ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Παράλληλη και Κατανεμημένη Υλοποίηση του SVP

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Σ. Ρίζος, 2454

ΕΠΙΒΛΕΠΩΝ: Κωνσταντίνος Δραζιώτης

ARISTOTLE UNIVERSITY OF THESSALONIKI
FACULTY OF SCIENCES
SCHOOL OF INFORMATICS

Parallel and distributed implementation of SVP

GRADUATE THESIS

Georgios S. Rizos, 2454

SUPERVISOR: Konstantinos Draziotis

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής

Υπεύθυνη Δήλωση

Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας, και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Πληροφορικής του Αριστοτέλειου Πανεπιστημίου Θεσσαλονίκης.

(Υπογραφή)

.....

Γεώργιος Σ. Ρίζος

Περίληψη

Στη σημερινή εποχή η ανθρώπινη ζωή δεν θεωρείται απλά συνυφασμένη με την τεχνολογία, αλλά στην πλειονότητα των περιπτώσεων εξαρτάται από αυτήν, σε τέτοιο βαθμό που η ψηφιακή ασφάλεια αποτελεί θεμελιώδη ανάγκη που επιζητά ο άνθρωπος στην καθημερινότητα του. Ειδικότερα, η κρυπτογραφία είναι η επιστήμη που κρατάει ασφαλείς τις ψηφιακές επικοινωνίες, διαφυλάττει τα προσωπικά δεδομένα, διασφαλίζει την ακεραιότητα των συναλλαγών, ενώ φαίνεται να επωμίζεται και το μέλλον της οικονομίας, φέρει γενικότερα εφαρμογές σε όλο το μήκος της ανθρώπινης ζωής διασφαλίζοντας τον ψηφιακό μας κόσμο.

Η τεχνολογία εξελίσσεται με φρενήρεις ρυθμούς. Η επέλαση των κβαντικών υπολογιστών θα αποτελέσει σημείο αναφοράς του άμεσου μέλλοντος, παρέχοντας τη δυνατότητα επίλυσης προβλημάτων των οποίων οι λύσεις θεωρούταν μέχρι σήμερα απροσέγγιστες. Ένας κβαντικός υπολογιστής δεν είναι πλήρως λειτουργικός σήμερα, λόγω εμποδίων υλικού που αφορούν θέματα μνήμης, αλλά δεν απέχει μακριά η ημέρα που θα εδραιωθεί η χρήση του. Χάρη σε επιστήμονες όπως ο Peter Shor είναι διαθέσιμοι πλέον κβαντικοί αλγόριθμοι, οι οποίοι επιλύουν το πρόβλημα του διακριτού λογαρίθμου και της παραγοντοποιήσης μεγάλων αριθμών σε κβαντικά συστήματα. Τα δύο αυτά προβλήματα αποτελούν τον πυρήνα των κρυπτοσυστημάτων που περιφρουρούν σήμερα τον ψηφιακό κόσμο, έτσι η άφιξη τους θα "αποκρυπτογραφήσει" την ανθρώπινη ζωή.

Η ανάγκη για ασφάλεια οδήγησε στην αναζήτηση νέων προβλημάτων, όπως αυτά που βασίζονται σε πλέγματα (Lattices, δικτυωτά) και θεωρούνται ισοδύναμα ως προς την επίλυση τους, τόσο σε συμβατικά, όσο σε κβαντικά συστήματα. Το Πρόβλημα του Εγγύτερου Διανύσματος (Shortest Vector Problen-SVP) σε Πλέγματα είναι αυτό που απασχολεί την παρούσα εργασία. Πιο συγκεκριμένα, στα πλαίσια της εργασίας αναπτύχθηκε λογισμικό που εστιάζει στην επίτευξη υψηλής απόδοσης, στοχεύοντας στην επίλυση του SVPγ. Ο σκοπός της προσπάθειας επίλυσης ενός τέτοιου προβλήματος, που θεωρητικά θα αποτελέσει την καρδιά των μελλοντικών κρυπτοσυστημάτων, έγκειται στα πλαίσια τεκμηρίωσης του βαθμού ασφαλείας του κρυπτοσυστήματος που μπορεί να παρέχει το πρόβλημα.

Το σενάριο υλοποίησης που αναπτύχθηκε στο παραχθέν λογισμικό προσομοιάζει σε πραγματικές συνθήκες τις προκλήσεις και τις ελλείψεις πόρων, που καλείται να αντιμετωπίσει ένας κακόβουλος χρήστης, προσδοκώντας να "σπάσει" το κρυπτοσύστημα. Η τεκμηρίωση της αναφερθείσας υπόθεσης εντοπίζεται στους περιορισμένους πόρους που βρίσκονται στη διάθεση του και στον τρόπο που δύναται να τους ελέγξει. Αυτοί είναι οι λόγοι που οδήγησαν στην επιλογή του αλγορίθμου απαρίθμησης (Enumeration) των Schnor και Euchner, ο οποίος αν και υστερεί σε θέματα απόδοσης συγκριτικά με τους αλγόριθμους

κοσκινίσματος (Sieving), έχει σημαντικά μικρότερες απαιτήσεις σε θέματα μνήμης, συνεπώς θεωρείται εφικτή η εκτέλεση του σε συστήματα καθημερινής χρήσης. Το κατανεμημένο σύστημα επίλυσης που υλοποιήθηκε φέρει πολλά κοινά με τις μεθόδους απόκτησης πόρων που εφαρμόζουν οι κακόβουλοι χρήστες, όπως αυτής της δημιουργίας botnet. Τέλος, η παραλληλοποίηση του αλγορίθμου βρίσκει εφαρμογή σε κάθε σύγχρονο υπολογιστικό σύστημα της τελευταίας δεκαετίας, τείνοντας να αξιοποιεί κάθε διαθέσιμο επεξεργαστικό του πόρο.

Λέξεις Κλειδιά. Κρυπτογραφία, Πλέγματα, Πρόβλημα μικρότερου Διανύσματος (SVP)

ABSTRACT

Nowadays, human life is not just considered intertwined with technology, but in most cases depends on it, to such an extent that digital security is a fundamental need that man seeks in his daily life. In particular, cryptography is the science that keeps digital communications secure, preserves personal data, ensures the integrity of transactions, while it seems to take on the future of the economy, generally brings applications throughout human life ensuring our digital world .

Technology is evolving at a frantic pace. The advent of quantum computers will be a point of reference for the immediate future, providing the ability to solve problems whose solutions were considered inaccessible until nowdays. A quantum computer is not fully functional today due to memory issues, however the day when its use will be established is not so far away. Thanks to scientists such as Peter Shor, quantum algorithms are now available that solve the problem of discrete logarithms and factorization of large numbers in quantum systems. These two problems are at the core of the cryptosystems that guard the digital world today, so their arrival will "decrypt" human life.

The need for security has led to the pursuit for new problems, such as lattice-based problems that are considered equivalent in solving by both conventional and quantum systems. The Shortest Vector Problem (SVP) in Lattices is the focus of this thesis. More specifically, in the context of this work, software was developed that focuses on achieving high performance, aiming at solving the SVPγ. The purpose of trying to solve such a problem, which in theory will be the heart of future cryptosystems, lies in documenting the degree of security of the cryptosystem that the problem may provide.

The implementation scenario developed in the produced software simulates in real conditions the challenges and the lack of resources, which a malicious user has to face, expecting to "break" the cryptosystem. The documentation of the mentioned case is located in the limited resources available to him and in the way he can control them. These are the reasons that led to the choice of the Enumeration algorithm by Schnor and Euchner, which, although lagging behind in performance compared to the sieving algorithms, has significantly lower memory requirements, so it is considered feasible in commodity systems. The distributed solution system implemented has a lot in common with the methods of obtaining resources used by malicious users, such as this botnet creation. Finally, the parallelization of the algorithm finds application in every modern computer system of the last decade, tending to utilize every available processing resource.

Key Words. Cryptography, Lattices, Shortest Vector Problem, SVP

Ευχαριστίες

Πρώτα απ΄ όλα, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή αυτής της πτυχιακής κ. Κωνσταντίνο Δραζιώτη, για την ευκαιρία που μου έδωσε, να ασχοληθώ με το συγκεκριμένο θέμα αλλά και για την έμπνευση και το ενδιαφέρον που μου καλλιέργησε, κατά τη διάρκεια των σπουδών μου.

Θα ήθελα επίσης να ευχαριστήσω όλους τους ανθρώπους που στάθηκαν δίπλα μου και συνέβαλαν κατά τρόπο ξεχωριστό, στο απαιτητικό αυτό διάστημα των σπουδών μου.

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

1.1 Προκλήσεις

Η ασφάλεια αποτελούσε ανέκαθεν κύριο μέλημα στη ζωή κάθε ανθρώπου, σήμερα διανύοντας την εποχή της ψηφιοποίησης η ασφάλεια μας βασίζεται στη κρυπτογραφία, καθώς είναι αυτή που μπορεί δώσει λύση στα περισσότερα θέματα ασφάλειας ψηφιακών συστημάτων. Κάθε προσωπικό ή ιατρικό δεδομένο, κάθε ψηφιακή επικοινωνία ή ψηφιακό έγγραφο, οτιδήποτε συνδέεται στο διαδίκτυο είναι εκτεθειμένο, εάν δεν έχει κρυπτογραφηθεί. Παράλληλα, η διαρκώς αυξανόμενη επεξεργαστική ισχύς θέτει διαρκώς σε κίνδυνο την ασφάλεια των κρυπτογραφικών αλγορίθμων που διαθέτουμε, έτσι δημιουργείται η ανάγκη εύρεσης νέων αλγόριθμων και νέων τεχνικών κρυπτογράφησης.

Τα σύγχρονα κρυπτογραφικά συστήματα αξιοποιούν δυσεπίλυτα μαθηματικά προβλήματα για την εφαρμογή των μεθόδων τους. Η έλευση των κβαντικών συστημάτων σηματοδοτεί την έναρξη της επίλυσης πολλών από αυτά, κάποια εκ των οποίων αποτελούν τη βάση των κρυπτογραφικών συστημάτων που χρησιμοποιούνται σήμερα. Έτσι δημιουργείται η ανάγκη εύρεσης προβλημάτων τα οποία εμφανίζουν ισοδύναμη πολυπλοκότητα επίλυσης, τόσο σε συμβατικούς ψηφιακούς επεξεργαστές όσο και σε κβαντικούς, ώστε να διατηρηθεί η ισότητα ανάμεσα στους χρήστες. Ένα τέτοιο πρόβλημα θεωρείται το πρόβλημα του μικρότερου διανύσματος σε πλέγματα (Shortest Vector Problem) που χρησιμοποιείται στην Post Quantum Cryptography.

Η επίλυση του Shortest Vector Problem απασχολεί πληθώρα ερευνητών σε διεθνές επίπεδο αρκετές δεκαετίες. Τα τελευταία χρόνια έχει κεντρίσει το ενδιαφέρον της διεθνής κοινότητας κυβερνοασφάλειας, λόγω της εφαρμογής του προβλήματος στη κρυπτογραφία συστημάτων ανθεκτικών σε επιθέσεις με

κβαντικούς υπολογιστές, με ανοιχτούς διαγωνισμούς επίλυσης του προβλήματος με συμμετοχές κορυφαίων επιστημόνων ώστε να εκτιμηθεί η αντοχή των αλγορίθμων που διέπουν το πρόβλημα. Η εφαρμογή αυτών των αλγορίθμων στα σύγχρονα κρυπτοσυστήματα παρέχει τη λύση στο πρόβλημα της ασφάλειας, κατά την ευρεία λειτουργία κβαντικών συστημάτων στο μέλλον.

1.2 Συνεισφορά

Στα πλαίσια της παρούσας πτυχιακής εργασίας, αναπτύσσεται μία προσπάθεια επίλυσης του Shortest Vector Problem παραλλάσοντας τον αλγόριθμο απαρίθμησης (enumeration algorithm) των Schnorr και Euchner, ο οποίος χρησιμοποιείται ως βασική υπορουτίνα στον αλγόριθμο αναγωγής BKZ. Η παραλλαγή του αλγορίθμου αναπτύχθηκε με κύριο μέλημα την εκμετάλλευση της επεξεργαστικής δύναμης που παρέχει η εφαρμογή παράλληλων και κατανεμημένων αλγορίθμων.

Η βασική ιδέα παραλληλοποίησης και κατανομής του προβλήματος στηρίζεται στο διαχωρισμό του δένδρου απαρίθμησης (enumeration tree) σε επιμέρους υποδένδρα αναζήτησης για τη εύρεση της λύσης του προβλήματος σ' αυτά, εφαρμόζοντας προηγμένες τεχνικές παράλληλου και ασύγχρονου προγραμματισμού. Διάφορες μελέτες[11] που έχουν δημοσιευθεί στο παρελθόν, εφαρμόζουν τεχνικές παράλληλης επεξεργασίας με τη χρήση κάρτας γραφικών (GPU) σε περιβάλλον CUDA, ενώ εδώ μελετάτε η παράλληλη επεξεργασία στη κεντρική μονάδα επεξεργασίας (CPU) του συστήματος. Η αρχιτεκτονική του κατανεμημένης συστήματος υλοποίησης δίνει επιπλέον τη δυνατότητα προσθαφαίρεσης κόμβων του δικτύου που συμμετέχουν στην επίλυση του προβλήματος ανεξαρτήτως φάσης της διεργασίας επίλυσης.

Η επιλογή του αλγορίθμου απαρίθμησης έγινε με κριτήριο τη χαμηλή απαίτησή του σε επίπεδο μνήμης, σε σύγκριση με τους αλγόριθμους κοσκινίσματος (sieving algorithms), που ενώ υπερτερούν σε επίπεδο πολυπλοκότητας, έχουν εκθετικά μεγαλύτερες απαιτήσεις μνήμης. Το γεγονός αυτό, σε συνδυασμό με την ευκολία της προσθαφαίρεση κόμβων, συνεπάγεται στο ότι μπορεί να συμβάλει στην επίλυση του προβλήματος οποιοσδήποτε επεξεργαστής είναι συνδεδεμένος στο δίκτυο και έχει δυνατότητες παράλληλης επεξεργασίας, δεχόμενος εντολές από έναν κεντρικό κόμβο.

Στόχος της εργασίας, είναι η εξέταση την αντοχής του προβλήματος αυτού, εξετάζοντας ένα κατανεμημένο σενάριο επίθεσής του. Στο σενάριο που υλοποιείται στην εργασία, θεωρείται ένας κεντρικός επιτιθέμενος κόμβος-σύστημα ο οποίος αφού διασπά το πρόβλημα σε επιμέρους προβλήματα, διαμοιράζει τα υποπροβλήματα στους κατανεμημένους κόμβους και αυτά με παράλληλη επεξεργασία αναζητούν την λύση του. Το δίκτυο αυτό μπορεί να θεωρηθεί ως ένα botnet network κατά το οποίο πολλά ανεξάρτητα

κατανεμημένα συστήματα ελέγχονται από έναν κεντρικό σύστημα, υπακούοντας απομακρυσμένα στις εντολές του.

1.3 Οργάνωση της εργασίας

Η παρούσα πτυχιακή εργασία είναι οργανωμένη σε πέντε κεφάλαια. Στο κεφάλαιο 2 παρουσιάζουμε το μαθηματικό υπόβαθρο γύρω από τα πλέγματα και τα προβλήματα που ορίζονται βάσει αυτών. Έπειτα στο 3ο κεφάλαιο παρουσιάζεται αναλυτικότερα το πρόβλημα του συντομότερου διανύσματος και η μεθοδολογία επίλυσής του. Στο κεφάλαιο 4 έχουμε την περιγραφή του συστήματος που υλοποιήσαμε καθώς και την παρουσίαση των μετρήσεών μας. Τέλος στο κεφάλαιο 5 καταθέτουμε τα συμπεράσματά μας, συνοψίζουμε τη μελέτη που έχει λάβει χώρα και προτείνουμε πιθανές επεκτάσεις.

ΚΕΦΑΛΑΙΟ 2

Μαθηματικό Υπόβαθρο

2.1 Πλέγματα

Ορισμός 1 *: **Πλέγμα** (ή δικτυωτό) ονομάζουμε ένα σύνολο σημείων L του n-διάστατου χώρου $\mathbb{R}^n$, τα οποία προκύπτουν ως ακέραιος γραμμικός συνδυασμός μιας σειράς γραμμικώς ανεξάρτητων διανυσμάτων $\{\boldsymbol{b}_i\}_{i=1}^{k}, (k \leq n)$.*

$$L(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) = \left\{ \sum_{i=1}^{k} x_i \cdot \boldsymbol{b}_i, x_i \in \mathbb{Z} \right\} = \left\{ \boldsymbol{xB}, \boldsymbol{x} \in \mathbb{Z}^k \right\}$$

Τα διανύσματα $\{\boldsymbol{b}_i\}$ αποτελούν τις γραμμές του πίνακα B, διάστασης $k \times n$ και ονομάζονται βάση του πλέγματος. Λέμε ότι μία βάση παράγει ένα πλέγμα, αλλά επίσης ισχύει ότι ένα πλέγμα μπορεί να παράγεται, από παραπάνω από μία βάσεις. Σε κάθε περίπτωση όλες οι βάσεις ενός πλέγματος έχουν τον ίδιο αριθμό στοιχείων k, που ονομάζεται τάξη (rank). Ο αριθμός n των στοιχείων που απαρτίζουν κάθε διάνυσμα ονομάζεται διάσταση του πλέγματος. Αν $n = k$ τότε το πλέγμα καλείται μέγιστης τάξης (full rank lattice). Στην παρούσα πτυχιακή εργασία ασχολούμαστε με πλέγματα αυτής της κατηγορίας.

Ορισμός 2 *: **Ορίζουσα** (determinant) ενός πλέγματος $L(\boldsymbol{B})$ καλούμε την ποσότητα:*

$$det(L(\boldsymbol{B})) = \sqrt{det(\boldsymbol{B}^T\boldsymbol{B})}$$

και στην ειδική περίπτωση των πλεγμάτων μέγιστης τάξης ισχύει:

$$det(L(\boldsymbol{B})) = |det(\boldsymbol{B})|$$

Δηλαδή η ορίζουσα του πλέγματος ισούται με την απόλυτη τιμή της ορίζουσας του πίνακα βάσης αυτού. Ωστόσο, αξίζει να σημειωθεί ότι η ορίζουσα αυτή είναι ανεξάρτητη από την επιλογή της βάσης.

Τέλος με τον όρο, **ελάχιστο** ενός δικτυωτού L, αναφερόμαστε στη νόρμα του μικρότερου μη-μηδενικού διανύσματος και συμβολίζεται $\lambda_1(L)$.

$$\lambda_1(L) = \inf\{\|\mathbf{x}\| : \mathbf{x} \in L - \{\mathbf{0}\}\}$$

2.2 n-διάστατη Μπάλα

Σε αυτή την παράγραφο παραθέτουμε τα βασικά μεγέθη που χαρακτηρίζουν μια $n-$διάστατη μπάλα του $\mathbb{R}^n$. Πρόκειται για μια μαθηματική έννοια, που θα μας βοηθήσει στη συνέχεια να προχωρήσουμε την ανάλυσή μας.

Ορισμός 3 *: Ως $n-$**διάστατη Μπάλα**, ορίζουμε το σύνολο των σημείων που περικλείονται από μία σφαίρα. Δηλαδή μια κλειστή μπάλα ακτίνας $R > 0$ με κέντρο την αρχή των αξόνων $\mathbf{0}$ είναι το σύνολο:*

$$B_n(R) = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| \leq R\}$$

Εν γένει ο όγκος της $n-$διάστατης μπάλας και η επιφάνεια της $(n-1)-$διάστατης σφαίρας που την περικλείει είναι αναλογικά της ακτίνας R, εις τη διάσταση του χώρου. Συνηθίζουμε να γράφουμε $V_n(R) = V_n R^n$ και $S_n(R) = S_n R^n$, όπου $V_n = V_n(1)$, $S_n = S_n(1)$, δηλαδή ο όγκος και το εμβαδόν της μοναδιαίας n-διάστατης μπάλας/σφαίρας, αντίστοιχα. Στη συνέχεια δίνουμε τους κλειστούς τύπους αυτών των μεγεθών [2].

Όγκος $n-$διάστατης μπάλας:

$$V_n(R) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} R^n \tag{1}$$

Επιφάνεια περιβάλλουσας $n-$διάστατης σφαίρας:

$$S_n(R) = \frac{2\pi^{\frac{n+1}{2}}}{\Gamma(\frac{n+1}{2})} R^n \tag{2}$$

με τη συνάρτηση γάμμα να ορίζεται ως:

$$\Gamma(s) = \int_0^\infty x^{s-1} e^{-x} dx, s > 0$$

2.3 Πρόβλημα του μικρότερου διανύσματος

Τα κυριότερα προβλήματα που ανακύπτουν στα πλέγματα και στα οποία μπορεί να βασιστεί ένα κρυπτογραφικό σύστημα, είναι το πρόβλημα της εύρεσης του μικρότερου (SVP) καθώς και του εγγύτερου διανύσματος (CVP). Παρουσιάζουμε αναλυτικά αυτά τα προβλήματα στη συνέχεια.

Ορισμός 4 *: **Shortest Vector Problem (SVP):** Δοθέντος ενός πλέγματος L τάξης n, να βρεθεί το μικρότερο μη μηδενικό διάνυσμα $\boldsymbol{v}$. Δηλαδή αναζητούμε το διάνυσμα $\boldsymbol{v}$ ώστε:*

$$\|\boldsymbol{v}\| = \lambda_1(L)$$

με $\lambda_1(L)$ συμβολίζουμε το μέτρο του ελάχιστου διανύσματος.

Αποδεικνύεται ότι τέτοιο διάνυσμα υπάρχει και μάλιστα δεν εξαρτάται από την επιλογή της βάσης του πλέγματος. Ορίζουμε επίσης και την προσεγγιστική εκδοχή του συγκεκριμένου προβλήματος (SVP_γ), όπου αναζητούμε όχι το μικρότερο πλέον, αλλά ένα διάνυσμα που δεν υπερβαίνει το μικρότερο κατά έναν πολλαπλασιαστικό παράγοντα γ, δηλ:

$$\|\boldsymbol{v}\| \leq \gamma \cdot \lambda_1(L), \gamma \geq 1$$

Ο Ajtai απέδειξε ότι το SVP είναι NP-hard υπό τυχαίες αναγωγές [4]. Για την επίλυση του συγκεκριμένου προβλήματος, δύο οικογένειες αλγορίθμων είναι γνωστές. Οι αλγόριθμοι που βασίζονται σε απαρίθμηση (enumeration algorithms) και οι αλγόριθμοι κοσκινίσματος (sieving algorithms).

2.4 Πρόβλημα του εγγύτερου διανύσματος

Ορισμός 5 *: **Closest Vector Problem (CVP):** Δοθέντος ενός πλέγματος L τάξης n, μιας απόστασης M, καθώς και ενός διανύσματος στόχου $\boldsymbol{t} \in \mathbb{R}^n$, να βρεθεί το πλησιέστερο στο $\boldsymbol{t}$, διάνυσμα του πλέγματος L. Δηλαδή αναζητούμε το $\boldsymbol{v}_* \in L$ ώστε:*

$$M(\boldsymbol{v}_* - \boldsymbol{t}) = \min_{\boldsymbol{v}} M(\boldsymbol{v} - \boldsymbol{t})$$

Αντίστοιχα και με το προηγούμενο πρόβλημα, δίνουμε την προσεγγιστική εκδοχή CVP_γ, όπου πλέον αναζητούμε ένα διάνυσμα $v \in L$ ώστε:

$$M(\boldsymbol{v}_* - \boldsymbol{t}) \leq \gamma \cdot \min_{\boldsymbol{v}} M(\boldsymbol{v} - \boldsymbol{t}), \gamma \geq 1$$

Το CVP είναι επίσης υπολογιστικά δυσεπίλυτο και ανήκει στην κλάση των NP-hard προβλημάτων. Τέλος συνηθέστερα στο CVP ως συνάρτηση απόστασης χρησιμοποιείται η Ευκλείδεια.

2.5 Θεώρημα του Minkowski

Από τους παραπάνω ορισμούς για το πρόβλημα του μικρότερου διανύσματος SVP, γεννάται το ερώτημα, ποιο είναι αυτό το ελάχιστο μήκος του διανύσματος που αναζητούμε. Η ακριβής εύρεση αυτού είναι μια εξαιρετικά δύσκολη υπόθεση. Σε αυτό έρχεται να μας βοηθήσει το θεώρημα Minkowski.

Θεώρημα 1 *(1ο Θεώρημα του Minkowski): Δοθέντος ενός πλέγματος διάστασης n, μιας ακτίνας $R > 0$ και της μοναδιαίας n-διάστατης σφαίρας του $\mathbb{R}^n$, της οποίας ο όγκος ικανοποιεί τη σχέση:*

$$V_n R^n > det(L) \tag{3}$$

υπάρχει διάνυσμα $\boldsymbol{v} \in L - \{\mathbf{0}\}$ με:

$$\|\boldsymbol{v}\| \leq 2R \tag{4}$$

Από τις εξισώσεις (3) και (4) γίνεται φανερό πως μία n-διάστατη μπάλα με ακτίνα τουλάχιστον:

$$R_o = 2\left(\frac{\det L}{V_n}\right)^{1/n} \overset{(1)}{=} 2\frac{(\det L)^{1/n}\Gamma(\frac{n}{2}+1)^{1/n}}{\sqrt{\pi}} \tag{5}$$

περιέχει ένα μη μηδενικό διάνυσμα του L.

Το Θεώρημα Minkowski καταφέρνει λοιπόν με αυτόν τον τρόπο να συνδέσει την ορίζουσα του πλέγματος, det(L), με το μήκος του μικρότερου διανύσματος του L.

2.6 Ευρετική του Gauss

Η ποσότητα που μόλις αναφέραμε προσεγγίζεται από την Ευρετική του Gauss:

Ορισμός 6 *: Ορίζουμε ως **Gaussian Heuristic** τον θετικό πραγματικό αριθμό:*

$$GH(L) = \left(\frac{\det L}{V_n}\right)^{1/n} \overset{(1)}{=} \frac{(\det L)^{1/n}\Gamma(\frac{n}{2}+1)^{1/n}}{\sqrt{\pi}} \approx \sqrt{\frac{n}{\pi e}}(\det L)^{1/n} \tag{6}$$

Αποδεικνύεται ότι μεταξύ του ελαχίστου ενός πλέγματος και της Gaussian Ευρετικής ισχύει η σχέση:

$$\lambda_1(L) < 2GH(L)$$

Ορισμός 7 *: (Ευρετική υπόθεση Gauss). Για τυχαίο ακέραιο πλέγμα L τάξης k και διάστασης n, θεωρούμε ότι:*

$$\lambda_1(L) \approx GH(L)$$

Η ευρετική του Gauss δεν ισχύει για όλα τα ακέραια πλέγματα, όμως ουσιαστικά πετυχαίνει να μειώσει στο μισό την ακτίνα μιας μπάλας με κέντρο την αρχή, προκειμένου εκείνη να περιέχει ένα μη μηδενικό σημείο του πλέγματος L. Συνεπώς ο χώρος στον οποίο χρειάζεται να αναζητήσουμε τέτοια διανύσματα περιορίζεται αισθητά.

2.7 Αλγόριθμος Αναγωγής LLL

Σε αυτή την παράγραφο περιγράφουμε τη λειτουργία του αλγορίθμου LLL. Ο αλγόριθμος αυτός χρησιμοποιείται προκειμένου να παράγει μια νέα ανηγμένη βάση. Αναλυτικότερα, δεδομένης μίας βάσης, μας επιστρέφει μία ισοδύναμη που όμως τα διανύσματα αυτής, έχουν καλύτερες ιδιότητες, δηλαδή είναι κατά το δυνατόν μικρότερα και ανά δύο, σχεδόν κάθετα.

Επειδή λοιπόν ο αλγόριθμος στοχεύει στην εύρεση των μικρότερων διανυσμάτων, μπορεί να χρησιμοποιηθεί και για την λύση της προσεγγιστικής εκδοχής του SVP. Μάλιστα ο αλγόριθμος LLL μπορεί να δώσει προσεγγιστική λύση σε πολυωνυμικό χρόνο ως προς τα μήκη των διανυσμάτων της βάσης όταν $\gamma = 2^{n/2}$. Την ίδια διαδικασία επιτελεί και ο αλγόριθμος BKZ που μάλιστα παράγει καλύτερης ποιότητας βάσεις, για μεγαλύτερες διαστάσεις, με αυξημένο υπολογιστικό κόστος όμως, καθώς είναι εκθετικής πολυπλοκότητας ως προς τη διάσταση του πλέγματος.

2.7.1 Ορθοκανονικοποίηση Gram-Schmidt

Για την ορθοκανονικοποίηση των διανυσμάτων ο αλγόριθμος LLL χρησιμοποιεί τη διαδικασία Gram-Schmidt, η οποία βασίζεται στην έννοια της προβολής. Όπως γνωρίζουμε η προβολή ενός διανύσματος $\boldsymbol{u}$ σε ένα άλλο $\boldsymbol{v}$, συμβολίζεται ως $proj_{\boldsymbol{v}}\boldsymbol{u}$ και ισχύει $proj_{\boldsymbol{v}}\boldsymbol{u} = \lambda\boldsymbol{v}, \lambda \in \mathbb{R}$. Αν τα διανύσματα είναι κάθετα, η προβολή είναι μηδενική. Σε κάθε άλλη περίπτωση έχουμε $\lambda \neq 0$.

Γνωρίζουμε ότι τα διανύσματα $proj_{\boldsymbol{v}}\boldsymbol{u} - \boldsymbol{u}$ και αυτό της $proj_{\boldsymbol{v}}\boldsymbol{u}$ είναι κάθετα μεταξύ τους συνεπώς:

$$(proj_{\boldsymbol{v}}\boldsymbol{u} - \boldsymbol{u}) \cdot proj_{\boldsymbol{v}}\boldsymbol{u} = (\lambda\boldsymbol{v} - \boldsymbol{u}) \cdot \lambda\boldsymbol{v} = 0$$

Λύνοντας ως προς $\lambda \neq 0$ έχουμε:

$$\lambda = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{||\boldsymbol{v}||^2}$$

οπότε το διάνυσμα προβολής γράφεται:

$$proj_{\boldsymbol{v}}\boldsymbol{u} = \frac{\boldsymbol{u}\cdot\boldsymbol{v}}{\boldsymbol{v}\cdot\boldsymbol{v}}\boldsymbol{v}$$

Τώρα προκειμένου να παράξουμε μία ορθογώνια βάση $B^* = \{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*\}$, δοθείσης μία αρχικής $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$, ορίζουμε το εξής απλό σχήμα:

$$\boldsymbol{b}_1^* = \boldsymbol{b}_1, \boldsymbol{b}_2^* = \boldsymbol{b}_2 - proj_{\boldsymbol{b}_1^*}\boldsymbol{b}2$$

Εύκολα παρατηρούμε ότι $\boldsymbol{b}_1^* \cdot \boldsymbol{b}_2^* = 0$. Στη γενική περίπτωση ορίζουμε τα διανύσματα ως:

$$\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} proj_{\boldsymbol{b}_j^*}\boldsymbol{b}_i$$

και θέτοντας

$$\mu_{i,j} = \frac{\boldsymbol{b}_i\boldsymbol{b}_j^*}{\boldsymbol{b}_j^*\boldsymbol{b}_j^*}, i > j$$

έχουμε,

$$\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\boldsymbol{b}_j^*$$

Επαγωγικά μπορούμε να επαληθεύσουμε τώρα ότι $\boldsymbol{b}_i^* \cdot \boldsymbol{b}_j^* = 0$ για κάθε i,j με $i \neq j$. Έχουμε λοιπόν μία ορθοκανονικοποιημένη βάση B^* κατά Gram-Schmidt.

Algorithm 2.1: Gram-Schmidt Algorithm

Είσοδος: Μία βάση $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ του $\mathbb{R}^n$.
΄Εξοδος: Μία ορθοκανονικοποιημένη βάση $B^* = \{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*\}$ και οι συντελεστές GSO $\mu_{i,j}$.

$\boldsymbol{b}_1^* \leftarrow \boldsymbol{b}_1$
for $i = 2, \ldots, n$ **do**
  for $j = 1, \ldots, i-1$ **do**
    $\mu_{i,j} = \frac{\boldsymbol{b}_i\boldsymbol{b}_j^*}{\boldsymbol{b}_j^*\boldsymbol{b}_j^*}$
    $\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\boldsymbol{b}_j^*$
  end
end
return $B^* = \{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*\}, \mu = (\mu_{i,j})_{i,j}$

2.7.2 Ο αλγόριθμος LLL

Στο σημείο αυτό, παραθέτουμε πρώτα τον ορισμό μιας LLL-ανηγμένης βάσης και στη συνέχεια σε μορφή ψευτοκώδικα τον αλγόριθμο που δημιουργεί μία τέτοια βάση.

Ορισμός 8 *Έστω μία βάση $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ ενός πλέγματος $L \in \mathbb{R}^n$. Η βάση καλείται LLL-ανηγμένη αν υπάρχουν παράμετροι δ, η με $1/4 < \delta \leq 1$ και $1/2 \leq \eta < \sqrt{\delta}$, ώστε να είναι συρρικνωμένη στο μέγεθος κατά τον παράγοντα η, και επιπλέον να ικανοποείται η συνθήκη του Lovász:*

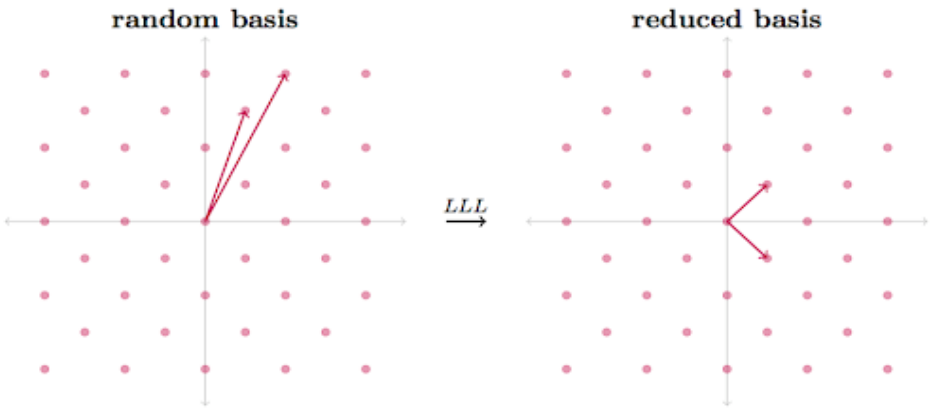
$$\delta \|\boldsymbol{b}_i^*\|^2 \leq \|\boldsymbol{b}_{i+1}^* + \mu_{i+1,i}\boldsymbol{b}_i^*\|^2, \ 1 < i < n$$

Ουσιαστικά ο αλγόριθμος χωρίζεται σε δύο στάδια που επαναλαμβάνονται επαναληπτικά. Κατά το πρώτο έχουμε τη συρρίκνωση των μεγεθών των διανυσμάτων, ενώ στο δεύτερο ελέγχεται αν η βάση που προέκυψε, ικανοποιεί τη συνθήκη του Lovász, διαφορετικά τα διανύσματα αντιμετατίθενται και η διαδικασία επαναλαμβάνεται.

Algorithm 2.2: LLL Algorithm

Είσοδος: Μία βάση $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ ενός πλέγματος
Έξοδος: Μία ανηγμένη βάση $B^* = \{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*\}$

$B^*, \mu = gso(B)$ # Calculate Gram-Schmidt base and corresponding coefs
$i = 2$
while $i \leq n$ **do**
  for $j = i - 1, \ldots, 1$ **do**
    $\boldsymbol{b}_i^* = \boldsymbol{b}_i^* - \lceil \mu_{i,j} \rfloor \boldsymbol{b}_j^*$
    Update $\mu_{i,j}, 1 \leq j \leq n$
  end
  if $\delta \|\boldsymbol{b}_i^*\|^2 > \|\mu_{i+1,i}\boldsymbol{b}_i^* + \boldsymbol{b}_{i+1}^*\|^2$ **then**
    swap$(\boldsymbol{b}_i^*, \boldsymbol{b}_{i+1}^*)$
    $i = \max(2, i - 1)$
    $B^*, \mu = gso(B)$
  else
    $i = i + 1$
  end
end
return $B^* = \{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*\}$



Σχήμα 1: Σχηματική απεικόνιση της αναγωγής βάσης κατά LLL, όπου διακρίνεται τόσο η ορθογωνιότητα των διανυσμάτων, όσο και η συρρίκνωσή τους.

Ιδιαίτερη σημασία για τη χρήση του αλγορίθμου LLL έχει το παρακάτω θεώρημα, βάση του οποίου ο αλγόριθμος επιτυγχάνει πολυωνυμική σύγκλιση.

Θεώρημα 2 : *Για μία n-διάστατη βάση* $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$, *ο αλγόριθμος LLL με* $\delta = 3/4$ *έχει πολυωνιμική πολυπλοκότητα ως προς το μήκος των διανυσμάτων της βάσης και συγκεκριμένα:* $O(n^4 log_2 M)$, *όπου* $M = \max_i(\|\boldsymbol{b}_i\|^2)$.

Απόδειξη. [16]

ΚΕΦΑΛΑΙΟ 3

Πρόβλημα Εύρεσης Μικρότερου Διανύσματος

Στην παρούσα πτυχιακή εργασία όπως έχουμε ήδη αναφέρει, ασχολούμαστε με την επίλυση του προβλήματος μικρότερου διανύσματος SVP. Προηγουμένως, αναφερθήκαμε σε αλγορίθμους που προσεγγίζουν το πρόβλημα ως αναγωγή της δοθείσας βάσης (lattice reduction).

Συγκεκριμένα αυτοί οι αλγόριθμοι, επιχειρούν διαδοχικούς μετασχηματισμούς στη βάση του πλέγματος, προκειμένου να καταλήξουν με μικρότερα και όσο το δυνατόν ορθογώνια διανύσματα, δίνοντας έτσι απάντηση και στο SVP_γ. Μολονότι για συνήθεις παραμέτρους απαιτούν πολυωνυμικό χρόνο, ο παράγοντας προσέγγισης γ που πετυχαίνουν είναι ασυμπτωτικά εκθετικός. Στην υλοποίησή μας, επιλέξαμε να χρησιμοποιήσουμε μία διαφορετική αλγοριθμική τεχνική, αυτή της απαρίθμησης. Στη συνέχεια παρουσιάζουμε τη συγκεκριμένη οικογένεια αλγορίθμων.

3.1 Αλγόριθμος απαρίθμησης KFP

Δοθείσας μίας βάσης $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ και ενός άνω ορίου R για το ελάχιστο μήκος διανύσματος $\lambda_1(L)$, αναζητούμε όλα τα διανύσματα $\boldsymbol{v} \in L - \{\mathbf{0}\}$ που ικανοποιούν τη συνθήκη $\|\boldsymbol{v}\| \leq R$.

Για το σκοπό αυτό, ο αλγόριθμος δημιουργεί ένα δέντρο απαρίθμησης (enumeration tree) που αποτελείται από όλα τα διανύσματα των πλεγμάτων προβολής $\pi_n(L), \pi_{n-1}(L), \ldots, \pi_1(L)$, μέτρου το πολύ R. Πιο συγκεκριμένα, το δέντρο απαρίθμησης έχει βάθος n και σε κάθε επίπεδο k βρίσκονται όλα τα διανύσματα του πλέγματος $\pi_{n+1-k}(L)$, υπό τον περιορισμό πάντοτε ότι η νόρμα τους δεν ξεπερνά το όριο R. Στη ρίζα του δέντρου βρίσκεται το μηδενικό διάνυσμα $\mathbf{0} = \pi_{n+1}(L)$. Η διάταξη αυτή των διαδοχικών προβολών, μας

εξασφαλίζει ότι όλοι οι πρόγονοι ενός κόμβου, έχουν το πολύ το ίδιο μήκος με τον κόμβο που έπεται. Δηλαδή:

$$\|\pi_n(\boldsymbol{v})\|^2 \leq \|\pi_{n-1}(\boldsymbol{v})\|^2 \leq \ldots \leq \|\pi_1(\boldsymbol{v})\|^2 = \|\boldsymbol{v}\|^2 \leq R^2$$

Επίσης ένα διάνυσμα $\boldsymbol{v}$ μπορεί να γραφτεί ως $\boldsymbol{v} = u_1\boldsymbol{b}_1 + \cdots + u_n\boldsymbol{b}_n$, όπου

$$\boldsymbol{b}_i = \boldsymbol{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j}\boldsymbol{b}_j^*$$

Δηλαδή:

$$\boldsymbol{v} = \sum_{j=1}^{n}(u_j + \sum_{i=j+1}^{n} \mu_{i,j}u_i)\boldsymbol{b}_j^*,$$

οπότε και για τις νόρμες των προβολών έχουμε:

$$\|\pi_{n+1-k}(\boldsymbol{v})\|^2 = \sum_{j=n+1-k}^{n}\left(u_j + \sum_{i=j+1}^{n} \mu_{i,j}u_i\right)^2 \|\boldsymbol{b}_j^*\|^2, 1 \leq k \leq n. \quad (7)$$

Προκύπτει λοιπόν ένα σύνολο n ανισοτήτων. Αν ξεκινήσουμε από το $\pi_n(\boldsymbol{v})$, το οποίο ορίζεται μόνο από το διάνυσμα βάσης $\boldsymbol{b}_n$, μπορούμε να αποφανθούμε για το διάστημα στο οποίο επιτρέπεται να λάβει τιμές, ο αντίστοιχος συντελεστής του διανύσματος αυτού. Έχοντας βρει (απαριθμήσει) τις ακέραιες πιθανές τιμές για τον πρώτο συντελεστή, μπορούμε να προχωρήσουμε στη δεύτερη ανισότητα και να εξασφαλίσουμε τις τιμές και για τον δεύτερο.

Η διαδικασία επαναλαμβάνεται διαδοχικά για όλες τις ανισότητες, οπότε και προκύπτει το δέντρο απαρίθμησης με όλους τους πιθανούς συντελεστές για κάθε $\boldsymbol{b}_i$, ανά επίπεδο. Ειδικά για τον πρώτο συντελεστή περιορίζουμε την αναζήτηση μόνο σε θετικές τιμές, καθώς διαφορετικά θα οδηγούμασταν σε ένα συμμετρικό δέντρο, λόγο του ότι, αν $\boldsymbol{x} \in L$ και $-\boldsymbol{x} \in L$.

Σημειώνουμε, ότι το πλήθος των κόμβων του δέντρου, εξαρτάται και από την ποιότητα της βάσης, για αυτό συνηθέστερα πριν την εφαρμογή του αλγορίθμου απαρίθμησης προηγείται κάποια αναγωγή βάσης, όπως η LLL.

Η διάσχιση τώρα του δέντρου απαρίθμησης γίνεται κατά βάθος, με χρήση του αλγορίθμου DFS, ώστε να έχουμε μειωμένες απαιτήσεις μνήμης. Αν ο αλγόριθμος καταλήξει σε κάποιο φύλλο του δέντρου, επιστρέφει το σχηματιζόμενο διάνυσμα, υπό την προϋπόθεσή ότι το μήκος του δεν ξεπερνά το όριο R. Η εκτέλεση του αλγορίθμου μπορεί να είναι αποτυχημένη, αν δεν βρεθεί κάποιο άλλο διάνυσμα πέρα από το μηδενικό. Στη συνέχεια παραθέτουμε σε μορφή ψευτοκώδικα τον αλγόριθμο που μόλις περιγράψαμε.

Όσο αφορά την πολυπλοκότητα του αλγορίθμου KFP, αυτή είναι πολυωνυμική ως προς το πλήθος N των κόμβων του δέντρου απαρίθμησης. Αποδεικνύεται [9] ότι ένα άνω όριο για το πλήθος είναι αυτό $N \leq n^{n/(2e)+o(n)}$ και ακόμα ότι υπάρχει πολυώνυμο $p(x, y) \in \mathbb{R}[x, y]$, ώστε για κάθε πλέγμα τάξης n, διάστασης m και με συντελεστές βάσης που φράσονται από το B, η πολυπλοκότητα του αλγορίθμου να είναι:

$$p(log_2(B), m)n^{n/(2e)+o(n)}$$

3.2 Βελτιστοποιήσεις του αλγορίθμου απαρίθμησης

Σε αυτήν παράγραφο αναφέρουμε κάποιες σημαντικές βελτιστοποιήσεις που έχουν προταθεί για τον αλγόριθμο KFP. Αρχικά οι Schnorr-Euchner πρότειναν ο συντελεστής x_i αντί να λαμβάνει διαδοχικά τιμές από την αρχή του διαστήματος I_i , να εκκινεί από την ποσότητα: $\lceil \sum_{j=i+1}^{n} \mu_{j,i} x_j \rfloor$, που βρίσκεται περίπου στη μέση του διαστήματος I_i και κατόπιν να αυξάνεται είτε να μειώνεται κατά 1. Η συγκεκριμένη παραλλαγή ονομάζεται και αλγόριθμος απαρίθμησης των Schnorr-Euchner.

Μία άλλη παραλλαγή του βασικού αλγορίθμου, εισάγει κάποιους όρους φθοράς $a_i \in (0, 1)$ με τους οποίους αναπροσαρμόζουμε το όριο αποδοχής ως $R_i = a_i R$ για κάθε βάθος i. Η παραλλαγή αυτή ονομάζεται και απαρίθμηση με κλάδεμα. Στόχος είναι να περιορίσουμε το υπολογιστικό κόστος, μειώνοντας το χώρο αναζήτησης διανυσμάτων. Ωστόσο, είναι πιθανό ο αλγόριθμός μας να μην επιστρέψει κάποια λύση, παρόλο που μπορεί να υπάρχει διάνυσμα με μήκος μικρότερου του R.

Algorithm 3.1: KFP Algorithm

Είσοδος: Μία βάση $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ του $\mathbb{R}^n$, η οποία ορίζει το πλέγμα $L(B)$ και ένα θετικό αριθμό R.
΄Εξοδος: Τα διανύσματα $\boldsymbol{x} \in L$ με $\|\boldsymbol{x}\| \leq R$.

$B^*, \mu = LLL(B)$ # Calculate a reduced base with an algorithm such as LLL
$\boldsymbol{x} = (x_i) \leftarrow \boldsymbol{0}_n, c = (c_i) \leftarrow \boldsymbol{0}_n, l = (l_i) \leftarrow \boldsymbol{0}_n$
$sumli \leftarrow 0, S = \emptyset, i \leftarrow 1$
while $i \leq n$ **do**
  $c_i \leftarrow -\sum_{j=i+1}^{n} x_j \mu_{ji}$
  $l_i \leftarrow B_i (x_i - c_i)^2$
  $sumli \leftarrow \sum_{j=1}^{n} l_j$
  if $sumli \leq R^2$ **then**
    if $i = 1$ **then**
      $S \leftarrow S \cup \{\sum_{j=1}^{n} x_j b_j\}$
      $x_1 \leftarrow x_1 + 1$
    else
      $i = i - 1$
      $x_i \leftarrow -\sum_{j=i+1}^{n} \mu_{j,i} x_j - \sqrt{\frac{R^2 - \sum_{j=i+1}^{n} I_j}{B_i}}$
      # left end of interval I_i
    end
  else
    $i = i + 1$
    $x_i \leftarrow x_i + 1$
  end
end
return S

ΚΕΦΑΛΑΙΟ 4

Υλοποίηση

Στο παρόν κεφάλαιο βρίσκεται ο πυρήνας της εργασίας μας. Αρχικά γίνεται η παρουσίαση των εργαλείων, στα οποία βασίστηκε η υλοποίησή μας, ενώ στη συνέχεια παρατίθενται οι μέθοδοι κατανομής και παραλληλοποίησης, του αλγορίθμου απαρίθμησης Schnorr και Euchner. Τέλος, εκτίθεται το εργαλείο που αναπτύχθηκε στα πλαίσια της εργασίας, με σκοπό την προσομοίωση της κατανεμημένης λειτουργίας υλοποίησης.

4.1 Τεχνολογίες και μέθοδοι υλοποίησης

4.1.1 Γενικά χαρακτηριστικά

Η ανάπτυξη της εργασίας καθορίστηκε από δύο παράγοντες. Ο πρώτος ήταν η ταχύτητα εκτέλεσης, ενώ ο δεύτερος, η ευκολία ανάπτυξης και διανομής της υλοποίησης. Οι δύο αυτοί παράγοντες υπαγόρευσαν τη χρήση τριών διαφορετικών γλωσσών προγραμματισμού, αλλά και την εκμετάλλευση δύο framework, του OpenMP για την παραλληλοποίηση των διεργασιών και του ZeroMQ για την κατανομή τους σε κόμβους του δικτύου.

Επιπλέον, χρησιμοποιήθηκαν τα εργαλεία AsyncIO και FPLLL, που αποτελούν προγραμματιστικές βιβλιοθήκες και αφορούν αντιστοίχως, την εφαρμογή ασύγχρονων μεθόδων προγραμματισμού και την επίλυση δύσκολων μαθηματικών προβλημάτων σε πλέγματα. Η ενορχήστρωση αυτή, οδήγησε στη δημιουργία ενός λογισμικού, που υπολογίζει προσεγγιστικά τη λύση του Shortest Vector Problem, ενώ παράλληλα μπορεί να διαμοιρασθεί εύκολα μέσω της πλατφόρμας λογισμικού ανοιχτού κώδικα Docker.

4.1.2 Κριτήρια επιλογής γλωσσών προγραμματισμού

Οι γλώσσες προγραμματισμού που χρησιμοποιήθηκαν, απαρτίζουν ένα δυνατό προγραμματιστικό πακέτο, το οποίο μπορεί να αποφέρει υψηλές ταχύτητες εκτέλεσης, ευκολία ανάπτυξης λογισμικού, ευανάγνωστο και συνοπτικό κώδικα. Αυτός ο συνδυασμός, έχει ως πυρήνα του, τη νέα, υβριδική γλώσσα προγραμματισμού Cython, στην οποία και υλοποιήσαμε τους βασικούς αλγορίθμους της λύσης μας.

Η Cython [6][22] μπορεί και συνδυάζει την ευκολία στην ανάπτυξη προγραμμάτων της Python, μειώνοντας δραματικά τα υπολογιστικά έξοδα κατά το χρόνο εκτέλεσης, προσφέροντας απόδοση εκτέλεσης αντίστοιχης αυτής των C-Compiled προγραμμάτων. Παράλληλα, διάφορες επεκτάσεις του λογισμικού, σημεία που απαιτούσαν προγραμματιστικές τεχνικές χαμηλότερου επιπέδου αλλά και δαπανηροί υπολογιστικοί αλγόριθμοι, υλοποιήθηκαν σε C++ και στη συνέχεια διοχετεύθηκαν στον κώδικα της Cython.

Ενώ ο πυρήνας είναι κράμα Cython και C++, το κέλυφος έχει συνταχθεί σε Python. Η Python προσφέρει την ευκολία της ταχύρρυθμης ανάπτυξης κώδικα, καθώς λόγω της ευρείας υιοθέτησής της, συνοδεύεται από μία πληθώρα πακέτων και εργαλείων (πχ ZeroMQ, AsyncIO, FPYLLL). Ταυτόχρονα είναι απόλυτα επεκτάσιμη και φορητή, με κύριο γνώρισμά της τον συνοπτικό και ευανάγνωστο κώδικα που παράγει . Επίσης, η PythonHighPython βρίσκεται προεγκατεστημένη στα περισσότερα εμπορικά και μη λειτουργικά συστήματα, κάτι που διευκολύνει τον διαμοιρασμό των αναπτυσσόμενων προγραμμάτων. Οι παραπάνω αναφορές επαληθεύουν τις προσδοκίες για το προγραμματιστικό πακέτο των αναγκών μας, καλύπτοντας όλες τις απαιτήσεις που φέρει το παραγόμενο λογισμικό.

4.1.3 Το framework OpenMP

Σύμφωνα με το νόμο του Moore "ο αριθμός των τρανζίστορ σ' ένα ολοκληρωμένο κύκλωμα διπλασιάζεται κάθε δύο χρόνια" κάτι που οδηγεί σε μεγαλύτερη συχνότητα, συνεπώς υψηλότερο κόστος ενέργειας. Τη λύση στο πρόβλημα αυτό δίνει η παραλληλοποίηση, όπου αντί για έναν επεξεργαστή υψηλής συχνότητας μπορούμε να έχουμε πολλούς χαμηλότερης. Ο τρόπος αυτός δεν προσφέρει υψηλότερη ταχύτητα υπολογισμών, αλλά κερδίζουμε τη δυνατότητα εκτέλεσης πολλών υπολογισμών ταυτόχρονα. Γι' αυτό και σήμερα κυκλοφορούν αποκλειστικά και μόνο πολυπύρηνοι επεξεργαστές.

Τα σύγχρονα λειτουργικά συστήματα κατανέμουν την πληθώρα των διεργασιών τους σε διαφορετικούς πυρήνες, ενώ η ίδια διαδικασία μπορεί να υλοποιηθεί και στις εφαρμογές. Οι κατηγορίες του παράλληλου προγραμματισμού είναι τρεις. Εμείς στην παρούσα εργασία θα αξιοποιήσουμε

τις δύο εκ αυτών, την Message-Passing Programming και την Shared-Memory Programming. Η τελευταία θεωρείται ως το απλούστερο μοντέλο παράλληλου προγραμματισμού και εφαρμόζεται στο framework OpenMP, το οποίο και αξιοποιούμε.

Το OpenMP (Open Multiprocessing) [18] είναι ένα Application Programming Interface (API) που υποστηρίζεται από τη C/C++, τη Fortran, πλέον και τη Cython, διαμέσω της C++ και διατίθεται για τα περισσότερα σύγχρονα λειτουργικά συστήματα όπως Linux, MacOS, Windows. Στην ανάπτυξη του λογισμικού της παρούσας εργασίας χρησιμοποιήθηκε το OpenMP σε δύο φάσεις, συμβάλλοντας στην παράλληλη εκτέλεση των ιδιαίτερα "βαριών" σημείων, που έφεραν υψηλό υπολογιστικό κόστος. Η πρώτη φάση αφορά την κλασική χρήση του, αποτελούμενη από εντολές, συναρτήσεις και μεταβλητές σε μέρος του προγράμματος που έχει συνταχθεί σε C++ ενώ η δεύτερη ποικίλλει και αφορά την παραλληλοποίηση τμημάτων της Cython.

Η Cython παρουσιάζει γενικότερα ιδιαιτερότητες ως γλώσσα προγραμματισμού, καθώς προσπαθεί να συνδυάσει με τον καλύτερο δυνατό τρόπο τις δυνατότητες δύο πολύ ισχυρών γλωσσών. Η γλώσσα αυτή δεν είναι thread-safe - τα νήματα του επεξεργαστή δουλεύουν αποκλειστικά στο δικό τους κομμάτι μνήμης- καθώς το Global Interpreter Lock (GIL) -λειτουργία που βοηθάει στη διαχείριση μνήμης της Python και της Cython- είναι ενεργοποιημένη ώστε να εμποδίζει πολλαπλά νήματα των πυρήνων του επεξεργαστή (threads) να εκτελέσουν μεταγλωττισμένα τμήματα κώδικα της Python (python's bytecodes). Εξαιτίας αυτού λοιπόν και αφού απενεργοποιηθεί το GIL, υλοποιεί την παραλληλοποίηση μέσω μίας μεθόδου της, που ονομάζεται prange, η οποία στο παρασκήνιο εκμεταλλεύεται το OpenMP για το διαμοιρασμό μνήμης και την πολυεπεξεργασία.

4.1.4 Το framework ZeroMQ

Ένας από τους σημαντικότερους αν όχι ο σημαντικότερος λόγος χρήσης το ψηφιακών συστημάτων σήμερα είναι η επικοινωνία, ενώ ο τρόπος ορθής διεξαγωγής της μεταξύ συστημάτων απασχολεί το κοινό τους, καθ΄ όλη τη διάρκεια της ύπαρξής τους. Το ZeroMQ [5][13] είναι μια προγραμματιστική βιβλιοθήκη που διευκολύνει την ασύγχρονη ανταλλαγή μηνυμάτων, μεταξύ κατανεμημένων ή συγχρονισμένων εφαρμογών, χωρίς την ανάγκη ύπαρξης προγράμματος διαμεσολάβησης, που να μεταφράζει τα μηνύματα από το επίσημο πρωτόκολλο ανταλλαγής του αποστολέα, στο αντίστοιχο του παραλήπτη.

Το framework αυτό αποτελεί μία υψηλού επιπέδου προσέγγιση, εμπεριέχοντας εύκολη και γρήγορη ανταλλαγή μηνυμάτων βασισμένη σε sockets. Χρησιμοποιώντας ένα επίπεδο αφαίρεσης, επιτρέπει στον χρήστη να

εφαρμόσει υψηλού επιπέδου τεχνικές, με ταχύτητες που εμφανίζονται μόνο σε χαμηλού επιπέδου υλοποιήσεις. Επιπλέον, εφαρμόζει ασύγχρονη υλοποίηση των sockets βάση της οποίας η φυσική σύνδεση, η εγκατάσταση, η επανασύνδεση, η αποτελεσματική μεταφορά των δεδομένων μεταξύ των κόμβων του δικτύου, βρίσκονται υπό ευθύνη του ZeroMQ και δεν απασχολούν τον χρήστη. Ακόμη υλοποιεί τους τρεις βασικούς τύπους επικοινωνίας request-reply, publish-subscribe και pipeline με μεγάλη ευκολία ως προς την εφαρμογή, τροποποίηση και ανάπτυξη τους. Παρέχει έτσι δυνατότητες δημιουργίας σύνθετων μοντέλων επικοινωνίας χωρίς ιδιαίτερο κόπο. Τα προαναφερθέντα είναι αναγκαία και ικανά για τη σύσταση του δικού μας μοντέλου επικοινωνίας, μεταξύ των κόμβων του δικτύου, το οποίο συστήνουμε στην πορεία της εργασίας.

4.1.5 Το εργαλείο AsyncIO

Με την έλευση των πολυπύρηνων επεξεργαστών, οι έννοιες του παράλληλου προγραμματισμού και του προγραμματισμού ταυτοχρονισμού που προϋπήρχε, άρχισαν να συγχέονται από πολλούς. Ο προγραμματισμός με ταυτοχρονισμό [7] σε αντίθεση με τον παράλληλο προγραμματισμό μπορεί και απευθύνεται τόσο σε μονοπύρηνα όσο και πολυπύρηνα συστήματα, καθώς η ιδέα που τον διέπει είναι η "αξιοποίηση του συστήματος, δια της εκτέλεσης μιας διεργασίας όταν αυτό είναι αδρανές, περιμένοντας την απόκριση μιας άλλης προηγηθείσας διεργασίας".

Η προγραμματιστική βιβλιοθήκη της Python, AsyncIO [1][10], υλοποιεί αυτή την ιδέα, δίνοντας μία εναλλακτική στη χρήση παράλληλου προγραμματισμού καθώς μπορεί και επιτελεί διεργασίες ταυτόχρονα, αποφεύγοντας τις δυσκολίες που περιέχει η παραλληλοποίηση. Ακόμη, δίνει τη λύση σε συστήματα εξυπηρέτησης (servers), προσφέροντας τη δυνατότητα διαχείρισης πολλών χιλιάδων ταυτόχρονων αιτημάτων από ένα και μόνο σύστημα. Η βιβλιοθήκη αυτή, αποτελεί πλέον μέλος των νέων εκδόσεων της Python, και η εφαρμογή της δε θα μπορούσε να λείπει από το framework ZeroMQ, όπως και απ' την ανάπτυξη της παρούσας εργασίας.

4.1.6 Το εργαλείο FPLLL

Η κοινότητα προγραμματιστών ανοιχτού κώδικα έχει προσφέρει πληθώρα εργαλείων, διευκολύνοντας έτσι το έργο, τόσο της επιστήμης όσο και της βιομηχανίας. Το FPLLL ανήκει σ' αυτή τη συλλογή εργαλείων ανοιχτού κώδικα, παρέχοντας υλοποιήσεις εφαρμοσμένων αλγορίθμων στην κρυπτογραφία με πλέγματα (lattice-based cryptography). Αλγόριθμοι παραγωγής πλεγμάτων, διάφορες εφαρμοσμένες πράξεις σ' αυτά, όπως η

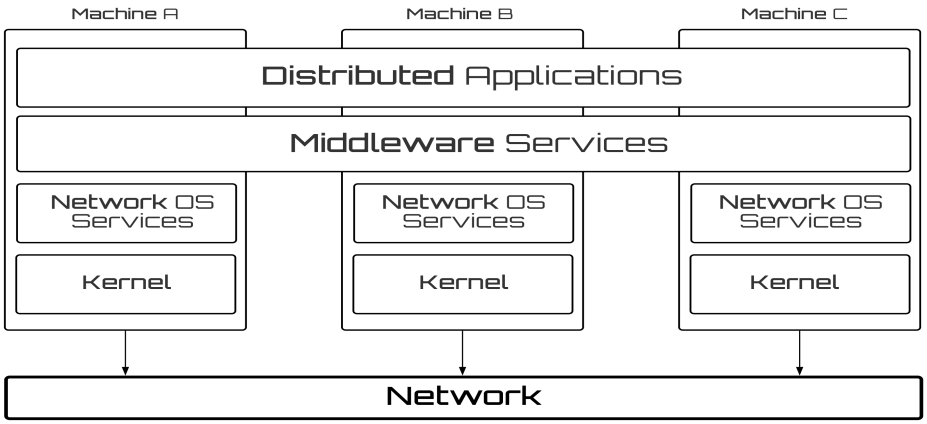
αναγωγή LLL, HKZ, BKZ, καθώς και η επίλυση του SVP και του CVP με περιορισμένες δυνατότητες, μπορούν να αξιοποιηθούν διαμέσου του εργαλείου. Το FPLL διατίθεται μέσω της πλατφόρμας GitHub ενώ παράλληλα έχει αναπτυχθεί ένας Python wrapper αυτού (FPYLLLL) τον οποίο και αξιοποιούμε.

4.2 Κατανεμημένη προσέγγιση υλοποίησης

4.2.1 Γενικά Χαρακτηριστικά

Ένα σύνολο από αυτόνομα υπολογιστικά συστήματα (κόμβοι) που συνδέονται μεταξύ τους και παρουσιάζονται στους χρήστες τους ως κάτι το ενιαίο και συνεκτικό, αποτελούν ένα κατανεμημένο σύστημα [23]. Σ' αυτήν την ιδέα θεμελιώνεται η υλοποίηση της εργασίας, καθώς διασφαλίζεται η διαμοίραση πόρων και επικοινωνίας, πετυχαίνοντας την καλύτερη σχέση μεταξύ απόδοσης και κόστους. Επιπλέον θεωρείται μία αξιόπιστη και ασφαλής λύση, με κύριο χαρακτηριστικό τις δυνατότητες επεκτασιμότητας του συστήματος.

Γι' αυτό το σκοπό, αναπτύχθηκε ειδικό λογισμικό για κατανεμημένα συστήματα (Middleware) με τη βοήθεια του framework ZeroMQ και του εργαλείου AsyncIO. Το λογισμικό αυτό, παρέχει υψηλού βαθμού διαφάνεια (transpareny), συγχρονισμό μεταξύ των υπολογιστικών συστημάτων (concurrency), ανεξαρτησία από το λειτουργικό σύστημα που διαθέτει ο κάθε κόμβος (openness), διαχείριση των πόρων του εκάστοτε συστήματος (resource sharing), ενώ είναι κλιμακούμενο (scalability) και χαρακτηρίζεται από ανεκτικότητα σε σφάλματα (fault tolerance).



Σχήμα 2: Γενική δομή ενός κατανεμημένου συστήματος ως middleware.

Τα χαρακτηριστικά που προσδοκήσαμε να αποδώσουμε στο middleware

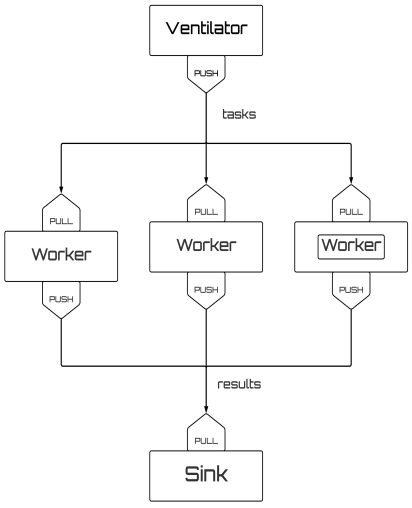
που παράχθηκε, βασίστηκαν στο μοντέλο κατανεμημένων συστημάτων Processor Pool Model (PPM) και την αρχιτεκτονική δικτύου, διαίρει και βασίλευε (Divide and Conquer - Parallel Pipeline). Το πρότυπο PPM απέφερε κατανεμημένη επεξεργασία υψηλής απόδοσης, η οποία θεωρείται απόγονος της κατηγορίας του παράλληλου προγραμματισμού Message-Passing Programming, που προαναφέρθηκε.

Το σύνολο των κόμβων διατηρεί σταθερή τη λειτουργία του, ανεξάρτητα με την τοποθεσία και τον τρόπο που διεξάγεται η αλληλεπίδραση μεταξύ του χρήστη και του συστήματος, δηλώνοντας τη συνοχή του. Η αρχιτεκτονική του δικτύου πετυχαίνει μέσω της παραλληλίας που τη διέπει, υψηλές επιδόσεις σε αλγορίθμους, ενώ παρέχεται ως βασικό πρότυπο σχεδίασης δικτύου, μέσω του ZeroMQ. Η ανάλυση των χαρακτηριστικών θα ακολουθήσει παρακάτω.

4.3 Αρχιτεκτονική δικτύου

Οι ανάγκες παραλληλίας, ταχύτητας και κλιμάκωσης είναι αυτές που καθόρισαν το μοντέλο που επιλέχθηκε ως βάση για την ανάπτυξη του δικτύου. Όπως και προαναφέρθηκε το μοντέλο που στηριχθήκαμε είναι το Divide and Conquer, το οποίο ονομάζει το ZeroMQ ως Parallel Pipeline. Το πρότυπο αυτό της παραλληλίας δίνει τη δυνατότητα συμμετοχής ακαθόριστου πλήθους συστημάτων-εργατών (worker), οι οποίοι δέχονται εντολές από ένα κεντρικό υπολογιστικό σύστημα (ventilator) και αποστέλλουν τα αποτελέσματα στο τελικό σύστημα συγκέντρωσης αποτελεσμάτων (sink).

Το μοντέλο που αναπτύχθηκε στα πλαίσια της εργασίας, αποτελεί μία παραλλαγή του παραπάνω έχοντας μίας βασική προσθήκη και αρκετές εσωτερικές τροποποιήσεις, έτσι ώστε να επιτευχθούν υψηλότερες επιδόσεις. Όπως παρατηρείται και στην απεικόνιση του κατανεμημένου συστήματος παρακάτω, η προσθήκη βρίσκεται στο κεντρικό σύστημα (server) με την προσάρτηση του κόμβου υποβοήθησης (secretary). Ο κόμβος-secretary αφού αλληλοεπιδράσει με τον κόμβο-server, εφαρμόζοντας το πρότυπο αίτημα-απάντηση (request-reply), μεταδίδει με χρήση του προτύπου μετάδοσης-εγγραφής



Σχήμα 3: Καθορισμός Parallel Pipeline.

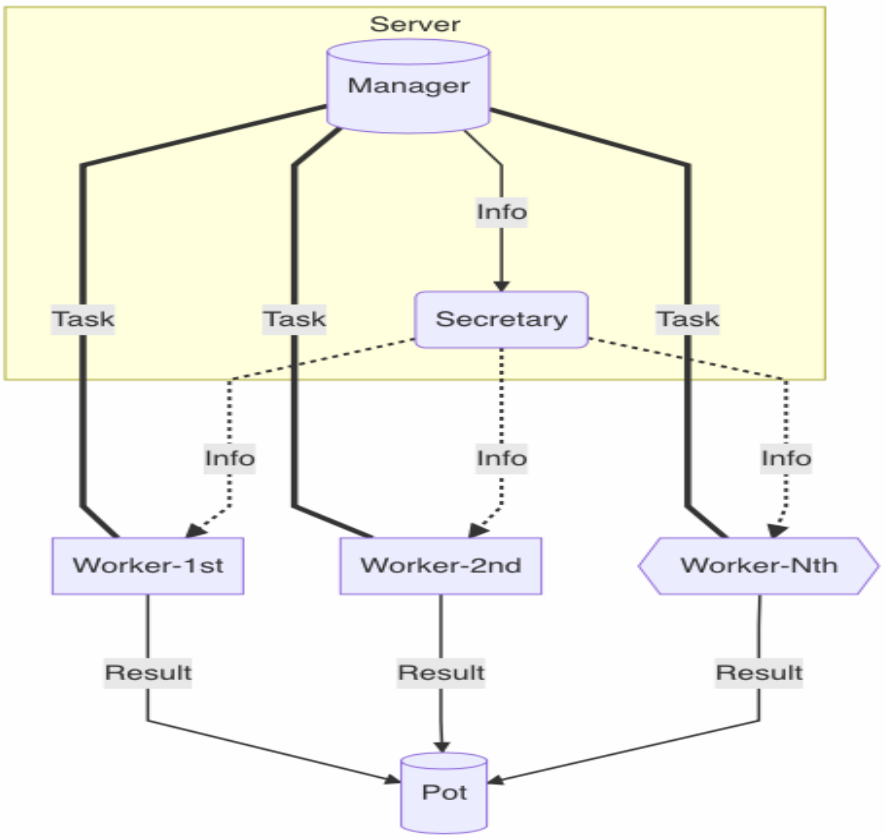
(publish-subscribe) δεδομένα στους κόμβους-worker. Κάθε νέος κόμβος-worker που εισάγεται στο κατανεμημένο σύστημα, ώστε να ενισχύσει με την επεξεργαστική του ισχύ το σύστημα, λαμβάνει πρώτα μηνύματα εγγραφής (subscribe) από τον κόμβο-secretary.

Οι κόμβοι-worker αφού λάβουν πληροφορίες (info) από τον βοηθητικό κόμβο (secretary), επικοινωνούν με το κόμβο-server εφαρμόζοντας το μοντέλο parallel pipeline. Το σύστημα συγκέντρωσης αποτελεσμάτων είναι ο κόμβος-δοχείο (pot). Οι κόμβοι-worker εκτελούν κάθε διεργασία (task) που λαμβάνουν από τον κόμβο-server, και αποστέλλουν τα αποτελέσματα (result) στο σύστημα συγκέντρωσης αποτελεσμάτων (pot), η διαδικασία αυτή επαναλαμβάνεται έως ότου ο κόμβος κόμβος-server διακόψει τη λειτουργία του. Ο τελικός κόμβος-pot αφού έχει λάβει στην αρχή τις απαραίτητες πληροφορίες, δέχεται επαναλαμβανόμενα τα αποτελέσματα των κόμβων-worker και στο τέλος ανακοινώνει τη λύση του προβλήματος. Το συνολικό σύστημα που προκύπτει θεωρείται συνεκτικό πληρώντας τις αρχές που διέπουν ένα κατανεμημένο σύστημα.

4.3.1 Δυνατότητες κόμβων

Το κατανεμημένο σύστημα που παρουσιάζεται, επιτρέπει την απόκρυψη ενδεχόμενων λαθών από τον τελικό χρήστη. Αυτό πρακτικά σημαίνει ότι, τα προγράμματα εφαρμογών μπορούν να συνεχίζουν τις ενέργειές τους παρά την πιθανή αποτυχία κάποιου τμήματος του hardware ή του software, εφαρμόζοντας την αρχή της διαφάνειας αποτυχίας. Το ZeroMQ είναι αυτό που αναλαμβάνει τη διαχείριση μίας τέτοιας κατάστασης αποτυχίας, απομονώνοντας το πρόβλημα, ώστε να μην επηρεάσει το συνολικό σύστημα και επιχειρώντας να επανακάμψει από την κατάσταση σφάλματος, εάν αυτό καθίσταται δυνατόν.

Συνεχίζοντας, σύμφωνα με τη διαφάνεια διανομής, ένας χρήστης του κατανεμημένου συστήματος δεν είναι σε θέση να γνωρίζει που πραγματοποιήθηκε ένας υπολογισμός. Επίσης, μία εφαρμογή θα πρέπει να είναι αδιάφορη για το που ακριβώς βρίσκονται αποθηκευμένα τα δεδομένα, ενώ δεν της είναι γνωστό εάν αυτά έχουν αντιγραφεί ή όχι. Η διαφάνεια διανομής εφαρμόζεται και στο εν λόγω σύστημα, καθώς το αρχικό πρόβλημα διασπάται σε υποπροβλήματα μέσω του server. Αυτά διανέμονται στους κόμβους-workers οι οποίοι με τη σειρά τους, δίνουν τα αποτελέσματα τους στο κόμβο-pot. Τέλος, κάποιος κόμβος-worker δεν δύναται να γνωρίζει εάν είναι αυτός που έχει βρει τη λύση του προβλήματος, καθώς αγνοεί ακόμα και το ποιοι άλλοι κόμβοι συμμετέχουν στο σύστημα, πόσο μάλλον το ποια προβλήματα επιλύουν εκείνοι.



Σχήμα 4: Διάγραμμα απεικόνισης της γενικής δομής του κατανεμημένου δικτύου.

Η σημασία του κόμβου-secretary

Η προσθήκη ενός επιπλέον κόμβου-secretary προσδίδει περισσότερες δυνατότητες στο κατανεμημένο σύστημα. Αρχικά επιτρέπει την είσοδο νέων κόμβων στο σύστημα σε ακαθόριστο χρόνο, υλοποιώντας το πρότυπο επικοινωνίας publish-subscribe. Κατά αυτόν τον τρόπο αποφορτίζεται ο κόμβος-server από τα αιτήματα νέων worker, ο οποίος μπορεί τώρα αδιάκοπα να επιτελεί τη λειτουργία του.

Επιπλέον, ο όγκος του κάθε μηνύματος, που αποστέλλει ο κόμβος-server στους κόμβους-worker καθορίζεται πλέον μόνο από τη φύση του προβλήματος, χωρίς να επαναλαμβάνονται γενικές πληροφορίες, παρά μόνο εντολές για τη διεργασία που καλείται να εκτελεσθεί. Οι γενικές πληροφορίες που καθορίζουν το πρόβλημα και είναι αναγκαίες στους νέους κόμβους-worker, μεταδίδονται μόνο μέσω του secretary. Έπειτα, ο κάθε κόμβος-worker τις αποθηκεύει τοπικά, και τις χρησιμοποιεί σε κάθε νέα διεργασία που λαμβάνει από τον κόμβο-server. Το γεγονός αυτό οδηγεί σε δραματικά μικρότερο μέγεθος μηνυμάτων, άρα μικρότερη διάρκεια κωδικοποίησης/αποκωδικοποίησης τους, λιγότερα σφάλματα μετάδοσης και συντομότερη αποστολή τους. Συνεπώς ταχύτερη επικοινωνία μεταξύ server και worker.

Σε συνδυασμό λοιπόν, με τους αλγορίθμους που αναπτύχθηκαν για τη διάσπαση του προβλήματος σε επιμέρους διεργασίες, η προσθήκη του κόμβου-secretary καθιστά το σύστημα επεκτάσιμο (sclabale). Έτσι, προσφέρεται η δυνατότητα στις εφαρμογές, να επεκτείνουν την κλίμακα τους, χωρίς να υπάρχει αλλαγή στη δομή και στους αλγορίθμους των εφαρμογών.

4.3.2 Η λειτουργία του προσομοιωτή (Simulator)

Στα πλαίσια της εργασίας, αναπτύχθηκε ένα λογισμικό προσομοίωσης της λειτουργίας (simulator) του παραπάνω κατανεμημένου δικτύου, για την επίλυση του προβλήματος εγγύτερου διανύσματος σε πλέγματα. Ο σκοπός που δημιουργήθηκε ήταν η απαίτηση μας να ελέγξουμε τη συμπεριφορά του κατανεμημένου συστήματος σε διάφορες καταστάσεις με διαφορετικές παραμέτρους, αλλά και η δυνατότητα που προέκυψε να εκτελέσουμε επιτυχώς το παραγόμενο λογισμικό σε κατανεμημένο σύστημα με τη χρήση ενός και μόνου πολυπύρηνου συστήματος. Οι πυρήνες του επεξεργαστή του μηχανήματος, προσομοίωσαν τη λειτουργία των διαφόρων τμημάτων του κατανεμημένου συστήματος. Η χρήση του AsyncIO επέτρεψε την προβολή της λειτουργίας όλου του συστήματος από ένα και μόνο τερματικό, εφαρμόζοντας μεθόδους ασύγχρονου προγραμματισμού.

Ο προσομοιωτής υποστηρίζει Command Line Interface (CLI) σε unix-based συστήματα, όπως MacOS και Linux. Έχει τη δυνατότητα να προσομοιώσει τη

λειτουργία κόμβων-worker ανάλογα με τους διαθέσιμους επεξεργαστικούς πυρήνες του συστήματος. Συνεπώς απαιτεί τουλάχιστον τετραπύρηνα συστήματα ώστε να προσομοιώσει τη λειτουργία των τεσσάρων κόμβων, 1 x server, 1 x secretary, 2 x worker και 1 x pot, που θεωρούνται τα ελάχιστα αναγκαία για την εκτέλεση ενός ολοκληρωμένου πειράματος.

Για τις ανάγκες των πειραμάτων, σε διάφορα συστήματα αλλά και για τον εύκολο διαμοιρασμό του λογισμικού, καθώς θεωρείται ιδιαίτερα απαιτητική η παραμετροποίηση των αναγκαίων προϋποθέσεων για την εκτέλεση του, χρησιμοποιήθηκε η πλατφόρμα ανοιχτού λογισμικού Docker[19][14]. Το container docker που παράχθηκε βασίζεται στο λειτουργικό σύστημα Ubuntu, ενώ με εντολές συστήματος οργανωμένες σε αρχεία (Makefiles) γίνεται η μεταγλώττιση (compile) όλων των αρχείων πηγαίου κώδικα (source code) της εργασίας, είτε είναι σε C++, είτε σε Cython, είτε μέσω του Python-Packaging[12] σε Python.

Σε επίπεδο λειτουργίας, ο προσομοιωτής εκμεταλλεύεται το τοπικό εικονικό δίκτυο (localhost) που παρέχει κάθε σύστημα μέσω του λειτουργικού του συστήματος, ώστε να αποφευχθεί η φυσική διασύνδεση στο δίκτυο. Δεσμεύει ένα πλήθος από πόρτες (ports) του localhost, ανάλογο με αυτό των κόμβων που θα λειτουργήσει. Για τον λόγο αυτό προσφέρεται μέσω του simulator, λειτουργία που κλείνει όλες τις πόρτες που θα χρειαστεί η εφαρμογή για να εκτελεσθεί. Ακόμη μπορεί να καθορισθεί ο ακριβής αριθμός των κόμβων-worker που θα προσομοιωθεί, αρκεί να υποστηρίζεται από την επεξεργαστική μονάδα του συστήματος, όπως επίσης και τα νήματα (threads) που είναι διαθέσιμα προς εκμετάλλευση από τα τμήματα του αλγορίθμου, που εμπεριέχουν παράλληλη επεξεργασία. Επίσης, δίνεται η δυνατότητα στον χρήστη να παραμετροποιήσει με κάθε τρόπο το πρόβλημα που καλείται να επιλύσει το σύστημα, είτε ακόμη να εισάγει ένα πρόβλημα επιλογής του. Τέλος, υποστηρίζει δύο λειτουργίες προσέγγισης της λύσης του προβλήματος, που θα παρουσιαστούν αναλυτικά στη συνέχεια.

4.4 Παραλληλοποίηση του Schnorr-Euchner Algorithm

4.4.1 Φιλοσοφία παραλληλοποίησης

Ο αλγόριθμος απαρίθμησης δημιουργεί στο παρασκήνιο ένα δένδρο (enumeration tree) το οποίο και προσπελαύνει με τη χρήση του αλγορίθμου διάσχισης γράφων Depth-First Search (DFS), με στόχο την εύρεση του διανύσματος που ικανοποιεί τις απαιτήσεις του προβλήματος. Ο αλγόριθμος DFS όπως προμηνύει και το όνομα του, διασχίζει το δένδρο σε βάθος, δηλαδή κατά μήκος του κάθε κλαδιού, έως ότου καταλήξει σ΄ ένα κόμβο-φύλλο της δενδρικής δομής, όπου ενδεχομένως αποτελεί υποψήφια λύση του

προβλήματος. Η ιδέα που στηρίχθηκε η παραλληλοποίηση του αλγορίθμου Schnorr-Euchner έγκειται στην παραλληλοποίηση της αναζήτησης στο δένδρο απαρίθμησης, δηλαδή του DFS.

Η παραλληλοποίηση στηρίζεται στην απλή ιδέα, της διάσπασης του δένδρου απαρίθμησης σε υποδένδρα. Η αρχική δενδρική δομή διασπάται σε μικρότερες, οι οποίες με τη σειρά τους διαμοιράζονται σε κατανεμημένα ανεξάρτητα συστήματα. Σε κάθε σύστημα επαναλαμβάνεται η διαδικασία διάσπασης. Ανάλογα με τους διαθέσιμους πόρους του συστήματος διασπάται το υποδένδρο σε μικρότερα, ώστε να εκτελεσθεί ταυτόχρονα σε κάθε δένδρο η αναζήτηση του μικρότερου διανύσματος. Ο τρόπος με τον οποίο εφαρμόζεται η διάσπαση σε υποσύνολα-υποδένδρα εκμεταλλεύεται την πρόταση του αλγορίθμου KFP, για τον ορισμό των διαστημάτων I_k.

$$I_k = \left[-\sum_{i=n+2-k}^{n} \mu_{i,n+1-k} u_i - \sqrt{\frac{R^2 - \sum_{j=n+2-k}^{n} l_j}{\|\boldsymbol{b}^*_{n+1-k}\|}}, -\sum_{i=n+2-k}^{n} \mu_{i,n+1-k} u_i + \sqrt{\frac{R^2 - \sum_{j=n+2-k}^{n} l_j}{\|\boldsymbol{b}^*_{n+1-k}\|}} \right]$$

Έπειτα, εφαρμόζεται ο αλγόριθμος Schnorr-Euchner, σύμφωνα με τον οποίο η αναζήτηση στα διαστήματα I_k ξεκινάει από τη μέση του διαστήματος.

4.4.2 Δυνατότητες υλοποίησης

Η μέθοδος με την οποία γίνεται η διάσπαση του δένδρου μπορεί να διαφέρει, γι' αυτό και στην παρούσα εργασία υλοποιήθηκαν δύο προσεγγίσεις. Η πρώτη στηρίζεται στο πλήθος των διεργασιών που θεωρείται επιθυμητό να διασπασθεί το πρόβλημα, ενώ η δεύτερη στην ιδέα της διάσπασης του προβλήματος, αφού πρώτα υπολογισθούν τα διαστήματα για κάποιο συγκεκριμένος βάθος του δένδρου απαρίθμησης. Οι δύο αυτές παραλλαγές ισοδυναμούν με τις δύο διαφορετικές λειτουργίες του Simulator που αναφέρθηκαν στην προηγούμενη ενότητα.

Η διάσπαση του δένδρου βάσει του επιθυμητού πλήθους διεργασιών, έχει καλή εφαρμογή όταν έχουμε γνώση του ακριβή αριθμού των κόμβων-worker, καθώς τότε ο διαμοιρασμός μπορεί να είναι ακριβής. Έτσι αποφεύγεται η εμφάνιση του σύνηθες προβλήματος, κατά το οποίο απομένει να εκτελείται μία περίσσεια διεργασία ενώ τα υπόλοιπα συστήματα έχουν ολοκληρώσει την επεξεργασία τους. Κάτι τέτοιο θα μπορούσε να οδηγήσει ακόμα και σε διπλασιασμό της εκτέλεσης του συστήματος. Φυσικά υπάρχουν ποικίλες λύσεις στο πρόβλημα αυτό, όπως το dynamic-scheduling ή ο αλγόριθμος Round Robin, αλλά στη δική μας περίπτωση αυξάνουν αισθητά την πολυπλοκότητα. Συνεπώς, καταλήξαμε πως εάν γνωρίζουμε τον αριθμό των κόμβων-worker, επιλέγουμε το πλήθος των διεργασιών έτσι ώστε να είναι ακέραιο

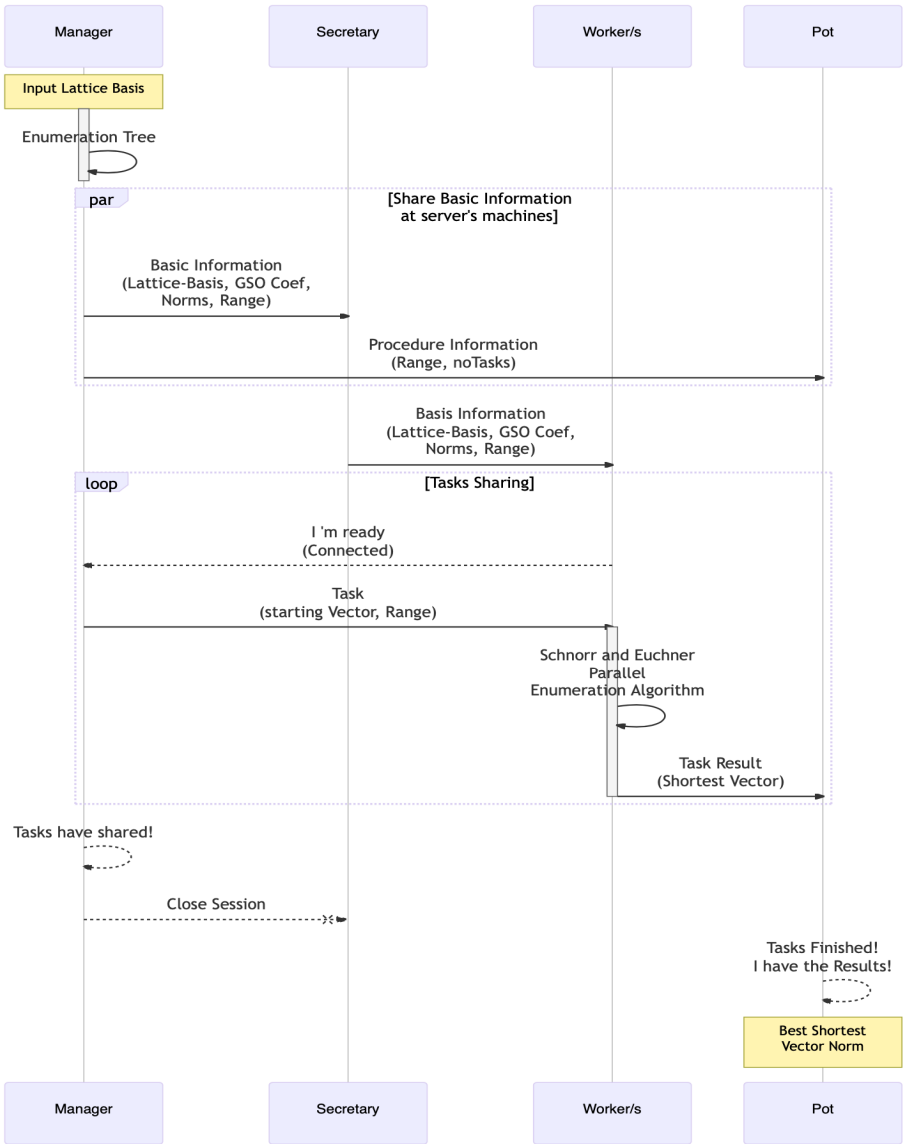
πολλαπλάσιο του πλήθους των κόμβων-worker, αυξάνοντας ενδεχομένως σε κάποιες περιπτώσεις τις απαιτήσεις μερικών διεργασιών. Πιο αναλυτικά, ενώ υπό ιδανικές συνθήκες τα υποσύνολα (υποδένδρα) που διαμοιράζονται έχουν τον ίδιο αριθμό στοιχείων, σε περιπτώσεις όπως την προαναφερθείσα διαφέρουν, ώστε να μην υπάρχει πλεόνασμα, γι' αυτό και οι απαιτήσεις των διαμοιραζόμενων διεργασιών δεν είναι ισοδύναμες.

Η έναρξη της παράλληλης επεξεργασίας αφού προϋπολογισθούν κάποια διαστήματα $U_n, U_{n-1}, \ldots, U_k$ όπου $k \in \mathbf{Z}$ είναι η δεύτερη λειτουργία που υλοποιήθηκε. Σύμφωνα μ' αυτή τη λειτουργία μπορούν να κλαδευτούν κάποια υποδένδρα τα οποία δεν θα οδηγούσαν σε κάποια λύση, γι' αυτό και συνίσταται η επιλογή του κατάλληλου k μετά από ανάλογο πειραματισμό, (σ' αυτή τη προσέγγιση θα μπορούσαν να εφαρμοσθούν τεχνικές Machine Learning για την επιλογή του k). Ο ακέραιος αριθμός k συμβολίζει το βάθος που θα προσπελαθεί στο δένδρο απαρίθμησης έως ότου διασπασθεί σε υποδένδρα. Το πλήθος των διεργασιών που προκύπτει διαφέρει ανάλογα την περίπτωση, αλλά η φιλοσοφία διαχωρισμού τους είναι η ίδια μ' αυτή που εφαρμόζεται στην προηγούμενη λειτουργία.

4.4.3 Ανάλυση συστήματος

Το κατανεμημένο σύστημα παράλληλης επεξεργασίας για m το πλήθος κόμβους-workers και n αριθμό νημάτων-threads σε κάθε κόμβο-worker, μπορεί να εκτελέσει παράλληλα, $m * n$ διεργασίες. Η λειτουργία του συστήματος αυτού, παρουσιάζεται αναλυτικά στο διάγραμμα 5.

Έπειτα, στους αλγορίθμους που ακολουθούν παρουσιάζουμε τις μεθόδους που ακολουθήθηκαν για την επίτευξη της παραλληλίας στο συνολικό σύστημα. Η λειτουργία της διάσπασης λαμβάνει χώρα στον server node όπου δημιουργείται ένα δέντρο απαρίθμησης μέσω του αλγορίθμου *4.1*, το οποίο διασπάται σε επιμέρους κλαδιά (συνδεδεμένη λίστα), που συμβολίζουν τις διεργασίες. Στη συνέχεια διαμοιράζονται σε καθέναν από τους worker nodes. Οι worker nodes αφού λάβουν τις διεργασίες, σύμφωνα με τον αλγόριθμο *4.2*, τις διασπούν σε υποδιεργασίες οι οποίες εκτελούνται παράλληλα στον επεξεργαστή του συστήματος. Κύριο μέλημα είναι καμία υποδιεργασία να μην παρεμβαίνει στο χώρο αναζήτησης κάποιας άλλης, ώστε να μην υπάρχει σπατάλη επεξεργαστικού χρόνου, αυτό επιτυγχάνεται με τον αλγόριθμο *4.3*, παραλλαγή του αλγορίθμου των Schnor και Euchner[21].



Σχήμα 5: Λειτουργία του κατανεμημένου συστήματος.

Algorithm 4.1: Sever-Node: Enumeration Tree for Distributed Operation

Input: μ, B^*, n
Output: Tree branches.

```
B_i^* = ||b_i^*||^2, μ = GSO(B)     #Calculate Gram-Schmidt Orthogonalization
tree = new Tree()
x_vector = new Vector(n)     # Fill with zeros
# Calculate I_k
upper_bound = find_range_KFP(M, B^*, n, tree.height + 1, x_vector).upper_bound
tree.new_root_node(upper_bound, n, x_vector)
BFS = tree.iterator.begin()
total_range = 1
expression = distribution_mode.expression() # modes: [via_Nodes, via_Depth]
while expression(total_range) do
    parent = bfs()
    height = parent.depth + 1
    if n < height then
        break
    end
    index = n − height
    for int i = 0; i ≤ parent.upper_bound; i++ do
        new_x_vector = parent.x_vector
        nex_x_vector[index] = i
        new_upper_bound = find_range_KFP(height, new_x_vector).upper_bound
        tree.new_node(new_upper_bound, n, new_x_vector, index)
    end
    total_range+ = parent.upper_bound
    BFS.next()
end
return tree.get_branches()
```

Algorithm 4.2: Worker-Node: Schnorr-Euchner Parallel Operation

Input: *server_msg*, *no_threads*
Output: S

```
μ = server_msg.get_GSO_Coefficients()
B* = server_msg.get_B_norms2()
n = server_msg.get_size()
R = server_msg.get_R()
branch = server_msg.get_process()
x_vector = branch.x_vector
index_border = branch.index                #constant
if index_border == 0 then
    return x_vector
end
range = branch.upper_bound + 1             #[0, branch.upper_bound]
no_tasks = range//threads                  #integer division
balance = range%threads                    #modulo
if no_tasks == 0 then
    no_threads = balance
end
x_parallel = newArray(n, no_threads)
parallel for t = 0, t < no_threads, t++ do
    copy(x_parallel[t], x_vector, n)
end
S = new ParallelHeap()
parallel for t = 1, t < no_threads, t++ do
    extra_task = (t + balance − no_threads) * ((t + balance) >= no_threads)
    upper_bound = no_tasks * t − 1 + extra_task
    lower_bound = upper_bound + 1 − no_tasks − ((t + balance) >= no_threads)
    S = S ∪ {Bounded_SchnorrEuchner(μ, B*, n, R,
          x_parallel[t_id], index_border, lower_bound, upper_bound)}
end
return S
```

Algorithm 4.3: Bounded Schnorr-Euchner Enumeration Algorithm

Input: $\mu, B^*, n, R, x, i_{border}, lower\_bound, upper\_bound,$
Output: S

$\boldsymbol{c} = (c_i) \leftarrow \mathbf{0}_n, l = (l_i) \leftarrow \mathbf{0}_n$
$\Delta x \leftarrow (1, \mathbf{0}_{n-1}), \Delta^2 x \leftarrow (1, (-1)_{n-1}), suml_i \leftarrow 0, S = \emptyset, i \leftarrow 0, S \leftarrow [])$
while $i < i_{bound}$ **do**
  $c_i \leftarrow -\sum_{j=i+1}^{n-1} x_j \mu_{ji}$
  $l_i \leftarrow B_i(x_i - c_i)^2$
  $sumli \leftarrow \sum_{j=i}^{n-1} l_j$
  if $sumli \leq R^2$ *and* $i = 0$ **then**
    $S \leftarrow S \cup \{\sum_{j=0}^{n-1} x_j b_j\}$
  end
  if $sumli \leq R^2$ *and* $i > 0$ **then**
    $i = i - 1$
    $c_i \leftarrow -\sum_{j=i+1}^{n-1} \mu_{j,i} x_j$ # *center of interval* I_{n+1-i}
    $x_i \leftarrow \lceil c_i \rfloor$
    $\Delta x_i \leftarrow 0$
    if $c_i < x_i$ **then**
      $\Delta^2 x_i \leftarrow 1$
    else
      $\Delta^2 x_i \leftarrow -1$
    end
  end
  if $i == i_{border}$ *OR* $(i == i_{border} - 1$ *AND* $(x[i] < lower\_bound$ *OR* $x[i] > upper\_bound))$ **then**
    break
  else
    $i \leftarrow i + 1$
    $\Delta^2 x_i \leftarrow -\Delta^2 x_i$
    $\Delta x_i \leftarrow -\Delta x_i + \Delta^2 x_i$
    $x_i \leftarrow x_i + \Delta x_i$
    if $i == i_{border} - 1$ *AND* $(x[i] < lower\_bound$ *OR* $x[i] > upper\_bound)$ **then**
      break
    end
  end
end
return S

ΚΕΦΑΛΑΙΟ 5

Επίλογος

5.1 Σύνοψη και Συμπεράσματα

Η εργασία εκπονήθηκε μετά από συστηματική μελέτη του μαθηματικού υπόβαθρου της επιστήμης των πλεγμάτων, της κρυπτογραφίας δημοσίου κλειδιού και της ανάλυσης βέλτιστων αλγορίθμων παραλληλοποίησης και ταυτοχρονισμού. Χρησιμοποιήθηκε πληθώρα προγραμματιστικών εργαλείων, ενώ αναπτύχθηκαν και νέα εργαλεία για τις ανάγκες υλοποίησης και προσομοίωσης της λειτουργίας του συστήματος. Η συγγραφή του λογισμικού σε τρεις γλώσσες προγραμματισμού κάλυψε τις υψηλές απαιτήσεις που τέθηκαν εκ των προτέρων.

Στόχος ήταν η επίλυση του προβλήματος του εγγύτερου διανύσματος, μέσω ενός συστήματος υψηλής απόδοσης, με ανοχή στα λάθη, πλήρως επεκτάσιμο και εύκολα διαμοιραζόμενο. Τα αποτελέσματα που προέκυψαν φανερώνουν πως η παράλληλη υλοποίηση των αλγορίθμων μπορεί να φέρει μεγέθη απόδοσης, εκθετικά μεγαλύτερα σε σύγκριση με τα γραμμικά συστήματα. Το σύστημα ολοκληρώθηκε εκπληρώνοντας κάθε προσδοκία, όντας πλήρως λειτουργικό.

5.2 Επεκτάσεις

Η εργασία αυτή αποτελεί τον εναρκτήριο λίθο της ανάπτυξης ενός λογισμικού με επίκεντρο την τεκμηρίωση ασφαλείας ενός κρυπτογραφικού συστήματος βασιζόμενο σε πλέγματα. Τα επόμενα βήματα ανάπτυξης θα μπορούσαν να περιέχουν, την προσθήκη αλγορίθμων επίλυσης σε CUDA, που εκτελούνται σε κάρτα γραφικών, ώστε να αξιοποιείται πλήρως κάθε σύστημα. Επίσης την εφαρμογή τεχνικών βελτιστοποίησης σε αλγορίθμους που αναπτύχθηκαν. Επιπλέον τη δημιουργία ενός συστήματος παροχής υπηρεσιών ιστού μέσω του οποίου διευκολύνεται η συμμετοχή νέων κόμβων στο κατανεμημένο σύστημα

και τελος τη διεξαγωγή πειραμάτων ευρείας κλίμακας υπό πραγματικές συνθήκες.

Αναφορές

[1] AsyncIO documentation. https://docs.python.org/3/library/asyncio.html. Accessed: 2021-09-06.

[2] Wikipedia n-ball (mathematics). https://en.wikipedia.org/wiki/Ball_(mathematics). Accessed: 2021-09-03.

[3] Κ. Α. Δραζιώτης Κρυπτογραφία μετά τους Diffie-Hellman, 2019.

[4] Miklós Ajtai. The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 10–19, New York, NY, USA, 1998. Association for Computing Machinery.

[5] Faruk Akgul. *ZeroMQ*. Packt Publishing, 2013.

[6] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[7] M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Pearson Education, Inc., 2nd edition, 2018.

[8] The FPLLL development team. fplll, a lattice reduction library, Version: 5.4.1. Available at https://github.com/fplll/fplll, 2021.

[9] Guillaume Hanrot and Damien Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. *CoRR*, abs/0705.0965, 2007.

[10] Caleb Hattingh. *Using Asyncio in Python*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 1st edition, 2020.

[11] Jens Hermans, Michael Schneider, Johannes Buchmann, Frederik Vercauteren, and Bart Preneel. Parallel shortest lattice vector enumeration on graphics cards. In Daniel J. Bernstein and Tanja Lange, editors,

AFRICACRYPT, volume 6055 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2010.

[12] Dane Hillard. *Publishing Python Packages*. MAEP, 1st edition, 2021.

[13] Pieter Hintjens. *Zero MQ, Messaging for Many Applications*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 1st edition, 2013.

[14] Aidan Hobson Sayers Ian Miel. *Docker in Pracatise*. Manning Publications Co.,20 Baldwin Road, PO Box 761, Shetler Island, NY 11964, 2nd edition, 2019.

[15] Ian Ozsvald Micha Gorelick. *High Performance Python*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 1st edition, 2014.

[16] Phong Q. Nguyen and Damien Stehlé. An lll algorithm with quadratic complexity. *SIAM J. Comput.*, 39(3):874–903, 2009.

[17] Phong Q. Nguyen and Brigitte Valle. *The LLL Algorithm: Survey and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[18] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008.

[19] Nigel Poulton. *Docker Deep Dive*. Leanpub, 2nd edition, 2018.

[20] Claus Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–199, 08 1994.

[21] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[22] Kurt W. Smith. *Using Asyncio in Python*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 1st edition, 2015.

[23] Maarten van Steen Tanenbaum Andrew S. *Distributed Systems*. Pearson Education, Inc. publishing as PRENTICE HALL PTR, 1st edition, 2002.