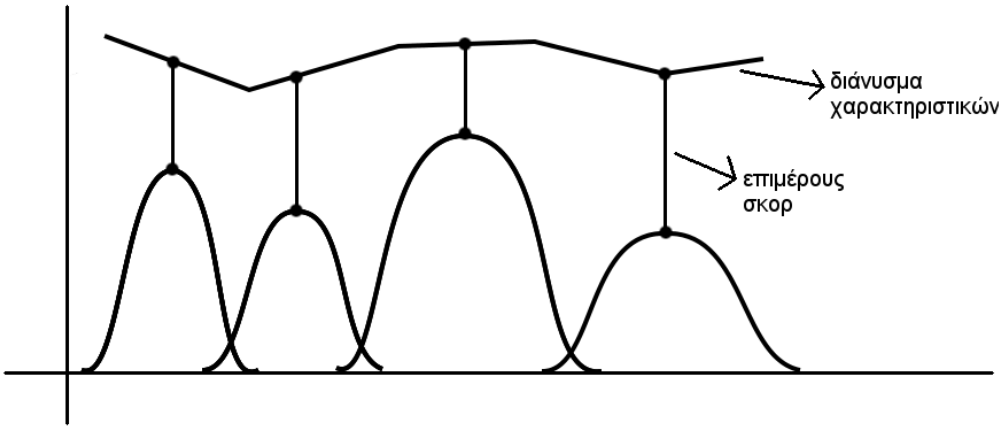
Σχήμα 4.3: Παράδειγμα λεκτικού δέντρου

Κάθε πλαίσιο ο αλγόριθμος, το συγκρίνει με όλα τα GMMs που περιέχονται στο ακουστικό μοντέλο παράγοντας έτσι τόσα σκορ για κάθε πλαίσιο όσα είναι και τα GMM μοντέλα. Η σύγκριση αυτή γίνεται με τον ακόλουθο τύπο

$$\frac{1}{\sqrt{2\pi^{\nu} det(\sigma^2)}} \exp\left[-\sum_{i=1}^{\nu} \frac{(\chi[i]-\mu[i])^2}{2\sigma^2[i]}\right] \tag{4.1}$$

όπου μ το διάνυσμα μέσων τιμών και σ^2 το διάνυσμα διασποράς μιας από τις Γκαουσιανές συνιστώσες στο εκάστοτε GMM και χ το διάνυσμα χαρακτηριστικών του εκάστοτε πλαισίου. Τέλος, ν το μήκος των διανυσμάτων.

Το αποτέλεσμα που προκύπτει από τον παραπάνω τύπο αποτελεί ένα επιμέρους σκορ. Είναι το σκορ μιας από τις συνιστώσες σε ένα GMM συγκρινόμενη με το διάνυσμα χαρακτηριστικών ενός δοθέντος πλαισίου. Για να υπολογισθεί το συνολικό σκορ ενός GMM σε σχέση με ένα πλαίσιο, πρέπει να αθροιστούν τα επιμέρους σκορ των Γκαουσιανών κατανομών. Έτσι, αν γίνεται χρήση GMM με 4 συνιστώσες, βρίσκονται τα τέσσερα επιμέρους σκορ και το άθροισμα τους είναι το σκορ του μοντέλου αυτού. Γραφικά η διαδικασία αυτή φαίνεται παρακάτω:



$$\text{σκορ} = \sum^{i=4} \text{επιμέρους σκορ}\,[i]$$

Σχήμα 4.4: Εύρεση σκορ σε ένα GMM 4 συνιστωσών

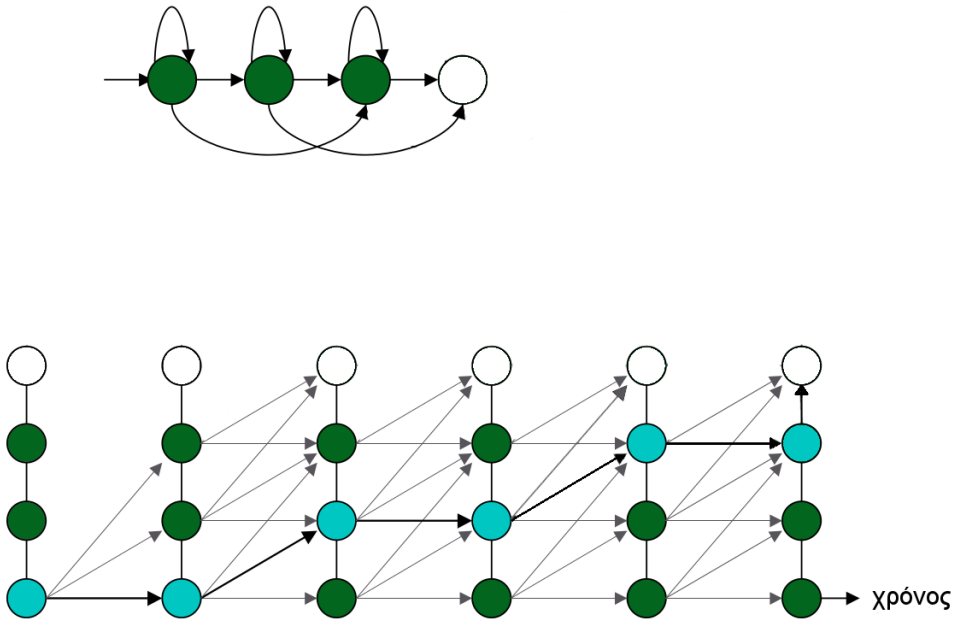
Στη συνέχεια ο αποκωδικοποιητής έχοντας τα σκορ των GMM, χρησιμοποιεί τον αλγόριθμο αναζήτησης Viterbi. Σκοπός αυτού είναι να βρεθεί το καλύτερο μονοπάτι καταστάσεων, δηλαδή η ακολουθία εκείνη από GMM που τα ακουστικά χαρακτηριστικά τους προσομοιώνουν καλύτερα τα πλαίσια της πρότασης που θέλουμε να αναγνωρίσουμε. Για να βρεθεί αυτό το μονοπάτι, ο αλγόριθμος Viterbi χρησιμοποιεί μία μετρική ώστε να συγκρίνει μεταξύ τους όλα τα δυνατά μονοπάτια και να επιλέξει το καλύτερο. Η μετρική αυτή είναι η ακόλουθη:

$$P_j(t) = max[P_i(t-1)a_{ij}b_j(t)]$$

όπου
$P_j(t)$: το συνολικό σκορ για ένα μονοπάτι που καταλήγει στην κατάσταση j για ένα συγκεκριμένο πλαίσιο
a_{ij} : η πιθανότητα μετάβασης από την κατάσταση i στην κατάσταση j
b_j : το σκορ μιας κατάστασης για ένα συγκεκριμένο πλαίσιο

Η παρουσία της μεταβλητής του χρόνου στη μετρική, συμβολίζει ότι η εύρεση των σκορ των μονοπατιών γίνεται για κάθε πλαίσιο αφού αυτά επεξεργάζονται σειριακά. Θα μπορούσε κάποιος να αντικαταστήσει τη χρονική μεταβλητή t με μία άλλη μεταβλητή που θα συμβόλιζε τον αριθμό του πλαισίου που επεξεργάζεται εκείνη τη στιγμή.

Ο αλγόριθμος Viterbi εφαρμόζεται σε κάθε HMM που αποτελεί κόμβο των λεκτικών δεντρών. Θεωρώντας ότι τα HMM είναι τριών καταστάσεων και επιλέγοντας ένα από αυτά που εν τέλει αντιπροσωπεύει τα πλαίσια εισόδου, η διαδικασία της ανεύρεσης του καλύτερου μονοπατιού περιγράφεται στο ακόλουθο σχήμα:



Σχήμα 4.5: Ο αλγόριθμος Viterbi σε ένα HMM 3 καταστάσεων

Με το μπλε χρώμα σημειώνεται η κατάσταση που έχει κάθε στιγμή το καλύτερο σκορ στο μονοπάτι.

Πρέπει να σημειωθεί, ότι μετά την εφαρμογή του Viterbi για ένα πλαίσιο και πριν την επεξεργασία του επόμενου, ο αποκωδικοποιητής ενημερώνει τα σκορ

στα HMMs σε δύο επίπεδα. Σε επίπεδο τριφθόγγων και σε επίπεδο λέξεων. Για το πρώτο, το σκορ εξόδου από ένα HMM, δηλαδή το σκορ του μονοπατιού μετά την εύρεση του για ένα πλαίσιο, περνάει σαν σκορ εισόδου στο HMM που αποτελεί παιδί του στο λεκτικό δέντρο. Με αυτό τον τρόπο, όταν εκτελεστεί ο Viterbi για το επόμενο πλαίσιο, τα σκορ που θα προκύψουν από αυτόν για κάθε HMM θα είναι αποτέλεσμα του συνυπολογισμού και των σκορ των HMM που προηγούνται στο λεκτικό δέντρο. Γίνεται αντιληπτό ότι, το καλύτερο μονοπάτι ενός τριφθόγγου μιας λέξης δεν εξαρτάται αποκλειστικά από τα GMM που το αντιπροσωπεύουν αλλά εξαρτάται και από αυτά του προηγούμενου φθόγγου μέσα στη λέξη.

Για το επίπεδο λέξεων πρέπει πρώτα να παρουσιάσουμε τη βασική δομή που χρησιμοποιεί ο αποκωδικοποιητής για την αποθήκευση των λέξεων που αναγνωρίζει κάθε στιγμή. Η δομή αυτή είναι ένας πίνακας ο οποίος διατηρείται σε όλη τη διάρκεια αναγνώρισης μιας πρότασης. Οι βασικές πληροφορίες που περιέχει μια καταχώρηση σε αυτόν τον πίνακα, είναι ο *χαρακτηριστικός αριθμός* της λέξης που αφορά, ο *χαρακτηριστικός αριθμός* της ακριβώς προηγούμενης χρονικά καταχώρησης και το σκορ που έχει προκύψει αθροίζοντας τα επιμέρους σκορ των λέξεων που προηγούνται. Επίσης κάθε καταχώρηση σχετίζεται με μια κατάσταση γλωσσικού μοντέλου. Μια κατάσταση γλωσσικού μοντέλου (ΓΜ κατάσταση) αποθηκεύει μία λέξη και την προηγούμενή της.

Οι καταχωρήσεις σε αυτόν τον πίνακα συμβολίζουν τις εξόδους λέξεων που γίνονται σε κάθε πλαίσιο. Πρακτικά, μια λέξη φτάνει στην έξοδο όταν αναφερόμαστε στον κόμβο - φύλλο του λεκτικού δέντρου που αναπαριστά τον τελευταίο τρίφθογγό της. Η πρώτη καταχώρηση δεν είναι για λέξη αλλά για το <s>, τον ειδικό χαρακτήρα που συμβολίζει τα πλαίσια σιωπής που υπάρχουν στην έναρξη κάθε πρότασης.

Για τη διάδοση των σκορ σε επίπεδο λέξεων, ο αποκωδικοποιητής βρίσκει σε κάθε λέξη την πιο πρόσφατη καταχώρηση του πίνακα και από αυτή, τις καταχωρήσεις που έχουν εγγραφεί στο διάστημα [endframe, endframe+1]. Όπου endframe ο αριθμός του πλαισίου εγγραφής της πιο πρόσφατης καταχώρησης. Με βάση αυτές τις καταχωρήσεις και τις λέξεις που αντιπροσωπεύουν, βρίσκονται από το γλωσσικό μοντέλο τα τριγράμματα αυτών των συνδυασμών λέξεων. Τελικά σαν καταχώρηση στον πίνακα εισάγονται τα τρίγραμμα που δεν έχουν ξανακαταχωρηθεί σαν ΓΜ κατάσταση. Τα τριγράμματα που υπάρχουν σε κάποια παλιότερη καταχώρηση δε δημιουργούν καινούρια αλλά ενημερώνουν το σκορ της υπάρχουσας αν είναι καλύτερο.

Έξοδος αποτελεσμάτων

Στο τέλος κάθε πρότασης ο αποκωδικοποιητής παράγει την *υπόθεση* αναγνώρισης γι'αυτή την πρόταση. Για να δημιουργήσει την υπόθεση εξάγει από τον πίνακα που περιγράφηκε, την τελευταία καταχώρηση και ακολουθώντας το δείκτη προς την προηγούμενη καταχώρηση διανύει όλες τις καταχωρήσεις μέχρι το πρώτο πλαίσιο. Κρατώντας κάθε φορά τις καταχωρήσεις εκείνες που έχουν το μεγαλύτερο σκορ δημιουργείται μια ακολουθία λέξεων που προσεγγίζει περισσότερο τις λέξεις που έχουν όντως ειπωθεί και που αποτελεί το τελικό αποτέλεσμα της αναγνώρισης.

4.3.1 Η έννοια του κλαδέματος

Το κλάδεμα είναι ένας βασικός παράγοντας στη λειτουργία του αποκωδικοποιητή. Στόχος του είναι να μειώσει το χρόνο επεξεργασίας των προτάσεων και τους υπολογιστικούς πόρους βάζοντας συνθήκες σε διάφορα στάδια της αποκωδικοποίησης. Η πιο σημαντική εφαρμογή του είναι στον καθορισμό των ενεργών HMMs, δηλαδή των HMMs με τα οποία ο αποκωδικοποιητής συγκρίνει τα εισερχομενα πλαίσια φωνής. Υπάρχουν δύο είδη κλαδέματος:

- Κλάδεμα με δέσμη
  Κάθε πρόταση επεξεργάζεται ανά πλαίσιο. Σε κάθε πλαίσιο ο αποκωδικοποιητής έχει έναν αριθμό ενεργών HMMs τα οποία θα χρησιμοποιήσει για το Viterbi. Από αυτά θα απορρίψει όσα έχουν σκορ μέχρι εκείνη τη στιγμή το οποίο είναι μικρότερο από ένα κατώφλι. Το κατώφλι αυτό ισούται με το γινόμενο μιας τιμής δέσμης με το σκορ της καλύτερης κατάστασης εκείνη τη στιγμή. Η δέσμη παίρνει τιμές στο διάστημα [0,1] με το μηδέν να επιτρέπει σε όλα τα HMMs να είναι ενεργά και με το 1 να επιτρέπει μόνο στο HMM με το καλύτερο σκορ να είναι ενεργό[8].

  Παρόμοιο κλάδεμα γίνεται και σε άλλα στάδια της αποκωδικοποιήσης, όπως στην εισαγωγή καταχωρήσεων στον πίνακα των λέξεων όπου κριτήριο για την εισαγωγή είναι πάλι μία τιμή δέσμης.

- Απόλυτο κλάδεμα
  Ακόμα και με το κλάδεμα με δέσμη, ο αριθμός των ενεργών HMMs μπορεί να γίνει πολύ μεγάλος και χρονοβόρος. Όταν αυτός ο αριθμός περάσει κάποια όρια τότε μειώνονται οι πιθανότητες να αναγνωριστεί σωστά μια

λέξη εξαιτίας των πολλών υποψηφίων μοντέλων. Τέτοια περίπτωση μπορεί να υπάρξει όταν τα ηχητικά δείγματα προς αναγνώριση είναι θορυβώδη ή δεν αντιπροσωπεύονται επαρκώς από τα ακουστικά μοντέλα. Σε αυτές τις περιπτώσεις δεν υπάρχει νόημα να αφήνονται πολλά HMMs ενεργά γι'αυτό ο αποκωδικοποιητής παίρνει σαν παραμέτρους τιμές που περιορίζουν τον απόλυτο αριθμό των ενεργών HMMs ανά πάσα στιγμή.

Και τα δύο είδη κλαδέματος παραμετροποιούνται εμπειρικά και πειραματικά. Δεν υπάρχει συγκεκριμένος τρόπος για να καθορίσει κανείς τις τιμές που πρέπει να περάσει στον αποκωδικοποιητή και που αφορούν το κλάδεμα.

Κεφάλαιο 5

Παράλληλος προγραμματισμός με χρήση καρτών γραφικών

5.1 Παράλληλος υπολογισμός

Ο παράλληλος υπολογισμός είναι μία μορφή υπολογισμού στην οποία πολλές υπολογιστικές πράξεις εκτελούνται ταυτόχρονα[10]. Βασίζεται στο αξίωμα ότι μεγάλα προβλήματα μπορούν συχνά να διασπαστούν σε μικρότερα, τα οποία μπορούν να επιλυθούν παράλληλα. Υπάρχουν πολλές μορφές παράλληλου υπολογισμού, όπως σε επίπεδο bit, σε επίπεδο εντολών, σε επίπεδο δεδομένων και παραλληλισμό εργασιών. Ο παραλληλισμός χρησιμοποιείται πολλά χρόνια αλλά η χρήση του έχει αυξηθεί τον τελευταίο καιρό εξαιτιάς των φυσικών περιορισμών που παρουσιάζονται στις σύγχρονες αρχιτεκτονικές υπολογιστών. Ένας τέτοιος περιορισμός είναι η κατανάλωση ενέργειας και η έκλυση θερμότητας των επεξεργαστών που έχει ως αποτέλεσμα να μην μπορεί να αυξηθεί η συχνότητα λειτουργίας τους. Αυτός είναι και ο λόγος που ο παράλληλος υπολογισμός χρησιμοποιείται πλέον κατά κόρον στην αρχιτεκτονική υπολογιστών κυρίως με τη μορφή πολυπύρηνων επεξεργαστών.

Οι υπολογιστές που κάνουν χρήση παραλληλισμού μπορούν να διακριθούν με βάση το επίπεδο υλικού τους που υποστηρίζει παραλληλισμό. Υπάρχουν τα πολυπύρηνα και τα πολυεπεξεργαστικά υπολογιστικά συστήματα που έχουν πολλές μονάδες επεξεργασίας σε ένα μόνο μηχάνημα. Επίσης υπάρχουν τα πλέγματα και οι υπολογιστές μαζικού παραλληλισμού που χρησιμοποιούν πολλούς υπολογιστές για την ίδια εργασία.

Η ανάπτυξη εφαρμογών που εκμεταλλεύονται τον παραλληλισμό είναι πιο

δύσκολη από αυτή των ακολουθιακών προγραμμάτων. Η αιτία είναι ότι η ταυτόχρονη εκτέλεση εργασιών μπορεί να δημιουργήσει σφάλματα στην εκτέλεση του κώδικα και εσφαλμένα αποτελέσματα που οφείλονται κατά κύριο λόγο στις εξαρτήσεις των δεδομένων. Το μεγαλύτερο εμπόδιο στη βελτίωση της απόδοσης ενός παράλληλου προγράμματος είναι ο συγχρονισμός και η επικοινωνία μεταξύ των διαφορετικών εργασιών που εκτελούνται παράλληλα.

Για τη μέτρηση της βελτίωσης ενός προγράμματος που έχει υποστεί παραλληλοποίηση σε κάποιο κομμάτι του, είναι αναγκαία η χρήση του νόμου του Amdahl. Πρέπει όμως πρώτα να ορίσουμε την έννοια της επιτάχυνσης ενός αλγορίθμου, που δεν είναι άλλη από το πηλίκο του χρόνου σειριακής εκτέλεσης του αλγορίθμου προς το χρόνο εκτέλεσης της παράλληλης έκδοσης.

$$επιταχυνση = \frac{χρονοσειριακησεκτελεσης}{χρονοσπαραλληλησεκτελεσης} \tag{5.1}$$

Ο νόμος του Amdahl είναι ένα μοντέλο συσχέτισης της αναμενόμενης επιτάχυνσης ενός αλγορίθμου όταν αυτός παραλληλοποιείται, με τη σειριακή έκδοσή του με την προϋπόθεση ότι το μέγεθος του προβλήματος είναι το ίδιο και στις δύο περιπτώσεις. Σύμφωνα μ'αυτόν αν Π το ποσοστό του αλγορίθμου που μπορεί να παραλληλοποιηθεί, $(1-\Pi)$ το ποσοστό που παραμένει σειριακό και ο χρόνος πριν την παραλληλοποίηση είναι ίσος με 1 ανεξαρτήτου μονάδας χρόνου, τότε η μέγιστη επιτάχυνση που μπορεί να επιτευχθεί χρησιμοποιώντας N επεξεργαστές είναι:

$$\frac{1}{(1-\Pi)+\frac{\Pi}{N}} \tag{5.2}$$

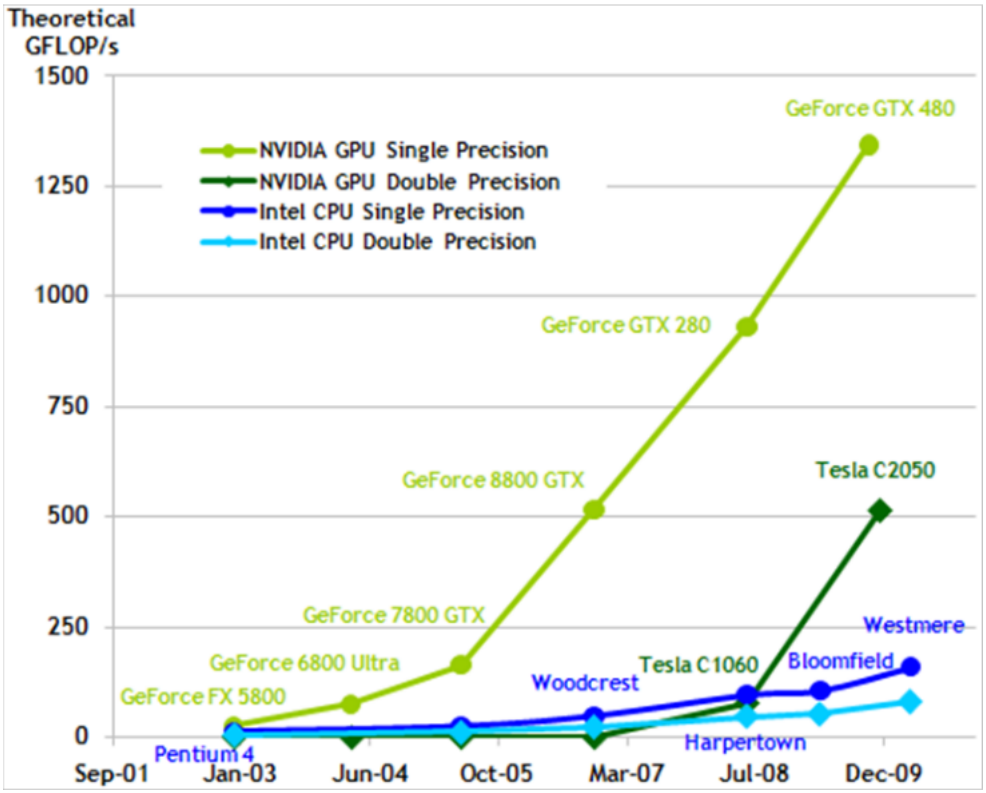
Όπως φαίνεται από τον τύπο, όταν ο αριθμός των επεξεργαστών αυξάνεται, στον παρονομαστή του κλάσματος επικρατεί ο όρος $(1-\Pi)$. Επομένως από ένα σημείο και μέτα όσους επεξεργαστές και αν χρησιμοποιήσει κανείς η επιτάχυνση δε θα αυξηθεί εξαιτίας του ποσοστού του αλγορίθμου που εκτελείται σειριακά. Ιδανικά, το ποσοστό αυτό πρέπει να είναι όσο το δυνατόν μικρότερο.

5.2 Η τεχνολογία GPGPU

Οι μονάδες επεξεργασίας γραφικών (GPU) είναι σχεδιασμένες με στόχο την ικανοποίηση των αναγκών της βιομηχανίας παχνιδιών. Τα γραφικά των παιχνιδιών

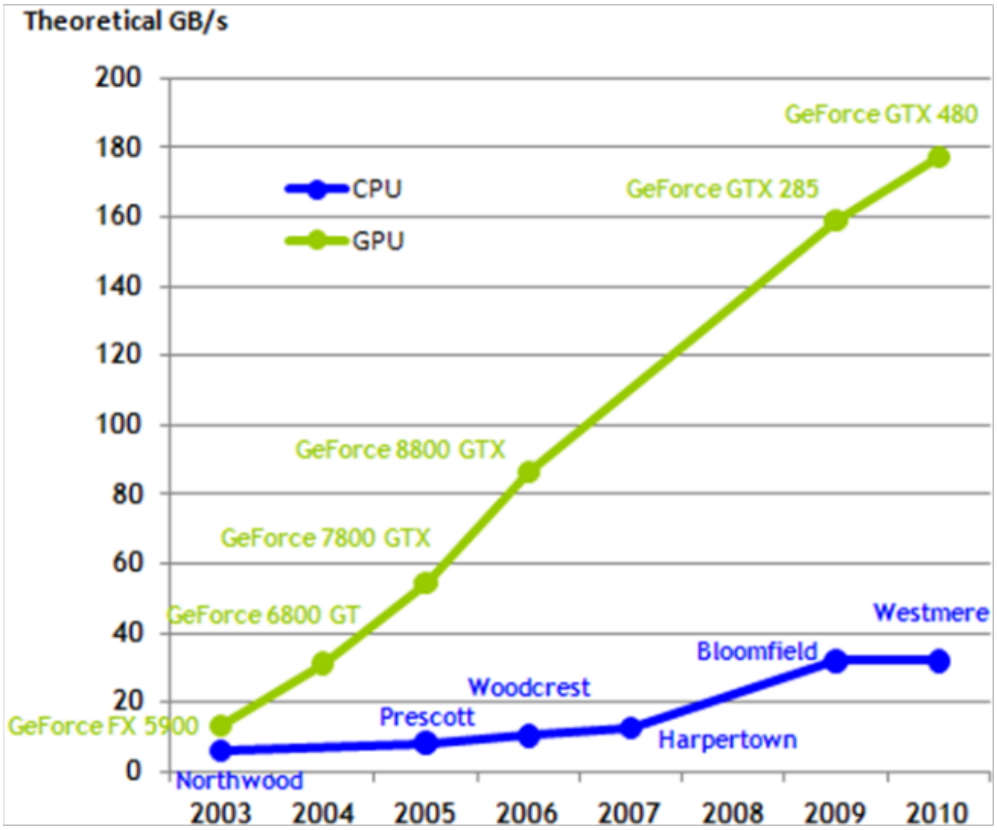
απαιτούν τον επαναλαμβανόμενο υπολογισμό της τιμής κάθε εικονοκυττάρου και γι'αυτό οι GPUs χρησιμοποιούν τα περισσότερα τρανζίστορς τους για επεξεργασία δεδομένων και αριθμητικές πράξεις. Πολύπλοκα συστήματα κρυφής μνήμης και ελεγκτές ροής απουσιάζουν γιατί δεν είναι απαραίτητα. Οι GPUs παρουσιάζουν μεγάλο βαθμό παραλληλίας σε επίπεδο υλικού καθώς κάθε πυρήνας επεξεργάζεται δεδομένα σαν ανεξάρτητο νήμα. Το γεγονός αυτό τις κάνει ιδανικές για διαχωρισμό δεδομένων προς επεξεργασία σε διαφορετικά νήματα με ελάχιστες εξαρτήσεις μεταξύ τους. Η μεγάλη υπολογιστική ισχύς που διαθέτουν είναι ικανή να βελτιώσει την απόδοση ενός κώδικα, αρκεί αυτός να πληρεί τις προϋποθέσεις, πέρα από τα όρια μιας CPU. Στον ίδιο τομέα συμβάλλει και η πολύ μεγάλη ταχύτητα μεταφοράς δεδομένων που διαθέτουν οι μνήμες των καρτών γραφικών. Στα σχήματα 5.1 και 5.2 μπορεί κανείς να δει την ανωτερότητα των καρτών γραφικών σε σχέση με τις CPUs στον αριθμό των πράξεων που μπορούν να εκτελέσουν ανά δευτερόλεπτο και στην ταχύτητα της μνήμης που διαθέτουν. Για τη σύγκριση έχουν χρησιμοποιηθεί προϊόντα των κορυφαίων εταιρειών κατασκευής GPU και CPU, της NVIDIA και της Intel αντίστοιχα.

Αυτή η μεγάλη υπολογιστική ισχύς των καρτών γραφικών έχει γίνει τον τελευταίο καιρό αντικείμενο εκμετάλλευσης από την επιστημονική κοινότητα με τη μορφή του *υπολογισμού γενικού σκοπού σε μονάδες επεξεργασίας γραφικών*. Ο όρος αυτός περιγράφει την τεχνική χρήσης μιας GPU για την πραγματοποίηση υπολογισμών σε εφαρμογές που παραδοσιακά αναλάμβανε η CPU. Η GPU χρησιμοποιείται σαν επιταχυντής σε συνδυασμό με τη CPU, δουλεύοντας σε επιλεγμένα κομμάτια του κώδικα που παρουσιάζουν μεγάλο βαθμό παραλληλίας. Τα δεδομένα δηλαδή που επεξεργάζονται είναι ανεξάρτητα ή έχουν μικρές εξαρτήσεις το ένα με το άλλο και η επεξεργασία που τους γίνεται είναι η ίδια. Εφαρμόζοντας αυτή τη τεχνική έχει επιτευχθεί η επιτάχυνση σε μεγάλο βαθμό προγραμμάτων που έχουν σχεδιαστεί αρχικά για τη CPU. Αξίζει ωστόσο να σημειωθεί ότι είναι ελάχιστες οι περιπτώσεις που το συντριπτικό ποσοστό ενός κώδικα μπορεί να εκτελεστεί στη GPU. Ένα μεγάλο ποσοστό συνεχίζει να τρέχει στη CPU γιατί η παραλληλοποίησή του δεν είναι εφικτή.



Σχήμα 5.1: Σύγκριση της υπολογιστικής ισχύος των GPUs και των CPUs με την πάροδο του χρόνου[11]

Στη διάδοση του *υπολογισμού γενικού σκοπού σε μονάδες γραφικών* βοήθησε η εξέλιξη των GPUs σε επίπεδο υλικού αλλά και η δημιουργία κατάλληλων προγραμματιστικών μοντέλων. Σε επίπεδο υλικού, αυξήθηκε η αριθμητική ακρίβεια κυρίως σε πράξεις κινητής υποδιαστολής. Σε επίπεδο διασύνδεσης με τον προγραμματιστή, οι εταιρείες NVIDIA και ATI με προεξέχοντα την πρώτη, παρουσίασαν ολοκληρωμένα προγραμματιστικά μοντέλα για τα προϊόντα τους. Τα μοντέλα αυτά ξεφεύγουν από τον καθιερωμένο προγραμματισμό καθαρά γραφικών εφαρμογών. Δίνουν τη δυνατότητα στον προγραμματιστή να υλοποιήσει αλγορίθμους που επιλύουν μαθηματικά προβλήματα κατά βάση, γράφωντας κώδικα που μπορεί να εκτελεστεί στην κάρτα γραφικών.



Σχήμα 5.2: Σύγκριση της ταχύτητας μεταφοράς δεδομένων σε μνήμες GPUs και CPUs με την πάροδο του χρόνου[11]

5.3 NVIDIA CUDA

Το 2008 η κατασκευάστρια εταιρεία καρτών γραφικών NVIDIA παρουσίασε στο κοινό το CUDA (Compute Unified Device Architecture). Το CUDA είναι μια αρχιτεκτονική παράλληλου υπολογισμού που υποστηρίζεται από τη NVIDIA στις κάρτες της σειράς G8x και τις νεότερες που περιλαμβάνουν αυτές των σειρών GeForce, Quadro και Tesla. Μπορεί να χρησιμοποιηθεί από τους προγραμματιστές μέσω διάφορων προγραμματιστικών γλωσσών όπως η Python, η Perl, η Fortran και η Java. Η γλώσσα ωστόσο που υποστηρίζεται επίσημα από την εταιρεία και που χρησιμοποιεί η πλειοψηφία των προγραμματιστών είναι η C for CUDA. Η τελευταία είναι μία διευρημένη έκδοση της C με εντολές που εξυπηρετούν το προγραμματιστικό μοντέλο του CUDA και τον προγραμματισμό μιας GPU.

5.3.1 Το προγραμματιστικό μοντέλο του CUDA

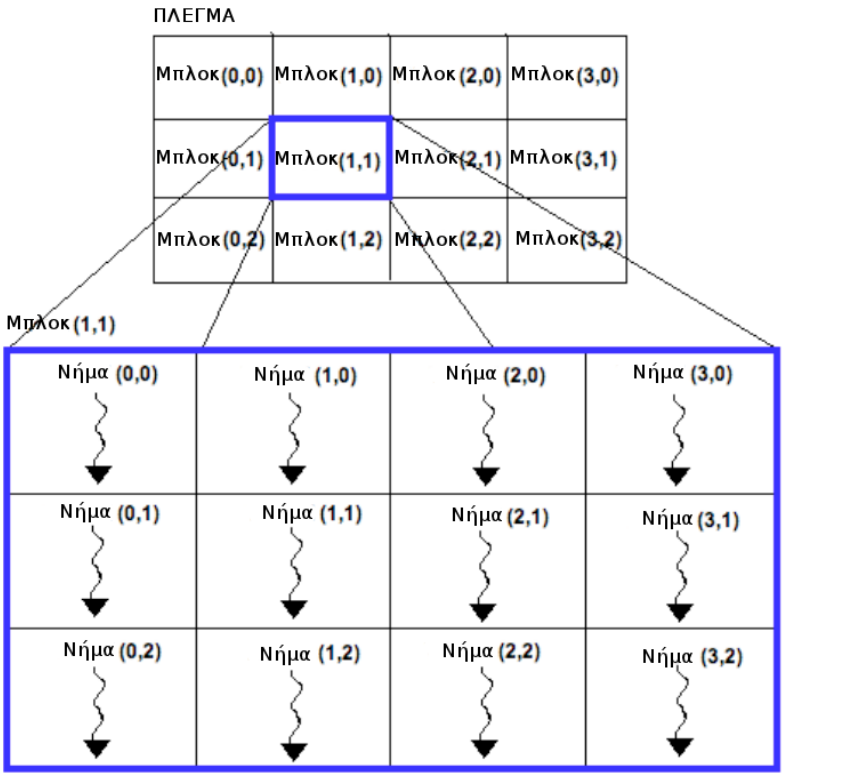
Η υλοποίηση αλγορίθμων που κάνουν χρήση του CUDA ακολουθεί ένα συγκεκριμένο πρότυπο. Σύμφωνα μ'αυτό, τα κομμάτια του κώδικα που δεν μπορούν να παραλληλοποιηθούν υλοποιούνται σαν τυπικά προγράμματα C που έχουν γραφτεί για να εκτελεστούν στη CPU ενώ τα κομμάτια του κώδικα που παρουσιάζουν μεγάλο βαθμό παραλληλισμού υλοποιούνται σαν πυρήνες. Ένας πυρήνας είναι μία συνάρτηση που εκτελείται στη GPU από κάθε νήμα που έχει δημιουργηθεί σ'αυτήν. Το σώμα της συνάρτησης είναι το ίδιο για όλα τα νήματα αλλά τα ορίσματα είναι διαφορετικά. Στην πιο απλή δομή ενός τέτοιου προγράμματος, υπάρχει μια κεντρική συνάρτηση όπως σε όλους τους κώδικες γραμμένους σε C. Αυτή μπορεί να χωριστεί σε 4 μέρη. Το πρώτο μέρος είναι αυτό το κομμάτι του προγράμματος που δεν παραλληλοποιείται και θα εκτελεστεί στη CPU. Το δεύτερο μέρος περιέχει τη δέσμευση μνήμης από τη διαθέσιμη μνήμη της GPU και τη μεταφορά δεδομένων από τη κύρια μνήμη του υπολογιστή σε αυτή. Το τρίτο μέρος είναι η κλήση του πυρήνα, ο οποίος θα επεξεργαστεί τα δεδομένα που έχουμε μεταφέρει στο προηγούμενο μέρος. Το τελευταίο μέρος είναι αυτό της μεταφοράς των αποτελεσμάτων που προέκυψαν από την εκτέλεση του πυρήνα, πίσω στην κύρια μνήμη. Ακολουθεί ένα απλό πρόγραμμα γραμμένο για CUDA το οποίο προσθέτει τα στοιχεία δύο διανυσμάτων και τα αποθηκεύει σε ένα τρίτο.

```
__global__ void VecAdd(int* A, int* B, int* C)
{
int i=blockDim.x * blockIdx.x + threadIdx.x;
if(i<N)
 C[i]=A[i]+B[i];
}

int main()
{
 int N=10000;

 int* h_A=(int*)malloc(N*sizeof(int));
 int* h_B=(int*)malloc(N*sizeof(int));

 int* d_A=cudaMalloc((void**)&d_A, N*sizeof(int));
 int* d_B=cudaMalloc((void**)&d_B, N*sizeof(int));
 int* d_C=cudaMalloc((void**)&d_C, N*sizeof(int));
```

```
cudaMemcpy(d_A,h_A, N*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpy(d_B,h_B, N*sizeof(int),cudaMemcpyHostToDevice);

 int threadsPerBlock=256;
int blocksPerGrid=(N+threadsPerBlock-1)/threadsPerBlock;

VecAdd<<<blocksPerGrid,threadsPerBlock>>>(d_A,d_B,d_C);

cudaMemcpy(h_C,d_C,N*sizeof(int),cudaMemcpyDeviceToHost);

 cudaFree(d_A);
 cudaFree(d_B);
 cudaFree(d_C);
}
```

Για να δηλώσει κανείς έναν πυρήνα, πρέπει να χρησιμοποιήσει τη δεσμευμένη λέξη __global__ . Για να ορίσει τον αριθμό των νημάτων για κάθε κλήση, γίνεται χρήση της σύνταξης <<<dimGrid,dimBlock>>>. Τα νήματα οργανώνονται σε μπλοκς και τα το σύνολο των μπλοκς αποτελούν το πλέγμα. Αυτή η οργάνωση των νημάτων παρουσιάζεται στο σχήμα 5.3 . Η επιλογή των παραμέτρων dimGrid και dimBlock απαιτεί πειραματισμό για την πλήρη εκμετάλλευση των δυνατοτήτων της κάρτας γραφικών. Παρόλ'αυτά υπάρχουν κάποιες βασικές αρχές που μπορεί να ακολουθήσει κανείς για να επιλέξει τις κατάλληλες τιμές. Ο αριθμός των μπλοκ σε ένα πλέγμα πρέπει να είναι αρκετά μεγάλος ώστε κάθε πολυεπεξεργαστής της κάρτας γραφικών να έχει τουλάχιστον ένα μπλοκ να επεξεργαστεί. Τα νήματα προγραμματίζονται να εκτελεστούν ανά ομάδες νημάτων. Κάθε τέτοια ομάδα αποτελείται από 32 νήματα και η μέγιστη απόδοση στην εκτέλεσή τους προκύπτει όταν και τα 32 ακολουθούν το ίδιο μονοπάτι εκτέλεσης. Οι ομάδες των νημάτων σε ένα μπλοκ πρέπει να είναι όσο το δυνατόν περισσότερες. Αυτό είναι επιθυμητό γιατί όταν τα νήματα μιας ομάδας αιτούνται δεδομένα από τις περιοχές μνήμης, τότε ο πολυεπεξεργαστής πρέπει να έχει τη δυνατότητα να εκτελέσει κάποια άλλη ομάδα νημάτων ώστε να μην παραμείνει ανενεργός όσο διάστημα διαρκεί η μεταφορά των δεδομένων από τη μνήμη. Πρέπει να σημειωθεί ότι οι ομάδες των νημάτων πρέπει να παρουσιάζουν πλήρη ανεξαρτησία δεδομένων μεταξύ τους. Ένας ακόμα παράγοντας στην επιλογή των παραμέτρων είναι η κατοχή. Η κατοχή είναι μία ένδειξη του ποσοστού χρήσης ενός πολυεπεξεργα-

στή και ορίζεται ως ο αριθμός των ομάδων νημάτων που έχουν ανατεθεί σε ένα πολυεπεξεργαστή, προς το μέγιστο αριθμό που μπορεί να του ανατεθεί. Αυτή τη στιγμή οι πιο σύγχρονες κάρτες γραφικών της NVIDIA υποστηρίζουν μέχρι 48 ομάδες νημάτων ανά πολυεπεξεργαστή, 1024 νήματα ανά μπλοκ και 1536 νήματα ανά πολυεπεξεργαστή.



Σχήμα 5.3: Η ιεραρχία των νημάτων στην αρχιτεκτονική του CUDA

Ενσωματωμένες μεταβλητές

Οι διαστάσεις του πλέγματος καθορίζονται από την πρώτη παράμετρο στην κλήση ενός πυρήνα, dimGrid, ενώ ο αριθμός των νημάτων ανά μπλοκ από τη δεύτερη παράμετρο, dimBlock. Το CUDA παρέχει μεταβλητές που διευκολύνουν

την προσπέλαση των νημάτων μέσα σε ένα μπλοκ και μέσα στο πλέγμα. Η μεταβλητή threadIdx είναι ένα τρισδιάστατο διάνυσμα που επιτρέπει την προσπέλαση των νημάτων μέσα σε ένα μπλοκ. Ένα μπλοκ μπορεί να έχει έως και τρεις διαστάσεις και ένα νήμα που ανήκει σ'αυτό χαρακτηρίζεται από τις μεταβλητές threadIdx.x , threadIdx.y και threadIdx.z . Η μεταβλητή blockIdx είναι ένα διάνυσμα 2 διαστάσεων, που δείχνει τη θέση ενός μπλοκ μέσα στο πλέγμα. Παρόμοια με τη threadIdx, έχει δύο δείκτες, blockIdx.x και blockIdx.y . Οι διαστάσεις ενός μπλοκ μπορούν να προσπελάστούν με τη μεταβλητή blockDim που είναι επίσης δύο διαστάσεων.

Χώροι μνήμης

Στο CUDA διακρίνονται τρείς χώροι μνήμης:

- Καθολική μνήμη
  Η καθολική μνήμη έχει πολύ μεγαλύτερη καθυστέρηση και χαμηλότερη ταχύτητα μεταφοράς δεδομένων σε σύγκριση με τους υπόλοιπους χώρους μνήμης στη GPU. Στην καθολική μνήμη βρίσκονται οι καθολικές μεταβλητές. Προαιρετικά όταν δηλώνονται, φέρουν μπροστά τη δεσμευμένη λέξη __device__ . Δημιουργούνται στο κομμάτι του κώδικα που τρέχει στη CPU. Οι βασικότερες συναρτήσεις που αφορούν την καθολική μνήμη και τον χειρισμό της, είναι η cudaMalloc και η cudaMemcpy. Με την πρώτη μπορεί κανείς να δεσμεύσει μνήμη για καθολικές μεταβλητές ενώ με τη δεύτερη μπορεί να μεταφέρει μπλοκ μνήμης από τη κεντρική μνήμη σε αυτή της GPU και το αντίστροφο. Οι καθολικές μεταβλητές είναι προσπελάσιμες από όλα τα νήματα και η διαρκούν μέχρι να απελευθερωθούν με τη cudaFree. Μπορούν έτσι να χρησιμοποιηθούν και απο διαφορετικούς πυρήνες ή σε πολλές κλήσεις του ίδιου.

- Κοινόχρηστη μνήμη
  Η καθολική μνήμη υπάρχει στη DRAM μιας GPU. Αντιθέτως η κοινόχρηστη μνήμη είναι στο τσιπ της. Κάθε πολυεπεξεργαστής της GPU έχει ένα μέγεθος κοινόχρηστης μνήμης, το οποίο μοιράζει ισόποσα στα μπλοκς νημάτων που έχουν προγραμματιστεί να εκτελεστούν σ'αυτόν. Το μέγεθος αυτό στις κάρτες της NVIDIA είναι αυτή τη στιγμή ίσο με 48KB. Η προσπέλαση των δεδομένων στη κοινόχρηστη μνήμη είναι πολύ ταχύτερη απ'ότι στην καθολική μνήμη. Χωρίζεται δε, σε μονάδες μνήμης ίδιου μεγέθους, οι οποίες ονομάζονται τράπεζες και μπορούν να προσπελαστούν ταυτόχρονα. Οι προσπελάσεις ωστόσο μπορούν να γίνουν ταυτόχρονα όταν ν αριθμός

διευθύνσεων μνήμης αντιστοιχούν σε ν διαφορετικές διευθύνσεις τραπεζών. Αν δύο ή παραπάνω διευθύνσεις αντιστοιχούν στην ίδια τράπεζα τότε αυτές οι προσπελάσεις θα σειριοποιηθούν. Οι κοινόχρηστες μεταβλητές δηλώνονται με τη δεσμευμένη λεξη __shared__ και είναι διαθέσιμες σε όλα τα νήματα του μπλοκ μέσα στο οποίο έχουν δηλωθεί. Η διάρκεια ζωής τους είναι ίση με το χρόνο διάρκειας του μπλοκ. Δε διατηρούνται μετά το τέλος της εκτέλεσης του πυρήνα που τις έχει δημιουργήσει.

- Καταχωρητές
  Οι καταχωρητές είναι μία ακόμα μνήμη που βρίσκεται στο τσιπ και έχει πολύ χαμηλή καθυστέρηση. Κάθε πολυεπεξεργαστής έχει έναν αριθμό από καταχωρητές που δίνονται στα νήματα. Σε αντίθεση με την κοινόχρηστη μνήμη που ανήκει σε όλα τα νήματα ενός μπλοκ, οι καταχωρητές ανήκουν αποκλειστικά στο κάθε νήμα. Οι μεταβλητές που μπαίνουν σε καταχωρητές δηλώνονται μέσα στον πυρήνα και δε φέρουν καμιά λέξη σαν χαρακτηριστικό μπροστά τους. Επειδή ο αριθμός των καταχωρητών είναι περιορισμένος, ενδέχεται κάποιες μεταβλητές να μη μπορούν να μπουν σ'αυτούς και ο μεταγλωττιστής να τις τοποθετήσει στην τοπική μνήμη. Η τοπική μνήμη είναι μια περιοχή της καθολικής μνήμης που δεσμεύεται για κάθε νήμα και προφανώς είναι το ίδιο αργή με την καθολική μνήμη.

Κεφάλαιο 6

Επιτάχυνση του Sphinx3 με χρήση της αρχιτεκτονικής CUDA

6.1 Σχετική εργασία

Μέχρι την εκπόνηση της παρούσας εργασίας, δεν είχε παρουσιαστεί, τουλάχιστον σε ακαδημαϊκό επίπεδο, κάποια εργασία ή κάποια δημοσίευση που να αφορά την επιτάχυνση του Sphinx3 με χρήση πολυεπεξεργαστικής κάρτας γραφικών. Ωστόσο, στην ηλεκτρονική σελίδα του έργου[1], υπάρχει ενημέρωση ότι γίνεται προσπάθεια για τη μεταφορά μερών του Sphinx3 στην αρχιτεκτονική του CUDA. Πρέπει να σημειωθεί όμως ότι τα μέρη αυτά αφορούν μόνο το SphinxTrain, δηλαδή τον αλγόριθμο που είναι υπεύθυνος για την εκπαίδευση των ακουστικών μοντέλων. Δεν υπάρχει καμία αναφορά στον αποκωδικοποιητή, με τον οποίο και καταπιάνεται η δική μας εργασία. Επίσης έχει παρουσιαστεί μία δημοσίευση με τίτλο "Three-Layer Optimizations for Fast GMM Computations on GPU-like Parallel Processors" από τους Kshitij Gupta και Hohn D. Owens, η οποία αν και σχετίζεται άμεσα με την παρούσα διπλωματική ως προς το σκοπό, δεν έχει κανένα πρακτικό ενδιαφέρον αφού παρουσιάζει αλλαγές στην εύρεση των σκορ για τα senones που ενδεχομένως αν υλοποιηθούν σε παράλληλη αρχιτεκτονική θα επιφέρουν βελτίωση στο χρόνο. Δεν έχει γίνει παρά ταύτα καμία υλοποίηση σε GPU και οι συγγραφείς αναφέρουν ότι αυτό είναι αντικείμενο μελλοντικής δημοσίευσης.

[1] http://cmusphinx.sourceforge.net/wiki/longaudiotraining/

6.2 Χρονική ανάλυση του αλγορίθμου

Εξαιτίας του πολύ μεγάλου μεγέθους της υλοποίησης του αποκωδικοποιητή s3.X, είναι αναγκαίο να γίνει η χρονική του ανάλυση. Το αποτέλεσμα αυτής θα μας υποδείξει ποια κομμάτια της υλοποίησης είναι αυτά που καταναλώνουν τον περισσότερο χρόνο για την εκτέλεσή τους. Αυτά τα κομμάτια θα κρίνουμε αν μπορούν να παραλληλοποιηθούν και να μεταφερθούν στην αρχιτεκτονική του CUDA.

Το Sphinx παρέχει δύο ηχητικές βάσεις δεδομένων. Με αυτές μπορεί κανείς να κάνει την εκπαίδευση των μοντέλων και στη συνέχεια να κάνει την αποκωδικοποίηση των προτάσεων που περιέχουν. Οι βάσεις αυτές είναι η AN4 και η RM1. Στην παρούσα εργασία επιλέχθηκε η RM1 επειδή είναι μεγαλύτερη σαν βάση και αυτό βοηθά σε καλύτερες επιδόσεις του αλγορίθμου. Η συγκεκριμένη βάση δεν περιέχει τα ηχητικά δείγματα στην αρχική τους μορφή εξαιτίας αδειών χρήσης αλλά περιέχει τη φασματική απεικόνισή τους. Περιέχει συνολικά 600 προτάσεις.

Για τη χρονική ανάλυση του κώδικα του αποκωδικοποιητή χρησιμοποιήθηκε ο χρονικός αναλυτής της Intel, Vtune Amplifier XE. Η CPU αναφοράς είναι αυτή του εξυπηρετητή Iraklis του εργαστηρίου MHL του Πολυτεχνείου Κρήτης. Πρόκειται για 2 τετραπύρηνους επεξεργαστές χρονισμένους στα 2,66GHz. Οι παράμετροι του αποκωδικοποιητή ήταν οι προκαθορισμένοι από τους προγραμματιστές του Sphinx και αυτοί που έχουν επιλεχθεί από τους ίδιους ως κατάλληλοι για τη βάση RM1. Από την εκτέλεση του αναλυτή προκύπτει ότι για την αναγνώριση όλων των προτάσεων απαιτούνται 220,928 δευτερόλεπτα από τη CPU. Ο χρόνος αυτός δεν περιέχει τον επιπλέον χρόνο που μπορεί να προστίθεται εξαιτίας διεργασιών του λειτουργικού που τρέχουν ταυτόχρονα. Είναι ο χρόνος για την αποκλειστική εκτέλεση του αποκωδικοποιητή. Για τη διευκόλυνσή μας, επιλέχθηκε από το σύνολο των προτάσεων του RM1 μία τυχαία και προχωρήσαμε στη χρονική ανάλυση της αναγνώρισης αυτής της πρότασης. Η πρόταση αυτή αποτελείται από 362 πλαίσια και αντιστοιχεί στην ηχογραφημένη πρόταση "GET ME THE LAST DATE ON FLASHER'S CAT-4 EQUIPMENT CASUALTY REPORTS". Τα αποτελέσματα από το χρονικό αναλυτή είναι τα παρακάτω

Συνάρτηση	χρόνος CPU(δευτερόλεπτα)
mgau_eval	0,292
mdef_init	0,040
parse_tmat_senmap	0,066
mgau_precomp	0,025
υπόλοιπες	0,135

Ο αναλυτής μας δείχνει τις τέσσερις συναρτήσεις που καταναλώνουν τον περισσότερο χρόνο και το χρόνο που καταναλώνουν όλες οι υπόλοιπες. Ο χρόνος κάθε συνάρτησης είναι το άθροισμα των χρόνων από όλες τις κλήσεις της. Το άθροισμα των χρόνων όλων των συναρτήσεων είναι ο συνολικός χρόνος για την αποκωδικοποίση της πρότασης. Αξίζει να σημειωθεί ότι για την εκτέλεση του αποκωδικοποιητή δε χρησιμοποιήθηκαν παραπάνω από δύο πυρήνες, εκ των οποίων ο ένας είχε πολύ μικρό ποσοστό συμμετοχής. Το πρόγραμμα ήταν ελεύθερο να χρησιμοποιήσει ακόμα και τους οκτώ πυρήνες.

Οι συναρτήσεις mdef_init και parse_tmat_senmap υλοποιούν την ανάγνωση από το σκληρό δίσκο του ακουστικού μοντέλου που παρέχεται σαν είσοδος στον αλγόριθμο και δημιουργούν τις απαραίτητες δομές για την αποθήκευση των senones. Εξαιτίας της ανάγνωσης από το σκληρό δίσκο, οι συναρτήσεις αυτές είναι αδύνατο να μεταφερθούν στην αρχιτεκτονική του CUDA υπό τη μορφή κώδικα που θα εκτελεστεί από την κάρτα γραφικών. Κάτι τέτοιο δεν υποστηρίζεται.

Στις υπόλοιπες συναρτήσεις περιλαμβάνονται όλες οι εναπομείναντες συναρτήσεις των οποίων ο αριθμός είναι πολύ μεγάλος. Αυτό έχει σαν συνέπεια, ελάχιστες να ξεπερνάνε τα 10ms σαν χρόνο εκτέλεσης. Γίνεται αντιληπτό ότι αυτές οι συναρτήσεις είναι μάταιο να μεταφερθούν στο CUDA. Η επιτάχυνση που μπορεί να επιτύχει κανείς θα είναι πολύ μικρή γιατί πρέπει να συνυπολογιστεί ο χρόνος που θα ξοδευτεί για τη μεταφορά των δεδομένων από την κύρια μνήμη στη μνήμη της GPU και αντίστροφα. Ακόμα όμως και να ήθελε κάποιος να το δοκιμάσει, θα διαπίστωνε ότι κάτι τέτοιο δεν είναι εφικτό. Οι λόγοι είναι δύο. Ο ένας είναι η ανάγνωση από το σκληρό δίσκο. Κάποιες από τις υπόλοιπες συναρτήσεις περιέχουν διάβασμα των εισόδων του αποκωδικοποιητή, οι οποίες είναι αρχεία στο σκληρό δίσκο. Ο δεύτερος λόγος είναι αλγοριθμικός. Από το κεφάλαιο 4 όπου έγινε η ανάλυση της λειτουργίας του αποκωδικοποιητή μπορεί να διαπιστώσει κανείς ότι υπάρχουν μέρη του αλγορίθμου που δεν μπορούν να παραλληλοποιηθούν. Η εύρεση του καλύτερου μονοπατιού με χρήση Viterbi δεν μπορεί να γίνει παράλληλα για όλα τα πλαίσια. Όπως έχει αναφερθεί, τα σκορ που προκύπτουν από το Viterbi για κάθε πλαίσιο εξαρτώνται από αυτά του προηγούμενου πλαισίου. Επομένως δεν είναι δυνατόν να παραλληλοποιηθεί η εύρεση του καλύτερου μονοπατιού σε επίπεδο πλαισίων. Στη διαδικασία αυτή περιλαμβάνονται και οι συναρτήσεις που υλοποιούν τη διάδοση των σκορ από τον ένα κόμβο του λεκτικού δέντρου στο άλλο και από τη μία λέξη στην άλλη. Κατά συνέπεια και αυτές δεν μπορούν να παραλληλοποιηθούν. Καταλήγουμε στο συμπέρασμα ότι οι συναρτήσεις που μπορούν να μεταφερθούν στην αρχιτεκτονική του CUDA είναι η mgau_eval και η mgau_precomp . Οι συναρτήσεις αυτές

χρησιμοποιούν τον τύπο 4.1 για να βρουν τα σκορ των GMM για κάθε πλαίσιο. Σύμφωνα με το χρονικό αναλυτή, χρειάζονται 0,317 δευτερόλεπτα, δηλαδή το άθροισμα των χρόνων των δύο συναρτήσεων, για να βρεθούν όλα τα σκορ για όλα τα πλαίσια.

6.3 Αλγόριθμος εύρεσης σκορ

Από την έκδοση X=5 και μετά, η διαδικασία εύρεσης των σκορ για τα GMM είναι η ακόλουθη:

```
Για όλα τα πλαίσια
  Για όλα τα senones
    Αν είναι Α.Σ. senone
      τότε υπολόγισε το σκορ
    Διαφορετικά αν είναι Ε.Σ. senone
      Αν η δέσμη για το κλάδεμα των Α.Σ. δεν έχει ορισθεί
       τότε υπολόγισε το σκορ
      Διαφορετικά
        Αν ο πατέρας του Ε.Σ έχει σκορ μικρότερο της δέσμης
          τότε υπολόγισε το σκορ
        Αν ο πατέρας του Ε.Σ έχει σκορ μεγαλύτερο της δέσμης
          Αν υπάρχει ο καλύτερος δείκτης από το προηγούμενο πλαίσιο
            τότε υπολόγισε το καλύτερο σκορ με χρήση του καλύτερου δείκτη
          Διαφορετικά
            χρησιμοποίησε το σκορ του πατέρα
```

Ο παραπάνω αλγόριθμος εισάγει στοιχεία που δεν έχουν αναφερθεί έως τώρα και είναι απαραίτητο να διευκρινιστούν για να γίνει κατανοητή η περαιτέρω προσέγγιση στην παραλληλοποίηση της εύρεσης των σκορ. Καταρχάς, με τους όρους Α.Σ. senone και Ε.Σ. senone αναφερόμαστε στα *ανεξαρτήτων συμφραζομένων* και *εξαρτημένων από συμφραζόμενα* senones. Η έννοια των συμφραζομένων έχει ορισθεί ήδη για τους τρίφθογγους. Ακριβώς τα ίδια ισχύουν για τα senones. Το κεντρικό senone σε ένα HMM έχει την προηγούμενη κατάσταση του μοντέλου σαν αριστερά συμφραζόμενα και τη δεξιά σαν δεξιά συμφραζόμενα. Δεύτερον, κάθε Ε.Σ. senone έχει ένα Α.Σ. senone σαν πατέρα. Για να αντιληφθεί κανείς τον όρο πατέρα, θα πρέπει να γνωρίζει ότι κατά τη διαδικασία της εκπαίδευσης των μοντέλων, κατασκευάζονται για κάθε senone δεντρικές δομές. Στις ρίζες αυτών

των δέντρων βρίσκονται τα senones των βασικών φθόγγων και στους κόμβους παιδιά, τα senones με συμφραζόμενα που συγκροτούν τους υπόλοιπους τρίφθογγους. Τρίτον, σε μια προσπάθεια βελτίωσης της ταχύτητας του αλγορίθμου, οι προγραμματιστές του Sphinx άρχισαν να κάνουν χρήση του καλύτερου δείκτη. Ο δείκτης αντιστοιχεί σε μία από τις συνιστώσες ενός GMM. Σε κάθε πλαίσιο αντί να χρησιμοποιούνται όλες οι συνιστώσες, να βρίσκονται όλα τα επιμέρους σκορ και να αθροίζονται, χρησιμοποιείται μόνο μία. Αυτή είναι ξεχωριστή για κάθε μοντέλο και είναι αυτή που στο προηγούμενο πλαίσιο είχε το καλύτερο επιμέρους σκορ. Με αυτόν τον τρόπο αποφεύγονται οι υπολογισμοί των υπολοίπων επιμέρους σκορ και κατά συνέπεια μειώνεται ο χρόνος αποκωδικοποίησης της πρότασης.

Η μέθοδος που περιγράφηκε παραπάνω παρουσιάζει δυσκολίες στην παραλληλοποίηση. Συγκεκρίμενα, γίνεται χρήση δέσμης και κλαδέματος για να περιοριστεί ο αριθμός των GMM που συγκρίνονται με το τρέχον πλαίσιο. Επειδή ο αποκωδικοποιητής δεν παίρνει σαν είσοδο κάποια τιμή που να καθορίζει άμεσα το κλάδεμα των GMM, αυτό γίνεται μέσω του κλαδέματος των HMM όπως περιγράφεται στην παράγραφο 4.3.1 . Κατ'αυτό τον τρόπο, αν ένα HMM δεν είναι ενεργό, τότε και τα GMM που το απαρτίζουν δεν είναι ενεργά και δε θα βρεθούν τα σκορ γι'αυτά με την παραπάνω διαδικασία. Το κλάδεμα όμως εξαρτάται από το σκορ της καλύτερης κατάστασης, δηλαδή από τα σκορ που έχουν βρεθεί στα προηγούμενα πλαίσια. Το γεγονός αυτό δε μας επιτρέπει να βρούμε παράλληλα τα σκορ για όλα τα μοντέλα και όλα τα πλαίσια χρησιμοποιώντας κλάδεμα. Μας αναγκάζει να χρησιμοποιήσουμε όλα τα μοντέλα χωρίς μάλιστα να κάνουμε διαχωρισμό με βάση το αν έχουν συμφραζόμενα ή όχι. Επίσης, δεν μπορεί να υλοποιηθεί στην αρχιτεκτονική του CUDA η βελτίωση με τον καλύτερο δείκτη με παραλληλοποίηση σε επίπεδο πλαισίων, Όπως αναφέραμε, ο καλύτερος δείκτης για ένα μοντέλο σε ένα πλαίσιο υπολογίζεται στο προηγούμενο πλαίσιο. Άρα δεν μπορούμε να βρούμε παράλληλα το σκορ για ένα μοντέλο σε όλα τα πλαίσια με χρήση του καλύτερου δείκτη. Πρέπει να σημειωθεί ότι η βελτίωση αυτή δεν έχει αποτυπωθεί στα αποτελέσματα του χρονικού αναλυτή γιατί δεν εφαρμόζεται εξ'ορισμού.

Καταλήγουμε έτσι στο συμπέρασμα ότι για μία δοθείσα πρόταση, θα υπολογίσουμε παράλληλα τα σκορ όλων των GMM για όλα τα πλαίσια. Στη συνέχεια, τα σκορ αυτά θα τροφοδοτούνται στο Viterbi ανά πλαίσιο χωρίς φυσικά να τα υπολογίσουμε ξανά.

Όπως φαίνεται στον τύπο 4.1, υπάρχει ένα κομμάτι που είναι ανεξάρτητο του διανύσματος χαρακτηριστικών, άρα και του πλαισίου. Το κομμάτι αυτό είναι το

$$\frac{1}{\sqrt{2\pi^{\nu} det(\sigma^2)}} \tag{6.1}$$

καθώς και το γινόμενο $1/2\sigma^2[i]$. Ο υπολογισμός αυτών των δύο γίνεται από τη συνάρτηση mgau_precomp μόνο μία φορά. Επειδή εξαρτώνται μόνο από το ακουστικό μοντέλο το οποίο δεν αλλάζει κατά τη διάρκεια εκτέλεσης του αποκωδικοποιητή, δε χρειάζεται να υπολογιστούν ξανά ακόμα και από πρόταση σε πρόταση. Το υπόλοιπο κομμάτι του τύπου 4.1 υλοποιείται από τη mgau_eval. Θα μεταφέρουμε και τις δύο συναρτήσεις στο CUDA ξεκινώντας από τη mgau_eval.

6.4 Παραλληλοποίηση του αλγορίθμου εύρεσης σκορ

6.4.1 Η συνάρτηση mgau_eval

Η συνάρτηση αυτή βρίσκει τα επιμέρους σκορ για όλες τις συνιστώσες ενός GMM και μετά τα αθροίζει για να προκύψει το συνολικό σκορ. Έχει ήδη αναφερθεί στη λειτουργία του αποκωδικοποιητή ότι οι τιμές των σκορ διατηρούνται στο λογαριθμικό τομέα και σε ακέραιες μεταβλητές. Ωστόσο, το γεγονός αυτό δημιουργεί ένα πρόβλημα όταν αθροίζονται δύο τιμές. Το πρόβλημα αυτό λύνεται με ένα πίνακα αναζήτησης. Ισχύει ότι:

$$b^z = b^x + b^y$$
$$b^z = b^x(1 + b^{y-x}) = b^y(1 + b^{x-y})$$
$$z = x + log_b(1 + b^{y-x}) = y + log_b(1 + b^{x-y})$$

όπου b η βάση του λογαρίθμου που έχει επιλεχθεί. Διακρίνονται έτσι οι εξής περιπτώσεις:

$$\text{Αν } y > x \text{ τότε } z = y + logadd\_table[-(x-y)]$$
$$\text{Αν } x > y \text{ τότε } z = x + logadd\_table[-(y-x)]$$
$$\text{όπου } logadd\_table[n] = log_b(1 + b^{-n})$$

με logadd_table ονομάζουμε τον πίνακα αναζήτησης. Ο πίνακας αυτός μπορεί να είναι πολύ μεγάλος και περιέχει τουλάχιστον 256 στοιχεία. Το πρώτο στοιχείο είναι το $log_b(2.0)$ για την περίπτωση που $y = x$ και συνεπώς $z = log_b(2) + x$. Το τελευταίο στοιχείο είναι το μηδέν για την περίπτωση που έχουμε απώλεια ακρίβειας οπότε επιστρέφεται το μεγαλύτερο εκ των x και y.

6.5 Πρώτη υλοποίηση

Καταρχάς πρέπει να μεταφέρουμε τα απαραίτητα δεδομένα από την κύρια μνήμη στη μνήμη της κάρτας γραφικών. Τα δεδομένα αυτά είναι τα χαρακτηριστικά των μοντέλων. Συγκεκριμένα, τα διανύσματα των μέσων τιμών, των διασπορών και των βαρών των συνιστωσών. Επίσης πρέπει να μεταφερθούν στη GPU τα διανύσματα των χαρακτηριστικών των πλαισίων, ο λογαριθμικός πίνακας αναζήτησης καθώς και το πρώτο κομμάτι του τύπου 4.1 που έχει υπολογισθεί από τη συνάρτηση mgau_precomp.

Το επόμενο βήμα είναι να καθορίσουμε πόσα νήματα θα χρησιμοποιήσουμε και ποιο το έργο αυτών. Σ'αυτή την πρώτη προσέγγιση, θα χρησιμοποιήσουμε τόσα νήματα όσα είναι τα πλαίσια της πρότασης που θέλουμε να αναγνωρίσουμε. Κάθε νήμα είναι υπεύθυνο για τον υπολογισμό όλων των σκορ από τα διαθέσιμα GMM. Αυτό σημαίνει ότι κάθε νήμα θα υπολογίσει τα επιμέρους σκορ από ένα μοντέλο σε σχέση με ένα συγκεκριμένο πλαίσιο, θα τα αθροίσει για να βρει το συνολικό σκορ του μοντέλου αυτού και στη συνέχεια θα πράξει το ίδιο για τα υπόλοιπα μοντέλα. Χρησιμοποιώντας για παράδειγμα την ίδια πρόταση που χρησιμοποιήσαμε στη χρονική ανάλυση του αλγορίθμου, η υλοποίηση μας ορίζει 362 νήματα, όσα και τα πλαίσια της πρότασης. Τα μοντέλα που παρέχονται με το ακουστικό μοντέλο είναι 1135 και οι συνιστώσες τους είναι 8. Επομένως 362 νήματα θα υπολογίσουν από 1135 σκορ το καθένα. Τα δεδομένα μας είναι αποθηκευμένα μόνο στο χώρο της καθολικής μνήμης και θέσαμε τα νήματα ανά μπλοκ ίσα με 128.

Για την χρονική ανάλυση της υλοποίησής μας χρησιμοποιήσαμε τον αναλυτή που παρέχει η NVIDIA για την αρχιτεκτονική του CUDA, Compute Visual Profiler. Η δε κάρτα γραφικών είναι η GTX280. Τα αποτελέσματα από τον αναλυτή ήταν τα ακόλουθα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
1,32 δευτ.	0,375	3,72Gb/δευτ.

Για να συγκρίνουμε την υλοποίηση αυτή με την υλοποίηση στη CPU, πρέπει να δούμε πρώτα πόσο χρόνο χρειάζεται η mgau_eval στη CPU όταν χρησιμοποιεί όλα τα μοντέλα. Για να γίνει αυτό, θέσαμε στην είσοδο του αποκωδικοποιητή την παράμετρο της δέσμης ίση με μηδέν. Από το χρονικό αναλυτή προέκυψε ότι η αναγνώριση της πρότασης διήρκεσε συνολικά 0,882 δευτερόλεπτα και η mgau_eval 0,436 δευτερόλεπτα. Στο χρόνο αυτό πρέπει να προσθέσουμε και το χρόνο της logmath_add, της συνάρτησης δηλαδή που κάνει τις προσπελάσεις στο λογαριθμικό πίνακα αναζήτησης. Ο χρόνος αυτός βρέθηκε ίσος με 10ms. Επομένως

0,436+0,010=0,446 δευτερόλεπτα. Σύγκριση χρόνων με την υλοποιήση στη CPU:

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
1,32 δευτ.	0,446	0,33x

Από τα αποτελέσματα είναι εμφανές ότι η πρώτη υλοποίηση δεν εκμεταλλεύεται τις δυνατότητες της κάρτας γραφικών και είναι πιο αργή ακόμα και από αυτή της CPU. Για να τη βελτιώσουμε μειώσαμε τον αριθμό των νημάτων ανά μπλοκ από 128 σε 64 με στόχο να έχουμε περισσότερα μπλοκ νημάτων και με αυτό τον τρόπο να "κρύψουμε" τις καθυστερήσεις που προκύπτουν όταν τα νήματα ζητάνε δεδομένα από την ίδια περιοχή της καθολικής μνήμης. Τα αποτελέσματα ήταν τα ακόλουθα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
1,27 δευτ.	0,375	3,85Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
1,27 δευτ.	0,446	0,35x

Αν και υπήρξε βελτίωση με τη μείωση των νημάτων ανά μπλοκ, ωστόσο ο χρόνος εκτέλεσης του πυρήνα παραμένει ακόμα μεγάλος. Παρατηρούμε ότι η ρυθμαπόδοση της μνήμης είναι μικρή. Για να την αυξήσουμε θα κάνουμε χρήση της κοινόχρηστης μνήμης η οποία είναι πολύ ταχύτερη της καθολικής. Στην κοινόχρηστη μνήμη θα περάσουμε δεδομένα από την καθολική, τα οποία χρησιμοποιούνται από κοινού από τα νήματα σε κάθε μπλοκ. Τα δεδομένα αυτά είναι τα διανύσματα των μέσων τιμών, των διασπορών και των βαρών των συνιστωσών των μοντέλων. Επίσης είναι τα αποτελέσματα από τη mgau_precomp. Ο λογαριθμικός πίνακας δεν μπορεί να περάσει στην κοινόχρηστη μνήμη λόγω του μεγέθους του που ξεπερνάει τα 16KB που διαθέτει κάθε πολυεπεξεργαστής της GTX280. Η πιο απλή τεχνική για να περάσει κανείς δεδομένα από την καθολική μνήμη στην κοινόχρηστη, είναι να χρησιμοποιήσει ένα νήμα, συνήθως το πρώτο από κάθε μπλοκ, και αυτό να αντιγράψει τα δεδομένα στην κοινόχρηστη μνήμη. Αυτή η τεχνική όμως δεν είναι η βέλτιστη όταν έχουμε πολλά δεδομένα προς αντιγραφή. Ο λόγος είναι ότι όταν το ένα νήμα αντιγράφει τα δεδομένα, τα υπόλοιπα παραμένουν ανενεργά χωρίς να εκτελούν κάποιο έργο. Γι'αυτό στη δική μας υλοποίηση κάναμε κάτι διαφορετικό. Για τα διανύσματα μέσων τιμών και διασπορών που το μήκος τους είναι πολύ πιθανόν να είναι μεγαλύτερο από τον αριθμό των νημάτων ανά μπλοκ ακόμα και όταν αυτά είναι περισσότερα από

64, την αντιγραφή αναλαμβάνουν πολλά νήματα. Συγκεκριμένα, διαιρέσαμε το μήκος των διανυσμάτων με τον αριθμό των νημάτων ανά μπλοκ. Το αποτέλεσμα μας δείχνει πόσα στοιχεία από τα διανύσματα μπορεί να αντιγράψει ένα νήμα. Αν το αποτέλεσμα της διαίρεσης έχει υπόλοιπο, τότε αυτά τα περισσεύοντα στοιχεία θα αντιγραφούν από ίσο αριθμό νημάτων. Για παράδειγμα, στη δική μας περίπτωση τα διανύσματα έχουν μήκος $8 \times 39 = 312$, όπου 8 ο αριθμός των συνιστωσών στο κάθε μοντέλο και 39 το μήκος μιας συνιστώσας. Διαιρώντας με τον αριθμό νημάτων ανά μπλοκ που είναι 64, $312 \div 64 = 4,875$, προκύπτει ότι κάθε νήμα σε ένα μπλοκ θα αντιγράψει στην κοινόχρηστη μνήμη 4 διαφορετικά στοιχεία. Τα υπόλοιπα $312 - (4 * 64) = 56$ στοιχεία θα αντιγραφούν από τα 56 πρώτα νήματα κάθε μπλοκ. Κάνοντας χρήση της κοινόχρηστης μνήμης με αυτό τον τρόπο, είχαμε τα παρακάτω αποτελέσματα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,98 δευτ.	0,312	4,42Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,98 δευτ.	0,446	0,45x

Παρατηρούμε ότι ο χρόνος εκτέλεσης του πυρήνα έπεσε σημαντικά αλλά και πάλι είναι μεγάλος σε σχέση με τη CPU. Σαν τελευταία βελτίωση αυτής της υλοποίησης, θα μεταφέρουμε στην κοινόχρηση μνήμη και τα χαρακτηριστικά των πλαισίων. Το μέγεθος των χαρακτηριστικών σε στοιχεία για κάθε μπλοκ είναι ίσο με το γινόμενο του αριθμού των νημάτων ανά μπλοκ επί το μήκος των χαρακτηριστικών. Κάθε νήμα, που αντιστοιχεί σε ένα πλαίσιο, αντιγράφει τα δικά του χαρακτηριστικά στην κοινόχρηστη μνήμη. Με αυτή τη βελτίωση, τα αποτελέσματα διαμορφώθηκαν όπως παρακάτω:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,94 δευτ.	0,062	0,02Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,94 δευτ.	0,446	0,47x

Ο χρόνος εκτέλεσης του πυρήνα μείωθηκε και άλλο αλλά και πάλι δεν υπήρξε επιτάχυνση. Αξίζει να σημειωθεί στα τελευταία αποτελέσματα, ότι η κατοχή και η ρυθμαπόδοση της μνήμης έπεσαν σε πολύ χαμηλά επίπεδα. Το γεγονός αυτό

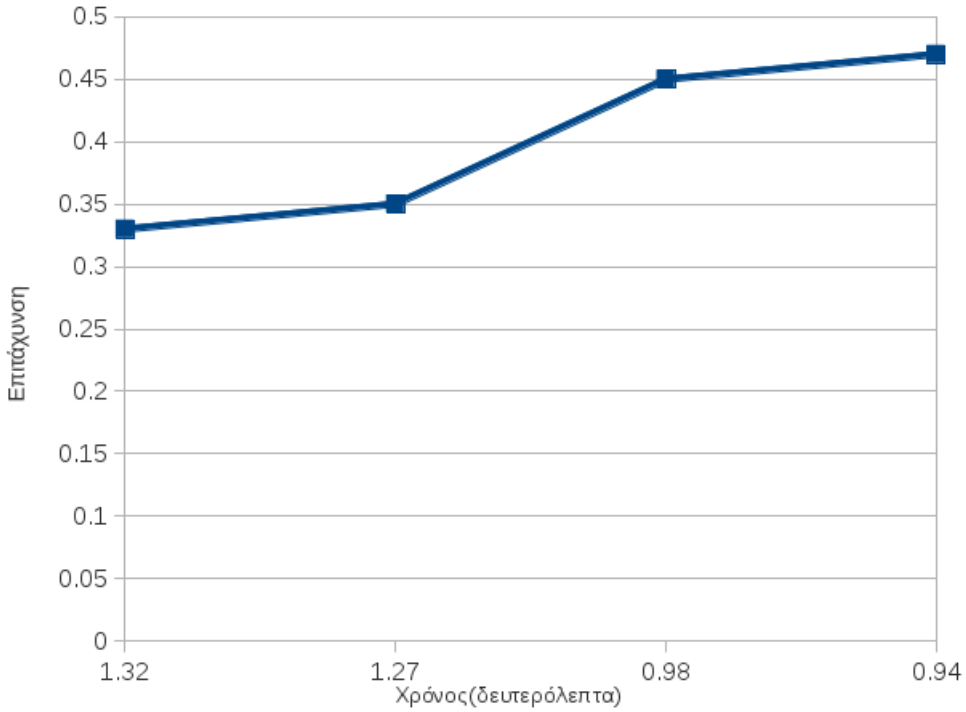
οφείλεται στην εκτεταμένη χρήση της κοινόχρηστης μνήμης. Περνώντας σ'αυτήν και τα χαρακτηριστικά των πλαισίων, αυξήθηκε το ποσοστό που χρησιμοποιεί κάθε μπλοκ. Αυτό είχε σαν αποτέλεσμα ο πολυεπεξεργαστής να μην μπορεί να έχει ταυτόχρονα πολλά μπλοκ νημάτων ενεργά αφού δεν μπορούσαν να ικανοποιηθούν οι απαιτήσεις σε κοινόχρηστη μνήμη. Επίσης μειώθηκαν οι αναγνώσεις από την καθολική μνήμη και γι'αυτό η ρυθμαπόδοση της είναι τόσο χαμηλή. Πρέπει να προσθέσουμε ότι η χρήση της κοινόχρηστης μνήμης έγινε με τέτοιο τρόπο ώστε δεν υπήρχαν συγκρούσεις στις αναγνώσεις από τις τράπεζες. Αυτό κατέστη δυνατό αφού τα νήματα σε κάθε μπλοκ έκαναν προσπέλαση του ίδιου στοιχείου από τη μνήμη κάθε στιγμή. Συγκεντρωτικά γι'αυτή την πρώτη υλοποίηση είχαμε τα παρακάτω αποτελέσματα:

Χρόνος πυρήνα	Νήματα/μπλοκ	Κοιν.Μνήμη	Κατοχή	Ρυθμαπόδοση μνήμης
1,32 δευτ.	128	όχι	0,375	3,72Gb/δευτ.
1,27 δευτ.	64	όχι	0,375	3,85Gb/δευτ.
0,98 δευτ.	64	ναι	0,312	4,42Gb/δευτ.
0,94 δευτ.	64	ναι	0,062	0,02Gb/δευτ.

Οι επιταχύνσεις από την πρώτη υλοποιήση φαίνονται στο γράφημα 6.1 .

6.6 Δεύτερη υλοποίηση

Στα αποτελέσματα της πρώτης υλοποίησης παρατηρήθηκε ότι δεν υπήρξε επιτάχυνση της mgau_eval εν συγκρίσει με την υλοποίησή της στη CPU. Με μία προσεκτικότερη ματιά στα αποτελέσματα μπορεί κανείς να εντοπίσει την αιτία στην κατοχή. Αν και δεν υπάρχει συγκεκριμένο νούμερο για την κατοχή που πρέπει να επιτευχθεί αφού εξαρτάται από τον εκάστοτε αλγόριθμο, ωστόσο κατοχή μικρότερη του 0,5 είναι συνήθως ένδειξη ότι η κάρτα γραφικών δε χρησιμοποιείται σε ικανοποιητικό βαθμό. Για να αυξήσουμε την κατοχή διαφοροποιήσαμε το έργο που εκτελεί κάθε νήμα. Σκοπός μας είναι να αυξήσουμε το συνολικό αριθμό νημάτων που χρειάζονται για την εύρεση των σκορ. Στην πρώτη υλοποίηση επιλέξαμε το κάθε νήμα να βρίσκει όλα τα σκορ για ένα πλαίσιο και έτσι ο αριθμός των νημάτων ήταν ίσος με τον αριθμό των πλαισίων. Σε αυτή την υλοποίηση κάναμε μια διαφορετική προσέγγιση, επιλέγοντας το κάθε νήμα

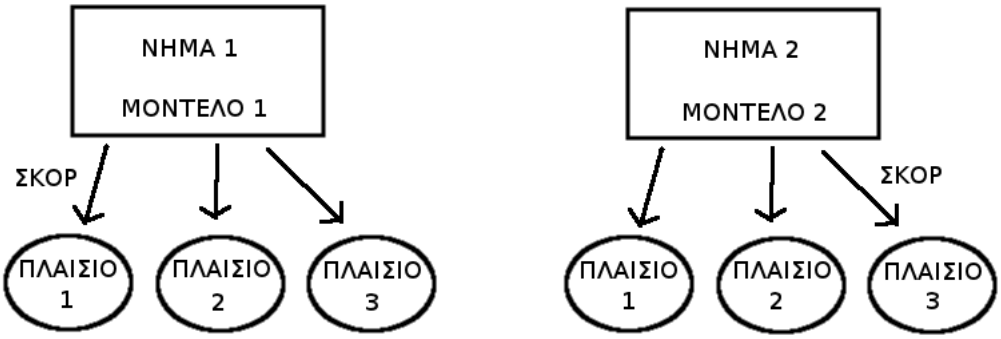


Σχήμα 6.1: Οι επιταχύνσεις της πρώτης υλοποίησης

να αντιπροσωπεύει στην ουσία ένα μοντέλο και να βρίσκει το σκορ αυτού του μοντέλου για τα εισερχόμενα πλαίσια φωνής. Στο σχήμα 6.2 αναπαριστάται η προσέγγιση αυτή με ένα απλό παράδειγμα δύο νημάτων και τριών πλαισίων φωνής. Στη δική μας περίπτωση τα νήματα είναι συνολικά 1135, όσα και τα μοντέλα, και τα πλαίσια 362. Κάνοντας χρήση μόνο καθολικής μνήμης η δεύτερη υλοποίηση είχε τα εξής αποτελέσματα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,44 δευτ.	0,75	19,82Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,44 δευτ.	0,446	1,01x



Σχήμα 6.2: Δομή δεύτερης υλοποίησης

Αυτή η δεύτερη υλοποίηση είναι σαφώς καλύτερη της πρώτης αφού ο χρόνος εκτέλεσης του πυρήνα μειώθηκε σε περισσότερο του μισού της πρώτης υλοποίησης. Επίσης η κατοχή και η ρυθμαπόδοση της μνήμης ανέβηκαν κατά πολύ αν και απέχουμε ακόμα αρκετά από τη μέγιστη θεωρητική ρυθμαπόδοση. Πρακτικά έχουμε πλέον μία υλοποίηση που χρονικά συγκρίνεται με την αντίστοιχη της CPU. Ο αριθμός των νημάτων ανά μπλοκ ορίστηκε ίσος με 128 και έχοντας την εμπειρία της πρώτης υλοποίησης το μειώσαμε σε 64 για να δούμε αν θα βελτιωθεί ο χρόνος του πυρήνα. Με 64 νήματα σε κάθε μπλοκ βρέθηκε ότι:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,43 δευτ.	0,5	20,54Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,43 δευτ.	0,446	1,03x

Η μείωση των νημάτων ανά μπλοκ βελτίωσε το χρόνο εκτέλεσης του πυρήνα μόλις ένα εκατοστό του δευτερολέπτου. Η επόμενη κίνηση είναι να κάνουμε χρήση της κοινόχρηστης μνήμης. Σε αντίθεση με την πρώτη υλοποίηση, εδώ δεν μπορούμε να κάνουμε εκτεταμένη χρήση της κοινόχρηστης μνήμης. Από τη στιγμή που κάθε νήμα αντιστοιχεί σε ένα μοντέλο, τα νήματα σε ένα μπλοκ δε μοιράζονται δεδομένα που αφορούν τα χαρακτηριστικά των μοντέλων. Κατά συνέπεια, τα διανύσματα των μέσων τιμών, των διασπορών και το κομμάτι που

έχει υπολογισθεί από τη mgau_precomp, παραμένουν στην καθολική μνήμη. Στην κοινόχρηστη μνήμη μπορούν να περασθούν τα χαρακτηριστικά των πλαισίων. Συγκεκριμένα, τα νήματα σε κάθε μπλοκ "διατρέχουν" όλα τα πλαίσια με τη σειρά για να βρουν τα αντίστοιχα σκορ. Επομένως την ίδια στιγμή απ'όλα τα νήματα σε ένα μπλοκ, χρησιμοποιείται το διάνυσμα χαρακτηριστικών του ίδιου πλαισίου. Αυτό το διάνυσμα αποθηκεύεται στην κοινόχρηστη μνήμη. Με αυτή τη βελτίωση προέκυψαν από το χρονικό αναλυτή τα ακόλουθα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,42 δευτ.	0,5	20,32Gb/δευτ.

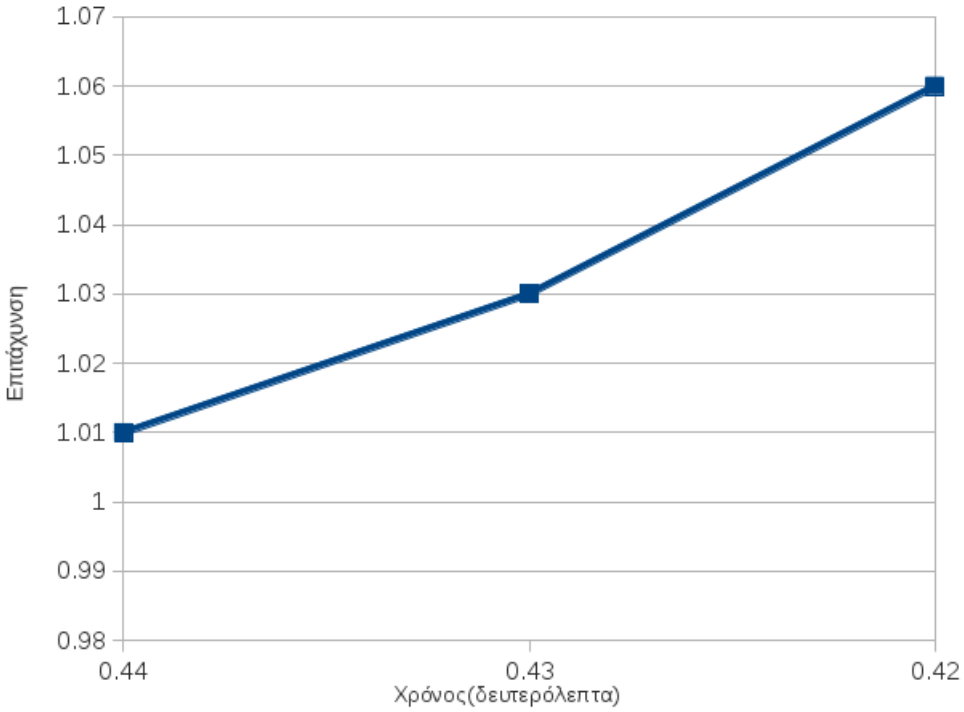
Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,42 δευτ.	0,446	1,06x

Η χρήση κοινόχρηστης μνήμης βελτίωσε ελάχιστα το χρόνο αλλά αυτό ήταν αναμενόμενο λόγω των αρκετών δεδομένων που αναγκαστικά παρέμειναν στην καθολική μνήμη. Τα νήματα ανά μπλοκ παρέμειναν 64. Οι επιταχύνσεις από τη δεύτερη υλοποίηση φαίνονται στο σχήμα 6.3 και ήταν όλες σχεδόν μηδενικές. Συγκεντρωτικά για τη δεύτερη υλοποίηση:

Χρόνος πυρήνα	Νήματα/μπλοκ	Κοιν.Μνήμη	Κατοχή	Ρυθμαπόδοση μνήμης
0,44 δευτ.	128	όχι	0,75	19,82Gb/δευτ.
0,43 δευτ.	64	όχι	0,5	20,54Gb/δευτ.
0,42 δευτ.	64	ναι	0,5	20,32Gb/δευτ.

6.7 Τρίτη υλοποίηση

Με τη δεύτερη υλοποίηση καταφέραμε να φτάσουμε το χρόνο εκτέλεσης της υλοποίησης της CPU. Με αυτή την υλοποίηση, στόχος μας είναι να αυξήσουμε ακόμα περισσότερο τον συνολικό αριθμό νημάτων και να παραλληλοποιήσουμε σε μεγαλύτερο βαθμό τον αλγόριθμο. Για να το επιτύχουμε αυτό, θα ορίσουμε τον αριθμό των νημάτων ίσο με το γινόμενο του αριθμού των πλαισίων με το αριθμό των μοντέλων. Με αυτή τη λογική, κάθε νήμα θα είναι υπεύθυνο για την εύρεση ενός μόνο σκορ το οποίο θα προκύπτει από ένα πλαίσιο και ένα μοντέλο. Η προσέγγιση αυτή διαφοροποιείται από τις προηγούμενες όπου ένα



Σχήμα 6.3: Οι επιταχύνσεις της δεύτερης υλοποίησης

νήμα ήταν υπεύθυνο για την εύρεση πολλών σκορ. Για να βρούμε ποιο μοντέλο και ποιο πλαίσιο θα χρησιμοποιηθεί από ένα νήμα, χρησιμοποιούμε το μαθηματικό τελεστή του υπολοίπου. Αυτό είναι υποχρεωτικό ώστε σε κάθε νήμα να αντιστοιχεί ένας μοναδικός συνδυασμός πλαισίου και μοντέλου. Αναλυτικότερα, ο αριθμός του πλαισίου είναι το υπόλοιπο της διαίρεσης του αριθμού του νήματος με το συνολικό αριθμό πλαισίων και ο αριθμός του μοντέλου είναι το υπόλοιπο της διαίρεσης του αριθμού του νήματος με το συνολικό αριθμό των μοντέλων. Χρησιμοποιώντας ως παράδειγμα τα μεγέθη των πλαισίων και των μοντέλων που χρησιμοποιούμε για τη χρονική ανάλυση, έχουμε 362 πλαίσια και 1135 μοντέλα. Συνολικά τα νήματα θα είναι $362 \times 1135 = 410870$. Διαλέγοντας ένα τυχαίο νήμα, έστω το χιλιοστό νήμα, το πλαίσιο του θα είναι το πλαίσιο νούμερο $1000 modulo 1135 = 1135$ και το μοντέλο, το μοντέλο νούμερο $1000 modulo 362 = 276$. Διακρίνεται μια ιδιαίτερη περίπτωση. Όταν ο αριθμός των πλαισίων και ο αριθμός των μοντέλων έχουν το ίδιο τελευταίο ψηφίο, με χρήση του υπολοίπου δεν μπορούμε να πάρουμε όλους τους δυνατούς συν-

δυασμούς. Σε αυτήν την περίπτωση, διαφοροποιούμε τον τρόπο εύρεσης των μοντέλων και των πλασίων για κάθε νήμα. Όταν ο αριθμός του νήματος είναι μικρότερος του γινομένου του αριθμού των μοντέλων με το μειωμένο κατά ένα, αριθμό των πλαισίων, τότε ο αριθμός του μοντέλου είναι το υπόλοιπο της διαίρεσης του αριθμού του νήματος με το συνολικό αριθμό των μοντέλων. Ο αριθμός όμως του πλαισίου είναι το υπόλοιπο της διαίρεσης του αριθμού του νήματος με το μειωμένο κατά ένα αριθμό των πλαισιών. Όταν ο αριθμός του νήματος είναι μεγαλύτερος ή ίσος του γινομένου του αριθμού των μοντέλων με το μειωμένο κατά ένα, αριθμό των πλαισίων, τότε ο αριθμός του μοντέλου είναι το υπόλοιπο της διαίρεσης του αριθμού του νήματος με το συνολικό αριθμό των μοντέλων. Ο αριθμός του πλαισίου είναι ίσος με τον αριθμό των νημάτων κατά ένα. Συνοπτικά:

Αν αριθμός νήματος < συνολ. αριθμός μοντέλων(συνολ. αριθμός πλαισίων -1)*
αριθμός μοντέλου=αριθμός νήματος modulo συνολικός αριθμός μοντέλων
αριθμός πλαισίου=αριθμός νήματος modulo (συνολικός αριθμός πλαισίων - 1)
Αν αριθμός νήματος >= συνολ. αριθμός μοντέλων(συνολ. αριθμός πλαισίων -1)*
αριθμός μοντέλου=αριθμός νήματος modulo συνολικός αριθμός μοντέλων
αριθμός πλαισίου=συνολικός αριθμός πλαισίων - 1

Επειδή αυτή η υλοποίηση χρησιμοποιεί μεγάλο αριθμό νημάτων, επιλέξαμε ο αριθμός νημάτων ανά μπλοκ να είναι ίσος με 256. Ο χρονικός αναλυτής παρουσίασε τα παρακάτω:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,17 δευτ.	1	72,72Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,17 δευτ.	0,446	2,62x

Παρατηρούμε ότι πλέον έχουμε επιτάχυνση σε σχέση με τη CPU, γεγονός που οφείλεται στο μεγάλο αριθμό νημάτων που χρησιμοποιήσαμε και εκτελούνταν παράλληλα. Σε αυτή την προσέγγιση που κάναμε, δεν μπορεί να χρησιμοποιηθεί κοινόχρηστη μνήμη αφού τα νήματα ενός μπλοκ δε μοιράζονται δεδομένα. Συνολικά τα αποτελέσματα ήταν:

Χρόνος πυρήνα	Νήματα/μπλοκ	Κοιν.Μνήμη	Κατοχή	Ρυθμαπόδοση μνήμης
0,17 δευτ.	256	όχι	1	72,72Gb/δευτ.

6.8 Τέταρτη υλοποίηση

Στην τελευταία υλοποίηση χρησιμοποιήσαμε τον ίδιο συνολικό αριθμό νημάτων με την τρίτη υλοποίηση, δηλαδή το γινόμενο του αριθμού των πλαισίων με τον αριθμό των μοντέλων. Όμως τώρα, ο αριθμός των νημάτων ανά μπλοκ δεν είναι μία ανεξάρτητη επιλογή. Αντιθέτως, το θέσαμε ίσο με τον αριθμό των πλαισίων. Το γεγονός αυτό μας βοηθάει να αποφύγουμε τη χρήση του τελεστή του υπολοίπου που είχαμε στην προηγούμενη υλοποίηση και ο οποίος είναι χρονοβόρος. Για την εύρεση του εκάστοτε μοντέλου και του εκάστοτε πλαισίου από κάθε νήμα, αρκούν οι ενσωματωμένες μεταβλητές blockIdx.x και threadIdx.x . Συγκεκριμένα, από τη στιγμή που σε κάθε μπλοκ έχουμε τόσα νήματα όσα και τα πλαίσια, ο συνολικός αριθμός των μπλοκ νημάτων είναι ίσος με τον αριθμό των μοντέλων. Για παράδειγμα, έχουμε 362 πλαίσια και 1135 μοντέλα. Συνολικά τα νήματα είναι ίσα με 362*1135 και σε κάθε μπλοκ τα νήματα είναι 362. Επομένως τα μπλοκ είναι 1135. Τα νήματα του ίδιου μπλοκ, βρίσκουν τα σκορ που αφορούν τα διαφορετικά πλαίσια αλλά το ίδιο μοντέλο. Έτσι, τα νήματα του πρώτου μπλοκ θα παράγουν 362 σκορ που αφορούν όλα τα πλαίσια και το πρώτο μοντέλο. Τα νήματα του δεύτερου μπλοκ θα παράγουν και αυτά 362 σκορ που αφορούν όλα τα πλαίσια και το δεύτερο μοντέλο και ούτω καθεξής. Άρα, ο αριθμός του μοντέλου δίνεται από το blockIdx.x και του πλαισίου από το threadIdx.x . Κάνοντας χρήση μόνο καθολικής μνήμης, ο χρονικός αναλυτής μας έδειξε γι'αυτή την προσέγγιση ότι:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,058 δευτ.	1	79,8Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,058 δευτ.	0,446	7,68x

Σε αντίθεση με την τρίτη υλοποίηση, εδώ μπορούμε να κάνουμε χρήση της κοινόχρηστης μνήμης. Όπως και στην πρώτη υλοποίηση έτσι και εδώ, περάσαμε στην κοινόχρηστη μνήμη τα διανύσματα των μέσων τιμών, των διασπορών, των βαρών των συνιστωσών και το κομμάτι από τη mgau_precomp. Ο μηχανισμός είναι ο ίδιος με αυτόν της πρώτης υλοποίησης ενώ τα διανύσματα των χαρακτηριστικών των πλαισίων δεν έχει νόημα να περαστούν στη κοινόχρηστη μνήμη αφού δε μοιράζονται από τα νήματα μέσα στο κάθε μπλοκ. Με την προσθήκη της κοινόχρηστης μνήμης είχαμε τα παρακάτω αποτελέσματα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,053 δευτ.	1	76,5Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,053 δευτ.	0,446	8,41x

6.8.1 Περαιτέρω βελτιστοποιήσεις

Η αρχιτεκτονική του CUDA μας προσφέρει τη δυνατότητα να δοκιμάσουμε εκτός από την κοινόχρηστη μνήμη και άλλα πράγματα που πιθανόν να βελτιώσουν το χρόνο εκτέλεσης ενός πυρήνα. Είναι προφανές ότι τυχόν βελτιστοποιήσεις που μπορεί να κάνει κανείς εξαρτώνται αποκλειστικά από τον αλγόριθμο και δεν είναι βέβαιο ότι όντως θα μειώσουν το χρόνο[12]. Στη δική μας περίπτωση, θα δοκιμάσουμε δύο πράγματα. Το πρώτο είναι το ξετύλιγμα βρόγχου και το δεύτερο η χρήση μιας μνήμης που διαθέτει το CUDA με την ονομασία *texture memory*.

Το ξετύλιγμα βρόγχου είναι μια μέθοδος που χρησιμοποιείται γενικά στον προγραμματισμό. Εμείς το χρησιμοποιήσαμε σε ένα βρόγχο. Συγκεκριμένα, όπως φαίνεται από τον τύπο 4.1, το κλάσμα μέσα στο άθροισμα υλοποιείται από ένα βρόγχο 39 επαναλήψεων, όσο είναι και το μήκος του διανύσματος χαρακτηριστικών οποιουδήποτε πλαισίου. Η τυπική προσέγγιση που χρησιμοποιούσαμε έως τώρα είναι η μεταβλητή ελέγχου του βρόγχου να κυμαίνεται από το μηδέν έως το 39 και σε κάθε επανάληψη να αυξάνεται κατά ένα. Αφού οι επαναλήψεις είναι 39 έχουμε δύο επιλογές για πως θα αλλάξουμε την αύξηση της μεταβλητής ελέγχου χωρίς να χρειαστεί να δημιουργήσουμε και δεύτερο βρόγχο. Η μία είναι να αυξάνεται κατά 13, οπότε προκύπτουν 3 μόνο επαναλήψεις και μέσα σε κάθε επανάληψη εκτελούμε ότι κώδικα θα εκτελούσαμε για 13 επαναλήψεις του βρόγχου. Αυτή η επιλογή απαιτεί πολλούς καταχωρητές και πιθανόν να μην μπορούμε καθόλου να μεταγλωτίσουμε τον κώδικά μας αφού μπορεί να ξεπεράσουμε τις δυνατότητες του υλικού. Στην καλύτερη περίπτωση θα χρησιμοποιηθεί η πολύ αργή τοπική μνήμη για τους πρόσθετους καταχωρητές. Γι'αυτό το λόγο την απορρίπτουμε και επιλέγουμε να αυξάνουμε τη μεταβλητή ελέγχου κατά 3. Ξετυλίγοντας λοιπόν το βρόγχο, είχαμε το παρακάτω αποτέλεσμα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,052 δευτ.	1	78,5Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,052 δευτ.	0,446	8,57x

Το ξετύλιγμα του βρόγχου βελτίωσε κατά 1ms το χρόνο του πυρήνα.

Η μνήμη texture είναι μια περιοχή μνήμης που οι κάρτες γραφικών χρησιμοποιούν για να επιταχύνουν συχνές διεργασίες όπως η μετατροπή μίας δισδιάστατης επιφάνειας σε τρισδιάστατο μοντέλο. Τέτοιες διεργασίες εκτελούνται συχνά σε παιχνίδια με σκοπό να δώσουν σε αντικείμενα με χαμηλή ανάλυση απεικόνισης, οπτικά μεγαλύτερη λεπτομέρεια. Το CUDA μας δίνει τη δυνατότητα να χρησιμοποιήσουμε αυτή την περιοχή μνήμης προς όφελός μας χωρίς να υπάρχει περιορισμός στο πως θα επεξεργαστούμε τα δεδομένα που αποθηκεύουμε εκεί. Ωστόσο η μνήμη αυτή έχει κάποια ιδιαίτερα χαρακτηριστικά. Καταρχάς τα νήματα ενός πυρήνα έχουν μόνο δυνατότητα ανάγνωσης της μνήμης texture. Επίσης, διαφορετικές αναφόρες σε τέτοια μνήμη μπορούν να αντιστοιχούν στην ίδια περιοχή της texture μνήμης. Τέλος, υπάρχει η δυνατότητα αποθήκευσης δεδομένων υπό τη μορφή κρυφής μνήμης. Ο προγραμματιστής όμως δεν έχει τη δυνατότητα να ελέγξει το ποια δεδομένα θα μείνουν στη κρυφή μνήμη ούτε μπορεί να τη διαχειριστεί με κάποιο τρόπο. Ισχύει ότι μία ανάγνωση από τη texture μνήμη κοστίζει όσο μία ανάγνωση από τη μνήμη της GPU όταν έχουμε αστοχία της κρυφής μνήμης. Διαφορετικά το κόστος είναι όσο μίας ανάγνωσης από την κρυφή μνήμη, γεγονός που σημαίνει ότι σε περίπτωση αστοχίας, θα επιβαρυνθούμε με εκατοντάδες κύκλους αναμονής. Τα δεδομένα που περάσαμε στη texture μνήμη είναι ο λογαριθμικός πίνακας αναζήτησης. Ο πίνακας αυτός είναι αρκετά μεγάλος με αποτέλεσμα να μην μπορεί να περαστεί στην κοινόχρηστη μνήμη. Άρα μία ενδεχομένως καλύτερη επιλογή από την καθολική μνήμη που χρησιμοποιούσαμε έως τώρα, να είναι η εν λόγω μνήμη. Τα υπόλοιπα δεδομένα που είναι ήδη στην κοινόχρηστη μνήμη δεν έχει νόημα να περαστούν στη texture. Είναι δεδομένο ότι η κοινόχρηστη μνήμη είναι πολύ ταχύτερη από την texture. Με χρήση λοιπόν της texture είχαμε τα εξής αποτελέσματα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,052 δευτ.	1	75,5Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,052 δευτ.	0,446	8,57x

Παρατηρούμε ότι η αποθήκευση του πίνακα σε διαφορετική μνήμη δεν επέφερε καμία βελτίωση στην απόδοση. Είναι προφανές ότι αιτία γι'αυτό είναι οι πάρα πολλές αστοχίες της κρυφής μνήμης που έκαναν την texture μνήμη να συμπεριφέρεται όπως η καθολική μνήμη. Άλλωστε η χρήση της συγκεκριμένης μνήμης δεν

εγγυάται ότι θα υπάρξει βελτίωση και γι'αυτό το λόγο θεωρείται σαν χαμηλής προτεραιότητας από τους προγραμματιστές.

Συγκεντρωτικά για την τέταρτη υλοποίηση είχαμε τα παρακάτω αποτελέσματα:

Χρόνος πυρήνα	Νήματα/μπλοκ	Κοιν.Μνήμη	Κατοχή	Ρυθμαπόδοση μνήμης
0,058 δευτ.	362	όχι	0,75	79,8Gb/δευτ.
0,053 δευτ.	362	ναι	0,5	76,5Gb/δευτ.
0,052 δευτ.	362	ναι	0,5	78,5Gb/δευτ.
0,052 δευτ.	362	ναι	0,5	75,5Gb/δευτ.

Οι επιταχύνσεις για την τέταρτη υλοποίηση φαίνονται στο σχήμα 6.4 .

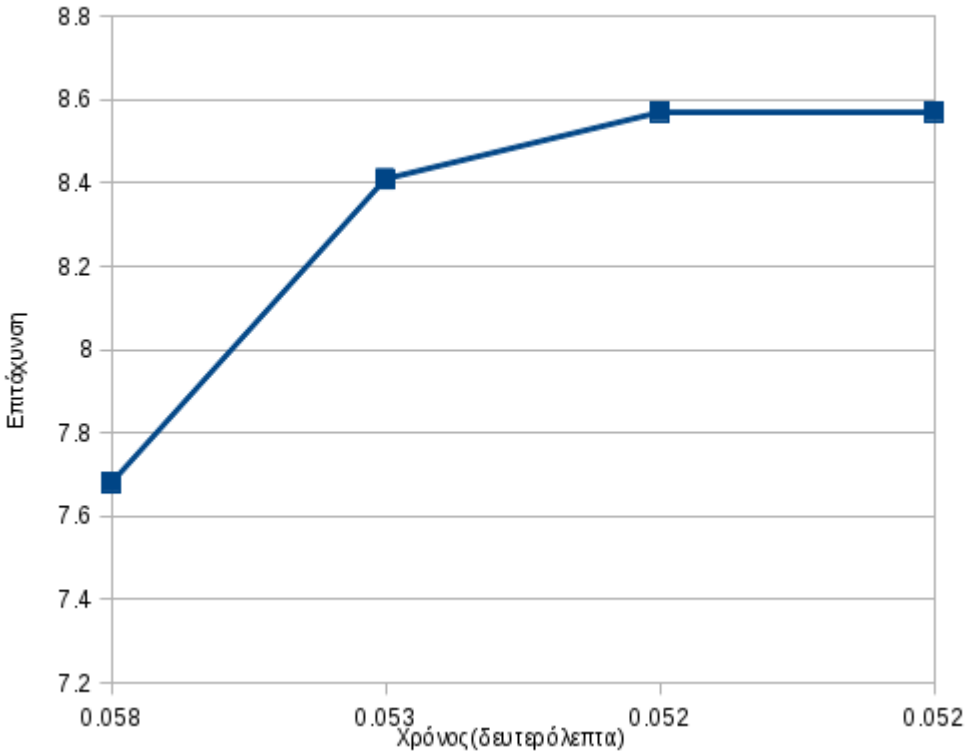
6.9 Υλοποίηση χωρίς το λογαριθμικό πίνακα αναζήτησης

Ο λογαριθμικός πίνακας αναζήτησης είναι προαιρετικός στην εκτέλεση του αποκωδικοποιητή αλλά είναι η προκαθορισμένη επιλογή καθώς βοηθά στην αποφυγή πολλών πράξεων οι οποίες επιβραδύνουν την εκτέλεση της mgau_eval . Στην περίπτωση που κάποιος δε θέλει να χρησιμοποιήσει τον πίνακα, τότε η πρόσθεση του σκορ μιας συνιστώσας με το συνολικό σκορ του μοντέλου δίνεται από τον τύπο:

$$z = log(b^x + b^y) \times \frac{1}{logb} \tag{6.2}$$

όπου x και y τα σκορ που προσθέτουμε και b η λογαριθμική βάση που έχει περαστεί σαν όρισμα στον αποκωδικοποιητή. Όλοι οι λογάριθμοι είναι φυσικοί. Υλοποιώντας αυτό τον τύπο και εισάγοντάς τον στην τέταρτη υλοποίηση που ήταν και η καλύτερη, λάβαμε τα ακόλουθα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,067 δευτ.	0,75	60,7Gb/δευτ.



Σχήμα 6.4: Οι επιταχύνσεις της τέταρτης υλοποίησης

Επιλέγοντας στην υλοποίηση της CPU να μην κάνουμε χρήση του λογαριθμικού πίνακα και διατηρώντας τη δέσμη ίση με μηδέν, πήραμε από το χρονικό αναλυτή τα αποτελέσματα:

Συνάρτηση	χρόνος CPU(δευτερόλεπτα)
logmath_add_exact	0,606
mgau_eval	0,432
logmath_log	0,166
logmath_exp	0,010

Στον παραπάνω πίνακα σημειώνονται μόνο οι συναρτήσεις που μας ενδιαφέρουν. Ο χρόνος που χρειάζεται η CPU για να εκτελέσει την ίδια εργασία με τον πυρήνα μας, είναι το άθροισμα των χρόνων αυτών των συναρτήσεων. Άρα συγκριτικά έχουμε:

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,067 δευτ.	1,214	18,11x

Αυτή τη φορά η επιτάχυνση που λάβαμε είναι αρκετά μεγάλη. Ο λόγος είναι ότι αφαιρώντας το λογαριθμικό πίνακα αναζήτησης μειώσαμε πάρα πολύ τις προσβάσεις στη μνήμη. Οι GPU είναι σχεδιασμένες να εκτελούν παράλληλα πολλές αριθμητικές πράξεις και εκεί είναι που υπερτερούν των CPU. Το γεγονός αυτό αποδεικνύεται και στη δική μας περίπτωση.

Θέλοντας να βελτιώσουμε το χρόνο εκτέλεσης του πυρήνα, κάναμε χρήση των *εγγενών συναρτήσεων*. Οι συναρτήσεις αυτές υποστηρίζονται μόνο στον κώδικα που εκτελείται στην κάρτα γραφικών και είναι λιγότερο ακριβείς αλλά ταχύτερες εκδόσεις μαθηματικών συναρτήσεων που υποστηρίζονται από το CUDA. Αντικαταστήσαμε τις συναρτήσεις που χρησιμοποιούσαμε για την υλοποίηση του τύπου 6.2 με εγγενείς συναρτήσεις. Το αποτέλεσμα αυτής της αντικατάστασης ήταν:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
0,051 δευτ.	0,75	80,1Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
0,051 δευτ.	1,214	23,80x

Βλέπουμε ότι ο χρόνος μειώθηκε αισθητά γεγονός που οφείλεται στην ευρεία χρήση του τύπου 6.2 . Ακόμα και αν οι εγγενείς συναρτήσεις είναι ελάχιστα πιο ταχείς από τις κλασικές συναρτήσεις, οι πάρα πολλές κλήσεις τους αποφέρουν βελτίωση της απόδοσης. Αξίζει να σημειωθεί ότι τα 51ms που έδειξε ο χρονικός αναλυτής είναι ελάχιστα καλύτερος χρόνος από την υλοποίηση που γινόταν χρήση του λογαριθμικού πίνακα. Αυτό μας υποδεικνύει ότι η χρήση της καθολικής μνήμης είναι τροχοπέδη στην απόδοση όταν δεν μπορεί να αποφευχθεί. Για να το αποδείξουμε, αντικαταστήσαμε στον πυρήνα μας τις αναγνώσεις από την καθολική μνήμη με σταθερές τιμές. Το αποτέλεσμα ήταν να μειωθεί ο χρόνος από 51ms σε 6,3ms. Επομένως λόγω της αναγκαστικής χρήσης της καθολικής μνήμης δεν μπορούμε να επιτύχουμε μεγαλύτερη, τουλαχιστόν αισθητά, επιτάχυνση.

6.10 Βελτίωση του χρόνου εκτέλεσης της mgau_precomp

Έχει ήδη αναλυθεί στην παράγραφο 6.3 η λειτουργία της mgau_precomp. Η συνάρτηση αυτή καλείται μόνο μία φορά και τα αποτελέσματά της διοχετεύονται

στη συνέχεια στη συνάρτηση mgau_eval που υλοποιήσαμε στις προηγούμενες παραγράφους. Πρέπει να υπενθυμίσουμε ότι τα σκορ στον αποκωδικοποιητή γίνονται στο λογαριθμικό τομέα. Αυτό σημαίνει ότι το πηλίκο 6.1 μετασχηματίζεται όπως παρακάτω:

$$log\left(\frac{1}{\sqrt{2\pi^{\nu}det(\sigma^2)}}\right) = 0 - log\left(\sqrt{2\pi^{\nu}det(\sigma^2)}\right) = \frac{1}{2}\nu log(2\pi) log(det(\sigma^2)) \tag{6.3}$$

Οι πίνακες των διασπορών μας δίνονται από το RM1 σε διαγώνια μορφή, επομένως η ορίζουσά τους είναι το γινόμενο των στοιχείων τους.

Στον πυρήνα που υλοποιήσαμε χρείαστηκε να περάσουμε μόνο τους πίνακες των διασπορών από όλα τα μοντέλα. Οργανώσαμε δε τα νήματα με τέτοιο τρόπο ώστε να έχουμε συνολικό αριθμό νημάτων ίσο με το γινόμενο του αριθμού των μοντέλων με τον αριθμό των συνιστωσών ανά μοντέλο. Κάθε νήμα αντιπροσωπεύει μία γκαουσιανή συνιστώσα και υλοποιεί τον τύπο 6.3 . Χρησιμοποιώντας μόνο καθολική μνήμη για την αποθήκευση των δεδομένων μας και 512 νήματα ανά μπλοκ, ο χρονικός αναλυτής μας έδειξε ότι:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
424 μsec.	1	54,2Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
424 μsec.	0,025 δευτ.	58,96x

Η υλοποίηση αυτή δε μας επιτρέπει να χρησιμοποιήσουμε κοινόχρηστη μνήμη αφού τα νήματα σε ένα μπλοκ δε μοιράζονται δεδομένα. Όμως το κάθε νήμα διαβάζει τα ίδια στοιχεία από την καθολική μνήμη περισσότερες από μία φορές και για να αποφύγουμε αυτές τις πολλαπλές αναγνώσεις, χρησιμοποιήσαμε καταχωρητές ώστε να διαβάζουμε από αυτούς τα αποτελέσματα της πρώτης ανάγνωσης. Αυτό είχε το ακόλουθο αποτέλεσμα:

Χρόνος πυρήνα	Κατοχή	Ρυθμαπόδοση αναγνώσεων καθολικής μνήμης
360 μsec.	1	31,8Gb/δευτ.

Χρόνος πυρήνα	Χρόνος CPU	Επιτάχυνση
360 μsec.	0,025 δευτ.	69,44x

Όπως ήταν αναμενόμενο, η χρήση καταχωρητών όπου ήταν δυνατό, μείωσε το χρόνο αφού σαν περιοχή μνήμης είναι πολύ ταχύτερη της καθολικής μνήμης.

6.11 Συνολικά αποτελέσματα

Έχοντας υλοποιήσει και τη mgau_eval αλλά και τη mgau_precomp με την αρχιτεκτονική του CUDA μπορούμε να παρουσιάσουμε συνολικά αποτελέσματα για διάφορες παραμετροποιήσεις του αποκωδικοποιητή. Τα αποτελέσματα αυτά θα μας δείξουν πόση ήταν εν τέλει η βελτίωση στο χρόνο που χρειάζεται ο αποκωδικοποιητής για να αναγνωρίσει μία πρόταση. Ας μην αγνοούμε το γεγονός ότι οι δύο συναρτήσεις που υλοποιήσαμε δεν είναι οι μοναδικές που εκτελούνται από τον αλγόριθμο καθώς και ότι πέρα από το χρόνο εκτέλεσης των πυρήνων μας, υπάρχουν και οι χρόνοι που απαιτούνται για τη μεταφορά των δεδομένων από την κάρτα στην κύρια μνήμη και αντιστρόφως. Επίσης στις υλοποιήσεις μας υπάρχουν και κάποιοι πρόσθετοι χρόνοι για τη σειριοποίηση πολυδιάστατων διανυσμάτων σε μονοδιάστατα. Η πρόταση που θα χρησιμοποιήσαμε για τις μετρήσεις είναι η ίδια που χρησιμοποιήσαμε και στις υλοποιήσεις της mgau_eval με τα 362 πλαίσια.

Η πρώτη παραμετροποίηση είναι με δέσμη ίση με μηδέν και χρήση λογαριθμικού πίνακα αναζήτησης. Το ότι η δέσμη είναι ίση με μηδέν προκαλεί τη χρήση όλων των μοντέλων για την εύρεση των σκορ άρα είναι στην ουσία η υλοποίηση που κάναμε και εμείς στην κάρτα γραφικών.

Χρόνος με χρήση CUDA	Χρόνος χωρίς CUDA	Επιτάχυνση
0,498 δευτ.	0,882 δευτ.	1,77x

Παρατηρούμε ότι με χρήση της τέταρτης υλοποίησης, που ήταν και η καλύτερη, συνολικά η επιτάχυνση του αλγορίθμου ήταν μικρή. Αυτό οφείλεται στο γεγονός ότι είχαμε τη δυνατότητα να βελτιώσουμε ένα ποσοστό μόνο του αλγορίθμου και όχι το 100% αυτού.

Η δεύτερη παραμετροποίηση είναι με δέσμη ίση με μηδέν αλλά χωρίς χρήση λογαριθμικού πίνακα αναζήτησης. Και πάλι χρησιμοποιούμε την τέταρτη υλοποίηση.

Χρόνος με χρήση CUDA	Χρόνος χωρίς CUDA	Επιτάχυνση
0,496 δευτ.	1,732 δευτ.	3,49x

Σε αυτή την περίπτωση η επιτάχυνση είναι διπλάσια.

6.11.1 Η επιρροή του κλαδέματος στα αποτελέσματα

Το κλάδεμα επηρεάζει σε μεγάλο βαθμό τις επιδόσεις του αλγορίθμου όσον αφορά την ταχύτητα αναγνώρισης καθώς περιορίζει σημαντικά τον αριθμό των χρησιμοποιούμενων μοντέλων. Παράλληλα, οι επιδόσεις όσον αφορά το ποσοστό αναγνώρισης μπορούν να παραμείνουν ίδιες αρκεί κάποιος να πειραματιστεί με τις τιμές των δεσμών. Για να γίνει αντιληπτό πως το κλάδεμα επηρεάζει τις επιδόσεις, θα κάνουμε μία ανόμοια σύγκριση. Θα συγκρίνουμε τα αποτελέσματα που έχει ο αποκωδικοποιητής με τις προκαθορισμένες τιμές που η δέσμη δεν είναι ίση με μηδέν, με τα δικά μας αποτελέσματα που προέρχονται από χρήση όλων των μοντέλων. Έπισης θα γίνει χρήση λογαριθμικού πίνακα αναζήτησης. Πρέπει να σημειώσουμε εδώ ότι η τιμή της δέσμης εξακουλουθεί να επηρεάζει τη δική μας υλοποίηση. Αυτό γίνεται γιατί από την τιμή της δέσμης εξαρτάται ο αριθμός των ενεργών HMMs που θα χρησιμοποιήσει ο αλγόριθμος του Viterbi. Έτσι αν η τιμή της δέσμης είναι ίση με $1e^{-120}$ όπως είναι άλλωστε και η προκαθορισμένη τιμή, η υλοποίησή μας θα χρησιμοποιήσει όλα τα μοντέλα για την εύρεση των σκορ των GMM σαν να ήταν η δέσμη ίση με μηδέν, αλλά για τον αλγόριθμο Viterbi θα χρησιμοποιήσει μόνο όσα HMMs είναι ενεργά λόγω της δέσμης που είναι ίση με $1e^{-120}$. Αυτό σημαίνει ότι εμείς υπολογίζουμε και σκορ τα οποία στη συνέχεια δε θα χρησιμοποιηθούν αλλά έχουμε ήδη εξηγήσει τους λόγους που δεν μπορούμε να το αποφύγουμε.

Τιμή δέσμης	Χρόνος με χρήση CUDA	Χρόνος χωρίς CUDA	Επιτάχυνση
1e-120	0,504 δευτ.	0,558 δευτ.	1,11x

Με αυτή την τιμή δέσμης και για τη συγκεκριμένη πρόταση, είχαμε ενεργά σε κάθε πλαίσιο κατά μέσο όρο 696 senones εκ των οποίων τα 135 είναι ανεξάρτητα συμφραζομένων και παραμένουν ενεργά σε κάθε πλαίσιο. Παρατηρούμε λοιπόν ότι ακόμα και σε αυτή την περίπτωση που η CPU βρήκε τα σκορ για περίπου τα μισά senones απ'ότι εμείς, η δική μας υλοποίηση είναι πιο γρήγορη.

Πώς επηρεάζει όμως η δέσμη την επίδοση του αλγορίθμου όσον αφορά την ορθότητα της αναγνώρισης; Σε όλες τις μετρήσεις που κάναμε έως τώρα, λάβαμε τα ίδια αποτελέσματα με τη CPU. Η πρόταση "GET ME THE LAST DATE ON FLASHER'S CAT-4 EQUIPMENT CASUALTY REPORTS" αναγνωριζόταν πάντα ως "GIVE ME THE LAST DATE ON FLASHER'S CAT-4 EQUIPMENT CASUALTY REPORTS". Αυτό πέρα από την ορθότητα των υλοποιήσεών μας, δείχνει ότι η τιμή της δέσμης δεν επηρέασε την αναγνώριση της πρότασης. Θέλοντας να δούμε τι επίπτωση έχει η αλλαγή της τιμής της δέσμης, τρέξαμε πολλές φορές τον αποκωδικοποιητή με διάφορες τιμές για τη δέσμη και χρησιμοποιώντας όλες

τις προτάσεις του RM1 που υπενθυμίζουμε ότι είναι 600. Τα αποτελέσματα που παράχθηκαν ήταν τα παρακάτω:

Τιμή δέσμης	Ποσοστό λανθασμένων προτάσεων	Ποσοστό λανθασμένων λέξεων
1e-30	87%(522/600)	63,9%(3661/5733)
1e-60	36,2%(217/600)	6,7%(381/5733)
1e-100	35,3%(212/600)	6,5%(373/5733)
1e-120	35,3%(212/600)	6,5%(373/5733)
1e-150	35,3%(212/600)	6,5%(373/5733)
0	35,3%(212/600)	6,5%(373/5733)

Βλέπουμε ότι πλησιάζοντας στο μηδέν, η απόδοση του αλγορίθμου όπως ήταν αναμενόμενο βελτιώνεται και σταθεροποιείται. Εξάλλου από το μηδέν έως το $1e^{-120}$ η απόδοση δεν αλλάζει και αυτό σημαίνει πρακτικά ότι μπορούμε να έχουμε τα ίδια αποτελέσματα με αρκετά λιγότερους υπολογισμούς, όπως αναφέρθηκε και παραπάνω.

Καταλήγοντας, η τιμή της δέσμης που δίνει την καλύτερη σχέση ακρίβειας και ταχύτητας της αναγνώρισης, εξαρτάται από τις προτάσεις που θέλουμε να αναγνωρίσουμε, από την πρόταιρη εκπαίδευση των μοντέλων καθώς και από άλλες παραμέτρους του αποκωδικοποιητή. Αυτό σημαίνει ότι ενδεχομένως σε ένα άλλο σετ δεδομένων να χρειαζόμασταν μια τιμή δέσμης ακόμα πιο κοντά στο μηδέν για βέλτιστα αποτελέσματα. Ως εκ τούτου, η δική μας υλοποίηση και προσέγγιση δεν υστερεί σε σχέση με την υλοποίηση της CPU όταν είναι εφικτό το κλάδεμα σε μεγάλο βαθμό, αφού δείξαμε ότι ακόμα και τότε είχαμε επιτάχυνση, ενώ είναι βέβαιο ότι η δική μας υλοποίηση εξασφαλίζει ότι έχει γίνει η καλύτερη δυνατή επιλογή για τη δέσμη. Βέβαια σε κάποιο διαφορετικό σετ προτάσεων, ενδέχεται η καλύτερη επίδοση της CPU να είναι καλύτερη της δικής μας.

Τελειώνοντας, παραθέτουμε τα συγκριτικά αποτελέσματα της πιο χρονοβόρας παραμετροποίησης του αποκωδικοποιητή για την αναγνώριση ολόκληρου του RM1. Η τιμή της δέσμης είναι ίση με μηδέν και δε γίνεται χρήση του λογαριθμικού πίνακα αναζήτησης.

Χρόνος με χρήση CUDA	Χρόνος χωρίς CUDA	Επιτάχυνση
223,7 δευτ.	912,1 δευτ.	4,07x

Πρέπει να σημειώσουμε το εξής για τα παραπάνω αποτελέσματα. Η κάρτα γραφικών που είχαμε στη διάθεσή μας, υποστηρίζει έως 512 νήματα ανά μπλοκ. Με δεδομένο ότι η τέταρτη υλοποίησή μας θέτει τα νήματα ανά μπλοκ ίσα

με τον αριθμό των πλαισίων, δημιουργείται πρόβλημα όταν τα πλαίσια είναι περισσότερα των 512. Ωστόσο, νεότερες κάρτες γραφικών όπως αυτές της αρχιτεκτονικής Fermi, υποστηρίζουν μέχρι 1024 νήματα ανά μπλοκ. Αυτό σημαίνει ότι η υλοποίησή μας μπορεί να τρέξει σε νεότερες κάρτες χωρίς κανένα πρόβλημα, δεδομένου ότι και ο αλγόριθμος δεν επιτρέπει προτάσεις μεγαλύτερες των χιλίων πλαισίων. Για να μπορέσουμε να καλύψουμε τις προτάσεις που ξεπερνούσαν τα 512 πλαίσια στην τελευταία σύγκριση, χρησιμοποιήσαμε την τέταρτη υλοποίηση για προτάσεις μικρότερες από 512 πλαίσια και την τρίτη υλοποίηση για προτάσεις μεγαλύτερες απο 512 πλαίσια. Συγκεκριμένα, 75 από τις 600 προτάσεις έκαναν χρήση της τρίτης υλοποίησης. Τα 223,7 δευτερόλεπτα που χρειάστηκαν για την αναγνώριση όλου του RM1 δεν είναι επομένως ο καλύτερος δυνατός χρόνος, αφού ποσοστό του προέκυψε από υλοποίηση που δεν ήταν η ταχύτερη.

Κεφάλαιο 7

Επίλογος

7.1 Συμπεράσματα

Στόχος της παρούσας διπλωματικής εργασίας ήταν η επιτάχυνση του αλγορίθμου αναγνώρισης φωνής Sphinx3 με χρήση σύγχρονης τεχνολογίας που παρέχουν οι πολυεπεξεργαστικές κάρτες γραφικών.

Έχοντας παρουσιάσει συνολικά τέσσερις υλοποιήσεις για το σκοπό αυτό, με διάφορες βελτιστοποιήσεις σε κάθε μία, καταφέραμε να επιταχύνουμε τον αλγόριθμο σε βαθμό όμως που δεν μπορεί να θεωρηθεί πολύ μεγάλος. Αιτία γι'αυτό ήταν η μορφή του αλγορίθμου και οι βελτιώσεις που έχουν γίνει σε αυτόν από τους προγραμματιστές στο Carnegie Mellon.

Όντας ένας αλγόριθμός που έχει σχεδιαστεί με βάση το σειριακό πρότυπο εκτέλεσης κώδικα, παρουσιάζει λίγα σημεία που μπορούν να παραλληλοποιηθούν. Επίσης, αν και το μέγεθος του κώδικα είναι υπερβολικά μεγάλο, δεν μπορεί να θεωρηθεί χρονοβόρος λόγω ύπαρξης πολλών αριθμητικών πράξεων. Στον κώδικα γίνεται χρήση πάρα πολλών δομών στη μνήμη και η γενικότερη χρήση της μνήμης είναι μεγαλύτερη από την εκτέλεση αριθμητικών πράξεων χωρίς αυτό να σημαίνει ότι υπάρχουν μεγάλες απαιτήσεις μνήμης για να εφαρμοστεί ο αλγόριθμος. Τα χαρακτηριστικά αυτά φάνηκε στην πράξη ότι είναι ανασταλτικοί παράγοντες για την επιτάχυνση του αλγορίθμού, ακόμα και όταν χρησιμοποιήσαμε την κάρτα γραφικών με τέτοιο τρόπο ώστε αξιοποιήσαμε σε πολύ μεγάλο βαθμό τις δυνατότητές της.

Επίσης, η εισαγωγή του κλαδέματος στον αλγόριθμο, αποτελεί έναν πολύ βολικό και μη καταστρεπτικό για τα αποτελέσματα τρόπο, ώστε να αποφευχθούν πολλές αριθμητικές πράξεις και κατά συνέπεια να μειωθεί ο συνολικός χρόνος εκτέλεσης του κώδικα. Εμείς όμως δεν μπορούσαμε να εφαρμόσουμε το κλάδεμα

και ήμασταν αναγκασμένοι να βρούμε όλα τα σκορ. Αυτό είχε σαν συνέπεια η επιτάχυνση που πετύχαμε να είναι μικρή σε σχέση με την ταχύτερη περίπτωση της CPU.

Όσον αφορά τα συμπεράσματά μας από την ενασχόληση με την αρχιτεκτονική του CUDA, διαπιστώσαμε ότι οι δυνατότητες της κάρτας γραφικών αξιοποιούνται σε μεγάλο βαθμό όταν υπάρχουν πάρα πολλά νήματα. Αυτό φάνηκε από τις υλοποιήσεις μας όταν αυξάναμε από τη μία στην άλλη τον συνολικό αριθμό των νημάτων. Είναι επομένως πολύ σημαντικό να παραλληλοποιηθεί ο αλγόριθμος που μας απασχολεί σε όσο το δυνατόν μεγαλύτερη κλίμακα. Μάλιστα η αύξηση της πολυπλοκότητας της υλοποίησης δεν είναι κατ'ανάγκη αποτρεπτική αφού όπως διαπιστώθηκε και από την τρίτη υλοποίηση, ενώ χρησιμοποιήσαμε τον τελεστή του υπολοίπου που είναι από τους πιο χρονοβόρους μαθηματικούς τελεστές, η επιτάχυνση ήταν μεγαλύτερη σε σχέση με τη δεύτερη υλοποίηση.

Η επιλογή του αριθμού των νημάτων ανά μπλοκ φάνηκε ότι εξαρτάται από το συνολικό αριθμό των νημάτων. Αν συνολικά τα νήματα είναι λίγα και μένουν στην κάρτα αδρανείς πολυεπεξεργαστές, τότε η μείωση των νημάτων ανά μπλοκ βελτιώνει το χρόνο όπως φάνηκε και στην πρώτη υλοποίηση. Είναι επιθυμητό δηλαδή αφού τα συνολικά νήματα δεν μπορούν να γίνουν περισσότερα, να δημιουργηθούν αρκετά μπλοκ ώστε να μοιραστούν σε όλους ή σε όσο το δυνατόν περισσότερους πολυεπεξεργαστές. Στην περίπτωση που τα συνολικά νήματα είναι πολλά, όπως στην τρίτη και την τέταρτη υλοποίηση, τότε βέλτιστος αριθμός νημάτων ανά μπλοκ είναι ένας αριθμός μεταξύ 256 και 512. Αυτό επιτρέπει στη GPU να προγραμματίσει ομάδες νημάτων για εκτέλεση σε ένα μπλοκ την ίδια στιγμή που άλλες ομάδες μπορεί να περιμένουν τη μεταφορά δεδομένων από τη μνήμη. Αυτό μας βοήθησε στην τρίτη υλοποίηση που η χρήση της καθολικής μνήμης ήταν εκτεταμένη όσον αφορά τις αναγνώσεις.

Τα αποτελέσματα των υλοποιήσεων έδειξαν ότι η χρήση της κοινόχρηστης μνήμης είναι μία σίγουρη επιλογή για τη βελτίωση του χρόνου ενός πυρήνα. Όπου χρησιμοποιήθηκε αυτή, ο χρόνος εκτέλεσης μειώθηκε με πιο χαρακτηριστικό παράδειγμα την πρώτη υλοποίηση. Μάλιστα πρέπει κανείς να αποθηκεύσει όσα περισσότερα δεδομένα μπορεί στην κοινόχρηστη μνήμη έχοντας όμως κατά νου ότι το μέγεθός της είναι πάρα πολύ μικρότερο της καθολικής.

Η χρησιμοποίηση εγγενών συναρτήσεων βοήθησε και αυτή στην αύξηση της επιτάχυνσης. Εκμεταλλευτήκαμε όμως το γεγονός ότι η μικρότερή τους ακρίβεια σε σχέση με τις συνήθεις συναρτήσεις, δεν επηρέαζε τα αποτελέσματα της αποκωδικοποίησης. Επίσης το ξετύλιγμα βρόγχου είναι μία διαδεδομένη τεχνική που μπορεί όπως παρουσιάστηκε στην τελευταία υλοποίηση να έχει θετικό αντίκτυπο. Ωστόσο η μείωση που παρατηρήθηκε στη δική μας περίπτωση ήταν

αρκετά μικρή και πρέπει να συνυπολογίζει κανείς το γεγονός ότι ο αριθμός των καταχωρητών είναι περιορισμένος και το ξετύλιγμα βρόγχου πρέπει να γίνει με τέτοιο τρόπο ώστε να μην ελαττωθεί ο αριθμός των ενεργών μπλοκ σε κάθε πολυεπεξεργαστή.

Τέλος, η χρήση της καθολικής μνήμης πρέπει να είναι όσο το δυνατόν μικρότερη. Η αντικατάσταση του λογαριθμικού πίνακα με αριθμητικές πράξεις και η μείωση του χρόνου εκτέλεσης του πυρήνα υποδεικνύει ότι μερικές αριθμητικές πράξεις μπορεί να είναι καλύτερη επιλογή ακόμα και από μία ανάγνωση της καθολικής μνήμης. Πρώτα όμως πρέπει να δοκιμάζονται άλλες περιοχές μνήμης προς αντικατάσταση της καθολικής. Εμείς δοκιμάσαμε τη texture μνήμη η οποία δεν επέφερε καμία βελτίωση. Αποδεικνύεται έτσι ότι η συγκεκριμένη μνήμη δεν είναι αποδοτική για όλους τους τρόπους προσπέλασής της.

7.2 Μελλοντική εργασία

Έχοντας σαν δεδομένο ότι οι υλοποιήσεις μας δεν επηρέασαν την απόδοση του αποκωδικοποιητή ως προς το ποσοστό αναγνώρισης, περαιτέρω εργασία στον αλγόριθμο αφορά την ακόμα μεγαλύτερη επιτάχυνσή του με χρήση της αρχιτεκτονικής CUDA.

Η τελευταία υλοποίηση που ήταν και η καλύτερη, βασίστηκε στην τοποθέτηση σε κάθε μπλοκ τόσων νημάτων όσα είναι και τα πλαίσια της προς αναγνώριση πρότασης. Μια εναλλακτική υλοποίηση που θα μπορούσε να δοκιμάσει κανείς σε νεότερες κάρτες από τη GTX280 που είχαμε εμείς στη διάθεση μας, είναι να θέσει τον αριθμό των νημάτων ανά μπλοκ ίσο με τον αριθμό των senones. Αυτό προϋποθέτει την υποστήριξη από την κάρτα γραφικών, περισσοτέρων από 1024 νημάτων σε κάθε μπλοκ. Δεν υπάρχει κάποιος ελάχιστος αριθμός senones που μπορεί να περιέχει ένα ακουστικό μοντέλο, αλλά ένας τυπικός αριθμός είναι μεγαλύτερος του χίλια. Επομένως πρέπει να είμαστε σε θέση να υποστηρίξουμε τέτοια μεγέθη. Παραμένει ωστόσο αβέβαιο το αν αυτή η προσέγγιση θα μας παρείχε καλύτερο χρόνο εκτέλεσης.

Μία άλλη πιθανή βελτίωση θα μπορούσε να επιτευχθεί με την αποθήκευση του λογαριθμικού πίνακα αναζήτησης σε μία περιοχή μνήμης διαφορετική της καθολικής ή της texture που ήδη έχουν δοκιμαστεί. Δυστυχώς, οι υπάρχουσες δυνατότητες των καρτών γραφικών της Nvidia δεν επιτρέπουν μία τέτοια δοκιμή αφού το μέγεθος του πίνακα συνήθως είναι αποτρεπτικό.

Βιβλιογραφία

[1] R. Baken, *Clinical Measurement of Speech and Voice*. Taylor and Francis Ltd., 1987.

[2] J. Kingston, ``The Phonetics-Phonology Interface,'' *The Cambridge Handbook of Phonology*, 2007.

[3] D. Paul and J. Baker, ``The Design for the Wall Street Journal-Based Corpus,'' in *Proceedings ICSLP*, pp. 899--902, 1992.

[4] B. Juang and L. Rabiner, ``Hidden Markov Models for Speech Recognition,'' *Technometrics*, vol. 33, no. 3, pp. 251--272.

[5] J. Allen and L. Rabiner, ``A Unified Approach to Short-Time Fourier Analysis and Synthesis,'' in *Proceedings of the IEEE*, vol. 65, pp. 1558--1564, 1977.

[6] L. E. Baum and T. Petrie, ``Statistical Inference for Probabilistic Functions of Finite State Markov Chains,'' *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554--1563, 1966.

[7] Huang and et. al., *Spoken Language Processing*. Prentice Hall PTR.

[8] M. Ravishankar, ``Sphinx-3 s3.X Decoder(X=6),'' tech. rep., School of Computer Science, Carnegie Mellon University.

[9] T. Dunning, ``Statistical identification of language,'' tech. rep., New Mexico State University, 1994.

[10] G. S. Almasi and A. Gottlieb, *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc., 1989.

[11] Nvidia, *Nvidia CUDA C Programming Guide*, 3.2 ed., 2010.

[12] J. Sanders and E. Kandrot, *CUDA by Example*. Addison-Wesley, 2011.

Γλωσσάρι

Αλγόριθμος ομαλοποίησης smoothing algorithm.

Αναγνώριση φωνής και ταυτοποίησή της σε κείμενο speech to text recognition.

Δέσμη beam.

Δίγραμμο bigram.

Ευριστικές αναζήτησης με δέσμη beam search heuristics.

Εγγενείς συναρτήσεις intrinsic functions.

Εικονοκύτταρο pixel.

Ενεργό λεξιλόγιο effective vocabulary.

Κρυφή μνήμη cache memory.

Κρυμμένο μοντέλο Markov Hidden Markov Model (HMM).

Κατοχή occupancy.

Καθολική μνήμη global memory.

Κεντρική συνάρτηση main function.

Κλάδεμα pruning.

Κοινόχρηστη μνήμη shared memory.

Λέξη πλήρωσης filler.

Λεκτικό δέντρο lexical tree.

Μέθοδοι γραμμικής πρόβλεψης ελάχιστης φάσης όλων των πόλων all pole minimum-phase linear prediction(LPC) methods.

Μοντέλα οπισθοδρόμησης/μεταβαλλόμενης μέσης τιμής autoregressive/moving average models.

Μοντέλο Μείξης Γκαουσιανών Κατανομών Gaussian Mixture Model.

Μονόγραμμα unigram.

Νήμα thread.

Ξετύλιγμα βρόγχου unroll loop.

Ομάδα νημάτων warp.

Πυρήνας kernel.

Πίνακας αναζήτησης lookup table.

Παράλληλος υπολογισμός parallel computing.

Πλέγμα grid.

Πλαίσιο frame.

Τράπεζα μνήμης memory bank.

Τρίγραμμο trigram.

Υπολογιστές μαζικού υπολογισμού massively parallel computers.