ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη και Αξιολόγηση Παράλληλων Συστημάτων στους Αλγορίθμους Γραμμικού Προγραμματισμού

Αντώνιος Ν. Σπανός

Επιβλέποντες: **Ιωάννης Κοτρώνης,** Αναπληρωτής Καθηγητής
Ηλίας Κωνσταντινίδης, Υποψήφιος Διδάκτωρ

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2017

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη και Αξιολόγηση Παράλληλων Συστημάτων στους Αλγόριθμους Γραμμικού Προγραμματισμού

Αντώνιος Ν. Σπανός
Α.Μ.: 1115200700187

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Ιωάννης Κοτρώνης,** Αναπληρωτής Καθηγητής
Ηλίας Κωνσταντινίδης, Υποψήφιος Διδάκτωρ

ΠΕΡΙΛΗΨΗ

Η εργασία διαπραγματεύεται την μελέτη και την αξιολόγηση μεθόδων παράλληλων συστημάτων στον τομέα του γραμμικού προγραμματισμού. Αρχικά, μελετήθηκε ο αλγόριθμος της μεθόδου Simplex και οι δυνατότητες παραλληλοποίησης του. Αφού προέκυψαν τα απαραίτητα συμπεράσματα μελετήθηκαν δύο διαφορετικές παραλληλοποίησεις, η πρώτη του Mohamed Esseghir Lalami και η δεύτερη χειροποίητη με πολλαπλά νήματα στην κεντρική μονάδα επεξεργασίας μέσω Open MP. Ακολούθησε σύγκριση μεταξύ του απαιτούμενου χρόνου επεξεργασίας σε δεδομένο μέγεθος προβλήματος. Παρουσιάζεται με αυτόν τον τρόπο η δύναμη και η επεξεργαστική δεινότητα σε μια άμεση εφαρμογή του Open MP .

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Παράλληλα Συστήματα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: γραμμικός προγραμματισμός, αλγόριθμος Simplex, Open MP

ABSTRACT

This paper is about examining and evaluating the effectiveness of parallel programming in linear programming. Initially, this thesis presents the Simplex method and its possibilities of parallelization. After some preliminary study two different ways were implemented, the first one based of Mohamed Esseghir Lalami and one custom executed on CPU on multiple threads. It was followed up by a comparison of time spent during the various stages. It is beyond evident that the Open Mp standard offers brute force and compute ability in a simple way.

SUBJECT AREA: Parallel Systems

KEYWORDS: linear programming, Simplex algorithm, Open MP

Σε αυτούς που πίστεψαν στην προσπάθεια μου.

ΕΥΧΑΡΙΣΤΙΕΣ

Για την εργασία αυτή θα ήθελα να ευχαριστήσω τον επιβλέποντα αναπληρωτή Καθηγητή Κύριο Ιωάννη Κοτρώνη, για την εμπιστοσύνη που έδειξε στο πρόσωπό μου με την ανάθεση ενός ιδιαίτερα ενδιαφέροντος θέματος, ως πνευματικό επιστέγασμα της θητείας μου στο τμήμα Πληροφορικής και Τηλεπικοινωνιών, της Σχολής Θετικών Επιστημών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών. Ο καθηγητής Κύριος Κοτρώνης, καθώς επίσης και ο υποψήφιος διδάκτορας Κύριος Ηλίας Κωνσταντινίδης, στήριξαν την προσπάθειά μου, με καθοδήγησαν, μου αφιέρωσαν πολύ από τον χρόνο τους, προσφέροντάς μου τις πολύτιμες γνώσεις και συμβουλές τους, για να φτάσουμε ως εδώ.

Θερμές ευχαριστίες απευθύνω και σε όλους τους καθηγητές που είχα όλα αυτά τα χρόνια της εκπαίδευσής μου. Με βοήθησαν καθ΄ όλη τη διάρκεια της παρακολούθησης των μαθημάτων μου και αποτέλεσαν πρότυπα για μένα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η παρουσίαση της μελέτης που ακολουθεί εντάσσεται στα πλαίσια της πτυχιακής του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών (ΕΚΠΑ).

Η ιδέα της υλοποίησης του αλγόριθμου Simplex σε Open MP προέκυψε κατόπιν πρότασης των επιβλεπόντων μου κατά την διάρκεια της μελέτης και διερεύνησης των δυνατοτήτων του σύγχρονου παράλληλου προγραμματισμού που βρισκόταν σε εξέλιξη στα πλαίσια της παρούσας πτυχιακής.

Ο αλγόριθμος Simplex βρίσκει εφαρμογή σε πολλά προβλήματα της σύγχρονης Επιστήμης, τα οποία μάλιστα ανήκουν σε διαφορετικούς γνωστικούς τομείς. Η υλοποίηση που αναπτύχθηκε στην παρούσα πτυχιακή στοχεύει στην συνεισφορά στην προσπάθεια που καταβάλλεται από την επιστημονική κοινότητα για την βελτίωση της προσέγγισης ενός τόσο σημαντικού προβλήματος στα πλαίσια των δυνατοτήτων του παράλληλου προγραμματισμού που παρέχει η απλότητα του Open MP.

1. ΕΙΣΑΓΩΓΗ

1.1 Ιστορική Αναδρομή

Ο γραμμικός προγραμματισμός (Linear Programming, LP) είναι ένας σχετικά νέος μαθηματικός κλάδος, με έναρξη την μέθοδο Simplex από G. B. Dantzig το 1947. Βάση για την ανάπτυξη του γραμμικού προγραμματισμού είναι οι εφαρμογές του στην οικονομία και τη διοίκηση, καθώς συντελούν στον αμεσότερο προγραμματισμό και ακριβέστερο σχεδιασμό των επιχειρήσεων. Η ανάγκη για υλοποίηση της μεθόδου στον υπολογιστή ήταν άμεση καθώς μόνο τα πολύ μικρά προβλήματα γραμμικού προγραμματισμού είναι επιλύσιμα χωρίς υπολογιστή.

Η μέθοδος Simplex και διάφορες παραλλαγές της παραμένουν από τις αποτελεσματικότερες μεθόδους για μια μεγάλη πλειοψηφία πρακτικών προβλημάτων. Παρόλο, που δεν θεωρούνται αποτελεσματικές από θεωρητική άποψη και που καμία από αυτές δεν έχει πολυωνυμική χρονική πολυπλοκότητα, η πρακτική αποτελεσματικότητα τους είναι αδιαμφισβήτητη. Η μέθοδος Simplex σαν εκθετικής πολυπλοκότητάς πρόβλημα μπορεί να απαιτήσει υπολογιστική προσπάθεια που αυξάνεται εκθετικά με το μέγεθος των δεδομένων του προβλήματος. Ο πρώτος αποτελεσματικός αλγόριθμος του οποίου η υπολογιστική πολυπλοκότητα αυξάνεται πολυωνυμικά στο μέγεθος προβλήματος εφευρέθηκε από Khachian το 1979, γνωστός ως ελλειψοειδής αλγόριθμος, έχοντας όμως κακή απόδοση για τα πρακτικά προβλήματα συγκρινόμενα με τη μέθοδο Simplex.

Μέσα σε λίγα χρόνια ο Karmarkar παρουσίασε έναν νέο πολυωνυμικό αλγόριθμο για LP που πλησιάζει τη βέλτιστη λύση από το εσωτερικό της εφικτής περιοχής (Karmarkar 1984). Από τότε, πολλά άρθρα έχουν γραφτεί για τις μεθόδους εσωτερικού σημείου (Roos, 1997 και Wright, 1997). Όπως μαρτυρούν οι θεωρητικές ιδιότητές τους, οι μέθοδοι εσωτερικού σημείου λειτουργούν καλά για τα πρακτικά προβλήματα στην πραγματική ζωή και ξεπερνούν τη μέθοδο Simplex για τα πολύ μεγάλης κλίμακας προβλήματα (Lustig και λοιποί 1994).

Παρά την κακή απόδοσή της για τα προβλήματα μεγάλης κλίμακας, η μέθοδος Simplex λειτουργεί συχνά καλύτερα από τις μεθόδους εσωτερικού σημείου για τα μεσαία ή μικρού μεγέθους προβλήματα. Η μικρότερη υπολογιστική προσπάθεια για μερικές κατηγορίες LP προβλημάτων όπως τα προβλήματα δικτυακής βελτιστοποίησης, καθιστά την μέθοδος Simplex ελκυστικότερη από τις μεθόδους εσωτερικού σημείου. Επιπλέον έχει το σημαντικό προτέρημα σε μια σειρά σχετικών LP προβλημάτων της έλλειψης ενός ‘θερμού’ σημείου ξεκινήματος, υπολογισμός δύσκολος στις μεθόδους εσωτερικού σημείου.

Ο αλγόριθμος εξωτερικού σημείου τύπου Simplex (EPSA) αναπτύχθηκε αρχικά από τον Έλληνα ερευνητή, Κ. Παπαρρίζο το 1991 για το πρόβλημα αντιστοίχισης, ενώ αργότερα, το 1993, ο ίδιος γενίκευσε την μέθοδο εξωτερικού σημείου στο γενικό γραμμικό πρόβλημα με την ανάπτυξη ενός δυϊκού στη φύση αλγόριθμου [1] [2]. Στον πίνακα 1 συνοψίζεται η ανάπτυξη στους αλγορίθμους τύπου Simplex (Samaras 2001).

Πίνακας 1: Η ανάπτυξη Αλγορίθμων τύπου Simplex

No. |	Θεωρητική Ανάλυση	Συγγραφέας	Έτος
1	Πρωτεύων Αλγόριθμος Simplex	G. B. Dantzig	1947
2	Δυαδική Θεωρία και δυαδικός Αλγόριθμος Simplex	C. E. Lemke	1954
3	a. Degeneracy and Cycling in L.P.	E. M. L. Beale K.T. Marshall, J. W. Suurballe	1955 1969
	b. Lexicographic Rules for Anticycling	G.B.Dantzig, A.Orden, P. Wolfe P. Wolfe	1955 1963
	c. Bland's Anticycling Rule	R.G. Bland	1977
4	Complexity of Simplex Algorithm and Worst-Case behaviour Probabilistic / Average behaviour	V.Klee, G.J. Minty R. G. Jeroslow K. H. Borgwardt	1972 1973 1982
5	Primal exterior point Simplex	K. Paparrizos	1991

Στον πίνακα 2, αναφέρεται η ανάπτυξη στις μεθόδους εσωτερικού σημείου (Samaras 2001).

No.	Θεωρητική Ανάλυση	Συγγραφέας	Έτος
1	(Logarithmic barrier algorithm) Method of centers Affine scaling algorithm	K. Frisch P. Huard I. Dikin	1955 1967 1974
2	Karmarkar's algorithm	N. Karmarkar	1984
3	Recovering optimal solution	Y. Ye, M. Kojima	1987
4	Path following algorithm	M. Kojima, S. Mizuno, A. Yoshise	1989
5	Infeasible algorithms	I. Lustig	1990
6	Potential reduction algorithm	Y. Ye	1991
7	Positive semi definite programming	Y. Nesterov A. Nemirovskii	1993

Πίνακας 2: Η ανάπτυξη στις μεθόδους εσωτερικού σημείου.

Η αναθεωρημένη μέθοδος Simplex είναι μια παραλλαγή της μεθόδου Simplex που χρησιμοποιεί μια ελαφρώς τροποποιημένη μέθοδο ανανέωσης των δεδομένων σε κάθε επανάληψη. Χρησιμοποιεί τα αρχικά δεδομένα, αντί της ανανέωσης του προβλήματος και τη χρήση των αποτελεσμάτων της προηγούμενης επανάληψης καθιστώντας την πιο αποδοτική υπολογιστικά, ειδικά για τα μεγάλα και αραιά προβλήματα [3]. Στον πίνακα 3 εξηγούνται οι κύριες βελτιώσεις υπολογισμού που σχετίζονται με τους αλγόριθμους τύπου Simplex (Samaras 2001).

Πίνακας 3: Οι κύριες υπολογιστικές βελτιώσεις των αλγορίθμων της μορφής Simplex.

No.	Βελτιώσεις υπολογισμών	Συγγραφέας	Έτος
1	(Tableau format)	G. B. Dantzig	1947
2	(Revised format)	G. B. Dantzig,	1953
		W. Orchard-Hays, P. Wolfe	1954
3	(Elimination form of the inverse)	M.H. Markowitz	1954
		L.M.E. Beale	1971
		E. Hellerman	1971
4	(Sparse matrix updating techniques)	H. Bartels, H. Golub	1969
		H. Forrest, A. Tomlin	1972
5	(Pivoting techniques)	J. Harris	1973
		D. Goldfarb, K. Reid	1977
6	(Presolve procedures)	L. Brearley,	1975
		Mitra, P. Williams	

1.2 Εφαρμογές Γραμμικού Προγραμματισμού

Ο γραμμικός προγραμματισμός είναι ένα από τα πιο χρήσιμα και θεμελιώδη μαθηματικά πρότυπα προγραμματισμού που χρησιμοποιείται ευρέως στην επιχειρησιακή έρευνα και σε πολλούς άλλους τομείς των επιστήμων (Murty 1983). Η χρησιμότητα των γραμμικών και μαθηματικών προτύπων προγραμματισμού έχει καταδειχθεί κατά τη διάρκεια των τελευταίων πενήντα ετών. Ο προγραμματισμός μεταφορών, η επιλογή χαρτοφυλακίων και η ανάλυση παραγωγικότητας είναι καλά παραδείγματα των εφαρμογών του μαθηματικού προγραμματισμού στην επιχειρησιακή έρευνα (Park 1999). Στην πραγματική ζωή, πολλές εφαρμογές των μαθηματικών προτύπων προγραμματισμού προκαλούν τα πολύ μεγάλης κλίμακας προβλήματα LP. Επομένως, είναι πολύ ουσιαστικό να υπάρξουν οι μέθοδοι και οι εφαρμογές τους που λύνουν τα μεγάλα προβλήματα LP αποτελεσματικά. Υπάρχουν πολλές εφαρμογές για το γραμμικό προγραμματισμό όπως:

- Στρατιωτικές Εφαρμογές
- Βιομηχανικές Εφαρμογές
- Εφαρμογές Η/Υ
- Οικονομικές Εφαρμογές.

Οι εφαρμογές γραμμικού προγραμματισμού αντιπροσωπεύουν συχνά προβλήματα στα οικονομικά, παρά στις επιστήμες ή την εφαρμοσμένη μηχανική.

1.3 Γιατί ο Αλγόριθμος Simplex?

Οι Έλληνες και οι Ρωμαίοι εφηύραν πολλούς επιστημονικούς και εφαρμοσμένης μηχανικής αλγορίθμους, αλλά θεωρείται ότι ο όρος "αλγόριθμος" προέρχεται από το όνομα του άραβα μαθηματικού Al-khwarizmi (9ος αιώνας), ο οποίος έγραψε το βιβλίο Al -Al-jabr wa'l muqabalach, το οποίο εξελίχθηκε τελικά στα σημερινά εγχειρίδια άλγεβρας γυμνασίου. Οι Dongarra και Sullivan θέτουν μαζί έναν κατάλογο δέκα κορυφαίων αλγορίθμων του εικοστού αιώνα (Dongarra and Sullivan 2000). Σύμφωνα με αυτούς τους συγγραφείς, οι ακόλουθοι αλγόριθμοι (που απαριθμούνται κατά τη χρονολογική σειρά) είχαν τη μέγιστη επιρροή στην επιστήμη και την εφαρμοσμένη μηχανική στο παρελθόν:

1946: The Metropolis Algorithm for Monte Carlo.

1947: The Simplex Method for Linear Programming.

1950: The Krylov Subspace Iteration Methods.

1951: The Decompositional Approach to Matrix Computations.

1957: The Fortran Optimizing Compiler.

1959-1961: The QR Algorithm for Computing Eigenvalues.

1962: Quicksort Algorithm for Sorting.

1965: Fast Fourier Transform.

1977: Integer Relation Detection.

1987: Fast Multipole Method.

Ο αλγόριθμος Simplex του γραμμικού προγραμματισμού έχει ορισθεί σαν ένας από τους 10 αλγόριθμους με τη μεγίστη επιρροή στην ανάπτυξη και την πρακτική της επιστήμης και της εφαρμοσμένης μηχανικής στο 20ο αιώνα.

1.4 Γιατί ο παράλληλος υπολογισμός?

Ο παράλληλος υπολογισμός ορίζεται ως η πρακτική της χρησιμοποίησης ενός μεγάλου αριθμού συνεργαζόμενων επεξεργαστών, που επικοινωνούν μεταξύ τους για να λύσουν γρήγορα τα μεγάλα προβλήματα. Οι υπολογιστικές ανάγκες σε πόρους του Simplex έχουν οδηγήσει τους ερευνητές να εστιάσουν στην εύρεση μεθόδων παραλληλοποίησης του.

1.4.1 Η ανάγκη του παράλληλου Υπολογισμού

Δίνουμε το ακόλουθο παράδειγμα για να σκιαγραφήσουμε την ανάγκη για περισσότερη υπολογιστική δύναμη.

Υποθέτουμε ότι θέλουμε να προβλέψουμε τον καιρό στην Ευρώπη για τις επόμενες δύο ημέρες. Επίσης υποθέτουμε ότι θέλουμε να μοντελοποιήσουμε την ατμόσφαιρα από τη στάθμη θάλασσας μέχρι ένα ύψος 20 χιλιομέτρων, και πρέπει να κάνουμε μια πρόβλεψη του καιρού ανά ώρα για τις επόμενες ημέρες.

Μια τυποποιημένη προσέγγιση σε αυτόν τον τύπο προβλήματος είναι να καλυφθεί η περιοχή ενδιαφέροντος με ένα πλέγμα και έπειτα να προβλεφθεί ο καιρός σε κάθε κελί (vertex) του πλέγματος. Έτσι υποθέτουμε ότι χρησιμοποιούμε ένα κυβικό πλέγμα, με κάθε κύβο να έχει μήκος 0,1 χιλιόμετρο σε κάθε πλευρά. Δεδομένου ότι η περιοχή της Ευρώπης είναι περίπου 11 εκατομμύριο τετραγωνικά χιλιόμετρα, χρειαζόμαστε τουλάχιστον

$$11 \times 10^6 \times 20 \times 10^3 = 22 \times 10^{10} \text{ σημεία πλέγματος.} \quad (1.1)$$

Εάν χρειαστούν 100 υπολογισμοί για να καθοριστεί ο καιρός σε ένα χαρακτηριστικό σημείο του πλέγματος, κατόπιν προκειμένου να προβλεφθεί ο καιρός μια ώρα από τώρα, θα πρέπει να κάνουμε περίπου 22×10^{12} υπολογισμούς. Δεδομένου ότι θέλουμε να προβλέψουμε τον καιρό σε κάθε ώρα για 48 ώρες, πρέπει να κάνουμε συνολικά περίπου

$$22 \times 10^{12} \text{ υπολογισμούς} \times 48 \text{ ώρες} \approx 10^{15} \text{ υπολογισμούς} \quad (1.2)$$

Εάν ο υπολογιστής μας μπορεί να εκτελεί 10^9 υπολογισμούς ανά δευτερόλεπτο, θα διαρκέσει περίπου 10^{15} υπολογισμούς/ 10^9 υπολογισμούς ανά δευτερόλεπτο = 10^6 δευτερόλεπτα ≈ 11 ημέρες ! (1.3)

Με άλλα λόγια, ο υπολογισμός είναι μάταιος εάν μπορούμε να διενεργήσουμε μόνο 10^9 διαδικασίες ανά δευτερόλεπτο. Εάν, αφ' ετέρου, μπορούμε να πραγματοποιήσουμε 10^{12} υπολογισμούς ανά δευτερόλεπτο, θα μας πάρει περίπου 16 λεπτά για να πραγματοποιήσει τους υπολογισμούς. Έτσι θα είμαστε πραγματικά σε θέση να κάνουμε μια πλήρη πρόβλεψη του καιρού κατά τη διάρκεια κάθε μιας από τις επόμενες 48 ώρες. Δεν είναι δύσκολο να φανταστούμε τις απλές τροποποιήσεις σε αυτό το πρόβλημα έτσι ώστε οι 10^9 διαδικασίες ανά δευτερόλεπτο να μην είναι ικανοποιητικές. Παραδείγματος χάριν, να αντικαταστήσουμε την Ευρώπη με ολόκληρη τη γη. Κατόπιν η περιοχή θα πήγαινε από $11x10^6$ σε περίπου $5x10^8$ τετραγωνικά χιλιόμετρα Έτσι ο απαραίτητος χρόνος υπολογισμού θα αυξανόταν από 16 λεπτά σε 12 ώρες περίπου και οι πρώτες 11 προβλέψεις μας θα ήταν άχρηστες. Επιπλέον, δεν είναι δύσκολο να βρούμε τα απολύτως διαφορετικά προβλήματα με απέραντα μεγαλύτερη υπολογιστική δύναμη από αυτήν που κατέχουμε τώρα. Για παράδειγμα, οι λεπτομερείς ατομικού επιπέδου προσομοιώσεις των βιομορίων και οι πολυάριθμοι τύποι προσομοιώσεων που θα επέσπευδαν το σχέδιο και η κατασκευή των ολοκληρωμένων κυκλωμάτων απαιτούν απέραντα τη μεγαλύτερη υπολογιστική δύναμη από αυτή που εμείς κατέχουμε.

Είναι προφανές ότι ο μόνος τρόπος γύρω από το πρόβλημα αυτό είναι να χρησιμοποιήσουμε τον παραλληλισμό. Η ιδέα εδώ είναι ότι εάν αρκετές πράξεις εκτελούνται ταυτόχρονα, τότε ο απαιτούμενος για τον υπολογισμό χρόνος μπορεί να μειωθεί σημαντικά.

1.5 Οργάνωση της πτυχιακής

Η πτυχιακή αποτελείται 7 κεφάλαια τα οποία οργανώνονται ως εξής: κάθε κεφάλαιο αρχίζει με εισαγωγή στην οποία αναφέρονται πληροφορίες και ιδέες για τα περιεχόμενα του κεφαλαίου. Στο επόμενο κεφάλαιο, εισάγουμε μερικές παραλλαγές Αλγορίθμων της μορφής Simplex, τον Πρωταρχικό Αλγόριθμο Simplex και τον αντίστοιχο Δυαδικό Αλγόριθμο Simplex.

Στο κεφάλαιο τρία γίνεται παρουσίαση του ανοιχτού προτύπου Open MP καθώς και τα κύρια τμήματα του συντακτικού του, ενώ στο τέταρτο ακολουθεί η εφαρμογή Open MP στον Αλγόριθμο Simplex. Στο πέμπτο κεφάλαιο παρουσιάζουμε την υπολογιστική μελέτη της υλοποίησης Lalami, την σύγκριση δηλαδή της παράλληλης υλοποιήσης και της σειριακής. Στο έκτο κεφάλαιο περιγράφεται μια υλοποίηση του αλγορίθμου Simplex ενώ στο τελευταίο κεφάλαιο περιγράφονται τα συμπεράσματα από την έρευνα και προτάσεις για επέκτασή της.

2. SIMPLEX

2.1 Εισαγωγή

Σ' αυτό το κεφάλαιο περιγράφουμε δύο τύπους αλγορίθμων Simplex, τον πρωτεύοντα αλγόριθμο Simplex και τον δυϊκό αλγόριθμο Simplex ενώ απαντάμε στην ερώτηση "ποια είναι η γεωμετρική ερμηνεία της μεθόδου Simplex;". [4]

2.2 Το Πρωτεύον Πρόβλημα

Μπορούμε να γράψουμε τα γραμμικά προβλήματα ως εξής:

$$max\ z\ =\ c^T x$$
$$s.t.\quad Ax \otimes b$$
$$x\ \geq\ 0$$

Όπου c, x ∈ $\Re^n$, b ∈ $\Re^m$, A ∈ $\Re^{mxn}$, c^T είναι το ανάστροφο ενός διανύσματος c και ⊗ αναφέρεται σε οποιαδήποτε από τις σχέσεις (=,≤ and ≥).

Επίσης μπορούμε να γράψουμε τα γραμμικά προγράμματα με έναν άλλο τρόπο:

$$\max c_j x_j$$
$$subject\ to\colon \sum_{j=1}^{n} a_{ij} x_j\ \otimes\ b_i \qquad ,i = 1 \ldots m$$
$$x_j \geq 0 \qquad ,j = 1 \ldots n$$

2.2.1 Ορολογία

Η γραμμική συνάρτηση $c^T x$, την οποία προσπαθούμε να βελτιστοποιήσουμε, καλείται αντικειμενική συνάρτηση. Μια εφικτή λύση είναι μια ανάθεση τιμών στις μεταβλητές απόφασης $x_1, x_2, \ldots, x_n$, έτσι ώστε να ικανοποιούνται όλοι οι περιορισμοί. Μια εφικτή λύση που βελτιστοποιεί την αντικειμενική συνάρτηση είναι μια βέλτιστη λύση. Ένα γραμμικό πρόγραμμα που δεν έχει εφικτές λύσεις καλείται αδύνατο, έχει εφικτές λύσεις αλλά δεν έχει βέλτιστη λύση.

2.2.2 Ελαχιστοποίηση και Μεγιστοποίηση

Ένας άλλος χειρισμός προβλήματος είναι να μετατραπεί ένα πρόβλημα μεγιστοποίησης σε ένα πρόβλημα ελαχιστοποίησης και αντιθέτως. Σημειώστε ότι σε οποιαδήποτε περιοχή ισχύει η σχέση:

$$Μεγιστοποίηση \sum_{j=1}^{n} c_j x_j\ =\ Ελαχιστοποίηση\ -\sum_{j=1}^{n} c_j x_j$$

Έτσι ένα πρόβλημα μεγιστοποίησης (ελαχιστοποίησης) μπορεί να μετατραπεί σε ένα πρόβλημα ελαχιστοποίησης (μεγιστοποίησης) πολλαπλασιάζοντας τους συντελεστές της αντικειμενικής συνάρτησης με -1. Μετά τη βελτιστοποίηση του νέου προβλήματος, η βέλτιστη αντικειμενική τιμή του παλιού προβλήματος είναι -1 φορές της βέλτιστης τιμής του νέου προβλήματος.

2.2.3 Χαλαρές (Slack) και πλεονασματικές (Surplus) μεταβλητές

Έστω, μια ανισότητα, για παράδειγμα του περιορισμού r, της ακόλουθης μορφής:

$$\sum_{j=1}^{n} a_{rj}x_j \leq b_r \qquad (1.1)$$

Εισάγουμε μια νέα μεταβλητή, $s_r \geq 0$, που την ονομάζουμε χαλαρή μεταβλητή (slack), έτσι ώστε:

$$\sum_{j=1}^{n} a_{rj}x_j + s_r = b_r \qquad (1.2)$$

Έτσι,

$$s_r = b_r - \sum_{j=1}^{n} a_{rj}x_j \qquad (1.3)$$

Περιφραστικά, s_r είναι η (μη αρνητική) διαφορά μεταξύ της σταθεράς του δεξιού μέλους και τιμής του αριστερού μέλους. Από μια μαθηματική άποψη, το s_r μας επιτρέπει να εκφράσουμε μια ανισότητα της μορφής (1.1) με ένα καταλληλότερο σχήμα ισότητας.

Στην συνέχεια, θεωρούμε μια ανισότητα για παράδειγμα περιορισμού t, της μορφής:

$$\sum_{j=1}^{n} a_{tj}x_j \geq b_t \quad (1.4)$$

Στην περίπτωση αυτή, εισάγουμε μια νέα μεταβλητή, $s_t \geq 0$, έτσι ώστε:

$$\sum_{j=1}^{n} a_{tj}x_j - s_t = b_t \quad (1.5)$$

Σημειώνουμε ότι στην περίπτωση αυτή:

$$s_t = \sum_{j=1}^{n} a_{rj}x_j - b_t \quad (1.6)$$

Δηλαδή η μεταβλητή s_t αντιπροσωπεύει τη μη αρνητική διαφορά μεταξύ της αριστερής πλευράς και δεξιάς πλευράς της ανισότητας στην (1.4). Μια τέτοια μεταβλητή καλείται μεταβλητή πλεονάσματος (surplus) ή πλεονασματική μεταβλητή.

2.2.4 Το πρόβλημα κατανομής των πόρων

Ένα εργοστάσιο παράγει δύο προϊόντα: καρέκλες και πίνακες. Πραγματοποιούν ένα κέρδος 40€ σε κάθε παραχθείσα καρέκλα και 50€ σε κάθε πίνακα. Μια καρέκλα απαιτεί τους ακόλουθους πόρους για να παραχθεί: 2 ανθρωποώρες, 3 ώρες μηχανών και 1 μονάδα ξύλου. Ο πίνακας απαιτεί 2 ανθρωποώρες, 1 ώρα μηχανών και 4 μονάδες του ξύλου. Το εργοστάσιο έχει 60 ανθρωποώρες, 75 ώρες μηχανών, και 84 μονάδες ξύλου για κάθε ημέρα για την παραγωγή αυτών των δύο προϊόντων. Πώς θα έπρεπε οι πόροι (ανθρωποώρες, ώρες μηχανής, και ξύλο) να διατεθούν μεταξύ των δύο προϊόντων προκειμένου να μεγιστοποιηθεί το κέρδος του εργοστασίου;

Εάν θέσουμε x_1 τον αριθμό των καρεκλών που παράγονται ανά μέρα και x_2 τον αριθμό των πινάκων που παράγονται ανά μέρα, τότε το πρόβλημα μπορεί να τεθεί μαθηματικά ως εξής:

$$\text{Max } P=40x_1 + 50x_2$$

$$\text{Subject to: } 2x_1 + 2x_2 \leq 60,$$

$$3x_1 + x_2 \leq 75,$$

$$x_1 + 4x_2 \leq 84,$$

$$\text{Και τα όρια } x_1 \geq 0,\ x_2 \geq 0.$$

Η συνάρτηση προς μεγιστοποίηση, $40x_1 + 50x_2$, αναπαριστά το κέρδος, και οι περιορισμοί δηλώνουν ότι οι συνολικές ανθρωποώρες, ώρες μηχανών και ξύλου που θα χρησιμοποιηθούν δεν μπορούν να υπερβούν τα διαθέσιμα. Τα όρια θέτουν τα προφανή γεγονότα ότι το εργοστάσιο δεν μπορεί να παραγάγει έναν αρνητικό αριθμό καρεκλών ή πινάκων.

2.3 Πως δουλεύει γεωμετρικά η μέθοδος Simplex;

Ας θεωρήσουμε το παρακάτω πρόβλημα γραμμικού προγραμματισμού:

$$\text{Μεγιστοποίησε } x_1 + x_2$$

$$\text{Με περιορισμούς } 2x_1 + x_2 \leq 14$$

$$-x_1 + 2x_2 \leq 8 \qquad (2.2.1)$$

$$2x_1 - x_2 \leq 10$$

$$x_1 \geq 0,\ x_2 \geq 0$$

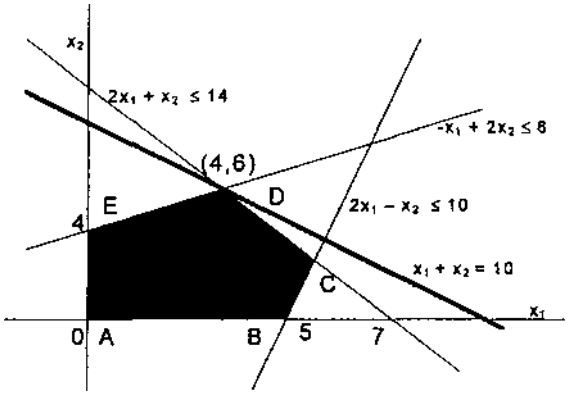
Η μέθοδος Simplex αποτελείται από 2 βασικά βήματα που καλούνται «φάσεις». Στην πρώτη φάση προσπαθούμε να βρούμε μια εφικτή λύση του προβλήματος. Για προβλήματα μικρού μεγέθους ή ακόμη και για μεγαλύτερα προβλήματα κάποιων μορφών η παραπάνω λύση δεν είναι δύσκολη. Συχνά μια τετριμμένη λύση, όπως η $x_1 = 0$, είναι μια εφικτή λύση.

Αφού βρεθεί μια εφικτή λύση στο πρόβλημα, η μέθοδος Simplex προσπαθεί να βελτιώσει σε κάθε επανάληψη την τιμή της συνάρτησης κόστους (αντικειμενική συνάρτηση). Αυτό επιτυγχάνεται βρίσκοντας μια μεταβλητή του προβλήματος η τιμή της οποίας μπορεί να αυξηθεί, μειώνοντας ταυτόχρονα κάποια άλλη μεταβλητή έτσι ώστε να προκύπτει βελτίωση της τιμής της συνάρτησης κόστους. Αυτό μπορεί να οπτικοποιηθεί γραφικώς, μετακίνησης στις πλευρές του εφικτού (πολυγώνου) από κορυφή σε κορυφή.

Το εφικτό σύνολο του προβλήματος 2.2.1 μπορεί να παρασταθεί σε 2 διαστάσεις όπως φαίνεται στην Σχήμα 1. Οι μη αρνητικοί περιορισμοί $x_1 > 0$ και $x_2 > 0$ περιορίζουν το εφικτό σύνολο στο πρώτο τεταρτημόριο. Οι υπόλοιποι 3 περιορισμοί είναι εξισώσεις ευθείας στο επίπεδο $x_1 - x_2$ όπως φαίνεται στο σχήμα. Η συνάρτηση κόστους $x_1 + x_2$, μπορεί να αναπαρασταθεί με μια ευθεία γραμμή που έχει συντελεστή διεύθυνσης ίσο με -1. Η τεταγμένη της γραμμής της συνάρτησης κόστους ισούται με την τιμή της συνάρτησης κόστους για οποιαδήποτε λύση που βρίσκεται επάνω στη γραμμή. Η έντονη γραμμή του σχήματος 1 αναπαριστά τη βέλτιστη λύση του προβλήματος, διότι είναι μια γραμμή που έχει συντελεστή διεύθυνσης ίσο με -1 και μέγιστη τεταγμένη ίση με 10 που τέμνει το εφικτό σύνολο. Η τιμή της συνάρτησης κόστους για την εφικτή λύση ισούται με 10 και η ευθεία $x_1 + x_2 = 10$ της συνάρτησης κόστους έχει ακριβώς ένα σημείο με συντεταγμένες $x_1 = 4$ και $x_2 = 6$ μέσα στο εφικτό σύνολο.

Η μέθοδος Simplex προσπαθεί να βρει μια εφικτή λύση (ένα σημείο), και μετά μετακινεί αυτό το σημείο προς οποιαδήποτε κορυφή του εφικτού συνόλου, με στόχο τη βελτίωση της συνάρτησης κόστους. Τελικά φθάνουμε σε μια κορυφή από την οποία αν μετακινη-

θούμε δεν βελτιώνεται η συνάρτηση κόστους. Αυτή είναι και η βέλτιστη λύση. Στο παράδειγμά μας η τιμή $x_1 = 0$ και $x_2 = 0$ είναι μια τετριμμένη εφικτή λύση, που δίνει στη συνάρτηση κόστους την τιμή 0. Αυτή είναι η κορυφή A του σχήματος 1. Από το σημείο αυτό μπορούμε να μετακινηθούμε είτε προς το σημείο B είτε προς το E. Το σημείο E (0, 4) αυξάνει την τιμή της συνάρτησης κόστους σε 4 ενώ το σημείο B (5, 0) την αυξάνει στην τιμή 5. Επειδή το σημείο B μας δίνει μεγαλύτερη βελτίωση, το επιλέγουμε στην πρώτη επανάληψη (θα μπορούσαμε επίσης να είχαμε επιλέξει και το σημείο E, κάτι που θα μας οδηγούσε πιο γρήγορα στη βέλτιστη λύση). Η τιμή της μεταβλητής x_1 αυξάνει από 0 σε 5, ενώ η τιμή της x_2 παραμένει 0.



Σχήμα 1: Γραφική επίλυση ενός γραμμικού προβλήματος

Από το σημείο B, ελέγχουμε αν μια μετακίνηση προς το σημείο C είναι επωφελής (γνωρίζουμε ότι μια ενδεχόμενη μετακίνηση προς το σημείο A δεν είναι επωφελής). Το σημείο C (6, 2) δίνει την τιμή 8 στη συνάρτηση κόστους, γεγονός που αποτελεί βελτίωση. Συνεπώς αυξάνουμε την τιμή της μεταβλητής χι από 5 σε 6 και επειδή ισχύουν οι περιορισμοί $2\ x_1 – x_2 \leq 10$ και $– x_1 + 2\ x_2 \leq 8$, πρέπει να αυξήσουμε και την τιμή της x_2 από 0 σε 2. Από το σημείο C, ελέγχουμε αν μια μετακίνηση προς το σημείο D βελτιώνει την κατάσταση. Για το σημείο D (4, 6) η τιμή της συνάρτηση κόστους ισούται με 10 και έτσι αποδεχόμαστε την μετακίνηση, αυξάνοντας την τιμή της x_2 από 2 σε 6. Αυτό σημαίνει ότι η τιμή της x_1 πρέπει να μειωθεί από 6 σε 4 λόγω των περιορισμών που ισχύουν. Τώρα επειδή μια μετακίνηση είτε προς το σημείο E (τιμή της συνάρτησης κόστους ίση με 4) είτε προς το σημείο C (τιμή της συνάρτησης κόστους ίση με 8) μειώνει την τρέχουσα τιμή της συνάρτησης κόστους, συμπεραίνουμε ότι το σημείο D είναι η βέλτιστη λύση του προβλήματος.

2.4 Πρωτεύων Αλγόριθμος Simplex

Ας θεωρήσουμε το παρακάτω γραμμικό πρόβλημα:

$$\text{Μεγιστοποίησε } z = cx$$

$$\text{Με περιορισμούς } Ax = b \quad (2.3.1)$$

$$x \geq 0$$

Μια βασική εφικτή λύση στο παραπάνω πρόβλημα αντιστοιχεί σε ένα ακραίο σημείο της εφικτής περιοχής και χαρακτηρίζεται μαθηματικά από τον διαμερισμό του πίνακα A σε έναν μη μοναδιαίο πίνακα βάσης B και στον πίνακα N που αποτελείται από τις μη βασικές στήλες. Δηλαδή,

$$A=(B:N) \quad (2.3.2)$$

Βασιζόμενοι σε αυτό το διαμερισμό, το γραμμικό σύστημα Ax=b μπορεί να ξαναγραφεί ως

$$Bx_B + Nx_n = b \quad (2.3.3)$$

Αυτό απλοποιείται περαιτέρω στο σύστημα

$$x_B+B^{-1}Nx_n=B^{-1}b \quad (2.3.4)$$

και λύνοντας ως προς x_B, με άγνωστο το x_n, παίρνουμε

$$x_B= B^{-1}b - B\,N\,x_n \quad (2.3.5)$$

Τώρα θέτοντας $x_n = 0$, βλέπουμε ότι η εξίσωση 2.3.5 έχει ως αποτέλεσμα την

$x_B = B^{-1}b$. Η λύση

$$x = \begin{pmatrix} x_b \\ x_n \end{pmatrix} = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$$

καλείται βασική λύση, το διάνυσμα x_B καλείται διάνυσμα των βασικών μεταβλητών, και το x_n καλείται διάνυσμα των μη βασικών μεταβλητών. Επιπροσθέτως αν $x_B = B^{-1}b \geq 0$, τότε η παρακάτω λύση

$$x = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$$

καλείται βασική εφικτή λύση.

Τώρα ας θεωρήσουμε την αντικειμενική συνάρτηση z = cx. Διαμερίζοντας το διάνυσμα κόστους c σε βασικά και μη-βασικά μέρη π.χ. c = (c_B, c_N), η αντικειμενική συνάρτηση γράφεται ως

$$z=c_Bx_B+c_Nx_N \quad (2.3.6)$$

Αντικαθιστώντας το x_B (εξίσωση 2.3.5) στην εξίσωση 2.3.6 παίρνουμε

$$z=c_B(B^{-1}b-B^{-1}Nx_n)+c_Nx_N \quad (2.3.7)$$

Η παραπάνω εξίσωση μπορεί να γραφεί ως:

$$z=c_BB^{-1}b-(c_BB^{-1}N-c_N)x_N \quad (2.3.8)$$

Συνεπώς το z ισούται με τη σταθερά $c_BB^{-1}b$ μείον τον όρο $(c_BB^{-1}N-c_N)x_N$. Αν θέσουμε $x_N = 0$, η εξίσωση 2.3.8 μετατρέπεται στην $z=c_BB^{-1}b$. Η τελευταία είναι η αντικειμενική τιμή που αντιστοιχεί στην τρέχουσα βασική εφικτή λύση. Έτσι η τρέχουσα λύση ακραίου σημείου μπορεί να παρασταθεί σε κανονική μορφή όπως παρακάτω:

$$z=c_BB^{-1}b-(c_BB^{-1}N-c_N)x_N \quad (2.3.9)$$

$$x_B= B^{-1}b – B^{-1}Nx_N \quad (2.3.10)$$

Η τρέχουσα βασική εφικτή λύση δίνεται από τις παρακάτω εξισώσεις

$$z= c_BB^{-1}b \quad (2.3.11)$$

$$x = \begin{pmatrix} x_b \\ x_n \end{pmatrix} = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix} \geq 0 \quad (2.3.12)$$

Αν τώρα αναδιατάξουμε τους όρους, έτσι ώστε όλες οι αναθεωρημένες μεταβλητές να βρίσκονται στο αριστερό μέρος της εξίσωσης ενώ οι σταθερές να βρίσκονται στο δεξιό μέρος παίρνουμε:

$$z+(c_BB^{-1}N-c_N)x_N = c_BB^{-1}b \quad (2.3.13)$$

$$x_B + B^{-1}Nx_N = B^{-1}b \quad (2.3.14)$$

Το Simplex ταμπλό είναι απλά ένας πίνακας που χρησιμοποιείται για την αποθήκευση των συντελεστών των εξισώσεων 2.3.13 και 2.3.14. Η επάνω γραμμή (γραμμή 0) του ταμπλό αποτελείται από τους συντελεστές της αντικειμενικής εξίσωσης 2.3.13 και το σώμα του ταμπλό (γραμμές 1 έως m) αποθηκεύει τους συντελεστές των εξισώσεων των περιορισμών (2.3.14). Η γενική μορφή του ταμπλό φαίνεται στον πίνακα 6 και αν ξαναγραφεί πιο συμπυκνωμένα προκύπτει η μορφή που φαίνεται στον πίνακα 7.

Πίνακας 4: Γενική μορφή ταμπλό

	z	x_B	x_N	Δεξί μέρος	
z	1	0	$c_B B^{-1} N - c_N$	$c_B B^{-1} b$	Γραμμή 0
x_B	0	1	$B^{-1} N$	$B^{-1} b$	Γραμμή 1-m

Πίνακας 5: Συμπυκνωμένη μορφή ταμπλό

	z	x	Δεξί μέρος	
z	1	$c_B B^{-1} A - c$	$c_B B^{-1} b$	Γραμμή 0
x_B	0	$B^{-1} A$	$B^{-1} b$	Γραμμή 1-m

Τώρα είμαστε έτοιμοι να συνοψίσουμε τα βήματα του αλγορίθμου Simplex όπως αυτά εφαρμόζονται στο Simplex ταμπλό.

Περιγραφή του Πρωτεύοντος Αλγορίθμου Simplex

Βήμα 0: (Αρχικοποίηση)

Ξεκίνα με ένα εφικτό βασικό σημείο και κατασκεύασε το αντίστοιχο Simplex ταμπλό.

Βήμα 1:(Επιλογή εισερχόμενης μεταβλητής)

Αν $a_{0j}>=0$ για j=1, 2, ..., n, STOP. Η λύση είναι βέλτιστη. Με κάποιον κανόνα περιστροφής, επέλεξε την εισερχόμενη μεταβλητή με δείκτη s:

$a_{0s}>0$, π.χ.

$$a_{0s}=\min\{a_{0j}: a_{0j}<0, \text{ για } j=1,2,...,n\}$$

Βήμα 2: (Επιλογή εξερχόμενης μεταβλητής)

Θέσε I = { i : a_{is} > 0}. Αν $I = \emptyset$, STOP. Το πρόβλημα είναι απεριόριστο. Με κάποιον κανόνα περιστροφής, βρες το δείκτη της στήλης περιστροφής r:

$$\frac{b_r}{a_{rs}} = \min\{\frac{b_i}{a_{is}} : i \in I\}$$

Βήμα 3: (Περιστροφή)

Σχημάτισε το επόμενο ταμπλό με τη μεταβλητή περιστροφής a_{rs} π.χ.

Θέσε $a_{rj} \leftarrow \frac{a_{rj}}{a_{rs}}$ όπου j=1, 2,... ,n, n+1

$a_{ij} \leftarrow a_{ij} - \frac{a_{rj}}{a_{rs}} a_{is}$ όπου i= 0,1,2,... m (i≠r) και j = 1,2,..., n, n+1 a„

και πήγαινε στο βήμα 1.

Με τον Πρωτεύοντα Αλγόριθμο Simplex επιλύουμε 2 προβλήματα. Το πρώτο πρόβλημα είναι βέλτιστο και το δεύτερο απεριόριστο.

Παράδειγμα 2.3.1:

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον πρωτεύοντα αλγόριθμο Simplex.

$$\max z = 2x_1 + 3x_2$$

$$\text{s.t } x_1-2x_2+x_3=4$$

$$2x_1+x_2+x_4=18$$

$$x_2+x_5=10$$

$$x_1, x_2, x_3, x_4 \text{ και } x_5 \geq 0$$

Λύση:

Επανάληψη 1

Βήμα 0: (Αρχικοποίηση)

Από το αρχικό πρόβλημα είναι φανερό ότι τα βασικά εφικτά σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 4$, $x_4 = 18$ και $x_5 = 10$. Έτσι προκύπτει το παρακάτω αρχικό Simplex ταμπλό:

	z	x_1	x_2	x_3	x_4	x_5	Δεξί Μέρος
z	1	-2	-3	0	0	0	0
x_3	0	1	-2	1	0	0	4
x_4	0	2	1	0	1	0	18
x_5	0	0	1	0	0	1	10

Βήμα 1: (επιλογή εισερχόμενης μεταβλητής)

Προσδιορισμός της στήλης περιστροφής και επιλογή εισερχόμενης μεταβλητής

$$a_{0s} = \min \{a_{01}, a_{02}\} = \min \{-2, -3\} = -3 = a_{02}$$

Επειδή s = 2, η μεταβλητή x_2 μπαίνει στη βάση.

Βήμα 2: (επιλογή εξερχόμενης μεταβλητής)

Θέσε I = {2, 3}και βρες το r = 3 διότι

$$\frac{b_r}{a_{r3}} = \min\left\{\frac{b_2}{a_{23}}, \frac{b_3}{a_{33}}\right\} = \min\left\{\frac{18}{1}, \frac{10}{1}\right\} = 10$$

Συνεπώς η μεταβλητή x_5 βγαίνει από τη βάση.

Βήμα 3: (Περιστροφή)

Σχημάτισε το επόμενο Simplex ταμπλό με το στοιχείο περιστροφής a_{32} π.χ.

	z	x_1	x_2	x_3	x_4	x_5	Δεξί Μέρος
z	1	-2	0	0	0	3	30
x_3	0	1	0	1	0	2	4
x_4	0	2	0	0	1	-1	8
x_2	0	0	1	0	0	1	10

Επειδή $a_{01} < 0$, στην αντικειμενική γραμμή, επαναλαμβάνουμε τον πρωτεύοντα αλγόριθμο Simplex.

Επανάληψη 2

Βήμα 1: (επιλογή εισερχόμενης μεταβλητής)

Προσδιορισμός της στήλης περιστροφής. Επέλεξε a_{0s}, = min $\{a_{01}, a_{02}\}$ = min {-2,0} = -2 = a_{01}. Το s = 1, συνεπώς η μεταβλητή x_1 μπαίνει στη βάση.

Βήμα 2: (επιλογή εξερχόμενης μεταβλητής). Θέσε I ={1,2} και βρες το r = 2 διότι

$$\frac{b_r}{a_{r3}} = \min\left\{\frac{b_1}{a_{13}}, \frac{b_2}{a_{23}}\right\} = \min\left\{\frac{24}{1}, \frac{8}{1}\right\} = 4$$

Συνεπώς η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα 3: (Περιστροφή)

Σχημάτισε το επόμενο Simplex ταμπλό με το στοιχείο περιστροφής a_{21} π.χ.

	z	x_1	x_2	x_3	x_4	x_5	Δεξί Μέρος
z	1	0	0	0	1	2	38
x_3	0	0	0	1	-1/2	5/2	20
x_1	0	1	0	0	1/2	-1/2	4
x_2	0	0	1	0	0	1	10

Επειδή $a_{0j} \geq 0$, όπου j =1, 2,...5 στην αντικειμενική γραμμή STOP. Η λύση είναι βέλτιστη και είναι η $x_1 = 4$, $x_2 = 10$, $x_3 = 20$, $x_4 = 0$, $x_5 = 0$. Η αντικειμενική συνάρτηση είναι $z = 2x_1 + 3x_2 = 38$.

Παράδειγμα 2.3.2

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον πρωτεύοντα αλγόριθμο Simplex.

$$\max z = 5x_1 + 3x_2$$

$$s.t\ -x_1 + x_2 \leq 4$$

$$x_1 - 2x_2 \leq 6$$

$$x_1 \text{ και } x_2 \geq 0$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_3 και x_4 έτσι ώστε το γραμμικό πρόβλημα να μετατραπεί στο παρακάτω:

$$\max z = 5x_1 + 3x_2$$

$$s.t\ -x_1 + x_2 + x_3 = 4$$

$$x_1 - 2x_2 + x_4 = 6$$

$$x_1, x_2, x_3 \text{ και } x_4 \geq 0$$

Επανάληψη 1

Βήμα 0 (Αρχικοποίηση)

Σχημάτισε το αρχικό πρόβλημα. Είναι φανερό ότι τα βασικά εφικτά σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 4$ και $x_4 = 6$. Έτσι κατασκευάζουμε το αρχικό Simplex ταμπλό

	z	x_1	x_2	x_3	x_4	Δεξί Μέρος
z	1	-5	-3	0	0	0
x_3	0	-1	1	1	0	4
x_4	0	1	-2	0	1	6

Βήμα 1: (επιλογή εισερχόμενης μεταβλητής)

Προσδιόρισε τη στήλη περιστροφής και επέλεξε

$$a_{0s} = \min\{a_{01}, a_{02}\} = \min\{-5, -3\} = -5 = a_{01}$$

Το s = 1, συνεπώς η μεταβλητή x_1 μπαίνει στη βάση.

Βήμα 2: (επιλογή εξερχόμενης μεταβλητής). Θέσε I = {2} και βρες το r = 2 διότι

$$\frac{b_r}{a_{r1}} = \min\left\{\frac{b_2}{a_{21}}\right\} = \min\left\{\frac{6}{1}\right\} = 6$$

Συνεπώς η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα 3: (Περιστροφή)

Σχημάτισε το επόμενο Simplex ταμπλό με το στοιχείο περιστροφής a_{21} π.χ.

	z	x_1	x_2	x_3	x_4	Δεξί Μέρος
z	1	0	-13	0	5	30
x_3	0	0	-1	1	1	10
x_1	0	1	-2	0	1	6

Επειδή υπάρχει το $a_{02} < 0$, στην αντικειμενική γραμμή, επαναλαμβάνουμε τον πρωτεύοντα αλγόριθμο Simplex.

Επανάληψη 2

Βήμα 1: (επιλογή εισερχόμενης μεταβλητής)

Προσδιόρισε τη στήλη περιστροφής και επέλεξε

$$a_{0s} = \min\{a_{02}\} = \min\{-13\} = -13 = a_{02}$$

Το s = 2, συνεπώς η μεταβλητή x_2 μπαίνει στη βάση.

Βήμα 2: (επιλογή εξερχόμενης μεταβλητής)

Θέσε $I = \emptyset$ έτσι ώστε το πρόβλημα να είναι απεριόριστο.

2.5 Το Δυικό πρόβλημα

Κάθε πρόβλημα μεγιστοποίησης γραμμικού προγραμματισμού έχει ένα ισοδύναμο πρόβλημα ελαχιστοποίησης, το δυϊκό πρόβλημα. [5] Αυτά τα δύο προβλήματα παρέχουν χρήσιμες πληροφορίες το ένα για το άλλο. Παραδείγματος χάριν, η τιμή της δυικής αντικειμενικής συνάρτησης παρέχει έναν ανώτερο όριο για την τιμή της πρωταρχικής αντικειμενικής συνάρτησης. Αυτό και άλλα αποτελέσματα συζητούνται κατωτέρω.

Η κανονική μορφή ενός προβλήματος γραμμικού προγραμματισμού είναι μια μορφή στην οποία η αντικειμενική συνάρτηση μεγιστοποιείται, όλοι οι περιορισμοί είναι της μορφής(>=), και όλες οι μεταβλητές είναι μη αρνητικές. Συνεπώς, το πρωτεύον πρόβλημα, με κανονική μορφή είναι:

$$max\ z\ =\ c^T x$$
$$s.t. Ax\ \geq\ b \qquad (P)$$
$$x\ \geq\ 0$$

Εάν (P) διευκρινίζεται να είναι το πρωτεύον πρόβλημα γραμμικού προγραμματισμού, το δυϊκό πρόβλημα γραμμικού προγραμματισμού του δίνεται από (DP):

$$min\ z\ =\ b^T w$$
$$s.t. Aw\ \leq\ c \qquad (DP)$$
$$w\ \geq\ 0$$

Αναφέρουμε τώρα μερικούς κανόνες υπολογισμού του δυϊκού προβλήματος.

1. Η αντικειμενική συνάρτηση του πρωτεύοντος μεγιστοποιείται, η αντικειμενική συνάρτηση δυϊκού ελαχιστοποιείται.
2. Το πρόβλημα μεγιστοποίησης πρέπει να έχει όλους τους περιορισμούς (≥) και το πρόβλημα ελαχιστοποίησης πρέπει να έχει όλους τους περιορισμούς (≤).
3. Όλες οι πρωτεύουσες και δυικές μεταβλητές πρέπει να είναι μη αρνητικές.
4. Κάθε περιορισμός σε ένα πρόβλημα αντιστοιχεί σε μια μεταβλητή του άλλου (και αντίστροφα).
5. Τα στοιχεία της δεξιάς πλευράς των περιορισμών σε ένα πρόβλημα είναι οι αντίστοιχοι συντελεστές της αντικειμενικής συνάρτησης στο άλλο πρόβλημα.
6. Ο πίνακας των σταθερών συντελεστών για ένα πρόβλημα είναι ο ανάστροφος του πίνακα σταθερών συντελεστών για το άλλο πρόβλημα.

Χρησιμοποιώντας τις σχέσεις στον πίνακα 4, είναι δυνατό να γραφτεί το δυϊκό πρόβλημα για ένα δεδομένο γραμμικό πρόγραμμα χωρίς το ενδιάμεσο βήμα του μετασχηματισμού του προβλήματος στην κανονική μορφή.

Πίνακας 6: Πρωταρχικές-δυαδικές σχέσεις

Μεγιστοποίηση προβλήματος	Ελαχιστοποίηση προβλήματος
Σταθερές	Μεταβλητές
≤	≥0
≥	≤0
=	Χωρίς περιορισμό
Μεταβλητές	Σταθερές
≥0	≥
≤0	≤
Χωρίς περιορισμό	=

Παράδειγμα 2.4.1

Να υπολογιστεί το δυϊκό πρόβλημα του επόμενου προβλήματος.

$$\text{Max } 4x_1+2x_2-x_3$$

$$\text{Subject to } x_1+x_2+x_3=20$$

$$2x_1-x_2\geq 6$$

$$3x_1+2x_2+x_3\leq 40$$

$$x_3 \text{ unrestricted}$$

Θέτοντας w_1, w_2 και w_3 τις δυικές μεταβλητές, τα αποτελέσματα του Πίνακα 4 παράγουν το ακόλουθο δυαδικό πρόβλημα:

$$\text{Min } 20w_1+6w_2+40w_3$$

$$\text{Subject to } w_1+2w_2+3w_3\geq 4$$

$$w_1-w_2+w_3\geq 2$$

$$w_1+ w_3=-1$$

$$w_1 \text{ unrestricted}$$

$$w_2\leq 0$$

$$w_3\geq 0$$

Ο πίνακας 5 παρουσιάζει διαφορετικούς συνδυασμούς πιθανούς για ένα πρωτεύον - δυϊκό ζευγάρι. Το δυϊκό LP αντιπροσωπεύει έναν γραμμικό συνδυασμό των πρωτευόντων περιορισμών LP. Η σημασία της δυϊκότητας, γνωστή ως Θεώρημα Δυϊκότητας, είναι ότι κάθε εφικτή λύση του δυϊκού LP παράγει ένα όριο στη βέλτιστη τιμή του πρωτεύοντος LP. Το Θεώρημα Δυϊκότητας παρέχει σε μας τη δυνατότητα να βεβαιώσουμε εάν μια λύση σε ένα LP είναι βέλτιστη ή όχι.

Πίνακας 7: Πρωτεύουσες-Δυικές δυνατότητες

		Δυαδικό LP		
		Βέλτιστο	Απραγματοποίητο	Απεριόριστο
Πρωτεύον LP	Βέλτιστο	Ναι	Όχι	Όχι
	Απραγματοποίητο	Όχι	Ναι	Ναι
	Απεριόριστο	Όχι	Ναι	Όχι

Τα ακόλουθα θεωρήματα περιγράφουν τις σχέσεις μεταξύ των πρωτευόντων και δυϊκών προβλημάτων. Για μια πληρέστερη συζήτηση αυτών των θεωρημάτων, αναφερθείτε στο (Chvatal 1983).

Θεώρημα 2.4.1. (Ασθενής δυϊκότητα στον Γραμμικό Προγραμματισμό).

Κάθε τιμή της πρωτεύουσας αντικειμενικής συνάρτησης παρέχει ένα κάτω όριο για κάθε τιμή της δυικής αντικειμενικής συνάρτησης.

Θεώρημα 2.4.2. (Ισχυρή δυϊκότητα στο Γραμμικό Προγραμματισμό).

Εάν ένα πρωτεύον πρόβλημα έχει μια βέλτιστη λύση, το δυϊκό του έχει επίσης μια βέλτιστη λύση και οι βέλτιστες τιμές των δύο προβλημάτων συμπίπτουν.

Θεώρημα 2.4.3. (Συμπληρωματική χαλαρότητα στο γραμμικό προγραμματισμό).

Στη βελτιστοποίηση, ή ο περιορισμός του πρωτεύοντος γραμμικού προβλήματος είναι ενεργός ή η αντίστοιχη δυική μεταβλητή είναι 0 ή και τα δύο. Επιπλέον, ή ο δυϊκός γραμμικός περιορισμός είναι ενεργός ή η αντίστοιχη πρωτεύουσα μεταβλητή είναι 0 ή και τα δύο.

Ορισμός 2.4.1. (Χάσμα δυϊκότητας).

Η διαφορά μεταξύ των πρωτευόντων και δυϊκών αντικειμενικών τιμών των πρωτευόντων και δυϊκών εφικτών λύσεων καλείται χάσμα δυαδικότητας (duality gap).

2.6 Δυϊκός Αλγόριθμος Simplex

Σε αυτή την ενότητα παρουσιάζουμε ένα από τα πιο σημαντικά ζητήματα του γραμμικού προγραμματισμού. Κατά την εκτέλεση του αλγορίθμου Simplex δεν βρίσκουμε μόνο τη βέλτιστη λύση ενός γραμμικού προβλήματος αλλά βρίσκουμε επίσης και ένα συνοδευτικό πρόβλημα που καλείται δυϊκό πρόβλημα. Η δυική μέθοδος Simplex αναπτύχθηκε από τον Lemke 1954 [6]. Στον γραμμικό προγραμματισμό χρησιμοποιούμε τη δυϊκότητα σε ένα μεγάλο πλήθος θεωρητικών και πρακτικών περιπτώσεων. Μεταξύ άλλων αυτές είναι οι παρακάτω:

- Σε μερικές περιπτώσεις είναι πιο εύκολο (π.χ. λιγότερες επαναλήψεις) να λύσουμε το δυϊκό πρόβλημα αντί του πρωτεύοντος.
- Οι δυικές μεταβλητές παρέχουν σημαντικές οικονομικές ερμηνείες όταν επιλύουμε ένα πρόβλημα γραμμικού προγραμματισμού.
- Η δυϊκότητα χρησιμοποιείται βοηθητικά όταν αλλάζουν οι τιμές των συντελεστών ενός προβλήματος γραμμικού προγραμματισμού.

- Η δυϊκότητα χρησιμοποιείται για να μας επιτρέψει να χρησιμοποιήσουμε τη μέθοδο Simplex για να επιλύσουμε προβλήματα στα οποία η αρχική βάση δεν είναι εφικτή (αυτή η τεχνική είναι γνωστή σαν δυϊκή Simplex).
- Η δυϊκότητα χρησιμοποιείται για να αναπτύξουμε έναν αριθμό από σημαντικά θεωρητικά αποτελέσματα στο γραμμικό προγραμματισμό.

Περιγράφουμε τη δυικη μέθοδο Simplex η οποία λύνει το δυϊκό πρόβλημα απευθείας πάνω στο Simplex ταμπλό του πρωτεύοντος. Εξετάζουμε το σύνηθες πρόβλημα του γραμμικού προγραμματισμού:

$$\max z = cx$$

$$\text{s.t. } Ax = c \qquad (2.4.1)$$

$$x \geq 0$$

Σε κάθε επανάληψη προχωράμε από μια βασική εφικτή λύση του δυϊκού προβλήματος σε μια βελτιωμένη βασική εφικτή λύση μέχρις ότου είτε φτάσουμε σε μια βέλτιστη λύση του δυϊκού (καθώς επίσης και του πρωτεύοντος) είτε συμπεράνουμε ότι το δυϊκό πρόβλημα είναι απεριόριστο και το πρωτεύων είναι μη εφικτό.

Περιγραφή του Δυϊκού Αλγορίθμου Simplex

Βήμα 0: (Αρχικοποίηση)

Ξεκίνα με ένα εφικτό βασικό σημείο και κατασκεύασε το αντίστοιχο Simplex ταμπλό.

Βήμα 1:(Επιλογή εισερχόμενης μεταβλητής)

Αν $b_i \geq 0$ για i=1, 2, ..., m, STOP. Η λύση είναι βέλτιστη. Διαφορετικά διάλεξε την εξερχόμενη μεταβλητή του δείκτη r έτσι ώστε:

$$b_r = \min\{ b_i : b_i < 0, \text{ για } i=1,2,...,m\}$$

Βήμα 2: (Επιλογή εξερχόμενης μεταβλητής)

Θέσε $J = \{ j : a_{rj} < 0\}$. Αν $J = \emptyset$, STOP. Το δυϊκό πρόβλημα είναι απεριόριστο. Διαφορετικά βρες τον δείκτη της στήλες περιστροφής s:

$$\frac{a_{0s}}{a_{rs}} = \min\{-\frac{a_{0j}}{a_{rj}} : j \in J \}$$

Βήμα 3: (Περιστροφή)

Από το επόμενο ταμπλό με τη μεταβλητή περιστροφής a_{rs} π.χ.

Θέσε $a_{rj} \leftarrow \frac{a_{rj}}{a_{rs}}$ όπου j=1, 2,... , n, n+1

$a_{ij} \leftarrow a_{ij} - \frac{a_{rj}}{a_{rs}} a_{is}$ όπου i= 0,1,2,... m ($i \neq r$) και j = 1,2,..., n, n+1

και πήγαινε στο βήμα 1.

Επιλύουμε 2 προβλήματα με τον Δυϊκό Πρωτεύοντα Αλγόριθμο Simplex. Το πρώτο πρόβλημα είναι βέλτιστο και το δεύτερο απεριόριστο.

Παράδειγμα 2.5.1:

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον δυϊκό πρωτεύοντα αλγόριθμο Simplex.

$$\max z = -2x_1 - x_2 - 4x_3$$

$$\text{s.t. } -2x_1 - 4x_2 - 2x_3 \leq -6$$

$$-x_1 + 2x_2 - 6x_3 \leq -5$$

$$-x_1 + 4x_2 + 3x_3 \leq 8$$

$$x_1, x_2, \text{και } x_3 \geq 0$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_4, x_5 και x_6, οπότε το γραμμικό πρόβλημα μετατρέπεται στο παρακάτω:

$$\max z = -2x_1 - x_2 - 4x_3$$

$$\text{s.t. } -2x_1 - 4x_2 - 2x_3 + x_4 = -6$$

$$-x_1 + 2x_2 - 6x_3 + x_5 = -5$$

$$-x_1 + 4x_2 + 3x_3 + x_6 = 8$$

$$x_1, x_2, x_3, x_4, x_5 \text{ και } x_6 \geq 0$$

Επανάληψη 1

Βήμα 0 (Αρχικοποίηση)

Από το αρχικό πρόβλημα είναι φανερό ότι τα δυικά βασικά εφικτά σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 0$, $x_4 = -6$, $x_5 = -5$ και $x_6 = 8$. Κατασκευάζουμε το αρχικό Simplex ταμπλό.

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	2	1	4	0	0	0	0
x_4	0	-2	-4	-2	1	0	0	-6
x_5	0	-1	2	-6	0	1	0	-5
x_6	0	-1	4	3	0	0	1	8

Βήμα 1: (επιλογή εξερχόμενης μεταβλητής)

Προσδιόρισε τη γραμμή περιστροφής και επέλεξε

$$b_r = \min \{ b_1 , b_2 \} = \min \{ -6 , -5 \} = -6 = b_1$$

Το r=1, συνεπώς η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα 2: (επιλογή εισερχόμενης μεταβλητής) θέσε J = {1, 2, 3} και βρες s = 2 διότι

$$\frac{a_{0s}}{a_{rs}} = \min\left\{-\frac{a_{0j}}{a_{rj}} : j \in J\right\} = \min\left\{-\frac{a_{01}}{a_{11}}, -\frac{a_{02}}{a_{12}}, -\frac{a_{03}}{a_{13}}\right\} = \min\left\{-\frac{2}{-2}, -\frac{1}{-4}, -\frac{4}{-2}\right\} = \frac{1}{4}$$

Συνεπώς η μεταβλητή x_2 μπαίνει στη βάση,

Βήμα 3: (Περιστροφή)

Σχημάτισε το επόμενο Simplex ταμπλό με το στοιχείο περιστροφής $a_{32}=a_{12}$ = -4 π.χ.

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	3/2	0	7/2	1/4	0	0	-3/2
x_4	0	1/2	1	1/2	-1/4	0	0	3/2
x_4	0	-4	0	-14	1	2	0	-16
x_5	0	-3	0	1	1	0	1	2

Επειδή υπάρχει $b_2 < 0$, στο δεξί μέρος, επαναλαμβάνουμε τον δυϊκό αλγόριθμο Simplex.

Επανάληψη 2

Βήμα 1: (επιλογή εξερχόμενης μεταβλητής)

Προσδιόρισε τη στήλη περιστροφής και επέλεξε

$$b_r = \min\{ b_2 \} = \min\{ -16 \} = -16 = b_2$$

Το r = 2, συνεπώς η μεταβλητή x_5 βγαίνει από τη βάση.

Βήμα 2: (επιλογή εισερχόμενης μεταβλητής) Θέσε J = {1, 3} και βρες s = 3 διότι

$$\frac{a_{0s}}{a_{rs}} = \min\left\{-\frac{a_{0j}}{a_{rj}} : j \in J\right\} = \min\left\{-\frac{a_{01}}{a_{21}}, -\frac{a_{03}}{a_{23}}\right\} = \min\left\{-\frac{6}{-4}, -\frac{14}{-14}\right\} = 1$$

Συνεπώς η μεταβλητή x_3 μπαίνει στη βάση.

Βήμα 3: (Περιστροφή)

Σχημάτισε το επόμενο Simplex ταμπλό με το στοιχείο περιστροφής

$a_{32}=a_{23}$ = -14 π.χ.

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	1/2	0	0	1/2	1/2	0	-11/2
x_2	0	5/14	1	1/20	-3/14	-1/14	0	13/14
x_3	0	2/7	0	1	-1/14	-1/7	0	8/7
x_6	0	-23/7	0	0	15/14	8/7	1	26/7

Επανάληψη 3

Βήμα 1: (επιλογή εξερχόμενης μεταβλητής)

Επειδή $b_i > 0$ όπου i = 1, 2, 3 STOP. Η λύση είναι η x_1 = 0, x_2 = 13/14, x_3 = 8/7, $x_4 = x_5$ = 0 και το z = -11/2.

Παράδειγμα 2.5.2:

Να λυθεί το παρακάτω γραμμικό πρόβλημα με τον δυϊκό πρωτεύοντα αλγόριθμο Simplex.

$$\max z = - x_1 + 2x_2 + 4x_3$$

$$\text{s.t. } x_1 + x_2 - 2x_3 \leq -4$$

$$2x_1 - x_2 - 2x_3 \leq -2$$

$$x_1 - x_2 + 2x_3 \leq -8$$

$$x_1, x_2, \text{ και } x_3 \geq 0$$

Λύση:

Προσθέτουμε τις χαλαρές μεταβλητές x_4, x_5 και x_6. Έτσι το γραμμικό πρόβλημα μετατρέπεται στο παρακάτω:

$$\max z = - x_1 + 2x_2 + 4x_3$$

$$\text{s.t. } x_1 + x_2 - 2x_3 + x_4 = -4$$

$$2x_1 - x_2 - 2x_3\ x_5 = -2$$

$$x_1 - x_2 + 2x_3 + x_6 = -8$$

$$x_1, x_2, x_3, x_4, x_5 \text{ και } x_6 \geq 0$$

Επανάληψη 1

Βήμα 0 (Αρχικοποίηση)

Από το αρχικό πρόβλημα είναι φανερό ότι τα δυικά βασικά εφικτά σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 0$, $x_4 = -4$, $x_5 = -2$ και $x_6 = -8$. Κατασκευάζουμε το αρχικό Simplex ταμπλό

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	1	-2	4	0	0	0	0
x_4	0	1	1	-2	1	0	0	-4
x_5	0	2	-1	-2	0	1	0	-2
x_6	0	1	-1	2	0	0	1	-8

Βήμα 1: (επιλογή εξερχόμενης μεταβλητής).

Προσδιόρισε τη γραμμή περιστροφής και επέλεξε

$$b_r = \min\{b_1, b_2, b_3\} = \min\{ -4, -2, -8 \} = -8 = b_3$$

Το r = 3, συνεπώς η μεταβλητή x_6 βγαίνει από τη βάση.

Βήμα 2: (επιλογή εισερχόμενης μεταβλητής)

Θέσε J = { 2 } και βρες s = 2 διότι:

$$\frac{a_{0s}}{a_{rs}} = \min\left\{-\frac{a_{0j}}{a_{rj}} : j \in J\right\} = \min\left\{-\frac{a_{02}}{a_{2132}}\right\} = \min\left\{-\frac{-2}{-1},\right\} = -2$$

Συνεπώς η μεταβλητή x_2 μπαίνει στη βάση.

Βήμα 3: (Περιστροφή)

Σχημάτισε το επόμενο Simplex ταμπλό με το στοιχείο περιστροφής $a_{rs}=a_{12}$ = -1 π.χ.

	z	x_1	x_2	x_3	x_4	x_5	x_6	Δεξί Μέρος
z	1	-1	0	-5	0	0	-2	16
x_4	0	2	0	0	1	0	0	-12
x_5	0	1	0	-4	0	1	0	6
x_6	0	1	1	-2	0	0	1	8

Επειδή υπάρχει $b_1 < 0$ στο δεξί μέρος επαναλαμβάνουμε τον δυϊκό αλγόριθμο Simplex

Επανάληψη 2

Βήμα 1: (επιλογή εξερχόμενης μεταβλητής)

Προσδιόρισε τη γραμμή περιστροφής και επέλεξε

$$b_r = \min\{ b_1 \} = \min\{-12\} = -12 = b_1$$

Το r = 1, συνεπώς η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα 2: (επιλογή εισερχόμενης μεταβλητής)

Θέσε J= ∅, STOP. Το δυϊκό πρόβλημα είναι απεριόριστο.

3. OPEN MP

3.1 Εισαγωγή

Στο κεφάλαιο αυτό κάνουμε μια σύνοψη του πρότυπου Open MP και παρουσιάζουμε ενδεικτικά σημεία που θα μας βοηθήσουν να κατανοήσουμε την εφαρμογή του.

3.2 Το Open MP

Το Open MP [7]είναι μια διεπαφή εφαρμογών προγραμμάτων (Application Program Interface - API) το οποίο χρησιμοποιείται για να παραλληλίσουμε προγράμματα σε αρχιτεκτονικές συστημάτων κοινής μνήμης. Αποτελείται από τρία συστατικά:

- οδηγίες για τον μεταφραστή,
- συναρτήσεις βιβλιοθηκών
- μεταβλητές περιβάλλοντος.

Το API του Open MP έχει οριστεί για τις γλώσσες προγραμματισμού C/C++ και Fortran, ενώ έχει υλοποιηθεί για τις πιο σημαντικές πλατφόρμες όπως το Unix και τα Windows NT, κάτι που το καθιστά αρκετά φορητό. Επιπλέον, το Open MP είναι κλιμακούμενο. Έχει οριστεί από κοινού, από τις πιο σημαντικές εταιρίες υλικού και λογισμικού και αναμένεται σε λίγα χρόνια να αποτελέσει ένα American National Standards Institute (ANSI) πρότυπο. Το όνομά του προέρχεται από τη φράση *Open specifications for Multi-Processing* που σημαίνει Ανοιχτές (εννοείται για όλες τις εταιρίες) προδιαγραφές για Πολύ-Επεξεργασία. Πρέπει να επισημάνουμε ότι το Open MP δεν προορίζεται για αρχιτεκτονικές συστημάτων κατανεμημένης μνήμης, ενώ δεν έχει υλοποιηθεί από όλες τις εταιρίες υλικού και λογισμικού. Επίσης, ο σχεδιασμός του είναι τέτοιος ώστε να μην αποδίδει τη μέγιστη αποδοτικότητα στη χρήση της κοινής μνήμης.

Η ιστορία του Open MP ξεκινάει από τις αρχές της δεκαετίας του ‘90. Εκείνη τη εποχή, οι εταιρίες συστημάτων κοινής μνήμης παρείχαν παρόμοιες επεκτάσεις προγραμμάτων Fortran, βασισμένες σε οδηγίες. Ο προγραμματιστής έδινε στον μεταφραστή ένα σειριακό πρόγραμμα Fortran με οδηγίες που καθόριζαν ποιοι βρόγχοι θα παραλληλοποιούνταν. Στη συνέχεια, ο μεταφραστής ήταν υπεύθυνος για την αυτόματη παραλληλοποίηση αυτών των βρόγχων στους παράλληλος επεξεργαστές (Symmetric Multiprocessors - SMP) [6]. Αν και οι υλοποιήσεις των διάφορων εταιριών ήταν παρόμοιες ως προς τη λειτουργικότητα, ωστόσο, διέφεραν αρκετά. Έτσι, έγινε μια προσπάθεια για να δημιουργηθεί ένα πρότυπο. Το πρώτο πρότυπο που δημιουργήθηκε ήταν το ANSI X3H5 το 1994, χωρίς όμως να υιοθετηθεί επειδή εκείνη την εποχή το ενδιαφέρον για αρχιτεκτονικές κοινής μνήμης έπεσε, ενώ ανέβηκε το ενδιαφέρον για συστήματα με αρχιτεκτονική κατανεμημένης μνήμης. Το 1997 προτάθηκε το πρότυπο Open MP συνεχίζοντας από εκεί που είχε μείνει το ANSI X3H5, μιας και πλέον τα συστήματα κοινής μνήμης άρχισαν να γίνονται δημοφιλέστερα. Εκείνη τη χρονιά είχε οριστεί μόνο το API για τη Fortran. Το 1998 ορίστηκε το API για τις γλώσσες C/C++, ενώ το 2000 φτιάχτηκε μια δεύτερη έκδοση για τη Fortran και το 2002 ακολούθησε η δεύτερη έκδοση για C/C++. Η τρίτη έκδοση κυκλοφόρησε το 2008 και το 2011 κυκλοφόρησε το revision 3.1. Η πιο πρόσφατη έκδοση είναι η 4.5, διαθέσιμη από τον Νοέμβριο του 2015. Οι εταιρίες που συμμετέχουν επισήμως στις προδιαγραφές του Open MP προτύπου είναι:

Εταιρίες Υλικού:

- Compaq / Digital

- Hewlett-Packard Company
- Intel Corporation
- International Business Machines (IBM)
- Kuck & Associates, Inc. (KAI)
- Silicon Graphics, Inc.
- Sun Microsystems, Inc.
- U.S. Department of Energy ASCI program

Εταιρίες Λογισμικού:

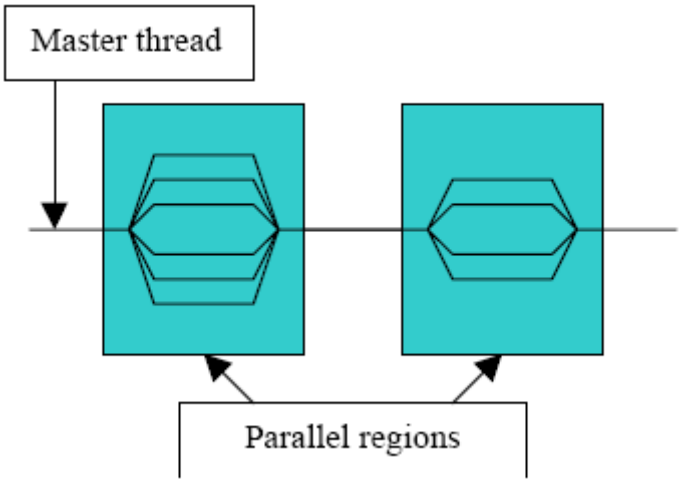
- Absoft Corporation
- Edinburgh Portable Compilers
- GENIAS Software GmBH
- Myrias Computer Technologies, Inc.
- The Portland Group, Inc. (PGI)

Οι στόχοι του Open MP είναι να παράσχει ένα πρότυπο που θα ισχύει για αρχιτεκτονικές και πλατφόρμες κοινής μνήμης. Ένα πρότυπο που θα είναι μικρό και εύκολα κατανοητό, θα παρέχει δηλαδή ένα απλό και περιορισμένο σύνολο από οδηγίες με το οποίο θα μπορεί να γίνεται σημαντική παραλληλοποίηση. Επίσης, θα πρέπει να είναι εύκολο στη χρήση του. Το Open MP παρέχει: α) τη δυνατότητα παραλληλοποίησης ενός προγράμματος σταδιακά (εν αντιθέσει, για παράδειγμα, με το Message Passing Interface (MPI) όπου κάθε φορά γίνεται include όλη η βιβλιοθήκη) και β) τη δυνατότητα τόσο χονδρού όσο και λεπτού κόκκου παραλληλίας. Τέλος, ο κυριότερος στόχος του Open MP είναι η φορητότητα.

3.3 Προγραμματιστικό Μοντέλο του Open MP

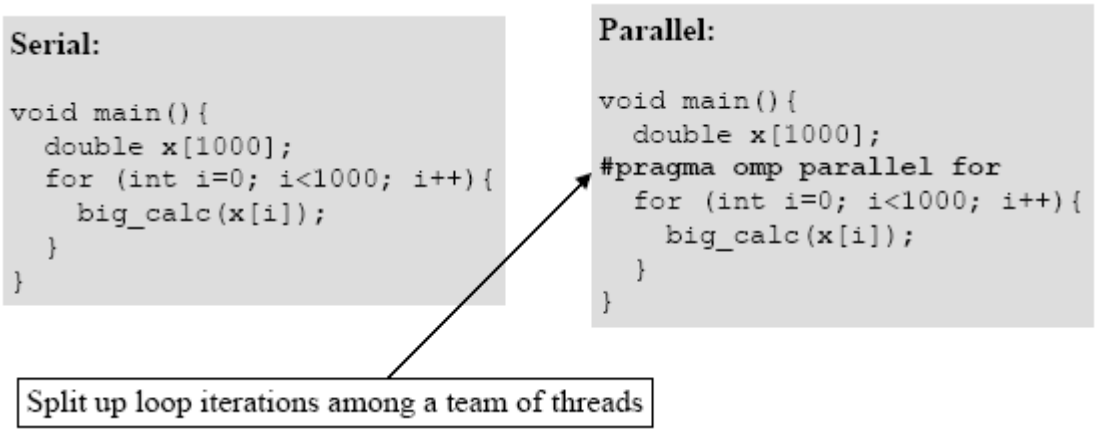
Το προγραμματιστικό μοντέλο του Open MP είναι βασισμένο σε παραλληλισμό με χρήση νημάτων. Μια διεργασία κοινής μνήμης αποτελείται από πολλά νήματα και το Open MP βασίζεται στο χαρακτηριστικό αυτό. Στο Open MP ο προγραμματιστής μπορεί να ελέγχει και να αυξάνει ή να μειώνει σταδιακά τον αριθμό των νημάτων. Η όλη διαδικασία δε, θυμίζει το μοντέλο fork-join που χρησιμοποιεί η C. Όπως φαίνεται στο Σχήμα 2, ένα *Master* νήμα γεννά *ομάδες* από νήματα όταν χρειάζεται. Το Master νήμα εκτελεί τις εντολές σειριακά μέχρι να φτάσει την πρώτη *παράλληλη περιοχή.* Η ομάδα των νημάτων που περιλαμβάνεται σε αυτή την παράλληλη περιοχή εκτελείται παράλληλα. Όταν η ομάδα τελειώσει, τότε τα νήματα συγχρονίζονται και τερματίζουν, αφήνοντας μόνο το Master νήμα. Η διαδικασία αυτή επαναλαμβάνεται όσες φορές χρειάζεται.

Τα νήματα επικοινωνούν μεταξύ τους με κοινές μεταβλητές. Ωστόσο, ο διαμοιρασμός των δεδομένων μπορεί να οδηγήσει σε συνθήκες ανταγωνισμού. Για να ελέγξουμε τις συνθήκες ανταγωνισμού, χρησιμοποιούμε *συγχρονισμό*. Ο συγχρονισμός, όμως, κοστίζει, με αποτέλεσμα κάθε φορά που παραλληλοποιούμε να πρέπει να οργανώνουμε τα δεδομένα κατά τέτοιο τρόπο ώστε να ελαχιστοποιούμε την ανάγκη για συγχρονισμό.



Σχήμα 2: Γενικό σχήμα παραλληλοποίησης στο Open MP

Για να γίνει αυτή η παραλληλοποίηση, ο προγραμματιστής θα πρέπει να δώσει στον μεταφραστή οδηγίες οι οποίες ενσωματώνονται στον κώδικα της C/C++ ή της Fortran. Στη συνέχεια, ο μεταφραστής μεταφράζει αυτές τις οδηγίες και παράγει κλήσεις βιβλιοθήκης έτσι ώστε να παραλληλοποιήσει τμήματα του κώδικα. Στο Σχήμα 3 φαίνεται ένα παράδειγμα παραλληλοποίησης κώδικα C με Open MP οδηγίες.



Σχήμα 3: Παράδειγμα παραλληλοποίησης ενός for βρόγχου

Ο αριθμός των νημάτων μπορεί να αλλάξει δυναμικά, μέσα από το ίδιο το πρόγραμμα, ή μέσω της μεταβλητής περιβάλλοντος *OMP_NUM_THREADS*. Για τον συγχρονισμό των νημάτων, όμως, καθώς και για τις εξαρτήσεις των δεδομένων υπεύθυνος είναι αποκλειστικά ο προγραμματιστής. Εξάλλου, το API του Open MP παρέχει την δυνατότητα τοποθέτησης παράλληλων τμημάτων μέσα σε άλλα παράλληλα τμήματα.

3.4 Οδηγίες Open MP για C/C++

Οι οδηγίες ακολουθούν τις συμβάσεις του προτύπου C/C++. Είναι case sensitive, ενώ θα πρέπει να υπάρχει ένα και μόνο ένα όνομα ανά οδηγία και κάθε οδηγία να εφαρμόζεται μόνο στο αμέσως επόμενο block εντολών το οποίο θα πρέπει να είναι ένα δομημένο block. Επίσης, μακρές οδηγίες μπορούν να συνεχιστούν σε επόμενες γραμμές αρκεί στο τέλος κάθε γραμμής να τοποθετείται ο χαρακτήρας της ανάποδης καθέτου (“\”).

3.4.1 Παράλληλες Περιοχές

Μια παράλληλη περιοχή είναι ένα τμήμα κώδικα το οποίο θα εκτελεστεί από πολλαπλά νήματα. Η σύνταξη μιας παράλληλης περιοχής είναι η εξής:

```
#pragma omp parallel                              \
        private (var1, var2, …)                   \
        shared (var1, var2, …)                    \
        firstprivate(var1, var2, …)               \
        copyin(var1, var2, …)                     \
        reduction(operator:var1, var2, …)         \
        if(expression)                            \
        default(shared|none)                      \
{

   …a structured block of code…

}
```

Εικόνα 1: Σύνταξη περιοχής Open MP

Όταν ένα νήμα φτάσει σε μια οδηγία παραλληλοποίησης (#pragma…), δημιουργεί μια ομάδα από νήματα και το ίδιο γίνεται Master της ομάδας. Το Master νήμα είναι και το ίδιο μέλος της ομάδας. Ο κώδικας της παράλληλης περιοχής αντιγράφεται τόσες φορές όσες είναι και το πλήθος των νημάτων και δίνεται σε αυτά για να τον εκτελέσουν. Γενικά, δεν υπάρχει συγχρονισμός μεταξύ των νημάτων. Κάθε νήμα μπορεί να φτάσει σε οποιοδήποτε σημείο μέσα στη παράλληλη περιοχή, σε ακαθόριστη χρονική στιγμή. Στο τέλος της παράλληλης περιοχής υπονοείται ένα φράγμα, πέρα από το οποίο μόνο το Master νήμα θα συνεχίσει την εκτέλεση του προγράμματος. Εξάλλου, το API παρέχει τη δυνατότητα δημιουργίας παράλληλης περιοχής μέσα σε μια άλλη παράλληλη περιοχή. Η φωλιασμένη αυτή παράλληλη περιοχή, καταλήγει στη δημιουργία μιας καινούριας ομάδας από νήματα η οποία αποτελείται από ένα νήμα, by default. Ωστόσο, διάφορες υλοποιήσεις μπορούν να επιτρέψουν παραπάνω από ένα νήμα σε μια φωλιασμένη παράλληλη περιοχή.

Ο αριθμός των νημάτων που δημιουργούνται κατά την είσοδο σε μια παράλληλη περιοχή μπορεί να καθοριστεί από τρεις παράγοντες, οι οποίοι κατά σειρά προτεραιότητας είναι:

α) η χρήση της συνάρτησης βιβλιοθήκης *omp_set_num_threads()*,

β) θέτοντας τη μεταβλητή περιβάλλοντος OMP_NUM_THREADS και

γ) χρησιμοποιώντας την default υλοποίηση.

Επίσης, δίνεται η δυνατότητα από το API, να προσαρμόζεται δυναμικά ο αριθμός των νημάτων για μία συγκεκριμένη παράλληλη περιοχή. Αυτό μπορεί να επιτευχθεί με τους εξής δύο τρόπους:

α) με τη χρήση της συνάρτησης βιβλιοθήκης *omp_set_dynamic()* και

β) θέτοντας τη μεταβλητή περιβάλλοντος OMP_DYNAMIC.

Η αρίθμηση των νημάτων ξεκινάει από τον αριθμό 0, ο οποίος δίνεται στο Master νήμα της κάθε ομάδας και φτάνει στον αριθμό N-1 όπου N είναι ο αριθμός των νημάτων. Η ταυτότητα κάθε νήματος μπορεί να βρεθεί χρησιμοποιώντας τη συνάρτηση βιβλιοθήκης *omp_get_thread_num()*, ενώ ο συνολικός αριθμός των νημάτων μπορεί να βρεθεί με τη χρήση της συνάρτησης βιβλιοθήκης *omp_get_num_threads()*. Αν και τα νήματα εκτελούν το ίδιο κομμάτι κώδικα, ωστόσο, θα θέλαμε να εκτελέσουν διαφορετικά μονοπάτια αυτού του κώδικα. Αυτό επιτυγχάνεται με την ανάθεση διαφορετικού κομματιού κώδικα σε κάθε νήμα (για παράδειγμα, με τη χρήση *if-else*).

Παρακάτω παραθέτουμε τον κώδικα του *Hello World* στον οποίο αναθέτουμε διαφορετικές εργασίες στο Master νήμα και στα υπόλοιπα νήματα. Στον κώδικα αυτόν, όλα τα νήματα τυπώνουν το *Hello World* και τον αριθμό τους, ενώ το Master νήμα τυπώνει το συνολικό αριθμό των νημάτων.

```
#include <omp.h>
int main ()  {
	int nthreads, tid;
	/* Fork a team of threads giving them their own copies of variables */
	#pragma omp parallel private(tid)
	{
		/*Obtain and print thread id */
		tid = omp_get_thread_num();
		printf("Hello World from thread = %d\n", tid);
		/* Only master thread does this */
		if (tid == 0)
		{
			nthreads = omp_get_num_threads();
			printf("Number of threads = %d\n", nthreads);
		}
	}
	/* All threads join master thread and terminate */
}
```

Σε αυτό το σημείο θα πρέπει να αναφέρουμε μερικούς περιορισμούς που πρέπει να ισχύουν όταν μιλάμε για παράλληλες περιοχές. Πρώτον, μια παράλληλη περιοχή θα πρέπει να είναι ένα δομημένο block, το οποίο δεν εκτείνεται σε πολλαπλές ρουτίνες ή σε αρχεία με κώδικα. Δεύτερον, όλες οι διακλαδώσεις που ξεκινούν μέσα σε μια παράλληλη περιοχή, θα πρέπει να καταλήγουν στην ίδια παράλληλη περιοχή. Τρίτον, μόνο μια συνθήκη *if-else* επιτρέπεται.

3.4.2 Συνθήκες Οδηγιών του Open MP

Οι συνθήκες οδηγιών που χρησιμοποιούνται στο Open MP είναι οι εξής:

shared(var1,var2,…): Καθολικές μεταβλητές, (*var1,var2,…*), οι οποίες θα προσπελαθούν από όλα τα νήματα (τα νήματα προσπελαύνουν ίδιες θέσεις μνήμης).

private(var1,var2,…): Κάθε νήμα έχει το δικό του αντίγραφο από αυτές τις ιδιωτικές μεταβλητές, (*var1,var2,…*), για τη διάρκεια της εκτέλεσης της παράλληλης περιοχής.

if(expression): Η παραλληλοποίηση γίνεται μόνο όταν το *expression* είναι αληθές.

default(shared|private|none): Καθορίζει την εμβέλεια των μεταβλητών στην παράλληλη περιοχή (αν θα είναι κοινές, ιδιωτικές, ή αν δεν θα έχουν εμβέλεια αντιστοίχως).

schedule(type [,chunk]): Ελέγχει τον τρόπο με τον οποίο οι επαναλήψεις ενός βρόγχου θα διαμοιραστούν στα νήματα. Το *type* του *schedule* μπορεί να είναι:

STATIC: Οι επαναλήψεις ενός βρόγχου διαμοιράζονται σε κομμάτια μεγέθους *chunk* και ανατίθεται στατικά στα νήματα. Αν το *chunk* δεν έχει καθοριστεί, τότε οι επαναλήψεις διαμοιράζονται ισομερώς (αν αυτό είναι δυνατό) και συνεχόμενα μεταξύ των νημάτων.

DYNAMIC: Οι επαναλήψεις ενός βρόγχου διαμοιράζονται σε κομμάτια μεγέθους *chunk* και ανατίθεται δυναμικά στα νήματα. Όταν ένα νήμα τελειώσει με ένα κομμάτι αναλαμβάνει δυναμικά ένα άλλο. Το προεπιλεγμένο μέγεθος του κομματιού (chunk) είναι 1.

GUIDED: Οι επαναλήψεις ανατίθενται δυναμικά στα νήματα σε μπλοκ, καθώς τα νήματα τα αιτούνται, μέχρι να μην υπάρχει μπλοκ για ανάθεση. Παρόμοιο με το DYNAMIC με την διαφορά ότι το μέγεθος του μπλοκ μειώνεται κάθε φορά που ένα πακέτο εργασίας ανατίθεται σε ένα νήμα. Το μέγεθος του αρχικού μπλοκ είναι ανάλογο του: number_of_iterations / number_of_threads. Τα επόμενα μπλοκ είναι ανάλογα του: number_of_iterations_remaining / number_of_threads. Η παράμετρος chunk ορίζει το ελάχιστο μέγεθος του μπλοκ. Η προεπιλεγμένη τιμή είναι 1.

RUNTIME: Η απόφαση διαμοιρασμού, αναβάλλεται μέχρι την εκτέλεση του προγράμματος με τη βοήθεια της μεταβλητής περιβάλλοντος *OMP_SCHEDULE*. Απαγορεύεται να καθοριστεί μέγεθος *chunk* για αυτή τη συνθήκη.

3.5 Περιοχές Διαμοιρασμού Εργασίας

Μια περιοχή διαμοιρασμού εργασίας διαμοιράζει τις εργασίες που περικλείονται στην παράλληλη περιοχή μεταξύ των νημάτων της ομάδας της περιοχής. Οι περιοχές διαμοιρασμού εργασίας δεν δημιουργούν καινούρια νήματα και δεν υπάρχει κάποιος συγχρονισμός κατά την είσοδό τους σε μια τέτοια περιοχή, υπάρχει όμως συγχρονισμός κατά την έξοδο. Όπως και στις παράλληλες περιοχές, έτσι και στις περιοχές διαμοιρασμού εργασίας, υπάρχουν κάποιοι περιορισμοί που πρέπει να ισχύουν.

Πρώτον, για να εκτελεσθεί παράλληλα μια οδηγία, θα πρέπει η περιοχή διαμοιρασμού εργασίας να εσωκλείεται δυναμικά μέσα σε μια παράλληλη περιοχή. Δεύτερον, οι περιοχές διαμοιρασμού εργασίας θα πρέπει να διαμοιράζουν τα δεδομένα (ή την εργασία) σε όλα τα νήματα ή σε κανένα. Τρίτον, διαδοχικές περιοχές διαμοιρασμού εργασίας θα πρέπει να προσπελαύνονται από τα μέλη μιας ομάδας με την ίδια σειρά.

3.5.1 Οδηγία for

Η οδηγία *for* καθορίζει ότι οι επαναλήψεις του βρόγχου που ακολουθεί πρέπει να εκτελεσθούν παράλληλα από την ομάδα. Αυτό προϋποθέτει ότι η παράλληλη περιοχή έχει ήδη αρχικοποιηθεί, αλλιώς εκτελεί σειριακά τον κώδικα σε έναν επεξεργαστή. Η σύνταξη μίας οδηγίας *for* φαίνεται παρακάτω:

```
#pragma omp for [clause [clause]….]
        for loop
```

όπου κάθε *clause* είναι μία από τις παρακάτω συνθήκες:

private(list)

firstprivate(list)

lastprivate(list)

reduction(operator: list)

ordered

schedule(type [,chunk])

nowait

Η συνθήκη *ordered* πρέπει να υπάρχει όταν διατεταγμένες οδηγίες συμπεριλαμβάνονται μέσα σε μια *for* οδηγία. Η συνθήκη *nowait* χρησιμοποιείται για να επιτρέψουμε στα νήματα να συνεχίσουν την εκτέλεσή τους χωρίς να περιμένουν τον τερματισμό και των υπόλοιπων νημάτων στο τέλος του παράλληλου βρόγχου.

Οι περιορισμοί που θα πρέπει να ισχύουν είναι ότι, πρώτον, η ορθότητα ενός προγράμματος δεν θα πρέπει να εξαρτάται από το ποιο νήμα θα εκτελέσει μια συγκεκριμένη επανάληψη. Δεύτερον, απαγορεύεται μια διακλάδωση να καταλήγει έξω από ένα βρόγχο ο οποίος σχετίζεται με μια οδηγία *for.* Τρίτον, το μέγεθος ενός κομματιού πρέπει να καθορίζεται σαν σταθερή έκφραση ακεραίου ενός βρόγχου, αφού δεν υπάρχει συγχρονισμός για την αποτίμηση του μεγέθους από τα νήματα. Τέταρτον, ο βρόγχος *for* θα πρέπει να είναι κανονικό σχήμα, και τέλος, οι συνθήκες *ORDERED* και *SCHEDULE* πρέπει να εμφανίζονται μια φορά η κάθε μία.

3.6 Συνδυασμένες Παράλληλες Περιοχές Διαμοιρασμού Εργασίας

Οι *Συνδυασμένες Παράλληλες Περιοχές Διαμοιρασμού Εργασίας* είναι συντομεύσεις για να καθορίσουμε μια παράλληλη περιοχή η οποία περιέχει μόνο μία περιοχή διαμοιρασμού εργασίας (ένα παράλληλο *for* ή παράλληλα *τμήματα*). Σημασιολογικά, οι συνδυασμένες παράλληλες περιοχές διαμοιρασμού εργασίας, είναι ισοδύναμες, με τη δήλωση μιας παράλληλης περιοχής, αμέσως ακολουθούμενης από μία περιοχή διαμοιρασμού εργασίας. Επιτρέπονται όλες οι υπαρκτές συνθήκες για μια παράλληλη περιοχή και για την σχετική περιοχή διαμοιρασμού εργασίας, εκτός από τη *nowait,* αφού, έτσι κι αλλιώς, στο τέλος κάθε παράλληλου τμήματος υπονοείται ένα φράγμα για το συγχρονισμό των νημάτων.

3.6.1 Οδηγία parallel for

Η *parallel for* οδηγία καθορίζει μια παράλληλη περιοχή που περιέχει μια απλή *for* οδηγία. Αυτή η *for* οδηγία, θα πρέπει να είναι η αμέσως επόμενη εντολή (μετά τη δήλωση της παράλληλης περιοχής) του κώδικα. Η σύνταξη μιας *parallel for* οδηγίας φαίνεται παρακάτω:

```
#pragma omp parallel for [clause [clause]….]
        for loop
```

όπου κάθε *clause* είναι μία από τις παρακάτω συνθήκες:

private(list)

firstprivate(list)

lastprivate(list)

reduction(operator: list)

schedule(type [,chunk])

if(expression)

default(shared|private|none)

copyin(list)

3.7 Οδηγίες Συγχρονισμού για C/C++

Οι *Οδηγίες Συγχρονισμού* χρησιμοποιούνται για να ελέγξουμε τη σειρά εκτέλεσης των νημάτων μιας ομάδας, έτσι ώστε να εξασφαλίσουμε την ακεραιότητα των δεδομένων. Οι οδηγίες συγχρονισμού που μας δίνει το API είναι οι εξής:

master οδηγία

critical οδηγία

barrier οδηγία

atomic οδηγία

flush οδηγία

ordered οδηγία

3.8 Κανόνες Φωλιάσματος

Όταν παράλληλες περιοχές είναι φωλιασμένες μέσα σε άλλες παράλληλες περιοχές, η συμπεριφορά χαρακτηρίζεται από κάποιους κανόνες που είναι οι εξής:

- Μια παράλληλη περιοχή μέσα σε μια άλλη, λογικά ορίζει μια νέα ομάδα νημάτων η οποία αποτελείται από το τρέχον νήμα.
- Οι οδηγίες *for, sections* και *single* που βρίσκονται μέσα στην ίδια παράλληλη περιοχή δεν επιτρέπεται να φωλιάσουν μεταξύ τους.
- Οι οδηγίες *for, sections* και *single* δεν επιτρέπονται μέσα στις οδηγίες (περιοχές) *critical, ordered, master*.
- Οδηγίες *critical* μα το ίδιο όνομα δεν επιτρέπεται να φωλιάσουν η μία μέσα στην άλλη.
- Οδηγίες *barrier* δεν επιτρέπονται μέσα στις οδηγίες (περιοχές) *critical, ordered, master, for, sections, single*.
- Οδηγίες *master* δεν επιτρέπονται μέσα στις οδηγίες (περιοχές) *for, sections, single*.
- Οδηγίες *ordered* δεν επιτρέπονται μέσα σε οδηγίες (περιοχές) *critical.*
- Κάθε οδηγία η οποία επιτρέπεται όταν εκτελείται δυναμικά μέσα σε μια παράλληλη περιοχή, επιτρέπεται κι όταν εκτελείται έξω από μια παράλληλη περιοχή. Όταν εκτελείται έξω από μια παράλληλη περιοχή, τότε η οδηγία εκτελείται σε μια ομάδα νημάτων αποτελούμενη μόνο από το master νήμα.

3.9 Runtime Συναρτήσεις Βιβλιοθήκης

Το Open MP παρέχει κάποιες συναρτήσεις βιβλιοθήκης (runtime) οι οποίες παρέχουν διάφορες λειτουργίες όπως τη παροχή του αριθμού των τρεχόντων νημάτων, το κλείδωμα κρίσιμων περιοχών, τη δυναμική προσαρμογή των νημάτων.

Οι runtime συναρτήσεις βιβλιοθήκης που χρησιμοποιήθηκαν:

1. void omp_set_num_threads (int num_threads)

Η συνάρτηση αυτή θέτει τον αριθμό των νημάτων που θα χρησιμοποιηθούν στην επόμενη παράλληλη περιοχή. Παίρνει σαν όρισμα έναν ακέραιο αριθμό ο οποίος εκφράζει τον αριθμό των νημάτων. Ο δυναμικός μηχανισμός των νημάτων μπορεί να τροποποιήσει το αποτέλεσμα της συνάρτησης αυτής. Έτσι, αν ο μηχανισμός αυτός είναι *"Enabled"*, τότε η συνάρτηση καθορίζει το μέγιστο αριθμό νημάτων που μπορεί να χρησιμοποιηθεί από κάθε περιοχή. Αν ο μηχανισμός είναι *"Disabled"*, τότε η συνάρτηση καθορίζει τον ακριβή αριθμό νημάτων προς χρήση, μέχρι την επόμενη κλήση της συνάρτησης. Η συνάρτηση αυτή, μπορεί να κληθεί μόνο από κάποιο σειριακό κομμάτι κώδικα και έχει προτεραιότητα έναντι της μεταβλητής περιβάλλοντος *OMP_NUM_THREADS.*

2. void omp_get_thread_num(void)

Επιστρέφει τον αριθμό εκείνου του νήματος, μέσα σε μια ομάδα, το οποίο έκανε την κλήση της συνάρτησης. Η αρίθμηση των νημάτων ξεκινάει από το 0 (που το έχει το master νήμα) εώς το *OMP_NUM_THREADS* – 1. Η συνάρτηση θα επιστρέψει 0 αν κληθεί μέσα από μια φωλιασμένη παράλληλη περιοχή ή μέσα από ένα σειριακό κομμάτι κώδικα.

3. void omp_set_dynamic(int dynamic_threads)

Επιτρέπει ή απαγορεύει την δυναμική ρύθμιση του αριθμού των νημάτων (από το runtime σύστημα) που είναι διαθέσιμα για εκτέλεση από τις παράλληλες περιοχές. Παίρνει σαν όρισμα έναν ακέραιο αριθμό. Αν ο αριθμός είναι διάφορος του 0, τότε ο μηχανισμός δυναμικής ρύθμισης του αριθμού των νημάτων, από το runtime σύστημα, ενεργοποιείται, αλλιώς, αν είναι 0, απενεργοποιείται. Η default κατάσταση του μηχανισμού αυτού εξαρτάται από την υλοποίηση. Η συνάρτηση μπορεί να κληθεί από κάποιο σειριακό κομμάτι του κώδικα και έχει προτεραιότητα έναντι της μεταβλητής περιβάλλοντος *OMP_DYNAMIC*.

4. ΕΦΑΡΜΟΓΗ ΤΟΥ OPEN MP ΣΤΟ SIMPLEX

4.1 Εισαγωγή

Στο κεφάλαιο αυτό εξετάζουμε πως το API του Open MP μπορεί να χρησιμοποιηθεί στην Simplex [8] [9].

4.2 Τα σημεία εν δυνάμη παραλληλίας

```
Αλγόριθμος Simplex
    Διάβασε Πίνακα
    Δημιούργησε το Tableau
    Εύρεση Σημείου Εκκίνησης
    Όσο το πρόβλημα δεν είναι αδύνατο ή αόριστο και η λύση μη βέλτιστη
        Επιλογή Εισερχόμενης Μεταβλητής σύμφωνα με κανόνα περιστροφής
        Επιλογή Εξερχόμενης Μεταβλητής σύμφωνα με κανόνα περιστροφής
        Ενημέρωσε Tableau
    Τέλος επανάληψης
    Εκτύπωσε Λύση
Τέλος Αλγόριθμος Simplex
```

Εικόνα 2: Αλγόριθμος Simplex σε Ψευδογλώσσα

Κοιτάζοντας την δομή του σε ψευδογλώσσα είναι προφανές ότι το μεγαλύτερο υπολογιστικό κόστος στον αλγόριθμο αυτόν αποτελεί ο υπολογισμός της ανανεωμένης βάσης που προκύπτει από την περιστροφή. [10] Το πλήθος των αριθμητικών πράξεων είναι εστιασμένο στον συγκεκριμένο υπολογισμό ενώ ο εντοπισμός της εισερχόμενης και της εξερχόμενης μεταβλητής γίνεται μέσω συγκρίσεων, των οποίων η παραλληλοποίηση έχει μεγάλο κόστος επικοινωνίας. Στην ανανέωση της βάσης, ο υπολογισμός των στοιχείων είναι μεταξύ τους ανεξάρτητος, με αποτέλεσμα να ευνοείται η παραλληλοποίηση της. Η βάση διασπάται στους διαθέσιμους πόρους με σκοπό την ενημέρωση των στοιχείων. [11]

Η ανανέωση της βάσης Tableau πρόκειται ουσιαστικά για μια τροποποιημένη απαλοιφή Gauss [12] [13] [14] [15], η οποία αποτελείται από 2 τμήματα, την κανονικοποίηση και την απαλοιφή γραμμών. Στη φάση της κανονικοποίησης διαιρούνται τα στοιχεία της γραμμής οδηγού με το μέγιστο της γραμμής. Η εύρεση μεγίστου/ελαχίστου μιας γραμμής ενός πίνακα είναι πράξη που προτιμάται η σειριακή εκτέλεσή της παρά της παράλληλης, λόγω του μεγάλου κόστους επικοινωνίας. Αντίθετα, η πράξη της διαίρεσης με το μέγιστο/ελάχιστο μπορεί να γίνει παράλληλα καθώς τα στοιχεία δεν είναι προσβάσιμα μέχρι την έναρξη της φάσης απαλοιφής γραμμών. Το κέρδος της παραλληλοποίησης δεν είναι ιδιαίτερο λόγω του εύρους της μίας γραμμής αλλά είναι υπαρκτό. [16]

Στην φάση απαλοιφής γραμμών, ωστόσο, η παραλληλοποίηση παίζει καταλυτικό ρόλο. Πρόκειται για έναν στοιχειώδη γραμμικό μετασχηματισμό των γραμμών όλου του πίνακα.

Όσο περισσότερο τμήμα των υπολογισμών εκτελείται παράλληλα τόσο πιο πολύ μειώνεται ο χρόνος εκτέλεσης. [6] Το γεγονός αυτό επιβεβαιώθηκε στις μετρήσεις από την υπολογιστική μελέτη καθώς η διαδικασία ανανέωσης της βάσης καταλαμβάνει σημαντικό μερίδιο από το συνολικό χρόνο εκτέλεσης.

4.3 Εφαρμογή Παραλληλίας στην Υλοποίηση Lalami

Η εφαρμογή των προσεγγίσεων αυτών στην ενημέρωση της γραμμής οδηγού μπορεί στο θεωρητικό επίπεδο να φαίνεται ιδανική αλλά κατά την εκτέλεση ενδέχεται να παρουσιάζει προβλήματα. Η διαδικασία αυτή πρόκειται ουσιαστικά για την προσπέλαση όλων των στηλών σε μια σταθερή γραμμή, πράγμα που σημαίνει ότι δεν προσπελαύνονται συγκεκριμένες συνεχόμενες περιοχές της μνήμης. Οι προσπελάσεις αντιθέτως γίνονται κάθε m κελιά. Το γεγονός αυτό αναιρεί εντελώς τον χαρακτήρα της τοπικότητας που πρέπει να παρουσιάζεται για να υπάρχει άμεση πρόσβαση στην μνήμη και άρα επιτάχυνση. Κατά συνέπεια, τόσο ο χειροκίνητος όσο και ο αυτόματος διαμοιρασμός δεν παρουσιάζουν τα αναμενόμενα οφέλη καθότι δεν μπορούν να εκμεταλλευτούν τις μικρές γρήγορες μνήμες (caches) που προσφέρουν οι σύγχρονοι επεξεργαστές.

```
#pragma omp parallel for shared(T) private(i,w)
for ( j = 0; j < m + 1; ++j) {
        if (j != index_leaving){
                w = T[j][index_entering];
                for ( i = 0; i < n + 1; ++i) {
                        T[j][i] -= w*T[index_leaving][i];
                }
        }
}
```

Εικόνα 3: Ενημέρωση πίνακα βάσης Simplex με χρήση Open MP (αυτόματος διαμοιρασμός)

Παρομοίως, η ενημέρωση των υπολοίπων οδηγών στοιχείων πρέπει να προσαρμοστεί στα δεδομένα της τοπικότητας. Τα νήματα έχουν πρόσβαση σε -πολλές φορές μεγαλύτερου μεγέθους- δεσμευμένη μνήμη, η οποία πρέπει να διαμοιραστεί ανάμεσα στα υπόλοιπα νήματα και να εκμεταλλευτεί καταλλήλως, χωρίς να υποπίπτει στο αδιέξοδο ανώφελου κορεσμού των κοινών πόρων. Τόσο η αυτόματη όσο και η -με συγκεκριμένο schedule- διάσπαση της διαδικασίας ενημέρωσης της βάσης αντιμετωπίζουν το προαναφερθέν πρόβλημα καθώς ενημερώνουν την ίδια γραμμή στην μνήμη cache. Η διαρκής ενημέρωση μεμονωμένων στοιχείων έχει ως αποτέλεσμα την καθυστέρηση, αφού απαιτείται η διαρκής ανανέωση των μνημών cache χωρίς την πλήρη εκμετάλλευση όλων των κελιών της. Το κόστος της φόρτωσης και εκφόρτωσης των μνημών ακόμα και με την κατά συνθήκες ίδια γραμμή στοιχείων μειώνει την αποδοτικότητα της λύσης αυτής.

Μολαταύτα, αν εφαρμοστεί ο χειροκίνητος διαμοιρασμός της περιοχής που αναλαμβάνει το κάθε νήμα, η συνεκτικότητα των μνημών cache βρίσκεται σε υψηλά επίπεδα, με αποτέλεσμα την άμεση εμφάνιση επιτάχυνσης από την διάσπαση σε πολλαπλά νήματα. Το κάθε νήμα ενημερώνει το δικό του τμήμα συνεχόμενης μνήμης, ελαχιστοποιώντας τις διαδικασίες φορτοεκφόρτωσης. Η επιτάχυνση στην ενημέρωση του πίνακα εξαρτάται πλέον από τα διαθέσιμα νήματα και τις διαθέσιμες κρυφές μνήμες cache. Τα περισσότερα νήματα εξασφαλίζουν καλύτερη εκμετάλλευση των σύγχρονων πολυπύρηνων επεξεργαστών, ενώ η ανάλογη ποσότητα μνήμης cache την μείωση των συχνών συναλλαγών με την αρκετά αργότερη κύρια μνήμη. Χρήσιμο, τέλος, είναι να υπάρχει ισορροπία μεταξύ των νημάτων καθώς τα πολλά νήματα με λίγη μνήμη οδηγούνται σε λιμοκτονία.

Στα παρακάτω τμήματα κώδικα φαίνεται καθαρά το κομμάτι ενημέρωσης της βάσης. Η αρχή γίνεται με το όρισμα του παράλληλου τμήματος κώδικα, καθώς και το είδος των μεταβλητών που θα χρησιμοποιηθούν, αν θα είναι ιδιωτικές σε κάθε νήμα ή κοινές. Το κάθε νήμα λοιπόν έχει την δικιά του ταυτότητα, δικούς του μετρητές και δείκτες για το εύρος που θα κυμανθεί και τέλος το οδηγό στοιχείο κάθε γραμμής. Η περιοχή της μνήμης που περιέχει τον πίνακα είναι προσπελάσιμη από όλα τα νήματα. Το κάθε νήμα με βάση το νούμερο του, αναλαμβάνει το αντίστοιχο κομμάτι του πίνακα, το οποίο και ενημερώνει. Η μοιρασιά γίνεται όσο πιο δίκαια μπορεί μέσω της διαίρεσης του πίνακα αλλά σε περίπτωση που δεν είναι τέλεια η διαίρεση με το μέγεθος του πίνακα, το τελευταίο νήμα ενδέχεται να έχει μικρότερο φόρτο εργασίας.

```
#pragma omp parallel private (i,j,tid,chunk,mystart,myend,w) shared(T)
{
	tid = omp_get_thread_num();
	chunk = m / nProcessors;
	mystart = tid*chunk;
	myend = min(mystart + chunk, m+1);
	//printf("\n%d %d %d\n", tid, mystart, myend);
	for (i = mystart; i < myend; i++){
		if (i != index_leaving){
			w = T[i][index_entering];
				for (j = 0; j < n + 1; j++) {
					T[i][j] -= w*T[index_leaving][j];
				}
		}
	}
}
```

Εικόνα 4: Ενημέρωση πίνακα βάσης Simplex με χρήση Open MP (χειροκίνητος διαμοιρασμός)

5. ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ

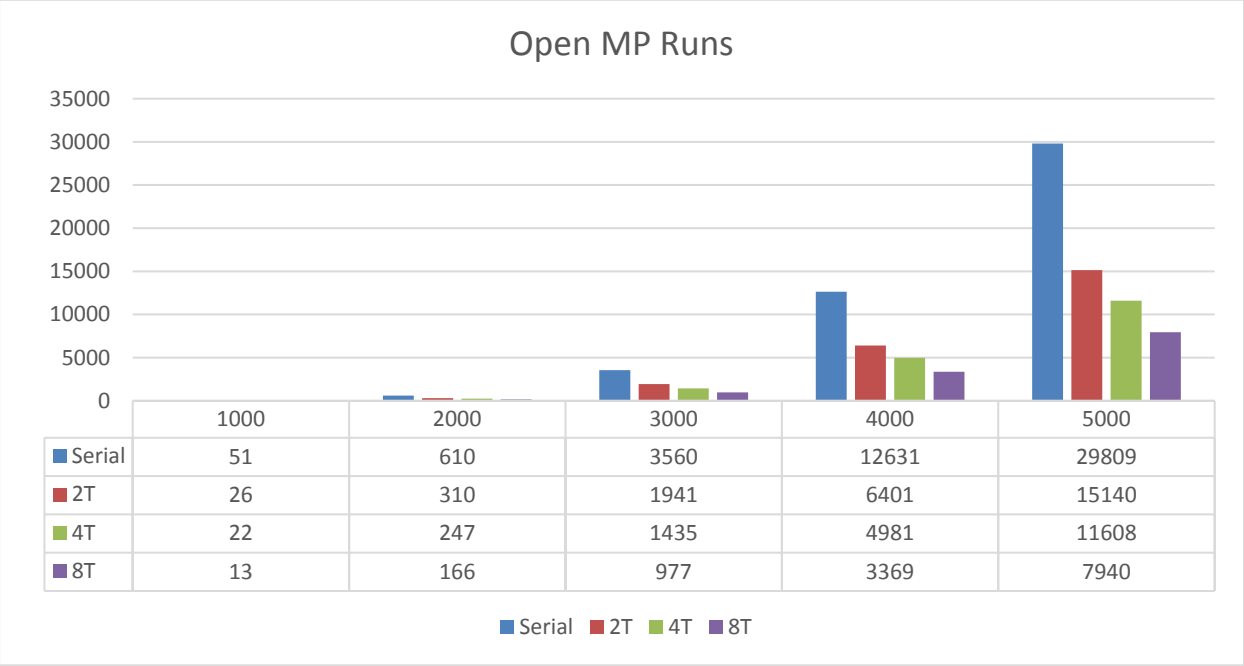
5.1 Εισαγωγή

Για την υπολογιστική σύγκριση αλγορίθμων απαραίτητη προϋπόθεση αποτελεί η υλοποίηση του αλγόριθμου σε κάποια γλώσσα προγραμματισμού και η εκτέλεση του για πολλαπλά μεγέθη. Αρχικά μελετήθηκε η σειριακή υλοποίηση του αλγόριθμου από τον Mohammed Esseghir Lalami ενώ δημιουργήθηκε η αντίστοιχη Open MP τροποποιώντας την σειριακή.

Ο αλγόριθμος υλοποιήθηκε σε γλώσσα προγραμματισμού C++ στο περιβάλλον του Visual Studio 2015 με το πρόσθετο Intel Parallel Studio 2017. Η υπολογιστική μελέτη διενεργήθηκε σε έναν Intel Core i7 6700K επεξεργαστή, χρονισμένο στα 4GHz, 4 πυρήνων/8 νημάτων, 16GB κύριας μνήμης RAM. Το λειτουργικό σύστημα που χρησιμοποιήθηκε ήταν τα Windows 7 Professional 64bit. Όλοι οι χρόνοι επεξεργασίας υπολογίστηκαν σε δευτερόλεπτα (sec).

5.2 Υλοποίηση Lalami

Οι μετρήσεις του Open MP προέκυψαν τρέχοντας 5 φορές για σταθερή φύτρα το κάθε πρόβλημα. Χρησιμοποιήθηκαν τόσο διάφορο πλήθος νημάτων όσο και μέγεθος προβλήματος.

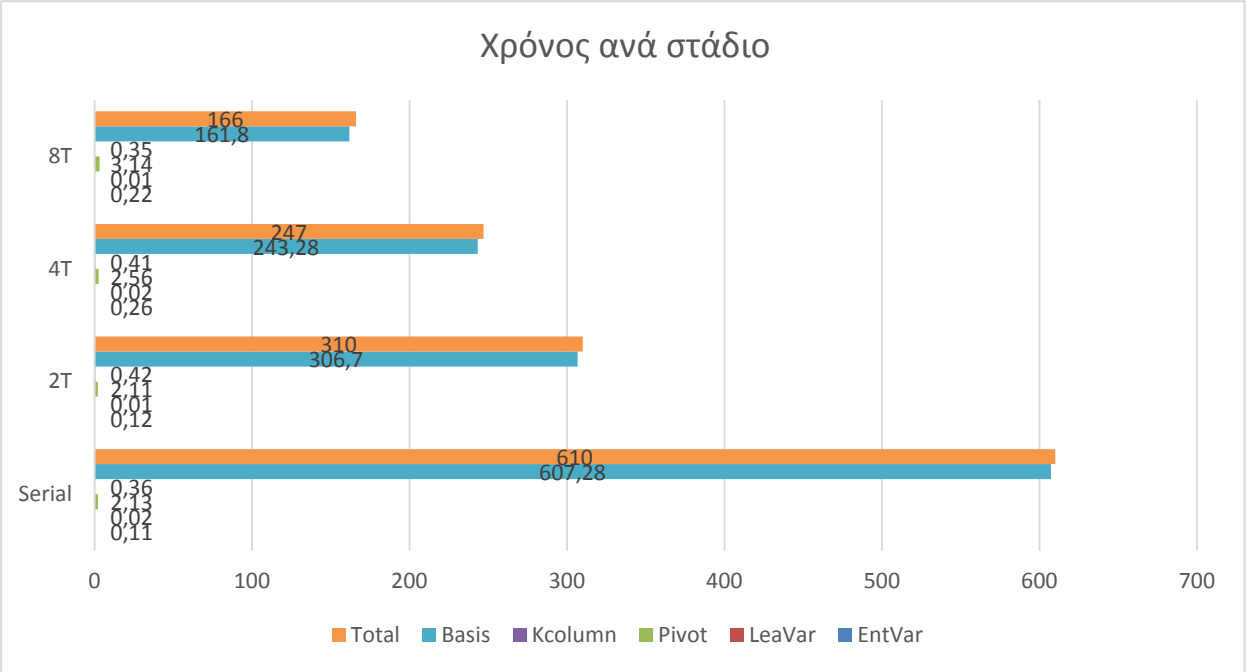


	1000	2000	3000	4000	5000
Serial	51	610	3560	12631	29809
2T	26	310	1941	6401	15140
4T	22	247	1435	4981	11608
8T	13	166	977	3369	7940

Εικόνα 5. Συγκεντρωτικός πίνακας εκτελέσεων Open MP

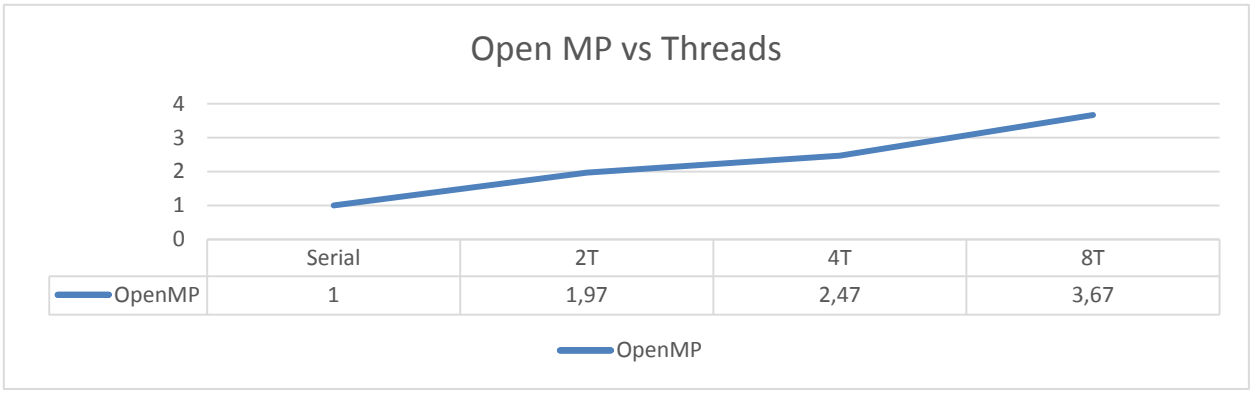
Από τα παραπάνω γραφήματα προκύπτει ότι η συνεχή προσθήκη νημάτων ολοένα και μείωνε τον χρόνο εκτέλεσης. Το μεγαλύτερο άλμα στην επιτάχυνση έγινε κατά την πρώτη διάσπαση σε 2 νήματα. Αυτό οφείλεται στο γεγονός ότι τα νήματα είχαν παραπάνω διαθέσιμη κρυφή μνήμη cache για λιγότερες μετακινήσεις δεδομένων από και προς την κύρια μνήμη RAM και μεγαλύτερη συνοχή στα περιεχόμενα τους.

Χαρακτηριστικά για το πρόβλημα των 2000 μεταβλητών ο επιμερισμός του χρόνου εκτέλεσης ήταν:



Εικόνα 6. Σύγκριση νημάτων Open MP 2000 μεταβλητών

Όπως φαίνεται από το παραπάνω γράφημα, τα περισσότερα νήματα βοηθούν στην γρηγορότερη ενημέρωση της βάσης. Συνεπώς η επιτάχυνση που επιτεύχθηκε ήταν:



Εικόνα 7. Γράφημα επιτάχυνσης νημάτων Open MP 2000 μεταβλητών

6. ΥΛΟΠΟΙΗΣΗ ΤΟΥ SIMPLEX

6.1 Εισαγωγή

Στο κεφάλαιο αυτό μελετάμε μια υλοποίηση ενός αλγόριθμου Simplex.

6.2 Υλοποίηση σε Ψευδογλώσσα

Με βάση την περιγραφή του Πρωτεύοντος Αλγορίθμου Simplex υλοποιήθηκε μια ελαφρά τροποποιημένη [17], βέλτιστη υπολογιστικά έκδοση του σε σειριακή μορφή σε ψευδο-γλώσσα:

```
Αλγόριθμος Simplex
        Διάβασε Πίνακα
        Δημιούργησε το Tableau
        Φάση 1η ()
        Επίλυση ()
        Εκτύπωσε Λύση
Τέλος Αλγόριθμος Simplex

Φάση 1η()
        Έλεγχος για υπαρκτή λύση / αδυναμία
        Εντόπισε μεταβλητή σύμφωνα με κανόνα περιστροφής
        Ενημέρωσε Tableau
Τέλος Φάση 1η

Επίλυση()
        Επίσκεψη όλων των κορυφών
        Έλεγχος για αοριστία
        Εντόπισε μεταβλητή σύμφωνα με κανόνα περιστροφής
        Ενημέρωσε Tableau
Τέλος Επίλυση()
```

Εικόνα 8. Σκελετός Αλγόριθμου Simplex σε Ψευδογλώσσα

6.3 Μελέτη Υλοποίησης

Για την είσοδο των δεδομένων για γνωστά προβλήματα που έχουν δημοσιοποιηθεί χρησιμοποιήθηκε το ευρέως διαδεδομένο πρότυπο mps. Το φορμάτ mps δημιουργήθηκε την εποχή των διάτρητων καρτών και περιγράφει προβλήματα γραμμικου προγραμματισμού. Το μεγαλύτερο πλήθος προβλημάτων που είναι διαθέσιμα στην βιβλιοθήκη MIPLIB [18] πρόκειται για προβλήματα ελαχιστοποίησης για αυτό και υλοποιήθηκε η μορφή εύρεσης ελαχίστου. [16]

Τα στοιχεία του ταμπλό διαβάζονται από την συνάρτηση glp_read_mps() σε μια δομή glp_prob σύμφωνα με την βιβλιοθήκη GNU GLPK [19] και ανάλογα με το τι περιορισμό εκφράζουν τοποθετούνται με τα ανάλογα πρόσημα (αρνητικά θετικά για μεγαλύτερο μικρότερο), τις ανάλογες φορές (2 σε περίπτωση ύπαρξης ισότητα) και δημιουργούνται οι απαραίτητες χαλαρές μεταβλητές. Κατά συνέπεια η πρώτη σειρά του πίνακα περιέχει τις τιμές C του διανύσματος κόστους, η πρώτη στήλη X του πίνακα περιέχει τις σταθερές τιμές των περιορισμών και κάθε γραμμή T(i,1) έως T(i,m) τους συντελεστές των περιορισμών. Το ταμπλό διαμορφώνεται στην μνήμη όπως στην παρακάτω εικόνα.

C	C_1	...	...	C_{m-1}	C_m
X_1	$T_{(1,1)}$	...	...	$T_{(1,m-1)}$	$T_{(1,m)}$
...	...	...	...	...	...
...	...	...	...	...	...
X_{n-1}	$T_{(n-1,1)}$	...	...	$T_{(n-1,m-1)}$	$T_{(n-1,m)}$
X_n	$T_{(n,1)}$	...	...	$T_{(n,m-1)}$	$T_{(n,m)}$

Εικόνα 9. Το Ταμπλό στην μνήμη

Στην συνέχεια, ο πίνακας περνάει την πρώτη φάση επεξεργασίας. Σε αύτην την φάση γίνεται το μεγαλύτερο υπολογιστικό μέρος, καθώς ελέγχεται η δυνατότητα ικανοποίησης των περιορισμών με συνέπεια τον περιορισμό του πλήθους τους και η εύρεση ενός σημείου εκκίνησης. Σε περίπτωση που δεν βρεθεί τέτοιο σημείο το πρόβλημα είναι αδύνατο. Σε κάθε άλλη περίπτωση ο πίνακας μετασχηματίζεται σε ένα πιο βέλτιστο υπολογιστικά πρόβλημα πιο κοντά στην λύση, εντοπίζοντας το μέγιστο στοιχείο και διαιρόντας με αυτό.

```
for (idx = 0; idx < row_size; idx++) {
  SimplexTableau[(r)*row_size + idx + 1] = SimplexTableau[(r)*row_size + idx + 1] / w;
}
```

Εικόνα 10. Ενημέρωση οδηγού γραμμής

Μετά το πέρας αυτής της τροποποίησης, επισκεπτόμαστε τις κορυφές που έχουν δημιουργηθεί και ελέγχουμε αν κάποια αποτελεί βέλτιστη λύση ή έχουμε αόριστο πρόβλημα. Ο πίνακας μετασχηματίζεται και σε αυτήν την φάση αλλά αντίθετα με την πρώτη φάση εντοπίζουμε το ελάχιστο στοιχείο.

```
for ( idx = 0; idx < height; idx++) {
        if (idx != r) {

                for (j = 0; j < row_size; j++) {
                        SimplexTableau[idx*row_size + j + 1] -= SimplexTableau[(r)*row_size + j + 1] *
ColumnK[idx + 1];

                        if (fabs(SimplexTableau[idx*row_size + j + 1]) < epsilon) {
                                SimplexTableau[idx*row_size + j + 1] = 0;
                        }
                        if (j == k) {
                                if (fabs(SimplexTableau[idx*row_size + j + 1]) > epsilon)
                                        SimplexTableau[idx*row_size + j + 1] = 0;
                        }
                }
        }
}
```

Εικόνα 11. Σειριακή ενημέρωση πίνακα βάσης Simplex

6.4 Εφαρμογή Παραλληλίας στην Υλοποίηση

Όπως αναφέραμε στην προηγούμενη ενότητα, το κυριότερο σημείο προς παραλληλοποίηση είναι η ενημέρωση της βάσης [20] [21]. Η διαδικασία αυτή, στην υλοποίηση μας, περιλαμβάνει δύο φάσεις. Μία με την οποία ενημερώνεται μόνο η οδηγός γραμμή και μια με την οποία ενημερώνονται τα υπόλοιπα στοιχεία του πίνακα.

Παράλληλη Έκδοση Φάση 1η()

Έλεγχος για βέλτιστη λύση

Εντόπισε κορυφή σύμφωνα με κανόνα περιστροφής

Παράλληλη Έκδοση Ενημέρωσε Tableau

Τέλος Φάση 1η

Παράλληλη Έκδοση Επίλυση()

Έλεγχος για αοριστία

Εντόπισε κορυφή σύμφωνα με κανόνα περιστροφής

Παράλληλη Έκδοση Ενημέρωσε Tableau

Τέλος Επίλυση()

Εικόνα 12. Σημεία εν δυνάμη παραλληλίας στον Simplex

Η διάσπαση των νημάτων μπορεί να γίνει με ποικίλους τρόπους. Η απλοϊκή υλοποίηση περιλαμβάνει να χρησιμοποιηθεί η απλή δομή *#pragma omp parallel for* είτε με συγκεκριμένο schedule είτε με αυτόματο. Το API αναλαμβάνει μόνο του τον διαμοιρασμό εργασίας

μεταξύ των νημάτων ή διασπά στα chunks που ορίζονται. Εναλλακτικά, μπορεί να υλοποιηθεί χειροκίνητος διαμοιρασμός της περιοχής που αναλαμβάνει το κάθε νήμα.

```
#pragma omp parallel for private(idx)
for (idx = 0; idx < row_size; idx++) {
  SimplexTableau[(r)*row_size + idx + 1] = SimplexTableau[(r)*row_size + idx + 1] / w;
}
```

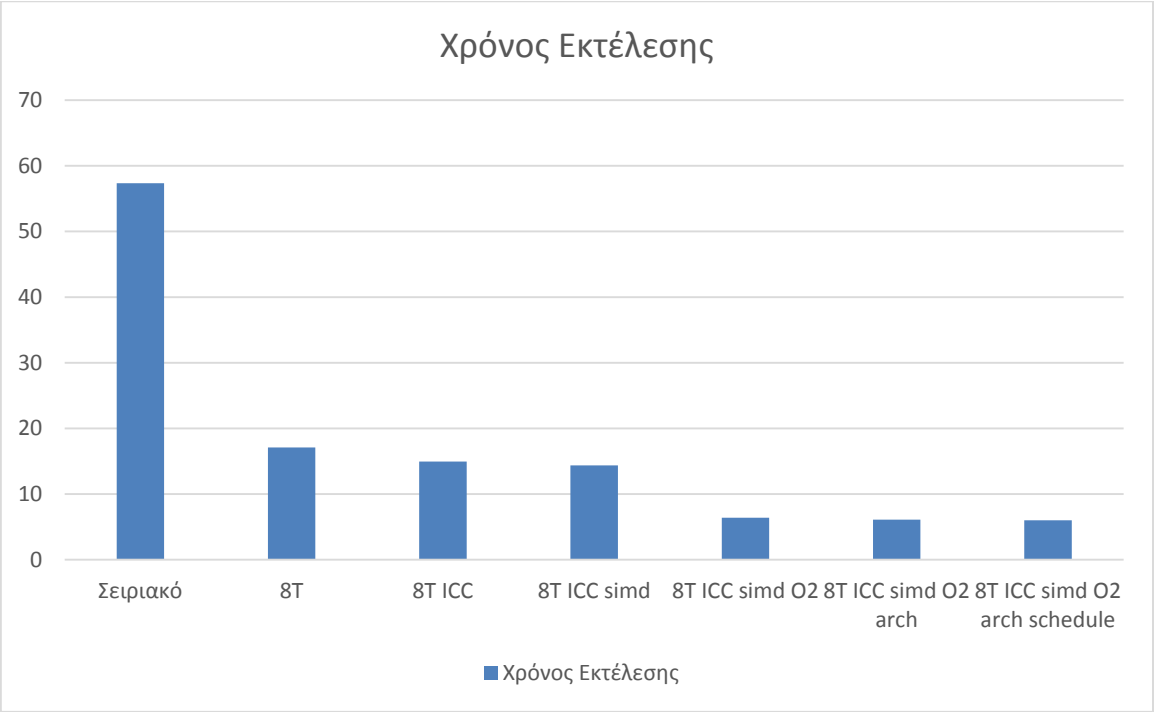
Εικόνα 13. Ενημέρωση οδηγού γραμμής Simplex με χρήση Open MP (δομή for)

```
#pragma omp parallel for private (idx,j) shared(SimplexTableau)
for ( idx = 0; idx < height; idx++) {
        if (idx != r) {

                #pragma simd
                for (j = 0; j < row_size; j++) {
                        SimplexTableau[idx*row_size + j + 1] -= SimplexTableau[(r)*row_size + j + 1] *
ColumnK[idx + 1];

                        if (fabs(SimplexTableau[idx*row_size + j + 1]) < epsilon) {
                                SimplexTableau[idx*row_size + j + 1] = 0;
                        }
                        if (j == k) {
                                if (fabs(SimplexTableau[idx*row_size + j + 1]) > epsilon)
                                        SimplexTableau[idx*row_size + j + 1] = 0;
                        }
                }
        }
}
```

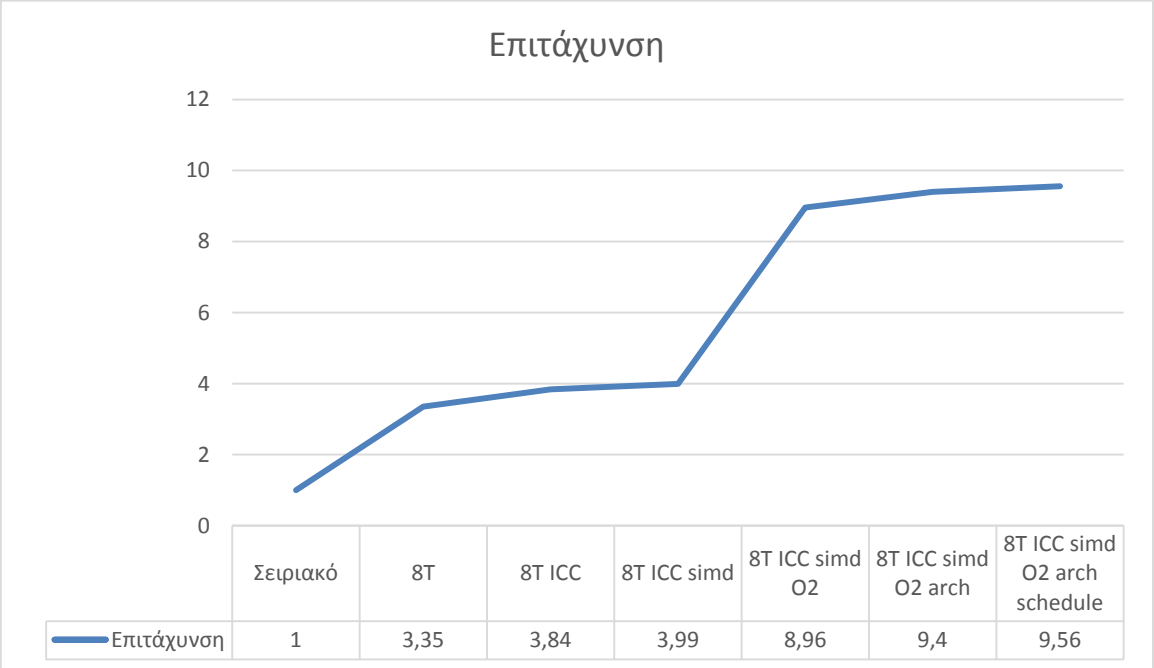
Εικόνα 14. Ενημέρωση πίνακα βάσης Simplex με χρήση Open MP (δομή for)

6.5 Υπολογιστική Μελέτη Υλοποίησης

Για την υπολογιστική μελέτη χρησιμοποιήθηκε το δημοσιευμένο πρόβλημα ic97_potential, το οποίο πρόκειται για ένα μοντέλο βελτιστοποίησης δρομολογίων τραίνων με 523 μεταβλητές. Η πρώτη εκτέλεση έγινε σειριακά όπου και απαιτήθηκαν 57,36 δευτερόλεπτα κατά μέσο όρο για την επιλυσή του. Έπειτα εκτελέστηκε η έκδοση Open MP όπως παρουσιάστηκε στο προηγούμενο υποκεφάλαιο. Ο χρόνος έπεσε στα 17,1 δευτερόλεπτα περίπου, παρουσιάζοντας επιτάχυνση 3,35 φορές. Η μετάβαση στον Intel C++ Compiler οδήγησε σε περεταίρω μείωση του χρόνου εκτέλεσης στα 14,94 δευτερόλεπτα, δηλαδή 3,84 φορές από το σειριακό. Λόγω της δομής if που υπάρχει μέσα στην ενημέρωση του πίνακα βάσης του Simplex, ο ICC πρότεινε την χρήση του #pragma simd το οποίο βοηθά στην επιταχυνση, αν και ελάχιστο, στις 3,99 φορές από το σειριακό. Σειρά στην μελέτη είχε η εκτέλεση με ενεργοποίηση του δευτέρου επιπέδου βελτιστοποίησης O2 στον μεταγλωτιστή το οποίο οδήγησε σε χρόνο εκτέλεσης 6,4 δευτερολέπτων, το οποίο μεταφράζεται σε επιτάχυνση 8,96 φορές. Συνεχίζοντας την βελτιστοποίηση η ενεργοποίηση της υποστήριξης AVX εντολών τύπου Skylake στον επεξεργαστή ρίχνει τον χρόνο εκτέλεσης στα 6,1 δευτερόλεπτα ή 9,4 φορές πιο γρήγορα. Τέλος η προσθήκη static schedule λόγω της σταθερής φύσης πράξεων του προβλήματος έριξε το χρόνο στα 6 δευτερόλεπτα, πράγμα που σημαίνει επιτάχυνση 9,56 φορές για το δεδομένο πρόβλημα.



Εικόνα 15. Χρόνος Εκτέλεσης Simplex ic97_potential



Εικόνα 16. Επιτάχυνση στο ic97_potential

7. ΣΥΜΠΕΡΑΣΜΑΤΑ

Η παρούσα μελέτη επικεντρώθηκε στην εφαρμογή του Open MP API στον αλγόριθμο Simplex. Γίνεται αντιληπτή η ευκολία με την οποία υλοποιήθηκε η εν λόγω εφαρμογή καθώς και τα άμεσα ώφελη που προέκυψαν με τις ενδιάμεσες βελτιστοποιήσεις. Με την προσθήκη δύο στοχευμένων γραμμών κώδικα και το συμπλήρωμα των αντίστοιχων υποστηρικτικών συναρτήσεων, το σειριακό πρόγραμμα μετατράπηκε σε παράλληλο άμεσα.

Στην έκδοση του Lalami επιτεύχθηκε επιτάχυνση 3,37 φορών. Επιπλέον η επιτάχυνση 9,56 φορών σε σχέση με τον σειριακό που επιτεύχθηκε στο συγκεκριμένο πρόβλημα είναι ενθαρρυντική. Ανοικτή είναι η εμβάθυνση σε αρχιτεκτονικές ακόμα μεγαλύτερης παραλληλίας όπως οι Intel Xeon Phi παράλληλοι βοηθητικοί επεξεργαστές των 240 νημάτων.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός Όρος**
Parallel Programming	Παράλληλος προγραμματισμός
Multiprocessors	Πολυεπεξεργαστές
Parallel System	Παράλληλα Συστήματα
Instruction-level parallelism	Παραλληλία σε επίπεδο εντολών

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

CPU	Central Processing Unit
Open MP	Open Multi-Processing
POSIX	Portable Operating System Interface
API	Application Programming Interface

ΑΝΑΦΟΡΕΣ

[1] N. Ploskas, *Παράλληλος προγραμματισμός περιστροφικών αλγορίθμων εξωτερικών σημείων τύπου simplex,* Πανεπιστήμιο Μακεδονίας, 2009.

[2] E.-S. Badr, K. Paparrizos, N. Samaras και A. Sifaleras, «On the basis inverse of the exterior point simplex algorithm,» σε *E.E.E.E. "Διαχείρηση Κινδύνων"*, 2005.

[3] M. E. Thomadakis και J.-C. Liu, «An Efficient Steepest-Edge Simplex Algorithm for SIMD Computers,» Texas A&M University, Department of Computer Science, 1996.

[4] M. A. Louka, *Παράλληλες Επαναληπτικές Μέθοδοι,* Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών, Σχολή Θετικών Επιστημών, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, 2005.

[5] D. E. Elbeltagi, *The Simplex Method,* System Analysis.

[6] E.-S. M. Badr, *Παράλληλος Προγραμματισμός Αλγορίθμων για Προβλήματα Γραμμικού Προγραμματισμού,* Thessaloniki, 2006.

[7] OpenMP, *Application Program Interface Version 4.5,* 2015.

[8] M. E. Lalami, *Contribution à la résolution de problèmes d'optimisation combinatoire : méthodes séquentielles et parallèles.,* UniversitÉ De Toulouse, 2012.

[9] M. E. Lalami και D. E.-B. Vincent Boyer, «Efficient Implementation of the Simplex Method on a CPU-GPU System,» σε *IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, 2011.

[10] N. Ploskas και N. Samaras, «A Computational Comparison of Basis Updating Schemes for the Simplex Algorithm on a CPU-GPU System,» *American Journal of Operations Research,* αρ. 3, pp. 497-505, 2013.

[11] G. Yarmish, *A Distributed Implementation of the Simplex Method,* Polytechnic University, 2011.

[12] P. D. Michailidis και K. G. Margaritis, «Open Multi Processing (OpenMP) of Gauss-Jordan Method for Solving System of Linear Equations,» σε *11th IEEE International Conference on Computer and Information Technology*, 2011.

[13] P. D. Michailidis και K. G. Margaritis, «Parallel direct methods for solving the system of linear equations with pipelining on a multicore using OpenMP,» *Journal of Computational and Applied Mathematics,* αρ. 236, pp. 326-341, 2011.

[14] K. Border, «The Gauss–Jordan and Simplex Algorithms,» California Insitute of Technology, Division of the Humanities and Social Sciences, 2004.

[15] A. Buluc¸, J. R. Gilbert και C. Budak, «Gaussian Elimination Based Algorithms on the GPU,» University of California, Santa Barbara, 2008.

[16] D. Lee και M. Wiswall, «A Parallel Implementation of the Simplex Function Minimization Routine,» New York University, Department of Economics.

[17] K. Mehlhorn, *Implementation of the Simplex Algorithm,* 2010.

[18] G. Gamrath, «MIPLIB – the Mixed Integer Programming LIBrary,» Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), [Ηλεκτρονικό]. Available: http://miplib.zib.de/. [Πρόσβαση 24 5 2017].

[19] A. Makhorin, «GLPK (GNU Linear Programming Kit),» Free Software Foundation, Inc., [Ηλεκτρονικό]. Available: http://www.gnu.org/software/glpk/#TOCdocumentation. [Πρόσβαση 17 6 2015].

[20] M. Lubin, J. A. J. Hall, C. G. Petra και M. Anitescu, *Parallel distributed-memory simplex for large-scale stochastic LP problems,* 2000.

[21] M. Bradsher, K. Fang, A. Kaiser και M. Ortega, *Simplex Algorithm - Parallel Optimizations,* University of California, Berkeley, 2012.