ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗ ΜΕΘΕΥΡΕΤΙΚΟΥ ΑΛΓΟΡΙΘΜΟΥ ΓΙΑ ΠΡΟΒΛΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΑΠΟΘΕΜΑΤΩΝ : ΜΙΑ ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ ΜΕ OPENMP ΚΑΙ OPENACC

Διπλωματική Εργασία

του

Νικολάου Αντωνιάδη

Θεσσαλονίκη, Ιούνιος 2016

ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗ ΜΕΘΕΥΡΕΤΙΚΟΥ ΑΛΓΟΡΙΘΜΟΥ ΓΙΑ ΠΡΟΒΛΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΑΠΟΘΕΜΑΤΩΝ : ΜΙΑ ΥΠΟΛΟΓΙΣΤΙΚΗ ΜΕΛΕΤΗ ΜΕ OPENMP ΚΑΙ OPENACC

Νικόλαος Αντωνιάδης

Πτυχίο Μηχανικού Πληροφορικής, ΤΕΙ Αθήνας, 1995

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Άγγελος Σιφαλέρας

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21/06/2016

Άγγελος Σιφαλέρας	Κωνσταντίνος Μαργαρίτης	Νικόλαος Σαμαράς
.....	.....	.....

Νικόλαος Αντωνιάδης

.....

Περίληψη

Η παρούσα εργασία έχει ως στόχο την παραλληλοποίηση ενός NP-Hard προβλήματος διαχείρισης αποθεμάτων, χρησιμοποιώντας τα μοντέλα παράλληλου προγραμματισμού OpenMP και OpenACC. Στην πορεία της εργασίας, θα δοθεί η σημασία των μεθευρετικών αλγορίθμων (metaheuristics) και θα αναλυθεί ιδιαίτερα ο αλγόριθμος Αναζήτησης Μεταβλητής Γειτονιάς (VNS). Γίνεται, επίσης, μια παρουσίαση των προβλημάτων διαχείρισης αποθεμάτων. Αφού αποτυπωθεί το πλαίσιο το οποίο πραγματεύεται η παρούσα εργασία, γίνεται ανάλυση των παράλληλων μεθευρετικών αλγορίθμων και των τεχνολογιών με τις οποίες μπορούν να υλοποιηθούν, ιδιαιτέρως των OpenMP, OpenACC, καθώς και παραδείγματα παραλληλοποίησης του αλγορίθμου VNS στον οποίο και θα βασιστούμε. Μελετώντας τους αλγορίθμους και τις τεχνικές παραλληλοποίησης καταλήγουμε στο μοντέλο που ταιριάζει καλύτερα στο πρόβλημά μας και το υλοποιούμε, αποτυπώνοντας τις τεχνικές λεπτομέρειες. Αφού ολοκληρωθεί η υλοποίηση, διενεργούνται δοκιμές του παραλληλοποιημένου αλγορίθμου τόσο με την τεχνολογία OpenMP όσο και με την τεχνολογία OpenACC και τα αποτελέσματα παρουσιάζονται με σχετικούς πίνακες και διαγράμματα. Η εργασία ολοκληρώνεται με τα συμπεράσματα της έρευνας και τις μελλοντικές κατευθύνσεις της.

Λέξεις Κλειδιά: μεθευρετικοί αλγόριθμοι, παραλληλοποίηση, προβλήματα διαχείρισης παραγωγής και αποθεμάτων, OpenMP, OpenACC

Abstract

Parallelization of a metaheuristic algorithm, in order to be able to solve complex inventory control problems, using OpenACC as well as OpenMP technologies is a challenging problem nowadays. This thesis tries to clear out this parallelization process firstly by introducing the terms metaheuristics, Variable Neighborhood Search (VNS) as well as giving some information about inventory control problems. Next, the parallelization process of metaheuristics using OpenMP and OpenACC technologies is explained and appropriate examples are introduced. The next step is to present the implementation of the inventory control problem using the above parallelization process, visualizing the results of the computational experiments mentioning the quality of the results. Closing, the summary of the thesis is presented as well as future research.

Keywords: metaheuristics, parallelization, inventory control problems, OpenMP, OpenACC

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Άγγελο Σιφαλέρα για την πολύτιμη βοήθειά του σε όλη τη διάρκεια της διπλωματικής. Επίσης, ευχαριστώ ιδιαίτερα την Κική που εκτός την καθημερινή της στήριξη με παρότρυνε να παρακολουθήσω το παρόν ΠΜΣ. Τέλος, αισθάνομαι την ανάγκη να ευχαριστήσω θερμά τη Βάϊα και τον Αντώνη που χωρίς αυτούς δε θα ήταν δυνατό να τα καταφέρω.

Αφιερώνεται στη μνήμη της γιαγιάς μου, Ασπασίας και της θείας μου, Ελένης

ΠΕΡΙΕΧΟΜΕΝΑ

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κατάλογος Αλγορίθμων

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Σημαντικότητα του θέματος

Τα προβλήματα διαχείρισης αποθεμάτων αποτελούν, πλέον, καθημερινότητα στον τομέα της βιομηχανίας. Για να επιλυθούν, λόγω της πολυπλοκότητά τους (NP-Hard), δεν αρκούν οι ακριβείς (exact) λύσεις, που επιλύουν τέτοια προβλήματα αλλά σε μη ικανοποιητικό χρόνο. Στα προβλήματα αυτά εφαρμόζονται μεθευρετικοί αλγόριθμοι με καλά αποτελέσματα, σε ότι αφορά την ποιότητα των λύσεων, και σε σχετικά ικανοποιητικό χρόνο. Το ιδανικό θα ήταν να πετυχαίνουμε όσο το δυνατόν καλύτερα αποτελέσματα σε ελάχιστο χρόνο.

Σκοπός - στόχοι

Η παρούσα εργασία έχει ως κύριο στόχο την παραλληλοποίηση ενός μεθευρετικού αλγορίθμου για ένα πρόβλημα διαχείρισης αποθεμάτων, ώστε:

- να διερευνηθεί ποιες τεχνολογίες παραλληλοποίησης ταιριάζουν στο πρόβλημα και το μεθευρετικό αλγόριθμο
- να εφαρμοστούν οι τεχνολογίες παραλληλοποίησης OpenMP και OpenACC
- να βελτιωθούν τόσο ο χρόνος εκτέλεσης όσο και η ποιότητα των λύσεων

Διάρθρωση της εργασίας

Αρχικά, στην εργασία, παρατίθεται μια εισαγωγή στους μεθευρετικούς αλγορίθμους και ιδιαίτερα στον αλγόριθμο Αναζήτησης Μεταβλητής Γειτονιάς (Variable Neighborhood Search - VNS). Έπειτα, παρουσιάζονται τα προβλήματα βελτιστοποίησης στον κλάδο της διαχείρισης αποθεμάτων και παραγωγής. Στη συνέχεια, αφιερώνεται ένα κεφάλαιο για τη διαδικασία παραλληλοποίησης μεθευρετικών αλγορίθμων καθώς και των τεχνολογιών παραλληλοποίησης.

Στο επόμενο κεφάλαιο περιγράφονται τα τεχνικά θέματα υλοποίησης. Έπειτα, αφιερώνεται ένα κεφάλαιο στην παρουσίαση μίας σειράς υπολογιστικών πειραμάτων τόσο για τον έλεγχο της ποιότητας των λύσεων όσο και για τον έλεγχο απόδοσης. Το συγκεκριμένο κεφάλαιο ολοκληρώνεται με την οπτικοποιημένη παρουσίαση των αποτελεσμάτων των υπολογιστικών πειραμάτων. Τέλος, η ολοκλήρωση της παρούσας εργασίας πραγματοποιείται με την παράθεση σχετικών συμπερασμάτων καθώς και των κατευθύνσεων της μελλοντικής έρευνας.

ΚΕΦΑΛΑΙΟ 2

Μεθευρετικές Μέθοδοι

Εισαγωγή

Οι **μεθευρετικές** (Metaheuristics) **μέθοδοι** (Luke 2013, Sörensen and Glover 2013, Hertz and Widmer 2003, Blum and Roli 2003) αποτελούν ένα αλγοριθμικό πλαίσιο υψηλού επιπέδου στο πεδίο της στοχαστικής βελτίωσης. Είναι ανεξάρτητες από προβλήματα και προσφέρουν ένα σύνολο από κατευθύνσεις/στρατηγικές για την ανάπτυξη ευρετικών αλγορίθμων βελτιστοποίησης. Οι τεχνικές που χρησιμοποιούν ενσωματώνουν κάποιο βαθμό τυχαιότητας για την εύρεση όσο το δυνατό καλύτερων λύσεων σε δύσκολα προβλήματα. Συνήθως, στα προβλήματα αυτά δε γνωρίζουμε από πριν πώς «μοιάζει» η βέλτιστη λύση, ούτε μπορούμε να την αναζητήσουμε σε όλες τις πιθανές λύσεις επειδή αυτές είναι πάρα πολλές. Οπότε, έχοντας ως δεδομένη μια πιθανή λύση του προβλήματος, ελέγχουμε την ποιότητά της. Χαρακτηριστικά παραδείγματα μεθευρετικών αλγορίθμων περιλαμβάνουν τους γενετικούς/εξελικτικούς αλγορίθμους (genetic/evolutionary algorithms), tabu search, simulated annealing, ant colony optimization, αλγόριθμος μεταβλητής γειτονιάς (VNS) και άλλους. Ο όρος χρησιμοποιήθηκε αρχικά από τον Glover (Glover 1986) και συνδυάζει το πρόθεμα μετα- (με την έννοια του υψηλότερου επιπέδου αφαίρεσης) με τη λέξη ευρετικός (από τη λέξη εὑρίσκειν), με άλλα λόγια ευρετικές μέθοδοι για τις ευρετικές μεθόδους.

Οι μεθευρετικοί αλγόριθμοι είναι ευρετικοί από τη φύση τους(Sörensen and Glover 2013), όπως δηλώνει και το όνομά τους. Αυτό, στην πράξη, τους διαχωρίζει από τις ακριβείς (exact) μεθόδους οι οποίες αποδεδειγμένα βρίσκουν βέλτιστη λύση σε πεπερασμένο χρόνο, ωστόσο συνήθως αυτός είναι απαγορευτικά μεγάλος. Οι μεθευρετικοί αλγόριθμοι αναπτύσσονται ώστε να βρίσκουν μια «αρκετά καλή» λύση σε «αρκετά μικρό» υπολογιστικό χρόνο. Πολλές φορές, μάλιστα, οι λύσεις που βρίσκουν είναι ισάξιες ή ακόμα και καλύτερες από τις παραδοσιακές ακριβείς (exact) μεθόδους όπως από την branch and bound και από αυτή του δυναμικού προγραμματισμού (dynamic programming). Ειδικά για τα πολύ δύσκολα προβλήματα (NP-

Hard) ή τα προβλήματα με πολλά στιγμιότυπα (instances) προσφέρουν καλύτερη ποιότητα λύσης ανά μονάδα υπολογιστικού χρόνου. Επιπροσθέτως, οι μεθευρετικές μέθοδοι είναι περισσότερο ευέλικτες από τις ακριβείς (exact) μεθόδους για δύο λόγους: Κατ' αρχάς, επειδή μπορούν να ταιριάξουν στις ανάγκες των περισσότερων καθημερινών (real-life) προβλημάτων βελτιστοποίησης. Δεύτερον, επειδή οι μεθευρετικές μέθοδοι δεν έχουν απαιτήσεις σε ότι αφορά τη διατύπωση του προβλήματος (όπως περιορισμούς ή αντικειμενικές συναρτήσεις). Ωστόσο αυτή η ευελιξία έχει ως κόστος τη σημαντική προσαρμογή της κάθε μεθευρετικής μεθόδου σε κάθε πρόβλημα ώστε να επιτευχθεί καλή απόδοση.

Παρότι υπάρχουν πολλές κριτικές στις μεθευρετικές μεθόδους κυρίως σε ότι αφορά την έλλειψη κοινής μεθοδολογίας σχεδιασμού, την έλλειψη αυστηρότητας στον έλεγχο και τη σύγκριση διαφορετικών υλοποιήσεων αλλά και την τάση να δημιουργούνται πολύπλοκες μέθοδοι με πολλούς διαφορετικούς τελεστές, κανείς δεν μπορεί να διαφωνήσει με την επιτυχία τους. Η ικανότητά τους να πετυχαίνουν καλές λύσεις εκεί που οι άλλες μέθοδοι αποτυγχάνουν έχουν κάνει τις μεθευρετικές μεθόδους την κύρια επιλογή για την επίλυση της πλειοψηφίας των μεγάλων καθημερινών προβλημάτων βελτιστοποίησης είτε σε ερευνητικές είτε σε εταιρικές εφαρμογές (Hertz and Widmer 2003).

Κατηγοριοποίηση μεθευρετικών μεθόδων

Οι μεθευρετικές μέθοδοι βασίζονται σε (Blum and Roli 2003, Sörensen and Glover 2013) παραδείγματα από τη φύση πολλές φορές άσχετα με την διαδικασία της βελτιστοποίησης όπως τη φυσική εξέλιξη (genetic/evolutionary algorithms), την ψύξη ενός κρυσταλλικού στερεού (simulated annealing) ή τη συμπεριφορά σμήνους ζώων (π.χ., ant colony optimization). Άλλες μέθοδοι, όπως το tabu search, επικεντρώνονται στη διερεύνηση της δομής του προβλήματος ώστε να βελτιωθεί η αναζήτηση καλών λύσεων. Σε γενικές γραμμές, οι μεθευρετικές μέθοδοι βασίζονται κυρίως στην τύχη αν και έχουν προταθεί και ντετερμινιστικές στρατηγικές.

Ανάλογα από ποια σκοπιά τους παρατηρούμε, υπάρχουν διαφορετικοί τρόποι για να ταξινομηθούν και να περιγραφούν οι μεθευρετικοί αλγόριθμοι. Παρακάτω, παρουσιάζονται συνοπτικά οι πιο σημαντικοί τρόποι κατηγοριοποίησης μεθευρετικών αλγορίθμων (Blum and Roli 2003):

Εμπνευσμένοι από τη φύση ή όχι

Υπάρχουν αλγόριθμοι εμπνευσμένοι από τη φύση όπως οι γενετικοί (genetic algorithms) και ο Ant Colony Optimization αλλά και μη εμπνευσμένοι από τη φύση όπως ο Tabu Search και ο Iterated Local Search. Όμως, πολλοί υβριδικοί αλγόριθμοι δεν μπορούν να κατηγο-

ριοποιηθούν σε καμία από τις δύο κατηγορίες και είναι δύσκολο να διακρίνουμε εάν κάποιος αλγόριθμος είναι εμπνευσμένος από τη φύση. Για παράδειγμα, δεν είναι αυτονόητο ότι η χρήση της μνήμης στο Tabu Search είναι εμπνευσμένη από τη φύση.

Βασισμένοι σε πληθυσμό λύσεων ή όχι

Άλλο ένα χαρακτηριστικό που μπορεί να χρησιμοποιηθεί ώστε να κατηγοριοποιηθούν οι μεθευρετικοί αλγόριθμοι είναι ο αριθμός των λύσεων που χρησιμοποιούνται κάθε στιγμή. Αλγόριθμοι που χρησιμοποιούν μία λύση τη φορά είναι οι Tabu Search, Iterated Local Search και ο Variable Neighborhood Search. Όλοι οι παραπάνω περιγράφουν μια τροχιά στο χώρο αναζήτησης κατά τη διαδικασία της αναζήτησης. Αντιθέτως, οι βασισμένοι σε πληθυσμό λύσεων αλγόριθμοι πραγματοποιούν διαδικασίες αναζήτησης οι οποίες περιγράφουν την εξέλιξη ενός συνόλου σημείων στο χώρο αναζήτησης.

Δυναμική ή στατική αντικειμενική συνάρτηση

Οι μεθευρετικοί αλγόριθμοι μπορούν να κατηγοριοποιηθούν σύμφωνα με τον τρόπο που χρησιμοποιούν την αντικειμενική συνάρτηση. Παρότι κάποιοι αλγόριθμοι διατηρούν «ως έχει» την αντικειμενική συνάρτηση που δίνεται από το πρόβλημα, κάποιοι άλλοι όπως ο Guided Local Search (GLS), την τροποποιούν κατά τη διάρκεια της αναζήτησης. Αυτό γίνεται ώστε να ξεφύγουμε από τοπικά ελάχιστα μεταβάλλοντας το χώρο αναζήτησης. Έτσι, η αντικειμενική συνάρτηση αλλάζει κατά τη διάρκεια της αναζήτησης ώστε να ενσωματώνει πληροφορίες της διαδικασίας αναζήτησης.

Μία ή πολλές δομές γειτονιάς

Οι περισσότεροι μεθευρετικοί αλγόριθμοι χρησιμοποιούν **μία δομή γειτονιάς** ούτως ώστε η δομή της να μην αλλάζει κατά τη διάρκεια της αναζήτησης. Άλλοι μεθευρετικοί αλγόριθμοι όπως ο Variable Neighborhood Search (VNS) χρησιμοποιούν ένα **σύνολο από δομές γειτονιάς** ούτως ώστε να δίνουν τη δυνατότητα να διαφοροποιηθεί η αναζήτηση μέσω της εναλλαγής χώρου αναζήτησης.

Σύνοψη κεφαλαίου

Συνοψίζοντας (Blum and Roli 2003), οι θεμελιώδεις ιδιότητες που χαρακτηρίζουν τις μεθευρετικές μεθόδους είναι:

- Οι μεθευρετικές μέθοδοι είναι στρατηγικές που καθοδηγούν τη διαδικασία αναζήτησης.

- Ο στόχος είναι να εξερευνηθεί αποδοτικά ο χώρος αναζήτησης ώστε να βρεθούν βέλτιστες λύσεις (ή κοντά σε αυτές, αν δεν υπάρχουν βέλτιστες).
- Οι τεχνικές από τις οποίες αποτελούνται οι μεθευρετικοί αλγόριθμοι κυμαίνονται από απλές διαδικασίες τοπικής αναζήτησης μέχρι σύνθετες διαδικασίες εκμάθησης.
- Οι μεθευρετικοί αλγόριθμοι είναι προσεγγιστικοί και συχνότατα στοχαστικοί.
- Μπορεί να περιλαμβάνουν μηχανισμούς ώστε να αποφεύγουν την παγίδευσή τους σε περιορισμένες περιοχές του χώρου αναζήτησης.
- Επιτρέπουν μια περιληπτική περιγραφή των επιπέδων του αλγορίθμου.
- Οι μεθευρετικές μέθοδοι δεν κατασκευάζονται για κάποιο συγκεκριμένο πρόβλημα.

ΚΕΦΑΛΑΙΟ 3

Αναζήτηση Μεταβλητής Γειτονιάς (VNS)

Εισαγωγή

Η αναζήτηση μεταβλητής γειτονιάς Variable neighbourhood search (VNS) είναι (Hansen, Mladenović, and Moreno Pérez 2010) ένας μεθευρετικός αλγόριθμος που βασίζεται στις συστηματικές αλλαγές γειτονιών τόσο κατά τη φάση της κατάβασης (descent), για την εύρεση τοπικού ελάχιστου, όσο και κατά τη φάση της διατάραξης (perturbation), ώστε να ξεφύγει από κάποιο τοπικό ελάχιστο.

Τοπική αναζήτηση

Ο ευρετικός αλγόριθμος τοπικής αναζήτησης βασίζεται στην επιλογή μιας αρχικής λύσης x, την εύρεση της κατεύθυνσης της κατάβασης από το x μέσα στη γειτονιά $N(x)$ μετακινούμενοι στο ελάχιστο της $f(x)$ μέσα στη $N(x)$ κατά την ίδια κατεύθυνση. Αν δεν υπάρχει καμία κατεύθυνση στην κατάβαση ο ευρετικός αλγόριθμος σταματά, αλλιώς επαναλαμβάνεται η διαδικασία. Συνήθως χρησιμοποιείται η πιο απότομη κατεύθυνση της κατάβασης η οποία αναφέρεται και ως η καλύτερη βελτίωση. Το σύνολο των κανόνων συνοψίζονται στον αλγόριθμο 1 (Hansen, Mladenović, and Moreno Pérez 2010) όπου θεωρούμε ότι δίνεται μια αρχική λύση x. Η έξοδος αποτελείται από ένα τοπικό ελάχιστο, το οποίο επίσης συμβολίζεται με x και την τιμή του. Παρατηρούμε ότι η δομή γειτονιάς $N(x)$ ορίζεται $\forall x \in X$. Έπειτα, σε κάθε βήμα, η γειτονιά $N(x)$ του x εξερευνείται ολοκληρωτικά. Επειδή αυτό μπορεί να είναι χρονοβόρο εναλλακτικά χρησιμοποιούμε τον ευρετικό αλγόριθμο κατάβασης. Τα διανύσματα $x_i \in N(x)$ απαριθμούνται συστηματικά και γίνεται η μετακίνηση μόλις βρεθεί μια κατεύθυνση για την κατάβαση όπως φαίνεται στον αλγόριθμο 2 (Hansen, Mladenović, and Moreno Pérez 2010).

Η λογική του αλγορίθμου μεταβλητής γειτονιάς είναι η συστηματική εναλλαγή γειτονιών

Αλγόριθμος 1 Ευρετικός αλγόριθμος καλύτερης βελτίωσης (steepest descent)

```
function BestImprovement(x)
    repeat
        x' ← x
        x ← argmin_{y∈N(x)} f(y)
    until (f(x) ≥ f(x'))
end function
```

Αλγόριθμος 2 Ευρετικός αλγόριθμος πρώτης βελτίωσης

```
function FirstImprovement(x)
    repeat
        x' ← x
        i ← 0
        repeat
            i ← i + 1
            x ← argmin{f(x), f(x_i)}, x_i ∈ N(x)
        until (f(x) < f(x_i) or i = |N(x)|)
    until (f(x) ≥ f(x'))
end function
```

με σκοπό τον απεγκλωβισμό από τα τοπικά βέλτιστα. Η αναζήτηση μπορεί να γίνει με τρεις διαφορετικούς τρόπους:

- Με ντετερμινιστικό τρόπο (αναζητώντας όλη τη γειτονιά της τρέχουσας λύσης)
- Με στοχαστικό τρόπο (επιλέγουμε τυχαία ένα γείτονα της τρέχουσας λύσης)
- Με συνδυασμό ντετερμινιστικού και στοχαστικού τρόπου

Κατάβαση Μεταβλητής Γειτονιάς (VND)

Η μέθοδος κατάβασης μεταβλητής γειτονιάς (Variable Neighbourhood Descent - VND) χρησιμοποιείται όταν η αλλαγή γειτονιάς συμβαίνει με ντετερμινιστικό τρόπο. Τα βήματα του αλγορίθμου παρουσιάζονται στους αλγορίθμους 3 και 4 (Hansen, Mladenović, and Moreno Pérez 2010). Στον αλγόριθμο 3 η συνάρτηση (NeighbourhoodChange) συγκρίνει τη νέα τιμή $f(x')$ με την τρέχουσα τιμή $f(x)$ από τη γειτονιά k. Αν σημειωθεί βελτίωση τότε το k επανέρχεται στην αρχική του τιμή (1) και ανανεώνεται η νέα τρέχουσα τιμή αλλιώς, πηγαίνει στην επόμενη γειτονιά. Οι περισσότερες ευρετικές μέθοδοι αναζήτησης στη φάση κατάβασης χρησιμοποιούν πολύ λίγες γειτονιές (συνήθως μία ή δύο, για παράδειγμα $k'_{max} \leq 2$). Να σημειωθεί ότι η τελική λύση πρέπει να είναι ένα τοπικό ελάχιστο σε σχέση με όλες τις k'_{max} γειτονιές επομένως οι πιθανότητες να βρούμε ένα ολικό ελάχιστο είναι μεγαλύτερες με τη μέθοδο (VND) από ότι αν χρησιμοποιήσουμε μία γειτονιά. Επιπροσθέτως, σε αυτή τη σειριακή επιλογή δομών γειτονιάς στη (VND) μπορεί να αναπτυχθεί εμφωλευμένη στρατηγική. Για παράδειγμα αν $k'_{max} = 3$ τότε μια πιθανή εμφωλευμένη στρατηγική μπορεί να είναι: εφαρμόζουμε τη VND για τις πρώτες δύο γειτονιές σε κάθε σημείο x που ανήκει στην τρίτη γειτονιά.

Αλγόριθμος 3 Συνάρτηση μετακίνησης μέσα στη γειτονιά ή αλλαγή γειτονιάς

function NEIGHBOURHOODCHANGE(x, x', k)
  if $f(x') < f(x)$ **then**
    $x \leftarrow x'$
    $k \leftarrow 1$ ▷ /* Μετακινείται */
  else
    $k \leftarrow k+1$ ▷ /* Αλλάζει γειτονιά */
  end if
end function

Αλγόριθμος 4 Βήματα του βασικού αλγορίθμου κατάβασης (VND)

function VND(x, k'_{max})
  repeat
    $k \leftarrow 1$
    repeat
      $x' \leftarrow argmin_{y \in N'_k(x)} f(x)$ ▷ /* Βρίσκει τον καλύτερο γείτονα στη γειτονιά $N_k(x)$ */
      $NeighbourhoodChange(x, x', k)$ ▷ /* Αλλάζει γειτονιά */
    until $k = k'_{max}$
  until δεν υπήρξε βελτίωση
end function

Μειωμένη Αναζήτηση Μεταβλητής Γειτονιάς (RVNS)

Η μέθοδος μειωμένης αναζήτησης μεταβλητής γειτονιάς (Reduced VNS - RVNS) αποτελεί μια στοχαστική μέθοδο με την οποία αν επιλεχθούν από τη γειτονιά $N_k(x)$ τυχαία σημεία και δε σημειωθεί κατάβαση οι τιμές των νέων αυτών σημείων συγκρίνονται με τα τρέχοντα και, σε περίπτωση βελτίωσης, ενημερώνονται. Συνήθως, ως συνθήκη τερματισμού επιλέγεται ο μέγιστος επιτρεπόμενος χρόνος στην ΚΜΕ (CPU) t_{max} ή ο μέγιστος αριθμός επαναλήψεων μεταξύ δύο βελτιώσεων. Τα βήματα του αλγορίθμου παρουσιάζονται στον αλγόριθμο 5 (Hansen, Mladenović, and Moreno Pérez 2010). Με τη συνάρτηση *Shake* στη γραμμή 4, δημιουργείται ένα σημείο x' τυχαία στη γειτονιά k του x, για παράδειγμα $x' \in N_k(x)$. Η μέθοδος RVNS είναι χρήσιμη σε πολύ μεγάλα στιγμιότυπα για τα οποία η τοπική αναζήτηση έχει κόστος. Έχει παρατηρηθεί ότι η καλύτερη τιμή της παραμέτρου k_{max} είναι συχνά 2. Η μέθοδος RVNS μοιάζει με τη μέθοδο Monte-Carlo (Nicholas Metropolis 1949), αλλά είναι πιο συστηματική.

Βασική Αναζήτηση Μεταβλητής Γειτονιάς (BVNS)

Η μέθοδος βασικής αναζήτησης μεταβλητής γειτονιάς (Mladenović and Hansen 1997) (Basic VNS - BVNS) συνδυάζει ντετερμινιστικές και στοχαστικές αλλαγές στη γειτονιά. Στον

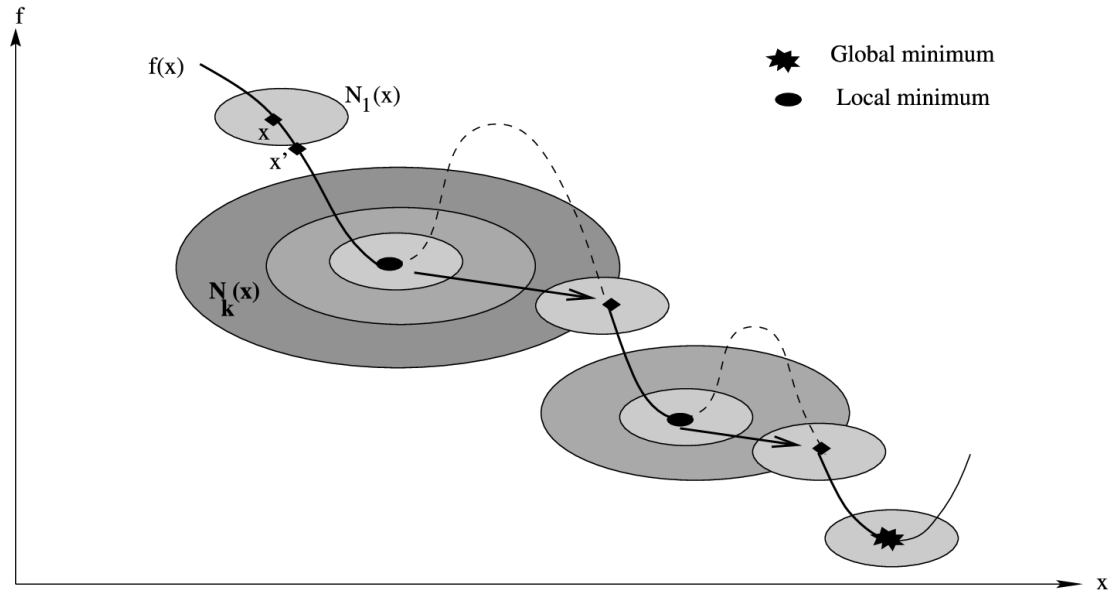
Αλγόριθμος 5 Βήματα του αλγορίθμου RVNS

```
function RVNS(x, k'_max, t'_max)
    repeat
        k ← 1
        repeat
            x' ← Shake(x,k)
            NeighbourhoodChange(x,x',k)
        until k = k_max
        t ← CpuTime()
    until t > t_max
end function
```

αλγόριθμο 6 (Hansen, Mladenović, and Moreno Pérez 2010) παρουσιάζονται τα βήματα του αλγορίθμου. Συχνά οι διαδοχικές γειτονιές N_k είναι εμφωλευμένες. Παρατηρούμε ότι το σημείο x δημιουργείται τυχαία στο βήμα 4 ώστε να αποφευχθεί η συνεχόμενη επανάληψη, το οποίο μπορεί να συμβεί αν εφαρμοστεί ένας ντετερμινιστικός κανόνας. Στη γραμμή 5 συχνά εφαρμόζεται ο ευρετικός αλγόριθμος πρώτης βελτίωσης (αλγόριθμος 2). Ωστόσο, μπορεί να αντικατασταθεί από τον ευρετικό αλγόριθμο καλύτερης βελτίωσης (αλγόριθμος 1). Τα βήματα του BVNS δίνονται στο σχήμα 1.

Αλγόριθμος 6 Βήματα του αλγορίθμου BVNS

```
function VNS(x, k_max, t_max)
    repeat
        k ← 1
        repeat
            x' ← Shake(x,k)                          ▷ /* Ανακάτεμα */
            x'' ← FirstImprovement(x')               ▷ /* Τοπική αναζήτηση */
            NeighbourhoodChange(x,x'',k)             ▷ /* Αλλαγή γειτονιάς */
        until k = k_max
        t ← CpuTime()
    until t > t_max
end function
```



Σχήμα 1: Βασική Αναζήτηση Μεταβλητής Γειτονιάς (Hansen, Mladenović, and Moreno Pérez 2010)

Γενική Αναζήτηση Μεταβλητής Γειτονιάς (GVNS)

Στον αλγόριθμο 6 και ειδικότερα στη γραμμή 5 αντί για τους ευρετικούς αλγορίθμους πρώτης ή καλύτερης βελτίωσης μπορούμε να κάνουμε την τοπική αναζήτηση με τον αλγόριθμο κατάβασης μεταβλητής γειτονιάς (VND) - (αλγόριθμος 4). Η γενική αναζήτηση μεταβλητής γειτονιάς (General VNS - GVNS) βρίσκει εφαρμογή σε πολλά προβλήματα (Hansen, Mladenović, and Moreno Pérez 2010) όπως το πρόβλημα φυλογένεσης, το πρόβλημα Weber, το πρόβλημα Prize-Collecting Steiner Tree σε γράφους, τα προβλήματα ακραίων (extremal) γράφων, το πρόβλημα ομαδοποίησης με βάση το ελάχιστο του αθροίσματος των τετραγώνων (minimum sum-of-squares clustering) και το πρόβλημα των ελάχιστων γεννητικών δέντρων (minimum spanning tree). Τα βήματα του αλγορίθμου παρουσιάζονται στον αλγόριθμο 7.

Αλγόριθμος 7 Βήματα του αλγορίθμου GVNS

function GVNS(x, k'_{max}, k_{max}, t_{max})
  repeat
    $k \leftarrow 1$
    repeat
      $x' \leftarrow Shake(x,k)$
      $x'' \leftarrow VND(x',k'_{max})$
      $NeighbourhoodChange(x,x'',k)$
    until $k = k_{max}$
    $t \leftarrow CpuTime()$
  until $t > t_{max}$
end function

Ασύμμετρη Αναζήτηση Μεταβλητής Γειτονιάς (SVNS)

Η μέθοδος ασύμμετρης αναζήτησης μεταβλητής γειτονιάς (Skewed VNS - SVNS) (Hansen, Jaumard, Mladenović, and Parreira 2000) αντιμετωπίζει το πρόβλημα απομακρυνόμενη από την τρέχουσα λύση. Πράγματι, όταν βρεθεί η καλύτερη λύση σε μια μεγάλη περιοχή, είναι απαραίτητο να εξακολουθήσει ώστε να βρει μια βελτιωμένη λύση. Οι λύσεις που ανακτώνται τυχαία από απομακρυσμένες γειτονιές μπορεί να διαφέρουν σημαντικά από την τρέχουσα λύση και το VNS μπορεί να υποβαθμιστεί, σε ένα βαθμό, σε πολυεναρκτήρια (Multistart) ευρετική μέθοδο, στην οποία η κατάβαση γίνεται επαναληπτικά από τυχαία παραγόμενες λύσεις και είναι γνωστή ως μια όχι πολύ αποδοτική μέθοδο. Επομένως, πρέπει να εφαρμοστεί κάποιο αντιστάθμισμα για την απόσταση από την τρέχουσα λύση. Η μέθοδος Skewed VNS προτείνεται για αυτό το σκοπό και τα βήματά της παρουσιάζονται στους αλγορίθμους 8 και 9 όπου η συνάρτηση $KeepBest(x,x')$ απλά κρατά την καλύτερη τιμή από τα x και x':

```
if f(x') < f(x) then
    x ← x'
end if
```

Αλγόριθμος 8 Βήματα της συνάρτησης αλλαγής γειτονιάς για το SVNS

```
function NeighbourhoodChangeS(x, x'', k, a)
    if f(x'') − ar(x,x'') < f(x) then
        x ← x''
        k ← 1
    else
        k ← k + 1
    end if
end function
```

Αλγόριθμος 9 Βήματα του αλγορίθμου SVNS

```
function SVNS(x, k_max, t_max, a)
    repeat
        k ← 1
        x_best ← x
        repeat
            x' ← Shake(x,k)
            x'' ← FirstImprovement(x')
            KeepBest(x_best,x)
            NeighbourhoodChangeS(x,x'',k,a)
        until k = k_max
        x ← x_best
        t ← CpuTime()
    until t > t_max
end function
```

Η μέθοδος SVNS χρησιμοποιεί τη συνάρτηση $r(x,x'')$ ώστε να μετρά την απόσταση μεταξύ της τρέχουσας λύσης x και του τοπικού βέλτιστου που βρέθηκε x''. Η παράμετρος a πρέπει να επιλεγεί ούτως ώστε να δέχεται την εξερεύνηση σε περιοχές πολύ μακριά από το x όταν

το $f''(x)$ είναι μεγαλύτερο από το $f(x)$ αλλά όχι πολύ μεγαλύτερο. Στην αντίθετη περίπτωση πάντα θα έμενε x. Η τιμή του a επιλέγεται πειραματικά. Επιπροσθέτως, για να αποφευχθούν συχνές μετακινήσεις από το x σε μια κοντινή λύση, όταν το $r(x, x'')$ είναι μικρό στο a μπορούμε να βάλουμε μια μεγάλη τιμή.

ΚΕΦΑΛΑΙΟ 4

Προβλήματα διαχείρισης αποθεμάτων

Εισαγωγή

Ένα από τα πιο σημαντικά θέματα που αντιμετωπίζει μια επιχείρηση είναι η πολιτική ανεφοδιασμού διαφορετικών αγαθών που συμμετέχουν στην εφοδιαστική αλυσίδα. Το πρόβλημα αυτό γίνεται ακόμα πιο σύνθετο όταν η ζήτηση για καθένα από τα K προϊόντα μεταβάλλεται μέσα σε T περιόδους. Λόγω της αυξανόμενης προσοχής που δείχνουν οι βιομηχανικές επιχειρήσεων για τη βιωσιμότητά τους, τα προβλήματα προγραμματισμού παραγωγής (production planning) και διαχείρισης αποθεμάτων (inventory control) μελετώνται στο πεδίο των συστημάτων εφοδιαστικής αλυσίδας κλειστού βρόχου (closed-loop supply chain). Ειδικότερα, τις τελευταίες δύο δεκαετίες, οι τομείς της **αντίστροφης εφοδιαστικής** (reverse logistics), της **εφοδιαστικής αλυσίδας κλειστού βρόχου** (closed-loop supply chain) καθώς και τα προβλήματα **διαχείρισης αποθεμάτων** (inventory control) και **δυναμικού μεγέθους παρτίδας** (dynamic lot sizing) έχουν κερδίσει τόσο το επιχειρηματικό όσο και το ερευνητικό ενδιαφέρον (Sifaleras and Konstantaras 2015a, Sifaleras and Konstantaras 2015b, Sifaleras, Konstantaras, and Mladenović 2015).

Μια εφοδιαστική αλυσίδα κλειστού βρόχου (closed-loop supply chain) επικεντρώνεται στα επιστρεφόμενα χρησιμοποιημένα προϊόντα από τους πελάτες και να ανακτούν την αξία τους, ανακατασκευάζοντας ολόκληρα τα προϊόντα ή/και μερικά από τα μέρη που τα αποτελούν. Τα ανακατασκευασμένα προϊόντα συχνά έχουν την ίδια ποιότητα με τα καινούργια προϊόντα και αλλά έχουν μικρότερο κόστος κάτι που βοηθά στην ικανοποίηση της ζήτησης. Το πρόβλημα του δυναμικού μεγέθους παρτίδας (dynamic lot sizing), π.χ. ο προγραμματισμός παραγγελιών κατασκευής σε συγκεκριμένες χρονικές περιόδους στις οποίες η ζήτηση είναι γνωστή και δυναμική, είναι ένα από τα θεμελιώδη όσο και διεξοδικά μελετημένα μοντέλα της επιχειρησιακής έρευνας στον έλεγχο παραγωγής και αποθεμάτων.

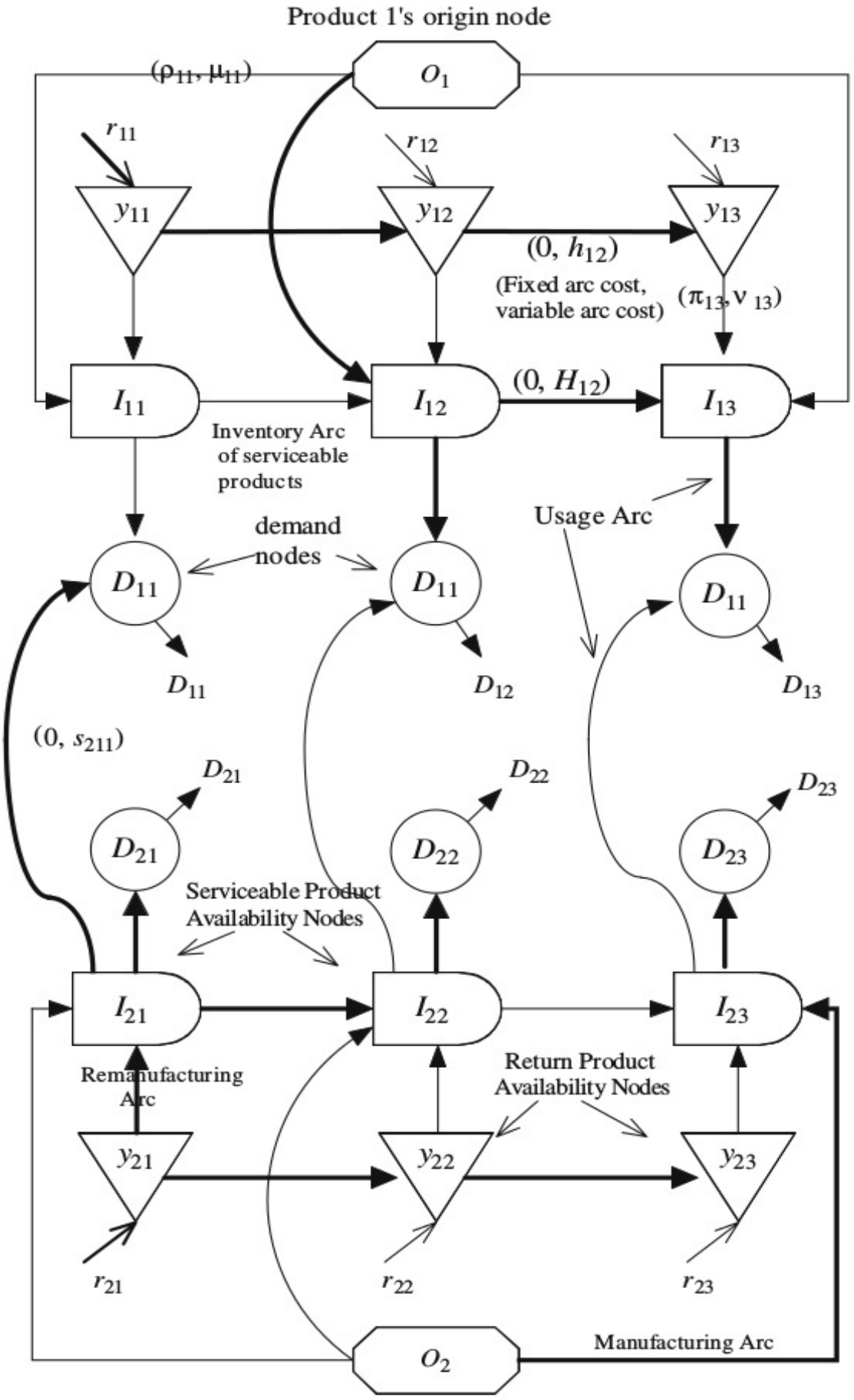
Η αντίστροφη εφοδιαστική (reverse logistics) αντιπροσωπεύει όλες τις λειτουργίες που σχε-

τίζονται με προϊόντα και υλικά. Είναι μια διαδικασία σχεδιασμού, υλοποίησης και ελέγχου της αποδοτικής, από οικονομικής άποψης, ροής των ακατέργαστων υλικών, των τρεχόντων αποθεμάτων, των έτοιμων προϊόντων καθώς και σχετικών πληροφοριών από την αρχή της διαδικασίας μέχρι το σημείο της κατανάλωσης. Η επαναχρησιμοποίηση προϊόντων ή οι επιστροφές υλικών έχουν κερδίσει την σημαντική προσοχή τόσο στις βιομηχανικές επιχειρήσεις όσο και στην ερευνητική κοινότητα για οικονομικούς, περιβαλλοντικούς και νομικούς λόγους. Η εξισορρόπηση της οικονομικής ανάπτυξης με την προστασία του περιβάλλοντος είναι μία καίρια πρόκληση ώστε να είναι βιώσιμες οι κατασκευαστικές εταιρίες. Η κατασκευή προϊόντων με παραδοσιακό τρόπο είναι πλέον μη βιώσιμη λόγω των σημαντικών αρνητικών περιβαλλοντικών επιπτώσεων που επιφέρει. Η ανακατασκευή μπορεί να βοηθήσει τις επιχειρήσεις να παραμείνουν βιώσιμες μέσω της μείωσης της κατανάλωσης φυσικών πόρων. Αυτό έχει ως συνέπεια την ελάττωση των εξόδων, αφού δεν καταναλώνονται τόσοι φυσικοί πόροι. Η ανακατασκευή μπορεί επίσης να βοηθήσει ώστε να μειωθούν οι περιβαλλοντικές επιπτώσεις. Έτσι, μειώνονται τα σκουπίδια στις χωματερές και ανακυκλώνονται ενέργεια και πόροι τα οποία ήδη έχουν καταναλωθεί στην αρχική κατασκευή των προϊόντων. Εκτός από τα περιβαλλοντικά οφέλη η ανακατασκευή προσφέρει οικονομικά κίνητρα στις εταιρίες μέσω της πώλησης των ανακατασκευασμένων προϊόντων και παρατείνοντας τον κύκλο ζωής τους. Η ανακατασκευή μετασχηματίζει τα χρησιμοποιημένα προϊόντα σε παρόμοια των καινούργιων. Μετά την αποσυναρμολόγηση, την ταξινόμηση και το καθάρισμα, τα μέρη που απαρτίζουν τα προϊόντα ελέγχονται εξονυχιστικά και τα προβληματικά μέρη επιδιορθώνονται ή, αν δεν είναι δυνατή η επιδιόρθωσή τους, αντικαθίστανται με νέα. Αυτές οι λειτουργίες επιτρέπουν να ανακτηθεί σημαντική αξία που είναι ενσωματωμένη στο χρησιμοποιημένο προϊόν. Το σημαντικό με την ανακατασκευή είναι ότι επιτρέπει στους κατασκευαστές να είναι σύννομοι, σε ότι αφορά τους ρύπους, ενώ μπορούν να διατηρούν υψηλή παραγωγικότητα για προϊόντα υψηλής ποιότητας, χαμηλού κόστους με λιγότερα απόβλητα όπως και λιγότερη κατανάλωση ενέργειας και ακατέργαστων υλικών (Sifaleras, Konstantaras, and Mladenović 2015).

Προβλήματα στον τομέα της εφοδιαστικής αλυσίδας κλειστού βρόχου

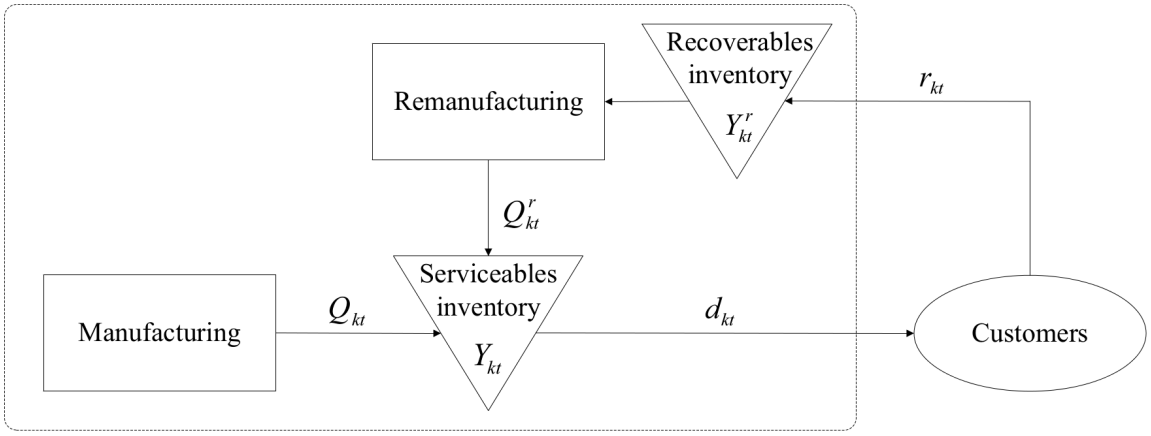
Οι κατασκευαστές έχουν ήδη ξεκινήσει να ενσωματώνουν τις εγκαταστάσεις ανακατασκευής μέσα στις εγκαταστάσεις παραγωγής. Τα ποσοτικά μοντέλα στον έλεγχο αποθεμάτων και τον προγραμματισμό παραγωγής για την εφοδιαστική αλυσίδα κλειστού βρόχου (closed-loop supply chain) έχουν αυξηθεί σημαντικά τις τελευταίες δυο δεκαετίες. Συνεπώς, τα προβλή-

ματα σε αυτή την κατηγορία συνήθως αναφέρονται ως μοντέλα μεγέθους δυναμικής παρτίδας πολλών προϊόντων (multi-product dynamic lot-sizing model) (Sifaleras and Konstantaras 2015b). Το **πρόβλημα δυναμικού μεγέθους παρτίδας πολλών προϊόντων με δραστηριότητες ανακατασκευής** (Multi-product Dynamic Lot Sizing Problem with Remanufacturing activities - MDLSRP) είναι ένα σημαντικό πρόβλημα που εμφανίζεται τα τελευταία χρόνια στον τομέα της εφοδιαστικής αλυσίδας κλειστού βρόχου. Στη βιβλιογραφία υπάρχουν μερικές μελέτες για τα προβλήματα δυναμικού μεγέθους παρτίδας πολλών προϊόντων και ανακατασκευή. Για παράδειγμα, προτάθηκε (Li, Chen, and Cai 2006) για το πρόβλημα δυναμικού μεγέθους παρτίδας πολλών προϊόντων με δυνατότητες ανακατασκευής και αντικατάσταση ζήτησης χωρίς περιορισμό στη χωρητικότητα (Multi-period Multi-product Production-planning with Remanufacturing and Substitutions - MMPRS) μία προσεγγιστική διαδικασία για τον υπολογισμό μια λύσης «κοντά» στη βέλτιστη. Στο σχήμα 2 η γενικευμένη ροή δικτύου ορίζεται μέσω ενός κατευθυνόμενου γράφου G ο οποίος περιέχει τέσσερις τύπους κόμβων: τους κόμβους-ρίζα (Origin Nodes) O_i, τους κόμβους διαθεσιμότητας επιστρεφόμενων προϊόντων (Return Product Availability Nodes) y_{it}, τους κόμβους διαθεσιμότητας επισκευάσιμων προϊόντων (Serviceable Product Availability Nodes) I_{it} και τους κόμβους ζήτησης επισκευάσιμων προϊόντων (Serviceable Product Demand Nodes) D_{it}, $\forall\, i \in\ I$ και $\forall\, t \in\ T$. Επίσης, υπάρχουν πέντε τύποι ακμών που συνδέουν τους κόμβους: ακμή κατασκευής (Manufacturing arc), ακμή ανακατασκευής (Remanufacturing arc), ακμή αποθέματος προϊόντων που έχουν επιστραφεί (Inventory arc of returned products), ακμή αποθέματος επισκευάσιμων προϊόντων (Inventory arc of serviceable products) και ακμή χρήσης (Usage arc). Στο σχήμα 2 παρουσιάζεται ένα τέτοιο δίκτυο για δύο προϊόντα και τρεις περιόδους.



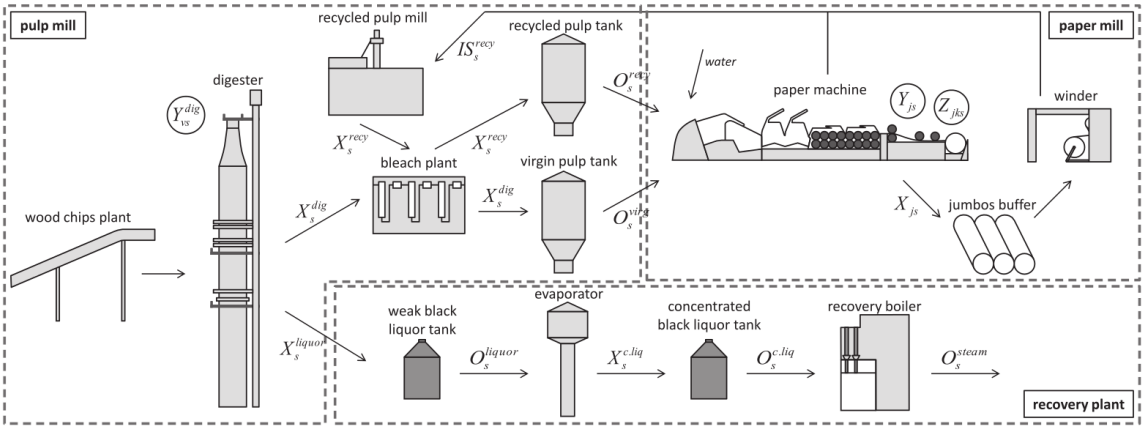
Σχήμα 2: Αναπαράσταση γενικευμένης ροής δικτύου στο πρόβλημα MMPRS (Li, Chen, and Cai 2006)

Προτάθηκε επίσης (Sahling 2013) μία νέα διατύπωση του μοντέλου για το πρόβλημα δυναμικού μεγέθους παρτίδας με περιορισμό στη χωρητικότητα πολλών προϊόντων με επιστροφές προϊόντων και ανακατασκευές (Dynamic Capacitated Lot-Sizing with Product Returns and Remanufacturing - CLSP-RM). Σε αυτό το σχέδιο παραγωγής περιλαμβάνονται οι ποσότητες παραγωγής Q_{kt} και οι ποσότητες ανακατασκευής Q^r_{kt} καθώς και τα επίπεδα αποθεμάτων στο τέλος της περιόδου των προϊόντων προς επισκευή Y_{kt} και των επιστρεφόμενων προϊόντων Y^r_{kt} του προϊόντος k στην περίοδο t. Για κάθε προϊόν k η εξωτερική ζήτηση d_{kt} πρέπει να ικανοποιηθεί στην αντίστοιχη περίοδο t, ενώ η συσσώρευση προϊόντων δεν επιτρέπεται. Τα προς επισκευή προϊόντα του k μπορούν να κρατηθούν ως απόθεμα (Y_{kt}) ώστε να ικανοποιηθεί η ζήτηση σε μεταγενέστερες περιόδους. Τέλος, είναι γνωστές από την αρχή οι ποσότητες των επιστρεφόμενων προϊόντων r_{kt} του προϊόντος k στην περίοδο t .



Σχήμα 3: Μοντέλο για το πρόβλημα δυναμικού μεγέθους παρτίδας με περιορισμό στη χωρητικότητα πολλών προϊόντων με επιστροφές προϊόντων και ανακατασκευή (CLSP-RM) (Schulz 2011, Sahling 2013)

Άλλο ένα πρόβλημα παρουσιάστηκε (Figueira, Santos, and Almada-Lobo 2013) για ένα πρόβλημα εφοδιαστικής αλυσίδας κλειστού βρόχου (closed-loop supply chain) στη βιομηχανία χαρτοπολτού (σχήμα 4).



Σχήμα 4: Ολοκληρωμένο εργοστάσιο χαρτοπολτού (Figueira, Santos, and Almada-Lobo 2013)

Διαμόρφωση του μοντέλου MDLSRP

Το πρόβλημα που αντιμετωπίζουμε στην παρούσα εργασία προέρχεται από το μοντέλο μεγέθους παρτίδας με κόστος εγκατάστασης ανακατασκευής και κόστος εγκατάστασης κατασκευής (Sifaleras and Konstantaras 2015b). Αποτελεί μία τροποποιημένη έκδοση του μοντέλου που παρουσιάστηκε χωρίς περιορισμούς στη χωρητικότητα (Sahling 2013). Πιο συγκεκριμένα, το πρόβλημα επικεντρώνεται στην ικανοποίηση της ζήτησης προϊόντων σε κάθε περίοδο με το, όσο το δυνατόν, μικρότερο συνολικό κόστος. Η ζήτηση δίνεται για ένα πεπερασμένο χρονικό ορίζοντα, θεωρείται ότι δεν είναι στατική και μπορεί να ικανοποιηθεί είτε από τα νέα προϊόντα που κατασκευάζονται, είτε από αυτά που ανακατασκευάζονται, γνωστά και ως «επισκευάσιμα». Επίσης, ο αριθμός των επιστροφών είναι γνωστός για όλες τις περιόδους και θεωρείται μη στατικός. Τα προϊόντα που έχουν επιστραφεί μπορούν να ανακατασκευαστούν πλήρως και να πωληθούν ως καινούργια. Στο πρόβλημα μεγέθους παρτίδας με ξεχωριστά κόστη εγκατάστασης τόσο για την κατασκευή όσο και για την ανακατασκευή, το ζητούμενο είναι ο καθορισμός του αριθμού των κατασκευασμένων και των ανακατασκευασμένων προϊόντων ανά περίοδο, έτσι ώστε να ελαχιστοποιηθεί το άθροισμα, κάτω από διάφορους περιορισμούς, τόσο του κόστους εγκατάστασης των διαδικασιών κατασκευής και ανακατασκευής διαδικασιών όσο και του κόστους αποθήκευσης των επιστρεφόμενων και επισκευάσιμων προϊόντων.

Συμβολισμοί

Παρακάτω αναφέρονται οι συμβολισμοί που χρησιμοποιήθηκαν για τη διαμόρφωση του μοντέλου MDLSRP:

k: Προϊόν, $k = 1,2,...K$

t: Χρονική περίοδος, $t = 1,2,...,T$

$D(k,t)$: Ζήτηση του προϊόντος k στη χρονική περίοδο t

$R(k,t)$: Αριθμός τεμαχίων του προϊόντος k που επεστράφησαν στην περίοδο t και τα οποία μπορούν να ανακατασκευαστούν εξ ολοκλήρου και να πωληθούν ως καινούργια

$h_M(k)$: Κόστος αποθήκευσης των επισκευάσιμων τεμαχίων του προϊόντος k ανά μονάδα χρόνου

$h_R(k)$: Κόστος αποθήκευσης των τεμαχίων που επεστράφησαν από το προϊόν k ανά μονάδα χρόνου

$z_M(k,t)$: Μεταβλητή απόφασης, δυαδική, που δηλώνει την έναρξη μιας παρτίδας κατασκευής του προϊόντος k στην περίοδο t

$z_R(k,t)$: Μεταβλητή απόφασης, δυαδική, που δηλώνει την έναρξη μιας παρτίδας ανακατασκευής του προϊόντος k στην περίοδο t

$x_M(k,t)$: Αριθμός κατασκευασμένων τεμαχίων του προϊόντος k στην περίοδο t

$x_R(k,t)$: Αριθμός τελικά ανακατασκευασμένων τεμαχίων του προϊόντος k στην περίοδο t

$k_M(k)$: Κόστος εγκατάστασης κατασκευής του προϊόντος k

$k_R(k)$: Κόστος εγκατάστασης ανακατασκευής του προϊόντος k

$p_M(k)$: Κόστος παραγωγής του προϊόντος k ανά τεμάχιο

$p_R(k)$: Κόστος ανακατασκευής του προϊόντος k ανά τεμάχιο

$y_M(k,t)$: Επίπεδο αποθεμάτων επισκευάσιμων τεμαχίων του προϊόντος k την περίοδο t

$y_R(k,t)$: Επίπεδο αποθεμάτων ανακατασκευάσιμων τεμαχίων του προϊόντος k την περίοδο t

M: Ένας αρκετά μεγάλος αριθμός

Διαμόρφωση MDLSRP

Η διαμόρφωση του MDLSRP είναι μια γενίκευση της διαμόρφωσης του προβλήματος οικονομικού μεγέθους παρτίδας με δυνατότητες ανακατασκευής (Economic Lot Sizing Problem with remanufacturing options - ELSRP) (Schulz 2011), αφ' ενός αναφερόμενο σε αρκετά προϊόντα αντί του ενός και αφ' ετέρου λαμβάνοντας υπόψη κόστη παραγωγής και ανακατασκευής του προϊόντος k ανά τεμάχιο. Η διαμόρφωση Μεικτού Ακέραιου Γραμμικού Προγραμματισμού (Mixed Integer Linear Programming - MILP) του MDLSRP (Sifaleras and Konstantaras 2015b), που παρατίθεται παρακάτω, διαφέρει από αυτή που προτάθηκε από τον Sahling (Sahling 2013) μόνο στο ότι δε λαμβάνει υπόψη τις διαθέσιμες χωρητικότητες των

πόρων κατασκευής και ανακατασκευής σε μια περίοδο.

$$\begin{aligned} min\ z &= \sum_{k=1}^{K}\sum_{t=1}^{T}(h_R(k)y_R(k,t)+h_M(k)y_M(k,t)) \\ &+ \sum_{k=1}^{K}\sum_{t=1}^{T}(p_R(k)x_R(k,t)+p_M(k)x_M(k,t)) \end{aligned} \tag{1}$$

όπου

$$z_R(k,t)=\begin{cases}1 & \text{αν } x_R(k,t)>0,\\ 0 & \text{σε άλλη περίπτωση}\end{cases} \qquad z_M(k,t)=\begin{cases}1 & \text{αν } x_M(k,t)>0,\\ 0 & \text{σε άλλη περίπτωση}\end{cases} \tag{2}$$

είναι δυαδικές μεταβλητές απόφασης που δηλώνουν την έναρξη, αντίστοιχα, μιας παρτίδας κατασκευής ή ανακατασκευής.

$$\begin{aligned} y_R(k,t) &= y_R(k,t-1)+R(k,t)-x_R(k,t), \\ y_M(k,t) &= y_M(k,t-1)+x_R(k,t)+x_M(k,t)-D(k,t), \\ &\forall t = 1,2,...,T, \forall k = 1,2,...,K \end{aligned} \tag{3}$$

$$\begin{aligned} y_M(k,t) &\geq \sum_{s=t+1}^{t+p}(D(k,s)-M(k,s)(z_M(k,s)+z_R(k,s))), \\ y_R(k,t) &\geq \sum_{s=t-p}^{t}(R(k,s)-M(k,s)z_R(k,s)), \\ &\forall k, \forall t = 1,2,...,T-1, \forall p = 1,2,...,T-t \end{aligned} \tag{4}$$

$$\begin{aligned} &\forall k, \forall t = 2,...,T, \forall p = 1,...,t-1, \\ x_R(k,t) &\leq Mz_R(k,t),\ x_M(k,t) \leq Mz_M(k,t), \\ &\forall t = 1,2,...,T, \forall k = 1,2,...,K \end{aligned} \tag{5}$$

$$\begin{aligned} &y_R(k,t), y_M(k,t), x_R(k,t), x_M(k,t) \geq 0, z_R(k,t), z_M(k,t) \in \{0,1\}, \\ &y_R(k,0) = y_M(k,0) = 0,\ \forall\, t = 1,2,...,T,\ \forall\, k = 1,2,...,K \end{aligned} \tag{6}$$

Οι περιορισμοί που ορίζονται στην εξίσωση 3 είναι οι εξισώσεις ισορροπίας αποθεμάτων με τις οποίες υπολογίζονται τόσο το απόθεμα των επιστρεφόμενων προϊόντων όσο και το απόθεμα των επισκευάσιμων προϊόντων. Οι ανισότητες στην εξίσωση 4 έχουν προσαρμοστεί για το

MDLSRP (Helmrich, Jans, van den Heuvel, and Wagelmans 2014). Οι σχέσεις στην εξίσωση 5 διασφαλίζουν ότι υπάρχει πάντα κάποιο σταθερό κόστος εγκατάστασης είτε όταν έχουμε ανακατασκευή, είτε όταν έχουμε κατασκευή. Τέλος, στην εξίσωση 6 οι σχέσεις διασφαλίζουν ότι αρχικά δεν υπάρχουν αποθέματα, αρχικοποιούν τις μεταβλητές ενδείξεων και προλαμβάνουν τυχόν αρνητικές τιμές στις κατασκευές, τις ανακατασκευές και το απόθεμα.

Μεθευρετικοί αλγόριθμοι επίλυσης προβλημάτων διαχείρισης αποθεμάτων

Στη βιβλιογραφία έχουν προταθεί διάφορες μεθευρετικές μέθοδοι επίλυσης (Sifaleras and Konstantaras 2015b, Sifaleras, Konstantaras, and Mladenović 2015) για προβλήματα διαχείρισης αποθεμάτων. Πιο σημαντικές υλοποιήσεις, εκτός από την αναζήτηση μεταβλητής γειτονιάς (Variable Neighborhood Search - VNS), είναι:

- η κατάβαση μεταβλητής γειτονιάς (Variable Neighborhood Descent - VND),
- η απαγορευμένη έρευνα (Tabu Search - TS),
- ο αλγόριθμος δυναμικού προγραμματισμού (Dynamic Programming algorithm),
- η ευρετική μέθοδος Silver-Meal (SM),
- το ελάχιστο κόστος μονάδας (Least Unit Cost),
- η εξισορρόπηση στοιχείων κόστους (Part Period Balancing - PPB),
- το μοντέλο Μεικτού Ακέραιου Γραμμικού Προγραμματισμού (Mixed Integer Linear Programming - MILP) για την εύρεση ακριβούς (exact) λύσης και
- η βελτιστοποίηση σμήνους σωματιδίων (Particle Swarm Optimization - PSO).

Παραλλαγές του αλγορίθμου (VNS) για την επίλυση προβλημάτων διαχείρισης αποθεμάτων

Τελευταία, στη βιβλιογραφία έχει προταθεί ο αλγόριθμος (VNS) και παραλλαγές του για την επίλυση προβλημάτων διαχείρισης αποθεμάτων. Πιο συγκεκριμένα, προτάθηκε ο αλγόριθμος GVNS για το πρόβλημα MDLSRP (Sifaleras and Konstantaras 2015a) με πολύ καλά αποτελέσματα, αφού, με χρονικό όριο τη μία ώρα, ο αλγόριθμος VNS ξεπερνά σε απόδοση τον λύτη Gurobi δίνοντας ποιοτικές λύσεις, σε μεγάλα προβλήματα, σε λίγο χρόνο.

Επίσης, προτάθηκε (Sifaleras and Konstantaras 2015b) ο αλγόριθμος (VND) επίσης για το πρόβλημα MDLSRP με εξίσου καλά αποτελέσματα.

Τέλος, για το πρόβλημα οικονομικού μεγέθους παρτίδας με δυνατότητες ανακατασκευής (Economic Lot Sizing Problem with remanufacturing options - ELSRP) έχουν προταθεί (Sifaleras, Konstantaras, and Mladenović 2015) ο αλγόριθμος ταξινομημένης γενικής αναζήτησης μεταβλητής γειτονιάς (Ordered GVNS - OGVNS) και ο αλγόριθμος τυχαίας γενικής αναζήτησης μεταβλητής γειτονιάς (Random GVNS - RGVNS).

ΚΕΦΑΛΑΙΟ 5

Παραλληλοποίηση μεθευρετικών αλγορίθμων

Εισαγωγή

Τα προβλήματα βελτιστοποίησης είναι, στην πράξη, συχνά NP-Hard (Non-deterministic Polynomial-time Hard), σύνθετα και καταναλώνουν πολύ χρόνο στην ΚΜΕ (CPU). Παραδοσιακά, χρησιμοποιούνται δύο κύριες προσεγγίσεις για να αντιμετωπίσουν αυτά τα προβλήματα: ακριβείς (exact) μέθοδοι και μεθευρετικές μέθοδοι (Alba, Talbi, Luque, and Melab 2005). Οι ακριβείς μέθοδοι επιτρέπουν τη εύρεση λύσεων με ακρίβεια αλλά δεν είναι πρακτικές αφού έχουν τεράστιο χρόνο εκτέλεσης. Αντίθετα, οι μεθευρετικές μέθοδοι προσφέρουν μη βέλτιστες λύσεις σε λογικό χρόνο.

Οι μεθευρετικές μέθοδοι αναγνωρίζονται ως βασικά εργαλεία στην αντιμετώπιση δύσκολων προβλημάτων σε πολλά και ποικίλα πεδία (Crainic and Toulouse 2010) και συχνά αποτελούν τη μόνη πρακτική προσέγγιση για την επίλυση σύνθετων προβλημάτων σε ρεαλιστικό χρόνο. Όμως, ακόμα και με τη χρήση των μεθευρετικών μεθόδων τα χρονικά όρια εύκολα ξεπερνιούνται λόγω των αυξημένων υπολογιστικών αναγκών είτε στην έρευνα είτε στις επιχειρήσεις. Οι ευρετικές μέθοδοι (heuristics) δεν εξασφαλίζουν, γενικά, βέλτιστες λύσεις. Επιπροσθέτως, η απόδοση συχνά εξαρτάται από τις ρυθμίσεις του συγκεκριμένου προβλήματος. Οι μεθευρετικές μέθοδοι κατηγοριοποιούνται σε: **μεθευρετικές μεθόδους τοπικής αναζήτησης** (local search metaheuristics - LSM) ή **μεθόδους βασισμένες σε τροχιά** (Trajectory-based) και **εξελικτικούς αλγορίθμους** (evolutionary algorithms - EA) ή **αλγορίθμους βασισμένους σε πληθυσμό** (population-based). Η τοπική αναζήτηση ξεκινά με μία αρχική λύση. Σε κάθε βήμα της αναζήτησης η τρέχουσα λύση αντικαθίσταται από άλλη που βρέθηκε στη γειτονιά της, συνήθως την καλύτερη. Πολύ συχνά, οι μεθευρετικές μέθοδοι τοπικής αναζήτησης επιτρέπουν την εύρεση τοπικής βέλτιστης λύσης και για αυτό αποκαλούνται και **μέθοδοι προσανατολισμένες στην εκμετάλλευση** (exploitation-oriented methods). Από την άλλη, οι εξελεκτικοί αλγόριθμοι χρησιμοποιούν έναν, τυχαία δημιουργημένο, πληθυσμό λύσεων. Ο αρχικός πλη-

θυσμός βελτιώνεται μέσω μιας εξελεκτικής διαδικασίας που βασίζεται στη φύση. Σε κάθε γενιά της διαδικασίας, ολόκληρος ο πληθυσμός ή ένα μέρος του αντικαθίσταται από νεοδημιουργημένους απογόνους, συνήθως τους καλύτερους. Οι εξελεκτικοί αλγόριθμοι συχνά ονομάζονται **μέθοδοι προσανατολισμένες στην εξερεύνηση** (exploration-oriented methods). Παρότι η χρήση των μεθευρετικών μεθόδων επιτρέπει τη σημαντική μείωση της χρονικής πολυπλοκότητας της διαδικασίας αναζήτησης αυτή η μείωση δεν αρκεί για πραγματικά προβλήματα. Ως εκ τούτου, **ο παραλληλισμός είναι απαραίτητος όχι μόνο για τη μείωση του χρόνου επίλυσης αλλά και για τη βελτίωση της ποιότητας των προσφερόμενων λύσεων**.

Γενικές κατευθύνσεις για την υλοποίηση της παραλληλοποίησης

Για κάθε μία από τις δύο οικογένειες μεθευρετικών μεθόδων, στη βιβλιογραφία έχουν προταθεί διαφορετικά μοντέλα παραλληλισμού το κάθε ένα από τα οποία αποτυπώνει μια εναλλακτική προσέγγιση για το χειρισμό και αξιοποίηση του παραλληλισμού. Δυστυχώς, όμως, **ο παραλληλισμός** λειτουργιών και δεδομένων **δεν είναι αυτονόητος** στις μεθευρετικές μεθόδους. Για παράδειγμα, στο tabu search υπάρχουν ισχυρές αλληλεξαρτήσεις δεδομένων ανάμεσα στις επαναλήψεις, στους γενετικούς αλγορίθμους το πέρασμα από μια γενιά στην επόμενη πρέπει να γίνει σειριακά, ενώ στο simulated annealing η τρέχουσα λύση μπορεί να ανανεωθεί μόνο σειριακά. Ωστόσο, κάποια βήματα των αλγορίθμων μπορούν να επιτρέψουν παραλληλισμό λειτουργιών ή δεδομένων. Ένας μεθευρετικός αλγόριθμος που ξεκινά από διαφορετικές αρχικές λύσεις, σχεδόν σίγουρα, θα εξερευνήσει διαφορετικές περιοχές του χώρου λύσεων και θα επιστρέψει διαφορετικές λύσεις. Εξερευνώνται οι διαφορετικές περιοχές του χώρου λύσεων ώστε να αποτελέσουν βάση για παραλληλισμό, ωστόσο η παραλληλοποίηση δεν επιστρέφει την ίδια λύση με τη σειριακή έκδοση. Για αυτό το λόγο έχουν εφαρμοστεί κριτήρια αξιολόγησης βασισμένα στην ποιότητα της λύσης πριν να ελεγχθεί η επιτάχυνση του αλγορίθμου από την παραλληλοποίηση. Μια κατηγοριοποίηση των στρατηγικών παραλληλοποίησης που εφαρμόζονται σε μεθευρετικές μεθόδους είναι (Crainic and Toulouse 2003):

- **Τύπος 1**: Αυτή η στρατηγική, που ονομάζεται και παραλληλισμός χαμηλού επιπέδου, προσανατολίζεται αποκλειστικά στην επιτάχυνση των υπολογισμών χωρίς καμία προσπάθεια να εξερευνήσει καλύτερα το χώρο λύσεων ή να βρει καλύτερης ποιότητας λύσεις.

- **Τύπος 2**: Αυτή η προσέγγιση μειώνει το μέγεθος του χώρου αναζήτησης αλλά θα πρέπει να επαναλαμβάνεται ώστε να εξερευνηθεί όλος ο χώρος λύσεων. Προφανώς, το σύνολο των λύσεων που έχει επισκεφθεί ο αλγόριθμος χρησιμοποιώντας αυτή τη στρατηγική

είναι διαφορετικό από το σύνολο των λύσεων της σειριακής υλοποίησης. Γενικά, σε αυτή τη στρατηγική εφαρμόζεται ένα είδος πλαισίου master-slave όπου η διαδικασία master χωρίζει τις μεταβλητές απόφασης στατικά ή δυναμικά ενώ οι διαδικασίες slave εξερευνούν, ανεξάρτητα και ταυτόχρονα, το χώρο που τους έχει ανατεθεί.

- **Τύπος 3**: Ο παραλληλισμός επιτυγχάνεται από πολλές ταυτόχρονες αναζητήσεις στο χώρο λύσεων. Κάθε ταυτόχρονη αναζήτηση είναι πραγματικά ανεξάρτητη: μπορεί να εκτελεί τη δική της ή την κοινή ευρετική μέθοδο, μπορεί να ξεκινά από την ίδια ή διαφορετική αρχική λύση και μπορεί να επικοινωνεί με τις άλλες κατά τη διάρκεια της αναζήτησης (συνεργατικές πολυνηματικές στρατηγικές - co-operative multi-thread strategies) ή μόνο στο τέλος για να προσδιορίσουν την καλύτερη λύση συνολικά (ανεξάρτητες μέθοδοι αναζήτησης - independent search methods). Η επικοινωνία μεταξύ τους μπορεί να είναι σύγχρονη ή ασύγχρονη και μπορεί εκτελείται είτε ανάλογα από το γεγονός (event-driven), είτε σε προκαθορισμένες στιγμές ή να αποφασίζεται δυναμικά κατά την εκτέλεση. Συχνά ο παραλληλισμός τύπου 3 χρησιμοποιείται για να διεξαχθεί μια πιο αναλυτική αναζήτηση του χώρου λύσεων με καλύτερα αποτελέσματα.

Επίσης, ανάλογα με το βαθμό ανάλυσής τους, όπου ως βαθμός ανάλυσης ορίζεται ο λόγος υπολογισμού προς επικοινωνία (computation to communication rate), οι παράλληλες μεθευρετικές μέθοδοι τοπικής αναζήτησης κατηγοριοποιούνται σε:

- **Αδρομερείς** (Coarse-grained) όταν ο υπολογιστικός φόρτος μιας παράλληλης εργασίας είναι πολύ μεγαλύτερος από την επικοινωνία της
- **Λεπτομερείς** (Fine-grained) όταν ο υπολογιστικός φόρτος μια παράλληλης εργασίας είναι συγκρίσιμος με την επικοινωνία της

Επειδή από πλευράς υλικού (Margaritis 2015) ο υπολογιστικός φόρτος και η επικοινωνία εξαρτώνται από την ισχύ του επεξεργαστή και την ταχύτητα του δικτύου ή διαύλου, η επικοινωνία είναι πολύ πιο δαπανηρή από τον υπολογισμό, ιδιαίτερα στα συστήματα κατανεμημένης μνήμης, με εξαίρεση εξειδικευμένα συστήματα.

Επιπρόσθετα, οι εξελεκτικοί αλγόριθμοι χρησιμοποιούν τις προσεγγίσεις παραλληλοποίησης υπολογισμών και παραλληλοποίησης πληθυσμού. Στο πρώτο μοντέλο οι πράξεις εφαρμόζονται σε κάθε απόγονο παράλληλα ενώ στο άλλο μοντέλο ο πληθυσμός χωρίζεται σε διαφορετικούς «υποπληθυσμούς» (subpopulations) οι οποίοι μπορούν να αντιμετατεθούν ή να εξελιχθούν ξεχωριστά και να ενωθούν αργότερα.

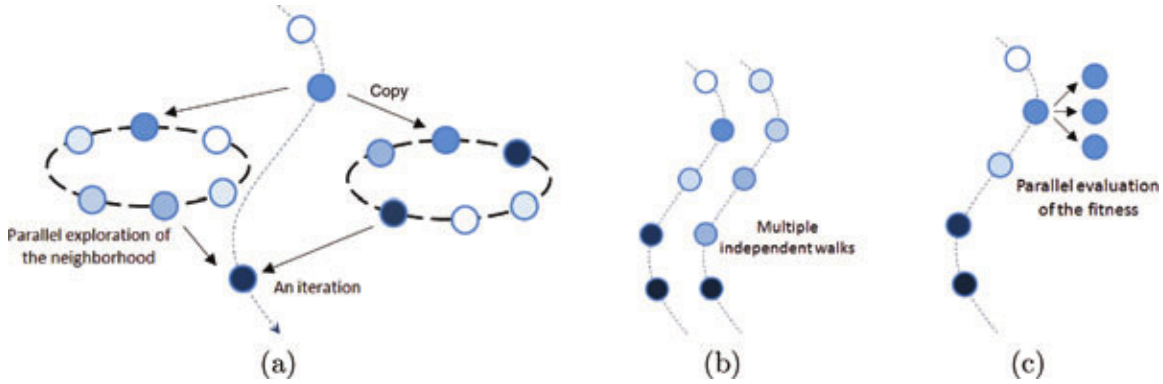
Μεθευρετικοί αλγόριθμοι βασισμένοι σε τροχιά (Trajectory-based metaheuristics)

Οι μεθευρετικές μέθοδοι (Alba, Luque, and Nesmachnow 2013) για την επίλυση προβλημάτων βελτιστοποίησης μπορούν να χαρακτηριστούν ως «περίπατοι διά μέσου των γειτονιών» ("walks through neighborhoods"). Ουσιαστικά, ιχνηλατούν τις τροχιές αναζήτησης στο χώρο λύσεων του προβλήματος. Οι οικογένειες μεθευρετικών μεθόδων που βασίζονται στον έλεγχο μίας λύσης περιλαμβάνουν τα simulated annealing (SA), tabu search (TS), iterated local search (ILS), variable local search (VNS) και greedy randomized adaptive search procedures (GRASP).

Αλγόριθμος 10 Γενικό σχήμα μεθευρετικού αλγορίθμου βασισμένου σε τροχιά

```
Generate(s(0))                                          ▷ /* Αρχική λύση */
t ← 0                                                   ▷ /* Αριθμητικό βήμα */
while not(Termination_Criterion(s(t))) do
    s'(t) ← SelectMove(s(t))                            ▷ /* Εξερεύνηση της γειτονιάς */
    if AcceptMove(s'(t)) then
        s(t) ← ApplyMove(s'(t))
    end if
    t ← t + 1
end while
```

Οι περίπατοι εκτελούνται από επαναληπτικές διαδικασίες που επιτρέπουν τη μετακίνηση από μία λύση σε έναν άλλο χώρο λύσεων (αλγόριθμος 10). Μια τέτοια μεθευρετική μέθοδος εκτελεί τις μετακινήσεις της στη γειτονιά της τρέχουσας λύσης. Οι περίπατοι ξεκινούν από μια λύση που έχει δημιουργηθεί τυχαία ή που έχει αποκτηθεί από άλλο αλγόριθμο βελτιστοποίησης. Σε κάθε επανάληψη, η τρέχουσα λύση αντικαθίσταται από άλλη η οποία επιλέγεται από ένα σύνολο των υποψηφίων γειτόνων της. Η διαδικασία αναζήτησης σταματά όταν ικανοποιηθεί ένα κριτήριο, για παράδειγμα αν φτάσει στο μέγιστο αριθμό μετακινήσεων, αν βρει μια λύση με συγκεκριμένη ποιότητα ή αν «κολλήσει» για κάποιο χρόνο. Για να πετύχουμε υψηλή απόδοση στις μεθόδους βασισμένες σε τροχιά πρέπει να χρησιμοποιήσουμε παραλληλισμό. Έχουν προταθεί πολλά μοντέλα παραλληλοποίησης, από τα οποία τρία χρησιμοποιούνται ευρέως (σχήμα 5):



Σχήμα 5: Τα τρία κλασικά παράλληλα μοντέλα για τις μεθευρετικές μεθόδους που βασίζονται σε τροχιά (trajectory-based metaheuristics) (Alba, Luque, and Nesmachnow 2013)

- **Μοντέλο παράλληλων μετακινήσεων (Parallel moves model)**: Είναι ένα μοντέλο χαμηλού επιπέδου, master-slave το οποίο δε μεταβάλλει τη συμπεριφορά της τεχνικής. Το ίδιο αποτέλεσμα, αλλά σε περισσότερο χρόνο, μπορεί να το υπολογίσει και μία σειριακή αναζήτηση. Στην αρχή κάθε επανάληψης, το master thread διανέμει την τρέχουσα λύση στους απομακρυσμένους κόμβους. Κάθε ένας τους ξεχωριστά διαχειρίζεται την υποψήφια λύση του και τα αποτελέσματα επιστρέφουν στο master thread.

- **Μοντέλο παράλληλης πολλαπλής εκκίνησης (Parallel multistart model)**: Περιλαμβάνει την ταυτόχρονη εκκίνηση αρκετών μεθευρετικών μεθόδων που βασίζονται σε τροχιά ώστε να υπολογίζει καλύτερες και ισχυρότερες λύσεις. Μπορεί να είναι ετερογενές ή ομογενές, ανεξάρτητο ή συνεργατικό, να ξεκινά από την ίδια ή διαφορετική λύση και να ρυθμίζεται με τις ίδιες ή διαφορετικές παραμέτρους.

- **Μοντέλο επιτάχυνσης μετακινήσεων (Move acceleration model)**: Η ποιότητα κάθε μετακίνησης αξιολογείται με έναν παράλληλο τρόπο. Το μοντέλο συγκεκριμένα ενδιαφέρεται για το πότε μπορεί να παραλληλοποιηθεί η συνάρτηση αξιολόγησης επειδή καταναλώνει χρόνο στη ΚΜΕ (CPU) ή/και είναι I/O intensive. Σε εκείνη την περίπτωση, η συνάρτηση μπορεί να θεωρηθεί ως μια συγκέντρωση συγκεκριμένου αριθμού επί μέρους συναρτήσεων που μπορούν να εκτελεστούν παράλληλα.

Μεθευρετικοί αλγόριθμοι βασισμένοι σε πληθυσμό (Population-based metaheuristics)

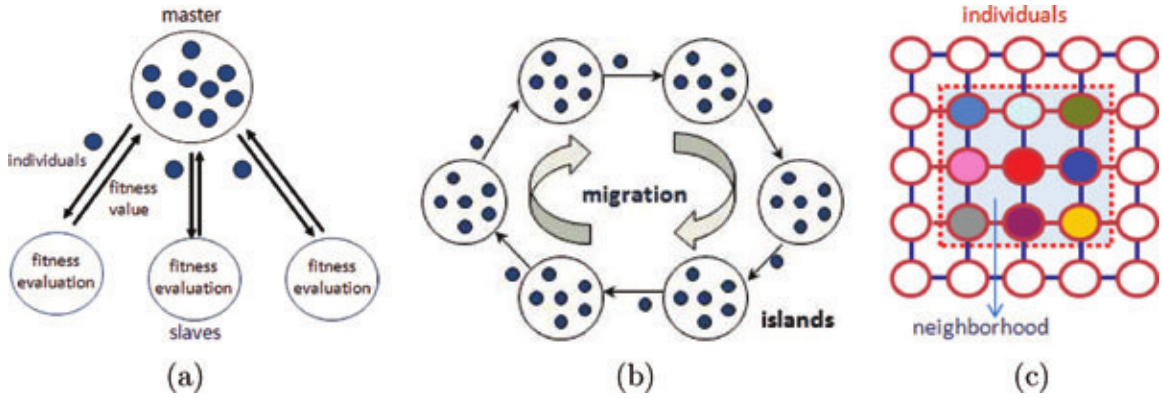
Οι μεθευρετικές μέθοδοι που βασίζονται σε πληθυσμό (Alba, Luque, and Nesmachnow 2013) είναι στοχαστικές τεχνικές αναζήτησης που έχουν εφαρμοστεί με επιτυχία σε πολλές πραγματικές και περίπλοκες εφαρμογές. Ένας τέτοιος αλγόριθμος είναι μια επαναληπτική τεχνική που εφαρμόζει στοχαστικούς τελεστές σε ένα σύνολο ατόμων, τον πληθυσμό (αλγόριθ-

μος 11). Κάθε άτομο στον πληθυσμό είναι η κωδικοποιημένη έκδοση μιας αβέβαιης λύσης. Μια συνάρτηση αξιολόγησης αντιστοιχεί μια τιμή καταλληλότητας (fitness) σε κάθε άτομο υποδεικνύοντας την καταλληλότητά του για το πρόβλημα. Επαναληπτικά, η πιθανή εφαρμογή «τελεστών μεταβολής» (variation operators) σε συγκεκριμένα άτομα οδηγεί τον πληθυσμό σε αβέβαιες λύσεις υψηλότερης ποιότητας. Οι πολύ γνωστές οικογένειες μεθευρετικών μεθόδων που βασίζονται στον έλεγχο ενός πληθυσμού λύσεων περιλαμβάνουν τα genetic algorithms (GAs), ant colony optimization (ACO), particle swarm optimization (PSO), scatter search (SS), differential evolution (DE), evolutionary strategies (ES) και estimation distribution algorithms (EDA).

Αλγόριθμος 11 Ψευδοκώδικας μεθευρετικού αλγορίθμου βασισμένου σε πληθυσμό

```
Generate(P(0))                                   ▷ /* Αρχικός πληθυσμός */
t ← 0                                            ▷ /* Αριθμητικό βήμα */
while not(Termination_Criterion(P(t))) do
    Evaluate(P(t))                               ▷ /* Αξιολόγηση του πληθυσμού */
    P''(t) ← Apply_Variation_Operators(P'(t))    ▷ /* Δημιουργία νέων λύσεων */
    P(t+1) ← Replace(P(t), P''(t))               ▷ /* Δημιουργία επόμενου πληθυσμού */
    t ← t+1
end while
```

Για τα σημαντικά προβλήματα, η εκτέλεση ενός αναπαραγωγικού κύκλου μιας μεθόδου βασισμένης σε πληθυσμό με μεγάλα άτομα και μεγάλους πληθυσμούς συχνά απαιτεί τεράστιους υπολογιστικούς πόρους. Γενικά, η αξιολόγηση της συνάρτησης καταλληλότητας για κάθε άτομο είναι συχνά η πιο κοστοβόρα πράξη αυτού του αλγορίθμου. Πολλές μελέτες έχουν γίνει για την εύρεση αποδοτικότερων τεχνικών είτε με νέους τελεστές, είτε με υβριδικούς αλγορίθμους, είτε με άλλα παράλληλα μοντέλα κλπ. Ο παραλληλισμός προκύπτει ως φυσική εξέλιξη, όταν αναφερόμαστε σε πληθυσμούς, αφού κάθε άτομο του πληθυσμού ανήκει σε αυτόν ως ανεξάρτητη μονάδα. Πράγματι, η απόδοση των αλγορίθμων που βασίζονται σε πληθυσμό συχνά βελτιώνεται όταν εκτελούνται παράλληλα. Δύο στρατηγικές παραλληλισμού ειδικά επικεντρώνονται στους αλγορίθμους που βασίζονται στον πληθυσμό (σχήμα 6):



Σχήμα 6: Τα πιο σημαντικά κλασικά παράλληλα μοντέλα για τις μεθευρετικές μεθόδους που βασίζονται σε πληθυσμό (population-based metaheuristics) (Alba, Luque, and Nesmachnow 2013)

- **Παραλληλισμός υπολογισμών (parallelization of computations)**, στο οποίο οι λειτουργίες που εφαρμόζονται κοινά σε κάθε άτομο εκτελούνται παράλληλα. Στις πρώτες προσπάθειες παραλληλοποίησης των αλγορίθμων αυτών χρησιμοποιήθηκε η μέθοδος "master-slave", γνωστή και ως «καθολικός παραλληλισμός» ("global parallelization") ή ως "farming". Σε αυτή την προσέγγιση, ο κεντρικός επεξεργαστής (master) εκτελεί τις εντολές επιλογής ενώ οι συνδεδεμένοι με αυτόν επεξεργαστές (slaves) εκτελούν τον τελεστή μεταβολής (variation operator) και την συνάρτηση καταλληλότητας (fitness function). Αυτός ο αλγόριθμος έχει την ίδια συμπεριφορά με τον σειριακό, όπως φαίνεται στο σχήμα 6a, ωστόσο η υπολογιστική αποτελεσματικότητά του είναι καλύτερη, ειδικά για χρονοβόρες αντικειμενικές συναρτήσεις (objective functions). Από την άλλη, πολλοί ερευνητές χρησιμοποιούν μια ομάδα επεξεργαστών ώστε να επιταχύνουν την εκτέλεση του σειριακού αλγορίθμου, μόνο και μόνο γιατί οι «ανεξάρτητες εκτελέσεις» ("independent runs") μπορούν να γίνουν πιο γρήγορα με τη χρήση αρκετών επεξεργαστών αντί μόνο ενός. Σε αυτή την περίπτωση, δεν υπάρχει καμία επικοινωνία μεταξύ των ανεξάρτητων εκτελέσεων, κάτι που βελτιώνει την παραγωγικότητα των ερευνών.

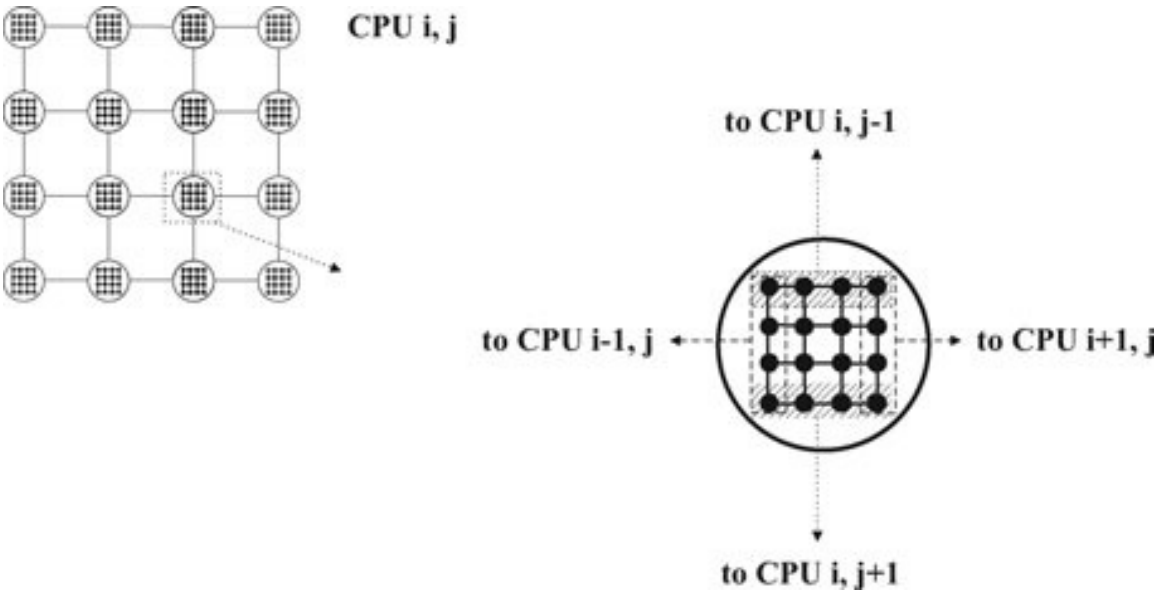
- **Παραλληλισμός πληθυσμού (parallelization of population)**, στο οποίο ο πληθυσμός διαιρείται σε διαφορετικά μέρη τα οποία μπορούν απλά να αλλαχθούν ή να εξελιχθούν ξεχωριστά και έπειτα να ενωθούν. Ανάμεσα στους πιο γνωστούς τύπους δομημένων μεθευρετικών μεθόδων οι παρακάτω είναι πολύ δημοφιλείς:

  - **Κατανεμημένοι ή αδρομερείς (distributed or coarse-grain)** (σχήμα 6b). Σε αυτή την περίπτωση ο πληθυσμός χωρίζεται σε ένα μικρό σύνολο υποπληθυσμών ή «νησιών» (islands) στα οποία εκτελούνται απομονωμένοι σειριακοί αλγόριθμοι. Ανάμεσα σε αυτά τα νησιά γίνονται, όχι τόσο συχνά, ανεξάρτητες ανταλλαγές με

στόχο να διαφοροποιήσουν τους υποπληθυσμούς ώστε να μπορούν να ξεφύγουν από τα τοπικά ελάχιστα. Κατά το σχεδιασμό ενός κατανεμημένου μεθευρετικού αλγορίθμου πρέπει να ληφθούν υπόψη η τοπολογία, οι λογικές συνδέσεις μεταξύ των νησιών, ο ρυθμός μετανάστευσης, ο αριθμός των ατόμων που μεταναστεύουν σε κάθε ανταλλαγή, η περίοδος μετανάστευσης, ο αριθμός των εκτελεσμένων βημάτων σε κάθε υποπληθυσμό μεταξύ δύο διαδοχικών ανταλλαγών και η επιλογή/αντικατάσταση των μεταναστών.

– **Κυτταρικοί ή λεπτομερείς cellular or fine-grain** (σχήμα 6c). Σε αυτή την περίπτωση εισάγεται η έννοια της «γειτονιάς» ("neighborhood") έτσι ώστε ένα άτομο να μπορεί να επικοινωνεί μόνο με τους κοντινούς του γείτονες στο βρόχο αναπαραγωγής. Η επικαλυπτόμενη μικρή γειτονιά του αλγορίθμου βοηθά στην εξερεύνηση του χώρου αναζήτησης επειδή η αργή εξάπλωση των λύσεων στον πληθυσμό προσφέρει ένα είδος εξερεύνησης, ενώ η εκμετάλλευση συμβαίνει μέσα σε κάθε γειτονιά.

Γενικά, το υψηλό επίπεδο παραλληλισμού είναι μια αδρομερής (coarse-grain) εφαρμογή, για παράδειγμα ένα σύνολο νησιών και κάθε νησί εκτελεί διαφορετικό παράλληλο μοντέλο όπως κυτταρικό, master-slave ή ακόμα κατανεμημένη μέθοδο (παράδειγμα στο σχήμα 7).



Σχήμα 7: Παράδειγμα χρήσης κατανεμημένου μοντέλου σε κυτταρικές μεθόδους (Alba, Luque, and Nesmachnow 2013)

Γενικά, ο παραλληλισμός είναι ένας ισχυρός και απαραίτητος τρόπος ώστε να μειωθεί ο υπολογιστικός χρόνος των μεθευρετικών μεθόδων ή/και να βελτιωθεί η ποιότητα των προσφερόμενων λύσεων. Έχουν προταθεί διαφορετικά μοντέλα για την εκμετάλλευση του παραλ-

ληλισμού στις μεθευρετικές μεθόδους για μεγάλη ποικιλία προβλημάτων σε διαφορετικούς τομείς.

Εφαρμογές που επιλύονται με παράλληλους μεθευρετικούς αλγορίθμους

Οι παράλληλοι μεθευρετικοί αλγόριθμοι έχουν αποδειχθεί ότι είναι χρήσιμοι στην πράξη για την επίλυση ενός μεγάλου αριθμού εφαρμογών. Πολλά πραγματικά προβλήματα (real-life) μπορεί να χρειάζονται μέρες ή εβδομάδες υπολογιστικού χρόνου για να επιλυθούν με σειριακό τρόπο. Οι τεχνικές της παραλληλοποίησης συχνά επιτρέπουν τη μείωση του χρόνου εκτέλεσης σε λογικά επίπεδα. Αυτό είναι ένα μεγάλο πλεονέκτημα για τις επιχειρήσεις που επιζητούν επίλυση προβλημάτων για τη λήψη αποφάσεων στο μικρότερο δυνατό χρόνο ώστε να είναι ανταγωνιστικές. Επιπροσθέτως, τα παράλληλα μοντέλα συχνά επιτρέπουν την καλύτερη εξερεύνηση εναλλακτικών λύσεων στο χώρο αναζήτησης. Οι παράλληλοι μεθευρετικοί αλγόριθμοι βρίσκουν εφαρμογή (Alba, Luque, and Nesmachnow 2013) στην επιχειρησιακή έρευνα, στη μηχανική, στις κατασκευές, στις τηλεπικοινωνίες και σε άλλους τομείς. Οι πιο σημαντικοί από αυτούς είναι:

- Αυτοματισμός και ρομποτική (Automation and robotics),
- Βιοπληροφορική (Bioinformatics),
- Μηχανολογικός σχεδιασμός (Engineering design),
- Υδραυλική μηχανική (Hydraulic engineering),
- Επεξεργασία πληροφοριών, κατηγοριοποίηση και εξόρυξη πληροφοριών (Information processing, classification, and data mining),
- Βιομηχανική παραγωγή και εφαρμογές (Manufacturing and industrial applications),
- Δρομολόγηση, εφοδιαστική και οργάνωση οχημάτων (Routing, logistics and vehicle planning),
- Χρονοπρογραμματισμός (Scheduling),
- Μηχανική και ανάπτυξη λογισμικού (Software engineering and software development),
- Τηλεπικοινωνίες (Telecommunications),

- Ενέργεια και βελτιστοποίηση ενεργειακού δικτύου (energy and power network optimization),
- Υγεία και φαρμακευτική (health and medicine),
- Στρατιωτικές και στρατηγικές εφαρμογές (strategic and military applications),
- Οικονομία και οικονομικά (economy and finance),
- Διαχείριση εργατικού δυναμικού (workforce planning) και
- Επεξεργασία εικόνας (image processing)

Τεχνολογίες για παράλληλους μεθευρετικούς αλγορίθμους

Όταν μελετάμε έναν παράλληλο αλγόριθμο είναι σημαντικό να λάβουμε υπόψη σε ποια υπολογιστική πλατφόρμα έχει υλοποιηθεί, αφού η αρχιτεκτονική του υλικού επηρεάζει σημαντικά το χρόνο υπολογισμών, επικοινωνίας, συγχρονισμών και διαμοιρασμού δεδομένων. Μέχρι την προηγούμενη δεκαετία οι κλασικές προτάσεις για εφαρμογή παράλληλων μεθευρετικών αλγορίθμων αφορούσαν παραδοσιακούς υπερυπολογιστές (supercomputers) είτε συστοιχίες σταθμών εργασίας (clusters of workstations). Σήμερα, η τεχνολογική πρόοδος έχει φέρει στην καθημερινότητά μας νέες τεχνολογίες όπως τους πολυπύρηνους επεξεργαστές (multicore processors) και τις γραφικές μονάδες επεξεργασίας (GPUs), προσφέροντας νέες ευκαιρίες για την ανάπτυξη παράλληλων υπολογιστικών τεχνικών ώστε να βελτιωθεί η επίλυση προβλημάτων και να μειωθούν οι χρόνοι υπολογισμών (Margaritis 2015, Alba, Luque, and Nesmachnow 2013).

Υλικό (hardware) για παράλληλο υπολογισμό

Η κατηγοριοποίηση που χρησιμοποιείται περισσότερο για παράλληλες υπολογιστικές πλατφόρμες είναι αυτή του Flynn (Alba, Luque, and Nesmachnow 2013), η οποία ξεχωρίζει τέσσερις κατηγορίες ανάλογα με τον τρόπο που αντιμετωπίζονται οι εντολές και τα δεδομένα (Margaritis 2015, Alba, Luque, and Nesmachnow 2013):

- **Απλή εντολή - Απλό δεδομένο** (single instruction-single data - SISD)
- **Απλή Εντολή - Πολλαπλά Δεδομένα** (single instruction-multiple data - SIMD)
- **Πολλαπλές Εντολές - Απλό Δεδομένο** (multiple instruction-single data - MISD)

- **Πολλαπλές Εντολές - Πολλαπλά Δεδομένα** (multiple instruction-multiple data - MIMD)

Οι παραδοσιακές αρχιτεκτονικές SIMD χρησιμοποιήθηκαν κυρίως τη δεκαετία του 1980 και στις αρχές της δεκαετίας του 1990, ενώ οι αρχιτεκτονικές MIMD χρησιμοποιήθηκαν κατά τη δεκαετία του 1990. Μετά το 2000 εμφανίστηκαν νέες αρχιτεκτονικές. Αυτές μπορούν να ταξινομηθούν σε:

- Προγραμματιζόμενα κυκλώματα, όπως Field Programmable Gate Array - FPGA
- Πολυεπεξεργαστές, όπως πολυπύρηνους υπολογιστές (multicore computers) και γραφικές μονάδες επεξεργασίας (GPUs)
- Κατανεμημένες υπολογιστικές πλατφόρμες, όπως συστοιχίες (clusters) υπολογιστών, peer to peer (P2P), υπολογιστικό πλέγμα (grid) και υπολογιστικό νέφος (cloud).

Λογισμικό (software) για παράλληλο υπολογισμό

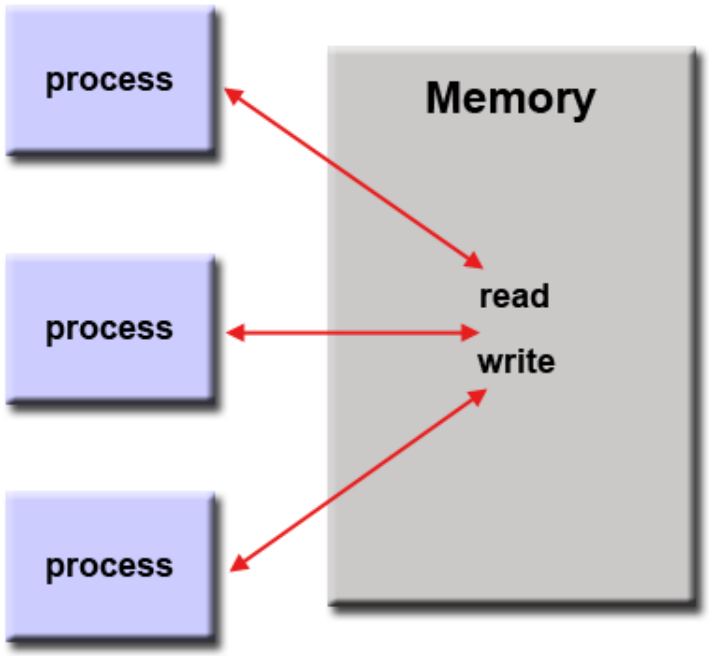
Για την ανάπτυξη παράλληλων μεθευρετικών αλγορίθμων παλιότερα χρησιμοποιήθηκαν κυρίως η C/C++ και η Java. Μερικές φορές χρησιμοποιήθηκε και το Matlab λόγω της μεγάλης χρήσης του στην επιστημονική κοινότητα. Οι παράλληλοι μεθευρετικοί αλγόριθμοι αναπτύσσονταν σχεδιάζοντας ad-hoc εφαρμογές οι οποίες χρησιμοποιούσαν μηχανισμούς επικοινωνίας χαμηλού επιπέδου ανάμεσα στις διεργασίες, όπως τα TCP/IP sockets ή τα pthreads. Αυτή η προσέγγιση χρησιμοποιούνταν συχνά περίπου πριν από 20 χρόνια αλλά σπάνια εφαρμόζεται σήμερα. Και αυτό γιατί αναπτύχθηκαν βιβλιοθήκες για την ανάπτυξη παράλληλων ad-hoc εφαρμογών. Οι πιο κοινά χρησιμοποιούμενες βιβλιοθήκες περιλαμβάνουν τις υλοποιήσεις του προτύπου MPI (MPICH, LAM/MPI) και του προτύπου MPI-2 (MPICH2, OpenMPI, LAM/MPI) για πλατφόρμες κατανεμημένης μνήμης, του προτύπου OpenMP για πολυπύρηνους υπολογιστές, από την έκδοση 4.0 και για (GPUs), των προτύπων CUDA, OpenCL, OpenACC για γραφικές μονάδες επεξεργασίας (GPUs). Για τα περισσότερα πρότυπα η πιο κοινή γλώσσα προγραμματισμού αλλά και αυτή που υποστηρίζεται εγγενώς είναι η C/C++. Όμως, με τη χρήση εμπορικών μεταφραστών (compilers) μπορεί να υποστηριχθεί η Fortran για τα πρότυπα OpenACC, CUDA με τον PGI Compiler της NVIDIA, η Python για το πρότυπο OpenACC με τον Anaconda Compiler και με χρήση βιβλιοθηκών της κοινότητας υποστηρίζεται η Python για το πρότυπο CUDA και η Fortran για το πρότυπο OpenCL. Ενδιαφέρον έχει πως μπορούν να συνδυαστούν τα πρότυπα για πλατφόρμες κατανεμημένης μνήμης με πρότυπα για πολυπύρηνους υπολογιστές και τα πρότυπα για γραφικούς επεξεργαστές ώστε

να μπορούμε να εκμεταλλευτούμε όλες τις μορφές παραλληλοποίησης που μας προσφέρεται από το υλικό, όπως για παράδειγμα MPI με OpenMP και CUDA.

Επειδή οι ad-hoc υλοποιήσεις των παράλληλων μεθευρετικών αλγορίθμων συχνά είναι συνδεδεμένες με το συγκεκριμένο πρόβλημα, δύσκολα επαναχρησιμοποιείται ο υπάρχων κώδικας για την επίλυση άλλου προβλήματος. Έτσι, έχουν αναπτυχθεί γενικά πλαίσια υλοποίησης παράλληλων μεθευρετικών αλγορίθμων όπως MALLBA, ParadisEO, pALS, ECJ, OPT4J, DGPF, PMF, gpuMF. Η μελέτη αυτών των πλαισίων είναι έξω από τους σκοπούς της παρούσας εργασίας.

Αρχιτεκτονική OpenMP

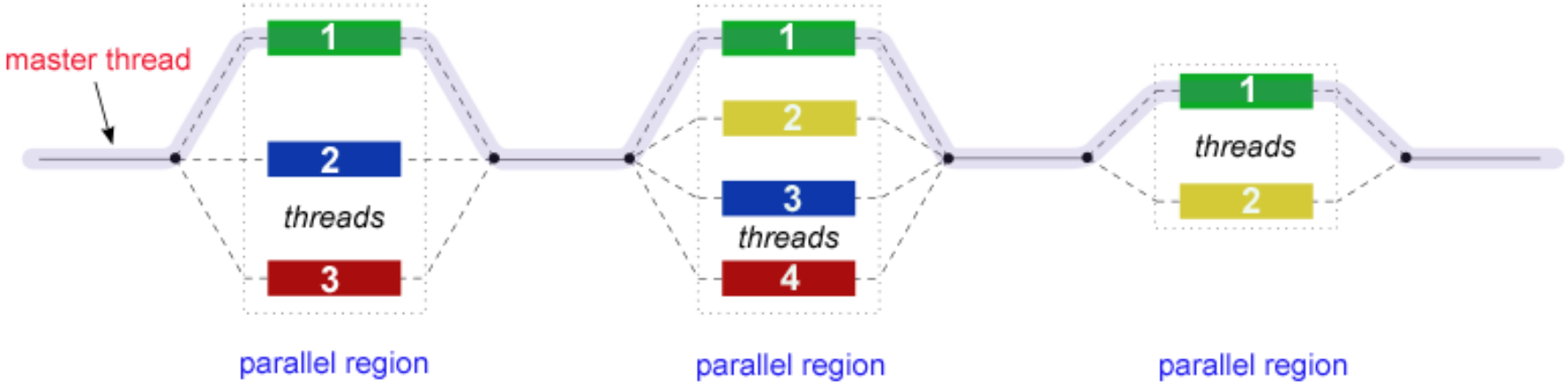
Το OpenMP (Open Multi-Processing) είναι (Margaritis 2015) μία Διεπαφή Προγραμματισμού Εφαρμογών (API - Application Programming Interface) η οποία επιτρέπει την παράλληλη επεξεργασία με χρήση νημάτων (threads) σε συστήματα είτε με πολυπύρηνους επεξεργαστές είτε σε συστήματα πολλαπλών επεξεργαστών. Κοινό χαρακτηριστικό των συστημάτων αυτών είναι ότι υποστηρίζουν την αρχιτεκτονική της διαμοιραζόμενης μνήμης (shared memory architecture), αποτελούνται δηλαδή από μια σειρά διαφορετικών επεξεργαστών ή διαφορετικών πυρήνων μέσα στον ίδιο επεξεργαστή, οι οποίοι μπορούν να επικοινωνούν αλλά και να συγχρονίζονται μέσω μιας κοινής μνήμης. Η βασική μορφή της τεχνικής της διαμοιραζόμενης μνήμης απεικονίζεται στο σχήμα 8.



Σχήμα 8: Η τεχνική της διαμοιραζόμενης μνήμης (Barney 2016a)

Η χρήση αυτού του API επιτρέπει τη συγγραφή παράλληλων εφαρμογών χωρίς την ανά-

γκη για τη διαχείριση των απαιτούμενων νημάτων - έναρξης, συγχρονισμού, επικοινωνίας, τερματισμού. Αποφεύγεται, με αυτό τον τρόπο, μια χρονοβόρα και επίπονη διαδικασία. Με το OpenMP ο χρήστης αρκεί να επισημάνει τα τμήματα της εφαρμογής τα οποία επιθυμεί να παραλληλοποιηθούν με τη χρήση συγκεκριμένων οδηγιών (directives) προς το μεταγλωττιστή. Για τη διαχείριση των νημάτων, το OpenMP χρησιμοποιεί την τεχνική της διακλάδωσης/ένωσης (fork/join), όπως φαίνεται στο σχήμα 9.



Σχήμα 9: Η τεχνική νημάτων fork/join (Barney 2016b)

Στην τεχνική της διακλάδωσης/ένωσης, όταν αρχικά εκτελείται ένα πρόγραμμα, ένα μόνο νήμα παραμένει ενεργό, το νήμα-συντονιστής (master thread). Εάν κατά τη ροή εκτέλεσης του προγράμματος βρεθεί μια παράλληλη περιοχή, τότε το νήμα-συντονιστής δημιουργεί επιπλέον νήματα, συνήθως όσα και οι διαθέσιμοι επεξεργαστές ή/και πυρήνες επεξεργαστών, για όλη τη διάρκεια εκτέλεσης της περιοχής αυτής. Μόλις σταματήσουν να είναι αναγκαία, τα επιπλέον νήματα παύουν να υφίστανται και ενώνονται με το νήμα-συντονιστή. Η τεχνική της διακλάδωσης/ένωσης ξεχωρίζει το μοντέλο της διαμοιραζόμενης μνήμης από το μοντέλο της μεταβίβασης μηνυμάτων που χρησιμοποιεί το MPI, σύμφωνα με το οποίο όλες οι διεργασίες παραμένουν ενεργές καθ' όλη τη διάρκεια εκτέλεσης μιας παράλληλης εφαρμογής, ανεξάρτητα από το εάν απαιτούνται ή όχι. Το OpenMP υποστηρίζει επίσης τόσο τον παραλληλισμό δεδομένων (data parallelism) όσο και τον παραλληλισμό σε επίπεδο συναρτήσεων (function parallelism). Με τον παραλληλισμό δεδομένων τα δεδομένα μοιράζονται στα διαθέσιμα νήματα τα οποία θα τα επεξεργαστούν με τη σειρά ενώ με τον παραλληλισμό σε επίπεδο συναρτήσεων ένα πρόγραμμα χωρίζεται σε τμήματα, κάθε ένα από τα οποία θα εκτελεστεί από διαφορετικό νήμα.

Το OpenMP παρέχει μια σειρά από οδηγίες (directives), οι οποίες μπορούν να χρησιμοποιηθούν για να καθοριστεί ο τρόπος με τον οποίο θα διανεμηθεί ένα παράλληλο κομμάτι κώδικα. Οι οδηγίες αυτές μπορούν να συνδυαστούν με μια σειρά από βασικές γλώσσες προγραμματισμού (Fortran, C, C++) δίνοντάς τους τη δυνατότητα να υποστηρίξουν την παραλληλοποίηση εφαρμογών μέσω της χρήσης της αρχιτεκτονικής της διαμοιραζόμενης μνήμης.

Εκτός από τις οδηγίες προς το μεταγλωττιστή, το OpenMP περιλαμβάνει και ένα μικρό υποσύνολο βιβλιοθηκών (library routines) και μεταβλητών περιβάλλοντος (environment variables) οι οποίες συνήθως χρησιμοποιούνται για την αλλαγή παραμέτρων κατά την εκτέλεση των παράλληλων προγραμμάτων. Οι οδηγίες ξεκινούν πάντα με τον χαρακτήρα # στη C/C++ και με το χαρακτήρα `$` στη Fortran, ώστε να ξεχωρίζουν από τις υπόλοιπες εντολές και συναρτήσεις, ενώ η σύνταξή τους είναι η εξής:

`#pragma omp <rest of pragma>` ή `$omp <rest of pragma>`

Στον πίνακα **5.1** περιέχονται οι βασικές οδηγίες του OpenMP:

Πίνακας 5.1: Βασικές οδηγίες του OpenMP για τον παραλληλισμό δεδομένων

Όνομα οδηγίας	**Σκοπός**
`parallel`	Προηγείται παράλληλου τμήματος κώδικα
`for`	Προηγείται παράλληλης επανάληψης for
`parallel for`	Συνδυασμός των οδηγιών parallel και for

Η οδηγία `parallel` παίζει πολύ σημαντικό ρόλο στη συγγραφή παράλληλων προγραμμάτων αφού καθορίζει ποια τμήματα πρόκειται να εκτελεστούν παράλληλα. Όταν το νήμα-

συντονιστής συναντήσει την οδηγία `parallel` σε ένα τμήμα κώδικα, θα δημιουργήσει μια σειρά από επιπλέον νήματα, κάθε ένα από τα οποία θα εκτελέσει ένα μέρος αυτού του κώδικα. Στο τέλος του παράλληλου τμήματος, τα επιπλέον νήματα συγχρονίζονται, περιμένουν δηλαδή την ολοκλήρωση όλων των υπολοίπων και στη συνέχεια ενώνονται πίσω στον νήμα-συντονιστή. Η σύνταξη της οδηγίας είναι:

`#pragma omp parallel` ή `$omp parallel`

Η οδηγία `parallel for` αποτελεί ουσιαστικά το συνδυασμό των οδηγιών `parallel` και `for` και τοποθετείται ακριβώς πριν από μια επανάληψη `for`. Σκοπός της οδηγίας είναι η ενημέρωση του μεταγλωττιστή για την παράλληλη εκτέλεση της συγκεκριμένης επανάληψης. Για να μπορέσει ο μεταγλωττιστής να μετατρέψει τη σειριακή επανάληψη `for` σε παράλληλη θα πρέπει να ισχύουν μια σειρά από προϋποθέσεις: θα πρέπει να είναι γνωστός από την αρχή ο αριθμός των επαναλήψεων, να μην εξαρτάται η τιμή της μεταβλητής σε μια επανάληψη από την τιμή μεταβλητής άλλης επανάληψης ενώ τέλος μία επανάληψη δε θα πρέπει να περιέχει συναρτήσεις οι οποίες επιτρέπουν τον πρόωρο τερματισμό της εκτέλεσης της (για παράδειγμα τις `break, return, exit, goto`). Η σύνταξη της οδηγίας είναι:

`#pragma omp parallel for` ή `$omp parallel for`

Εκτός από τις βασικές αυτές οδηγίες, το OpenMP παρέχει και μια σειρά από συναρτήσεις, οι οποίες είναι απαραίτητες στη συγγραφή παράλληλων προγραμμάτων (πίνακας **5.2**).

Πίνακας 5.2: Χρήσιμες συναρτήσεις του OpenMP

Όνομα συνάρτησης	**Σκοπός**
`omp_get_num_procs`	Επιστρέφει τον αριθμό των διαθέσιμων επεξεργαστών
`omp_get_num_threads`	Επιστρέφει τον αριθμό των ενεργών νημάτων
`omp_get_thread_num`	Επιστρέφει τον αύξοντα αριθμό του τρέχοντος νήματος
`omp_set_num_threads`	Επιτρέπει την αυξομείωση του αριθμού των νημάτων

Όπως προαναφέρθηκε, το OpenMP βασίζεται στην τεχνική της διαμοιραζόμενης μνήμης όπου όλα τα δεδομένα μπορούν να προσπελαστούν από όλα τα νήματα. Πολλές φορές, όμως, χρειάζονται μεταβλητές που να έχουν διαφορετική τιμή για κάθε νήμα. Μια μεταβλητή ονομάζεται ιδιωτική (private) όταν κάθε νήμα κρατά αντίγραφό της ώστε να μπορεί να της αποδοθεί διαφορετική τιμή σε κάθε ένα από αυτά. Η τιμή μιας ιδιωτικής μεταβλητής δεν έχει αρχική τιμή, ενώ δε μπορεί να προσπελαστεί μετά το πέρας της εκτέλεσης του παράλληλου τμήματος στο οποίο χρησιμοποιείται. Ο καθορισμός μιας μεταβλητής ως ιδιωτική γίνεται με τη χρήση του ορίσματος `private` το οποίο προστίθεται στο τέλος μιας οδηγίας προς το μεταγλωττιστή. Η σύνταξη του ορίσματος είναι:

`private (<variable list>)`

Παραλληλοποίηση VNS με OpenMP

Η παραλληλοποίηση μιας μεθευρετικής μεθόδου μπορεί και πρέπει να πετυχαίνει τη μείωση του υπολογιστικού χρόνου, χωρίζοντας το σειριακό πρόγραμμα, ή την αύξηση της εξερεύνησης στο χώρο αναζήτησης, με την εφαρμογή ανεξάρτητων νημάτων αναζήτησης (Pérez, Hansen, and Mladenović 2005). Αρκετές στρατηγικές παραλληλοποίησης ενός αλγορίθμου VNS έχουν προταθεί και αναλυθεί στη βιβλιογραφία (García-López, Melián-Batista, Moreno-Pérez, and Moreno-Vega 2003, Crainic, Gendreau, Hansen, and Mladenović 2004).

Το OpenMP βασίζεται σε ένα συνδυασμό οδηγιών μεταφραστή (compiler directives), ρουτίνες βιβλιοθήκης και μεταβλητών περιβάλλοντος που μπορούν να χρησιμοποιηθούν ώστε να καθορίσουν παραλληλισμό διαμοιραζόμενης μνήμης σε προγράμματα C/C++ και Fortran. Για την παραλληλοποίηση με το OpenMP πρέπει να αντικατασταθούν, από συγκεκριμένες οδηγίες του μεταφραστή, λίγες μόνο γραμμές του σειριακού κώδικα. Οι δύο απλές στρατηγικές παραλληλοποίησης του VNS περιλαμβάνουν την παραλληλοποίηση της τοπικής αναζήτησης και την επανάληψη όλου του VNS στους επεξεργαστές.

Η πρώτη από τις δύο απλές στρατηγικές παραλληλοποίησης προσπαθεί να μειώσει τον υπολογιστικό χρόνο παραλληλοποιώντας την τοπική αναζήτηση στο σειριακό VNS και αναφέρεται ως **σύγχρονο παράλληλο VNS** (Synchronous Parallel VNS - SPVNS). Η δεύτερη υλοποιεί μια ανεξάρτητη στρατηγική αναζήτησης η οποία εκτελεί μία ανεξάρτητη διαδικασία VNS σε κάθε επεξεργαστή και αναφέρεται ως **επαναλαμβανόμενο παράλληλο VNS** (Replicated Parallel VNS - RPVNS).

Η παράλληλη τοπική αναζήτηση προσπαθεί να πετύχει ένα ισορροπημένο φόρτο στους επεξεργαστές. Η διαδικασία μοιράζει το σύνολο των $p(n-p)$ λύσεων της γειτονιάς της τρέχουσας λύσης στους διαθέσιμους *num_proc* επεξεργαστές ώστε να αναζητήσει την καλύτερη. Ο αλγόριθμος 12 δείχνει τον κώδικα της διαδικασίας `par_local_search` που υλοποιεί την παράλληλη τοπική αναζήτηση.

Ο ψευδοκώδικας του σύγχρονου παράλληλου VNS (SPVNS) παρουσιάζεται στον αλγόριθμο 13.

Ο δεύτερος απλός τρόπος παραλληλοποίησης του VNS είναι το επαναλαμβανόμενο παράλληλο VNS (Replicated Parallel VNS - RPVNS) ο οποίος προσπαθεί να αναζητήσει μια καλύτερη λύση εξερευνώντας μια πιο ευρεία ζώνη του χώρου λύσεων χρησιμοποιώντας πολυενακτηριακές (multistart) στρατηγικές. Αυτό επιτυγχάνεται αυξάνοντας τον αριθμό των γειτονικών λύσεων που ξεκινούν μια τοπική αναζήτηση, με μερικές αρχικές λύσεις είτε στην ίδια γειτονιά

Αλγόριθμος 12 Ψευδοκώδικας παράλληλης τοπικής αναζήτησης

```
function PAR_LOCAL_SEARCH(cur_sol)
    init_sol ← cur_sol
    while improved() do
        load ← (n − p) div (num_proc)
        parfor pr ← 0, num_proc − 1 do
            tmp_sol(pr) ← init_sol
            low ← pr ∗ load
            high ← low + load
            for i ← low, high − 1 do
                for j ← 0, p − 1 do
                    exchange(init_sol, new_sol, i, j)
                    if improve(new_sol, tmp_sol(pr)) then
                        tmp_sol(pr) ← new_sol
                    end if
                end for
            end for
            critical
            if improve(tmp_sol(pr), cur_sol) then
                cur_sol ← tmp_sol(pr)
            end if
        end parfor
    end while
end function
```

Αλγόριθμος 13 Ψευδοκώδικας σύγχρονου παράλληλου VNS (SPVNS)

```
initialize(best_sol)
k ← 0
while k < k_max do
    k ← k + 1
    cur_sol ← shake(best_sol, k)
    par_local_search(cur_sol)
    if improved(cur_sol, best_sol) then
        best_sol ← cur_sol
        k ← 0
    end if
end while
```

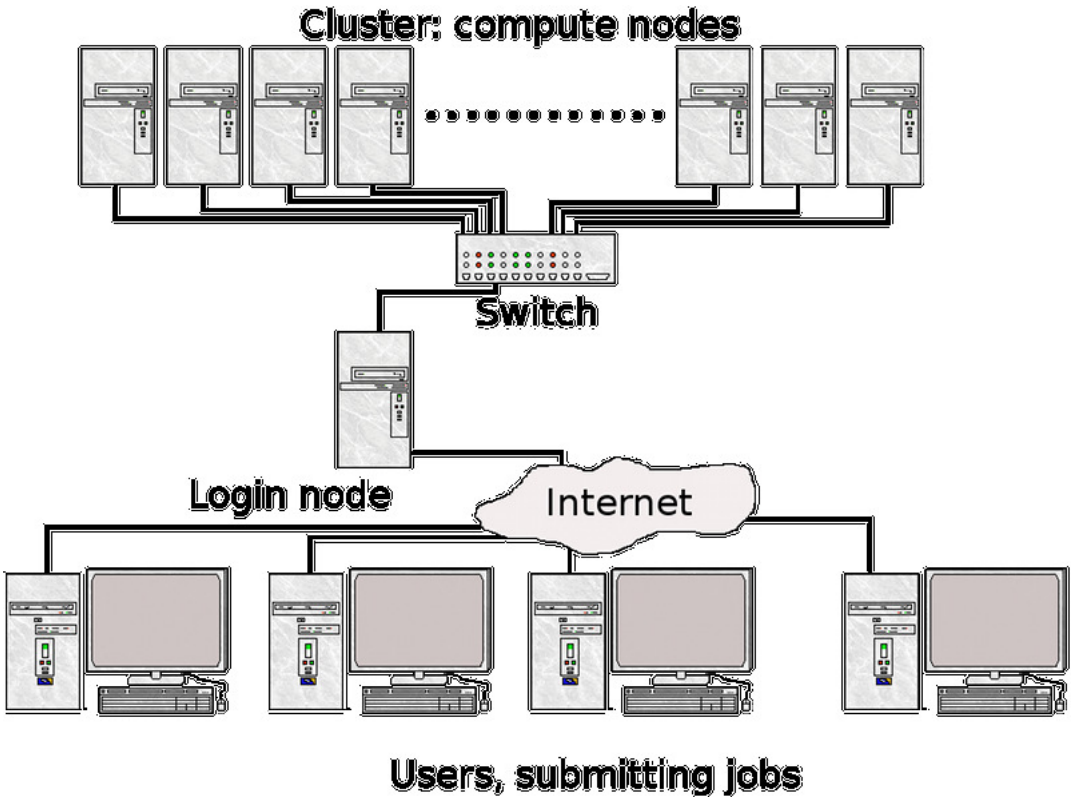
ή σε διαφορετικές. Αυτή η μέθοδος μοιάζει με μια πολυενακτηριακή (multistart) διαδικασία όπου κάθε τοπική αναζήτηση αντικαθίσταται από το VNS. Ο ψευδοκώδικας του RPVNS περιγράφεται στον αλγόριθμο 14.

Αλγόριθμος 14 Ψευδοκώδικας επαναλαμβανόμενου παράλληλου VNS (RPVNS)

```
initialize(joint_best_sol)
parallel pr ← 0 num_proc − 1
    Initialize(best_sol(pr))
    k(pr) ← 0
    while k(pr) < k_max do
        k(pr) ← k(pr) + 1
        cur_sol ← shake(best_sol(pr),k(pr))
        local_search(cur_sol(pr))
        if improved(cur_sol(pr),best_sol(pr)) then
            best_sol(pr) ← cur_sol(pr)
            k(pr) ← 0
        end if
    end while
    critical
    if improve(best_sol(pr), joint_best_sol) then
        joint_best_sol ← best_sol(pr)
    end if
end parallel
```

Αρχιτεκτονική MPI

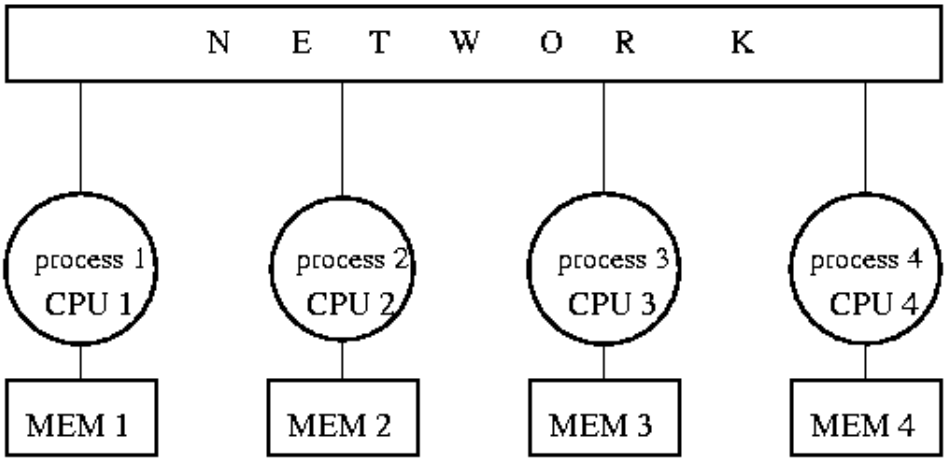
Τελευταία (Margaritis 2015), υπάρχει η τάση να απομακρυνθούμε από τους πολύ ακριβούς, εξειδικευμένους υπερυπολογιστές ενώ ταυτόχρονα αυξάνεται η καθιέρωση δικτύων υπολογιστών καθημερινής χρήσης. Αυτό οφείλεται στη ραγδαία αύξηση της χρήσης και της διαθεσιμότητας των προσωπικών ηλεκτρονικών υπολογιστών. Έτσι, η δημιουργία συστοιχιών υπολογιστών (computer clusters) καθίσταται μια πολύ οικονομική και αποδοτική λύση. Η συστοιχίες υπολογιστών είναι συστήματα τα οποία επιτρέπουν την εκτέλεση απαιτητικών παράλληλων εφαρμογών, χρησιμοποιούν εξελιγμένους μηχανισμούς για το διαμοιρασμό των πόρων όλων των διαθέσιμων κόμβων και είναι εύκολα επεκτάσιμα.



Σχήμα 10: Η αρχιτεκτονική συστοιχιών υπολογιστών (HPC2N 2016)

Στο σχήμα 10 απεικονίζεται η αρχιτεκτονική συστοιχιών υπολογιστών. Μια τυπική συστοιχία αποτελείται από υπολογιστές συνδεδεμένους μεταξύ τους και έναν κεντρικό υπολογιστή ο οποίος αναλαμβάνει το συντονισμό τους. Καθένας από αυτούς τους υπολογιστές έχει εγκατεστημένο ειδικό λογισμικό που χρησιμεύει στη διασύνδεση των κόμβων μεταξύ τους και ονομάζεται ενδιάμεσο στρώμα (middleware).

Στις συστοιχίες υπολογιστών χρησιμοποιείται το μοντέλο μεταβίβασης μηνυμάτων (message-passing), μια τεχνική παράλληλου προγραμματισμού, σύμφωνα με το οποίο οι κόμβοι που συμμετέχουν στη συστοιχία έχουν ξεχωριστή μνήμη ο καθένας, αντίθετα από τα μοντέλα διαμοιραζόμενης μνήμης, επικοινωνώντας μεταξύ τους με ανταλλαγή μηνυμάτων. Το MPI (Message Passing Interface) είναι ένα τέτοιο πρότυπο μεταβίβασης μηνυμάτων (σχήμα 11) για τον προγραμματισμό εφαρμογών παράλληλης επεξεργασίας. Το πρότυπο MPI καθορίζει τη σύνταξη ενός αριθμού από συναρτήσεις, οι οποίες είναι χρήσιμες για τη δημιουργία προγραμμάτων μεταβίβασης μηνυμάτων σε γλώσσες προγραμματισμού C/C++ και Fortran.



Σχήμα 11: Το μοντέλο μεταβίβασης μηνυμάτων (Kotelnikov 2004)

Ένα πρόγραμμα βασισμένο στο πρότυπο MPI αποτελείται από πολλές διεργασίες που εκτελούνται παράλληλα. Κάθε διεργασία εκτελεί ένα τμήμα του ίδιου προγράμματος χρησιμοποιώντας τη δική της θέση μνήμης, επικοινωνώντας με τις υπόλοιπες διεργασίες μεταβιβάζοντας μηνύματα. Ο προγραμματιστής μπορεί να καλέσει τις συναρτήσεις αποστολής και παραλαβής των μηνυμάτων μέσα από την εφαρμογή του, ενώ, για απλότητα, όλες οι λεπτομέρειες συντονίζονται από το MPI. Το μοντέλο μεταβίβασης μηνυμάτων έχει γίνει εξαιρετικά δημοφιλές λόγω της ευκολίας στη χρήση του αλλά και του μεγάλου αριθμού από πλατφόρμες που το υποστηρίζουν. Τα προγράμματα που έχουν αναπτυχθεί με αυτό τον τρόπο μπορούν να εκτελούνται σε διαφορετικές αρχιτεκτονικές υπολογιστών, ενώ ο εκτελέσιμος κώδικας που παράγεται είναι αρκετά αποδοτικός.

Το MPI απαρτίζεται από ένα μεγάλο αριθμό συναρτήσεων, οι οποίες μπορούν να χρησιμοποιηθούν στη συγγραφή εφαρμογών. Αρκετά προγράμματα, όμως, μπορούν να γραφούν με τη χρήση μόνο των επόμενων βασικών συναρτήσεων (πίνακας **5.3**).

Πίνακας 5.3: Οι βασικές συναρτήσεις του MPI

Όνομα συνάρτησης	**Σκοπός**
`MPI_Init()`	Αρχικοποίηση του MPI
`MPI_Comm_rank()`	Αριθμός διεργασίας
`MPI_Comm_size()`	Σύνολο διεργασιών
`MPI_Recv()`	Παραλαβή μηνυμάτων
`MPI_Send()`	Αποστολή μηνυμάτων
`MPI_Finalize()`	Τερματισμός του MPI

Κάθε πρόγραμμα που χρησιμοποιεί τις συναρτήσεις του MPI θα πρέπει να συμπεριλαμβάνει το αρχείο επικεφαλίδας του MPI στο οποίο περιέχονται πληροφορίες σχετικά με τις διαθέσιμες συναρτήσεις:

```
#include "mpi.h"
```

Πριν κληθεί οποιαδήποτε άλλη συνάρτηση, πρέπει πρώτα να κληθεί η συνάρτηση `MPI_Init()`, ώστε να αρχικοποιηθεί το MPI. Η συνάρτηση `MPI_Init()` παίρνει ως ορίσματα δείκτες στα ορίσματα της main, argc και argv:

```
MPI_Init(int argc, char **argv);
```

Όταν ένα πρόγραμμα που χρησιμοποιεί τη βιβλιοθήκη του MPI ολοκληρώνεται, η συνάρτηση `MPI_Finalize()` πρέπει να κληθεί για να ελευθερωθούν οι πόροι που έχει δεσμεύσει το MPI. Από τη στιγμή που θα γίνει κλήση στην `MPI_Finalize()`, δεν μπορεί να χρησιμοποιηθεί οποιοδήποτε άλλη συνάρτηση του MPI:

```
MPI_Finalize();
```

Στο MPI, κάθε διεργασία έχει το δικό της αριθμό, με βάση τον οποίο μπορεί να ξεχωρίζει από τις υπόλοιπες διεργασίες. Η συνάρτηση `MPI_Comm_rank` επιστρέφει τον αριθμό αυτό κάθε διεργασίας και η σύνταξή της είναι η ακόλουθη:

```
MPI_Comm_rank(MPI_Comm comm, int *rank)
```

Το πρώτο όρισμα ονομάζεται περιβάλλον επικοινωνίας (communicator) και ουσιαστικά αποτελεί μία τεχνική ομαδοποίησης των διεργασιών. Αρχικά, όλες οι διεργασίες ανήκουν σε ένα προκαθορισμένο περιβάλλον επικοινωνίας, το `MPI_COMM_WORLD` ενώ εκτός του προκαθορισμένου περιβάλλοντος μπορούν να δημιουργηθούν και νέα περιβάλλοντα επικοινωνίας τα οποία χρησιμεύουν κυρίως στην οργάνωση των κόμβων σε τοπολογίες. Πολλές φορές η λειτουργία ενός προγράμματος μπορεί να εξαρτάται από το συνολικό αριθμό των διεργασιών στις οποίες εκτελείται. Για την εύρεσή του χρησιμοποιείται η συνάρτηση `MPI_Comm_size` της οποίας η σύνταξη είναι:

```
MPI_Comm_size(MPI_Comm comm, int *size)
```

Η ανταλλαγή μηνυμάτων μεταξύ των διεργασιών εκτελείται με τις συναρτήσεις `MPI_Send()` και `MPI_Recv()`. Η πρώτη συνάρτηση αποστέλλει ένα μήνυμα σε μία καθορισμένη διεργασία, ενώ η δεύτερη λαμβάνει ένα μήνυμα από μία διεργασία. Για να αποσταλεί επιτυχώς το μήνυμα αυτό, το σύστημα πρέπει να συμπεριλάβει τις ακόλουθες πληροφορίες:

- Τον αριθμό του παραλήπτη,
- τον αριθμό του αποστολέα,
- ένα επίθεμα,
- το περιβάλλον επικοινωνίας.

Το επίθεμα (tag) μπορεί να χρησιμοποιηθεί από τον αποστολέα ώστε να κάνει διαχωρισμό μεταξύ των εισερχόμενων μηνυμάτων.

Παραλληλοποίηση του VNS με MPI

Οι Davidović και Crainic (Davidović and Crainic 2012) περιγράφουν πέντε στρατηγικές παραλληλοποίησης για το VNS με χρήση MPI. Οι στρατηγικές αυτές κυμαίνονται από λεπτομερή (fine-grained) παράλληλη αναζήτηση μέχρι αδρομερή (coarse-grained) παράλληλη αναζήτηση:

- **PVNSPLS**. Η λεπτομερής (fine-grained) παράλληλη αναζήτηση στο VNS καταδεικνύει ότι βελτιώνεται τόσο η ποιότητα της τελικής λύσης όσο και ο χρόνος εκτέλεσης. Αντιπροσωπεύει την επιτάχυνση του σειριακού VNS παραλληλοποιώντας το πιο υπολογιστικά εντατικό κομμάτι, την τοπική αναζήτηση.

- **Distributive VNS (DVNS)**. Βασίζεται σε μια ιδέα για την εξερεύνηση διαφορετικών γειτονιών παράλληλα. Το πετυχαίνει εκτελώντας τα βασικά βήματα του VNS σε διαφορετικές γειτονιές σε διαφορετικούς επεξεργαστές την ίδια στιγμή.

- **CVNSall**. Αντιπροσωπεύει την κεντρική, ασύγχρονη συνεργασία διαφορετικών αλγορίθμων VNS μεσαίας λεπτομέρειας. Κάθε επεξεργαστής εκτελεί διαφορετικό βήμα του VNS και αναφέρεται στην κεντρική μνήμη. Η καλύτερη λύση του επεξεργαστή συγκρίνεται με το ολικό βέλτιστο από την κεντρική μνήμη και το καλύτερο μεταξύ τους γίνεται το νέο σημείο αναφοράς για περαιτέρω αναζήτηση.

- **CVNSkkALL**. Αντιπροσωπεύει την αδρομερή, κεντρική, ασύγχρονη συνεργασία διαφορετικών αλγορίθμων VNS. Στην περίπτωση του CVNSkkALL κάθε επεξεργαστής εκτελεί ολόκληρη την επανάληψη του VNS, μέχρι $k = k\_max$, πριν αναφερθεί στην κεντρική μνήμη. Υλοποιείται, όπως και η CVNSall σε αρχιτεκτονική πολυεπεξεργασίας αστέρα με τον κεντρικό επεξεργαστή να παίζει το ρόλο της κεντρικής μνήμης. Ανανεώνει την τρέχουσα ολική καλύτερη λύση και τη στέλνει στους άλλους επεξεργαστές έπειτα από αίτησή τους. Οι υπόλοιποι επεξεργαστές εκτελούν διάφορους αλγορίθμους VNS.

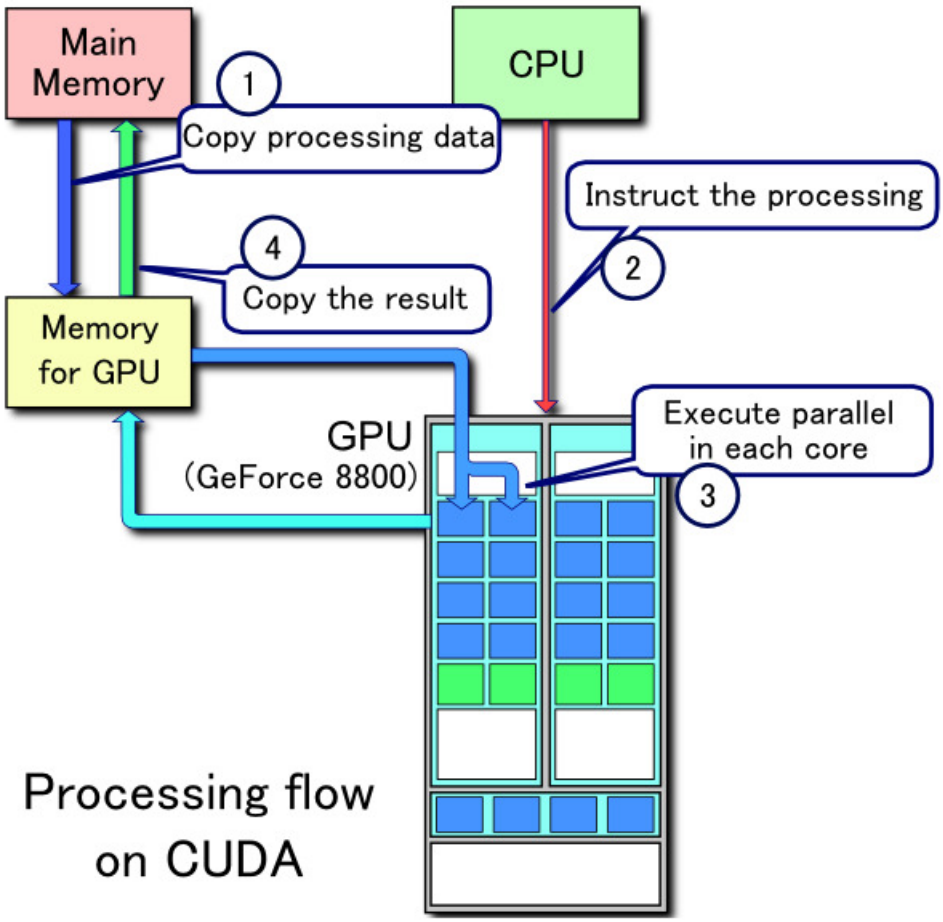
- **CVNSring**. Αντιπροσωπεύει τη μη-κεντρική, ασύγχρονη συνεργασία διαφορετικών αλγορίθμων VNS μεσαίας λεπτομέρειας. Υλοποιείται με ένα δικατευθυντήριο δακτύλιο διαδικασιών. Περιλαμβάνει διαφορετικούς αλγορίθμους VNS και πάντα η καλύτερη λύση γίνεται νέο σημείο αναφοράς.

Αρχιτεκτονική CUDA

Η αρχιτεκτονική CUDA (Compute Unified Device Architecture) είναι μία Διεπαφή Προγραμματισμού Εφαρμογών (API - Application Programming Interface) αλλά και μια παράλ-

ληλη υπολογιστική πλατφόρμα που αναπτύχθηκε από την εταιρία NVIDIA. Επιτρέπει στους προγραμματιστές να χρησιμοποιούν τις γραφικές μονάδες επεξεργασίας (GPU) με δυνατότητες CUDA για γενική επεξεργασία, μια προσέγγιση γνωστή ως GPGPU. Η πλατφόρμα CUDA είναι ένα επίπεδο προγραμματισμού που επιτρέπει την άμεση πρόσβαση στο εικονικό σύνολο εντολών της GPU για τον υπολογισμό πυρήνων.

Η πλατφόρμα CUDA έχει σχεδιαστεί για τις γλώσσες προγραμματισμού C/C++ και Fortran. Έτσι, επιτρέπει την ευκολότερη ανάπτυξη και χρήση των πόρων της GPU. Επίσης, η CUDA επιτρέπει υπολογιστικά πλαίσια όπως το OpenACC και OpenCL. Στο σχήμα 12 απεικονίζεται η διαδικασία επεξεργασίας της CUDA.

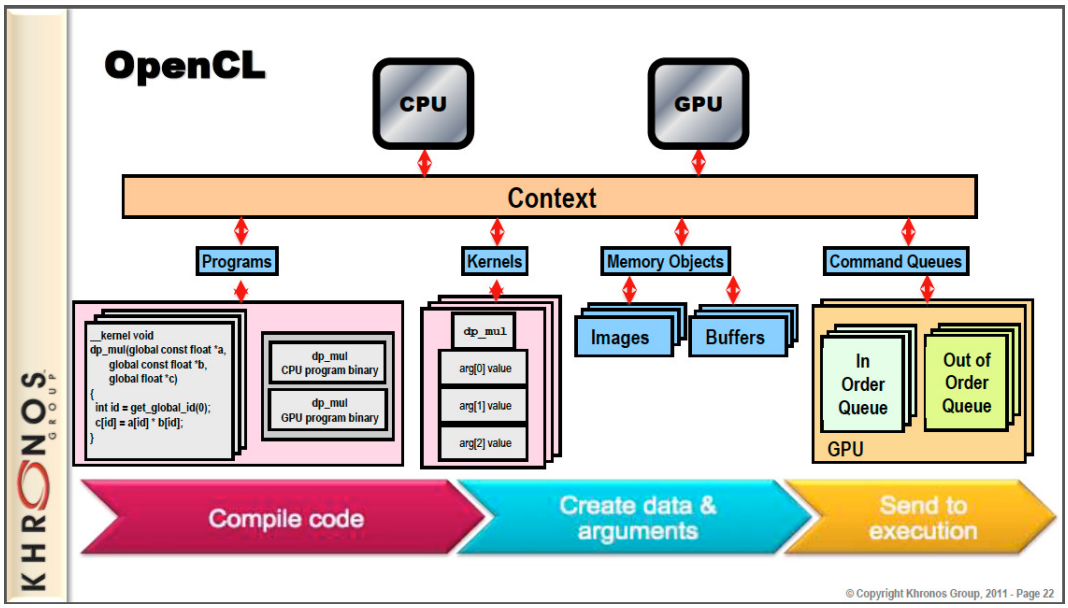


Σχήμα 12: Η αρχιτεκτονική CUDA (Tosaka 2008)

Αρχιτεκτονική OpenCL

Το OpenCL (Open Computing Language) είναι ένα ανοιχτό πρότυπο (Margaritis 2015) για την υλοποίηση εφαρμογών γενικού σκοπού, οι οποίες εκτελούνται σε ετερογενείς πλατφόρμες αποτελούμενες από CPU, GPU και άλλους επεξεργαστές, δίνοντας έτσι την δυνατότητα στους προγραμματιστές να εκμεταλλευτούν στο έπακρο όλη την υπολογιστική ισχύ ενός συστήματος.

Αρχικά, το OpenCL κατασκευάστηκε από την Apple Inc., η οποία κατέχει και τα δικαιώματα του εμπορικού σήματος. Το πρότυπο αυτό συνεχώς εξελίσσεται από την ομάδα Khronos, η οποία δημιουργήθηκε από το καλοκαίρι του 2008 με τη συνεργασία πολλών εταιριών όπως των AMD, IBM, Intel και Nvidia. Το OpenCL περιλαμβάνει δύο βασικά στοιχεία (σχήμα 13) Το API που χρησιμοποιείται για τον συντονισμό των παράλληλων εφαρμογών σε ετερογενείς επεξεργαστές και μια γλώσσα προγραμματισμού ανεξαρτήτου πλατφόρμας, την OpenCL C (βασιζόμενη στην C99).



Σχήμα 13: Η αρχιτεκτονική OpenCL (Rosenberg 2011)

Αρχιτεκτονική OpenACC

Το OpenACC (for Open Accelerators) είναι ένα προγραμματιστικό πρότυπο για παράλληλο υπολογισμό που αναπτύχθηκε από τις εταιρίες Cray, CAPS, Nvidia και PGI. Το πρότυπο αυτό σχεδιάστηκε ώστε να απλοποιήσει τον παράλληλο προγραμματισμό σε ετερογενή συστήματα CPU/GPU.

Όπως στο OpenMP, ο προγραμματιστής μπορεί να επισημάνει τα κομμάτια του κώδικα, σε γλώσσες C/C++, Fortran, που επιθυμεί να παραλληλοποιηθεί/επιταχυνθεί χρησιμοποιώντας οδηγίες (directives) και συναρτήσεις. Όπως στο OpenMP, από την έκδοση 4.0 και έπειτα, ο κώδικας μπορεί να γίνει offload και στη CPU και στη GPU. Γίνονται προσπάθειες ενοποίησης OpenACC, OpenMP, αφού πλέον το OpenMP υποστηρίζει GPU και το OpenACC υποστηρίζει CPU, έτσι ώστε να δημιουργηθεί ένα κοινό πρότυπο το οποίο να επεκτείνει το OpenMP.

Η υποστήριξη του OpenACC προέρχεται κυρίως από εμπορικούς μεταφραστές (commercial compilers) από τις εταιρίες PGI, Cray, CAPS. Μεταφραστές ανοιχτού κώδικα είναι οι OpenUH,

OpenARC, accULL οι περισσότεροι σε πειραματικές εκδόσεις. Η υποστήριξη του GCC για το OpenACC ξεκίνησε πειραματικά από την έκδοση 5.1 υποστηρίζοντας, μερικώς, την έκδοση OpenACC 2.0a.

ΚΕΦΑΛΑΙΟ 6

Τεχνικά θέματα υλοποίησης

Εισαγωγή

Το πρόβλημα διαχείρισης αποθεμάτων που επιλέξαμε να παραλληλοποιήσαμε είναι το πρόβλημα δυναμικού μεγέθους παρτίδας πολλών προϊόντων με δραστηριότητες ανακατασκευής (Multi-product Dynamic Lot Sizing Problem with Remanufacturing activities - MDLSRP) (Sifaleras and Konstantaras 2015b) το οποίο αναφέραμε στο τέταρτο κεφάλαιο της παρούσας εργασίας. Το πρόβλημα έχει επιλυθεί (Sifaleras and Konstantaras 2015b, Sifaleras, Konstantaras, and Mladenović 2015) χρησιμοποιώντας τη μεθευρετική μέθοδο της κατάβασης μεταβλητής γειτονιάς (Variable Neighbourhood Descent - VND), στο οποίο αναφερθήκαμε στο τρίτο κεφάλαιο της παρούσας εργασίας.

Για την παραλληλοποίηση της μεθευρετικής μεθόδου VNS έχουν προταθεί τεχνικές είτε με OpenMP είτε με MPI (Pérez, Hansen, and Mladenović 2005, Davidović and Crainic 2012). Τα μοντέλα που προτείνονται προϋποθέτουν, όμως, τη σχεδίαση από την αρχή του αλγορίθμου ώστε να μπορεί να εκτελεστεί παράλληλα. Στόχος μας, στην παρούσα εργασία, είναι, χωρίς να αλλάξουμε δραστικά το σχεδιασμό του αλγορίθμου, να προσπαθήσουμε να παραλληλοποιήσουμε τον αλγόριθμο, με τη χρήση των τεχνολογιών OpenMP και OpenACC, ώστε να μειώσουμε το χρόνο εκτέλεσης αλλά και να προσπαθήσουμε να μεγαλώσουμε το χώρο αναζήτησης πετυχαίνοντας καλύτερα, πιο ποιοτικά, αποτελέσματα. Και οι δύο αυτές τεχνολογίες χρησιμοποιούν οδηγίες (directives) ώστε να πετύχουν την παραλληλοποίηση. Η απόδοσή τους είναι συγκρίσιμη με αυτή μεθόδων όπως OpenCL, CUDA αλλά είναι πολύ πιο εύκολες στην υλοποίηση.

Ανάλυση του σειριακού αλγορίθμου VND

Για την επίλυση του προβλήματος MDLSRP χρησιμοποιήσαμε ως βάση τον αλγόριθμο που προτείνεται από (Sifaleras, Konstantaras, and Mladenović 2015) ο οποίος βασίζεται στον

υπολογισμό του συνολικού αριθμού των βελτιώσεων της τοπικής αναζήτησης ανά γειτονιά. Με αυτό τον τρόπο, οι πιο συχνά χρησιμοποιούμενες μετακινήσεις ανά γειτονιά υπολογίζονται πρώτες. Έτσι, γίνεται ταξινόμηση των γειτονιών με βάση την επιτυχία τους στις βελτιώσεις στην τοπική αναζήτηση ανά γειτονιά. Αυτή η ταξινόμηση θα χρησιμοποιηθεί κατά τη φάση τοπικής αναζήτησης του VNS. Αυτή η παραλλαγή της τοπικής αναζήτησης ονομάζεται (Sifaleras, Konstantaras, and Mladenović 2015) αλγόριθμος ταξινομημένης γενικής κατάβασης μεταβλητής γειτονιάς (Ordered GVND - OGVND) και το VNS που παίρνουμε αντίστοιχα ονομάζεται αλγόριθμος ταξινομημένης γενικής αναζήτησης μεταβλητής γειτονιάς (Ordered GVNS - OGVNS).

Στη αρχή, χρησιμοποιείται μια απλή ευρετική μέθοδος αρχικοποίησης ώστε να υπάρξει η αρχική λύση. Σε αυτή η συνολική ζήτηση ικανοποιείται την πρώτη περίοδο, από μία μόνο παρτίδα, χωρίς ανακατασκευασμένα κομμάτια (αλγόριθμος 15).

Αλγόριθμος 15 Ευρετική μέθοδος αρχικοποίησης

procedure HEURISTIC_START(K, T, R, D, x_R, x_M, y_R, y_M, z_R, z_M)
  $x_R(:,:), z_R(:,:), x_M(:,:), z_M(:,:), y_R(:,:), y_M(:,:) \leftarrow 0$
  $y_R(:,1) \leftarrow R(:,1)$
  for $i \leftarrow 1, K$ **do**
    for $j \leftarrow 2, T$ **do**
      $y_R(i,j) \leftarrow y_R(i,j-1) + R(i,j)$
      $y_M(i, T+1-j) \leftarrow y_M(i, T+2-j) + D(i, T+2-j)$
    end for
  end for
  $x_M(:,1) \leftarrow y_M(:,1) + D(:,1)$
  $z_M(:,1) \leftarrow 1$
end procedure

Μετά εφαρμόζεται η ευρετική μέθοδος VND ώστε να βελτιώσει την αρχική λύση. Ο αριθμός των διαφορετικών γειτονιών και η αντίστοιχη σειρά τους στο VND είναι πολύ σημαντικός παράγοντας για την αποτελεσματικότητα οποιασδήποτε μεθοδολογίας βασισμένης σε VNS. Ο αλγόριθμος VND αποτελεί ένα ντετερμινιστικό αλγόριθμο που σκοπό έχει να εντατικοποιήσει την τοπική αναζήτηση στις γειτονιές, των οποίων για τον τροποποιημένο αλγόριθμο OGVNS η σειρά αποφασίστηκε (Sifaleras, Konstantaras, and Mladenović 2015) να είναι για τα πειράματα η εξής: $N_4, N_{15}, N_1, N_2, N_{10}, N_{19}, N_{12}, N_8, N_5, N_{18}, N_{16}, N_7, N_{17}, N_{14}, N_{13}, N_3, N_6, N_{11}, N_9$.

Αν και αρκετοί προτείνουν μικρό αριθμό από γειτονιές, 3 ή 4, ακολουθήθηκε διαφορετική προσέγγιση για την ανάπτυξη αποτελεσματικού αλγορίθμου VND. Ο λόγος που έγινε αυτό είναι ότι κάθε γειτονιά αντιστοιχεί σε ένα συνδυασμό διαφορετικών μετακινήσεων, οι οποίες μεταβάλλουν τις μεταβλητές ελέγχου, με τέτοιο τρόπο ώστε πάντοτε οι λύσεις να είναι εφικτές (αλγόριθμος 16).

Αλγόριθμος 16 Σειριακό VND

```
procedure VND(K, T, R, D, h_R, h_M, k_R, k_M, p_R, p_M, s, k_max)
    repeat
        improvement ← 0
        for k ← 1, k_max do
            for i ← 1, K do
                for t ← 1, T do
                    Βρες τον καλύτερο γείτονα s' του s(s' ∈ N_k(s))
                    if Η λύση s' είναι καλύτερη του s then
                        s ← s'
                        improvement ← 1
                    end if
                end for
            end for
        end for
    until improvement = 0
end procedure
```

Παραλληλοποίηση του αλγορίθμου VND

Για να παραλληλοποιήσουμε τον αλγόριθμο χωρίς να αλλάξουμε τη δομή των γειτονιών του VND έπρεπε να χρησιμοποιήσουμε κάποια τεχνική όπως OpenMP, μόνο με οδηγίες (directives) ώστε γρήγορα να πετύχουμε, αδρομερώς (coarse-grained), μια πολύ καλή επιτάχυνση. Πριν φτάσουμε εκεί, προσπαθήσαμε να υλοποιήσουμε κάποιες αλγοριθμικές βελτιώσεις στο σειριακό αλγόριθμο, κάτι που πράξαμε με επιτυχία.

Το επόμενο βήμα ήταν να προσπαθήσουμε να παραλληλοποιήσουμε την εσωτερική των βρόχων συνάρτηση «βρες τον καλύτερο γείτονα» όπου ουσιαστικά υλοποιεί την αναζήτηση σε μια συγκεκριμένη γειτονιά. Αναφέραμε, όμως, πριν ότι σε κάθε γειτονιά γίνονται μετακινήσεις οι οποίες μεταβάλλουν τις μεταβλητές ελέγχου ώστε πάντοτε οι λύσεις να είναι εφικτές. Το κόστος αυτής της τεχνικής είναι ότι τα δεδομένα αλληλοεξαρτώνται με αποτέλεσμα να είναι ανέφικτη η παραλληλοποίηση της τοπικής αναζήτησης σε κάθε γειτονιά.

Προσπαθήσαμε, λοιπόν, να παραλληλοποιήσουμε τους δύο εσωτερικούς βρόχους, για τα προϊόντα και τις περιόδους. Αναγκαστήκαμε να καταφύγουμε σε κάποιες προχωρημένες προγραμματιστικές τεχνικές (Süß and Leopold 2008) ώστε να αποφύγουμε τα λανθασμένα αποτελέσματα της παραλληλοποίησης σε ότι αφορά την τιμή της μεταβλητής *improvement* αλλά και για την αλλαγή της τιμής του καλύτερου αποτελέσματος. Επίσης, λόγω της αλληλεξάρτησης των δεδομένων αναγκαστήκαμε να χαρακτηρίσουμε ως «κρίσιμο τμήμα» τη συνάρτηση «βρες τον καλύτερο γείτονα» για κάθε γειτονιά και, για αυτό το λόγο, εκτελείται σειριακά.

Αφού κάναμε τις δοκιμές μας με το OpenMP δοκιμάσαμε και το πρότυπο OpenACC ώστε να προσπαθήσουμε να εκμεταλλευτούμε (και) τη GPU. Βέβαια, το πρόβλημά μας, δεν είναι compute-intensive και περιλαμβάνει, λόγω των πολλών γειτονιών και των εξαρτήσεων των

Αλγόριθμος 17 Παράλληλο VND με χρήση OpenMP

```
procedure VNDOpenMP(K, T, R, D, h_R, h_M, k_R, k_M, p_R, p_M, s, k_max)
    repeat
        improvement ← 0
        for k ← 1, k_max do
            !$OMP PARALLEL PRIVATE (s)                          ▷ s:Η τιμή του αποτελέσματος
            !$OMP DO SCHEDULE(STATIC) REDUCTION(+ : diff)       ▷ diff:Βοηθητική μεταβλητή
            for i ← 1, K do
                for t ← 1, T do
                    !$OMP CRITICAL
                    Βρες τον καλύτερο γείτονα s'_i του s(s'_i ∈ N_k(s))
                    !$OMP END CRITICAL
                end for
                diff ← diff + s − s'_i
            end for
            !$OMP END DO
            !$OMP END PARALLEL
            s ← s − diff
            if diff > 0 then
                improvement ← 1
            end if
        end for
    until improvement = 0
end procedure
```

δεδομένων, πολλές μετακινήσεις δεδομένων στη μνήμη, πράγμα που είναι αποθαρρυντικό όταν προσπαθούμε να παραλληλοποιήσουμε κάποιο πρόγραμμα στη GPU. Χρησιμοποιώντας τις κατάλληλες οδηγίες (directives) ώστε να αποφύγουμε άσκοπες μετακινήσεις δεδομένων από και προς τη GPU, με την οδηγία $*acc data copy*, καταλήξαμε στον αλγόριθμο 18 που ακολουθεί:

Αλγόριθμος 18 Παράλληλο VND με χρήση OpenACC

```
procedure VNDOpenACC(K, T, R, D, h_R, h_M, k_R, k_M, p_R, p_M, s, k_max)
    !$acc data copy(< global variables >)
    repeat
        improvement ← 0
        for k ← 1, k_max do
            !$acc parallel loop private(< private variables >) reduction(+ : diff) gang vector
            for i ← 1, K do
                !$acc loop
                for t ← 1, T do
                    Βρες τον καλύτερο γείτονα s'_i του s(s'_i ∈ N_k(s))
                end for
                diff ← diff + s − s'_i
            end for
            s ← s − diff
            if diff > 0 then
                improvement ← 1
            end if
        end for
    until improvement = 0
    !$acc end data
end procedure
```

Παρατηρούμε ότι ο κώδικας μοιάζει πάρα πολύ με αυτόν με OpenMP μόνο που εδώ δε

χρειαζόμαστε κρίσιμο τμήμα, αρκεί μέσα στη συνάρτηση «βρες τον καλύτερο γείτονα» να υπάρχει η οδηγία !$*acc routine*. Προσπαθήσαμε να χρησιμοποιήσουμε μια πιο «αυτόματη» οδηγία την !$*acc kernels* με την οποία ο μεταφραστής βρίσκει την κατάλληλη παραλληλοποίηση, όμως λόγω της εξάρτησης των δεδομένων δεν μπορούσαμε να εξάγουμε ορθά αποτελέσματα.

ΚΕΦΑΛΑΙΟ 7

Αποτελέσματα πειραμάτων

Εισαγωγή

Ο παράλληλος μεθευρετικός αλγόριθμος που αναπτύξαμε εφαρμόστηκε σε ένα σύνολο μετροπροβλημάτων (Sifaleras and Konstantaras 2015b, Sifaleras, Konstantaras, and Mladenović 2015) με 300 προϊόντα, 52 περιόδους και μεταβλητές παραμέτρους όπως $h_R(k) = \{0.2, 0.5, 0.8\}$, $k_R(k) = \{200, 500, 2000\}$ και $k_M(k) = \{200, 500, 2000\}$. Τα 108 αυτά στιγμιότυπα, διαθέσιμα `http://users.uom.gr/~sifalera/benchmarks.html` έχουν δημιουργηθεί χρησιμοποιώντας τους 27 συνδυασμούς των τιμών των $h_R(k)$, $k_R(k)$ και $k_M(k)$ για τέσσερα διαφορετικά σύνολα δεδομένων από τιμές ζήτησης και επιστροφών, δηλαδή $27 \times 4 = 108$ ακολουθώντας κανονική κατανομή με μικρή ή μεγάλη διακύμανση. Για κάθε σύνολο δεδομένων εκτελέστηκε πείραμα 10 φορές και μετρήσαμε το μέσο όρο του χρόνου εκτέλεσής του. Εδώ να αναφέρουμε ότι επειδή πειραματιζόμαστε με πολλά νήματα ο χρόνος εκτέλεσης με τη χρήση της συνάρτησης `cpu_time` δε θα έβγαζε σωστά αποτελέσματα. Έτσι, όταν ξεκινάμε καλούμε τη συνάρτηση `system_clock`: `call system_clock(count_rate = tick)` και `call system_clock(start_time)` και όταν τελειώσουμε την ξανακαλούμε: `call system_clock(end_time)` και `time = (end_time - start_time) / real(tick)` και ο χρόνος εκτέλεσης βρίσκεται ως πραγματικός αριθμός, σε δευτερόλεπτα, στη μεταβλητή *time*.

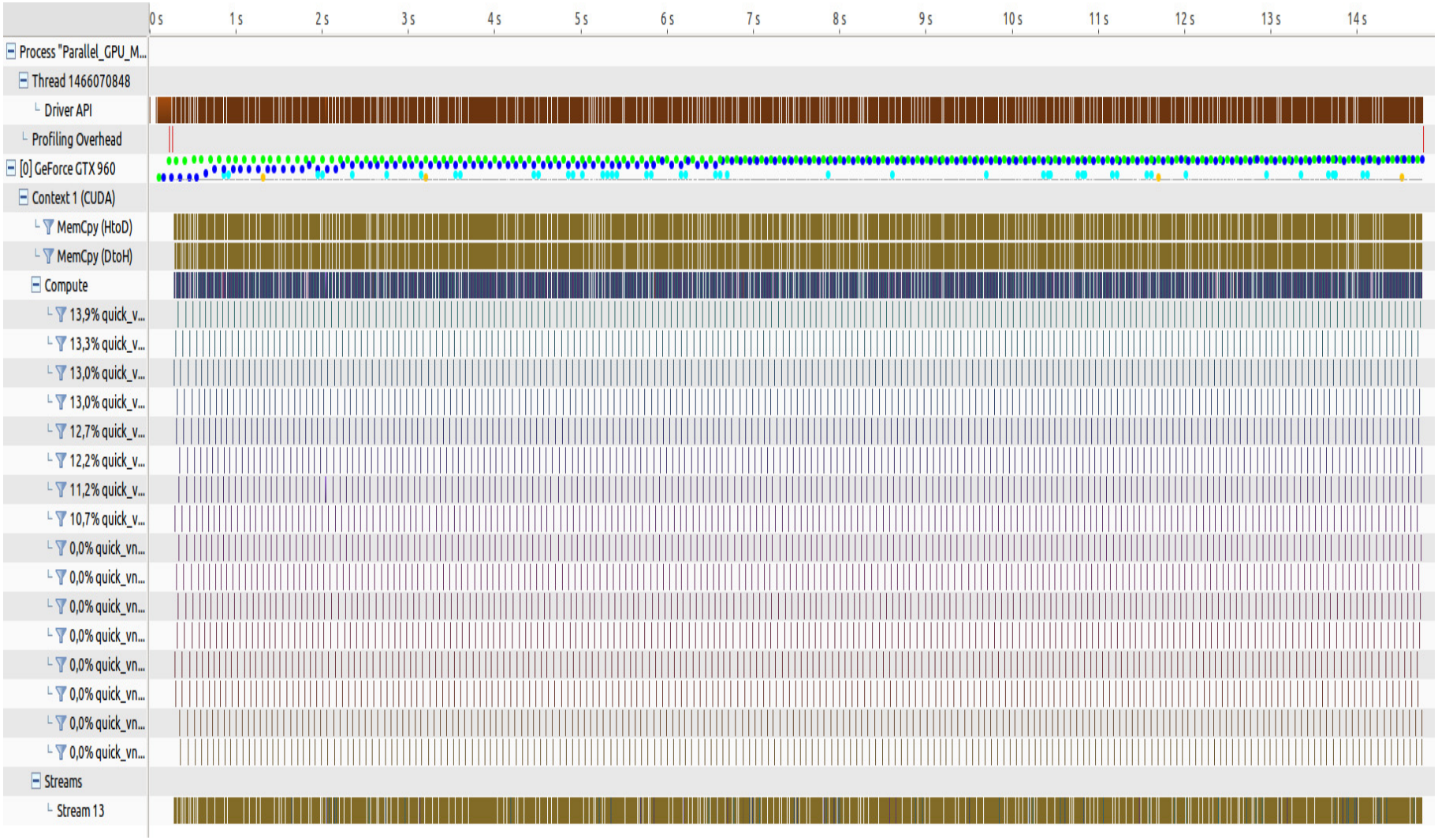
Τα πειράματα εκτελέστηκαν σε υπολογιστή με Ubuntu Linux 14.04 64bit με επεξεργαστή Intel Core i7 CPU 920 στα 2.793 GHz με 8 MB SmartCache και 6 GB DDR3 400 MHz κεντρική μνήμη. Η κάρτα γραφικών που χρησιμοποιήθηκε είναι η NVIDIA GeForce GTX 960 με μνήμη 2 GB GDDR5/128bit σε δίαυλο PCI Express x16 Gen2. Η υπολογιστική ικανότητα (compute capability) της κάρτας γραφικών, κατά NVIDIA, είναι 5.2 κάτι που επιτρέπει τη χρήση του OpenACC, αφού η PGI απαιτεί κατ' ελάχιστο compute capability 2.0. Η υλοποίηση έγινε με τη χρήση της γλώσσας προγραμματισμού Fortran και μεταφράστηκε με τη GNU

Fortran 4.8.4 για το μοντέλο OpenMP και τη PGI Fortran 15.10 (academic edition) για τα μοντέλα OpenMP, OpenACC. Να σημειωθεί ότι στην ακαδημαϊκή έκδοση του PGI compiler μπορούμε να έχουμε μέχρι 4 νήματα στο OpenMP, κάτι που επηρεάζει τα αποτελέσματα, αφού ο επεξεργαστής στον οποίο εκτελέστηκαν τα πειράματα διαθέτει 8 πυρήνες. Επίσης, στην ακαδημαϊκή έκδοση δε διατίθεται το PGPROF, ο αναλυτής επιδόσεων της PGI, που βοηθά στη βελτίωση της παραλληλοποίησης. Κάναμε ανάλυση μόνο με το nvprof και το Visual Profiler που διατίθενται από την CUDA.

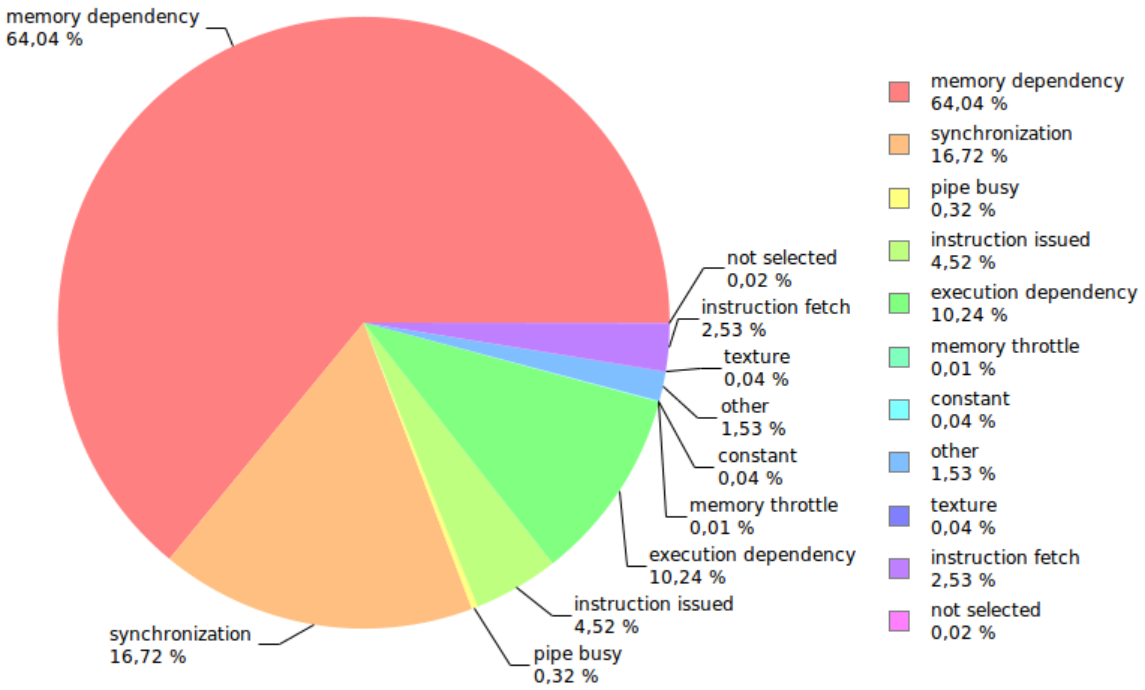
Πίνακας 7.1: Αποτελέσματα του nvprof

Time(%)	**Name**
98.31%	cuMemcpyDtoHAsync
1.06%	cuDevicePrimaryCtxRetain
0.24%	cuLaunchKernel
0.14%	cuMemHostAlloc
0.12%	cuMemcpyHtoDAsync
0.06%	cuMemFreeHost
0.04%	cuStreamSynchronize
0.02%	cuModuleLoadData
0.01%	cuEventRecord
0.01%	cuMemAlloc
0.01%	cuEventSynchronize
0.00%	cuMemAllocHost
0.00%	cuEventCreate
0.00%	cuStreamCreate
0.00%	cuModuleGetFunction
0.00%	cuCtxSetCurrent
0.00%	cuDeviceGetCount
0.00%	cuMemFree
0.00%	cuDeviceGet
0.00%	cuDeviceComputeCapability
0.00%	cuCtxGetDevice

Στον πίνακα **7.1** παρουσιάζονται τα αποτελέσματα της εκτέλεσης της εντολής nvprof για το παράλληλο πρόγραμμα που κατασκευάσαμε. Παρατηρούμε ότι το 98.31% του χρόνου «σπαταλάται» στην αντιγραφή δεδομένων από την κάρτα γραφικών στην κεντρική μνήμη (cuMemcpyDtoHAsync) ενώ μόνο στο 0.24% του χρόνου για υπολογισμούς στη GPU. Στο σχήμα 14 από το NVIDIA Visual Profiler αποτυπώνεται καθαρά και οπτικά η ανάλυση από τον πίνακα **7.1**.



Σχήμα 14: Χρονοσειρά εκτέλεσης από το NVIDIA Visual Profiler



Σχήμα 15: Ανάλυση εκτέλεσης από το NVIDIA Visual Profiler

Στο σχήμα 15 παρατηρούμε τις εξαρτήσεις που έχουν τα δεδομένα, όπως είδαμε θεωρητικά στο προηγούμενο κεφάλαιο.

Αξιολόγηση

Η αντικειμενική μέτρηση μιας παράλληλης μεθευρετικής μεθόδου είναι η εύρεση μιας «καλής» λύσης σε «λογικό» χρόνο (Alba, Luque, and Nesmachnow 2013). Έτσι, η επιλογή μετρικών απόδοσης για τα πειράματα περιλαμβάνουν, απαραίτητα, τόσο την ποιότητα της λύσης, όσο και την υπολογιστική προσπάθεια. Η σωστή μέτρηση της απόδοσης ενός αλγορίθμου είναι ένα κρίσιμο ζήτημα ώστε να γίνουν οι σωστές συγκρίσεις των παράλληλων μεθευρετικών μεθόδων και να εξαχθούν ασφαλή συμπεράσματα. Υπάρχει μεγάλος αριθμός μετρικών για τις παράλληλες μεθευρετικές μεθόδους. Η πιο σημαντική και χρησιμοποιούμενη μέτρηση είναι η **επιτάχυνση** s_m (speedup). Αν δηλώσουμε ως T_m το χρόνο εκτέλεσης για έναν αλγόριθμο που χρησιμοποιεί m επεξεργαστές, η επιτάχυνση είναι ο λόγος μεταξύ του, μεγαλύτερου, χρόνου εκτέλεσης σε έναν επεξεργαστή T_1 και του, μικρότερου, χρόνου εκτέλεσης σε m επεξεργαστές T_m:

$$s_m = \frac{T_1}{T_m} \tag{1}$$

Άλλη μετρική είναι η **αποδοτικότητα** e_m (efficiency), ουσιαστικά μια κανονικοποίηση της επιτάχυνσης. Έτσι, όταν $e_m = 1$ έχουμε γραμμική επιτάχυνση (linear speedup), όταν $e_m < 1$ έχουμε υπογραμμική επιτάχυνση (sublinear speedup) και όταν $e_m > 1$ έχουμε υπεργραμμική επιτάχυνση (superlinear speedup):

$$e_m = \frac{s_m}{m} \tag{2}$$

Τέλος, υπάρχει και η μετρική **Karp-Flatt** f_m. Ιδανικά, το κλάσμα αυτό πρέπει να μένει σταθερό για έναν αλγόριθμο. Αν η τιμή της επιτάχυνσης είναι μικρή μπορούμε να συμπεράνουμε ότι το αποτέλεσμα είναι καλό αν το f_m παραμένει σταθερό για διαφορετικές τιμές του m, αφού η έλλειψη αποδοτικότητας οφείλεται στον περιορισμένο παραλληλισμού του προγράμματος. Από την άλλη, αν το f_m αυξάνει ελαφρά αυτό σημαίνει ότι η παραλληλοποίηση είναι πολύ λεπτομερής. Αν, τέλος, σημειωθεί μείωση του f_m όσο αυξάνει το m τότε το f_m μπορεί να πάρει αρνητική τιμή και σημαίνει ότι έχουμε υπεργραμμική επιτάχυνση:

$$f_m = \frac{1/s_m - 1/m}{1 - 1/m} \tag{3}$$

Στον πίνακα **7.2** παρατηρούμε ότι χρησιμοποιώντας GNU Fortran και το μοντέλο OpenMP η **επιτάχυνση** είναι περίπου $3,3$, δηλαδή πετυχαίνουμε $3,3$ φορές καλύτερους χρόνους και η **αποδοτικότητα** είναι περίπου $0,4$ δηλαδή έχουμε υπογραμμική επιτάχυνση.

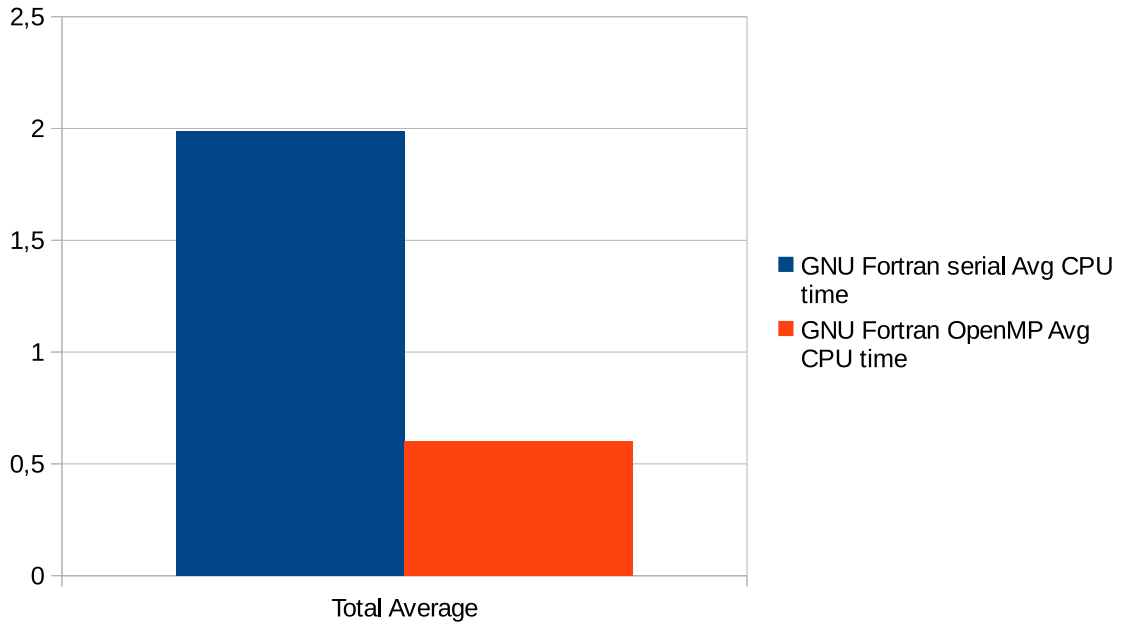
Στον πίνακα **7.3** παρατηρούμε ότι χρησιμοποιώντας PGI Fortran το σειριακό πρόγραμμα επιταχύνεται κατά περίπου $1,7$ φορές.

Στον πίνακα **7.4** παρατηρούμε ότι χρησιμοποιώντας PGI Fortran και το μοντέλο OpenMP, με 4 νήματα, όμως, λόγω περιορισμού της ακαδημαϊκής έκδοσης, πετυχαίνουμε **επιτάχυνση** είναι περίπου $3,2$, δηλαδή πετυχαίνουμε $3,2$ φορές καλύτερους χρόνους και η **αποδοτικότητα** είναι περίπου $0,8$ δηλαδή έχουμε μεν υπογραμμική επιτάχυνση αλλά σχεδόν διπλάσια αποδοτικότητα από ότι με τον GNU compiler. Αυτό μας δίνει το δικαίωμα να πιθανολογούμε ότι με 8 νήματα θα πετυχαίναμε ακόμα καλύτερη επιτάχυνση στον PGI Fortran compiler.

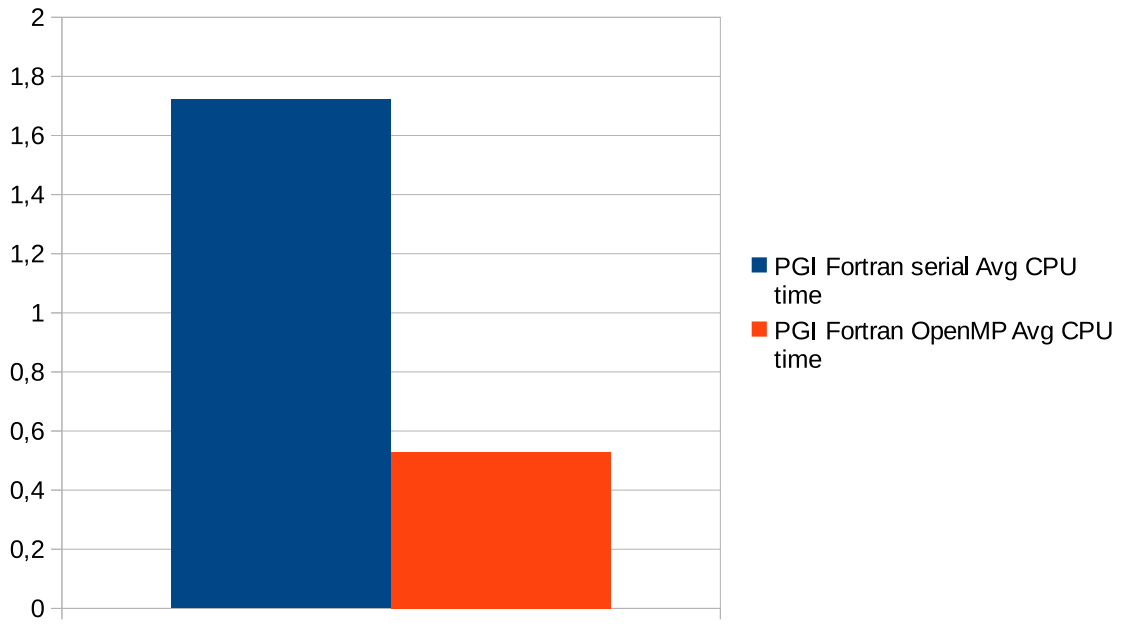
Στον πίνακα **7.5** παρατηρούμε ότι χρησιμοποιώντας PGI Fortran και το μοντέλο OpenACC πετυχαίνουμε σχεδόν την ίδια απόδοση με το OpenMP με 4 νήματα. Ουσιαστικά, δεν κερδίζουμε κάτι καλύτερο από το OpenACC. Σε αυτό ευθύνεται η φύση του προβλήματος, το οποίο δεν έχει πολλούς υπολογισμούς αλλά πολλές μεταφορές δεδομένων, κάτι που δεν είναι το δυνατό σημείο του OpenACC, αλλά ίσως και ο δίαυλος PCI Express x2 που διαθέτει ο

Πίνακας 7.2: Serial vs OpenMP in GNU Fortran (8 threads)

Instance Set	Instance No.	GNU Fortran serial	GNU Fortran OpenMP		
		Avg CPU time	Avg CPU time	Speedup	Efficiency
1	1	1,99833345	0,624333203		
	2	1,99518514	0,612259269		
	3	2,01496291	0,627777755		
	4	2,00170374	0,622148216		
	5	1,99807417	0,652777851		
	6	2,05281496	0,624777675		
	7	1,98466659	0,613333285		
	8	2,0501852	0,610629618		
	9	1,98607385	0,618444383		
	10	1,99929607	0,613407493		
	Average	*2,008129608*	*0,6219888748*	*3,2285619395*	*0,4035702424*
2	1	1,94240749	0,578740776		
	2	2,00774097	0,596333265		
	3	2,01370382	0,61637044		
	4	1,99585199	0,59440738		
	5	1,95507383	0,588740766		
	6	2,01359248	0,588444471		
	7	1,95511138	0,598185241		
	8	2,03825927	0,58937037		
	9	2,01329589	0,584296346		
	10	1,96892607	0,586814821		
	Average	*1,990396319*	*0,5921703876*	*3,3611885374*	*0,4201485672*
3	1	1,95251846	0,602148116		
	2	1,9962225	0,580666602		
	3	1,99625909	0,5865556		
	4	1,99233329	0,593370318		
	5	2,00422215	0,586370409		
	6	1,972	0,581962943		
	7	1,96114826	0,594148219		
	8	1,98962998	0,590629697		
	9	1,9601481	0,585629702		
	10	1,96840727	0,587000012		
	Average	*1,97928891*	*0,5888481618*	*3,3612891037*	*0,420161138*
4	1	1,97962976	0,610444486		
	2	2,01566648	0,591148078		
	3	1,96329606	0,594185174		
	4	1,97818518	0,593370378		
	5	1,97274089	0,65474081		
	6	2,00681472	0,590740681		
	7	1,9710741	0,612222254		
	8	1,96725917	0,594185174		
	9	1,98025942	0,593444407		
	10	1,98266673	0,602666736		
	Average	*1,981759251*	*0,6037148178*	*3,2826082656*	*0,4103260332*
	Total Average	**1,989893522**	**0,6016805605**	**3,3072258814**	**0,4134032352**



Σχήμα 16: Χρόνοι εκτέλεσης (σε sec) GNU Fortran



Σχήμα 17: Χρόνοι εκτέλεσης (σε sec) PGI Fortran

Πίνακας 7.3: Serial GNU Fortran vs Serial PGI Fortran

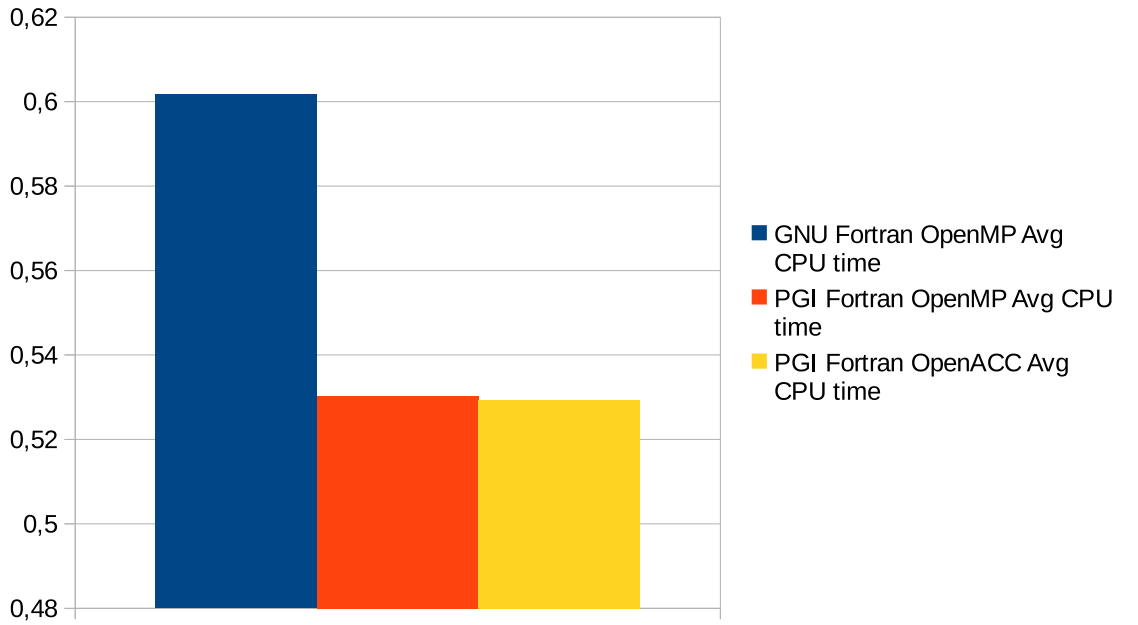
Instance Set	Instance No.	GNU Fortran serial	PGI Fortran serial	
		Avg CPU time	Avg CPU time	Speedup
1	1	1,99833345	1,730341	
	2	1,99518514	1,729662	
	3	2,01496291	1,756054	
	4	2,00170374	1,728637	
	5	1,99807417	1,761593	
	6	2,05281496	1,715436	
	7	1,98466659	1,767645	
	8	2,0501852	1,729966	
	9	1,98607385	1,727319	
	10	1,99929607	1,729002	
	Average	*2,008129608*	*1,7375655*	*1,1557144798*
2	1	1,94240749	1,705843	
	2	2,00774097	1,68514	
	3	2,01370382	1,687488	
	4	1,99585199	1,678265	
	5	1,95507383	1,684555	
	6	2,01359248	1,689215	
	7	1,95511138	1,698785	
	8	2,03825927	1,684837	
	9	2,01329589	1,685136	
	10	1,96892607	1,690641	
	Average	*1,990396319*	*1,6889905*	*1,1784532352*
3	1	1,95251846	1,746872	
	2	1,9962225	1,708481	
	3	1,99625909	1,712654	
	4	1,99233329	1,74453	
	5	2,00422215	1,714071	
	6	1,972	1,740911	
	7	1,96114826	1,749253	
	8	1,98962998	1,745579	
	9	1,9601481	1,702131	
	10	1,96840727	1,745906	
	Average	*1,97928891*	*1,7310388*	*1,1434110605*
4	1	1,97962976	1,73383	
	2	2,01566648	1,760165	
	3	1,96329606	1,711192	
	4	1,97818518	1,720772	
	5	1,97274089	1,758136	
	6	2,00681472	1,713277	
	7	1,9710741	1,725977	
	8	1,96725917	1,731262	
	9	1,98025942	1,72455	
	10	1,98266673	1,722716	
	Average	*1,981759251*	*1,7301877*	*1,1454013059*
	Total Average	**1,989893522**	**1,721945625**	**1,1556076412**

Πίνακας 7.4: Serial vs OpenMP in PGI Fortran (4 threads)

Instance Set	Instance No.	PGI Fortran serial	PGI Fortran OpenMP		
		Avg CPU time	Avg CPU time	Speedup	Efficiency
1	1	1,730341	0,5281321		
	2	1,729662	0,5381887		
	3	1,756054	0,5301083		
	4	1,728637	0,5170892		
	5	1,761593	0,522432		
	6	1,715436	0,5149528		
	7	1,767645	0,5097229		
	8	1,729966	0,5431371		
	9	1,727319	0,5165958		
	10	1,729002	0,5368447		
	Average	*1,7375655*	*0,52572036*	*3,3051135779*	*0,8262783945*
2	1	1,705843	0,538693		
	2	1,68514	0,5192302		
	3	1,687488	0,5146535		
	4	1,678265	0,5045097		
	5	1,684555	0,5573312		
	6	1,689215	0,6164681		
	7	1,698785	0,5079927		
	8	1,684837	0,5312083		
	9	1,685136	0,5525261		
	10	1,690641	0,5049966		
	Average	*1,6889905*	*0,53476094*	*3,1584028931*	*0,7896007233*
3	1	1,746872	0,5205088		
	2	1,708481	0,5420012		
	3	1,712654	0,5303211		
	4	1,74453	0,5165843		
	5	1,714071	0,5335916		
	6	1,740911	0,5085733		
	7	1,749253	0,5207612		
	8	1,745579	0,5529506		
	9	1,702131	0,5199752		
	10	1,745906	0,5019776		
	Average	*1,7310388*	*0,52472449*	*3,298947987*	*0,8247369967*
4	1	1,73383	0,5186781		
	2	1,760165	0,6044075		
	3	1,711192	0,5070056		
	4	1,720772	0,514464		
	5	1,758136	0,5526323		
	6	1,713277	0,5788596		
	7	1,725977	0,5105294		
	8	1,731262	0,5068787		
	9	1,72455	0,5053807		
	10	1,722716	0,5592744		
	Average	*1,7301877*	*0,53581103*	*3,2291005655*	*0,8072751414*
	Total Average	**1,721945625**	**0,530254205**	**3,2473964539**	***0,8118491135***

Πίνακας 7.5: Serial vs OpenMP vs OpenACC in PGI Fortran

Instance Set	Instance No.	PGI Fortran serial	PGI Fortran OpenMP		PGI Fortran OpenACC		
		Avg CPU time	Avg CPU time	Speedup	Avg CPU time	Speedup vs serial	Speedup vs OpenMP
1	1	1,730341	0,5281321		0,524456		
	2	1,729662	0,5381887		0,5239334		
	3	1,756054	0,5301083		0,5243782		
	4	1,728637	0,5170892		0,5242513		
	5	1,761593	0,522432		0,5244823		
	6	1,715436	0,5149528		0,5237587		
	7	1,767645	0,5097229		0,5241603		
	8	1,729966	0,5431371		0,5237933		
	9	1,727319	0,5165958		0,5244889		
	10	1,729002	0,5368447		0,5245594		
	Average	*1,7375655*	*0,52572036*	*3,3051135779*	*0,52422618*	*3,3145340052*	*1,0028502583*
2	1	1,705843	0,538693		0,5280248		
	2	1,68514	0,5192302		0,5282099		
	3	1,687488	0,5146535		0,5276599		
	4	1,678265	0,5045097		0,5273229		
	5	1,684555	0,5573312		0,5280693		
	6	1,689215	0,6164681		0,5265282		
	7	1,698785	0,5079927		0,5277112		
	8	1,684837	0,5312083		0,5273771		
	9	1,685136	0,5525261		0,5266654		
	10	1,690641	0,5049966		0,5267273		
	Average	*1,6889905*	*0,53476094*	*3,1584028931*	*0,5274296*	*3,2023051038*	*1,01390013*
3	1	1,746872	0,5205088		0,5278185		
	2	1,708481	0,5420012		0,5261192		
	3	1,712654	0,5303211		0,5268579		
	4	1,74453	0,5165843		0,5256334		
	5	1,714071	0,5335916		0,5269288		
	6	1,740911	0,5085733		0,5284846		
	7	1,749253	0,5207612		0,5256684		
	8	1,745579	0,5529506		0,526413		
	9	1,702131	0,5199752		0,5257337		
	10	1,745906	0,5019776		0,5279091		
	Average	*1,7310388*	*0,52472449*	*3,298947987*	*0,52675666*	*3,2862210038*	*0,9961421086*
4	1	1,73383	0,5186781		0,5401573		
	2	1,760165	0,6044075		0,5402232		
	3	1,711192	0,5070056		0,538147		
	4	1,720772	0,514464		0,539937		
	5	1,758136	0,5526323		0,5390256		
	6	1,713277	0,5788596		0,5402011		
	7	1,725977	0,5105294		0,5380191		
	8	1,731262	0,5068787		0,5382425		
	9	1,72455	0,5053807		0,5385788		
	10	1,722716	0,5592744		0,5379466		
	Average	*1,7301877*	*0,53581103*	*3,2291005655*	0,53904782		
	Total Average	**1,721945625**	**0,530254205**	**3,2473964539**	**0,529365065**	**3,2528508941**	**1,0016796348**



Σχήμα 18: Χρόνοι εκτέλεσης (σε sec) GNU Fortran OpenMP (8 threads) vs PGI Fortran OpenMP (4 threads) vs PGI Fortran OpenACC

υπολογιστής που εκτελέστηκαν τα πειράματα που είναι πιο αργός από το δίαυλο PCI Express x3 που διαθέτει η κάρτα γραφικών.

Η μετάφραση εκτελείται με την εξής εντολή:

```
pgfortran -Mipa=inline,fast -Mfprelaxed -O3 -fast -Minfo=all
-Mpreprocess -ta=tesla:cc50
```

Να τονίσουμε ότι σε όλα τα πειράματα τα αποτελέσματα είναι ακριβώς τα ίδια με αυτά της σειριακής λύσης, παρόλο που υπήρχε πάντοτε το στοιχείο της τυχαιότητας αφού η παράλληλη εκτέλεση σε νήματα είτε της CPU, είτε της GPU είναι μια διαδικασία δυναμική, στοχαστική που δεν μπορεί να επαναληφθεί με τον ίδιο ακριβώς τρόπο.

Τέλος, παρόλο που πετύχαμε πολύ καλύτερους χρόνους παραλληλοποιώντας τον αλγόριθμο, δεν βρέθηκαν καλύτερες λύσεις αφού δεν επεκτάθηκε η αναζήτηση.

ΚΕΦΑΛΑΙΟ 8

Συμπεράσματα και μελλοντικές κατευθύνσεις

Στην παρούσα εργασία καταφέραμε να παραλληλοποιήσουμε, με πολύ καλά αποτελέσματα, ένα NP-Hard πρόβλημα διαχείρισης αποθεμάτων, χρησιμοποιώντας τα μοντέλα παράλληλου προγραμματισμού OpenMP και OpenACC. Δείξαμε ότι, ακόμα και χωρίς να προχωρήσουμε στην εφαρμογή ενός παράλληλου μεθευρετικού αλγορίθμου, μπορούμε με τις τεχνολογίες αυτές να επιταχύνουμε το χρόνο εκτέλεσης, σε αρχικό στάδιο, χωρίς όμως την εύρεση καλύτερων λύσεων αφού η αναζήτηση δεν επεκτάθηκε περαιτέρω.

Μετά από τα διάφορα πειράματα που εκτελέσαμε, καταλήξαμε στο συμπέρασμα, ότι η τεχνολογία OpenMP είναι, πλέον, αρκετά ώριμη, φορητή, απλή και μπορεί να εφαρμοστεί σχετικά εύκολα σε σειριακά προγράμματα, κυρίως για περιβάλλοντα με πολυπύρηνους επεξεργαστές. Επίσης υποστηρίζει, από την έκδοση OpenMP 4.0 παραλληλοποίηση σε GPUs και υποστηρίζεται από μια πληθώρα compilers εμπορικών ή μη. Από την άλλη είναι δύσκολο να εκσφαλματωθεί.

Η τεχνολογία OpenACC είναι επίσης αναπτυσσόμενη και μπορεί να χρησιμοποιήσει τη GPU για να εκτελέσει παράλληλους υπολογισμούς με οδηγίες στον compiler, με παρόμοιο τρόπο με το OpenMP. Το OpenACC υποστηρίζεται κυρίως από κάρτες γραφικών της NVIDIA με compute capability πάνω από 2.0 και κυρίως από εμπορικούς compilers C/C++, Fortran. Το κύριο πρόβλημα του OpenACC είναι η ανωριμότητά του: ακόμη και στον PGI compiler που χρησιμοποιήσαμε δεν έχουν ενσωματωθεί τα νέα χαρακτηριστικά του OpenACC API. Επίσης, για να αποδώσει τα μέγιστα πρέπει να σχεδιαστεί το πρόβλημα με τέτοιο τρόπο ώστε να ελαχιστοποιηθούν οι αναφορές στη μνήμη, αφού σε αντίθετη περίπτωση το κόστος μεταφοράς δεδομένων στη μνήμη της GPU είναι τεράστιο και εξαρτάται από το δίαυλο επικοινωνίας της μητρικής με τη GPU.

Στον τομέα της ετερογενούς παραλληλοποίησης με την τεχνολογία OpenACC για παραλληλοποίηση μεθευρετικών αλγορίθμων έχουν γίνει ελάχιστες έρευνες. Ιδιαίτερα για παραλληλοποίηση προβλημάτων διαχείρισης αποθεμάτων με την τεχνολογία OpenACC δεν εντοπίσαμε

άλλη έρευνα. Στο πρόβλημα που παραλληλοποιήσαμε στην παρούσα εργασία η τεχνολογία OpenACC απέδωσε ελάχιστα καλύτερα από το OpenMP. Το OpenACC αποδίδει καλύτερα σε προβλήματα compute-intensive, χαρακτηριστικό που δε διέθετε το πρόβλημά μας. Επίσης, ανασταλτικός παράγοντας για την καλύτερη απόδοση είτε σε OpenMP, είτε σε OpenACC αποτέλεσε η αλληλεξάρτηση των δεδομένων.

Πετύχαμε, παρ' όλα αυτά, με χρήση των OpenMP, OpenACC να επιταχύνουμε το χρόνο εκτέλεσης κατά περίπου 3,3 φορές. Στην εργασία μας χρησιμοποιήσαμε ουσιαστικά τη στρατηγική παραλληλισμού χαμηλού επιπέδου (Crainic and Toulouse 2003) αφού επιταχύναμε αποκλειστικά τους υπολογισμούς χωρίς να εξερευνήσουμε καλύτερα το χώρο λύσεων. Αναμενόμενο αποτέλεσμα αποτέλεσε η μη εύρεση καλύτερων λύσεων.

Μελλοντικά, θα μπορούσαμε να διερευνήσουμε τη διαδικασία shaking του αλγορίθμου VNS για να επιτύχουμε καλύτερα αποτελέσματα. Μπορούμε, ακόμη, να εφαρμόσουμε την τεχνολογία OpenACC και σε άλλα προβλήματα διαχείρισης αποθεμάτων, περισσότερο compute-intensive. Επιπροσθέτως, θα είχε ενδιαφέρον η παραλληλοποίηση με CUDA, ώστε να αποτυπωθεί η διαφορά σε απόδοση μεταξύ CUDA, OpenACC, OpenMP. Επίσης, θα μπορούσε να συνδυαστεί, με χρήση και άλλων μονάδων GPU είτε στον ίδιο υπολογιστή είτε σε άλλους στο δίκτυο, με τις τεχνολογίες MPI, OpenMP, OpenACC. Η χρήση MPI, OpenMP, OpenACC θα μπορούσε να δώσει μια νέα διάσταση στην παραλληλοποίηση μεθευρετικών αλγορίθμων σε ετερογενή συστήματα για προβλήματα διαχείρισης αποθεμάτων ή ακόμα και άλλα NP-Hard προβλήματα βελτιστοποίησης, όταν χρειαζόμαστε να μειώσουμε κυρίως το χρόνο εκτέλεσης.

Τέλος, η σχεδίαση από την αρχή της λύσης ώστε να εφαρμοστεί στο πρόβλημα κάποιος παράλληλος μεθευρετικός αλγόριθμος, όπως αυτοί που παρουσιάστηκαν στο πέμπτο κεφάλαιο της παρούσας εργασίας, ή ακόμα και κάποιος παράλληλος υπερευρετικός αλγόριθμος (parallel hyperheuristic) πιθανότατα θα έφερνε καλύτερα αποτελέσματα σε τέτοιου είδους προβλήματα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Alba, E., G. Luque, and S. Nesmachnow (2013). Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research 20*(1), 1–48.

Alba, E., E.-G. Talbi, G. Luque, and N. Melab (2005). *Metaheuristics and Parallelism*, pp. 79–103. John Wiley & Sons, Inc.

Barney, B. (2016a). Introduction to Parallel Computing. Available at `https://computing.llnl.gov/tutorials/parallel_comp/images/sharedMemoryModel.gif` (last accessed June 12, 2016).

Barney, B. (2016b). OpenMP. Available at `https://computing.llnl.gov/tutorials/openMP/images/fork_join2.gif` (last accessed June 12, 2016).

Blum, C. and A. Roli (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM COMPUTING SURVEYS*, 268–308.

Crainic, T. G., M. Gendreau, P. Hansen, and N. Mladenović (2004). Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics 10*(3), 293–314.

Crainic, T. G. and M. Toulouse (2003). *Handbook of Metaheuristics*, Chapter Parallel Strategies for Meta-Heuristics, pp. 475–513. Boston, MA: Springer US.

Crainic, T. G. and M. Toulouse (2010). Parallel meta-heuristics. In *Handbook of metaheuristics*, pp. 497–541. Springer US.

Davidović, T. and T. G. Crainic (2012). Mpi parallelization of variable neighborhood search. *Electronic Notes in Discrete Mathematics 39*, 241–248.

Figueira, G., M. O. Santos, and B. Almada-Lobo (2013). A hybrid {VNS} approach for the short-term production planning and scheduling: A case study in the pulp and paper industry. *Computers & Operations Research 40*(7), 1804 – 1818.

García-López, F., B. Melián-Batista, J. A. Moreno-Pérez, and J. M. Moreno-Vega (2003). Parallelization of the scatter search for the p-median problem. *Parallel Computing 29*(5), 575 – 589. Parallel computing in logistics.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research 13*, 533–549.

Hansen, P., B. Jaumard, N. Mladenović, and A. Parreira (2000). Variable neighbourhood search for maximum weight satisfiability problem. *Cahiers du GERAD* (G-2000-62).

Hansen, P., N. Mladenović, and J. A. Moreno Pérez (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research 175*(1), 367–407.

Helmrich, M. J. R., R. Jans, W. van den Heuvel, and A. P. Wagelmans (2014). Economic lot-sizing with remanufacturing: complexity and efficient formulations. *IIE Transactions 46*(1), 67–86.

Hertz, A. and M. Widmer (2003). Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research 151*(2), 247 – 252. Meta-heuristics in combinatorial optimization.

HPC2N, H. P. C. C. N. (2016). Beginner's Guide. Available at `https://www.hpc2n.umu.se/sites/default/files/images/cluster.png` (last accessed June 12, 2016).

Kotelnikov, D. A. (2004). Linux Clusters. Available at `http://coewww.rutgers.edu/linux/images/Mpassing.gif` (last accessed June 12, 2016).

Li, Y., J. Chen, and X. Cai (2006). Uncapacitated production planning with multiple product types, returned product remanufacturing, and demand substitution. *OR Spectrum 28*(1), 101–125.

Luke, S. (2013). *Essentials of Metaheuristics* (second ed.). Lulu. Available for free at `https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf` (last accessed June 12, 2016).

Margaritis, K. (2015). Educational material in parallel computing. Available at `https://sites.google.com/site/kgmargaritis/home/yliko` (last accessed June 4, 2016).

Mladenović, N. and P. Hansen (1997). Variable neighborhood search. *Computers & Operations Research 24*(11), 1097 – 1100.

Nicholas Metropolis, S. U. (1949). The monte carlo method. *Journal of the American Statistical Association 44*(247), 335–341.

Pérez, J. A. M., P. Hansen, and N. Mladenović (2005). *Parallel Variable Neighborhood Search*, pp. 247–266. John Wiley & Sons, Inc.

Rosenberg, O. (2011). OpenCL Overview. Available at `http://haifux.org/lectures/267/OpenCL_Overview.pdf` (last accessed June 12, 2016).

Sahling, F. (2013). A column-generation approach for a short-term production planning problem in closed-loop supply chains. *Business Research 6*(1), 55–75.

Schulz, T. (2011). A new silver-meal based heuristic for the single-item dynamic lot sizing problem with returns and remanufacturing. *International Journal of Production Research 49*(9), 2519–2533.

Sifaleras, A. and I. Konstantaras (2015a). General variable neighborhood search for the multi-product dynamic lot sizing problem in closed-loop supply chain. *Electronic Notes in Discrete Mathematics 47*, 69 – 76. The 3rd International Conference on Variable Neighborhood Search (VNS'14).

Sifaleras, A. and I. Konstantaras (2015b). Variable neighborhood descent heuristic for solving reverse logistics multi-item dynamic lot-sizing problems. *Computers & Operations Research*, –.

Sifaleras, A., I. Konstantaras, and N. Mladenović (2015). Variable neighborhood search for the economic lot sizing problem with product returns and recovery. *International Journal of Production Economics 160*, 133 – 143.

Sörensen, K. and F. W. Glover (2013). Metaheuristics. In *Encyclopedia of Operations Research and Management Science*, pp. 960–970. Springer US.

Süß, M. and C. Leopold (2008). *OpenMP Shared Memory Parallel Programming: International Workshops, IWOMP 2005 and IWOMP 2006, Eugene, OR, USA, June 1-4, 2005, Reims, France, June 12-15, 2006. Proceedings*, Chapter Common Mistakes in OpenMP and How to Avoid Them, pp. 312–323. Berlin, Heidelberg: Springer Berlin Heidelberg.

Tosaka (2008). CUDA processing flow. Available at `https://upload.wikimedia.org/wikipedia/commons/5/59/CUDA_processing_flow_%28En%29.PNG` (last accessed June 12, 2016).