Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων Και Μηχανικων Υπολογιστων

Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Αλγόριθμοι δυναμικής δρομολόγησης εφαρμογών που περιέχουν φωλιασμένους βρόχους σε ετερογενή δίκτυα υπολογιστών

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Ιωάννης Ρυακιωτάκης

Διπλωματούχος Ηλεκτρονικός Μηχανικός

Αθήνα, Οκτώβριος 2008

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Αλγόριθμοι δυναμικής δρομολόγησης εφαρμογών που περιέχουν φωλιασμένους βρόχους σε ετερογενή δίκτυα υπολογιστών

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Ιωάννης Ρυακιωτάκης
Ηλεκτρονικός Μηχανικός, Πανεπιστημίο του Πλύμουθ, Μ.Βρετανία (2000)

Συμβουλευτική Επιτροπή: Γεώργιος Παπακωνσταντίνου
Παναγιώτης Τσανάκας
Νεκτάριος Κοζύρης

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την .....

.....	.....	.....
Γεώργιος Παπακωνσταντίνου	Παναγιώτης Τσανάκας	Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκ. Καθηγητής Ε.Μ.Π.
.....	.....	.....
Α. Γ. Σταφυλοπάτης	Γ. Στασινόπουλος	Σ. Κόλλιας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

.....
Θεόδωρος Ανδρόνικος
Λέκτορας Ιόνιο
Πανεπιστήμιο

Αθήνα, Οκτώβριος 2008

.....

Ιωάννης Ρυακιωτάκης

Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π

Περίληψη

Ο στόχος της έρευνας ήταν η ανεύρεση αποδοτικών αλγορίθμων για την παραλληλοποίηση φωλιασμένων βρόχων που περιέχουν εξαρτήσεις σε κατανεμημένα συστήματα (δίκτυα υπολογιστών). Τα κατανεμημένα συστήματα είναι ένα ελκυστικό περιβάλλον εκτέλεσης παράλληλων εφαρμογών λόγω του χαμηλού κόστους συγκρότησης, σε σχέση με τα παραδοσιακά παράλληλα μηχανήματα. Ωστόσο, πολλές φορές χαρακτηρίζονται από ανομοιογένεια (heterogeneity) σε διάφορα επίπεδα, όπως ανομοιογένεια στους τύπους των υπολογιστών και στο δίκτυο που τα συνδέει και ανομοιογένεια και μεταβλητότητα στο φόρτο εργασίας (workload variation). Τα ιδιαίτερα αυτά χαρακτηριστικά κάνουν επιτακτική την ανάγκη ανάπτυξης νέων αλγορίθμων δρομολόγησης. Η έρευνα επικεντρώνεται στους δυναμικούς αλγόριθμους γιατί έχουν την δυνατότητα να προσαρμόζονται στις συνθήκες που επικρατούν στο περιβάλλον που εκτελούνται, δίνοντας έτσι την δυνατότητα να αντιμετωπίσουμε την ανομοιογένεια των πόρων και την διακύμανση του φόρτου εργασίας. Με την ολοκλήρωση της διατριβής παρουσιάστηκε μια σειρά νέων αλγορίθμων για την δυναμική δρομολόγηση εφαρμογών που περιέχουν φωλιασμένους βρόχους σε αρχιτεκτονικές κατανεμημένης μνήμης. Οι προτεινόμενοι αλγόριθμοι υλοποιήθηκαν, επαληθεύτηκαν και αξιολογήθηκαν με εξαντλητικά πειράματα.

Abstract

This research was focused in the establishment of efficient alorithms for the parallelization of nested loops containing dependencies in distributed systems (networks of computers). Distributed systems offer an attractive environment for the execution of parallel applications because of the cost benefit that they offer when compared to the traditional parallel machines. However, distributed systems are characterized by various levels of heterogeneity, heterogeneity in the machine types (CPU, memory, architecture), in the type and variation of the workload, heterogeneity of the interconnection links (network heterogeneity). These characteristics necessitate the developement of new scheduling algorithms. This thesis focuses on dynamic algorithms because of their inherent ability to adapt to changing execution environments, addressing in that way the heterogeneity of the resources and the workload variation. With the completion of this thesis, a series of new dynamic scheduling algorithms was presented, for applications containing nested dependent loops in distributed systems. The proposed algorithms were implemented, verified and evaluated experimentally.

Περιεχόμενα

Αντί προλόγου

Σε αυτό το σημείο, πριν προχωρήσουμε στο επιστημονικό μέρος του κειμένου, θα ήθελα να αναφερθώ στους ανθρώπους που συνέβαλαν στην ολοκλήρωση αυτής της διδακτορικής διατριβής.

Η διατριβή εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων, της Σχολής Ηλεκτρολόγων Μηχανικών, του Εθνικού Μετσόβιου Πολυτεχνείου. Το εργαστήριο αυτό έγινε κατά την διάρκεια των τελευταίων χρόνων το δεύτερο σπίτι μου, ο τόπος όπου μπορούσα να δουλέψω, να συζητήσω, να συναντήσω τους συνεργάτες και τους φίλους μου. Αισθάνομαι ιδιαίτερα τυχερός πού μου δόθηκε η ευκαιρία να εκπονήσω την διδακτορική μου διατριβή σε ένα τέτοιο ευχάριστο και οικείο περιβάλλων, μακριά από τον ανταγωνισμό και την καχυποψία που μπορεί κανείς να συναντήσει στον στίβο της επιστημονικής έρευνας. Για την διαμόρφωση αυτού του ιδανικού περιβάλλοντος εργασίας αξίζουν συγχαρητήρια στα μέλη ΔΕΠ του εργαστηρίου καθ. Γ. Παπακωνσταντίνου, καθ. Π. Τσανάκα και επ.καθ. Ν. Κοζύρη, αλλά και σε όλους τους μεταπτυχιακούς φοιτητές που το απαρτίζουν.

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον επιβλέποντα μου, καθηγητή Γ. Παπακωνσταντίνου για την σημαντικότατη επιστημονική του καθοδήγηση, αλλά κυρίως για την ακεραιότητα του, το ήθος, την αφοσίωση και την εκτίμηση με την οποία περιβάλλει όλους ανεξαιρέτως τους φοιτητές του. Θέλω ακόμα να ευχαριστήσω τον καθηγητή Π.Τσανάκα για την βοήθεια πού μου προσέφερε στην επιλογή του θέματος της διατριβής καθώς και για την υποστήριξη του στα πρώτα ερευνητικά μου βήματα. Επίσης θέλω να ευχαριστήσω το τρίτο μέλος της συμβουλευτικής μου επιτροπής, τον αναπληρωτής Καθηγητή Ν.Κοζύρη γιατί είναι πάντα πρόθυμος για συνεργασία και γιατί προσφέρει πολλά στο εργαστήριο.

Νιώθω επίσης την ανάγκη να ευχαριστήσω θερμά τα μέλη της ερευνητικής ομάδας με τα οποία εργάστηκα πολύ στενά τα τελευταία χρόνια, την Διδάκτορα Φ. Κιόρμπα και τον λέκτορα Θ. Ανδρόνικο με οποίους ξενυχτίσαμε πολλές φορές στο εργαστήριο κάνοντας πειράματα και δοκιμάζοντας τις ιδέες μας, όπως επίσης και τον καθηγητή Α. Χρονόπουλο ο οποίος μας αφιέρωσε μεγάλο μέρος του χρόνου του.

Θέλω επίσης να αναφερθώ και να ευχαριστήσω τους διδάκτορες και υποψήφιους διδάκτορες του εργαστηρίου, που δεν είχαμε άμεση ερευνητική επαφή, τους οποίους όμως θεωρώ κάτι παραπάνω από φίλους και συνεργάτες αφού περάσαμε μαζί ατελείωτες ώρες μέσα και έξω από το εργαστήριο, τους Χ. Παυλάτο, Α. Δημόπουλο και Σ. Μαρίνο, επίσης τους Δ. Βουδούρη, Μ. Καλαθά, Γ. Γκούμα και Β. Κούκη καθώς και τα υπόλοιπα παιδιά που είναι ή πέρασαν αυτά τα χρόνια από το εργαστήριο.

Κεφάλαιο 1

Εισαγωγή

Ο στόχος της έρευνας ήταν η ανεύρεση αποδοτικών αλγορίθμων για την παραλληλοποίηση φωλιασμένων βρόχων με ομοιόμορφες εξαρτήσεις (Nested Loops with uniform dependencies) σε κατανεμημένα συστήματα υπολογιστών (distributed systems), είτε αυτά είναι χαλαρά συνδεδεμένα δίκτυα υπολογιστών (Networks of Computers/Workstations), είτε συστοιχίες υπολογιστών (Computer Clusters).

Τα κατανεμημένα συστήματα είναι ένα ελκυστικό περιβάλλον εκτέλεσης παράλληλων εφαρμογών λόγω του χαμηλού κόστους συγκρότησης, σε σύγκριση με τα παραδοσιακά παράλληλα μηχανήματα. Ωστόσο, παρουσιάζουν ορισμένες ιδιαιτερότητες που μειώνουν την απόδοση των αλγορίθμων που είχαν αναπτυχθεί μέχρι τώρα για τις παραδοσιακές παράλληλες πλατφόρμες κοινής/μοιραζόμενης μνήμης. Τα κατανεμημένα συστήματα χαρακτηρίζονται από ανομοιογένεια (heterogeneity) σε διάφορα επίπεδα, όπως στους τύπους των υπολογιστών και στο δίκτυο διασύνδεσης. Επίσης παρουσιάζουν ανομοιογένεια ή μεταβλητότητα στο φόρτο εργασίας (workload variation), λόγω του ότι δεν είναι αφοσιωμένα σε ένα μοναδικό χρήστη (non-dedicated, multi-user systems). Τα χαρακτηριστικά αυτά κάνουν επιτακτική την ανάγκη ανάπτυξης νέων αλγορίθμων που να καλύπτουν τις ιδιαιτερότητες και να αξιοποιούν καλύτερα τις δυνατότητες που προσφέρουν τα κατανεμημένα συστήματα.

Υπάρχουν δυο κατηγορίες αλγορίθμων δρομολόγησης, οι δυναμικοί και οι στατικοί. Η έρευνα στο πλαίσιο αυτής της διατριβής επικεντρώνεται στους δυναμικούς αλγόριθμους, γιατί τα χαρακτηριστικά τους ταιριάζουν καλύτερα στο περιβάλλον εκτέλεσης που μας απασχολεί. Οι αλγόριθμοι αυτοί έχουν την δυνατότητα να προσαρμόζονται στις συνθήκες που επικρατούν στο περιβάλλον που εκτελούνται, δίνοντας έτσι την δυνατότητα να αντιμετωπίσουμε, ως ένα

βαθμό, την ανομοιογένεια και την όποια διακύμανση του φόρτου εργασίας.

Έχουν προταθεί στο παρελθόν δυναμικοί αλγόριθμοι που προορίζονταν κυρίως για ομογενή συστήματα υπολογιστών. Οι αλγόριθμοι που μας απασχολούν από αυτή την κατηγορία είναι η τάξη των αυτόδρομολογούμενων αλγορίθμων (self-scheduling algorithms). Οι αλγόριθμοι αυτοί είχαν αναπτυχθεί αρχικά για μηχανήματα μοιραζόμενης μνήμης (shared-memory multiprocessors). Κάθε επεξεργαστής έπαιρνε από μόνος του μέρος του συνολικού έργου και από εκεί προέκυψε και η ονομασία 'αυτοδρομολόγηση'. Η χρήση των αλγορίθμων αυτών επεκτάθηκε σε αρχιτεκτονικές κατανεμημένης μνήμης (distributed memory multiprocessors) μέσω του μοντέλου συντονιστή-εργάτη (master-worker), στο οποίο κάθε κόμβος (worker) ζητούσε εργασία από ένα κεντρικό κόμβο συντονιστή (Master). Όλοι οι δυναμικοί αλγόριθμοι που έχουν παρουσιαστεί επιλύουν αποκλειστικά προβλήματα φωλιασμένων βρόχων χωρίς εξαρτήσεις (Parallel ή Doall Loops). Στόχος μας ήταν να συμπληρώσουμε αυτό το κενό με τη δημιουργία δυναμικών αλγορίθμων που έχουν τη δυνατότητα να επιλύουν προβλήματα που περιέχουν εξαρτήσεις (Doacross Loops). Η ύπαρξη εξαρτήσεων προσθέτει την ανάγκη για επικοινωνία μεταξύ των επεξεργαστικών στοιχείων με σκοπό την ανταλλαγή δεδομένων. Αυτό επηρεάζει τον αλγόριθμο δρομολόγησης, ειδικά σε δίκτυα στα οποία το κόστος επικοινωνίας είναι υψηλό.

1.1 Συμβολή

Η παρούσα διδακτορική διατριβή παρουσιάζει καινοτόμες μεθόδους παραλληλοποίησης των φωλιασμένων βρόχων που περιέχουν εξαρτήσεις, για διάφορες περιπτώσεις υπολογιστικών συστημάτων. Συγκεκριμένα, η συμβολή της διατριβής συνοψίζεται παρακάτω:

- Μέχρι την ολοκλήρωση της διατριβής παρουσιάστηκε μια σειρά νέων αλγορίθμων για την δυναμική δρομολόγηση εφαρμογών που περιέχουν φωλιασμένους βρόχους σε αρχιτεκτονικές κατανεμημένης μνήμης. Αρχικά παρουσιάζεται η εφαρμογή των αυτοδρομολογούμενων αλγορίθμων για την παραλληλοποίηση μιας πραγματικής επιστημονικής εφαρμογής, η οποία αφορούσε τον προσδιορισμό της τροχιάς ιόντων υδρογόνου και οξυγόνου στην μαγνητόσφαιρα της Γης και είχε στόχο την διευκρίνιση της σχέσης μαγνητικών καταιγίδων - υποκαταιγίδων και την πρόβλεψη του διαστημικού καιρού [C.2.].

- Στην κατεύθυνση των βρόχων που περιέχουν εξαρτήσεις ((DOACROSS loops), αρχικά σχεδιάστηκε και υλοποιήθηκε ένας νέος δυναμικός αλ-

γόριθμος [C.7.], προορισμένος για χαλαρά συνδεδεμένα δίκτυα υπολογιστών. Αυτός ο αλγόριθμος πραγματοποιούσε ανταλλαγές μηνυμάτων αποκλειστικά με την χρήση TCP/IP sockets ώστε να μπορεί να χρησιμοποιηθεί σε οποιοδήποτε TCP/IP δίκτυο υπολογιστών. Ο Αλγόριθμος αυτός αποτελεί τον πρώτο αλγόριθμο στην βιβλιογραφία που συνδυάζει δυναμική δρομολόγηση φωλιασμένων βρόχων που περιέχουν εξαρτήσεις δεδομένων και παραλληλισμό χονδρού κόκκου (coarse-grain parallelism). Δοκιμάστηκε με επιτυχία σε ένα τοπικό δίκτυο υπολογιστών κάτω από διαφορετικά σενάρια λειτουργίας.

- Στην συνέχεια η προσπάθεια επικεντρώθηκε σε συστοιχίες υπολογιστών και προς την κατεύθυνση αυτή τροποποιήθηκαν υπάρχοντες δυναμικοί αυτοδρομολογούμενοι αλγόριθμοι, έτσι ώστε να μπορέσουν να υποστηρίξουν εφαρμογές με εξαρτήσεις δεδομένων [C.6],[J.2.]. Προτάθηκαν επίσης γενικευμένες μέθοδοι-εργαλεία για την υποστήριξη εξαρτήσεων από οποιοδήποτε υπάρχον αλγόριθμο [J.1.]. Ακόμα μέσα σε αυτή την εργασία προτείνεται ένας πρακτικός τελεστής που βελτιώνει την εξισορρόπηση του υπολογιστικού φορτίου και της συνολικής απόδοση των αυτοδρομολογούμενων αλγορίθμων που δεν είχαν δημιουργηθεί αρχικά για ετερογενή, μη-αφοσιωμένα συστήματα υπολογιστών.

- Στην ίδια κατεύθυνση, προτάθηκαν θεωρητικά μοντέλα για την ανεύρεση της βέλτιστης συχνότητας επικοινωνίας μεταξύ των επεξεργαστών για την εκτέλεση των DOACROSS βρόχων σε ετερογενή συστήματα [C.3.],[C.4.],[U.R.1],[U.R.2].

- Επίσης, διερευνήθηκε αναλυτικά η επίδραση της διακύμανσης του εξωγενούς φόρτου εργασίας (exogenous workload), δηλαδή αυτού που δεν οφείλεται στην παράλληλη εφαρμογή. Το εξωγενές φορτίο μοντελοποιήθηκε ως μια στοχαστική ανέλιξη (stochastic process). Με βάση αυτό το μοντέλο αναπτύχθηκε ένας νέο δυναμικός αλγόριθμος [C.5.], πάντα για την επίλυση προβλημάτων που περιέχουν φωλιασμένους βρόχους με ομοιόμορφες εξαρτήσεις, ο οποίος υπερτερεί των προηγούμενων δυναμικών αλγορίθμων κυρίως κάτω από γρήγορα κυμαινόμενο εξωγενές φόρτο.

- Στην προσπάθεια για περαιτέρω βελτίωση των δυναμικών αλγορίθμων προτάθηκε ένας ακόμα αλγόριθμος ([UR3]), ο οποίος αντίθετα με τους προηγούμενους αλγορίθμους δεν βασίζεται στο μοντέλο εργάτη-συντονιστή αλλά ακολουθεί μια κατανεμημένη προσέγγιση. Τα πλεονεκτήματα της κατανεμημένης προσέγγισης, όπως αποδείχθηκε και πειραματικά, περιλαμβάνουν την καλύτερη κατανομή του υπολογιστικού φορτίου σε συστήματα με μεταβαλλόμενο εξωγενές φορτίο, αλλά και την μεγαλύτερη ικανότητα

κλιμάκωσης του αριθμού των επεξεργαστών που μπορούν να χρησιμοποιηθούν αποδοτικά σε σχέση με τους αλγορίθμους που βασίζονται στο μοντέλο συντονιστή-εργάτη.

- Τέλος, διερευνήθηκε η χρήση των δυναμικών αλγορίθμων και για πλατφόρμες επαναπρογραμματιζόμενου υλικού (FPGAs). Δημιουργήθηκε και υλοποιήθηκε σε γλώσσα περιγραφής υλικού verilog, ένα νέος δυναμικός αλγόριθμος [C.1.], ειδικά προσαρμοσμένος για εφαρμογή σε πλατφόρμες επαναπρογραμματιζόμενου υλικού. Η απόδοση του ελέγχθηκε μέσω εξομοίωσης και αποδείχθηκε ικανοποιητική συγκρινόμενη με άλλες υπάρχουσες μεθόδους δρομολόγησης βρόχων με εξαρτήσεις σε επαναπρογραμματιζόμενο υλικό.

Το κύριο βάρος δόθηκε στην οικοδόμηση και μελέτη των θεωρητικών μοντέλων που παρουσιάζονται στα [C.3.],[C.4.],[U.R.1],[U.R.2], καθώς μέσα από αυτή την μελέτη έγινε πιο ξεκάθαρη η επίδραση των επικοινωνιών στην συνολική απόδοση των δυναμικών αλγορίθμων. Επίσης ένα σημαντικό μέρος της διατριβής είναι αφιερωμένο στην μελέτη της διακύμανσης του εξωτερικού φορτίου του συστήματος, κυρίως στις εργασίες [C.5.] και [UR2], όπου παρουσιάζονται αλγόριθμοι ειδικά σχεδιασμένοι για να προσαρμόζονται σε αυτή την διακύμανση.

1.2 Δημοσιεύσεις

Οι δημοσιεύσεις που προέκυψαν στην διάρκεια της Διατριβής:

Διεθνή Περιοδικά

J1. F. M. Ciorba, I. Riakiotakis, T. Andronikos, G. Papakonstantinou and A. T. Chronopoulos, "Enhancing self-scheduling algorithms via synchronization and weighting", *Journal of Parallel and Distributed Computing*, vol 68, no. 2, pp. 246-264, 2008.

J2. G. Papakonstantinou, I. Riakiotakis, T. Andronikos, F. M. Ciorba and A. T. Chronopoulos, "Dynamic Scheduling for Dependence Loops on Heterogeneous Clusters", *Journal of Neural, Scientific and Parallel Computing*, vol 14, no. 4, pp. 359-384, 2006.

Διεθνή Περιοδικά υπό κρίση

UR1. I. Riakiotakis, F. M. Ciorba, T. Andronikos, G. Papakonstantinou and A. T. Chronopoulos, "Optimal Synchronization Frequency for Dynamic Pipelined Computations on Heterogeneous Systems", *Concurrency and Computation: Practice & Experience*.

UR2. F. M. Ciorba, I. Riakiotakis, T. Andronikos, A. T. Chronopoulos, and G. Papakonstantinou, "Studying the impact of synchronization frequency on scheduling tasks with dependencies in heterogeneous systems", *Journal of Performance Evaluation*.

UR3. I. Riakiotakis, F. M. Ciorba, T. Andronikos, G. Papakonstantinou and A. T. Chronopoulos, "A Distributed Scheme for Dynamic Load Balancing of Pipelined Computations", *Journal of Parallel Computing*.

Διεθνή Συνέδρια

C1. I. Riakiotakis, G. Papakonstantinou and A. T. Chronopoulos, "Implementation of Dynamic Loop Scheduling in Reconfigurable Platforms", *In Procs of the 3rd International Symposium of Industrial embedded systems, SIES'08*, Montpellier, France 2008.

C2. I. Riakiotakis, G. Goumas, N. Koziris, F.A. Metallinou, I. Daglis, "Evaluation of Dynamic Scheduling Methods in Simulations of Storm-time Ion Acceleration", *In Proc. of the 22th IEEE International Parallel and Distributed Processing Symposium (IPDPS '08), The 9th International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC'08)*, Miami, Florida, USA, April 18, 2008.

C3. F. M. Ciorba, I. Riakiotakis, T. Andronikos, A. T. Chronopoulos, and G. Papakonstantinou, "Optimal Synchronization Frequency for Dynamic Pipelined Computations on Heterogeneous Systems", *In Procs of the IEEE International Conference on Cluster Computing (CLUSTER 2007)*, Austin, TX USA, September 17-20, 2007.

C4. F. M. Ciorba, I. Riakiotakis, T. Andronikos, A. T. Chronopoulos, and G. Papakonstantinou, "Studying the impact of synchronization frequency on scheduling tasks with dependencies in heterogeneous systems", *PACT '07: In Procs of the 16th International Conference on Parallel Architectures and Compilation Techniques*, pp. 403, Brasov, Romania, September 15-19, 2007.

C5. I. Riakiotakis, F. M. Ciorba, T. Andronikos, and G. Papakonstantinou, "Self-Adapting Scheduling for Tasks with Dependencies in Stochastic Environments", *In Procs of the IEEE International Conference on Cluster Computing (CLUSTER 2006), Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar '06)*, pp. 1-8, Barcelona, Spain, September 25-28, 2006.

C6. F. M. Ciorba, T. Andronikos, I. Riakiotakis, A. T. Chronopoulos, and G. Papakonstantinou, "Dynamic Multi Phase Scheduling for Heterogeneous Clusters", *In Procs. of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS '06)*, pp. 10-, Rhodes, Greece, April 25-29, 2006.

C7. I. Riakiotakis and P. Tsanakas. "Dynamic Scheduling of Nested Loops with Uniform Dependencies in Heterogeneous Networks of Workstations", *in Procs of the 8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05)*, Las Vegas, NV, USA, 2005.

Κεφάλαιο 2

Βασικές έννοιες

2.1 Κατανεμημένα συστήματα

Κατανεμημένο σύστημα ονομάζουμε μια συλλογή από ξεχωριστά υπολογιστικά στοιχεία τα οποία μπορούν να επικοινωνήσουν μεταξύ τους μέσω ενός δικτύου. Τα υπολογιστικά στοιχεία αυτά μπορεί να αποτελούνται από απλούς προσωπικούς υπολογιστές, σταθμούς εργασίας (workstations) ή ακόμα και εξυπηρετητές (servers) με πολλαπλούς επεξεργαστές. Ακόμα το κατανεμημένο σύστημα μπορεί να περιλαμβάνει επιστημονικά όργανα, όπως τηλεσκόπια ή διάφορους αισθητήρες, τα οποία παρέχουν στο σύστημα ροές δεδομένων. Το κατανεμημένο σύστημα έχει στόχο να ενοποιήσει όλους αυτούς τους ετερογενείς πόρους και να τους παρουσιάσει σαν ένα ενιαίο σύνολο. Σε ένα τέτοιο σύστημα κάθε χρήστης μπορεί να προσπελάσει τοπικούς και απομακρυσμένους πόρους και να εκτελέσει διεργασίες οπουδήποτε χωρίς να ενδιαφέρεται που ακριβώς αυτές εκτελούνται.

Υπάρχει ένας διαχωρισμός ανάμεσα στα κατανεμημένα και τα παράλληλα συστήματα. Αυτός ο διαχωρισμός σχετίζεται με τον τρόπο διασύνδεσης των επεξεργαστικών στοιχείων. Ένα παραδοσιακό παράλληλο σύστημα αποτελείται από επεξεργαστικά στοιχεία που επικοινωνούν μέσω κοινής μνήμης και στα οποία τα δεδομένα είναι διαθέσιμα σε όλους τους επεξεργαστές ή η ανταλλαγή δεδομένων μπορεί να γίνει πολύ γρήγορα, έτσι αυτά τα συστήματα ονομάζονται στενά συνδεδεμένα (tightly coupled). Αντίθετα, στα κατανεμημένα συστήματα η ανταλλαγή δεδομένων γίνεται μέσω του επικοινωνιακού δικτύου, το οποίο τις περισσότερες φορές δεν είναι σχεδιασμένο για παράλληλες εφαρμογές υψηλής απόδοσης, έτσι αυτά τα συστήματα ονομάζονται χαλαρά συνδεδεμένα (loosely coupled). Στην συνέχεια θα αναφερόμαστε και στα κατανεμημένα συστήματα σαν παράλληλα συστήματα από την άποψη ότι τα χρησιμοποιούμε για την ανάπτυξη και εκτέλεση παράλληλων εφαρμογών, δηλαδή ότι οι διαθέσιμοι ε-

πεξεργαστές δουλεύουν παράλληλα για να ολοκληρώσουν από κοινού κάποια εργασία. Τα κατανεμημένα συστήματα παρουσιάζουν ορισμένα πλεονεκτήματα έναντι των παραδοσιακών παράλληλων συστημάτων:

- Χαμηλό κόστος. Η διαθεσιμότητα υπολογιστών και δικτυακού εξοπλισμού χαμηλού κόστους ευνοεί την ανάπτυξη κατανεμημένων συστημάτων που μπορεί να περιλαμβάνουν εκατοντάδες ή και χιλιάδες επεξεργαστές.

- Δυνατότητα κλιμάκωσης (scalability). Η επέκταση ενός κατανεμημένου συστήματος γίνεται εύκολα με την προσθήκη νέων υπολογιστών.

- Αξιοπιστία (reliability). Η αποτυχία ενός ή περισσότερων υπολογιστών δεν προκαλεί την κατάρρευση όλου του συστήματος όσο εξακολουθούν να υπάρχουν διαθέσιμοι υπολογιστές.

Ωστόσο, παρουσιάζουν και ορισμένες ιδιαιτερότητες:

- Το επικοινωνιακό δίκτυο, όπως προαναφέρθηκε, δεν είναι πάντα σχεδιασμένο για παράλληλες εφαρμογές υψηλής απόδοσης.

- Τα δίκτυα υπολογιστών αποτελούνται συνήθως από ετερογενείς πόρους. Η ανομοιογένεια συνήθως αναφέρεται στους τύπους των επεξεργαστών και στο μέγεθος της μνήμης, στο επικοινωνιακό δίκτυο, αλλά και στο διαθέσιμο λογισμικό.

- Τα κατανεμημένα συστήματα συνθέτονται από αυτόνομους υπολογιστές, με πολλαπλούς χρήστες να χρησιμοποιούν τους πόρους του συστήματος ταυτόχρονα (μη αφοσιωμένα συστήματα non-dedicated system).

2.1.1 Μοντέλα κατανεμημένης κοινής μνήμης και ανταλλαγής μηνυμάτων (Distributed shared memory - message passing).

Για την ανάπτυξη παράλληλων εφαρμογών σε κατανεμημένα συστήματα, λόγω της απουσίας κοινής / μοιραζόμενης μνήμης που επιτρέπει την άμεση ανταλλαγή δεδομένων, υπάρχουν δυο μοντέλα επικοινωνιών, το μοντέλο ανταλλαγής μηνυμάτων (message passing), και το μοντέλο κατανεμημένης κοινής μνήμης (distributed shared memory (DSM)).

Το μοντέλο DSM παρέχει στους χρήστες του μια εικονική κοινή μνήμη η οποία είναι διαθέσιμη στις διεργασίες που εκτελούνται στους υπολογιστές που συνθέτουν το κατανεμημένο σύστημα. Η εικονική μνήμη απαρτίζεται από τμήματα της τοπικής μνήμης των μηχανημάτων του δικτύου και παρουσιάζεται

στον χρήστη σαν ένας μεγάλος, ενιαίος χώρος διευθύνσεων μέσα στον οποίο μπορεί να προσπελάσει δεδομένα ακριβώς όπως και σε ένα σύστημα ενός επεξεργαστή. Για να γίνει αυτό υπάρχει ένα κρυμμένο επίπεδο που χρησιμοποιεί κάποια βιβλιοθήκη επικοινωνιών για την μεταφορά δεδομένων μεταξύ των απομακρυσμένων επεξεργαστών μέσω του δικτύου. Αυτό προσφέρει μεγάλη ευκολία στον χρήστη, ωστόσο είναι προφανές ότι η το κόστος που προσθέτει αυτό το επιπλέον επίπεδο μειώνει την απόδοση συγκριτικά με την απευθείας χρήση του μοντέλου message passing στο οποίο έχουμε απευθείας ανταλλαγή μηνυμάτων με την χρήση κλήσεων του τύπου send - receive. Οι κλήσεις αυτές παρέχονται στον προγραμματιστή σε μορφή βιβλιοθηκών που έχουν στόχο να διευκολύνουν τον προγραμματιστή, αποκρύπτοντας λεπτομέρειες σχετικές με το δίκτυο και τα δικτυακά πρωτόκολλα που χρησιμοποιούνται και προσφέροντας ομοιόμορφη πρόσβαση στους πόρους του συστήματος. Οι βασικές ρουτίνες που παρέχονται στις βιβλιοθήκες message passing είναι:

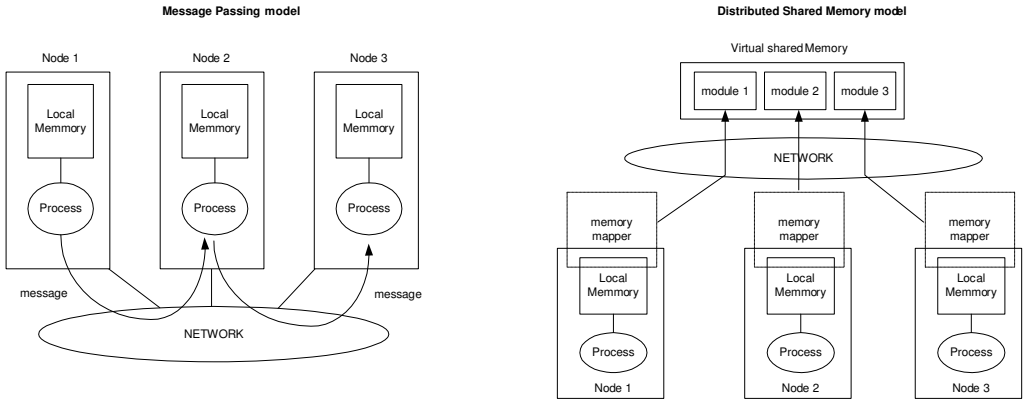
- ρουτίνες διαχείρισης διεργασιών, όπως αρχικοποίηση και τερματισμός και προσδιορισμός αριθμού διεργασιών.
- ρουτίνες επικοινωνιών point-to-point, αποστολή και λήψη δεδομένων μεταξύ μεμονωμένων διεργασιών (send-receive)
- ρουτίνες συγκεντρωτικών επικοινωνιών (collective communications) όπως broadcast, gather, scatter κτλ.

Είναι εμφανές ότι μια εφαρμογή που βασίζεται στο μοντέλο ανταλλαγής μηνυμάτων θα αποτελείται από δυο διακριτά μέρη, στο ένα θα εκτελεί υπολογισμούς ενώ στο δεύτερο θα πραγματοποιεί κλήσεις σε ρουτίνες επικοινωνιών. Τα μέρη αυτά βέβαια μπορούν να επαναλαμβάνονται και να εναλλάσσονται κατά την εκτέλεση της εφαρμογής, αλλά σε γενικές γραμμές μπορούμε να πούμε ότι διακρίνουμε περιόδους υπολογισμών που διακόπτονται από περιόδους επικοινωνιών.

2.2 Αλγοριθμικό Μοντέλο

Οι αλγόριθμοι που προσπαθούμε να παραλληλοποιήσουμε είναι αυτοί που περιέχουν φωλιασμένους βρόχους και έχουν την μορφή που δίνεται στον παρακάτω ψευδοκώδικα :

```
For{j1 =l1 to u1}{
    For {j2 = l2 to u2}{
        ...
```



Σχήμα 2.1: τα μοντέλα Message-Passing και DSM

```
        For {jn = ln to un}{
            S1(J)
            ...
            Sk(J)
        }
        ...
    }
}
```

όπου J_i , $0 \leq i \leq n-1$, είναι μεταβλητές του βρόχου με κάτω και άνω όρια που δηλώνονται από τις ακέραιες σταθερές l_i,u_i. Τα S_j, $1 \leq j \leq k$ αναπαριστούν k δηλώσεις ανάθεσης της μορφής $X_o = E(X_1, X_2, \ldots, X_k)$, όπου X_o είναι η έξοδος που παράγεται από την έκφραση E όταν εφαρμοστεί στις μεταβλητές εισόδου $X_1, X_2, \ldots, X_k$. Οι μεταβλητές εισόδου παράγονται είτε από άλλες δηλώσεις είτε από την ίδια δήλωση σε μια προηγούμενη της εκτέλεση.

Οι φωλιασμένοι βρόχοι ορίζουν ένα χώρο επαναλήψεων (iteration space) ο οποίος μπορεί να παρασταθεί σαν ένας διακριτός, καρτεσιανός χώρος του οποίου ο αριθμός των διαστάσεων δίνεται από το βάθος n του φωλιασμένου βρόχου. Κάθε επανάληψη σχετίζεται με ένα δείκτη της μορφής $\overrightarrow{J} = [J_1, J_2, \ldots, J_n]^T$, και το σύνολο αυτών των δεικτών ορίζει τον χώρο δεικτών (index space) J^n=$\{[j_0, j_1, \ldots, j_n]^T : j_i \in Z \wedge l_i \leq j_i \leq u_i, \forall i$ όπου $0 \leq i \leq n-1\}$. Τα σημεία του J^n είναι διατεταγμένα λεξικογραφικά και αυτή η διάταξη διατηρείται όταν ο φωλιασμένος βρόχος εκτελείται ακολουθιακά. Η διάταξη αυτή δεν είναι απαραίτητο να διατηρηθεί αρκεί να τηρούνται κάποιες προϋποθέσεις που εξασφαλίζουν την ορθότητα των αποτελεσμάτων σε σχέση με αυτών που προκύπτουν από την ακολουθιακή εκτέλεση. Το γεγονός αυτό εκμεταλλευόμαστε

για να ανακαλύψουμε τον διαθέσιμο παραλληλισμό ώστε να μειώσουμε τον χρόνο εκτέλεσης.

Υπάρχει ένας βασικός διαχωρισμός σε αυτή την κατηγορία αλγορίθμων ανάλογα αν οι επαναλήψεις τους είναι ανεξάρτητες ή όχι (parallel or dependence loops). Αν δεν υπάρχουν εξαρτήσεις, οι επαναλήψεις τους μπορούν να εκτελεστούν σε οποιαδήποτε σειρά ή ακόμα και ταυτόχρονα. Η ύπαρξη εξαρτήσεων υπονοεί ότι η πρέπει να τηρηθεί κάποια σειρά στην εκτέλεση των επαναλήψεων, μειώνοντας έτσι τον διαθέσιμο παραλληλισμό.

Υπάρχουν τέσσερα είδη εξαρτήσεων:

1. Εξαρτήσεις ροής (flow dependencies). Οι εξαρτήσεις ροής εμφανίζονται όταν μια τιμή που ανατίθεται σε μια μεταβλητή σε ένα στιγμιότυπο του βρόχου χρησιμοποιείται σε ένα επόμενο στιγμιότυπο είτε στην ίδια ή σε μια διαφορετική δήλωση ανάθεσης.

2. Αντί-εξαρτήσεις (anti-dependencies). Οι εξαρτήσεις αυτές παρουσιάζονται όταν η τιμή μιας μεταβλητής που διαβάστηκε από μια δήλωση σε ένα στιγμιότυπο του βρόχου μεταβάλλεται από μια οποιαδήποτε δήλωση ανάθεσης σε ένα επόμενο στιγμιότυπο.

3. Εξαρτήσεις εξόδου (output dependencies). Αυτής της μορφής οι εξαρτήσεις προκύπτουν όταν η τιμή που δίνεται σε μια μεταβλητή σε μια εκτέλεση του βρόχου μεταβάλλεται από μια οποιαδήποτε δήλωση κατά την εκτέλεση ενός επόμενου στιγμιότυπου.

4. Εξαρτήσεις ελέγχου (control dependencies). Εξαρτήσεις ελέγχου μεταξύ δυο δηλώσεων υπάρχουν όταν μια συνθήκη ελέγχου καθορίζει αν θα εκτελεστεί ή όχι μια δήλωση μετά από μια άλλη.

Από τα παραπάνω είδη εξαρτήσεων θα μας απασχολήσουν μόνο οι εξαρτήσεις ροής γιατί αυτές έχουν άμεση σχέση με τον υπολογισμό, ενώ τα δυο επόμενα είδη αφορούν το μοντέλο αποθήκευσης των μεταβλητών και το τελευταίο είδος δεν έχει να κάνει με μεταβλητές αλλά με την μεταβολή της ροής του προγράμματος. Οι εξαρτήσεις ελέγχου προκαλούν ανομοιομορφία στο βάρος του σώματος του βρόχου και θα μας απασχολήσουν στο μέλλον όταν θα ασχοληθούμε με αλγόριθμους με μη ομοιόμορφο υπολογιστικό φορτίο (non-uniform loops), ένα κατεξοχήν πεδίο εφαρμογής των δυναμικών αλγορίθμων δρομολόγησης. Στην συνέχεια, όταν αναφερόμαστε σε εξαρτήσεις, πάντα θα μιλάμε για εξαρτήσεις ροής.

Οι εξαρτήσεις εκφράζονται από τα διανύσματα εξάρτησης. Αν υπάρχει εξάρτηση μεταξύ δυο στιγμιότυπων $S_j(i_1)$ και $S_j(i_2)$ τότε το διάνυσμα εξάρτησης δίνεται ως $\overrightarrow{d} = \overrightarrow{i_2} - \overrightarrow{i_1}$. Ένας αλγόριθμος μπορεί να έχει ένα πλήθος τέτοιων διανυσμάτων εξάρτησης $\overrightarrow{d_1}, \ldots \overrightarrow{d_p}$, $p < n$ τα οποία σχηματίζουν τις στήλες του πίνακα εξαρτήσεων $D = \{\overrightarrow{d_1}, \ldots, \overrightarrow{d_1}\}$. Όταν τα διανύσματα αυτά παραμένουν σταθερά και διατηρούνται σε όλο τον χώρο δεικτών τότε αυτά ονομάζονται σταθερά και ομοιόμορφα διανύσματα εξάρτησης (constant and uniform dependence vectors). Η κατηγορία αλγορίθμων που θα μας απασχολήσει είναι αυτή των φωλιασμένων βρόχων με ομοιόμορφες και σταθερές εξαρτήσεις (Uniform Depedence Loops -UDLs).

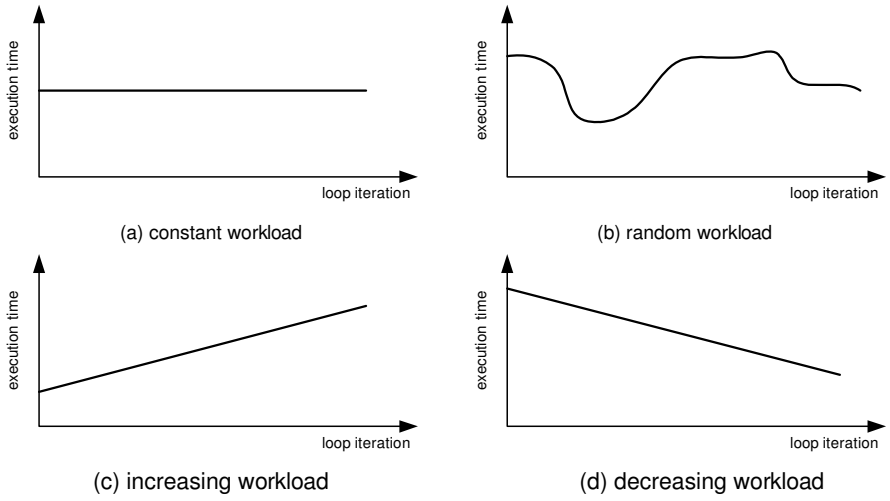
Τέλος ο φωλιασμένος βρόχος ορίζεται από τον χώρο δεικτών του J^n και από τον πίνακα εξαρτήσεων D και συμβολίζεται ως $L(J^n, D)$.

2.2.1 Κατανομές υπολογιστικού φορτίου.

Ορίζουμε σαν υπολογιστικό φορτίο, το σύνολο των εντολών που περιέχονται σε ένα φωλιασμένο βρόχο. Η κατανομή του υπολογιστικού φορτίου μπορεί να είναι είτε ομοιόμορφη είτε ανομοιόμορφη. Ο φόρτος ενός βρόχου είναι **ομοιόμορφα** κατανεμημένος (Σχήμα 2.2a) αν ο χρόνος εκτέλεσης όλων των επαναλήψεων του είναι ίδιος (σταθερός), σε κάθε άλλη περίπτωση, έχουμε να κάνουμε με **ανομοιόμορφα** κατανεμημένο βρόχο. Οι ανομοιόμορφοι βρόχοι μπορεί να ακολουθούν κάποιο πρότυπο, όπως στην περίπτωση των γραμμικά κατανεμημένων βρόχων στους οποίους το υπολογιστικό φορτίο μπορεί είτε να αυξάνεται (linearly increasing workload, σχήμα 2.2c), ή να μειώνεται (linearly decreasing workload, σχήμα 2.2d). Παραδείγματα τέτοιων βρόχων δίνεται παρακάτω:

```
/* increasing */
    DOALL K = 1 TO I
        Serial DO J = 1 TO K
            Serial Loop Body
        End Serial DO
    END DOALL

/* decreasing */
    DOALL K = 1 TO I
        Serial DO J = 1 TO I-K+1
         Serial Loop Body
        End Serial DO
    END DOALL
```

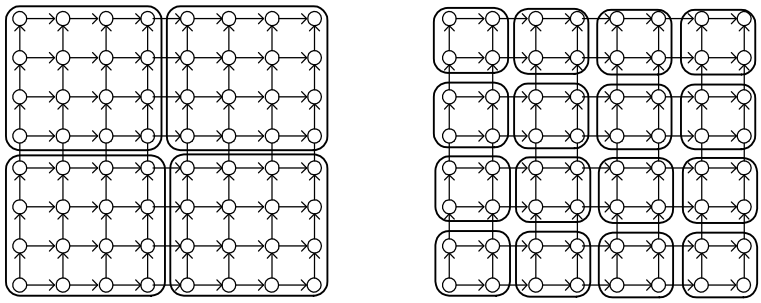


Σχήμα 2.2: Κατανομές υπολογιστικού φορτίου

Όπως βλέπουμε στον ψευδοκώδικα παραπάνω, όσο αυξάνεται η τιμή του εξωτερικού δείκτη K, το βάρος του εσωτερικού (σειριακού) βρόχου είτε αυξάνεται είτε μειώνεται γραμμικά. Τέλος, σε βρόχους που περιέχουν δηλώσεις διακλάδωσης (π.χ. if, $else$ κτλ.), η κατανομή του υπολογιστικού φορτίου μπορεί να είναι απρόβλεπτη. Οι βρόχοι αυτοί λέμε ότι έχουν **τυχαία** ή **ακανόνιστη** κατανομή υπολογιστικού φορτίου (random workload, σχήμα 2.2b).

2.2.2 Διαμερισμός του χώρου επαναλήψεων.

Για να εκτελέσουμε παράλληλα ένα αλγόριθμο, πρέπει να ορίσουμε ένα αριθμό εργασιών και να αναθέσουμε τις εργασίες αυτές στους διαθέσιμους επεξεργαστές. Σε γενικές γραμμές είναι επιθυμητό να έχουμε ένα μεγάλο αριθμό εργασιών έτσι ώστε να μπορούν να δουλεύουν ταυτόχρονα πολλοί επεξεργαστές, δηλαδή να έχουμε ένα υψηλό βαθμό παραλληλισμού. Ο διαχωρισμός σε εργασίες μπορεί να πάρει δυο κατευθύνσεις, είτε να γίνει διαχωρισμός του χώρου επαναλήψεων του αλγορίθμου, όποτε μιλάμε για data parallel εφαρμογές ή για διαχωρισμό των διακριτών λειτουργιών του αλγορίθμου, οπότε έχουμε task parallel εφαρμογές. Για ένα αλγόριθμο της μορφής που περιγράφεται παραπάνω, θα μπορούσαμε να έχουμε ένα αριθμό εργασιών ίσο με τον αριθμό επαναλήψεων που περιέχει ο χώρος επαναλήψεων του. Αυτή η προσέγγιση δίνει ένα πολύ μεγάλο αριθμό εργασιών, ωστόσο δεν οδηγεί αναγκαστικά σε ένα αποδοτικό παράλληλο αλγόριθμο για δυο λόγους: ο πρώτος λόγος είναι η ύπαρξη εξαρτήσεων η οποία μειώνει τον διαθέσιμο παραλληλισμό, δηλαδή αν



Σχήμα 2.3: Επίδραση του μεγέθους κόκκου (granularity) στον όγκο επικοινωνιών

και έχουμε ένα μεγάλο αριθμό εργασιών, αυτές δεν μπορούν να εκτελεστούν όλες ταυτόχρονα. Ο δεύτερος λόγος είναι το κόστος επικοινωνίας, το οποίο είναι αρκετά σημαντικός παράγοντας, ιδιαίτερα στα κατανεμημένα συστήματα. Οι επεξεργαστές σε κάποιο σημείο θα πρέπει να σταματήσουν να εκτελούν χρήσιμους υπολογισμούς για να ανταλλάξουν δεδομένα. Η απόδοση μπορεί να βελτιωθεί μειώνοντας τον χρόνο ανταλλαγής δεδομένων, ανταλλάσσοντας δηλαδή λιγότερα δεδομένα. Επίσης, το κόστος επικοινωνίας αποτελείται από ένα σταθερό μέρος (startup cost) και ένα μεταβλητό μέρος που εξαρτάται από τον όγκο τον δεδομένων, άρα, ο χρόνος ανταλλαγής μηνυμάτων μπορεί να μειωθεί ανταλλάσσοντας τον ίδιο όγκο δεδομένων σε λιγότερα μηνύματα. Για να μειώσουμε τον όγκο των δεδομένων που ανταλλάσσονται, συνδυάζουμε έναν αριθμό εργασιών σε μια μεγαλύτερη εργασία (agglomeration). Στην περίπτωση των φωλιασμένων βρόχων, συνδυάζουμε διαδοχικές επαναλήψεις σε ομάδες επαναλήψεων (chunks of iterations), οι οποίες ανατίθενται στους επεξεργαστές. Όσο μεγαλύτερες είναι αυτές οι ομάδες, τόσο πιο μικρή είναι η συχνότητα και ο όγκος των επικοινωνιών (σχήμα 2.3).

Ανάλογα με τον αριθμό το μέγεθος των εργασιών και την συχνότητα των επικοινωνιών χωρίζουμε τις παράλληλες εφαρμογές σε δυο κατηγορίες. Αυτές που με παραλληλισμό λεπτού κόκκου (fine grain) και σε αυτές με παραλληλισμό χονδρού κόκκου (coarse grain). Στον παραλληλισμό λεπτού κόκκου έχουμε σχετικά λίγους υπολογισμούς ανάμεσα στις επικοινωνίες και μεγάλη συχνότητα επικοινωνιών, δηλαδή χαμηλό λόγο χρόνου υπολογισμών προς τον χρόνο επικοινωνιών. Αυτό σημαίνει ότι σε αρχιτεκτονικές στις οποίες το κόστος επικοινωνίας είναι υψηλό ο παραλληλισμός λεπτού κόκκου είναι δύσκολο να επιτύχει καλή απόδοση. Σε αρχιτεκτονικές όμως όπου το κόστος επικοινωνίας είναι αμελητέο,

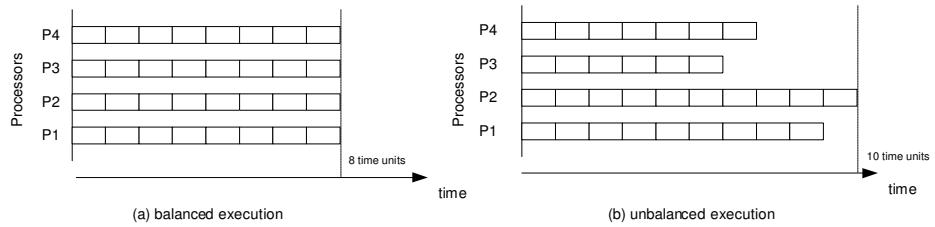
όπως σε μηχανήματα μοιραζόμενης μνήμης επιλέγεται ο παραλληλισμός λεπτού κόκκου γιατί μπορεί να κατανείμει καλύτερα το υπολογιστικό φορτίο στους διαθέσιμους επεξεργαστές. Αντίθετα στον παραλληλισμό χονδρού κόκκου έχουμε μεγάλο αριθμό υπολογισμών και μικρή συχνότητα επικοινωνιών, δηλαδή υψηλό λόγο χρόνου υπολογισμών προς τον χρόνο επικοινωνιών γεγονός που βρίσκει εφαρμογή σε αρχιτεκτονικές κατανεμημένης μνήμης στις οποίες το κόστος επικοινωνίας είναι υψηλότερο. Ο λόγος του χρόνου υπολογισμών προς τον χρόνο επικοινωνιών ή λόγος υπολογισμών - επικοινωνιών (computation to communication ratio) είναι πολύ σημαντικός στην σχεδίαση παράλληλων εφαρμογών και η συνθήκη που χρησιμοποιείται συνήθως είναι $\frac{computation\ time}{communication\ time} \geq 1$, που εξασφαλίζει ότι η εφαρμογή ξοδεύει περισσότερο ή τουλάχιστον ίδιο χρόνο σε υπολογισμούς με τον χρόνο που ξοδεύει σε επικοινωνίες.

2.3 Αλγόριθμοι δρομολόγησης

Κατά την ανάθεση των επαναλήψεων ενός φωλιασμένου βρόχου στους διαθέσιμους επεξεργαστές έχουμε δυο δυνατότητες, μπορούμε είτε να αποφασίσουμε εξαρχής (compile-time) ποιες επαναλήψεις θα αναθέσουμε σε κάθε επεξεργαστή ή να αναθέτουμε σταδιακά μέρος των επαναλήψεων καθώς προχωράει η εκτέλεση (run-time). Στη πρώτη περίπτωση λέμε ότι η ανάθεση γίνεται στατικά με την χρήση ενός στατικού αλγόριθμου δρομολόγησης ενώ στην δεύτερη δυναμικά με την χρήση ενός δυναμικού αλγόριθμου. Αν μπορεί να προβλεφθεί με ακρίβεια ο χρόνος εκτέλεσης κάθε επανάληψης ανά επεξεργαστή, ένας στατικός αλγόριθμος μπορεί να πετύχει τέλεια κατανομή του υπολογιστικού φόρτου (load balance), έτσι ώστε να ελαχιστοποιηθεί ο συνολικός χρόνος εκτέλεσης όπως φαίνεται στο Σχήμα 2.4*a*. Ωστόσο πολλές φορές η πρόβλεψη αυτή δεν είναι δυνατό να γίνει με ακρίβεια, είτε λόγο του της ανομοιογένειας του συστήματος, είτε λόγο της φύσης της εφαρμογής και έτσι οδηγούμαστε σε άνιση κατανομή του υπολογιστικού φόρτου (load-imbalance) (Σχήμα 2.4*b*). Με την χρήση δυναμικών αλγορίθμων μεταφέρουμε την ανάθεση των επαναλήψεων στους επεξεργαστές στον χρόνο εκτέλεσης, με αυτόν τον τρόπο δεν χρειάζονται ακριβείς προβλέψεις του χρόνου εκτέλεσης των επαναλήψεων γιατί η κατανομή του υπολογιστικού φόρτου διορθώνεται δυναμικά καθώς προχωράει η εκτέλεση.

2.3.1 Δυναμική Εξισορρόπηση φορτίου.

Σε γενικές γραμμές, ένας αλγόριθμος δυναμικής εξισορρόπησης φορτίου παρέχει τους μηχανισμούς που επιτρέπουν την δίκαιη κατανομή του υπολογιστικού φορτίου με στόχο την επίτευξη του ελάχιστου δυνατού παράλληλου χρόνου.



Σχήμα 2.4: Εξισορροπημένη (balanced) και μη εξισορροπημένη (unbalanced) παράλληλη εκτέλεση

Οι όροι δυναμική δρομολόγηση και δυναμική εξισορρόπηση φορτίου χρησιμοποιούνται εναλλάξιμα, αν και ο πρώτος αναφέρεται απλά στην ανάθεση εργασιών στους διαθέσιμους επεξεργαστές, ενώ ο δεύτερος είναι πιο ειδικός και αναφέρεται στην βελτίωση της απόδοσης μέσω της εξισορροπημένης ανάθεσης των εργασιών στους επεξεργαστές. Ωστόσο, αφού η χρήση των δυναμικών αλγορίθμων αποσκοπεί στην βελτίωση της απόδοσης έναντι των στατικών αλγορίθμων, οι δυναμικοί αλγόριθμοι δρομολόγησης υπονοείται ότι ασχολούνται με την εξισορρόπηση του υπολογιστικού φορτίου κατά την ανάθεση του στους διαθέσιμους επεξεργαστές.

Οι μέθοδοι δυναμικής εξισορρόπησης φορτίου χωρίζονται σε δύο κατηγορίες: τις **συγκεντρωτικές (centralized)** και τις **αποκεντρωμένες (decentralized)**.

Στην **συγκεντρωτική** προσέγγιση, οι αποφάσεις σχετικά με την ανάθεση των εργασιών στους επεξεργαστές λαμβάνονται από ένα κεντρικό κόμβο που συντονίζει και καθορίζει την πορεία της εκτέλεσης. Επίσης, οι εργασίες είναι αποθηκευμένες σε μια κεντρική τοποθεσία (work-queue) από όπου μοιράζονται στους επεξεργαστές. Το μοντέλο αυτό υπολογισμού είναι γνωστό ως μοντέλο συντονιστή-εργάτη (master-slave ή τα τελευταία χρόνια master-worker). Ο συντονιστής έχει στην κατοχή του τις εργασίες που πρέπει να εκτελεστούν από τους επεξεργαστές, οι οποίοι στην περίπτωση αυτή ονομάζονται εργάτες. Όταν ένας εργάτης ολοκληρώσει την εκτέλεση μιας εργασίας, ζητάει μια νέα εργασία από τον συντονιστή. Το πλεονέκτημα της συγκεντρωτικής προσέγγισης είναι ότι ο κόμβος συντονιστής μπορεί να έχει την πλήρη εικόνα του συστήματος καθώς και των εργασιών που πρέπει να εκτελεστούν, και να χρησιμοποιήσει τα δεδομένα κατά την διάρκεια της διαδικασίας της δρομολόγησης. Από την άλλη μεριά, η ύπαρξη ενός μοναδικού κόμβου συντονιστή μπορεί να προκαλέσει συμφόρηση και καθυστέρηση στην εξυπηρέτηση των αιτήσεων των εργατών, βλάπτοντας έτσι την συνολική απόδοση. Το συγκεντρωτικό μοντέλο είναι λοι-

πόν κατάλληλο όταν ο αριθμός των διαθέσιμων εργατών είναι σχετικά μικρός και όταν οι εργασίες είναι επαρκώς υπολογιστικά εντατικές, ώστε οι εργάτες να μην ζητάνε συνέχεια δουλεία.

Οι **αποκεντρωμένες** μέθοδοι έχουν στόχο να μοιράσουν τις αποφάσεις σχετικά με την δρομολόγηση των εργασιών σε περισσότερους κόμβους συντονιστές. Μια προσέγγιση είναι αυτή των ιεραρχικών μοντέλων. Τα ιεραρχικά μοντέλα αποτελούνται από περισσότερους συντονιστές, οι οποίοι μοιράζονται τον συνολικό αριθμό εργασιών στις τοπικές ουρές εργασιών τους. Κάθε συντονιστής αναλαμβάνει ένα μέρος του συνόλου των διαθέσιμων εργατών και έχει έτσι την εικόνα ενός μόνο μέρους του συστήματος. Με αυτόν τον ιεραρχικό τρόπο αυξάνεται ο αριθμός εργατών που το σύστημα μπορεί να χειριστεί αποδοτικά, ωστόσο υπάρχει μια επιπλέον δυσκολία στον συγχρονισμό των συντονιστών, ειδικά αν η εφαρμογή που παραλληλοποιείτα περιέχει εξαρτήσεις και απαιτεί την συχνή μεταφορά δεδομένων. Τέλος, υπάρχει η **πλήρως αποκεντρωμένη** (fully distributed) προσέγγιση, στην οποία κάθε εργάτης αποφασίζει για τον όγκο δουλείας που θα εκτελέσει. Συνήθως, σε αυτή την προσέγγιση δεν υπάρχει συντονιστής και το σύνολο των εργασιών μοιράζεται με κάποιο κριτήριο στους διαθέσιμους εργάτες. Στην διάρκεια της εκτέλεσης οι εργάτες ανταλλάσσουν μέρος των εργασιών ανάλογα με την επεξεργαστική τους ικανότητα, έτσι ώστε να εξισορροπήσουν το υπολογιστικό φορτίο. Η προσέγγιση αυτή είναι γνωστή σαν κλέψιμο εργασιών (work stealing)

Η πιο συνηθισμένη προσέγγιση όσον αφορά την δυναμική δρομολόγηση φωλιασμένων βρόχων είναι η συγκεντρωτική. Αυτό οφείλεται κυρίως στο ότι είναι πιο εύκολο να σχεδιαστούν και να υλοποιηθούν συγκεντρωτικοί από ότι αποκεντρωμένοι αλγόριθμοι. Επίσης, μέχρι προσφάτως, δεν ήταν εύκολο σε μία εγκατάσταση να συγκεντρωθεί ένας πολύ μεγάλος αριθμός επεξεργαστών, μειώνοντας έτσι την ανάγκη για αποκεντρωμένους αλγορίθμους. Στης μέρες μας με την καθιέρωση των πολυπύρηνων επεξεργαστών (multicore processor), όπως επίσης και με την ανάπτυξη του υπολογιστικού πλέγματος (the grid), ο αριθμός των διαθέσιμων επεξεργαστικών στοιχείων έχει αυξηθεί δραματικά. Αυτή η εξέλιξη στην αγορά θα δώσει ώθηση στην ανάπτυξη των αποκεντρωμένων αλγορίθμων.

2.3.2 Απλοί αλγόριθμοι δυναμικής δρομολόγησης.

Μια σημαντική κατηγορία δυναμικών αλγορίθμων είναι οι αυτοδρομολογούμενοι αλγόριθμοι (self scheduling algorithms). Οι αλγόριθμοι αυτοί είχαν δημιουργηθεί αρχικά για την δρομολόγηση βρόχων χωρίς εξαρτήσεις (parallel loops) σε ομογενή συστήματα είτε μοιραζόμενης είτε κατανεμημένης μνήμης. Στό-

χος τους ήταν να αντιμετωπίσουν περιπτώσεις βρόχων στους οποίους ο χρόνος εκτέλεσης από επανάληψη σε επανάληψη διέφερε σημαντικά (non-uniform loop-body), π.χ. λόγο της ύπαρξης διάφορων δηλώσεων διακλάδωσης (conditional statements), και επομένως δεν μπορεί να προβλεφθεί.

Στον πιο βασικό αλγόριθμο αυτοδρομολόγησης (simple self-scheduling) κάθε επεξεργαστής αναλαμβάνει να εκτελέσει μια επανάληψη του βρόχου μόλις γίνει διαθέσιμος, δηλαδή όταν τελειώσει την εκτέλεση της προηγούμενης επανάληψης που είχε αναλάβει. Αυτό γίνεται (σε μοντέλο shared-memory) μέσω μιας κοινής μεταβλητής η οποία δείχνει την επόμενη επανάληψη. Κάθε επεξεργαστής προσπαθεί να κερδίσει την αποκλειστική πρόσβαση σε αυτή την μεταβλητή και μόλις το καταφέρει αναλαμβάνει να εκτελέσει την επόμενη επανάληψη. Στην συνέχεια ελευθερώνει την μεταβλητή η οποία τώρα είναι διαθέσιμη για τους υπόλοιπους επεξεργαστές. Αυτός ο αλγόριθμος επιτυγχάνει πολύ καλή κατανομή φορτίου αφού οι επεξεργαστές μπορούν να τελειώσουν την εκτέλεση το πολύ με διαφορά μιας επανάληψης. Ωστόσο είναι φανερό ότι το κόστος δρομολόγησης (scheduling overhead), δηλαδή ο χρόνος που απαιτείται για την ανάθεση των επαναλήψεων στους επεξεργαστές είναι πολύ υψηλό. Σε μοντέλα κατανεμημένης μνήμης στα οποία το κόστος επικοινωνίας (communication cost) είναι υψηλό ένας τέτοιος αλγόριθμος δεν είναι πρακτικός.

Για να μειωθεί το κόστος δρομολόγησης επινοήθηκε ο αλγόριθμος **Chunk-scheduling (CSS)**[12], στον οποίο κάθε επεξεργαστής αναλαμβάνει μια ομάδα επαναλήψεων (chunk) την φορά αντί για μια και μόνο επανάληψη. Μέσω στατιστικής ανάλυσης οι Kruskal και Weiss πρότειναν ένα τύπο που συνδέει το βέλτιστο μέγεθος του της ομάδας επαναλήψεων με το κόστος δρομολόγησης. Αν το κόστος αυτό είναι αμελητέο τότε το βέλτιστο μέγεθος ομάδας είναι 1, οπότε έχουμε την περίπτωση του simple self-scheduling ενώ όταν αυτό είναι ιδιαιτέρα υψηλό τότε καταλήγουμε σε στατική δρομολόγηση, δηλαδή αν έχουμε N επαναλήψεις και k επεξεργαστές τότε το βέλτιστο ομάδας επαναλήψεων είναι $\frac{N}{k}$. Σε γενικές γραμμές όσο μεγαλύτερο είναι το μέγεθος της ομάδας επαναλήψεων τόσο μειώνεται το κόστος δρομολόγησης, ενώ μειώνεται ο διαθέσιμος παραλληλισμός. Σε ένα πρακτικό αλγόριθμο είναι επιθυμητό να έχουμε μεταβλητό μέγεθος ομάδων. Αρχικά δίνονται μεγάλες ομάδες, έτσι ώστε να μειωθεί το κόστος δρομολόγησης, τα οποία σταδιακά μειώνονται σε μέγεθος έτσι ώστε να βελτιωθεί η κατανομή του φόρτου εργασίας. Οι αλγόριθμοι που ακολουθούν αυτή την προσέγγιση είναι γνωστοί σαν αλγόριθμοι με φθίνων μέγεθος ομάδας (reducing chunk size algorithms).

Το **Guided self-scheduling (GSS)** [13] είναι ένας τέτοιος (reducing chunk size) αλγόριθμος. Το μέγεθος ομάδας επαναλήψεων c_i που αναθέτει κά-

θε φορά στον επόμενο επεξεργαστή δίνεται από τον αριθμό των υπολειπόμενων επαναλήψεων R_i προς τον αριθμό των διαθέσιμων επεξεργαστών k ,c_i= $\lceil \frac{R_i}{k} \rceil$ αν N είναι ο συνολικός αριθμός επαναλήψεων τότε, R_{i+1}= R_i - c_i και c_i = $\lceil \frac{R_i}{k} \rceil$ = $\lceil (1-\frac{1}{k})^i \times \frac{N}{k} \rceil$. 'Ενα πρόβλημα του GSS είναι ότι αναθέτει ένα μεγάλο μέρος των επαναλήψεων στους πρώτους επεξεργαστές και αυτό οδηγεί σε άνιση κατανομή υπολογιστικού φορτίου και αύξηση του συνολικού χρόνου εκτέλεσης.

Το **Factoring (FSS)** [83] έρχεται να βελτιώσει το GSS αναθέτοντας ομάδες επαναλήψεων ίδιου μεγέθους σε όλους τους επεξεργαστές πριν μειώσει το μέγεθος τους. Με αυτό τον τρόπο αποφεύγεται η μεγάλη διαφορά στα αρχικά μεγέθη ομάδων που παρουσιάζει ο GSS. Σε κάθε βήμα ο FSS αναθέτει $c_i = \lceil \frac{R_i}{2 \times k} \rceil$ επαναλήψεις σε κάθε επεξεργαστή. Στη συνέχεια η τιμή του R_i μειώνεται σε $R_{i+1} = R_i$- $(k \times c_i)$. Ο FSS όπως και ο GSS χρησιμοποιούν μια μη γραμμική συνάρτηση για να υπολογίσουν το μέγεθος των επόμενων ομάδων επαναλήψεων και αυτό μπορεί να δημιουργήσει ένα υψηλό κόστος δρομολόγησης συγκριτικά με αλγορίθμους όπως το CSS.

Το **Trapezoid self-scheduling (TSS)** [90] χρησιμοποιεί μια απλή γραμμική συνάρτηση μειώνοντας έτσι το κόστος δρομολόγησης. Ο TSS ζητάει να του ορίσουν το μέγεθος της αρχικής και τελικής ομάδας επαναλήψεων (C_f και C_l αντίστοιχα). Μια συντηρητική επιλογή δίνεται συνήθως ως $C_f = \frac{N}{2 \times k}$ και C_l=1. Αυτή η επιλογή εξασφαλίζει ότι δεν θα υπάρξει άνιση κατανομή φορτίου λόγο μεγάλων αρχικών ομάδων. Από αυτές τις τιμές ο TSS καθορίζει τον συνολικό αριθμό των ομάδων επαναλήψεων ως $T = \frac{2 \times N}{(C_f + C_l)}$ και από αυτό τον παράγοντα μείωσης (decrement) D=$\frac{(C_f - C_l)}{(N-1)}$. 'Ετσι τα μεγέθη των ομάδων που προκύπτουν είναι $c_1 = C_f$, c_2=$C_f - D$, c_3= $C_f - 2D$,. . . .

2.4 Βασικοί ορισμοί

Θα κλείσουμε αυτό το κεφάλαιο με την εισαγωγή κάποιων βασικών όρων στους οποίους θα αναφερθούμε στα επόμενα κεφάλαια:

- PE είναι ένα γενικό επεξεργαστικό στοιχείο (επεξεργαστής).
- m είναι ο αριθμός των εργατών σε ένα σύστημα.
- $P_1, \ldots, P_m$ είναι οι εργάτες (worker).
- VP_k είναι η σχετική υπολογιστική ισχύς του εργάτη P_k.

- $VP = \sum_{k=1}^{m} VP_k$ είναι η συνολική εικονική δύναμη υπολογισμών της συστοιχίας υπολογιστών.
- Q_k είναι ο αριθμός διεργασιών στην ουρά εκτέλεσης (run-queue) του P_k, αντικατοπτρίζοντας το φορτίο του P_k.
- $A_k = \left\lfloor \frac{VP_k}{Q_k} \right\rfloor$ είναι η διαθέσιμη υπολογιστική δύναμη (available computing power -ACP) του P_k.
- $A = \sum_{k=1}^{m} A_k$ είναι η συνολική διαθέσιμη υπολογιστική ισχύς της συστοιχίας.

Ανεξάρτητα από το την προσέγγιση που χρησιμοποιείται στον σχεδιασμό και την υλοποίηση ενός αλγορίθμου δρομολόγησης, τα στοιχεία που πραγματοποιούν την επεξεργασία των δεδομένων καλούνται *εργάτες*. Ένας εργάτης είναι μια διεργασία και εκτελείτε σε ένα επεξεργαστή. Συνήθως αποφεύγουμε να τοποθετούμε περισσότερους από έναν εργάτες στον ίδιο επεξεργαστή, για τον λόγο αυτό οι οροί εργάτης και επεξεργαστής ταυτίζονται. Υπό αυτή την έννοια, θεωρούμε ότι κάθε εργάτης έχει μια ουρά εκτέλεσης. Σε ένα αφοσιωμένο σύστημα, στο οποίο δεν υπάρχουν άλλοι χρήστες και στο οποίο εκτελείται αποκλειστικά η εφαρμογή μας, η ουρά εκτέλεσης περιέχει μόνο μια εργασία. Σε αντίθετη περίπτωση η ουρά περιέχει και άλλες εργασίες τις οποίες αντιλαμβανόμαστε σαν *εξωτερικό ή εξωγενές* φορτίο.

Σε ένα ετερογενές σύστημα κάθε εργάτης μπορεί να επεξεργάζεται στην μονάδα του χρόνου διαφορετικό αριθμό δεδομένων από τους υπόλοιπους εργάτες δηλαδή υπάρχουν γρήγοροι και αργοί εργάτες. Για να εκφράσουμε την ικανότητα του εργάτη να εκτελεί υπολογισμούς χρησιμοποιούμε τον αφηρημένο όρο της *υπολογιστικής ισχύος*. Η υπολογιστική ισχύς καθορίζεται με την χρήση τον κατάλληλων μετροπρογραμμάτων. Ένας πιο χρήσιμος όρος είναι η *σχετική υπολογιστική ισχύς* του εργάτη. Ο όρος αυτός αναφέρετε στην κανονικοποιημένη υπολογιστική δύναμη του εργάτη, σε σχέση με τους υπόλοιπους εργάτες πού είναι διαθέσιμοι την δεδομένη χρονική στιγμή.

Θεωρούμε ότι κάθε εργασία που βρίσκεται στην ουρά εκτέλεσης απασχολεί εξίσου τούς επεξεργαστικούς πόρους ενός εργάτη. Αυτό σημαίνει στην πράξη ότι, ένας εργάτης που έχει στην ουρά εκτέλεσης του δύο εργασίες θα είναι δύο φορές πιο αργός από έναν άλλο εργάτη με την ίδια υπολογιστική ισχύ και που έχει στην ουρά εκτέλεσης μόνο μια εργασία. Για να εκφράσουμε την τελική δύναμη του εργάτη πού είναι διαθέσιμη στην εφαρμογή μας χρησιμοποιούμε τον όρο της *διαθέσιμης υπολογιστικής ισχύος*. Η διαθέσιμη υπολογιστική ισχύ, προκύπτει αν διαιρέσουμε την υπολογιστική ισχύ με τον αριθμό των εργασιών στην ουρά εκτέλεσης του εργάτη.

Κεφάλαιο 3

Αξιολόγηση δυναμικών μεθόδων δρομολόγησης

Σε αυτό το κεφάλαιο ερευνούμε την εφαρμογή των απλών δυναμικών αλγορίθμων που περιγράφηκαν στο προηγούμενο κεφάλαιο, σε μια πραγματική επιστημονική εφαρμογή και πιο συγκεκριμένα σε μια αριθμητική εξομοίωση που υπολογίζει τις τροχιές φορτισμένων σωματιδίων στην μαγνητόσφαιρα της γης Η εξομοίωση αυτή εξετάζει τις διαταραχές της μαγνητόσφαιρας εξαιτίας των ηλεκτρικών πεδίων που προκαλούνται από ηλιακές υποκαταιγίδες και που απειλούν τις ανθρώπινες δραστηριότητες και τεχνολογικές υποδομές στον εγγύ διαστημικό χώρο. Ο χρόνος για τον υπολογισμό της τροχιάς κάθε σωματιδίου εξαρτάται από τις αρχικές συνθήκες τις εξομοίωσης και μπορεί διαφέρει σημαντικά σε διαφορετικά σωματίδια. Το γεγονός αυτό μπορεί να οδηγήσει σε ανισοκατανομή του υπολογιστικού φορτίου σε σενάρια παράλληλης εκτέλεσης με αποτέλεσμα την αύξηση του συνολικού χρόνου. Για αυτόν τον λόγο εφαρμόζουμε τεχνικές δυναμικής δρομολόγησης για την εξισορρόπηση του φορτίου σε ομοιογενείς, ετερογενείς και μη αφοσιωμένες παράλληλες πλατφόρμες κατανεμημένης μνήμης.

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο αξιολογούμε την χρήση δυναμικών αλγορίθμων δρομολόγησης ([12],[13], [90], [83]) για την βελτίωση της απόδοσης ενός κώδικα που εξομοιώνει την τροχιά φορτισμένων σωματιδίων κατά την διάρκεια μαγνητικών καταιγίδων στην γήινη μαγνητόσφαιρα. Ο στόχος της εξομοίωσης είναι να ξεκαθαρίσει την σχέση καταιγίδων - υποκαταιγίδων και να συμβάλει στην κατανόηση της συνδετική σχέση ήλιου-Γης. Κάθε απότομη αλλαγή στο μαγνητικό πεδίο της Γης ονομάζεται γεωμαγνητική δραστηριότητα. Διαταράξεις αυτού

του είδους προκαλούνται από ηλεκτρικά ρεύματα που ρέουν στην μαγνητόσφαιρα και ιονόσφαιρα της γης εξαιτίας του ηλιακού άνεμου. Παρατηρούνται δυο τύποι γεωμαγνητικής δραστηριότητας στο εγγύς διάστημα, μαγνητικές καταιγίδες και μαγνητοσφαιρικές υποκαταιγίδες. Οι μαγνητικές καταιγίδες προκαλούν μείωση της οριζόντιας συνιστώσας του γεωμαγνητικού πεδίου, παρατηρούνται σε οποιαδήποτε θέση σε μέσα γεωγραφικά πλάτη και διαρκούν μερικές μέρες. Κατά την διάρκεια μιας μαγνητικής καταιγίδας, η μεταφορά ενεργητικών ιόντων στην εσωτερική μαγνητόσφαιρα δημιουργεί ένα ισχυρό ηλεκτρικό ρεύμα που ρέει στο διάστημα γύρω από την Γη και ονομάζεται δακτυλιοειδής ρεύμα. Συνήθως, όταν οι καταιγίδες είναι έντονες, συνοδεύονται από υποκαταιγίδες. Οι μαγνητοσφαιρικές υποκαταιγίδες είναι έντονες μαγνητικές διαταράξεις που διαρκούν μερικές ώρες. Μια σειρά γεγονότων συμβαίνουν κατά την διάρκεια μια υποκαταιγίδας, όπως για παράδειγμα σελάϊκα τόξα και ρευματοχείμαροι σέλαως.

Έχει παρατηρηθεί ότι ο διαστημικός καιρός μπορεί να επηρεάσει τις δραστηριότητες των ανθρώπων και τις τεχνολογικές υποδομές στο εγγύς διάστημα αλλά και στο έδαφος. Οι συνέπειες των γεωμαγνητικών διαταράξεων περιλαμβάνουν την διείσδυση φορτισμένων σωματιδίων στα ηλεκτρονικά συστήματα των δορυφόρων, την φόρτιση των διαστημοπλοίων, τον κίνδυνο για την υγεία των αστροναυτών, παρεμβολές και διακοπή των τηλεπικοινωνιών, προβλήματα στο σύστημα παγκόσμιου εντοπισμού. Επίσης λόγο των γεωμαγνητικάεπαγόμενων ρευμάτων (geomagnetically-induced currents), έχουν αναφερθεί ζημιές σε γραμμές μεταφοράς ηλεκτρικής ενέργειας, υπερθέρμανση και κάψιμο μετασχηματιστών και ζημιές σε εκτεταμένες σωληνώσεις.

Η σχέση καταιγίδων-υποκαταιγίδων παραμένει ένα ανοιχτό θέμα την φυσική της μαγνητόσφαιρας. Η αποσαφήνιση της παραπάνω σχέσης θα βοηθήσει στην κατανόηση της επιρροής του Ήλιου πάνω στην Γη

Ο κώδικας της εξομοίωσης μας επιτρέπει να υπολογίσουμε την τροχιά φορτισμένων σωματιδίων που κινούνται κάτω από τις συνθήκες που επικρατούν στο γεωμαγνητικό πεδίο όταν κατά την διάρκεια μαγνητικών καταιγίδων, δημιουργούνται γεωμαγνητικές υποκαταιγίδες. Η ακριβής τροχιά των σωματιδίων εξαρτάται από πολλές παραμέτρους όπως το είδος του σωματιδίου, την χρονική στιγμή κατά την οποία ξεκινά την τροχιά του, την μορφή του μαγνητικού πεδίου την στιγμή αυτή και τις συντεταγμένες στις οποίες βρίσκεται. Ανάλογα με τις παραπάνω παραμέτρους, η τροχιά των σωματιδίων μπορεί να διαφέρει σημαντικά, μερικά σωματίδια μπορεί να σταματάν την κίνηση τους σύντομα, όταν εισέλθουν σε μια περιοχή που ονομάζεται μαγνητόπαυση, ενώ άλλα να συνεχίζουν την πορεία τους στην εσωτερική μαγνητόσφαιρα, συνεισφέροντας στον σχηματισμό του δακτυλιοειδούς ρεύματος. Όταν θέλουμε να υπολογίσουμε τις τροχιές ενός μεγάλου αριθμού σωματιδίων οι διαφορές στον όγκο υπολογισμών ανά σωματίδιο γίνονται εμφανείς. Για να αντιμετωπίσουμε αυτό

το πρόβλημα εφαρμόζουμε κλασικές δυναμικές μεθόδους δρομολόγησης, ώστε να κατανείμουμε ισομερώς το υπολογιστικό φορτίο στους διαθέσιμους επεξεργαστές. Πειραματιζόμαστε σε μια συστοιχία υπολογιστών μέσης κλίμακας και για τις περιπτώσεις των ομοιογενών - ετερογενών και μη αφοσιωμένων υπολογιστικών κόμβων. Σε κάθε περίπτωση τα πειραματικά αποτελέσματα αποδεικνύουν ότι οι δυναμικοί αλγόριθμοι βελτιώνουν σημαντικά την απόδοση και ειδικά σε περιπτώσεις που η ανομοιογένεια του συστήματος προστίθεται στις μη ομοιόμορφες υπολογιστικές απαιτήσεις της παρούσας εφαρμογής.

3.2 Ο κώδικας της εξομοίωσης

Η συνοπτική περιγραφή του φυσικού φαινομένου των υποκαταιγίδων βρίσκεται στο [107], ωστόσο εδώ δίνεται μια σύντομη περίληψη του κώδικα της εξομοίωσης, ώστε να γίνει κατανοητή η φύση του προβλήματος από την άποψη των υπολογισμών. Ο ψευδοκώδικας της εξομοίωσης για ένα χρονικό παράθυρο k χρονικών βημάτων πραγματοποιεί τις ακόλουθες τρεις βασικές λειτουργίες:

1. Αρχικοποίηση
    - Θέτουμε συνθήκες καταιγίδας στην μαγνητόσφαιρα
    - Θέτουμε τις παραμέτρους του σωματιδίου (είδος σωματιδίου, ενέργεια, αρχική θέση, γωνία, χρονική στιγμή έναρξης της πορείας μέσα στο μαγνητικό πεδίο)
    - Θέτουμε το παράθυρο παρατήρησης σε μέγεθος K και το τρέχων χρονικό βήμα $k = 0$
2. Υπολογισμός θέσης σωματιδίου
    - Υπολογίζουμε το μαγνητικό πεδίο με το μοντέλο του Tsyganenko [17]
    - Υπολογίζουμε το ηλεκτρικό πεδίο μεταφοράς με το μοντέλο του Volland-Stern[19],[16]
    - Υπολογίζουμε το επαγόμενο ηλεκτρικό πεδίο με την τεχνική του Delcourt[6]
    - Υπολογίζουμε την νέα θέση του σωματιδίου σύμφωνα με τον νόμο του Lorentz
    - $k = k + 1$
3. Τέλος του ελέγχου κίνησης

- Αν το σωματίδιο συνεχίζει την πορεία του και ισχύει $k < K$ πάμε στο βήμα 2

Παρατηρώντας τον παραπάνω ψευδοκώδικα μπορούμε να αντιληφθούμε τον λόγο για τον οποίο υπάρχουν μεγάλες διαφορές στον χρόνο υπολογισμού της τροχιάς διαφορετικών σωματιδίων. Υπάρχουν περιπτώσεις που το σωματίδιο σταματάει την πορεία του μετά από μερικά βήματα, συγκρουόμενο για παράδειγμα με την την επιφάνεια της Γης, ή μπορεί να παραμένει σε κίνηση για ολόκληρο το παράθυρο παρατήρησης. Η τροχιά κάθε σωματιδίου εξαρτάται από τις αρχικές συνθήκες που θέτουμε στο βήμα 1. Οι αρχικές συνθήκες παρέχονται από ένα σύνολο τιμών που περιγράφουν τον πληθυσμό σωματιδίων που επιθυμούμε να μελετήσουμε. Έχουμε δηλαδή, για τον υπολογισμό την τροχιάς κάθε σωματίδιου ένα βρόχο με μεταβλητό αριθμό επαναλήψεων. Όταν είναι επιθυμητή η μελέτη ενός πληθυσμού σωματιδίων, τότε προσθέτουμε ένα εξωτερικό βρόχο με αριθμό επαναλήψεων ίσο με τον αριθμό των σωματιδίων που θέλουμε να μελετήσουμε. Δεν λαμβάνουμε υπόψη μας αλληλεπιδράσεις μεταξύ των σωματιδίων, δηλαδή η τροχιά ενός σωματιδίου δεν εξαρτάται από άλλα σωματίδια, άρα το μοντέλο μας είναι αυτό των φωλιασμένων βρόχων χωρίς εξαρτήσεις. Είναι προφανές ότι όσο μεγαλώνει ο αριθμός των σωματιδίων, τόσο πιο χρονοβόρα γίνεται η συνολική εξομοίωση και μεγαλώνουν οι απαιτήσεις σε υπολογιστική ισχύ.

3.2.1 Εφαρμογή της δυναμικής δρομολόγησης.

Για να μειώσουμε τον χρόνο υπολογισμού των τροχιών των σωματιδίων θα εκτελέσουμε την εξομοίωση παράλληλα σε μια συστοιχία υπολογιστών. Η εξομοίωση αποτελείται από ένα βρόχο χωρίς εξαρτήσεις και με ανομοιόμορφες επαναλήψεις, μοντέλο ιδανικό για παραλληλοποίηση με την χρήση δυναμικών αλγορίθμων. Θα χρησιμοποιήσουμε τους αλγόριθμους Simple Self-scheduling (SS), Chunk Self-scheduling (CSS), Trapezoid Self-scheduling (TSS), Factoring Self-scheduling (FSS), Guided Self-scheduling (GSS). Για λόγους σύγκρισης παραθέτουμε επίσης και ένα απλό στατικό αλγόριθμο (static-chunking ή απλά block-scheduling) καθώς και την σταθμισμένη (weighted) έκδοση του. Ο στατικός αλγόριθμος υπολογίζει τις ομάδες επαναλήψεων που θα αναθέσει σε κάθε εργάτη διαιρώντας τον συνολικό αριθμό επαναλήψεων με τον αριθμό των διαθέσιμων εργατών και αναθέτει μια μόνο ομάδα επαναλήψεων σε κάθε εργάτη. Η σταθμισμένη έκδοση του στατικού αλγορίθμου αυξομειώνει το μέγεθος τις ομάδας επαναλήψεων ανάλογα με την υπολογιστική δύναμη του εργάτη. Οι αυτοδρομολογούμενοι αλγόριθμοι χωρίζουν τον χώρο επαναλήψεων σε μικρότερες ομάδες επαναλήψεων τις οποίες και μοιράζουν δυναμικά στους διαθέσιμους εργάτες. Κάθε στιγμιότυπο του χώρου επαναλήψεων είναι στην

περίπτωση μας, μια ξεχωριστή και ανεξάρτητη εκτέλεση του κώδικα της εξομοίωσης, ο οποίος καλείται με διαφορετικές αρχικές συνθήκες. Οι δυναμικοί αλγόριθμοι που χρησιμοποιήθηκαν προσφέρουν έναν ευέλικτο συμβιβασμό μεταξύ του κόστους δρομολόγησης με την ικανότητα ισοστάθμισης της κατανομής του υπολογιστικού φορτίου. Ανάλογα με τα χαρακτηριστικά τις συγκεκριμένης εφαρμογής (υπολογιστικό βάρος του σώματος του βρόχου, κατανομή του φορτίου μέσα στις ομάδες επαναλήψεων), που στην περίπτωση μας είναι ο κώδικας της εξομοίωσης, και τα χαρακτηριστικά της παράλληλης πλατφόρμας, μπορεί κάποιος αλγόριθμος να υπερτερεί έναντι των υπολοίπων όσον αφορά το συνολικό παράλληλο χρόνο. Αυτός ο αλγόριθμος θα βρεθεί πειραματικά στο επόμενο κεφάλαιο, στο οποίο παρέχουμε τα σχετικά πειραματικά αποτελέσματα για την παράλληλη εκτέλεση της εξομοίωσης.

3.3 Πειραματική αξιολόγηση

Πειραματική διάταξη. Σε αυτό το κεφάλαιο αξιολογούμε την απόδοση των αλγορίθμων δρομολόγησης με βάση τον παράλληλο χρόνο εκτέλεσης της εξομοίωσης. Ο κώδικας της εξομοίωσης έχει γραφτεί σε γλώσσα Fortran 90 και έχει μεταγλωττιστεί με την χρήση του μεταγλωττιστή ifort v. 8.1. Οι αλγόριθμοι δρομολόγησης έχουν υλοποιηθεί σε γλώσσα C++ με την χρήση κλήσεων στην βιβλιοθήκη μηνυμάτων MPI (MPICH v. 1.2.7) και ο μεταγλωττιστής που χρησιμοποιήθηκε είναι ο icc v. 8.1. Εκτελέσαμε τρεις σειρές πειραμάτων.

1. Αρχικά εκτελέσαμε την εξομοίωση σε μια ομοιογενή πλατφόρμα που αποτελούνταν από μια συστοιχία υπολογιστών των 16 κόμβων. Ο κάθε κόμβος περιλαμβάνει δυο τετραπύρινους Xeon, βασισμένους στην Core 2 αρχιτεκτονική της Intel (E53352GHz). Δύο πυρήνες σε κάθε πακέτο μοιράζονται $4MB$ L2 cache. Το δίκτυο διασύνδεσης ήταν Gigabit Ethernet. Σε αυτή την συστοιχία εκτελέσαμε 64 παράλληλες (-MPI-) διεργασίες.

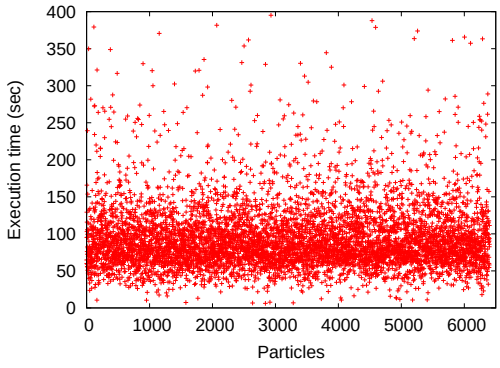
2. Στην δεύτερη περίπτωση εκτελέσαμε την εξομοίωση σε μια ετερογενή πλατφόρμα που περιελάμβανε 48 διεργασίες από την παραπάνω συστοιχία και 16 διεργασίες από μια δεύτερη συστοιχία που αποτελούνταν από 8 κόμβους με διπλούς Intel P4 Xeon (2-way SMP) επεξεργαστές με $1MB$ L1 cache.

3. Στην τρίτη περίπτωση επαναλάβαμε την πρώτη περίπτωση αφού φορτώσαμε τους κόμβους με τεχνητά εξωτερικά φορτία εξομοιώνοντας έτσι ένα μη αφοσιωμένο και φορτωμένο σύστημα. όλοι οι κόμβοι τρέχουν το λειτουργικό σύστημα Linux διανομής debian με πυρήνα 2.6.23.1. όλα τα

πειράματα επαναλήφθηκαν δέκα φορές και τα αποτελέσματα που παρουσιάζονται σε αυτό το κεφάλαιο είναι ο μέσος όρος αυτών των επαναλήψεων.

3.3.1 Αποτελέσματα.

Ομοιογενής Πλατφόρμα. Στην πρώτη σειρά πειραμάτων εκτελέσαμε την εξομοίωση με την χρήση των δυναμικών αλγορίθμων SS, TSS, FSS, GSS, και συγκρίναμε τα αποτελέσματα με αυτά που έδωσε ο στατικός αλγόριθμος. Για αυτό το πείραμα χρησιμοποιήσαμε 64 διεργασίες από την ομοιογενή συστοιχία των 16, τετραπύρηνων κόμβων. Υποθέτουμε ότι όλες οι διεργασίες έχουν την ίδια υπολογιστική ικανότητα, δηλαδή ότι όλες οι διεργασίες μπορούν να επεξεργαστούν τα ίδια δεδομένα στον ίδιο χρόνο, αλλά και ότι ο χρόνος υπολογισμού κάθε σωματιδίου είναι μεταβλητός. Οι χρόνοι υπολογισμού για ένα τυχαίο δείγμα 6400 σωματιδίων δίνεται στην εικόνα 3.1. Πιστεύουμε ότι αυτή η διακύμανση στους χρόνους υπολογισμού είναι που θα δώσει προβάδισμα την απόδοση των δυναμικών αλγορίθμων.

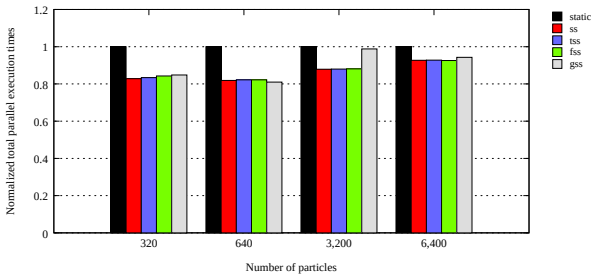


Σχήμα 3.1: Κατανομή των χρόνων εκτέλεσης ανά σωματίδιο για 6400 σωματίδια.

Όπως μπορούμε να δούμε από τα αποτελέσματα που δίνονται στην εικόνα 3.2, όλοι οι δυναμικοί αλγόριθμοι υπερνικούν σε απόδοση τον στατικό αλγόριθμο. Η βελτίωση της απόδοσης κυμαίνεται από 10% μέχρι 20%. Μπορούμε να δούμε ότι η βελτίωση της απόδοσης μειώνεται καθώς αυξάνεται ο αριθμός των σωματιδίων. Η εξήγηση για αυτό το γεγονός βρίσκεται στην κατανομή των χρόνων υπολογισμού των σωματιδίων. Όπως βλέπουμε στο σχήμα 3.1 η κατανομή των χρόνων είναι ομοιόμορφη. Όσο αυξάνεται ο αριθμός των σωματιδίων, ο χρόνος υπολογισμού των ομάδων σωματιδίων που αναθέτει ο στατικός αλγόριθμος προσεγγίζει μια μέση τιμή, η οποία είναι ίδια για όλες τις ομάδες σωματιδίων, έτσι όλες οι διεργασίες ολοκληρώνουν την εκτέλεση στον ίδιο πε-

ρίπου χρόνο. Αυτό δεν ισχύει όταν έχουμε μικρό αριθμό σωματιδίων και έτσι σε αυτή την περίπτωση η διαφορά στην απόδοση είναι μεγαλύτερη.

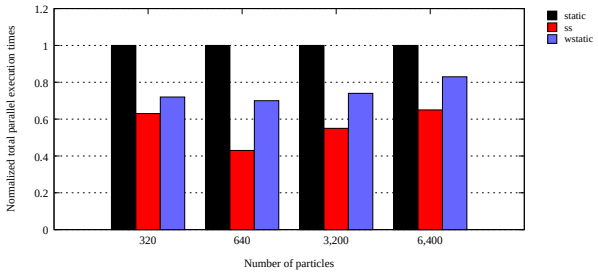
Μπορούμε επίσης να παρατηρήσουμε ότι σε όλες τις περιπτώσεις, ο αλγόριθμος Simple Self-scheduling (SS), έχει την καλύτερη (ή ίδια) απόδοση από τους υπόλοιπους δυναμικούς αλγόριθμους. Όπως είναι γνωστό, ο αλγόριθμος SS μπορεί να επιτύχει την καλύτερη κατανομή φορτίου από όλους τους άλλους δυναμικούς αλγόριθμους αυτής της κατηγορίας, με το τίμημα του αυξημένου κόστους δρομολόγησης. Στην παρούσα εφαρμογή, το κόστος δρομολόγησης, που είναι της τάξης των μερικών χιλιοστών του δευτερολέπτου, είναι ασήμαντο μπροστά στον χρόνο υπολογισμού κάθε επανάληψης ή ομάδας επαναλήψεων που κυμαίνεται σε μερικές δεκάδες ή εκατοντάδες δευτερόλεπτα. Με βάση τα παραπάνω, είναι λογικό ο αλγόριθμος SS να υπερτερεί σε απόδοση έναντι των υπόλοιπων αλγορίθμων, και έτσι φαίνεται ότι αυτός ο αλγόριθμος είναι ο πιο κατάλληλος για την δρομολόγηση εφαρμογών με αυτό τον συνδυασμό κόστους υπολογισμού - κόστους δρομολόγησης. Για αυτό τον λόγο στα επόμενα πειράματα θα περιοριστούμε στην σύγκριση του αλγόριθμου SS με τον στατικό αλγόριθμο.



Σχήμα 3.2: Σύγκριση των δυναμικών και των στατικών αλγορίθμων στην περίπτωση ομοιογενούς συστήματος για διαφορετικούς αριθμούς σωματιδίων.

Ετερογενής Πλατφόρμα. Σε αυτή την σειρά πειραμάτων προσθέσαμε ανομοιογένεια του συστήματος στην ήδη υπολογιστικά ανομοιογενή εφαρμογή. Δηλαδή εδώ δεν έχουμε μόνο διακύμανση στον χρόνο υπολογισμού των διαφορετικών επαναλήψεων αλλά και ανόμοια υπολογιστική ικανότητα ανά κόμβο. Αυτή την φορά χρησιμοποιήσαμε 64 διεργασίες, 48 από την προηγούμενη συστοιχία και 16 από την δεύτερη συστοιχία των 8 διπλοπύρηνων κόμβων. Για να προσδιορίσουμε την διαφορά στην υπολογιστική ικανότητα, εκτελέσαμε και στους δύο διαφορετικούς τύπους διεργασιών ένα πείραμα μικρής κλίμακας, που περιελάμβανε 10 όμοια σωματίδια. Με αυτό το πείραμα προσδιορίστηκε ότι ο πρώτος τύπος μπορεί να εκτελέσει 1.9 φορές περισσότερους υπολογισμούς από τον δεύτερο τύπο στον ίδιο χρόνο, δηλαδή έχουμε μια αναλογία στην υπολογι-

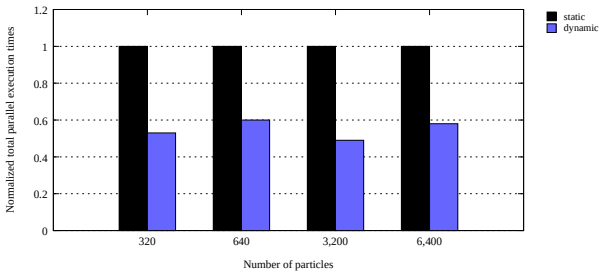
στική ικανότητα των δυο τύπων διεργασιών 1.9 : 1. Στην συνέχεια συγκρίθηκε η απόδοση του καλύτερου δυναμικού αλγορίθμου όπως προέκυψε από την πρώτη σειρά πειραμάτων, δηλαδή του SS με αυτή του στατικού αλγόριθμου και του σταθμισμένου στατικού αλγόριθμου. Ο σταθμισμένος στατικός αλγόριθμος είναι ίδιος με τον απλό στατικό με την διαφορά ότι αναθέτει πιο πολλά σωματίδια στις πιο γρήγορες διεργασίες, με αναλογία 1.9 : 1. Στο Σχήμα 3.3 μπορούμε να δούμε τους κανονικοποιημένους χρόνους εκτέλεσης για κάθε ένα από τους παραπάνω αλγόριθμους. Η βελτίωση της απόδοσης του SS σε σχέση με τον στατικό αλγόριθμο στη ετερογενή περίπτωση κυμαίνεται από 34% έως 46%. Η διαφορά μεταξύ του δυναμικού και του στατικού αλγόριθμου διευρύνεται σε σχέση με την ομοιογενή περίπτωση και αυτό εξηγείται από την αύξηση της ανισορροπία κατανομής του φορτίου που οφείλεται στην διαφορά της υπολογιστικής ικανότητας των υπολογιστικών κόμβων. Ο σταθμισμένος στατικός αλγόριθμος λαμβάνει υπόψη του αυτή την διαφορά και όπως είναι αναμενόμενο, αποδίδει καλύτερα από τον απλό στατικό αλγόριθμο. Ωστόσο, ο δυναμικός αλγόριθμος είναι καλύτερος και από τον σταθμισμένο στατικό (13% με 26%), και αυτό οφείλεται στο γεγονός ότι ακόμα και αν μπορούσαμε να ισοσταθμίσουμε τέλεια την υπολογιστική ικανότητα των δυο διαφορετικών τύπων διεργασιών, έχουμε πάλι ανισοκατανομή φορτίου που οφείλεται στην ίδια την εφαρμογή, όπως είδαμε στην πρώτη περίπτωση.



Σχήμα 3.3: Σύγκριση των δυναμικών και των στατικών αλγορίθμων στην περίπτωση ετερογενούς συστήματος για διαφορετικούς αριθμούς σωματιδίων.

Ομοιογενής πλατφόρμα με εξωτερικά φορτία. Στη τρίτη σειρά πειραμάτων χρησιμοποιήσαμε πάλι το ομοιογενές σύστημα της πρώτης περίπτωσης αλλά αυτή την φορά φορτώσαμε το σύστημα με εξωτερικό φορτίο, δηλαδή εκτελέσαμε ένα αριθμό, ανεξαρτήτων από τις διεργασίες του πειράματος στο σύστημα. Με τον τρόπο αυτό εξομοιώσαμε ένα μη αφοσιωμένο σύστημα. Κάναμε την απλή υπόθεση, η οποία είναι κοντά στην πραγματικότητα, ότι η υπολογιστική ικανότητα ενός υπολογιστικού κόμβου είναι αντιστρόφως ανάλογη με τον αριθμό διεργασιών που υπάρχουν στην ουρά έτοιμων διεργασιών του (run-queue). Σε κάθε κόμβο εκτελέσαμε μια γεννήτρια φορτίων, η οποία δημιουργεί

φορτία με χρόνο άφιξης και χρόνο ζωής που ακολουθούν μια συγκεκριμένη κατανομή poisson. Οι μέσοι χρόνοι άφιξης και ζωής είναι διαφορετικοί σε κάθε κόμβο έτσι ώστε να δημιουργηθεί ένα τελείως τυχαίο και απρόβλεπτο περιβάλλον, αρχίζοντας με ένα φορτίο που έρχεται κατά μέσο όρο κάθε ένα δέκα δευτερόλεπτα με χρόνο ζωής γύρω στα πέντε δευτερόλεπτα, μέχρι ένα φορτίο που έρχεται κάθε τρία δευτερόλεπτα με μέσο χρόνο ζωής τα είκοσι δευτερόλεπτα. Σε αυτή την περίπτωση δεν έχουμε σταθμισμένο στατικό αλγόριθμο αφού δεν έχουμε κάποια σαφή εικόνα για την κατάσταση του συστήματος, δηλαδή δεν ξέρουμε πόσο ακριβώς μειώνεται η υπολογιστική ικανότητα κάθε κόμβου κάτω από την επιρροή του εξωτερικού φορτίου. Τα αποτελέσματα δίνονται στο Σχήμα 3.4. Παρατηρούμε ότι ο δυναμικός αλγόριθμος είναι πάντα καλύτερος από τον στατικό, και η βελτίωση στην απόδοση δεν πέφτει ποτέ κάτω από 40%, με μέγιστη τιμή το 50%. Αυτή είναι η περίπτωση με την μεγαλύτερη διαφορά μεταξύ του δυναμικού και του στατικού αλγόριθμου και αυτό εξηγείται από το γεγονός ότι η ύπαρξη του εξωτερικού φορτίου δημιουργεί το πιο απρόβλεπτο και δυναμικό περιβάλλον, με την μεγαλύτερη πιθανότητα για ανισοκατανομή του υπολογιστικού φορτίου.



Σχήμα 3.4: Σύγκριση των δυναμικών και των στατικών αλγορίθμων στην περίπτωση υπερφορτωμένου ομοιογενούς συστήματος για διαφορετικούς αριθμούς σωματιδίων.

3.4 Συμπεράσματα

Σε αυτό το κεφάλαιο διερευνήσαμε την απόδοση των δυναμικών αλγορίθμων για την δρομολόγηση ενός αριθμητικού κώδικα ο οποίος εξομοιώνει τις τροχιές φορτισμένων σωματιδίων, όταν αυτά κινούνται στην μαγνητόσφαιρα κάτω από το σενάριο συνύπαρξης μαγνητικών καταιγίδων και υποκαταιγίδων. Η εφαρμογή αυτή είναι σημαντική για την πρόβλεψη του διαστημικού καιρού και των επιπτώσεων που αυτός έχει στην ζωή και τις τεχνολογικές υποδομές των ανθρώπων. Για να μοντελοποιήσουμε επαρκώς φαινόμενα που σχετίζονται με

την γεωμαγνητική δραστηριότητα πρέπει να εξομοιώσουμε την κίνηση ενός μεγάλου αριθμού σωματιδίων, με διαφορετικές αρχικές συνθήκες (ενέργειας, συντεταγμένες, κ.τ.λ.). Οι διαφορετικές αυτές συνθήκες οδηγούν σε σημαντικές διακυμάνσεις στον χρόνο εξομοίωσης/ υπολογισμού των διαφορετικών σωματιδίων, προκαλώντας ανισότητα στην κατανομή του υπολογιστικού φορτίου στους διαθέσιμους υπολογιστικούς κόμβους και μείωση της απόδοσης κατά την παράλληλη εκτέλεση της εξομοίωσης.

Για την δρομολόγηση της εξομοίωσης πειραματιστήκαμε με ένα στατικό και αρκετούς δυναμικούς αλγόριθμους (SS, TSS, FSS, GSS), οι οποίοι προσφέρουν διαφορετικές σχέση ικανότητας κατανομής φορτίου με κόστος δρομολόγησης. Η φύση του κώδικα της εξομοίωσης περιλαμβάνει μεγάλους και χρονοβόρους υπολογισμούς (της τάξης των 100 δευτερολέπτων ανά σωματίδιο), σχετικά ομοιόμορφα κατανεμημένους, αφού οι αρχικές συνθήκες επιλέγονται τυχαία βάση μιας ομοιόμορφης κατανομής για κάθε σωματίδιο.

Από τα αποτελέσματα συμπεραίνουμε ότι ακόμα και στην πιο απλή περίπτωση, η οποία είναι αυτή της ομοιογενούς συστοιχίας υπολογιστών και για μεγάλο αριθμό σωματιδίων, οι δυναμικοί αλγόριθμοι επιτυγχάνουν σημαντική βελτίωση έναντι των στατικών, της τάξης του 10%. Όταν ο αριθμός των σωματιδίων είναι σχετικά μικρός η διαφορά στην απόδοση αυξάνεται σημαντικά και αγγίζει το 20%. Η καλύτερη απόδοση επιτυγχάνεται από τον πιο απλό αλγόριθμο, δηλαδή τον SS, γεγονός που αποδίδεται στο μεγάλο βάρος τον υπολογισμών της εξομοίωσης και στο χαμηλό κόστος δρομολόγησης, το οποίο επιτρέπει στον SS να επιτύχει την καλύτερη δυνατή κατανομή φορτίου.

Η θετική συμβολή των δυναμικών αλγορίθμων γίνεται πιο εμφανής όταν στο σύστημα προστίθενται και άλλοι παράγοντες ανομοιογένειας. Στην περίπτωση της ετερογενούς πλατφόρμας ο SS πετυχαίνει βελτίωση 26% έναντι του σταθμισμένου στατικού αλγόριθμου, ενώ στο μη αφοσιωμένο περιβάλλον η διαφορά στατικού - δυναμικού είναι κοντά στο 50%.

Κεφάλαιο 4

Δυναμική δρομολόγηση σε ετερογενή δίκτυα υπολογιστών

Σε αυτό το κεφάλαιο παρουσιάζεται ένας δυναμικός αλγόριθμος δρομολόγησης που δημιουργήθηκε για ετερογενή δίκτυα υπολογιστών. Το βασικό του χαρακτηριστικό είναι ότι είναι βασισμένο στην χρήση TCP sockets, οι οποίες προσφέρουν προσφέρουν ένα αξιόπιστο επίπεδο επικοινωνιών χωρίς την ανάγκη για κάποια ειδική βιβλιοθήκη ανταλλαγής δεδομένων όπως το MPI, το οποίο χρησιμοποιείται στους αλγόριθμους που παρουσιάζονται στα επόμενα κεφάλαια. Ο αλγόριθμος αυτός μπορεί έτσι να εφαρμοστεί σε οποιοδήποτε δίκτυο υπολογιστών που υποστηρίζει το πρωτόκολλο TCP. Η χρήση κάποιου πιο γρήγορου ασυνδεσμικού (conectionless) πρωτοκόλλου επικοινωνιών όπως το UDP δεν προσφέρει αξιόπιστες επικοινωνίες και έτσι δεν χρησιμοποιήθηκε στον αλγόριθμο που παρουσιάζεται σε αυτό το κεφάλαιο.

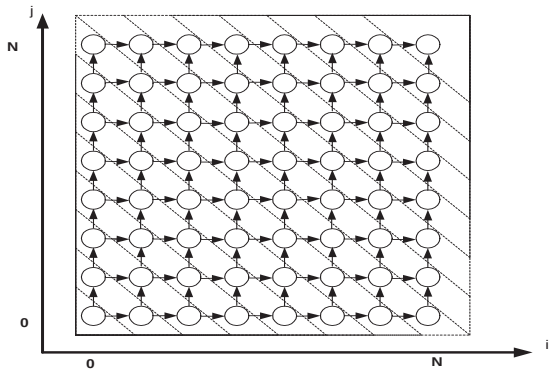
4.1 Εισαγωγή

Τα δίκτυα υπολογιστών έχουν αρχίσει να προσελκύουν το ενδιαφέρον των ερευνητών σαν εναλλακτικές πλατφόρμες για την ανάπτυξη και εκτέλεση παράλληλων εφαρμογών, λόγο του χαμηλού κόστους συγκρότησης τους σε σχέση με τα παραδοσιακά παράλληλα μηχανήματα. Τα δίκτυα αυτά παρ όλη την πρόοδο που έχει γίνει στις τεχνολογίες διασύνδεσης παρουσιάζουν ακόμα μεγάλες καθυστερήσεις στην μεταφορά δεδομένων (latency και delay) και έτσι το κόστος επικοινωνίας παραμένει υψηλό. Αυτός είναι και ο λόγος που σε πλατφόρμες αυτής της μορφής προτιμούνται μοντέλα που παρέχουν παραλληλισμό χονδρούκόκκου (coarse-grain parallelism) σε αντίθεση με πλατφόρμες μοιραζόμενης

μνήμης όπου ο παραλληλισμός ψιλού-κόκκου (fine-grain parallelism) προσφέρει καλύτερη απόδοση. Ο coarse-grain παραλληλισμός επιτυγχάνεται με την χρήση αλγορίθμων ομαδοποίησης όπως το tiling [51], [34], [48], οι οποίοι ομαδοποιούν πολλούς υπολογισμούς με στόχο να μειώσουν την συχνότητα των απαιτούμενων επικοινωνιών. Μερικοί από αυτούς τους αλγόριθμους προσπαθούν επίσης να χειριστούν το ζήτημα της ανομοιογένειας με το να διαμορφώνουν το μέγεθος των ομάδων υπολογισμών ανάλογα με την δύναμη των διαθέσιμων επεξεργαστών [84]. Ωστόσο οι αλγόριθμοι αυτοί είναι στατικοί και δεν μπορούν να χειριστούν δυναμικά χαρακτηριστικά του συστήματος όπως την διακύμανση του φόρτου εργασίας ή την διαθεσιμότητα των επεξεργαστών. Οι δυναμικοί αλγόριθμοι δρομολόγησης [90], [83], [37], [27], [91], [10] από την άλλη μεριά είναι προσηλωμένοι σε προβλήματα που δεν περιέχουν εξαρτήσεις δεδομένων αφήνοντας έτσι ένα κενό για την κατηγορία προβλημάτων που περιέχουν εξαρτήσεις. Ο αλγόριθμος που παρουσιάζεται σε αυτό το κεφάλαιο συμπληρώνει αυτό το κενό με την μορφή ενός δυναμικού αλγορίθμου που στοχεύει σε προβλήματα με εξαρτήσεις. Χρησιμοποιεί ομαδοποίηση των υπολογισμών για να παράγει χονδροκομμένο παραλληλισμό "πολλαπλών επιπέδων" και στην συνέχεια ένα μοντέλο συντονιστή-εργάτη για να δρομολογήσει τους ομαδοποιημένους υπολογισμούς σε ένα ετερογενές δίκτυο υπολογιστών.

4.2 περιγραφή Αλγορίθμου δρομολόγησης

Ο χώρος επαναλήψεων του φωλιασμένου βρόχου χωρίζεται σε ομάδες επαναλήψεων (chunks of iterations). Κάθε ομάδα χωρίζεται σε μικρότερες ομάδες έτσι ώστε να έχουμε πολλαπλά επίπεδα ομάδων, με τις ομάδες κάθε επίπεδου να περιέχουν διαφορετικό αριθμό επαναλήψεων. Αυτές οι ομάδες τοποθετούνται στους κόμβους ενός τετραδικού δέντρου. Η τοποθέτηση των ομάδων στο δέντρο γίνεται σύμφωνα με τις εξαρτήσεις δεδομένων, δηλαδή ομάδες που μπορούν να εκτελεστούν νωρίτερα τοποθετούνται πιο αριστερά στο δέντρο. Χρησιμοποιώντας αυτό το δέντρο ο δρομολογητής μπορεί να παράγει τις απαραίτητες επικοινωνιακές συνδέσεις μεταξύ επεξεργαστών έτσι ώστε να ικανοποιήσει τις εξαρτήσεις δεδομένων σύμφωνα με μια μέθοδο που θα αναλυθεί στη συνέχεια. Οι ομάδες εργασιών ανατίθενται δυναμικά στους επεξεργαστές σύμφωνα με τις εξαρτήσεις δεδομένων, χρησιμοποιώντας ένα μοντέλο συντονιστή-εργάτη. Η εξισορρόπηση φορτίου (load-balancing) επιτυγχάνεται με δυο τρόπους: Κάθε εργάτης λαμβάνει μια καινούργια ομάδα επαναλήψεων αφού τελειώσει αυτή που είχε αναλάβει νωρίτερα, και επιπροσθέτως η εργασία που λαμβάνει είναι ανάλογη με την σχετική υπολογιστική του ισχύ, δηλαδή την ικανότητα του να εκτελεί υπολογισμούς σε σχέση με τους υπόλοιπους εργάτες. Επίσης, για να μειωθεί το κόστος επικοινωνίας που σχετίζεται με τις εξαρτήσεις δεδομένων, οι εργά-



Σχήμα 4.1: αναπαράσταση του $2-D$ χώρου επαναλήψεων

τες ανταλλάσσουν δεδομένα μεταξύ τους απευθείας, χωρίς την μεσολάβηση του συντονιστή.

Στην συνέχεια η παρουσίαση θα περιοριστεί για λόγους απλότητας σε βρόχους δυο διαστάσεων, ωστόσο η μεθοδολογία μπορεί να επεκταθεί σε περισσότερες διαστάσεις με μερικές μετατροπές. Επίσης για λόγους απλότητας υποθέτουμε ότι ο χώρος επαναλήψεων είναι τετράγωνος αν και αυτή η υπόθεση δεν είναι δεσμευτική. Το παράδειγμα στο οποίο θα βασισθεί η ανάλυση αλλά και τα πειραματικά αποτελέσματα δίνεται παρακάτω.

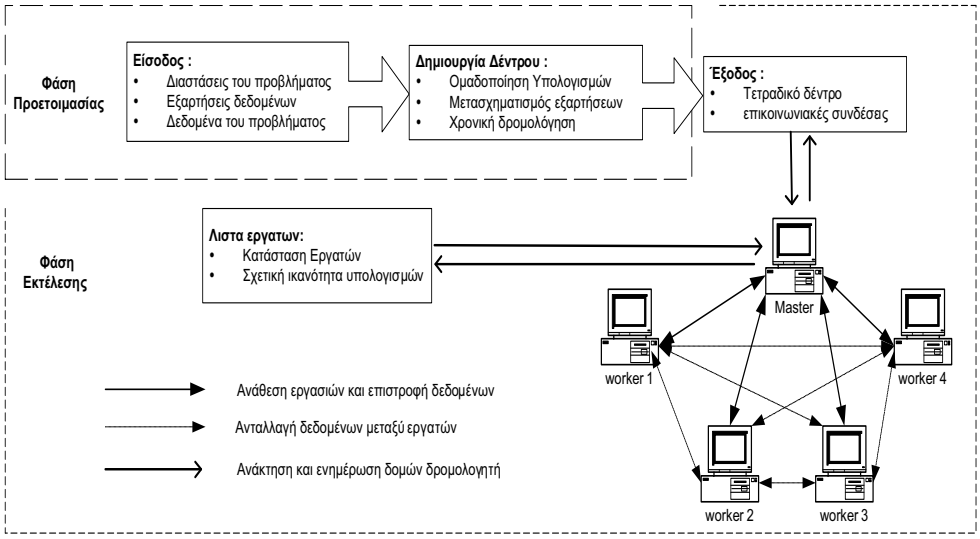
for $i = 0$ *to* N **do**
  for $j = 0$ *to* N **do**
    $A[i,j] = A[i-1,j] + A[i,j-1]$ $:S_2$
  end
end

Αυτός ο διπλός βρόχος αντιστοιχεί στο 2-D χώρο επαναλήψεων $I^2 = (i,j)|0 \leq i,j \leq N$. Κάθε σημείο αυτού του χώρου αντιστοιχεί σε μια επανάληψη του βρόχου. Όπως μπορούμε να δούμε στο σώμα του βρόχου υπάρχουν οι σταθερές και ομοιόμορφες εξαρτήσεις οι οποίες συνοψίζονται στον πίνακα εξαρτήσεων

$$D = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

4.3 Ανάλυση

Ο Δυναμικός αλγόριθμος δρομολόγησης παρουσιάζεται συνοπτικά στον παρακάτω ψευδοκώδικα.



Σχήμα 4.2: διάγραμμα αλγορίθμου δρομολόγησης και μοντέλο master-slave/peer-to-peer.

Πλευρά συντονιστή.

- Φάση προετοιμασίας:
  - Ανάγνωση των παραμέτρων του προβλήματος, δηλαδή των διαστάσεων (χώρος επαναλήψεων) και των εξαρτήσεων δεδομένων.
  - Δημιουργία του κατάλληλου δέντρου: Ομαδοποίηση επαναλήψεων σε ομάδες υπολογισμών (tasks), εξαγωγή εξαρτήσεων εργασιών από τις εξαρτήσεις δεδομένων με την χρήση του κατάλληλου μετασχηματισμού και ανεύρεση της πιθανής σειράς εκτέλεσης (time scheduling).
  - Δημιουργία λίστας εργατών. Εγγραφή των διαθέσιμων εργατών και ταξινόμηση τους ανάλογα με την σχετική ικανότητα υπολογισμών τους.
- Φάση εκτέλεσης:
  Όσο υπάρχουν διαθέσιμες εργασίες:
  - Επιλογή ενός ελεύθερου εργάτη από την λίστα εργατών. Επιλογή της κατάλληλης εργασίας από το δέντρο, ανάλογα με την σχετική ικανότητα υπολογισμών του εργάτη και τις εξαρτήσεις ομάδων.
  - Εξαγωγή των απαιτούμενων επικοινωνιακών συνδέσμων για την ικανοποίηση των εξαρτήσεων.

  - Αποστολή της εργασίας στον εργάτη και των ονομάτων των εργατών από τους οποίους πρέπει ο εργάτης να ζητήσει δεδομένα.
  - Έλεγχος για αιτήματα εγγραφής νέων εργατών στο σύστημα.

- Παρασκήνιο:
  - Μια διεργασία παρασκηνίου ελέγχει την κατάσταση στην οποία βρίσκονται οι διαθέσιμοι εργάτες. Όταν ένας εργάτης δηλώσει ότι έχει ολοκληρώσει την εκτέλεση μιας εργασίας γίνεται ξανά διαθέσιμος στο σύστημα για να λάβει μια καινούργια εργασία.

Πλευρά εργάτη.

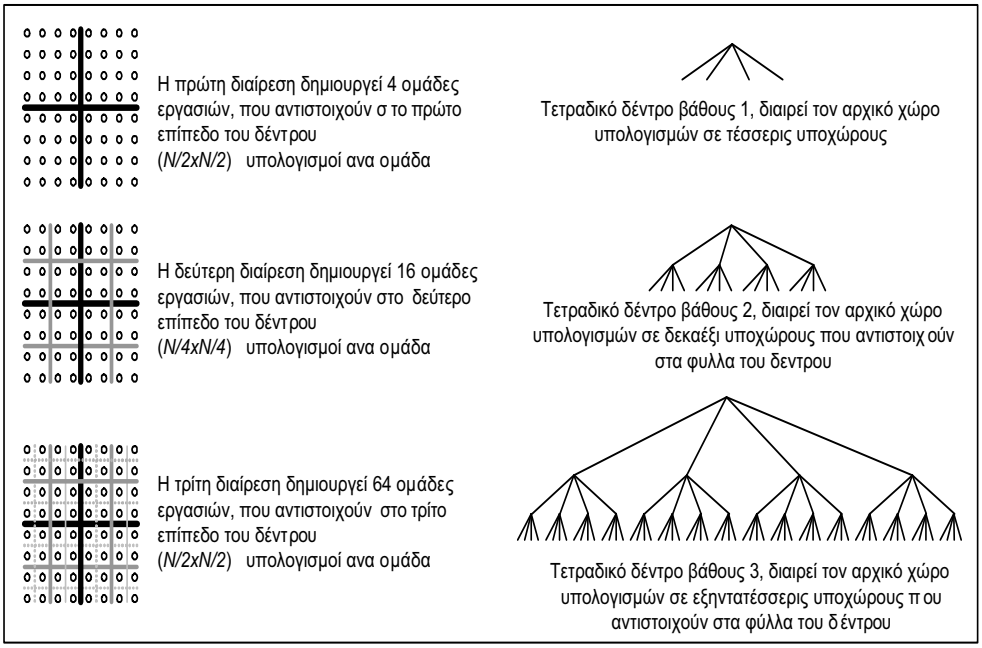
- Φάση εκτέλεσης:
  - Αποστολή αιτήματος εγγραφής στον συντονιστή και αναφορά της ικανότητας υπολογισμών του.
  - Αναμονή για λήψη νέας εργασίας από τον συντονιστή και για ονόματα εργατών από τους οποίους πρέπει να ζητήσει δεδομένα.
  - Έλεγχος για ειδοποίηση ολοκλήρωσης/ τερματισμού επεξεργασίας από τον συντονιστή και σε αυτήν την περίπτωση έξοδος. Αλλιώς, αν έχουν δοθεί ονόματα εργατών για επικοινωνία, αποστολή αιτήσεων για τα απαιτούμενα δεδομένα.
  - Αναμονή για απάντηση στις αιτήσεις δεδομένων και λήψη δεδομένων.
  - Εκτέλεση υπολογισμών.
  - Πακετάρισμα δεδομένων που πρέπει να αποσταλούν σε άλλους εργάτες και τοποθέτηση τους σε μια λίστα
  - Αποστολή μηνύματος ολοκλήρωσης εκτέλεσης στον συντονιστή και επιστροφή στην αναμονή για νέα εργασία.

- Παρασκήνιο:
  - Μια διεργασία παρασκηνίου ελέγχει και εξυπηρετεί αιτήσεις για αποστολή δεδομένων σε άλλους εργάτες.

4.3.1 Ομαδοποίηση των επαναλήψεων.

Ο χώρος επαναλήψεων αποδομείται σε ομάδες ύστερα από διαδοχικές διαιρέσεις. Αρχικά διαιρείται σε τέσσερις ίσους υποχώρους. Κάθε ένας από αυτούς τους υποχώρους διαιρείται ξανά σε τέσσερις ίσους υποχώρους και η διαδικασία

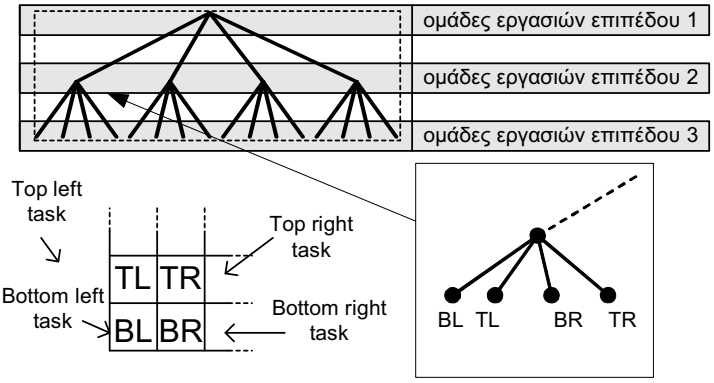
επαναλαμβάνεται μέχρι να φτάσουμε στο επιθυμητό βαθμό ανάλυσης (granularity). Η δομή που επιλέχθηκε για την αναπαράσταση του αποικοδομημένου αυτού χώρου είναι το τετραδικό δέντρο. Η μορφή αυτή αναπαράστασης χώρων δυο διαστάσεων είναι συνηθισμένη σε περιοχές εφαρμογών όπως η επεξεργασία εικόνας [85] και n-body simulations [53]. Το τετραδικό δέντρο χωρίζει φυσικά τον χώρο επαναλήψεων σε τέσσερις υποχώρους που αντιστοιχούν σε τέσσερις ομάδες επαναλήψεων. Προσθέτοντας ένα επίπεδο στο δέντρο, κάθε υποχώρος μοιράζεται σε τέσσερις υποχώρους σε αντιστοιχία με την διαίρεση που αναφέρθηκε νωρίτερα. Κάθε επίπεδο του δέντρου περιέχει συνολικά τον ίδιο αριθμό υπολογισμών με όλα τα άλλα επίπεδα, η διαφορά είναι στον αριθμό και στο μέγεθος των ομάδων υπολογισμών. Κάθε ομάδα του n-οστού επιπέδου περιέχει τετραπλάσιο αριθμό υπολογισμών από ότι αυτές του επιπέδου $n+1$. Επίσης, το επίπεδο $n+1$ περιέχει τετραπλάσιο αριθμό ομάδων από ότι το επίπεδο n. Από αυτή την άποψη, η ρίζα του δέντρου αντιπροσωπεύει ολόκληρο τον υπολογισμό, ενώ τα φύλα τις μικρότερες διαθέσιμες ομάδες, οι οποίες καλούνται και ‘ ατομικές εργασίες ’.



Σχήμα 4.3: Διαίρεση του χώρου επαναλήψεων και το τετραδικό δέντρο .

Χρησιμοποιώντας αυτό το δέντρο είναι εύκολο για τον δρομολογητή να αναθέσει στους διάφορους εργάτες, ομάδες υπολογισμών που ταιριάζουν στην σχετική υπολογιστική τους ισχύ. Το τετραδικό δέντρο παρέχει βασικά διάφο-

ρες δεξαμενές εργασιών (task pools) όπως φαίνεται στο σχήμα 4.4, μια για κάθε επίπεδο του δέντρου, από τις οποίες ο δρομολογητής μπορεί να αναθέσει εργασίες. Είναι προφανές ότι σε αυτές τις δεξαμενές, υπάρχουν πολλαπλά αντίγραφα των εργασιών τι οποίες ο δρομολογητής πρέπει να τις αναθέσει μόνο μια φορά.



Σχήμα 4.4: δεξαμενές εργασιών και συμβάσεις ονομασίας.

4.3.2 Μετασχηματισμός εξαρτήσεων και δίαυλοι επικοινωνίας.

Οι εξαρτήσεις δεδομένων υπονοούν ότι πρέπει να γίνουν ανταλλαγές δεδομένων μεταξύ των εργατών. Το πρόβλημα όμως εδώ είναι ότι η ανάθεση εργασιών στους επεξεργαστές είναι δυναμική και έτσι δεν μπορεί να υπάρξει γνώση των απαραίτητων επικοινωνιακών συνδέσμων, μεταξύ των εργατών, πριν γίνει η ανάθεση των εργασιών. Παρατηρώντας πάλι το παράδειγμα μας μπορούμε να δούμε ότι οι εξαρτήσεις περιγράφονται στον πίνακα εξαρτήσεων D. Επίσης η διαίρεση του χώρου επαναλήψεων μπορεί να παρομοιασθεί σαν μετασχηματισμός tiling με πίνακες tiling για το πρώτο και το δεύτερο επίπεδο αντίστοιχα.

$$H_1 = \begin{pmatrix} 2/N & 0 \\ 0 & 2/N \end{pmatrix} \text{ και } H_2 = \begin{pmatrix} 4/N & 0 \\ 0 & 4/N \end{pmatrix}$$

Αυτό μας δίνει την δυνατότητα να μετασχηματίσουμε τις εξαρτήσεις σημείων σε εξαρτήσεις ομάδων χρησιμοποιώντας αρχές δανεισμένες από το tiling. Το σύνολο των εξαρτήσεων μεταξύ των ομάδων μας δίνεται από τον παρακάτω μετασχηματισμό [48].

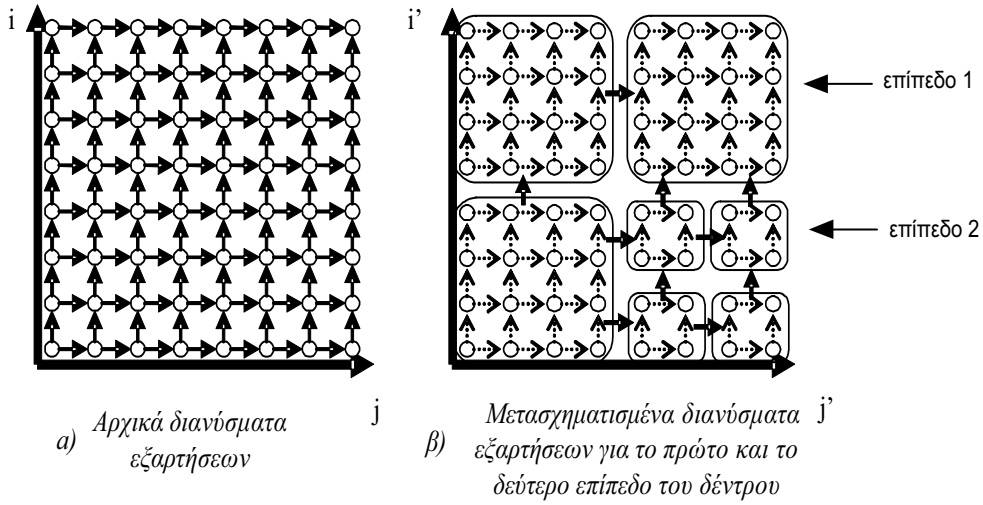
$$Dt = \{\lfloor (H(i+d)) \rfloor \mid 0 \leq H * i \leq 1, d \in D, i \in I^2\}$$

Οι μετασχηματισμένες εξαρτήσεις (εξαρτήσεις ομάδων) φαίνονται στο σχήμα 4.5 και είναι $D_t = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Αυτά τα διανύσματα εξάρτησης ισχύουν για όλα τα επίπεδα του δέντρου δηλαδή για τους πίνακες tiling όλων των επιπέδων του δέντρου , εφόσον οι προβολές των διανυσμάτων εξάρτησης στην διάσταση k δεν είναι μεγαλύτερες από τα διανύσματα μεγέθους $B = (B_1, \ldots, B_n)$ της κάθε ομάδας, όπου B_k είναι ο αριθμός επαναλήψεων που περιέχει κάθε ομάδα στην διάσταση k και n είναι ο αριθμός των διαστάσεων. Στην πράξη τα μεγέθη των ομάδων πρέπει να είναι σημαντικά μεγαλύτερα από τα διανύσματα εξάρτησης ώστε να αποφεύγονται συχνές επικοινωνίες και να επιτυγχάνεται ένας αποδεκτός λόγος επικοινωνιών-υπολογισμών. Αυτό εξασφαλίζει ότι μπορούν να υπάρχουν μόνο τρεις κατηγορίες διανυσμάτων εξαρτήσεων ομάδων:

1. Από μια οποιαδήποτε ομάδα σε μια ομάδα ακριβώς στα δεξιά, με διάνυσμα $d_t = (1\ 0)$ που ονομάζεται 'αριστερή εξάρτηση '.

2. Από μια οποιαδήποτε ομάδα σε μια ομάδα ακριβώς πάνω, με διάνυσμα $d_t = (0\ 1)$ που ονομάζεται 'κάτω εξάρτηση '.

3. Από μια οποιαδήποτε ομάδα σε μια ομάδα ακριβώς πάνω και δεξιά, με διάνυσμα $d_t = (1\ 1)$ που ονομάζεται 'διαγώνια εξάρτηση '.



Σχήμα 4.5: αρχικά και μετασχηματισμένα διανύσματα εξαρτήσεων .

Ο δρομολογητής προκειμένου να καθορίσει τους απαραίτητους επικοινωνιακούς συνδέσμους, δηλαδή τα ονόματα των εργατών που πρέπει να επικοινωνήσουν για να ανταλλάξουν δεδομένα (μερικά αποτελέσματα), καταγράφει τα

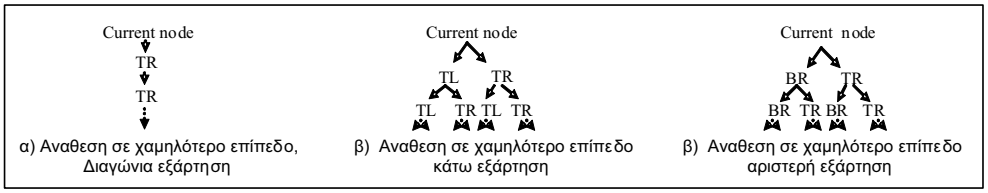
ονόματα των εργατών όταν τους αναθέτει εργασία. Για να βρει με ποιους εργάτες πρέπει να επικοινωνήσει ο κάθε εργάτης αφαιρεί από τις συντεταγμένες της ομάδας προς ανάθεση, τα μετασχηματισμένα διανύσματα εξάρτησης και καταλήγει στις συντεταγμένες και στα ονόματα των εργατών που κατέχουν τα δεδομένα που χρειάζονται για τον υπολογισμό της καινούργιας ομάδας. Τα ονόματα αυτά τα αποστέλλει στον εργάτη μαζί με την ομάδα υπολογισμών που έχει αναλάβει και έτσι ο εργάτης μπορεί να επικοινωνήσει και να ζητήσει τα δεδομένα που χρειάζεται. Αφού ο δρομολογητής μπορεί αν αναθέσει εργασίες στους εργάτες από διαφορετικά επίπεδα του δέντρου, η αφαίρεση μπορεί να μην υποδεικνύει απευθείας τους κατάλληλους εργάτες. Σε αυτήν την περίπτωση πρέπει να κάνουμε την ακόλουθη ανάλυση εξαρτήσεων:

- Ομάδες υπολογισμών από το ίδιο επίπεδο στο δέντρο. Σε αυτή την περίπτωση η αφαίρεση οδηγεί κατευθείαν στους αντίστοιχους εργάτες.

- Ομάδες υπολογισμών από υψηλότερο επίπεδο του δέντρου. Οι ομάδες που αναζητάμε είναι υποομάδες μιας μεγαλύτερης ομάδας, που βρίσκετε σε υψηλότερο επίπεδο του δέντρου και ανήκει σε ένα εργάτη. Εδώ αρκεί να ανεβούμε τα επίπεδα του δέντρο ωσότου βρούμε που έχει ανατεθεί αυτή η μεγαλύτερη ομάδα.

- Ομάδες υπολογισμών από χαμηλότερο επίπεδο του δέντρου. Σε αυτή την περίπτωση κάθε ομάδα έχει χωριστεί σε μικρότερες ομάδες σε χαμηλότερα επίπεδα του δέντρου και ανήκει σε πολλούς εργάτες. Η ανάλυση είναι ανάλογη με το είδος της εξάρτησης (αριστερή, κάτω ή διαγώνια εξάρτηση). Αν η εξάρτηση είναι διαγώνια τότε το δέντρο απλοποιείται σε μια λίστα όπως φαίνεται στο σχήμα 4.6. Αρκεί να διασχίσουμε την λίστα μέχρι να βρούμε που έχει ανατεθεί η ομάδα που αναζητάμε. Αντίστοιχα, αν η εξάρτηση είναι αριστερή ή κάτω, τότε το τετραδικό δέντρο μετατρέπεται σε δυαδικό όπως φαίνεται στο σχήμα 4.6. Στη περίπτωση της διαγώνιας εξάρτησης καταλήγουμε σε ένα μόνο εργάτη ενώ στις άλλες δυο περιπτώσεις σε περισσότερους εργάτες.

Η απλοποίηση από τετραδικό δέντρο σε δυαδικό ή σε απλή λίστα είναι δυνατή λόγο του ότι υπάρχουν εξαρτήσεις μόνο μεταξύ γειτονικών ομάδων υπολογισμών όπως περιγράφηκε νωρίτερα. Αν έχουμε μια κάτω εξάρτηση τότε μας απασχολούν μόνο οι πάνω ομάδες κάθε επιπέδου του τετραδικού δέντρου δηλαδή οι ομάδες TR, TL, αν η εξάρτηση είναι αριστερή τότε μας απασχολούν μόνο οι δεξιές ομάδες, δηλαδή οι TR, BR, και αν είναι διαγώνια τότε μόνο οι ομάδες TR.



Σχήμα 4.6: Απλοποίηση του τετραδικού δέντρου για την ανεύρεση επικοινωνιακών συνδέσμων.

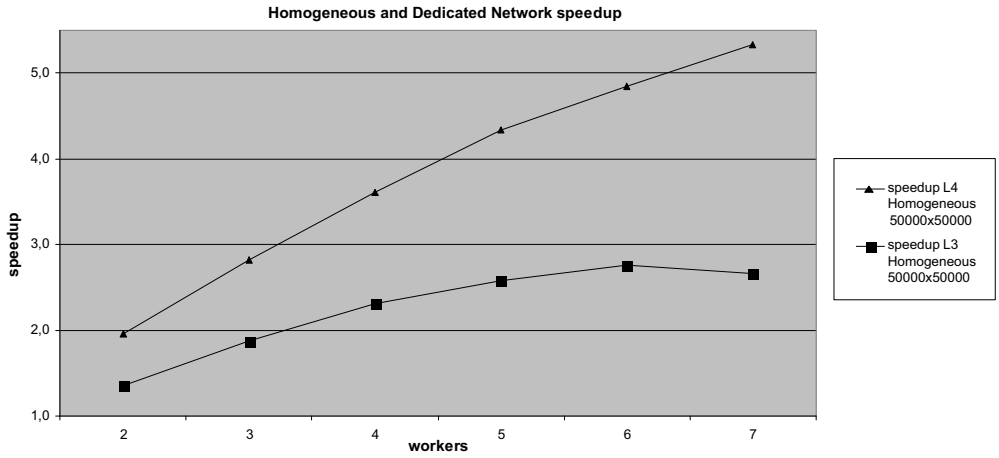
4.4 Πειραματική αξιολόγηση

Πειραματική διάταξη. Η υλοποίηση του αλγορίθμου έγινε σε γλώσσα C με την χρήση TCP sockets για την επικοινωνία και τα πειράματα βασίζονται στο παράδειγμα που παρουσιάστηκε νωρίτερα στην ενότητα 4.2. Σαν αποτέλεσμα των μετρήσεων, δίνεται η επιτάχυνση του παράλληλου αλγορίθμου, η οποία ορίζεται ως ο χρόνος εκτέλεσης του σειριακού αλγορίθμου, στον πιο γρήγορο εργάτη, προς τον παράλληλο χρόνο. Χρησιμοποιήθηκαν δυο τύποι εργατών προερχόμενοι από δυο διαφορετικές ομοιογενείς συστοιχίες υπολογιστών σε περιβάλλον Linux (Linux clusters). Η πρώτη συστοιχία (kids) συντίθεται από επεξεργαστές PIII 500MHZ με 256MB RAM και η δεύτερη (twins) από επεξεργαστές PIII 800MHz με 256MB RAM. Υποθέτουμε ότι οι εργάτες από την δεύτερη συστοιχία (twins) μπορούν να εκτελέσουν περίπου τον διπλάσιο αριθμό υπολογισμών από ότι οι εργάτες από τη πρώτη συστοιχία (kids) στον ίδιο χρόνο. Και οι δυο συστοιχίες είναι συνδεδεμένες μέσω διασύνδεσης fast-Ethernet. Χρησιμοποιήσαμε στο σύνολο 8 κόμβους από την πρώτη συστοιχία και 5 από την δεύτερη, με ένα κόμβο της δεύτερης να ενεργεί σαν συντονιστής. Σε μερικές περιπτώσεις δημιουργήσαμε τεχνητό φόρτο σε κάποιους κόμβους με σκοπό να προκαλέσουμε επιπλέον ανομοιογένεια. Σε αυτή την περίπτωση η ικανότητα υπολογισμών των φορτωμένων κόμβων μειώνεται στο μισό.

4.4.1 Αποτελέσματα.

1η περίπτωση: Ομοιογενές και αφοσιωμένο δίκτυο υπολογιστών. Για την πρώτη περίπτωση η δοκιμή έγινε με ένα παράδειγμα $50000x50000$ επαναλήψεων με 2 ως 7 εργάτες (kids) και τα αποτελέσματα παρουσιάζονται στο γράφημα 4.7. Σε αυτή την περίπτωση που έχουμε ομοιογενείς εργάτες με την ίδια περίπου ικανότητα υπολογισμών και έτσι οι εργασίες δίνονται από το ίδιο επίπεδο του δέντρου.

Για να δοκιμάσουμε την διαφορά που προκύπτει στην επιτάχυνση ανάλογα με το μέγεθος των εργασιών η εκτέλεση επαναλήφθηκε με ανάθεση εργασιών



Σχήμα 4.7: Επιτάχυνση σε σχέση με τον σειριακό αλγόριθμο σε ομογενές και αφοσιωμένο δίκτυο επεξεργαστών, για το παράδειγμα εφαρμογής.

από το τρίτο (L_3) και το τέταρτο (L_4) επίπεδο του δέντρου. Το μέγεθος ομάδων υπολογισμών για το τρίτο και το τέταρτο επίπεδο είναι 6250×6250 και 3125×3125 επαναλήψεις αντίστοιχα. Είναι φανερό από το γράφημα ότι η επιτάχυνση που επιτυγχάνεται στο τέταρτο επίπεδο είναι πολύ μεγαλύτερη από ότι στο τρίτο επίπεδο. Αυτό οφείλεται εν μέρη στο ότι στο τέταρτο επίπεδο υπάρχουν πολύ περισσότερες ομάδες εργασιών (64 ομάδες στο τρίτο έναντι 256 στο τέταρτο) και έτσι μπορεί να επιτευχθεί πολύ περισσότερος παραλληλισμός. Αν πάμε ακόμα πιο κάτω στο πέμπτο επίπεδο του δέντρου δεν έχουμε αύξηση την επιτάχυνσης και αυτό οφείλεται στο γεγονός ότι το μέγεθος των εργασιών γίνεται πολύ μικρό και έτσι το κόστος των επικοινωνιών υπερισχύει της βελτίωσης του διαθέσιμου παραλληλισμού. Είναι γνωστό ότι όσο αυξάνεται το μέγεθος των ομάδων μειώνεται η συχνότητα και άρα το κόστος των επικοινωνιών αλλά μειώνεται ταυτόχρονα και ο διαθέσιμος παραλληλισμός.

Ένα επιπλέον συμπέρασμα είναι ότι η δυνατότητα κλιμάκωσης του συστήματος μειώνεται όσο μειώνονται οι διαθέσιμες ομάδες εργασιών. Στο γράφημα μπορούμε να δούμε ότι για πάνω από 6 εργάτες η επιτάχυνση μειώνεται όταν η ανάθεση γίνεται από το τρίτο επίπεδο, πράγμα που δεν ισχύει για το τέταρτο επίπεδο του δέντρου. Λόγο της ύπαρξης των εξαρτήσεων υπάρχει ένας περιορισμός στον αριθμό των εργασιών που μπορούν να εκτελούνται ταυτόχρονα, και έτσι η ύπαρξη περισσότερων εργατών δεν συμβάλει στην μείωση του παράλληλου χρόνου, αντιθέτως συμβάλει αρνητικά γιατί αυξάνει τις απαραίτητες επικοινωνίες γιατί αυξάνεται η διασπορά των δεδομένων και η πιθανότητα απαραίτητα δεδομένα να ανήκουν σε άλλους εργάτες. Στον πίνακα 4.1 παρου-

σιάζεται η κατανομή του υπολογιστικού φόρτου στους εργάτες. Ο αλγόριθμος εξισορρόπησης φαίνεται αποδοτικός αφού σε όλες τις περιπτώσεις οι εργάτες εκτελούν περίπου τον ίδιο αριθμό εργασιών.

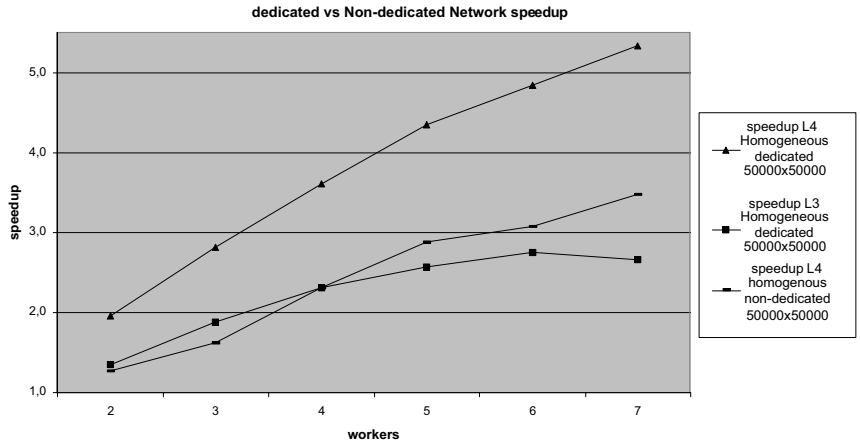
Πίνακας 4.1: Κατανομή φόρτου εργασίας (σε αριθμό ομάδων εργασιών) για την περίπτωση του ομογενούς και αφοσιωμένου δικτύου, για το παράδειγμα εφαρμογής

# εργατών	50000×50000	
	ομάδες από επίπεδο 4 ανά επεξεργαστή	ομάδες από επίπεδο 3 ανά επεξεργαστή
2	128 128	32 32
3	87 86 83	20 22 22
4	65 66 61 64	16 16 16 16
5	53 52 50 49 52	13 12 12 13 14
6	45 44 42 43 42 40	9 11 12 9 12 11
7	38 36 36 37 37 34 38	8 10 10 9 9 9 9

2η περίπτωση: Ομοιογενές, μη-αφοσιωμένο δίκτυο υπολογιστών. Στη δεύτερη περίπτωση παρατηρούμε την συμπεριφορά του αλγορίθμου σε ένα ομοιογενές αλλά όχι αφοσιωμένο δίκτυο υπολογιστών. Το παράδειγμα και αυτή την φορά έχει ένα χώρο 50000×50000 επαναλήψεων και η ανάθεση εργασιών γίνεται από το τέταρτο επίπεδο του δέντρου. Προσθέτουμε ένα φορτωμένο ή μη φορτωμένο εργάτη κάθε φορά, όπως φαίνεται στον πίνακα 4.2, μέχρι να φτάσουμε τους 7 εργάτες (πάλι kids). Στο γράφημα 4.8 μπορούμε να δούμε ότι ο αλγόριθμος δουλεύει ικανοποιητικά αφού κάθε μη-φορτωμένος εργάτης αναλαμβάνει περίπου τον διπλάσιο αριθμό εργασιών από ότι ο φορτωμένος εργάτης και η επιτάχυνση αυξάνεται καθώς προσθέτουμε καινούργιους εργάτες.

Αν συγκρίνουμε τα αποτελέσματα αυτής της περίπτωσης με την προηγούμενη διαπιστώνουμε ότι η προσθήκη φορτωμένων εργατών προσφέρει κάποια αύξηση της επιτάχυνσης, μικρότερη όμως από αυτή που προσφέρει η πρόσθεση ενός μη φορτωμένου εργάτη. Για παράδειγμα δυο φορτωμένοι και δυο μη-φορτωμένοι εργάτες έχουν μεγαλύτερη επιτάχυνση από ότι οι δυο μη-φορτωμένοι εργάτες αλλά μικρότερη από ότι τρεις μη-φορτωμένοι εργάτες.

3η περίπτωση: Ετερογενές και αφοσιωμένο δίκτυο υπολογιστών. Στην τελευταία περίπτωση τρέχουμε ένα πείραμα με 12800×12800 επαναλήψεις χρησιμοποιώντας εργάτες και από τις δυο συστοιχίες (kids και twins) ώστε να δημιουργήσουμε ένα δίκτυο ετερογενών υπολογιστών. Προσθέτουμε σταδιακά αργούς και γρήγορους εργάτες με την ίδια σειρά που προσθέσαμε φορτωμένους και μη-φορτωμένους εργάτες στην προηγούμενη περίπτωση (πί-



Σχήμα 4.8: Επιτάχυνση σε σχέση με τον σειριακό αλγόριθμο σε ομογενές αφοσιωμένο και ομογενές μη-αφοσιωμένο δίκτυο επεξεργαστών, για το παράδειγμα εφαρμογής.

Σχήμα 4.9: Επιτάχυνση σε σχέση με τον σειριακό αλγόριθμο σε ετερογενές και αφοσιωμένο δίκτυο επεξεργαστών για το παράδειγμα εφαρμογής.

νακας 4.3). Σε αυτή την περίπτωση αναθέτουμε εργασίες από δυο διαφορετικά επίπεδα του δέντρου, από το τέταρτο για τους αργούς εργάτες (kids) και από το τρίτο για τους γρήγορους εργάτες (twins). Συγκρίνουμε την απόδοση αυτής της προσέγγισης σε σχέση με την προηγούμενη που αναθέταμε εργασίες από ένα μόνο επίπεδο του δέντρου, είτε από το τρίτο είτε από το τέταρτο. Όπως φαίνεται από το γράφημα 4.9 η επιτάχυνση είναι πάντα καλύτερη όταν αναθέτουμε εργασίες από διαφορετικά επίπεδα του δέντρου.

Πίνακας 4.2: Κατανομή φόρτου εργασίας (σε αριθμό ομάδων εργασιών) σε φορτωμένους και μη-φορτωμένους εργάτες για την περίπτωση του ομογενούς και αφοσιωμένου δικτύου, για το παράδειγμα εφαρμογής

# εργατών	φορτωμένοι	μη-φορτωμένοι	# ομάδων φορτωμένων/ μη-φορτωμένων	επιτάχυνση
2	1	1	85 / 171	1.27
3	2	1	64 66 / 126	1.61
4	2	2	43 43 / 85 84	2.31
5	2	3	32 35 / 62 65 62	2.88
6	3	3	30 30 30 / 57 56 53	3.07
7	3	4	25 26 26 / 43 45 47 44	3.51

Πίνακας 4.3: Κατανομή φόρτου εργασίας (σε ποσοστό επί του συνόλου των υπολογισμών) σε γρήγορους και αργούς εργάτες για την περίπτωση του ομογενούς και αφοσιωμένου δικτύου

εργάτες	γρήγοροι/ αργοί	κατανομή φορτίου $L_3 - L_4(\%)$	κατανομή φορτίου $L_3(\%)$	κατανομή φορτίου $L_4(\%)$
2	1/1	71/29	78/22	55/45
3	1/2	58/21 21	62/19 19	38/32 30
4	2/2	35 35/15 15	40 30 /16 14	36 36/14 14
5	2/3	35 38/ 9 9 9	25 31/14 16 14	24 25 /17 17 17
6	3/3	28 27 27/ 6 6 6	20 20 25/13 11 11	23 23 24/10 10 10
7	3/4	28 25 22/7 6 6 6	19 20 20/9 11 11 10	22 20 22/9 9 9 9

4.5 Συμπεράσματα

Σε αυτό το κεφάλαιο παρουσιάστηκε ένας νέος αλγόριθμος για την δρομολόγηση φωλιασμένων βρόχων με ομοιόμορφες εξαρτήσεις σε ετερογενή και μη αφοσιωμένα δίκτυα υπολογιστών. Ο αλγόριθμος αυτός είναι ο πρώτος στην βιβλιογραφία που εφαρμόζει δυναμικές μεθόδους δρομολόγησης και παραλληλισμό χονδρού κόκκου για την δρομολόγηση βρόχων που περιέχουν εξαρτήσεις σε κατανεμημένα συστήματα. Ο σχεδιασμός του είναι γενικός και για τις επικοινωνίες βασίζεται σε tcp/ip sockets έτσι ώστε να μπορεί να εφαρμοστεί σε οποιοδήποτε δίκτυο υπολογιστών. Τα πειραματικά αποτελέσματα αποδεικνύουν ότι οι δυναμικές μέθοδοι μπορούν να εφαρμοστούν αποδοτικά στα συστήματα αυτά. Από τα πειράματα φαίνεται η σημασία της εξισορρόπησης του υπολογιστικού φορτίου, καθώς και του μεγέθους των ομάδων εργασιών που αναθέτονται στους διαθέσιμους εργάτες. Στην συνέχεια θα ασχοληθούμε με συστήματα που είναι πιο κατάλληλα για την εκτέλεση παράλληλων υπολογισμών, δηλαδή

σε συστοιχίες υπολογιστών.

Κεφάλαιο 5

Δυναμική δρομολόγηση πολλαπλών φάσεων

Στο προηγούμενο κεφάλαιο παρουσιάσαμε ένα νέο δυναμικό αλγόριθμο για την δρομολόγηση φωλιασμένων βρόχων με ομοιόμορφες εξαρτήσεις (UDLs) σε χαλαρά συνδεδεμένα δίκτυα υπολογιστών. Σε αυτό το κεφαλαίο ακολουθούμε μια διαφορετική προσέγγιση με στόχο να παρουσιάσουμε μια σειρά από πιο απλούς αλγόριθμους που βασίζονται στους ήδη γνωστούς δυναμικούς αυτοδρομολογούμενους αλγόριθμους. Οι αλγόριθμοι αυτοί είναι σημαντικά απλούστεροι στην λειτουργία τους σε σύγκριση με τον αλγόριθμο του προηγούμενου κεφαλαίου, αλλά δεν μπορούν να εφαρμοστούν σε UDLs γιατί δεν υποστηρίζουν επικοινωνίες μεταξύ των εργατών. Επεκτείνουμε τρεις αλγόριθμους (CSS,TSS,DTSS) προσθέτοντας τους ρουτίνες που επιτρέπουν την επικοινωνία μεταξύ των εργατών, έτσι ώστε να μην παραβιάζονται οι εξαρτήσεις δεδομένων. Η υλοποίηση των αλγορίθμων αυτών έχει γίνει με την χρήση της βιβλιοθήκης ανταλλαγής μηνυμάτων MPI,γεγονός που βελτιώνει την απόδοση τους σε σχέση με την προηγούμενη προσέγγιση που βασίζονταν σε TCP sockets.

5.1 Εισαγωγή

Οι αυτοδρομολογούμενοι αλγόριθμοι όπως έχουν παρουσιαστεί στο παρελθόν δεν μπορούν να εφαρμοστούν σε βρόχους που περιέχουν εξαρτήσεις. Ας πάρουμε ως παράδειγμα την εξίσωση διάδοσης της θερμότητας (heat equation), της οποίας ο ψευδοκώδικας δίνεται παρακάτω:

Αν εφαρμόσουμε ένα από του υπάρχοντες αλγορίθμους για να παραλληλοποιήσουμε τον παραπάνω βρόχο τότε ο χώρος επαναλήψεων θα χωριστεί σε ομάδες επαναλήψεων, οι οποίες θα ανατεθούν στους διαθέσιμους εργάτες. Αυτοί οι εργάτες στην συνεχεία θα υπολογίσουν το κομμάτι που τους έχει

for $i = l_i$ *to* U_i **do**
  for $j = l_j$ *to* U_j **do**
    $A[i,j] = \frac{1}{4}(A[i-1,j] + A[i,j-1] + A'[i+1,j] + A'[i,j+1])$
  end
end

ανατεθεί ανεξάρτητα από τους υπόλοιπους εργάτες. Όμως λόγο της ύπαρξης των εξαρτήσεων, οι εργάτες θα πρέπει να ανταλλάξουν δεδομένα. Αυτό όμως δημιουργεί ένα πρόβλημα αφού οι τρέχουσες μέθοδοι δυναμικής δρομολογήσεις δεν υποστηρίζουν επικοινωνίες μεταξύ των εργατών, παρά μόνο επικοινωνίες μεταξύ των εργατών και του συντονιστή. Άρα, για να εφαρμόσουμε δυναμικούς αλγόριθμους σε βρόχους με εξαρτήσεις πρέπει να εφαρμόσουμε παράλληλα μια μέθοδο που θα επιτρέπει την επικοινωνία μεταξύ των εργατών έτσι ώστε να μην παραβιάζονται οι εξαρτήσεις και να διατηρείται η απαραίτητη σειρά εκτέλεσης.

5.1.1 Ο αλγόριθμος DTSS

Σε αυτή την ενότητα δίνεται μια σύντομη περιγραφή του αλγόριθμου distributed Trapazoid Self-Scheduling (DTSS), [4], ο οποίος βελτιώνει τις επιδώσεις του απλού TSS σε ετερογενή συστοιχίες υπολογιστών. Ο DTSS χρησιμοποιεί τη έννοια των εικονικών επεξεργαστών. Πιο συγκεκριμένα θεωρεί ότι κάθε εργάτης P_k αντιστοιχεί σε ένα αριθμό εικονικών εργατών ανάλογα με την σχετική υπολογιστική του δύναμη ή αλλιώς την εικονική επεξεργαστική του δύναμη (virtual processing power - VP_k). Για παράδειγμα ένας φυσικός εργάτης που μπορεί να επεξεργάζεται δεδομένα δυο φορές γρηγορότερα από τους υπόλοιπους εργάτες αντιστοιχεί με δυο εικονικούς εργάτες και έχει $VP_k = 2$. Ακόμα χρησιμοποιεί την διαθέσιμη επεξεργαστική ισχύ, η οποία προκύπτει από την εικονική δύναμη ενός εργάτη προς τον αριθμό των εργασιών στην ουρά εκτέλεσης του ($A = \frac{vp}{\#Q}$). Η διαφορά του DTSS με τον TSS βρίσκεται στο ότι ο DTSS αναθέτει ομάδες υπολογισμών στους εικονικούς και όχι στους φυσικούς εργάτες. Τα βήματα που ακολουθούν οι δυο αλγόριθμοι είναι ανάλογα, Ο DTSS χρησιμοποιεί το μεγεθος της πρώτης και της τελευταίας ομάδας (F, L) για να υπολογίσει τον αριθμό βημάτων N και τον παράγοντα μείωσης D. Η επιλογή των αρχικών και τελικών ομάδων επαναλήψεων συνήθως δίνεται από : $F = \frac{|J|}{2A}$ και $L = 1$, και ο αριθμός βημάτων $N = \frac{(2\times|J|)}{(F+L)}$. Έτσι το μέγεθος της ομάδας επαναλήψεων δίνεται από τον τύπο: $C_i = A_k \times (F - D \times (S_k - 1 + \frac{(A_k-1)}{2})$, όπου $S_k - 1 = A_1 + .. + A_k - 1$. Παρατηρούμε ότι για την περίπτωση αφοσιωμένης συστοιχίας υπολογιστών έχουμε την περίπτωση $A_k = VP_k$. Αν έχουμε ισοδύναμους εργάτες τότε για όλους ισχύει $VP_k = 1$ και ο DTSS δίνει

τα ίδια μεγέθη ομάδων επαναλήψεων με τον TSS. Διαφορετικά τα μεγέθη των ομάδων επαναλήψεων είναι ανάλογα με την διαθέσιμη δύναμη των εργατών και ο αλγόριθμος DTSS αναθέτει σε γρηγορότερους εργάτες μεγαλύτερες ομάδες επαναλήψεων.

Ο πίνακας 5.1.1 δίνει τα μεγέθη των ομάδων επαναλήψεων που υπολογίστηκαν από τρεις αλγόριθμους δρομολόγησης: CSS,TSS και DTSS. Η διάσταση δρομολόγησης σε αυτό το παράδειγμα είναι 5000 σημεία ενώ υποθέτουμε ότι χρησιμοποιήθηκαν 10 εργάτες οι οποίοι ανήκουν σε τρεις κατηγορίες: 3 γρήγοροι εργάτες με $VP = 1$, 3 μεσαίοι εργάτες με $VP = 0,66$ και 3 αργοί εργάτες με $VP = 0,33$. Επίσης έχουμε δυο περιπτώσεις, την αφοσιωμένη «dedicated) στην οποία κανείς εργάτης δεν έχει επιπλέον φορτίο και την μη-αφοσιωμένη (non-dedicated) στην οποία οι 3 πιο αργοί εργάτες έχουν φορτωθεί έτσι ώστε η διαθέσιμη υπολογιστική ισχύ τους να πέσει στο μισό. Οι αλγόριθμοι CSS και TSS δεν λαμβάνουν υπόψη τους το επιπλέον φορτίο οπότε δεν υπάρχει διαφοροποίηση στα μεγέθη των ομάδων επαναλήψεων που υπολογίζουν.

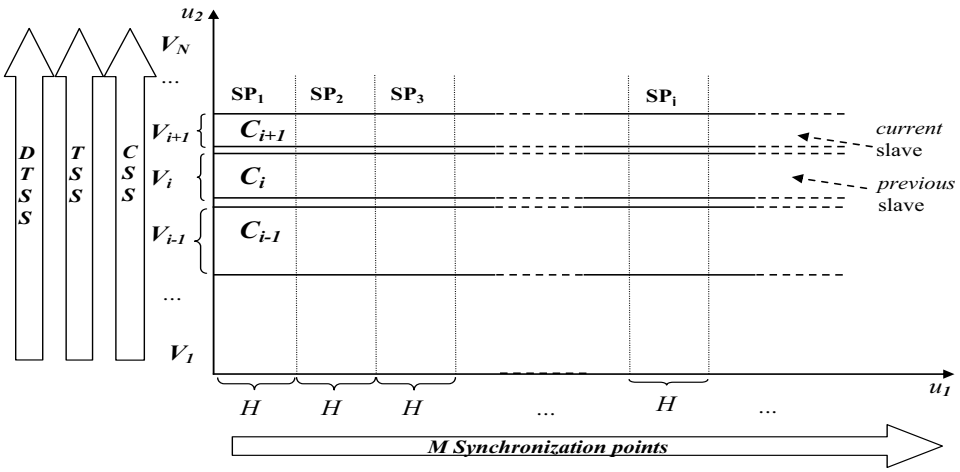
Πίνακας 5.1: Μεγέθη ομάδων επαναλήψεων που δίνουν οι αλγόριθμοι CSS,TSS και DTSS για διάσταση δρομολόγησης 5000 σημείων και για 10 εργάτες.

Αλγόριθμος	Ομάδες επαναλήψεων (Chunks)
CSS	300 300 300 300 300 300 300 300 300 300 300 300 300 300 300 300 200
TSS	250 244 238 232 226 220 214 208 202 196 190 184 178 172 166 154 148 142 136 130 124 118 112 106 100 94 88 82 76 70 40
DTSS αφοσιωμένο	402 393 384 63 62 61 241 237 233 229 336 327 318 52 51 50 197 193 189 185 270 261 252 14
DTSS μη-αφοσιωμένο	402 393 384 251 247 61 241 59 58 229 336 327 318 207 203 50 197 48 47 185 270 261 226

5.2 DMPS

Ο αλγόριθμος που περιγράφεται σε αυτό το κεφάλαιο προτείνει μια ευέλικτη μέθοδο επικοινωνίας και η οποία εφαρμόζεται σε τρεις γνωστούς αυτοδρομολογούμενους αλγόριθμους self-scheduling: CSS, TSS και $DTSS$. Σε γενικές γραμμές για την δρομολόγηση ενός φωλιασμένου βρόχου n διαστάσεων επιλέγουμε την μεγαλύτερη διάσταση, την οποία αποκαλούμε διάσταση συγχρονισμού (U_s), και την αμέσως μεγαλύτερη την οποία αποκαλούμε διάσταση δρομολόγησης (U_c). Στην διάσταση δρομολόγησης εφαρμόζουμε ένα από τους παραπάνω αυτοδρομολογούμενους αλγόριθμους, ενώ στην διάσταση συγχρονισμού εισάγουμε σημεία συγχρονισμού, στα οποία οι εργάτες μπορούν να αν-

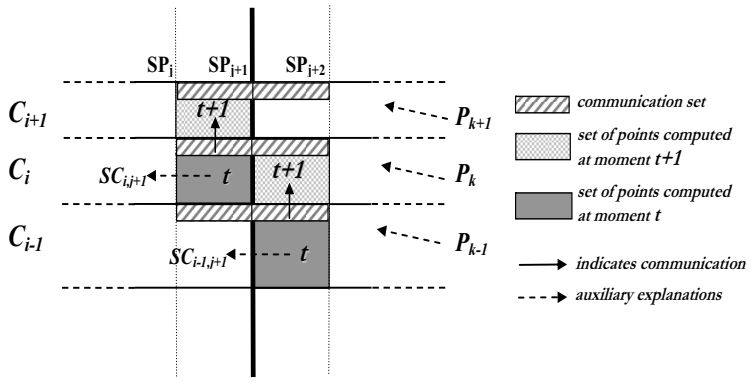
ταλλάσσουν δεδομένα όπως φαίνεται στο σχήμα 5.1. Έτσι έχουμε δυναμική δρομολόγηση, με τους εργάτες να εκτελούν υπολογισμούς σε μια μορφή εκτέλεσης σωλήνωσης (pipelining).



Σχήμα 5.1: Οι διαστάσεις δρομολόγησης (u_2) και συγχρονισμού (u_1) σε ένα χώρο δυο διαστάσεων.

Σε αυτό το σημείο είναι απαραίτητο να δώσουμε τους παρακάτω ορισμούς τους οποίους θα χρησιμοποιήσουμε και σε επόμενα κεφάλαια.

- SP: σε κάθε *chunk* εισάγουμε M σημεία συγχρονισμού (SP) κατανεμημένα ομοιόμορφα στην διάσταση συγχρονισμού u_1.
- H είναι το διάστημα μεταξύ δυο SPς (το H είναι το ίδιο για όλες τις ομάδες επαναλήψεων).
- Ο *τρέχον* (current) εργάτης είναι είναι αυτός στον οποίο έχει ανατεθεί η ομάδα επαναλήψεων i στο τρέχον βήμα δρομολόγησης, ενώ ο *προηγούμενος* (previous) εργάτης έχει αναλάβει την ομάδα επαναλήψεων $i-1$, ωσότου ανατεθεί η ομάδα $i+1$ σε ένα νέο εργάτη, ο οποίος και θα ονομαστεί εκ νέου *τρέχον* εργάτης.
- $SC_{i,j}$ είναι το σύνολο των επαναλήψεων στην ομάδα i, μεταξύ των SP_{j-1} και SP_j.
- *send-to*: είναι η ταυτότητα του εργάτη στον οποίο ο P_k πρέπει να στείλει δεδομένα. ενώ *receive-from* είναι η ταυτότητα του εργάτη από το οποίο ο P_k πρέπει να λάβει δεδομένα.



Σχήμα 5.2: Σημεία συγχρονισμού και τα βήματα της σωλήνωσης (pipelining).

Η εκτέλεση σωλήνωσης φαίνεται στο σχήμα 5.2, όπου οι ομάδες επαναλήψεων $i-1, i, i+1$ έχουν ανατεθεί στους εργάτες P_{k-1}, P_k, P_{k+1}. Σε αυτό το σχήμα τα κουτιά με την ίδια σκίαση υπολογίζονται στο ίδιο βήμα δρομολόγησης από διαφορετικούς εργάτες. Όταν ο P_k φτάσει στο σημείο συγχρονισμού SP_{j+1} στέλνει στον P_{k+1} μόνο τα δεδομένα που χρειάζεται ο P_{k+1} για να αρχίσει τον υπολογισμό του $SC_{i+1,j+1}$. Τα δεδομένα που έλαβε ο P_{k+1} αφορούν μόνο τις επαναλήψεις του $SC_{i,j+1}$ από τις οποίες εξαρτιούνται οι επαναλήψεις του $SC_{i+1,j+1}$, σύμφωνα με τα διανύσματα εξαρτήσεων. Αντίστοιχα, ο P_k λαμβάνει από τον P_{k-1} τα δεδομένα που χρειάζεσαι για να προχωρήσει στον υπολογισμό του $SC_{i,j+2}$. Εδώ πρέπει να σημειωθεί ότι οι εργάτες δεν φτάνουν στα ίδια σημεία συγχρονισμού στο ίδιο βήμα λόγο της εκτέλεσης σωλήνωσης. Για παράδειγμα, ο P_k φτάνει στο SP_{j+1} νωρίτερα από τον P_{k+1} και αργότερα από τον P_{k-1}.

Η ευελιξία της μεθόδου επικοινωνίας στηρίζεται σε δυο σημεία:

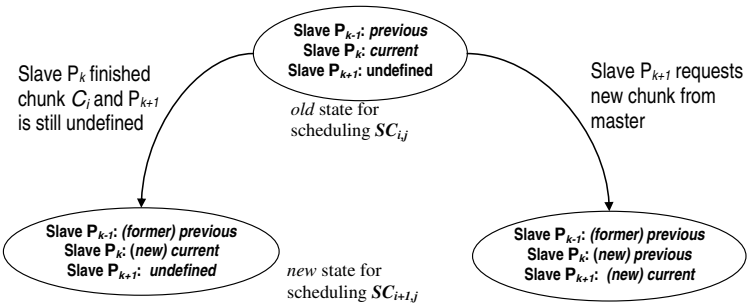
1. Στην περίπτωση που ο εργάτης P_k φτάσει σε ένα συγκεκριμένο σημείο συγχρονισμού, ας πούμε στο $SP_{i,j+1}$, και δεν έχει οριστεί ο εργάτης P_{k+1}, τότε ο P_k αποθηκεύει τα δεδομένα που θα χρειαστούν από τον P_{k+1}, μέχρι ο τελευταίος να οριστεί, δηλαδή μέχρι ένας νέος εργάτης αναλάβει τον υπολογισμό μιας καινούργιας ομάδας επαναλήψεων. Ο εργάτης P_k προχωράει κανονικά στον υπολογισμό της υπόλοιπης ομάδας που έχει αναλάβει. Αυτό σημαίνει ότι όταν ο ορισθεί ο P_{k+1}, θα μπορέσει να λάβει από τον P_k όλα τα δεδομένα που χρειάζεται και έχουν υπολογισθεί ως τότε από τον P_k. Οπότε, ο P_{k+1} δεν θα περιμένει στα αντίστοιχα σημεία συγχρονισμού αφού θα έχει διαθέσιμα τα δεδομένα που χρειάζεται. Επιπλέον, αφού ο P_k θα είναι τουλάχιστον $j+1$ σημεία

συγχρονισμού μπροστά από τον P_{k+1}, ο P_{k+1} θα έχει πάντα τα δεδομένα που χρειάζεται από τον P_k πριν φτάσει στο στο αντίστοιχο σημείο συγχρονισμού.

2. Στην περίπτωση που ο P_{k+1} δεν είναι ορισμένος την στιγμή που ο P_k φτάσει στο τελευταίο σημείο συγχρονισμού, αποθηκεύει τα δεδομένα που θα έστελνε στον P_{k+1} ,και στην συνέχεια κάνει αίτηση στον συντονιστή για να αναλάβει μια νέα ομάδα, αν οριστεί ο ίδιος σαν P_{k+1}, προχωράει στον υπολογισμό της νέας ομάδας · σε αντίθετη περίπτωση, στέλνει όλα τα αποθηκευμένα δεδομένα στον νέο P_{k+1}. Και στις δυο περιπτώσεις, ο P_{k+1} έχει όλα τα απαραίτητα δεδομένα και δεν χρειάζεται να περιμένει σε κανένα σημείο συγχρονισμού.

5.2.1 Σύνδεσμοι επικοινωνίας.

Στον αλγόριθμο DMPS η ανεύρεση των αναγκαίων επικοινωνιακών συνδέσμων είναι πολύ πιο απλή από ότι στον αλγόριθμο που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Ο λόγος είναι ότι η ο αλγόριθμος δρομολόγησης εφαρμόζεται σε μια μόνο διάσταση, στην διάσταση δρομολόγησης. Αυτό εξασφαλίζει ότι ο κάθε εργάτης για να ολοκληρώσει τον υπολογισμό μιας ομάδας επαναλήψεων θα πρέπει να επικοινωνήσει με δυο μόνο εργάτες, τον εργάτη στον οποίο έχει αναταθεί η προηγούμενη ομάδα, από τον οποίο θα λάβει δεδομένα, και με τον εργάτη που θα αναλάβει την επόμενη ομάδα, στον οποίο θα στείλει δεδομένα. Αν λάβουμε υπόψη μας το γεγονός ότι ο συντονιστής μπορεί να αναθέσει μόνο σε ένα εργάτη μια ομάδα σε κάθε βήμα δρομολόγησης καταλήγουμε στο παρακάτω συμπέρασμα: Αρκεί ο συντονιστής να κρατάει το όνομα του εργάτη που έχει αναλάβει την νέα ομάδα επαναλήψεων και το όνομα του εργάτη που είχε αναλάβει την προηγούμενη. Ο συντονιστής πάντα ονομάζει '*τρέχοντα (current)*' τον εργάτη στον οποίο έχει αναθέσει την τελευταία ομάδα επαναλήψεων, δηλαδή το C_i ενώ '*προηγούμενο (previous)*' τον εργάτη στον οποίο έχει αναθέσει την προηγούμενη ομάδα (C_{i-1}). Όταν ένας εργάτης αναλάβει μια νέα ομάδα επαναλήψεων τότε αυτός ονομάζεται 'νέος *τρέχον*', ενώ ο προηγούμενος 'τρέχοντας' τώρα καλείται 'νέος *προηγούμενος*'. Ο προηγούμενος εργάτης δεν χρειάζεται πια. Στο σχήμα 5.3 δίνεται το διάγραμμα καταστάσεως στο οποίο το έναυσμα για τις αλλαγές κατάστασης δίνεται από τις αιτήσεις των εργατών στον συντονιστή για καινούργια ομάδα επαναλήψεων. Όταν ένας εργάτης αναλαμβάνει τον υπολογισμό μίας ομάδας, ο συντονιστής μπορεί να τον ενημερώσει ταυτόχρονα για τον εργάτη στον οποίο είχε αναθέσει τη προηγούμενη ομάδα επαναλήψεων. Ωστόσο σε αυτό το βήμα δρομολόγησης δεν είναι γνωστή η ταυτότητα του εργάτη που θα αναλάβει την επόμενη ομάδα. Όταν ο συντονιστής αναθέσει μια νέα ομάδα θα ενημερώσει τον 'νέο προηγούμενο ' εργάτη, με την μορφή



Σχήμα 5.3: Διάγραμμα καταστάσεων εργατών από την πλευρά του συντονιστή.

μιας ασύγχρονης επικοινωνίας, για την ταυτότητα του 'νέου τρέχοντος εργάτη '.

Ο αλγόριθμος $DMPS(x)$ περιγράφεται στον ακόλουθο ψευδοκώδικα:

Είσοδος (α) Ένας φωλιασμένος βρόχος n-διαστάσεων με ομοιόμορφες εξαρτήσεις.

(β) Ένας από τους αλγόριθμους δρομολόγησηςCSS, TSS ή DTSS.

(γ) Αν έχει επιλεγεί ο CSS, τότε το μέγεθος της ομάδας C_i.

(δ) Το διάστημα συγχρονισμού H.

(ε) Ο αριθμός των εργατών m· και στην περίπτωση του DTSS την εικονική δύναμη του κάθε εργάτη.

Πλευρά Συντονιστή:

Αρχικοποίηση:
(α) Εγγραφή εργατών. Στην περίπτωση του DTSS, οι εργάτες αναφέρουν την διαθέσιμη υπολογιστική τους ισχύ ACP.

(β) Για τους TSS και DTSS υπολόγισε τα F, L, N, D. Για τον CSS χρησιμοποίησε το C_i που δόθηκε σαν είσοδος.

Όσο υπάρχουν διαθέσιμες επαναλήψεις:
(α) Αν έρθει καινούργια αίτηση από ένα εργάτη, τοποθέτησε την σε μια λίστα.

(β) Πάρε τη πρώτη αίτηση από την λίστα και υπολόγισε το μέγεθος της επόμενης ομάδας χρησιμοποιώντας τον αλγόριθμο δρομολόγησης που έχει επιλεγεί.

(γ) Ανανέωσε τις ταυτότητες (ranks) του τρέχοντος και του προηγούμενου εργάτη.

(δ) Στείλε την ταυτότητα του τρέχοντος εργάτη στον προηγούμενο εργάτη.

Πλευρά εργάτη P_k:

Αρχικοποίηση:
(α) Εγγραφή με τον συντονιστή. Στη περίπτωση του DTSS ανάφερε την ACP_k.

(β) Υπολόγισε τον αριθμό των M από το δοσμένο H.

1. Στείλε αίτηση για δουλειά στον συντονιστή.
2. Περίμενε για απάντηση· Αν στην απάντηση περιέχεται κάποια ομάδα επαναλήψεων.
3. Υπολόγισε την ομάδα i μέχρι να φτάσεις στο επόμενο σημείο συγχρονισμού.
4. Αν γνωρίζεις την ταυτότητα του εργάτη *send-to* πήγαινε στο βήμα 5
   αλλιώς πήγαινε στο βήμα 6.
5. Στείλε τα απαραίτητα δεδομένα στον εργάτη *send-to*.
6. Λάβε τα απαραίτητα δεδομένα από τον εργάτη *receive-from* και πήγαινε στο βήμα 3.

Έξοδος Συντονιστής: Αν δεν υπάρχουν άλλες ομάδες τότε ειδοποίησε όλους τους εργάτες και τερμάτισε.

Εργάτης P_k: Αν ο συντονιστής ειδοποιήσει ότι δεν υπάρχουν άλλες επαναλήψεις προς υπολογισμό τότε τερμάτισε.

5.3 Πειραματική αξιολόγηση

Πειραματική διάταξη. Η υλοποίηση του αλγορίθμου $DMPS$ στηρίζεται στο πλαίσιο κατανεμημένου προγραμματισμού που προσφέρεται από την έκδοση `mpich.1.2.6` της βιβλιοθήκης ανταλλαγής μηνυμάτων (MPI) (Pachecho, 1997), και έγινε σε γλώσσα C με την χρήση της έκδοση 1.2.6 του μεταγλωττιστή `gcc`.

Χρησιμοποιήσαμε ένα ετερογενές κατανεμημένο σύστημα που αποτελείτε από 13 υπολογιστές, με έναν από αυτούς να αναλαμβάνει τον ρόλο του συντονιστή. Πιο συγκεκριμένα χρησιμοποιήσαμε: (α) 4 Intel Pentiums III 1266MHz με 1GB RAM που ονομάζονται zealots, και τα οποία διαθέτουν $VP_k = 1$ (ένα

από αυτά στον ρόλο του συντονιστή)· (β) 4 Intel Pentiums III 800MHz με 256MB RAM τα οποία ονομάζονται twins, με $VP_k = 0.66$. Η εικονική δύναμη του κάθε τύπου υπολογιστή καθορίσθηκε από την αναλογία των χρόνων εκτέλεσης ενός μέτροπρογράμματος σε κάθε τύπο υπολογιστή σε σχέση με τους υπόλοιπους. Το δίκτυο διασύνδεσης ήταν 100Mbit fast Ethernet(switched).

Παρουσιάζουμε δυο περιπτώσεις: την *αφοσιωμένη* και την *μη-αφοσιωμένη*. Στην πρώτη περίπτωση, οι επεξεργαστές είναι αφοσιωμένοι στην εκτέλεση του προγράμματος, χωρίς να είναι επιβαρυμένοι με επιπρόσθετο φορτίο. Στην δεύτερη περίπτωση, εκτελούμε και επιπλέον διεργασίες σε κάποιους από τους εργάτες κατά την έναρξη της εκτέλεσης του προγράμματος. Εκτελούμε τρεις σειρές πειραμάτων για κάθε μια από της δυο περιπτώσεις: (1) $DMPS(CSS)$, (2) $DMPS(TSS)$, και (3) $DMPS(DTSS)$ για τέσσερις πραγματικές εφαρμογές και συγκρίνουμε τα αποτελέσματα. Σε όλες τις περιπτώσεις υπολογίζουμε την επιτάχυνση (speedup) σε σχέση με τον σειριακό κώδικα για m= 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 εργάτες. Η επιτάχυνση (S_p) υπολογίζεται σύμφωνα με τον ακόλουθο τύπο:

$$S_p = \frac{min\{T_{P_1}, T_{P_2}, \ldots, T_{P_m}\}}{T_{PAR}} \tag{5.1}$$

Όπου T_{P_i} είναι ο σειριακός χρόνος εκτέλεσης στον εργάτη P_i, $1 \leq i \leq m$, και T_{PAR} είναι ο παράλληλος χρόνος εκτέλεσης σε m εργάτες. Στα γραφήματα δίνουμε την επιτάχυνση (S_p) σε σχέση με την σχετική υπολογιστική ισχύ (VP) αντί του m στον άξονα x. Για παράδειγμα, στο πρώτο γράφημα που δίνεται στο σχήμα 5.5, στο οποίο δίνεται η επιτάχυνση για την εφαρμογή της εξίσωσης διάδοσης της θερμότητας, στην αφοσιωμένη περίπτωση, με την διάταξη $v1$ (πίνακας 5.3), $F = 500$ και $H = 150$. Όταν $m = 9$, $T_{P_i} = T_{P_{twin}} = 20.24s$ (πίνακας 5.3), και $T_{PAR} = 5.023s$, οπότε $S_p = T_{P_{twin}}/T_{PAR} = 4.03$. Όμως, για $m = 10$, $T_{P_i} = T_{P_{zealot}} = 10.93s$, $T_{PAR} = 3.506s$ και $S_p = T_{P_{zealot}}/T_{PAR} = 3.12$.

Τα πειράματα μας πραγματοποιήθηκαν στο ίδιο κατανεμημένο σύστημα χρησιμοποιώντας δυο διαφορετικές διατάξεις (πίνακας 5.3). Το μηχάνημα zealot1 έπαιξε τον ρόλο του συντονιστή και στις δυο διατάξεις. Τα μηχανήματα που αναγράφονται με έντονα γράμματα είναι αυτά που φορτώθηκαν στην μη-αφοσιωμένη περίπτωση. Όπως έχει αναφερθεί νωρίτερα η διαθέσιμη ισχύς των μηχανημάτων αυτών υποδιπλασιάστηκε με την εκτέλεση μια επιπλέον διεργασίας. Για $m = 2$ χρησιμοποιήθηκαν τα δυο πρώτα μηχανήματα από κάθε διάταξη, για $m = 3$ τα τρία πρώτα κτλ.

Οι χρόνοι σειριακής εκτέλεσης για κάθε παράδειγμα εφαρμογής και για κάθε τύπο μηχανήματος δίνονται στον πίνακα 5.3. Οι χρόνοι αυτοί χρησιμοποιούνται

Πίνακας 5.2: Οι δυο διατάξεις του συστήματος και η συνολική VP ανάλογα με τον αριθμό εργατών

Διάταξη	Συντονιστής	Εργάτες
$v1$	zealot1	**kid1**, kid2, **kid3**, kid4, **kid5**,twin1, **twin2**, twin3, **twin4**, zealot2,**zealot3**, zealot4
		VP: 0.66, 0.99, 1.32, 1.65, 2.31, 2.97, 3.63, 4.29, 5.29, 6.29, 7.29
$v2$	zealot1	zealot2, twin1, **kid1**, zealot3, twin2,**kid2**, zealot4, twin3, **kid3**, twin4,**kid4**, **kid5**
		VP: 1.66, 1.99, 2.99, 3.65, 3.98, 4.98, 5.64, 5.97, 6.63, 6.96, 7.29

για τον υπολογισμό της επιτάχυνσης σύμφωνα με τον τύπο (5.1).

Πίνακας 5.3: Χρόνοι σειριακής εκτέλεσης (σε secs) για κάθε παράδειγμα εφαρμογής, για τον δεδομένο χώρο επαναλήψεων, σε κάθε τύπο μηχανήματος.

Τύπος	Heat equation $(10k \times 20k)$	Floyd-Steinberg $(5k \times 10k)$	ADI $(5k \times 10k \times 3)$	Hydro $(5k \times 10k \times 6)$
zealots	10.93	11.93	13.46	24.24
twins	20.24	25.36	24.85	43.12
kids	34.18	29.62	39.77	64.60

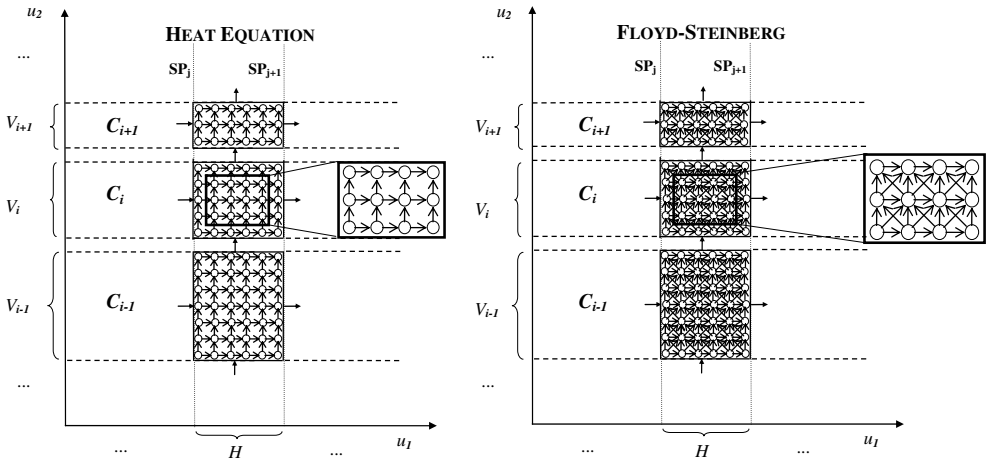
5.3.1 Παραδείγματα εφαρμογών.

Εξίσωση διάδοσης της θερμότητας. Ο υπολογισμός της εξίσωσης διάδοσης της θερμότητας είναι ένα από τα πιο γνωστά παραδείγματα εφαρμογών στην βιβλιογραφία, που περιέχουν φωλιασμένους βρόχους. Το σώμα του βρόχου είναι παρόμοιο στην πλειονότητα επίλυσης μερικών διαφορικών εξισώσεων με την χρήση επαναληπτικών μεθόδων. περιπτώσεων που χρησιμοποιούν αριθμητικές μεθόδους για την επίλυση μερικών διαφορικών εξισώσεων. Υπολογίζει την τιμή της θερμότητας σε κάθε σημείο ενός πλέγματος βάση δυο γειτονικών τιμών του τρέχοντος χρονικού βήματος `(A[i-1][j], A[i][j-1])` και δυο τιμών του προηγούμενου χρονικού βήματος `(A'[i+1][j], A'[i][j+1])`, για ένα αριθμό χρονικών βημάτων. Τα διανύσματα εξάρτησης είναι για το ίδιο χρονικό βήμα: $\vec{\mathbf{d}}_1 = (1, 0)$ και $\vec{\mathbf{d}}_2 = (0, 1)$. Ο υπολογισμός γίνεται σύμφωνα με τον παρακάτω ψευδοκώδικα:

```
/* Heat equation */
```

```
for (l=1; l<loop; l++) {
    for (i=1; i<width; i++){
        for (j=1; j<height; j++){
                A[i][j] = 1/4*(A[i-1][j] + A[i][j-1]
                       + A'[i+1][j] + A'[i][j+1]);
}}}
```

Στο σχήμα 5.4 δίνεται μια αναπαράσταση της μορφής των διανυσμάτων εξάρτησης. Η εκτέλεση των επαναλήψεων γίνεται έτσι ώστε να ικανοποιηθούν τα διανύσματα εξάρτησης.



Σχήμα 5.4: Διανύσματα εξάρτησης για την εξίσωση διάδοσης της θερμότητας (αριστερά) και Floyd-Steinberg (δεξιά).

Υπολογισμός Floyd-Steinberg. Ο υπολογισμός Floyd-Steinberg (Floyd & Steinberg, 1976)[79] χρησιμοποιείται σε εφαρμογές επεξεργασίας εικόνας (error-diffusion dithering). Οι εξαρτήσεις που περιέχει είναι: $\vec{\mathbf{d}}_1 = (1,0)$, $\vec{\mathbf{d}}_2 = (1,1)$, $\vec{\mathbf{d}}_3 = (0,1)$ και $\vec{\mathbf{d}}_4 = (1,-1)$. Ο ψευδοκώδικας του Floyd-Steinberg δίνεται ως:

```
/* Floyd-Steinberg */
for (i=1; i<width; i++){
    for (j=1; j<height; j++){
            I[i][j] = trunc(J[i][j]) + 0.5;
            err = J[i][j] - I[i][j]*255;
            J[i-1][j] += err*(7/16);
            J[i-1][j-1] += err*(3/16);
```

```
            J[i][j-1] += err*(5/16);
            J[i-1][j+1] += err*(1/16);
}}
```

Στο σχήμα 5.4 δίνεται μια αναπαράσταση της μορφής των διανυσμάτων εξάρτησης.

LL kernel 8 - ADI. Εδώ έχουμε ένα πυρήνα από τα μετροπρογράμματα Livermoore loops (LL-K8)[67]. Ο αλγόριθμος ADI (Alternating Direction Implicit integration) χρησιμοποιείται για την επίλυση μερικών διαφορικών εξισώσεων (McMahon, 1986), (Karniadakis & Kirby, 2002). Ο αλγόριθμος ADI είναι ένας τέλεια φωλιασμένος βρόχος τριών διαστάσεων με μοναδιαίες εξαρτήσεις που διασχίζουν και τις τρεις διαστάσεις του χώρου δεικτών. Ο ψευδοκώδικας του ADI δίνεται παρακάτω:

```
/* ADI integration */
for (l=1; l<=loop; l++) {
    nl1 = 0;
    nl2 = 1;
    for (kx=1; kx<3; kx++){
        for (ky=1; ky<n; ky++) {
           du1[ky] = u1[nl1][ky+1][kx] - u1[nl1][ky-1][kx];
           du2[ky] = u2[nl1][ky+1][kx] - u2[nl1][ky-1][kx];
           du3[ky] = u3[nl1][ky+1][kx] - u3[nl1][ky-1][kx];
           u1[nl2][ky][kx]=
             u1[nl1][ky][kx]+a11*du1[ky]+a12*du2[ky]+a13*du3[ky] + sig*
             (u1[nl1][ky][kx+1]-2.0*u1[nl1][ky][kx]+u1[nl1][ky][kx-1]);
             u2[nl2][ky][kx]=
             u2[nl1][ky][kx]+a21*du1[ky]+a22*du2[ky]+a23*du3[ky] + sig*
             (u2[nl1][ky][kx+1]-2.0*u2[nl1][ky][kx]+u2[nl1][ky][kx-1]);
             u3[nl2][ky][kx]=
             u3[nl1][ky][kx]+a31*du1[ky]+a32*du2[ky]+a33*du3[ky] + sig*
             (u3[nl1][ky][kx+1]-2.0*u3[nl1][ky][kx]+u3[nl1][ky][kx-1]);
}}}
```

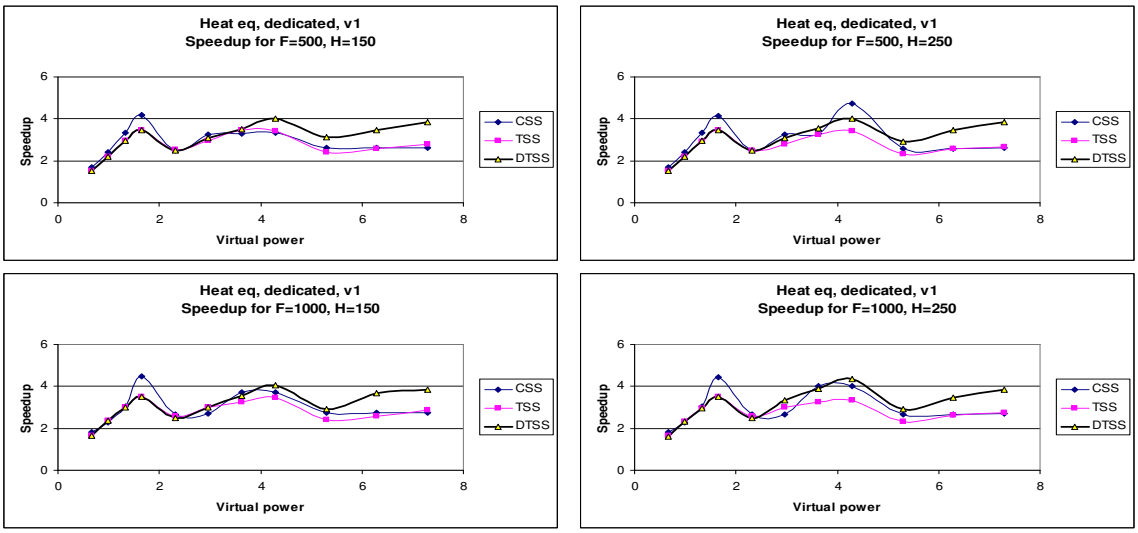
LL kernel 23 - Hydro. Ο τελευταίος αλγόριθμος που εξετάζουμε προέρχεται επίσης από τα μετροπρογράμματα Livermoore loops. Είναι ο πυρήνας LLK23 Hydro (Implicit Hydrodynamics fragment) (McMahon, 1986) που είναι πάλι ένας φωλιασμένος βρόχος τριών διαστάσεων με μοναδιαία διανύσματα εξάρτησης. Ο ψευδοκώδικας του δίνεται παρακάτω:

```
/* 2-D implicit hydrodynamics fragment */
for (l=1; l<=loop; l++) {
```

```
    for (j=1; j<6; j++) {
        for (k=1; k<n; k++) {
            qa = za[j+1][k]*zr[j][k] + za[j-1][k]*zb[j][k] +
                 za[j][k+1]*zu[j][k] + za[j][k-1]*zv[j][k] + zz[j][k];
            za[j][k] += 0.175*( qa - za[j][k] );
}}}
```

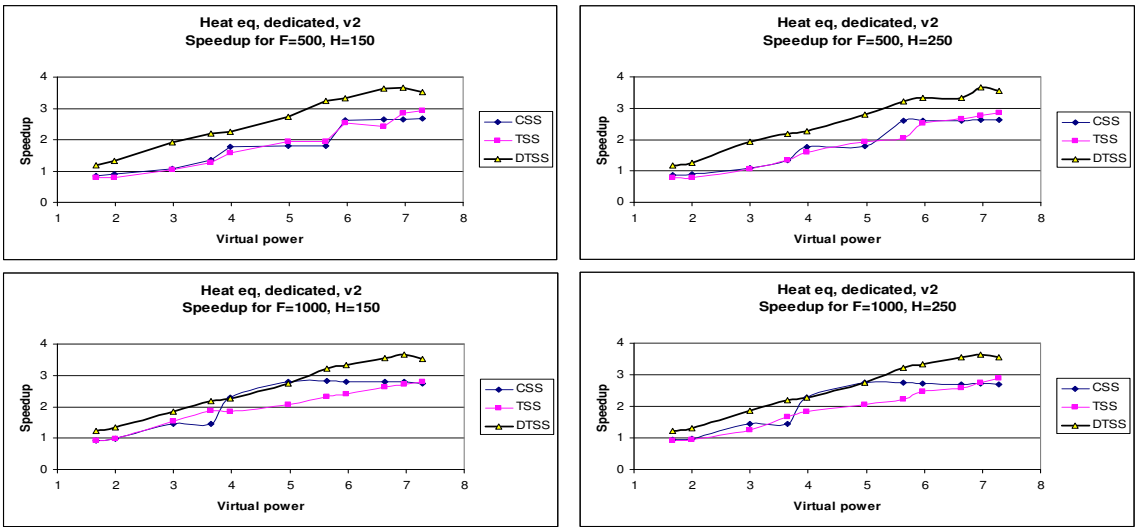
5.3.2 Αποτελέσματα.

Εξίσωση διάδοσης της θερμότητας. Στις γραφικές παραστάσεις στα σχήματα 5.5–5.8 δίνεται η επιτάχυνση που επιτεύχθηκε για την εφαρμογή της εξίσωσης διάδοσης της θερμότητας σε ένα χώρο 10000×20000 σημείων και για ένα χρονικό βήμα (loop=1), με την χρήση των τριών αλγορίθμων: $DMPS(CSS)$, $DMPS(TSS)$ και $DMPS(DTSS)$. Χρησιμοποιήθηκαν και οι δυο διατάξεις σε αφοσιωμένο και μη-αφοσιωμένο περιβάλλον. Οι τιμές του F που δόθηκαν είναι 500, 750 και 1000. Για να καθορίσουμε την επίδραση του δοσμένου μεγέθους ομάδας επαναλήψεων στην περίπτωση του $DMPS(CSS)$, χρησιμοποιήσαμε δυο τιμές: η μια είναι η τιμή του F που δίνεται από τις δυο αντίστοιχες άλλες περιπτώσεις, και η άλλη είναι η τιμή που υπολογίζεται από τον $DMPS(DTSS)$ για το πιο αργό τύπο μηχανήματος της συστοιχίας. Επίσης επιλέχθηκαν δυο διαστήματα συγχρονισμού, 150 και 250. Η συνολική VP κυμαίνεται ανάλογα με τον αριθμό των εργατών από 0.66 μέχρι 7.29 για την $v1$ και από 1.66 μέχρι 7.29 για την $v2$.

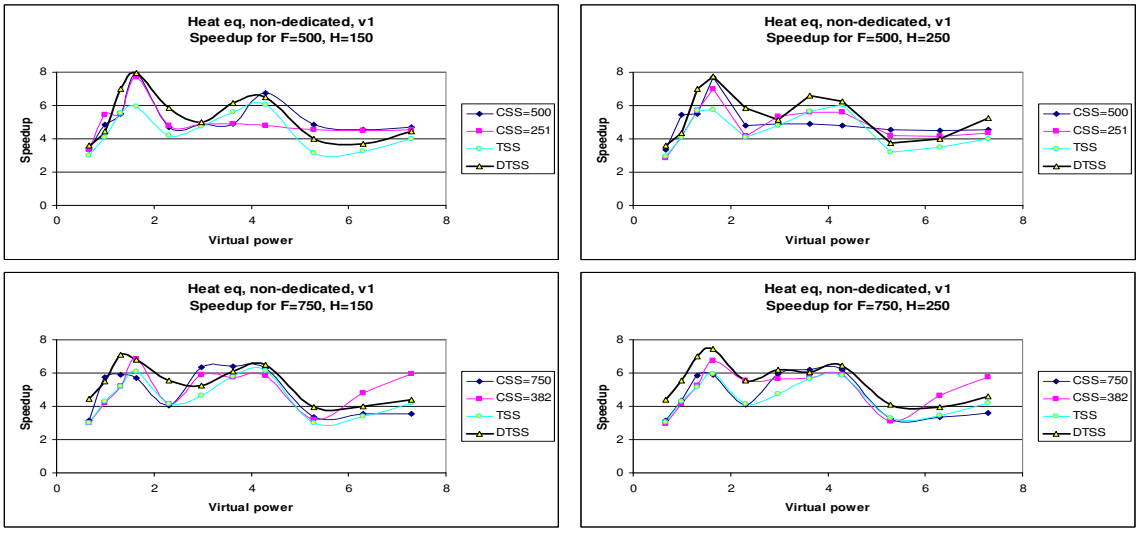


Σχήμα 5.5: Επιτάχυνση της εξίσωση διάδοσης της θερμότητας, αφοσιωμένο σύστημα, $v1$

Υπολογισμός Floyd-Steinberg. Στις γραφικές παραστάσεις στα σχήματα 5.9–5.12 δίνεται η επιτάχυνση που επιτεύχθηκε για τον αλγόριθμο Floyd-Steinberg



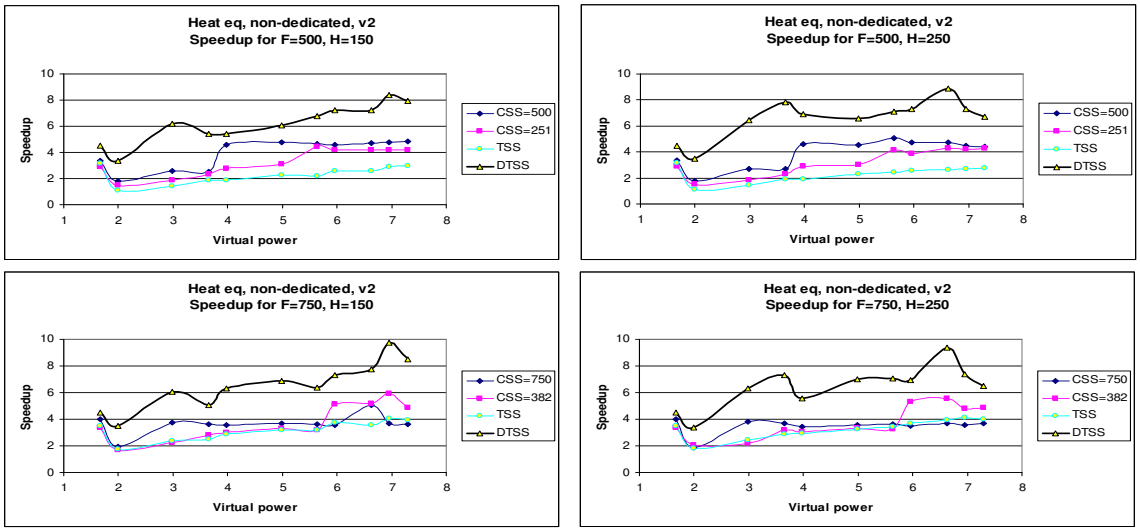
Σχήμα 5.6: Επιτάχυνση της εξίσωση διάδοσης της θερμότητας, αφοσιωμένο σύστημα , $v2$



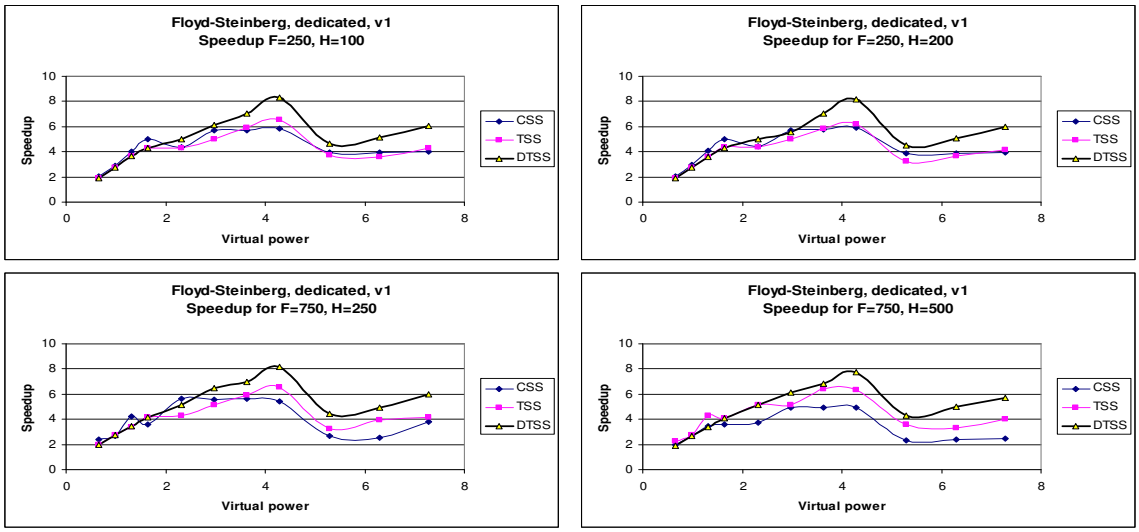
Σχήμα 5.7: Επιτάχυνση της εξίσωση διάδοσης της θερμότητας, μη-αφοσιωμένο σύστημα , $v1$

σε ένα χώρο επαναλήψεων 5000×10000 σημείων με την χρήση των τριών αλγορίθμων δρομολόγησης. Οι τιμές του F που δόθηκαν είναι 250 και 750.

LL kernel 8 - ADI. Στις γραφικές παραστάσεις στα σχήματα 5.13–5.16 δίνεται η επιτάχυνση που επιτεύχθηκε για τον αλγόριθμο ADI σε ένα χώρο επαναλήψεων $5000 \times 3 \times 10000$ σημείων με την χρήση των τριών αλγορίθμων δρομολόγησης. Οι



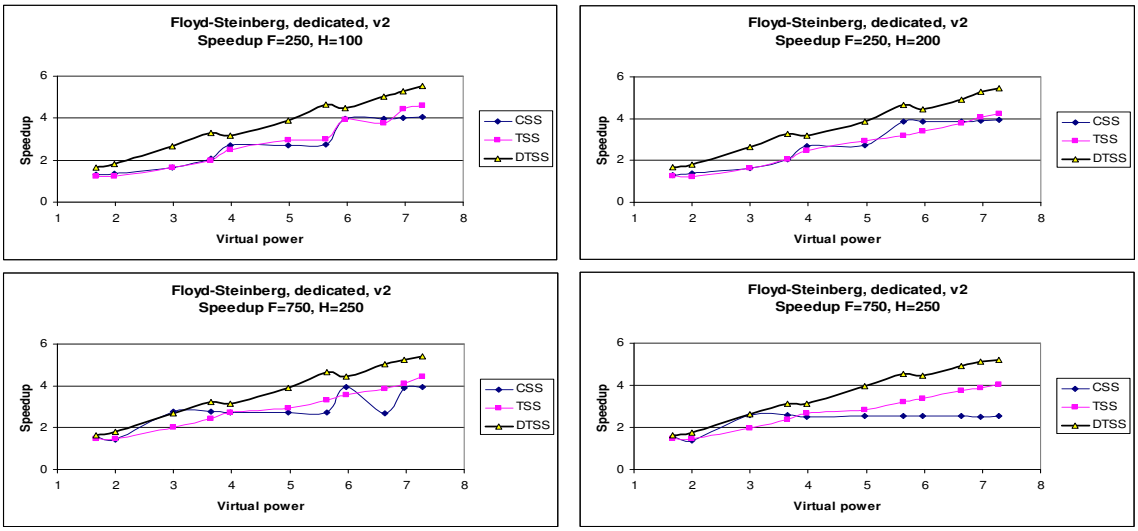
Σχήμα 5.8: Επιτάχυνση της εξίσωση διάδοσης της θερμότητας, μη-αφοσιωμένο σύστημα , $v2$



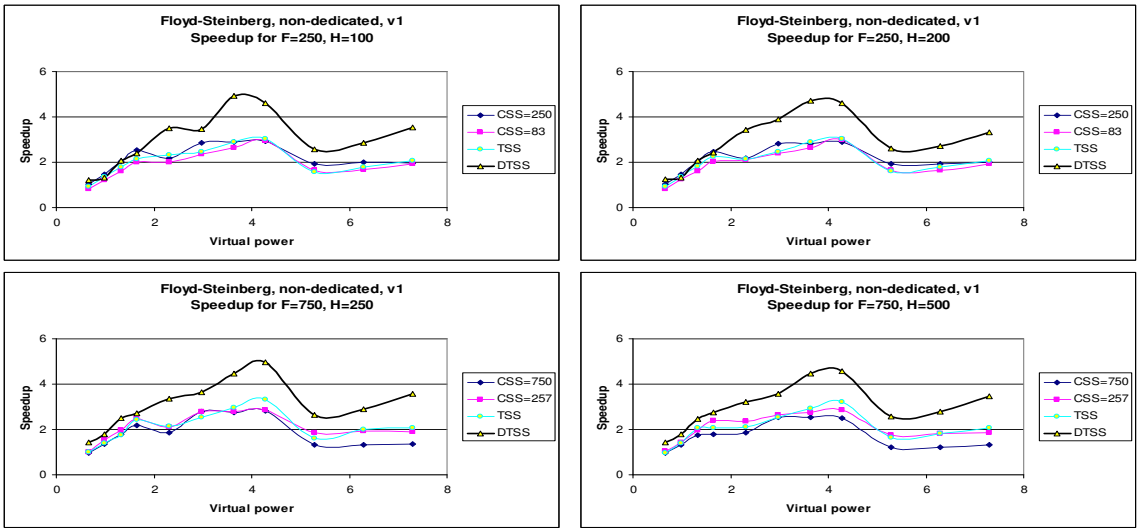
Σχήμα 5.9: Επιτάχυνση του αλγόριθμου Floyd-Steinberg, αφοσιωμένο σύστημα , $v1$

τιμές του F που δόθηκαν είναι 500 και 750.

LL kernel 23 - Hydro. Στις γραφικές παραστάσεις στα σχήματα 5.13–5.16 δίνεται η επιτάχυνση που επιτεύχθηκε για τον αλγόριθμο Hydro σε ένα χώρο επαναλήψεων $5000 \times 6 \times 10000$ σημείων. Οι τιμές του F που δόθηκαν είναι 500 και 750.

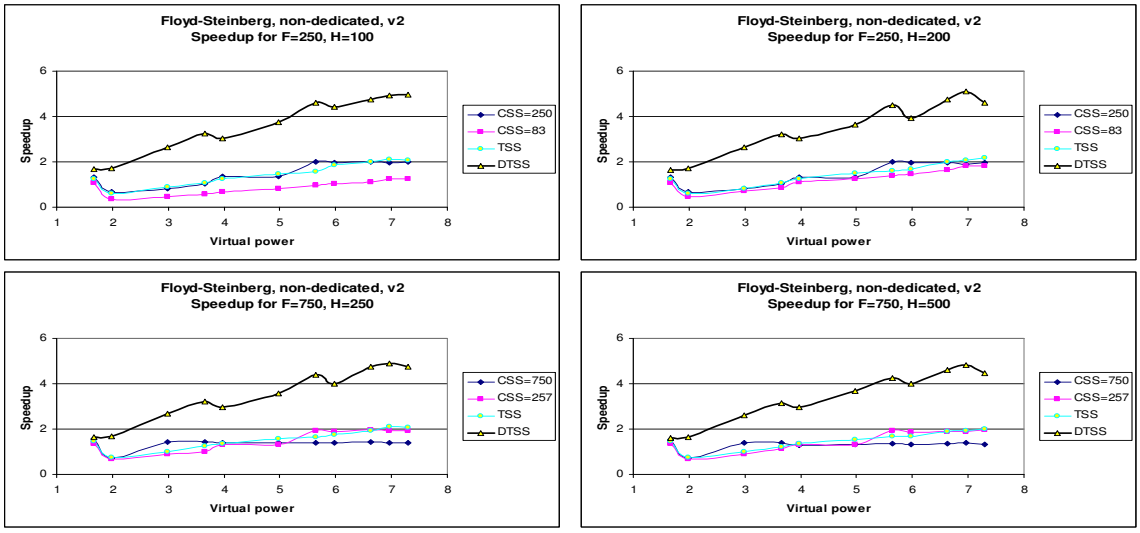


Σχήμα 5.10: Επιτάχυνση του αλγόριθμου Floyd-Steinberg, αφοσιωμένο σύστημα , $v2$

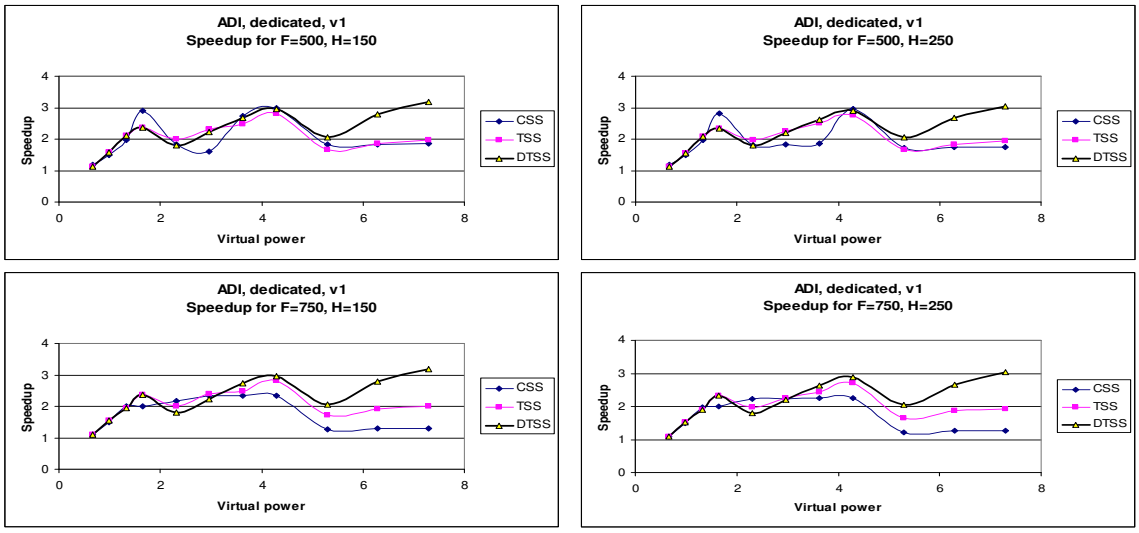


Σχήμα 5.11: Επιτάχυνση του αλγόριθμου Floyd-Steinberg, μη-αφοσιωμένο σύστημα , $v1$

Ανάλυση των αποτελεσμάτων. Δυο κατηγορίες παραγόντων επηρεάζουν την απόδοση του $DMPS(x)$: (1) εσωτερικοί παράγοντες, οι οποίες είναι για τον CSS η επιλογή του C_i και του H και για τους TSS και DTSS η επιλογή του F και H, και (2) εξωτερικοί παράγοντες, που επηρεάζουν όλους τους αλγόριθμους δρομολόγησης και είναι η σύσταση και διάταξη και το εξωτερικό φόρτο εργασίας της συστοιχίας. Σχετικά με τους εσωτερικούς παράγοντες, μπορεί να μετριαστεί η απόδοση εξαιτίας



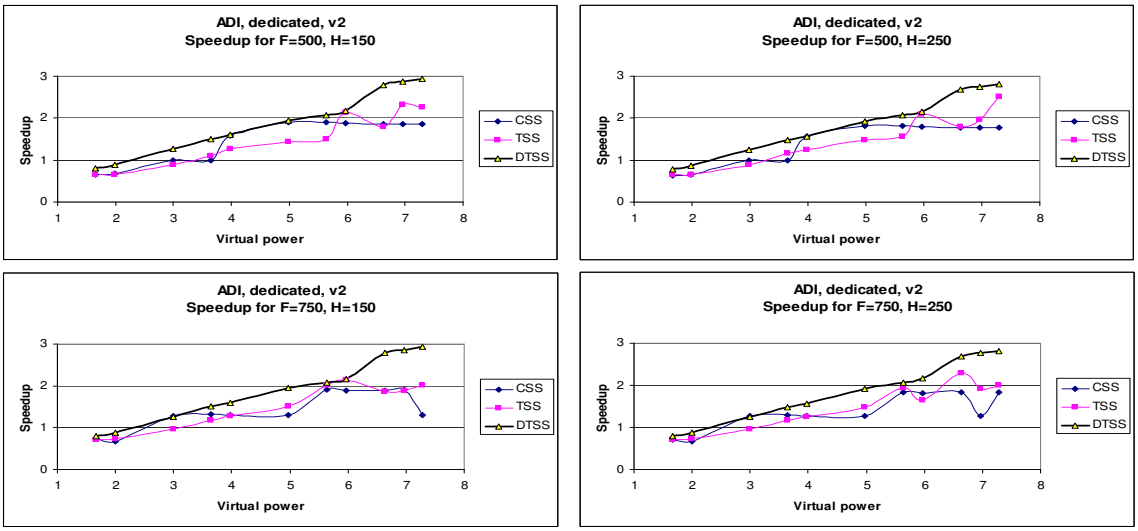
Σχήμα 5.12: Επιτάχυνση του αλγόριθμου Floyd-Steinberg, μη-αφοσιωμένο σύστημα , $v2$



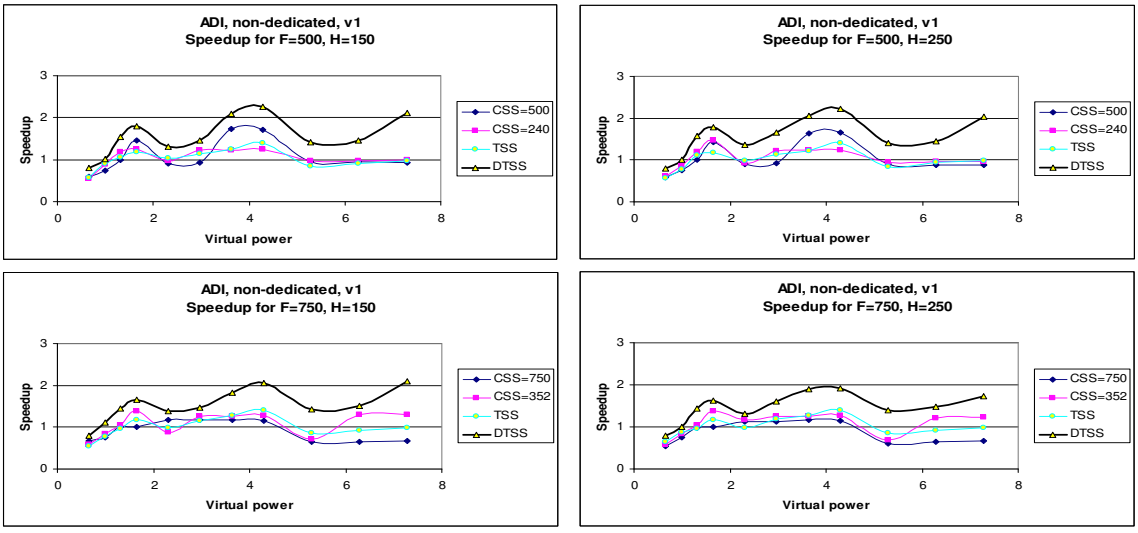
Σχήμα 5.13: Επιτάχυνση του αλγόριθμου ADI, αφοσιωμένο σύστημα , $v1$

αστοχίας στην επιλογή των κατάλληλων τιμών, ωστόσο είναι στην κρίση του χρήστη να επιλέξει λογικές τιμές έτσι ώστε να διατηρήσει την απόδοση μέσα σε αποδεκτά επίπεδα. Από την άλλη μεριά, σχετικά με τους εξωτερικούς παράγοντες, η απόδοση μπορεί εύκολα να υποβιβαστεί σημαντικά για διαφορετικές διατάξεις και ανάλογα με την διακύμανση του εξωτερικού φόρτου της συστοιχίας.

Η αναλογία χρόνου υπολογισμών προς τον χρόνο επικοινωνιών παίζει πολύ σημαντικό ρόλο στην απόδοση γενικά των κατανεμημένων συστημάτων και κατέπεκταση

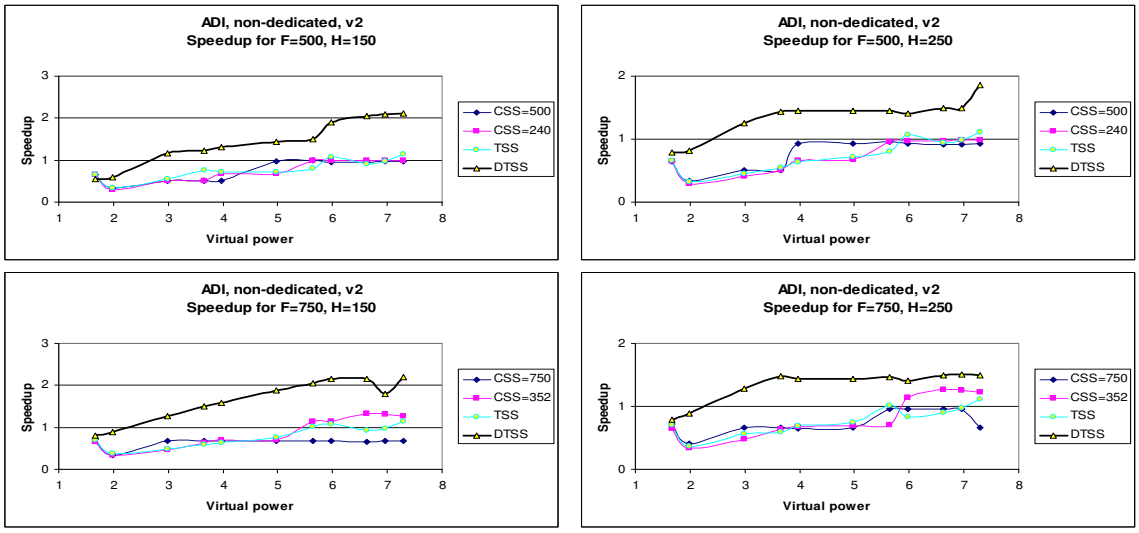


Σχήμα 5.14: Επιτάχυνση του αλγόριθμου ADI, αφοσιωμένο σύστημα , $v2$

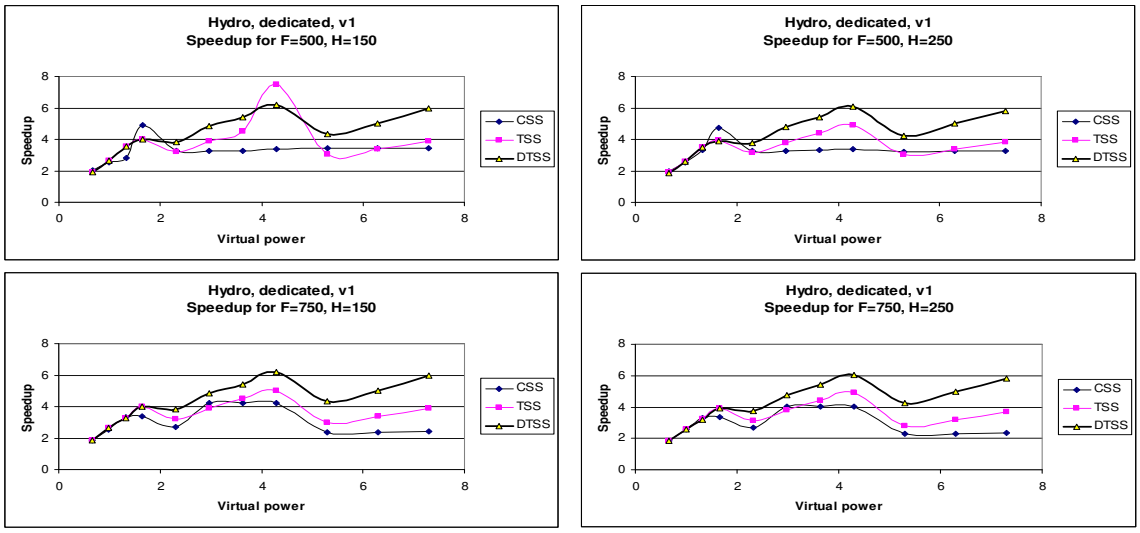


Σχήμα 5.15: Επιτάχυνση του αλγόριθμου ADI, μη-αφοσιωμένο σύστημα , $v1$

και στον αλγόριθμο που παρουσιάζεται σε αυτό το κεφάλαιο. Σε κάθε περίπτωση ο παραπάνω λόγος πρέπει να διατηρείται μεγαλύτερος από το ένα. Ο χρόνος που απαιτείται για τον υπολογισμό του $SC_{i,j}$ είναι ανάλογος του $H \times V_i \times \prod_{i \neq 1,2} u_i$. Ο χρόνος που απαιτείται για την ανταλλαγή των δεδομένων του $SC_{i,j}$ είναι ανάλογος του όγκου των σημείων που ανταλλάσσονται, που είναι ανάλογος του $H \times \prod_{i \neq 1,2} u_i$ και κάποιας σταθερής καθυστέρησης που είναι ένα χαρακτηριστικό του δικτύου αλλά και του πρωτοκόλλου που χρησιμοποιείται. Η επιλογή του κατάλληλου διαστήματος συγχρονισμού επηρεάζει αρκετά την απόδοση του αλγορίθμου. Το H πρέπει να επι-



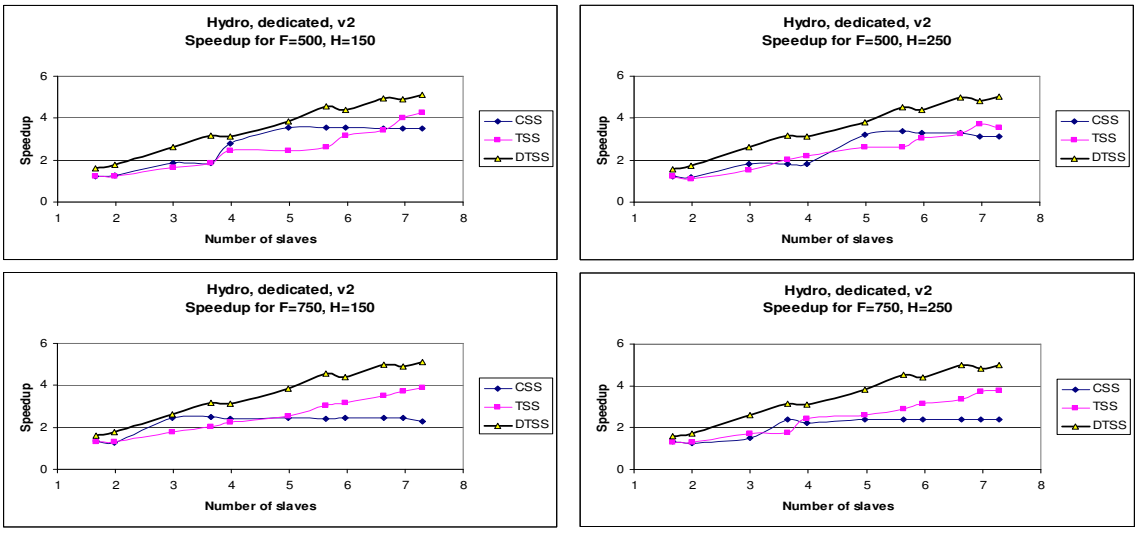
Σχήμα 5.16: Επιτάχυνση του αλγόριθμου ADI, μη-αφοσιωμένο σύστημα , $v2$



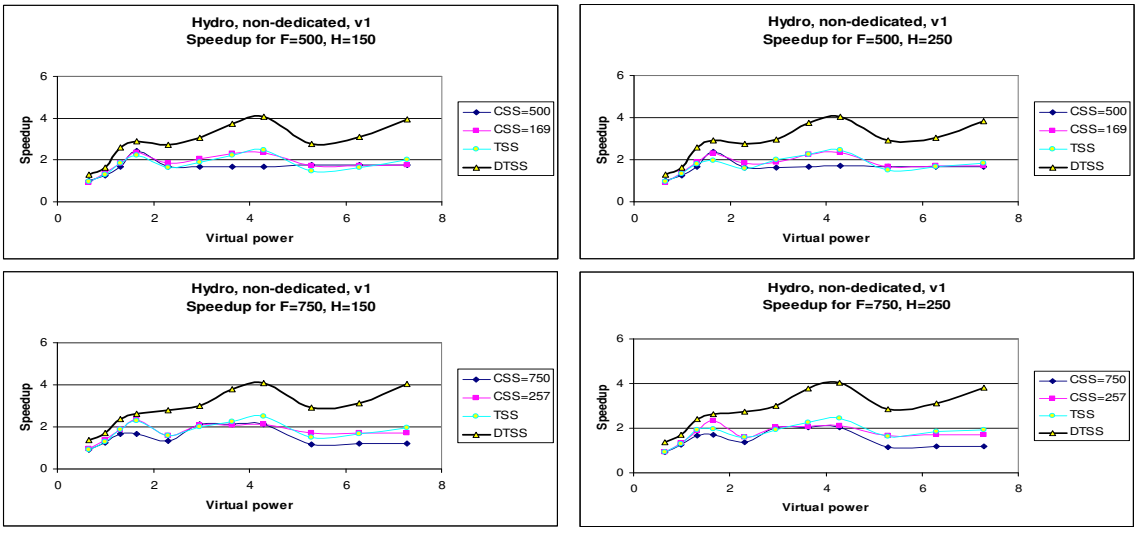
Σχήμα 5.17: Επιτάχυνση του αλγόριθμου Hydro, αφοσιωμένο σύστημα , $v1$

λέγεται ανάλογα με την εκάστοτε εφαρμογή και την πλατφόρμα εκτέλεσης, έτσι ώστε να εξασφαλίζει ότι ο λόγος $\frac{computation}{communication}$ διατηρείται πάνω από 1, ακόμα και όταν το V_i μειώνεται σε κάθε βήμα της δρομολόγησης. Αν υποθέσουμε ότι η τιμή του H εξασφαλίζει ότι $\frac{computation}{communication} \geq 1$, τότε μικρές αλλαγές του H δεν επηρεάζουν σημαντικά την απόδοση όπως μπορούμε να διαπιστώσουμε από τα αποτελέσματα .

Τα αποτελέσματα δείχνουν μια αυξημένη ευαισθησία στην σύσταση και διάταξη της συστοιχίας (τύπους και αριθμό εργατών) καθώς και στον φόρτο των μηχανημάτων. Για αυτόν τον λόγο πειραματιστήκαμε με δυο διατάξεις, $v1$ και $v2$ (πίνακας 5.3)

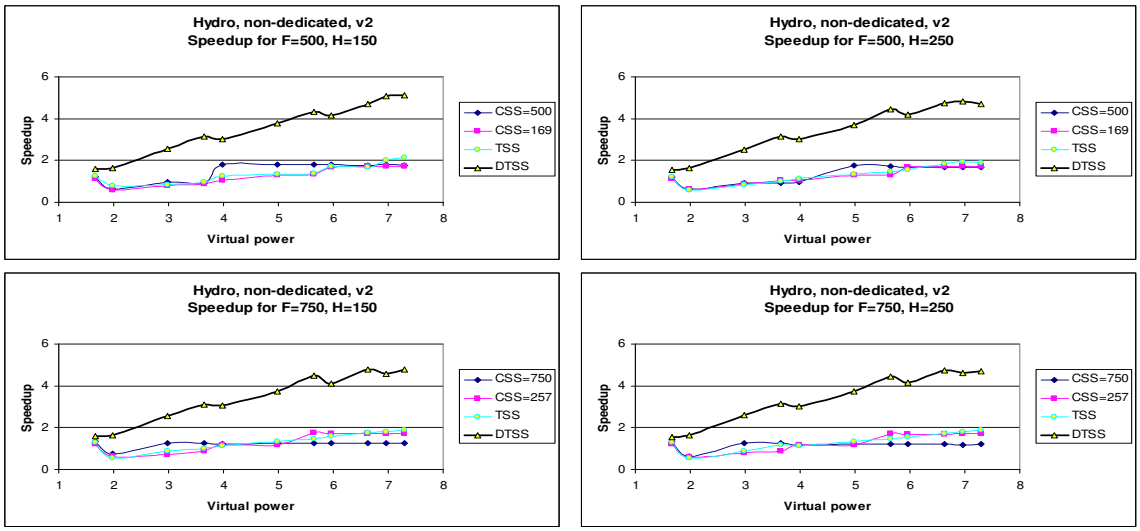


Σχήμα 5.18: Επιτάχυνση του αλγόριθμου Hydro, αφοσιωμένο σύστημα , $v2$



Σχήμα 5.19: Επιτάχυνση του αλγόριθμου Hydro, μη-αφοσιωμένο σύστημα , $v1$

και με διαφορετικά φορτωμένους εργάτες σε κάθε διάταξη. Όπως ήταν αναμενόμενο ο $DMPS(DTSS)$ δείχνει να προσαρμόζεται καλύτερα από τις άλλες δυο επιλογές, στην διακύμανσης του φόρτου εργασίας και στην ανομοιογένεια της συστοιχίας. Αυτός είναι και ο λόγος που παρουσιάζει καλύτερα αποτελέσματα από τις εκδοχές $DMPS(TSS)$ και $DMPS(CSS)$ οι οποίες δεν έχουν κάποια δυνατότητα προσαρμογής στις συνθήκες του περιβάλλοντος. Στην μη-αφοσιωμένη περίπτωση, η επιλογή ήταν να φορτωθούν οι πιο αργοί εργάτες έτσι ώστε να μεγαλώσει η διαφορά στην διαθέσιμη επεξεργαστική ισχύ μεταξύ των εργατών Ακόμα και σε αυτή την περίπτωση



Σχήμα 5.20: Επιτάχυνση του αλγόριθμου Hydro, μη-αφοσιωμένο σύστημα , $v2$

ο $DMPS(DTSS)$ αποδείχθηκε ότι επιτυγχάνει ικανοποιητικά αποτελέσματα. Η διαφορά στην απόδοση μεταξύ των $DMPS(DTSS)$, $DMPS(TSS)$ και $DMPS(CSS)$ μεγαλώνει καθώς το σύστημα γίνεται πιο ετερογενές και πιο φορτωμένο. Ακόμα, ο $DMPS(TSS)$ δεν παρουσιάζει σημαντικά καλύτερη απόδοση από τον $DMPS(CSS)$ στις εφαρμογές που εξετάστηκαν.

Εδώ πρέπει να αναφέρουμε ότι σε όλες τις γραφικές παραστάσεις, η επιτάχυνση υπολογίστηκε σύμφωνα με τον τύπο 5.1, δηλαδή βάση του σειριακού χρόνου του γρηγορότερου μηχανήματος που μετέχει στην συστοιχία. Αυτό σημαίνει ότι για την διάταξη $v1$ (αφοσιωμένο σύστημα ή μη-αφοσιωμένο), μέχρι τους πρώτους πέντε εργάτες, (kid1–kid5), ο σειριακός χρόνος είναι ο $T_{P_{kids}}$, μέχρι τους πρώτους εννιά εργάτες (kid1–twin4), ο σειριακός χρόνος είναι ο $T_{P_{twin}}$ και όταν χρησιμοποιούνται και οι δώδεκα εργάτες, ο σειριακός χρόνος είναι $T_{P_{zealots}}$ (πίνακας 5.3). Αυτό εξηγεί τις διακυμάνσεις στα διαγράμματα που παρουσιάζουν την επιτάχυνση στα σχήματα 5.5–5.7, 5.9–5.11, 5.13–5.15, 5.17–5.19. Πιο συγκεκριμένα, σε κάθε διάγραμμα παρατηρούμε τρεις κορυφές που αντιστοιχούν στο γεγονός ότι η επιτάχυνση υπολογίζεται με τρεις διαφορετικούς σειριακούς χρόνους, κάθε φορά που ένα πιο γρήγορο μηχάνημα εισέρχεται στο σύστημα. Παρατηρούμε επίσης ότι μέχρι τους πέντε πρώτους εργάτες (kid1–kid5), η συστοιχία είναι ουσιαστικά ομοιογενείς και έτσι δεν υπάρχει ουσιαστική διαφορά στην απόδοση των τριών εκδοχών του αλγόριθμου. Σε αυτή την περίπτωση, οι $DMPS(TSS)$ και $DMPS(DTSS)$ έχουν ακριβός την ίδια συμπεριφορά. Από αυτό το σημείο και μετά ,δηλαδή με περισσότερους από πέντε εργάτες, η συστοιχία γίνεται ετερογενείς και η απόδοση του $DMPS(DTSS)$ αρχίζει να διαφοροποιείται από αυτή των $DMPS(TSS)$ και $DMPS(CSS)$, μέχρι που μπαίνουν και οι πιο γρήγοροι εργάτες (zealots), και τότε η απόδοση του είναι σαφώς καλύτερη από αυτή που παρουσιάζουν οι άλλες δυο εκδοχές. Αντίθετα στην διάταξη $v2$, η γραμμή της επιτάχυνσης είναι πολύ πιο ομαλή, επειδή σαν σειριακός χρόνος πάντα λαμβάνεται

ο $T_{P_{zealots}}$.

Σε όλες της περιπτώσεις εφαρμογών, η ανωτερότητα του $DMPS(DTSS)$ είναι πιο εμφανής στην μη-αφοσιωμένη περίπτωση, λόγο της ικανότητας του να προσαρμόζεται στις μεταβολές του φόρτου εργασίας. Όπως ήταν αναμενόμενο, σε απόλυτες τιμές, η επιτάχυνση στην περίπτωση του αφοσιωμένου συστήματος είναι πάντα καλύτερη από ότι στην περίπτωση του μη-αφοσιωμένου συστήματος, εξαιτίας της μεγαλύτερης τιμής της συνολικής ACP.

5.4 Συμπεράσματα

Σε αυτό το κεφάλαιο παρουσιάστηκε ένα νέο δυναμικό σχήμα για την δρομολόγηση φωλιασμένων βρόχων σε ετερογενής συστοιχίες υπολογιστών. Ο αλγόριθμος $DMPS$ βασίζεται στην τάξη των αυτοδρομολογούμενων αλγορίθμων και επεκτείνει την λειτουργία τους σε εφαρμογές που περιέχουν φωλιασμένους βρόχους με εξαρτήσεις. Ο αλγόριθμος αυτός διαφέρει από αυτόν που παρουσιάστηκε στο προηγούμενο κεφάλαιο σε πολλά σημεία, η σημαντικότερη διαφορά είναι ίσως ότι ο $DMPS$ μοιράζει τον χώρο επαναλήψεων κατά την μια μόνο διάσταση, την διάσταση της δρομολόγησης. Η πρακτική αυτή έχει σαν συνέπεια ένα σημαντικά απλούστερο κωδικά. Αν και απλούστερος ο $DMPS$ μπορεί να μοιράσει το υπολογιστικό φορτίο με μεγαλύτερη ακρίβεια αφού οι ομάδες υπολογισμού που αναθέτει ο αλγόριθμος του προηγουμένου κεφαλαίου μπορούν να είναι μόνο ακέραια πολλαπλάσια των 'ατομικών εργασιών' που βρίσκονται στα φύλλα του τετραδικού δέντρου. Ο αλγόριθμος $DMPS$ αξιολογήθηκε πειραματικά σε μια ετερογενή συστοιχία υπολογιστών, σε αφοσιωμένο αλλά και σε μη-αφοσιωμένο σύστημα. Χρησιμοποιήθηκε ένα σύνολο από 4 εφαρμογές και τα αποτελέσματα φανερώνουν σε όλες τις περιπτώσεις την γενικότητα και αποτελεσματικότητα του προτεινόμενου αλγόριθμου.

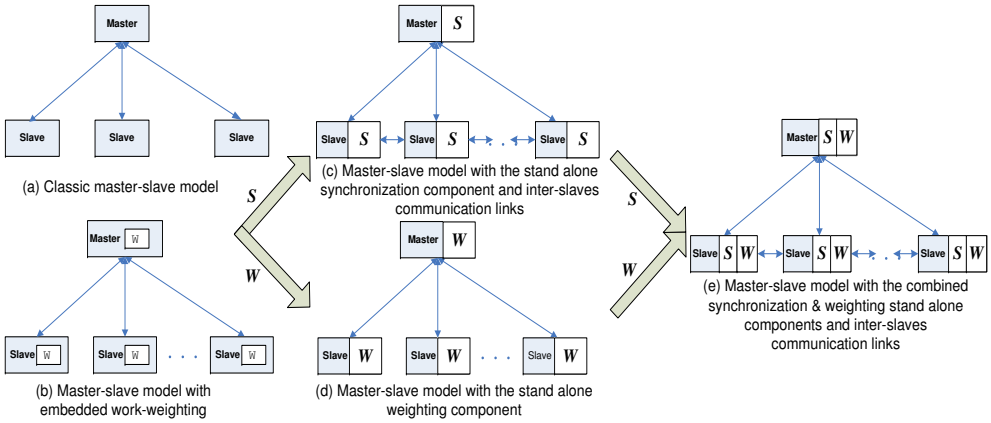
Κεφάλαιο 6

Οι μηχανισμοί συγχρονισμού και στάθμισης

Στο προηγούμενο κεφάλαιο παρουσιάστηκε μια πρώτη προσπάθεια για την επέκταση της κατηγορίας των αυτοδρομολογούμενων αλγορίθμων έτσι ώστε να μπορούν να εφαρμοστούν σε βρόχους με εξαρτήσεις. Προστέθηκαν ρουτίνες επικοινωνιών σε τρεις γνωστούς αλγορίθμους (CSS,TSS,DTSS) δημιουργώντας τον αλγόριθμο $DMPS$, ο οποίος επέδειξε αξιόλογη απόδοση κατά την διάρκεια των δοκιμών. σε αυτό το κεφάλαιο γενικεύουμε την ιδέα του προηγούμενου κεφαλαίου παρουσιάζοντας έναν ανεξάρτητο μηχανισμό επικοινωνιών ο οποίος μπορεί να εφαρμοστεί σε κάθε δυναμικό αλγόριθμο της κατηγορίας των αυτοδρομολογούμενων αλγορίθμων που βασίζεται στο μοντέλο συντονιστή-εργάτη. Επίσης παρουσιάζουμε τον μηχανισμό στάθμισης ο οποίος βελτιώνει την κατανομή υπολογιστικού φορτίου και κατ᾽ επέκταση την συνολική απόδοση των αλγορίθμων αυτών όταν εφαρμόζονται σε ετερογενή υπολογιστικά συστήματα. Με τους δυο αυτούς μηχανισμούς ενισχύεται σημαντικά η χρηστικότητα των αυτοδρομολογούμενων αλγορίθμων αφού μεγαλώνει το πεδίο εφαρμογών τους και ταυτόχρονα βελτιώνεται η απόδοση τους.

6.1 Εισαγωγή

Ο μηχανισμός συγχρονισμού $\mathcal{S}$ είναι εμπνευσμένος από την εργασία ([31]) που παρουσιάστηκε στη προηγούμενο κεφάλαιο. Μπορεί να εφαρμοστεί σε όλες τους αυτοδρομολογούμενους αλγορίθμους, επιτρέποντας του να χειριστεί βρόχους με εξαρτήσεις. Εισάγει σημεία συγχρονισμού στα οποία μπορούν να γίνουν ανταλλαγές δεδομένων μεταξύ των εργατών. Ο μηχανισμός αυτός δεν είναι ενσωματωμένος στον αλγόριθμο δρομολόγησης αλλά υλοποιείται σαν ένα επιπρόσθετο εξάρτημα που μπορεί να εφαρμοστεί χωρίς περαιτέρω τροποποιήσεις. Επιτρέποντας την εφαρμογή των αυτοδρομολογούμενων αλγορίθμων σε βρόχους που περιέχουν εξαρτήσεις, διευρύνουμε



Σχήμα 6.1: *Το μοντέλο συντονιστή-εργάτη με τους μηχανισμούς συγχρονισμού και στάθμισης.*

σημαντικά το πεδίο εφαρμογών τους. Δοθέντος του αρχικού αυτοδρομολογούμενου αλγόριθμου $\mathcal{A}$, η εκδοχή του που περιλαμβάνει τον μηχανισμό συγχρονισμού καλείται $\mathcal{S}$-$\mathcal{A}$ (συγχρονισμένη έκδοση του αλγορίθμου).

Επίσης, ορίζουμε τον μηχανισμό στάθμισης $\mathcal{W}$, ο οποίος στοχεύει στην βελτίωση της ικανότητας κατανομής φορτίου, των αυτοδρομολογούμενων αλγορίθμων που δεν είχαν δημιουργηθεί για μη-αφοσιωμένα, ετερογενή συστήματα. Ο μηχανισμός αυτός είναι εμπνευσμένος από την προσέγγιση που ακολουθείται από τον αλγόριθμο $DTSS$ ([4]). Ο $DTSS$ οποίος χρησιμοποιεί την σχετική υπολογιστική ισχύ των εργατών, συνδυασμένη με πληροφορίες που αφορούν την ουρά εκτέλεσης για να σταθμίσει το μέγεθος των ομάδων υπολογισμού. Και αυτός ο μηχανισμός αποτελεί αυτόνομο εξάρτημα που μπορεί να εφαρμοστεί σε κάθε αυτοδρομολογούμενο αλγόριθμο. Δοθέντος του αρχικού αυτοδρομολογούμενου αλγόριθμου $\mathcal{A}$, η εκδοχή του που περιλαμβάνει τον μηχανισμό στάθμισης καλείται $\mathcal{W}$-$\mathcal{A}$ (σταθμισμένη εκδοχή του αλγόριθμου).

Οι δύο μηχανισμοί μπορούν να συνδυαστούν και να εφαρμοστούν σε ένα βήμα σε κάθε αυτοδρομολογούμενη μεθοδολογία, όπως φαίνεται στο Σχήμα 6.1. Ξεκινώντας από το κλασσικό μοντέλο συντονιστή εργάτη (Σχ. 6.1(a)), προσθέτουμε κανάλια επικοινωνίας μέσω του μηχανισμού $\mathcal{S}$ (Σχ. 6.1(c)). Προσθέτοντας τον μηχανισμό στάθμισης $\mathcal{W}$ καταλήγουμε στο Σχήμα 6.1(d). Τέλος, αν συνδυάσουμε τους δύο μηχανισμούς $\mathcal{S}$ και $\mathcal{W}$, τότε καταλήγουμε στο σχήμα 6.1(e). Η συνεισφορά των προτεινόμενων μηχανισμών συνοψίζεται στον Πίνακα 6.1.

Προκειμένου να επιβεβαιώσουμε ότι ο μηχανισμός συγχρονισμού είναι γενικός και αποδοτικός τον εφαρμόσαμε στους πιο γνωστούς αυτοδρομολογούμενους αλγόριθμους: CSS, GSS, FSS και TSS, δημιουργώντας τους αλγορίθμους $\mathcal{S}-CSS$, $\mathcal{S}-GSS$, $\mathcal{S}-FSS$ και $\mathcal{S}-TSS$. Κάθε ένας από τους συγχρονισμένους αλγόριθμους συγκρίθηκε με τον αλγόριθμο $DTSS$, που στο εξής θα αναφέρεται ως $\mathcal{SW}-TSS$. Η σύγκριση φανέρωσε την ανωτερότητα του $\mathcal{SW}-TSS$ έναντι των άλλων αλγορίθμων. Στην συνέχεια, εφαρμόστηκε ο μηχανισμός στάθμισης στους ίδιους αρχικούς αλγορίθμους σχηματίζοντας τους $\mathcal{W}-CSS$,$\mathcal{W}-GSS$, $\mathcal{W}-FSS$ και $\mathcal{W}-TSS$. Η απόδοση

Πίνακας 6.1: Συνεισφορά των προτεινόμενων μηχανισμών.

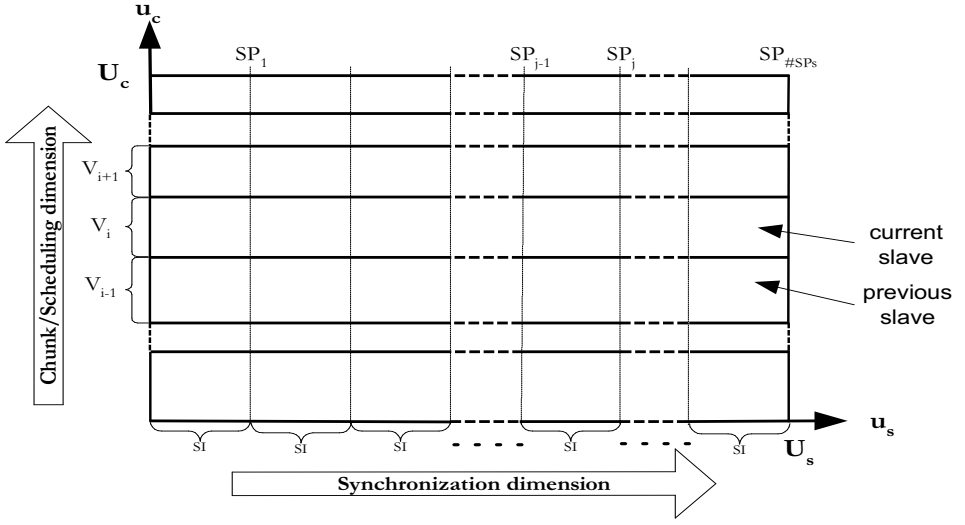
Αυτοδρομολογούμενος αλγόριθμος & αριθμός αναφοράς	Συγχρονισμένος αλγόριθμος	Σταθμισμένος αλγόριθμος	Συγχρονισμένος & Σταθμισμένος αλγόριθμος
CSS [12]	$\mathcal{S}-CSS$ [31]	$\mathcal{W}-CSS$	$\mathcal{SW}-CSS$
FSS [83]	$\mathcal{S}-FSS$	$\mathcal{W}-FSS$ άλλες σταθμισμένες προσεγγίσεις [82, 41]	$\mathcal{SW}-FSS$
GSS [13]	$\mathcal{S}-GSS$	$\mathcal{W}-GSS$	$\mathcal{SW}-GSS$
TSS [90]	$\mathcal{S}-TSS$ εμφανίστηκε ως $DMPS(TSS)$ [31]	$\mathcal{W}-TSS$ εμφανίστηκε ως $DTSS$ [4]	$\mathcal{SW}-TSS$ εμφανίστηκε ως $DMPS(DTSS)$ [31]

των σταθμισμένων αλγορίθμων συγκρίθηκε με αυτή των αρχικών. Σε κάθε περίπτωση οι σταθμισμένοι αλγόριθμοι απέδωσαν σημαντικά καλύτερα από τις μη-σταθμισμένες εκδοχές τους. Τέλος, εφαρμόσαμε και τους δύο μηχανισμούς ταυτόχρονα στους αρχικούς αλγορίθμους και συγκρίναμε την απόδοση τους με αυτή των $\mathcal{S}-CSS$, $\mathcal{S}-GSS$, $\mathcal{S}-FSS$ και $\mathcal{S}-TSS$. Και σε αυτή την περίπτωση, οι σταθμισμένες εκδοχές απέδωσαν πολύ καλύτερα από τις μη-σταθμισμένες. Από τα πειραματικά αποτελέσματα βγαίνει το συμπέρασμα ότι, οι βρόχοι που περιέχουν εξαρτήσεις μπορούν να δρομολογηθούν αποδοτικά με την χρήση κατάλληλα τροποποιημένων αυτοδρομολογούμενων αλγορίθμων.

6.2 Ο μηχανισμός Συγχρονισμού

Ο στόχος του μηχανισμού συγχρονισμού είναι να επιτρέψει ανοίξει το πεδίο των εφαρμογών που περιέχουν φωλιασμένους βρόχους με εξαρτήσεις στους αυτοδρομολογούμενους αλγόριθμους. Η λογική και η λειτουργικότητα του μηχανισμού συγχρονισμού είναι ίδια με το σχήμα επικοινωνιών που περιγράφηκε στο προηγούμενο κεφάλαιο, σχετικά με τον αλγόριθμο δρομολόγησης πολλαπλών φάσεων. Σε αυτή την ενότητα θα επικεντρωθούμε στην γενίκευση του μηχανισμού έτσι ώστε να μπορεί να εφαρμοστεί εύκολα και αποδοτικά σε όλους τους αυτοδρομολογούμενους αλγόριθμους. Θα πρέπει να τονίσουμε ότι ο μηχανισμός συγχρονισμού είναι τελείως ανεξάρτητος από τον αλγόριθμο δρομολόγησης και δεν επηρεάζει την ικανότητα του στην κατανομή φορτίου. Επομένως, ο συγχρονισμένος δυναμικός αλγόριθμος, θα έχει καλή απόδοση σε ετερογενή περιβάλλοντα μόνο αν ο αρχικός αλγόριθμος μπορεί να χειριστεί την ανομοιογένεια.

Ο μηχανισμός συγχρονισμού $\mathcal{S}$ παρέχει το διάστημα συγχρονισμού SI, κατά μήκος της διάστασης συγχρονισμού u_s και ένα πλαίσιο για ανταλλαγή δεδομένων μεταξύ των εργατών.



Σχήμα 6.2: *Καταμερισμός ενός βρόχου 2 διαστάσεων σε ομάδες επαναλήψεων και τοποθέτηση σημείων συγχρονισμού.*

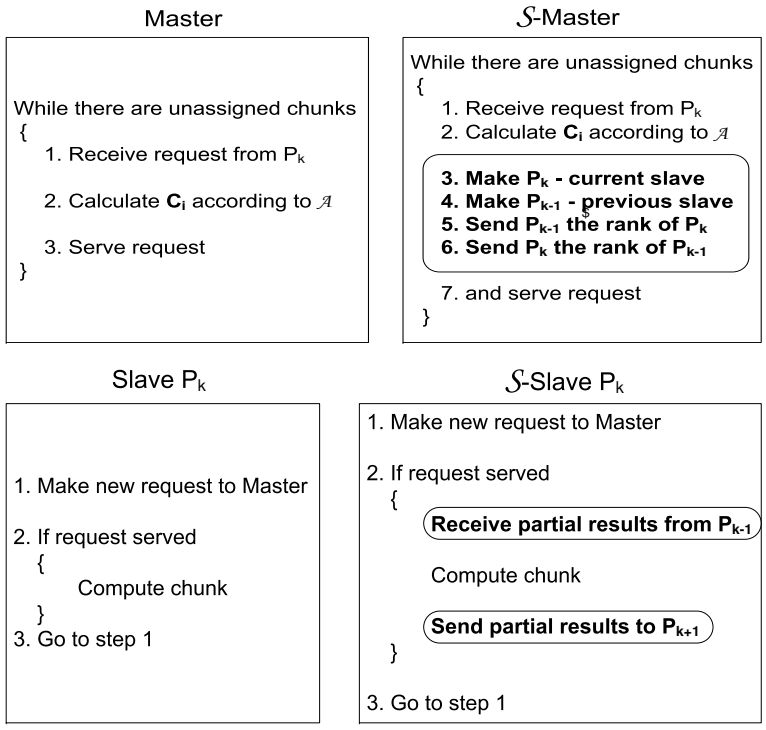
$$SI = \lceil \frac{U_s}{\#SPs} \rceil \tag{6.1}$$

Το Σχήμα 6.2 απεικονίζει το μέγεθος των ομάδων επαναλήψεων V_i, καθώς και τα διαστήματα συγχρονισμού SI. Στο ίδιο σχήμα, οι οριζόντιες λωρίδες αντιστοιχούν σε ομάδες επαναλήψεων. Τα σημεία συγχρονισμού τοποθετούνται στην διάσταση συγχρονισμού u_s. Σε αυτό το παράδειγμα, C_i είναι το πλήθος των επαναλήψεων που περιλαμβάνει η ομάδα επαναλήψεων i και $C_i = V_i \times U_s$.

6.2.1 Επέκταση συγχρονισμού των αυτοδρομολογούμενων αλγορίθμων

Ο μηχανισμός $\mathcal{S}$ προσθέτει τρία εξαρτήματα στον αρχικό αλγόριθμο $\mathcal{A}$, όπως φαίνεται στο Σχήμα 6.3: (1) Στη πλευρά συντονιστή, **καταγραφή συναλλαγών** (transaction accounting) - σύμφωνα με τις αιτήσεις των εργατών, ο συντονιστής εξακριβώνει τις ταυτότητες των εργατών που πρέπει να ανταλλάξουν δεδομένα (*προηγούμενος* και *τρέχων* εργάτης)· πλευρά εργάτη, (2) **λήψη δεδομένων** (receive part)- ο εργάτης λαμβάνει τα απαραίτητα δεδομένα σύμφωνα με τις υποδείξεις του συντονιστή, (3) **αποστολή δεδομένων** (transmit part)- ο εργάτης στέλνει δεδομένα στον επόμενο εργάτη σύμφωνα με τις υποδείξεις του συντονιστή. Περισσότερες πληροφορίες για τα παραπάνω εξαρτήματα δίνονται στον παρακάτω ψευδοκώδικα:

Λήψη δεδομένων: λήψη δεδομένων από τον P_{k-1}.
Στο σημείο συγχρονισμού SP_i έλεγξε για μερικά αποτελέσματα από τον προηγού-



Σχήμα 6.3: *Το μοντέλο συντονιστή-εργάτη με τον μηχανισμό συγχρονισμού.*

μενο εργάτη P_{k-1}. (Η ταυτότητα του P_{k-1} βρίσκεται στις πληροφορίες της ομάδας επαναλήψεων)

Αν ο προηγούμενος εργάτης ταυτίζεται με τον τρέχων ($P_k = P_{k-1}$)

- Όλα τα μερικά αποτελέσματα υπάρχουν ήδη στην μνήμη του SP_i
- Προχώρησε στον υπολογισμό της ομάδας επαναλήψεων, χωρίς να λάβεις άλλα δεδομένα

Αλλιώς ($P_k! = P_{k-1}$)

- Λάβε τα διαθέσιμα σύνολα επικοινωνιών (communication sets) από τον P_{k-1}
- Έλεγξε το πλήθος b των συνόλων επικοινωνιών που έχουν ληφθεί

  Αν $b > 1$, παράλειψε τον έλεγχο για τα επόμενα b σημεία συγχρονισμού (μέχρι το SP_{i+b})

προχώρα στην φάση υπολογισμού

Αποστολή δεδομένων: Αποστολή δεδομένων στον P_{k+1}.
Στο σημείο συγχρονισμού SP_i:

Αν είναι το πρώτο σημείο συγχρονισμού (SP_1)

- Ρώτα τον συντονιστή για την ταυτότητα του P_{k+1} (non-blocking request)
- **Αν ο συντονιστής δεν ξέρει ακόμα τον** P_{k+1}
  - Αποθήκευσε τα σύνολα επικοινωνιών στην τοπική μνήμη
  - Προχώρα στην φάση υπολογισμού
- **Αλλιώς**
  - Πάρε την ταυτότητα του P_{k+1}
  - Στείλε στον P_{k+1} το σύνολο επικοινωνιών

Αλλιώς ($SP_i, i \neq 1$)

- **Αν ο** P_k **ξέρει ήδη τον** P_{k+1}
  - Στείλε στον P_{k+1} τα σύνολα επικοινωνιών
- **Αλλιώς**
  - **Αν ο συντονιστής έχει στείλει στον** P_k **την ταυτότητα του** P_{k+1}
    - Στείλε στον P_{k+1} τα όλα τα αποθηκευμένα σύνολα επικοινωνιών σε ένα πακέτο
    - προχώρα στην φάση υπολογισμού
  - **Αλλιώς**
    - Αποθήκευσε το νέο σύνολο επικοινωνιών στην τοπική μνήμη
    - προχώρα στην φάση υπολογισμού

προχώρα στην φάση υπολογισμού.

6.2.2 Εμπειρικός προσδιορισμός αριθμού σημείων συγχρονισμού

Για γίνει κατανοητή η επίδραση της επιλογής του αριθμού των σημείων συγχρονισμού στη απόδοση, δίνεται στο Σχήμα 9.1 η παράλληλη εκτέλεση μιας υποθετικής εφαρμογής, σε $m = 4$ εργάτες (Για συντομία συμβολίζουμε τον 'αριθμό' με '#' και άρα ο αριθμός SP γίνεται $\#SP$). Για λόγους απλότητας, υποθέτουμε ότι σε κάθε εργάτη αναθέτονται μόνο δύο ομάδες επαναλήψεων καθώς και ότι όλες οι ομάδες έχουν το ίδιο μέγεθος. Στο ίδιο σχήμα δίνεται η σειρά των αιτήσεων των εργατών, που είναι P_1, P_2, P_3, P_4 και τα σημεία συγχρονισμού $SP_1, \ldots, SP_{12}$, τα οποία έχει τοποθετήσει ο μηχανισμός συγχρονισμού. Στο εσωτερικό κάθε υποομάδας επαναλήψεων $SC_{i,j}$ αναγράφεται η χρονική στιγμή την οποία εκτελείται. Η ροή της εκτέλεσης ακολουθεί ένα συγκεκριμένο μέτωπο κύματος (wavefront). Μπορεί να παρατηρηθεί ότι στα βήματα 1 με 3 (αρχικά βήματα) καθώς στα βήματα 25 με 27 (τελικά βήματα), δεν είναι ενεργοί όλοι οι εργάτες ταυτόχρονα, σε αντίθεση με τα βήματα 4 με 24 (ενδιάμεσα βήματα), στα οποία όλοι οι εργάτες εργάζονται ταυτόχρονα. Στο τέλος του 12ου βήματος, ο εργάτης P_1 έχει ολοκληρώσει την εκτέλεση της πρώτης ομάδας επαναλήψεων και ο συντονιστής του αναθέτει την δεύτερη ομάδα επαναλήψεων. Αρχίζει την

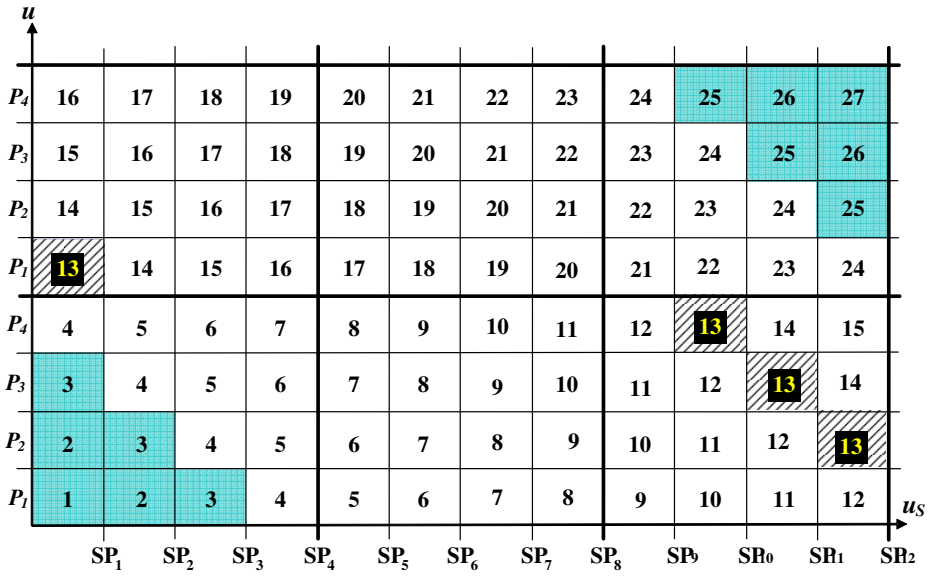
εκτέλεση της δεύτερης ομάδας στο βήμα 13, αφού έχει όλα τα απαραίτητα δεδομένα διαθέσιμα από τον εργάτη P_4. Στο ίδιο χρονικό βήμα, οι εργάτες P_2, P_3, P_4 εκτελούν ακόμα την πρώτη ομάδα επαναλήψεων. Ο εργάτης P_2 ολοκληρώνει την πρώτη ομάδα επαναλήψεων στο βήμα 13 και προχωράει στην επόμενη στο βήμα 14. Η μετάβαση στην δεύτερη ομάδα για τους εργάτες P_3 και P_4 λαμβάνει χώρα τα χρονικά βήματα 14 και 15 αντίστοιχα. Με άλλα λόγια, αν εξαιρέσουμε τα αρχικά και τα τελικά βήματα, η όλη εκτέλεση προχωράει χωρίς καθυστερήσεις, εκτός από τον συγχρονισμό των εργατών και τις ανταλλαγές δεδομένων στα σημεία συγχρονισμού. Ο αριθμός των *αρχικών βημάτων* και των *τελικών βημάτων* είναι $(m-1)$, άρα ο αριθμός των *ενδιάμεσων βημάτων* είναι $\#$*ενδιάμεσων βημάτων* $=$ $\#$*συνόλου βημάτων* - $2(m-1)$, όπως φαίνεται και στο Σχήμα 9.1. Ο συνολικός αριθμός των χρονικών βημάτων εξαρτάται από τον αριθμό των σημείων συγχρονισμού ($\#SPs$) και τον αριθμό των ομάδων επαναλήψεων που παράγει ο αλγόριθμος δρομολόγησης. Αφού ο αριθμός και το μέγεθος των ομάδων επαναλήψεων εξαρτάται από τον αλγόριθμο δρομολόγησης, η επιλογή του αριθμού των βημάτων δρομολόγησης θα πρέπει να μεγιστοποιεί την αναλογία των ενδιάμεσων χρονικών βημάτων σε σχέση με τα συνολικά χρονικά βήματα.

Η επιλογή του αριθμού των σημείων συγχρονισμού καθορίζει την ισορροπία μεταξύ του κόστους συγχρονισμού και του διαθέσιμου παραλληλισμού. Ένας μεγάλος αριθμός σημείων συγχρονισμού επιφέρει πολύ συχνές ανταλλαγές δεδομένων και μεγάλο κόστος συγχρονισμού, ενώ ένας μικρός αριθμός σημείων συγχρονισμού περιορίζει τον παραλληλισμό. Η βέλτιστη επιλογή εξαρτάται από πολλούς παράγοντες όπως, η μορφή των εξαρτήσεων του φωλιασμένου βρόχου, τα χαρακτηριστικά του δικτύου διασύνδεσης και των εργατών, ο αλγόριθμος δρομολόγησης που χρησιμοποιείται. Εκτεταμένες πειραματικές μετρήσεις για διάφορες περιπτώσεις εφαρμογών και αλγορίθμων δρομολόγησης (συνοψίζονται στην ενότητα 6), έχουν δείξει ότι μια αυθαίρετη αλλά καλή επιλογή είναι $\#SPs \geq 3 * m$. Στο παράδειγμα του Σχήματος 9.1, έχουμε $\#SPs = 12$, το οποίο δίνει $\#$*συνολικών βημάτων* $= 27$, $\#$*ενδιάμεσων βημάτων* $= 21$, και ένα ποσοστό 77% του χρόνου εκτέλεσης χωρίς καθυστερήσεις.

6.3 Ο Μηχανισμός Στάθμισης

Ο μηχανισμός στάθμισης χρησιμοποιείται για να τροποποιήσει τους αυτοδρομολογούμενους αλγόριθμους έτσι ώστε να προσαρμόζονται σε μεταβολές του φόρτου εργασίας και στην ανομοιογένεια του συστήματος. Για να επιτευχθεί ικανοποιητική κατανομή φορτίου στους διαθέσιμους εργάτες, οι εργασίες, δηλαδή οι ομάδες επαναλήψεων θα πρέπει να υπολογίζονται με βάση την τρέχουσα κατάσταση της ουράς εκτέλεσης του κάθε εργάτη. Η κατάσταση της ουράς εκτέλεσης δεν παραμένει σταθερή σε κάθε ανάθεση, αλλά αντικατοπτρίζει την μεταβολή του φόρτου εργασίας. Για αυτό το λόγο ορίζουμε έναν μηχανισμό στάθμισης, ο οποίος μπορεί να εφαρμοστεί σε κάθε αυτοδρομολογούμενο αλγόριθμο ο οποίος αυξομειώνει το μέγεθος της ομάδας επαναλήψεων ανάλογα με την σχετική υπολογιστική δύναμη και το μέγεθος της ουράς εκτέλεσης κάθε εργάτη.

Υποθέτουμε ότι ένας αυτοδρομολογούμενος αλγόριθμος $\mathcal{A}$ χρησιμοποιείται για



Σχήμα 6.4: *Παράλληλη εκτέλεση σε μορφή σωλήνωσης*

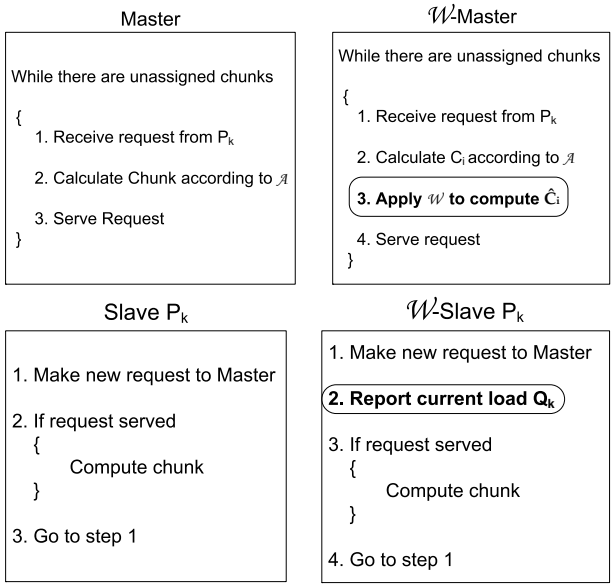
την δρομολόγηση ενός φωλιασμένου βρόχου σε ένα ετερογενές σύστημα που αποτελείται από m εργάτες, με σχετική υπολογιστική δύναμη $VP_1, \ldots, VP_m$. Επίσης, υποθέτουμε ότι στο i-οστό βήμα δρομολόγησης, ο εργάτης P_k με δύναμη VP_k, έχει Q_k διεργασίες στην ουρά εκτέλεσης του. Ο μηχανισμός στάθμισης $\mathcal{W}$, σταθμίζει το μέγεθος της ομάδας υπολογισμών $\widehat{C_i}$ που θα ανατεθεί στον εργάτη P_k, στο βήμα δρομολόγησης i, σύμφωνα με την παρακάτω σχέση:

$$\widehat{C_i} = C_i \times \frac{VP_k}{Q_k} \tag{6.2}$$

Προφανώς, σε περίπτωση που εφαρμόσουμε τον μηχανισμό στάθμισης σε ένα ομοιογενές και αφοσιωμένο σύστημα, το μέγεθος της ομάδας επαναλήψεων $\widehat{C_i}$ θα παραμείνει σταθερό, αφού ισχύει ότι $\forall k\ VP_k = 1$ και Q_k =1, δηλαδή όλοι οι εργάτες έχουν την ίδια υπολογιστική δύναμη, και δεν υπάρχουν άλλες διεργασίες στην ούρα εκτέλεσης εκτός από την εφαρμογή μας. Σε αυτή την περίπτωση, ο μηχανισμός δεν βελτιώνει την απόδοση του αλγορίθμου δρομολόγησης και άρα μπορεί να παραληφθεί.

6.3.1 Βελτίωση των αυτοδρομολογούμενων αλγορίθμων μέσω στάθμισης

Στο Σχήμα 6.5 βλέπουμε την προσθήκη του μηχανισμού σταθμίσματος σε ένα αλγόριθμο δρομολόγησης $\mathcal{A}$. Ο μηχανισμός προσθέτει δύο εξαρτήματα στον αρχικό αλγόριθμο: (1) Στην πλευρά συντονιστή, **στάθμιση ομάδας υπολογισμών** (chunk weighting) - Ο συντονιστής αυξομειώνει το μέγεθος των ομάδων επαναλήψεων σύμφωνα με την σχέση 6.2. (2) πλευρά εργάτη, **παρακολούθηση της ουράς**



Σχήμα 6.5: *Το μοντέλο συντονιστή-εργάτη με τον μηχανισμό στάθμισης.*

εκτέλεσης (run-queue monitor) -Καταγράφει το μέγεθος της ουράς εκτέλεσης, δηλαδή το Q_k και ενημερώνει τον συντονιστή για τον αριθμό διεργασιών στην ουρά του.

Ο Πίνακας 6.2 δίνει τα μεγέθη των ομάδων επαναλήψεων τα οποία προκύπτουν από ένα σύνολο αρχικών και των αντίστοιχων σταθμισμένων αυτοδρομολογούμενων αλγορίθμων. Τα μεγέθη αυτά αφορούν την εφαρμογή Mandelbrot, η οποία περιλαμβάνει ένα παράλληλο (Doall) φωλιασμένο βρόχο, με μεταβλητό βάρος, και για ένα χώρο επαναλήψεων 10000×10000 σημείων. Χρησιμοποιήθηκαν τέσσερις εργάτες, με σχετικές υπολογιστικές δυνάμεις $VP_1 = 1, VP_2 = 0.8, VP_3 = 1$ και $VP_4 = 0.8$. Οι δύο πιο αργοί εργάτες είχαν επιπλέον διεργασίες στην ουρά εκτέλεσης τους, $Q_2 = 2, Q_4 = 2$ έτσι, η διαθέσιμη υπολογιστική τους δύναμη ήταν $A_2 = 0.4$ και $A_4 = 0.4$.

Ο Πίνακας 6.2 δίνει επίσης την σειρά με την οποία ζήτησαν δουλειά οι εργάτες από τον συντονιστή, η οποία διαφέρει από αλγόριθμο σε αλγόριθμο. Οι αυτοδρομολογούμενοι αλγόριθμοι είχαν σχεδιαστεί για ομοιογενή συστήματα και έχουν την τάση να αναθέτουν μεγάλες ομάδες υπολογισμών στα πρώτα βήματα δρομολόγησης, σε όλους τους εργάτες. Βασίζονται στην υπόθεση ότι όλοι οι εργάτες θα εκτελέσουν τους υπολογισμούς που τους ανατέθηκαν στον ίδιο περίπου χρόνο και θα προχωρήσουν ταυτόχρονα στην επόμενη ομάδα υπολογισμών, όπως εξηγείται στο [29]. Αυτή η υπόθεση δεν ευσταθεί για τα ετερογενή συστήματα. Οι πιο αργοί εργάτες μπορεί να καθυστερήσουν έναντι των γρηγορότερων, αφού χρειάζονται περισσότερο χρόνο για να εκτελέσουν την ίδια ομάδα επαναλήψεων. Ο μηχανισμός στάθμισης αντισταθμίζει την έλλειψη υπολογιστικής δύναμης των αργών εργατών όπως φαίνεται στον Πίνακα

6.2. Στον ίδιο πίνακα μπορούμε επίσης να δούμε την μείωση του χρόνου εκτέλεσης των σταθμισμένων αλγορίθμων, έναντι των αντίστοιχων μη-σταθμισμένων.

6.4 Ο συνδυασμένος μηχανισμός $\mathcal{SW}$

Στην ενότητα 6.2 εφαρμόσαμε τον μηχανισμό συγχρονισμού $\mathcal{S}$ για την δρομολόγηση εφαρμογών που περιέχουν βρόχους με εξαρτήσεις, ενώ στην ενότητα 6.3 εφαρμόσαμε τον μηχανισμό στάθμισης $\mathcal{W}$ για την δρομολόγηση εφαρμογών που παρέχουν βρόχους χωρίς εξαρτήσεις. Σε αυτή την ενότητα, συνδυάζουμε και τους δύο μηχανισμούς. Ο μηχανισμός συγχρονισμού, αν και είναι απαραίτητος για την δρομολόγηση βρόχων που περιέχουν εξαρτήσεις, δεν βελτιώνει την ικανότητα κατανομής φορτίου του αρχικού αλγορίθμου. Οπότε σε ανομοιογενή, μη αφοσιωμένα συστήματα, είναι ωφέλιμο να εφαρμόζουμε και τους δύο μηχανισμούς ταυτόχρονα.

Πίνακας 6.2: Μεγέθη ομάδων επαναλήψεων που δίνονται από τους αρχικούς και τους σταθμισμένους αλγόριθμους για την εφαρμογή Mandelbrot, και για χώρο επαναλήψεων $|J| = 10000 \times 10000$ σημείων, για $m = 4$.

$\mathcal{A}$	Μεγέθη ομάδων του $\mathcal{A}$ και σειρά αιτήσεων των εργατών	Μεγέθη ομάδων του $\mathcal{W}-\mathcal{A}$ και σειρά αιτήσεων των εργατών	Παράλληλος χρόνος του $\mathcal{A}$	Παράλληλος χρόνος του $\mathcal{W}-\mathcal{A}$
CSS	1250(P_1) 1250(P_2) 1250(P_3) 1250(P_4) 1250(P_3) 1250(P_1) 1250(P_3) 1250(P_1)	1250(P_1) 1250(P_3) 500(P_4) 500(P_2) 1250(P_3) 500(P_2) 500(P_4) 1250(P_1) 1250(P_3) 500(P_4) 1250(P_1)	120.775s	66.077s
FSS	1250(P_1) 1250(P_3) 1250(P_2) 1250(P_4) 625(P_3) 625(P_3) 625(P_1) 625(P_3) 390(P_1) 390(P_1) 390(P_3) 390(P_1) 244(P_3) 244(P_4) 244(P_1) 208(P_3)	1250(P_1) 1250(P_3) 500(P_2) 500(P_4) 812(P_3) 324(P_2) 324(P_4) 324(P_1) 324(P_3) 812(P_3) 630(P_1) 630(P_1) 630(P_4) 252(P_3) 176(P_1) 441(P_4) 441(P_2) 176(P_3) 123(P_1) 308(P_2) 308(P_4) 113(P_1)	120.849s	56.461s
GSS	2500(P_1) 1875(P_2) 1406(P_3) 1054(P_4) 791(P_3) 593(P_3) 445(P_3) 334(P_1) 250(P_3) 188(P_1) 141(P_3) 105(P_1) 80(P_3) 80(P_1) 80(P_3) 78(P_1)	2500(P_1) 1875(P_3) 562(P_2) 506(P_4) 455(P_4) 410(P_2) 923(P_3) 692(P_3) 519(P_1) 155(P_4) 140(P_2) 315(P_3) 94(P_4) 213(P_1) 160(P_3) 120(P_1) 90(P_3) 80(P_2) 80(P_1) 80(P_3) 31(P_1)	145.943s	58.391s
TSS	1250(P_1) 1172(P_3) 1094(P_2) 1016(P_4) 938(P_3) 860(P_1) 782(P_3) 704(P_1) 626(P_3) 548(P_4) 470(P_2) 392(P_1) 148(P_3)	1250(P_1) 1172(P_3) 446(P_2) 433(P_4) 1027(P_3) 388(P_4) 375(P_2) 882(P_1) 804(P_3) 299(P_4) 286(P_2) 660(P_1) 582(P_3) 504(P_1) 179(P_4) 392(P_3) 134(P_2) 187(P_1)	89.189s	63.974s

Αφού το διάστημα συγχρονισμού παραμένει σταθερό για κάθε ομάδα επαναλήψεων, το πλήθος των δεδομένων που πρέπει να ανταλλάξουν δύο διαδοχικοί εργάτες, για κάθε υποομάδα υπολογισμών παραμένει σταθερό, και άρα ο χρόνος επικοινωνίας δεν μεταβάλλεται σημαντικά. Επίσης, αφού το μέγεθος των ομάδων επαναλήψεων σταθμίζεται ανάλογα με την δύναμη και το φόρτο του κάθε εργάτη, περιμένουμε ότι και ο χρόνος υπολογισμού κάθε υποομάδας επαναλήψεων δεν θα μεταβάλλεται σημαντικά από εργάτη σε εργάτη. Αυτό σημαίνει ότι ο λόγος επικοινωνιών / υπολογισμών θα παραμένει σταθερός, το οποίο με την σειρά του υπονοεί καλή κατανομή υπολογιστικού φορτίου. Είναι λοιπών ωφέλιμο να εφαρμόζουμε τον συνδυασμό των δύο μηχανισμών. Είναι προφανές ότι ένας αυτοδρομολογούμενος αλγόριθμος που περιλαμβάνει τον μηχανισμό στάθμισης θα υπερτερεί έναντι του μη σταθμισμένου αντίστοιχου αλγορίθμου στην περίπτωση του ανομοιογενούς και μη αφοσιωμένου συστήματος.

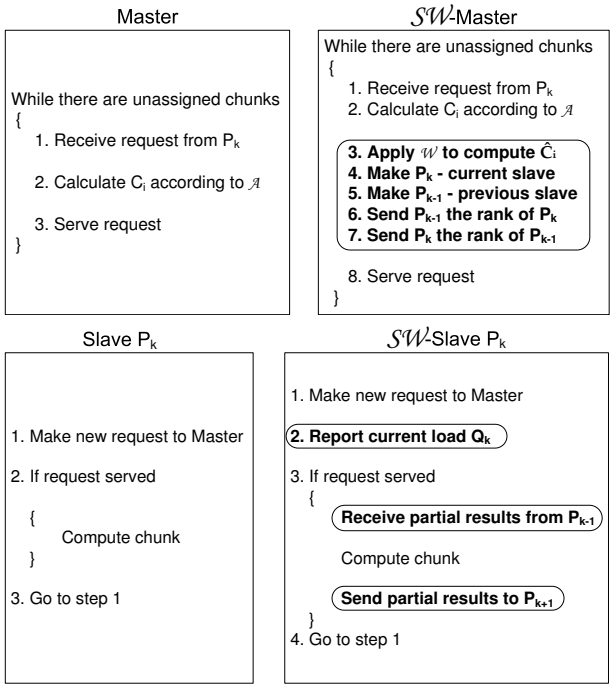
Το Σχήμα 6.6 παρουσιάζει την εφαρμογή των μηχανισμών συγχρονισμού και στάθμισης για την δρομολόγηση εφαρμογών που περιέχουν φωλιασμένους βρόχους με εξαρτήσεις σε ανομοιογενή και μη αφοσιωμένα συστήματα υπολογιστών. Όπως και στις προηγούμενες περιπτώσεις, ο συνδυασμένος μηχανισμός $\mathcal{SW}$ προσθέτει δύο επιπλέον συστατικά στον συντονιστή : (1) **στάθμιση ομάδας επαναλήψεων** (chunk weighting) και (2) **καταγραφή συναλλαγών** (transaction accounting) και τρία συστατικά στον εργάτη: (1) **παρακολούθηση της ουράς εκτέλεσης** (run-queue monitor)(2) **λήψη δεδομένων** (receive part) (3) **αποστολή δεδομένων** (transmit part). Τα συστατικά (1) του συντονιστή καθώς και το (1) του εργάτη σχετίζονται με τον μηχανισμό συγχρονισμού, ενώ τα υπόλοιπα σχετίζονται με τον μηχανισμό της στάθμισης.

Τα μεγέθη των ομάδων υπολογισμού πού υπολογίζονται από τους συγχρονισμένους και τους αντίστοιχους συγχρονισμένουσ-σταθμισμένους αλγόριθμους δίνονται στον Πίνακα 6.3. Η εφαρμογή που εξετάζεται είναι ο υπολογισμός Floyd-Steinberg (που έχει περιγραφεί σε προηγούμενη ενότητα) και ο χώρος επαναλήψεων αποτελείται από 10000×10000 σημεία. Σε αυτή την περίπτωση, λόγο της ύπαρξης των εξαρτήσεων, η σειρά των αιτήσεων για εργασία των εργατών διατηρείται σταθερή για κάθε αλγόριθμο. Και σε αυτή την περίπτωση μπορούμε να παρατηρήσουμε ότι οι συγχρονισμένεσ-σταθμισμένες εκδόσεις των αλγορίθμων υπερτερούν των μη-σταθμισμένων συγχρονισμένων εκδοχών τους.

Η υλοποίηση του συνδυασμένου μηχανισμού συνθέτεται από την εισαγωγή όλων των τμημάτων κώδικα και των δύο ανεξάρτητων μηχανισμών, όπως φαίνεται στο Σχήμα 6.6.

6.5 Πειραματική αξιολόγηση

Σύμφωνα με τον ψευδοκώδικα της ενότητας 6.2.1 καταλήγουμε σε δύο τμήματα κώδικά C τα οποία υλοποιούν τον μηχανισμό επικοινωνιών στην πλευρά του εργάτη. Σαν διάσταση συγχρονισμού έχουμε επιλέξει την εξωτερική διάσταση του φωλιασμένου βρόχου και την αμέσως επόμενη σαν διάσταση δρομολόγησης. Τα δύο τμήματα κώδικα που χειρίζονται τις επικοινωνίες εισέρχονται μεταξύ των δύο αυτών βρόχων. Το τμήμα λήψης ακριβώς πριν τον υπολογισμό των δεδομένων, ενώ το τμήμα αποστο-



Σχήμα 6.6: *Το μοντέλο συντονιστή-εργάτη με τους μηχανισμούς συγχρονισμού και στάθμισης.*

λής ακριβώς μετά τον υπολογισμό των δεδομένων. Για να υλοποιήσουμε τα σημεία συγχρονισμού , παρακολουθούμε την τιμή του δείκτη των επαναλήψεων της διάστασης συγχρονισμού και ορίζουμε τα σημεία $SP_1 = SI$ και $SP_{i+1} = SP_i + SI$, όπου $SI = \frac{U_s}{\#SP}$. Ο αριθμός των σημείων συγχρονισμού καθορίζεται εμπειρικά, όπως περιγράφεται στην επόμενη υποενότητα. Όταν ο δείκτης των επαναλήψεων της διάστασης u_s φτάσει σε κάποιο σημείο συγχρονισμού ενεργοποιεί το αντίστοιχο τμήμα κώδικα, ανάλογα με το αν θα πρέπει να λάβει ή να στείλει δεδομένα. Στην πλευρά του Συντονιστή, ένα κομμάτι κώδικα προστίθεται, έτσι ώστε μόλις ένας εργάτης κάνει μία νέα αίτηση για δουλεία, καταγράφεται σαν τρέχων *εργάτης*. Ο εργάτης που είχε κάνει αίτηση πριν τον *τρέχων* μετονομάζεται σε *προηγούμενο* εργάτης. Αυτή η πληροφορία αποστέλλεται ασύγχρονα στους δύο εργάτες. Έτσι, ο *τρέχων* εργάτης θα λάβει δεδομένα από τον *προηγούμενο* εργάτη. Όταν ένας νέος εργάτης κάνει αίτηση για δουλεία, θα γίνει αυτός *τρέχων* εργάτης και θα λάβει δεδομένα από τον εργάτη που μέχρι τώρα ονομαζόταν *τρέχων* και μετονομάστηκε σε *προηγούμενο* εργάτη.

Τα τμήματα κώδικα που υλοποιούν τον μηχανισμό στάθμισης είναι πολύ απλούστερα και συντομότερα από αυτά που υλοποιούν τον μηχανισμό συγχρονισμού, όπως φαίνεται και στο Σχήμα 6.5. Πιο συγκεκριμένα προσθέσαμε ένα τμήμα στον κώδικα του εργάτη που καταγράφει το τρέχων φορτίο την στιγμή που ο εργάτης πραγμα-

Πίνακας 6.3: Μεγέθη ομάδων επαναλήψεων που δίνονται από τους συγχρονισμένους και τους συγχρονισμένουσ-σταθμισμένους αλγόριθμους για την εφαρμογή Floyd-Steinberg, και για χώρο επαναλήψεων $|J| = 10000 \times 10000$ σημείων, για $m = 4$.

$\mathcal{A}$	Μεγέθη ομάδων του $\mathcal{S}$-$\mathcal{A}$ και σειρά αιτήσεων των εργατών	Μεγέθη ομάδων του $\mathcal{SW}$-$\mathcal{A}$ και σειρά αιτήσεων των εργατών	Παράλληλος χρόνος του $\mathcal{S}$-$\mathcal{A}$	Παράλληλος χρόνος του $\mathcal{SW}$-$\mathcal{A}$
CSS	1250(P_1) 1250(P_3) 1250(P_2) 1250(P_4) 1250(P_1) 1250(P_3) 1250(P_2) 1250(P_4)	1250(P_1) 1250(P_3) 500(P_2) 500(P_4) 1250(P_1) 1250(P_3) 500(P_2) 500(P_4) 1250(P_1) 1250(P_3) 500(P_2)	27.335s	16.582s
FSS	1250(P_1) 1250(P_3) 1250(P_2) 1250(P_4) 625(P_1) 625(P_3) 625(P_2) 625(P_4) 390(P_1) 390(P_3) 390(P_2) 390(P_4) 244(P_1) 244(P_3) 244(P_2) 208(P_4)	1250(P_1) 1250(P_3) 500(P_2) 500(P_4) 812(P_1) 812(P_3) 324(P_2) 324(P_4) 630(P_1) 630(P_3) 252(P_2) 252(P_4) 488(P_1) 488(P_3) 195(P_2) 195(P_4) 378(P_1) 378(P_3) 151(P_2) 151(P_4) 40(P_1)	27.667s	16.556s
GSS	2500(P_1) 1875(P_3) 1406(P_2) 1054(P_4) 791(P_1) 593(P_3) 445(P_2) 334(P_4) 250(P_1) 188(P_3) 141(P_2) 105(P_4) 80(P_1) 80(P_3) 80(P_2) 78(P_4)	2500(P_1) 1875(P_3) 562(P_2) 506(P_4) 1139(P_1) 854(P_3) 256(P_2) 230(P_4) 519(P_1) 389(P_3) 116(P_2) 105(P_4) 237(P_1) 178(P_3) 80(P_2) 80(P_4) 108(P_1) 81(P_3) 80(P_2) 80(P_4) 25(P_1)	28.526s	18.569s
TSS	1250(P_1) 1172(P_2) 1094(P_3) 1016(P_4) 938(P_1) 860(P_2) 782(P_3) 704(P_4) 626(P_1) 548(P_2) 470(P_3) 392(P_4) 148(P_1)	509(P_2) 1217(P_1) 464(P_4) 1105(P_3) 420(P_2) 995(P_1) 376(P_4) 885(P_3) 332(P_2) 775(P_1) 288(P_4) 665(P_3) 244(P_2) 555(P_1) 200(P_4) 445(P_3) 156(P_2) 335(P_1) 34(P_4)	25.587s	14.309s

τοποιεί μια αίτηση για δουλειά στον συντονιστή. Αυτό το φορτίο αναφέρεται στον συντονιστή μαζί με την αίτηση για δουλεία. Από την πλευρά του συντονιστή, το τμήμα του κώδικα που υλοποιεί τον μηχανισμό $\mathcal{W}$ εκτελεί τον τύπο που δίνεται στην σχέση 6.2.

Η εισαγωγή αυτών των τμημάτων κώδικα μπορεί να γίνει αυτόματα με την χρήση κάποιου προεπεξεργαστή που εντοπίζει την κατάλληλη θέση για την εισαγωγή τους στον αρχικό κώδικα των αλγορίθμων δρομολόγησης. Στα πειράματα που παρουσιάζονται στην συνέχεια, η εισαγωγή έγινε χειροκίνητα, ωστόσο σε καμία περίπτωση δεν χρειάστηκε να ξαναγράφει ο αρχικός κώδικας. Το ίδιο ισχύει και για την περίπτωση του μηχανισμού στάθμισης W.

6.5.1 Παραδείγματα εφαρμογών.

Για την αξιολόγηση των μηχανισμών συγχρονισμού και στάθμισης χρησιμοποιήθηκαν τρεις διαφορετικές πειραματικές εφαρμογές. Η πρώτη εφαρμογή υπολογίζει τα σημεία που ανήκουν στο σύνολο Mandelbrot. Το σύνολο Mandelbrot αποτελείται από τα σημεία στο μιγαδικό επίπεδο το όριο των οποίων σχηματίζει μορφοκλάσματα (fractals). Ο υπολογισμός αυτός βρίσκει εφαρμογή σε συστήματα που παρουσιάζουν χαοτική συμπεριφορά. Το σύνολο Mandelbrot παράγεται από την επαναληπτική εξίσωση $z_{n+1} = z_n^2 + C$, με $z_0 = C$, όπου τα σημεία C στο μιγαδικό επίπεδο για τα οποία η τροχεία του z_n δεν τείνει στο άπειρο ανήκουν στο σύνολο. Η εφαρμογή αυτή δεν έχει εξαρτήσεις (Doall loop), αλλά παρουσιάζει μεγάλη ανομοιομορφία στο υπολογιστικό φορτίο από επανάληψη σε επανάληψη (irregular, non-uniform workload). Παρουσιάζουμε αυτή την εφαρμογή για να εξετάσουμε την απόδοση του μηχανισμού στάθμισης. Ο ψευδοκώδικας της εφαρμογής Mandelbrot δίνεται παρακάτω:

```
/* Mandelbrot Computation*/
    for (hy=1; hy<=hyres; hy++)  { /* chunk dimension */
        for (hx=1; hx<=hxres; hx++)  {
            cx = (((float)hx)/((float)hxres)-0.5)/magnify*3.0-0.7;
            cy = (((float)hy)/((float)hyres)-0.5)/magnify*3.0;
            x = 0.0; y = 0.0;
            for (iteration=1; iteration<itermax; iteration++)  {
                xx = x*x-y*y+cx;
                y = 2.0*x*y+cy;
                x = xx;
                if (x*x+y*y>100.0)  iteration = 999999;
            }
            if (iteration<99999)  color(0,255,255);
            else color(180,0,0);
        }
    }
```

Οι δύο επόμενες εφαρμογές **Floyd-Steinberg και hydrodynamics code fragment** έχουν παρουσιαστεί στο προηγούμενο κεφάλαιο και εντάσσονται στην κατηγορία των βρόχων με ομοιόμορφες και σταθερές εξαρτήσεις (UDL loops).

Και οι δύο εφαρμογές περιέχουν μοναδιαία διανύσματα εξαρτήσεων, αυτό εξασφαλίζει μικρό όγκο επικοινωνιών και έτσι είναι εύκολο να διατηρηθεί ο λόγος επικοινωνιών και υπολογισμών κάτω από την μονάδα.

6.5.2 Αποτελέσματα.

Πειραματική διάταξη. Η υλοποίηση μοντέλου συντονιστή-εργάτη, καθώς και των μηχανισμών συγχρονισμού και στάθμισης έγιναν σε γλώσσα C, με την χρήση της βιβλιοθήκης ανταλλαγής μηνυμάτων MPI. Τα πειράματα διεξάχθηκαν σε μία ετερογενή συστοιχία υπολογιστών (linux cluster), που αποτελείτο από 13 κόμβους, (1 κόμβο συντονιστή και 12 εργάτες). Η συστοιχία περιελάμβανε δύο τύπους μηχανημάτων: (α) 7 Intel Pentiums III 800MHz με 256MB RAM (που ονομάζονταν twins) και με σχετική υπολογιστική δύναμη $VP_k = 1$ και (β) 6 Intel Pentiums III 500MHz με 512MB

RAM (που ονομάζονταν kids), με σχετική υπολογιστική δύναμη $VP_k = 0.8$. Για τον καθορισμό της σχετικής υπολογιστικής δύναμης τρέξαμε 10 φορές σειριακά, ένα παράδειγμα εφαρμογής το οποίο παρείχε φωλιασμένους βρόχους και πράξεις κινητής υποδιαστολής, και βρήκαμε τον μέσο όρο του χρόνου εκτέλεσης σε κάθε τύπο μηχανήματος. Αυτό, αν και είναι μία απλή ένδειξη της υπολογιστικής δύναμης στην πράξη δίνει καλά αποτελέσματα για τον τύπο μηχανημάτων και εφαρμογών που εξετάστηκαν. Το δίκτυο διασύνδεσης ήταν 100Mbits/sec Fast Ethernet.

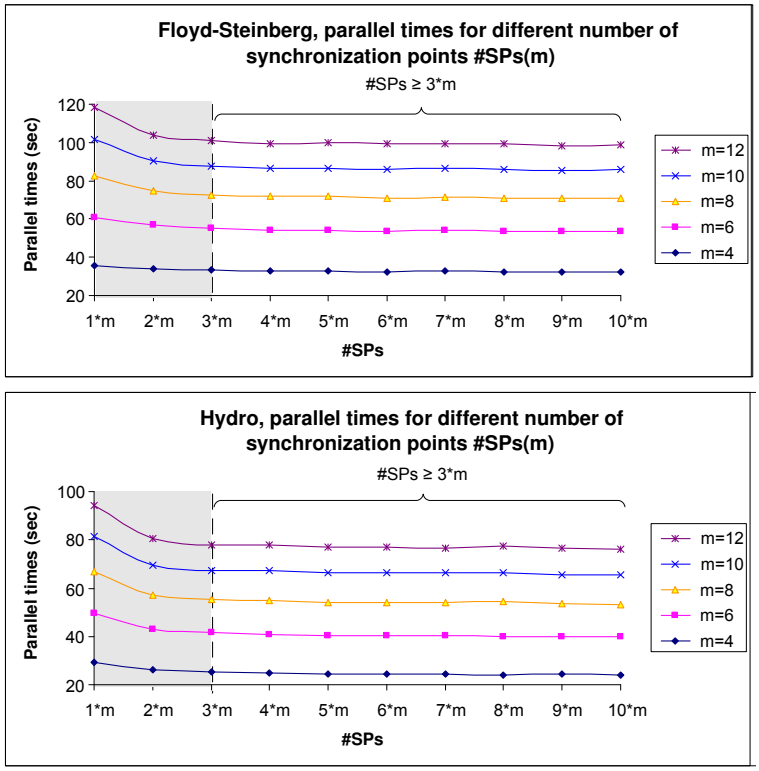
Για να εξομοιώσουμε τις συνθήκες που επικρατούν σε ένα μη-αφοσιωμένο σύστημα εκτελέσαμε στην αρχή κάθε σειράς πειραμάτων επιπλέον διεργασίες σε κάποιους από τους εργάτες όπως και στο προηγούμενο κεφάλαιο, με σκοπό να υποδιπλασιάσουμε την πραγματική υπολογιστική τους δύναμη. Τρέξαμε τρεις σειρές πειραμάτων: (1) για τον μηχανισμό συγχρονισμού ($\mathcal{S}$), (2) για τον μηχανισμό σταθμίσματος ($\mathcal{W}$) και (3) για τον συνδυασμό των δύο μηχανισμών ($\mathcal{SW}$). Εκτελέσαμε τις παραπάνω σειρές πειραμάτων για $m = 4, 6, 8, 10, 12$ εργάτες. Τα αποτελέσματα που δίνονται στις επόμενες υποενότητες αποτελούν τους μέσους όρους 10 εκτελέσεων για κάθε περίπτωση.

Χρησιμοποιήσαμε τα παρακάτω μηχανήματα: twin1(συντονιστής), twin2, **kid1**, twin3, **kid2**, twin4, **kid3**, twin5, **kid4**, twin6, **kid5**, twin7, **kid6**. Σε κάθε περίπτωση τα επιπλέον φορτία εκτελέστηκαν στα μηχανήματα που είναι γραμμένα με έντονα γράμματα. Στην περίπτωση $m = 4$, χρησιμοποιήθηκαν τα πρώτα τέσσερα μηχανήματα από την παραπάνω λίστα, όταν $m = 8$, τα πρώτα οκτώ, και ούτω καθεξής.

Πρώτη σειρά πειραμάτων. Στην πρώτη σειρά, πειραματιστήκαμε με τις δύο εφαρμογές που περιέχουν εξαρτήσεις: Τον αλγόριθμο Floyd-Steinberg και την εφαρμογή υδροδυναμικής (Livermoor Loop Kernel 23). Για κάθε μια από τις εφαρμογές χρησιμοποιήσαμε τρία διαφορετικά μεγέθη προβλήματος (χώρου επαναλήψεων), τα οποία δίνονται στον Πίνακα 6.4, για να μπορέσουμε να αναλύσουμε τις δυνατότητες κλιμάκωσης των εφαρμογών και των μηχανισμών δρομολόγησης. Σε κάθε περίπτωση, ο συγχρονισμός έγινε κατά μήκος της μεγαλύτερης διάστασης του χώρου επαναλήψεων, ενώ η δρομολόγηση κατά μήκος της δεύτερης μεγαλύτερης διάστασης. Ο χρόνος παράλληλης εκτέλεσης και των δύο εφαρμογών συγκρίνεται με τους αντίστοιχους χρόνους σειριακής εκτέλεσης.

Για την δρομολόγηση των δύο εφαρμογών χρησιμοποιήθηκαν οι αλγόριθμοι CSS, FSS, GSS, TSS και W-TSS, στους οποίους εφαρμόστηκε ο μηχανισμός συγχρονισμού $\mathcal{S}$. Οι παραπάνω αλγόριθμοι υλοποιήθηκαν σύμφωνα με τους τύπους που δίνονται στην ενότητα 2.3. Στην περίπτωση του αλγόριθμου CSS, το μέγεθος ομάδας επαναλήψεων δίνεται από τον τύπο $\frac{U_c}{2*m}$, έτσι ώστε να εξασφαλίσουμε ότι σε κάθε εργάτη θα ανατεθούν δύο ομάδες επαναλήψεων. Επίσης, εφαρμόστηκαν άνω και κάτω όρια στα μεγέθη των ομάδων επαναλήψεων (βλέπε Πίνακα 6.4) ώστε να αποφευχθούν οι μεγάλες καθυστερήσεις που προκαλούνται από της υπερβολικά μεγάλες ομάδες επαναλήψεων και το αυξημένο κόστος συγχρονισμού που προκαλείται από τις υπερβολικά μικρές ομάδες επαναλήψεων.

Το διάστημα συγχρονισμού SI δόθηκε από τον Τύπο 6.1, με $\#SPs = 3 * m$. Για να καταλήξουμε στην παραπάνω τιμή για τον αριθμό των σημείων συγχρονισμού, εκτελέσαμε εξαντλητικά πειράματα με διάφορες τιμές του $\#SPs$, στο διάστημα $1 * m$



Σχήμα 6.7: Χρόνοι παράλληλης εκτέλεσης των εφαρμογών Floyd-Steinberg και Hydro ανάλογα με τον αριθμό σημείων συγχρονισμού.

το $10 * m$ με σκοπό να εξακριβώσουμε πως αυτό επηρεάζει τον χρόνο της παράλληλης εκτέλεσης. Τα αποτελέσματα αυτών των πειραμάτων συνοψίζεται στο Σχήμα 6.7. Σύμφωνα με το σχήμα αυτό, μπορεί να επιτευχθεί ικανοποιητική απόδοση για τιμές του αριθμού σημείων συγχρονισμού $\#SPs \geq 3 * m$. Στο επόμενο κεφάλαιο, γίνεται μια εκτενής θεωρητική μελέτη για τον καθορισμό του βέλτιστου διαστήματος συγχρονισμού.

Στο Σχήμα 6.8 δίνονται τα αποτελέσματα των πειραμάτων και των δύο εφαρμογών, για όλα τα μεγέθη των χώρων επαναλήψεων. Ο παράλληλος και ο σειριακός χρόνος απεικονίζεται σε σχέση με την δύναμη του συστήματος που χρησιμοποιήθηκε. Η συνολική δύναμη έχει ως εξής: Για 4 εργάτες $VP = 3.6$, για 6 εργάτες $VP = 5.4$, για 8 εργάτες $VP = 7.2$, για 10 εργάτες $VP = 9$ και για 12 εργάτες $VP = 10.8$. Ο σειριακός χρόνος μετρήθηκε στους πιο γρήγορους τύπους μηχανημάτων, δηλαδή στα twin.

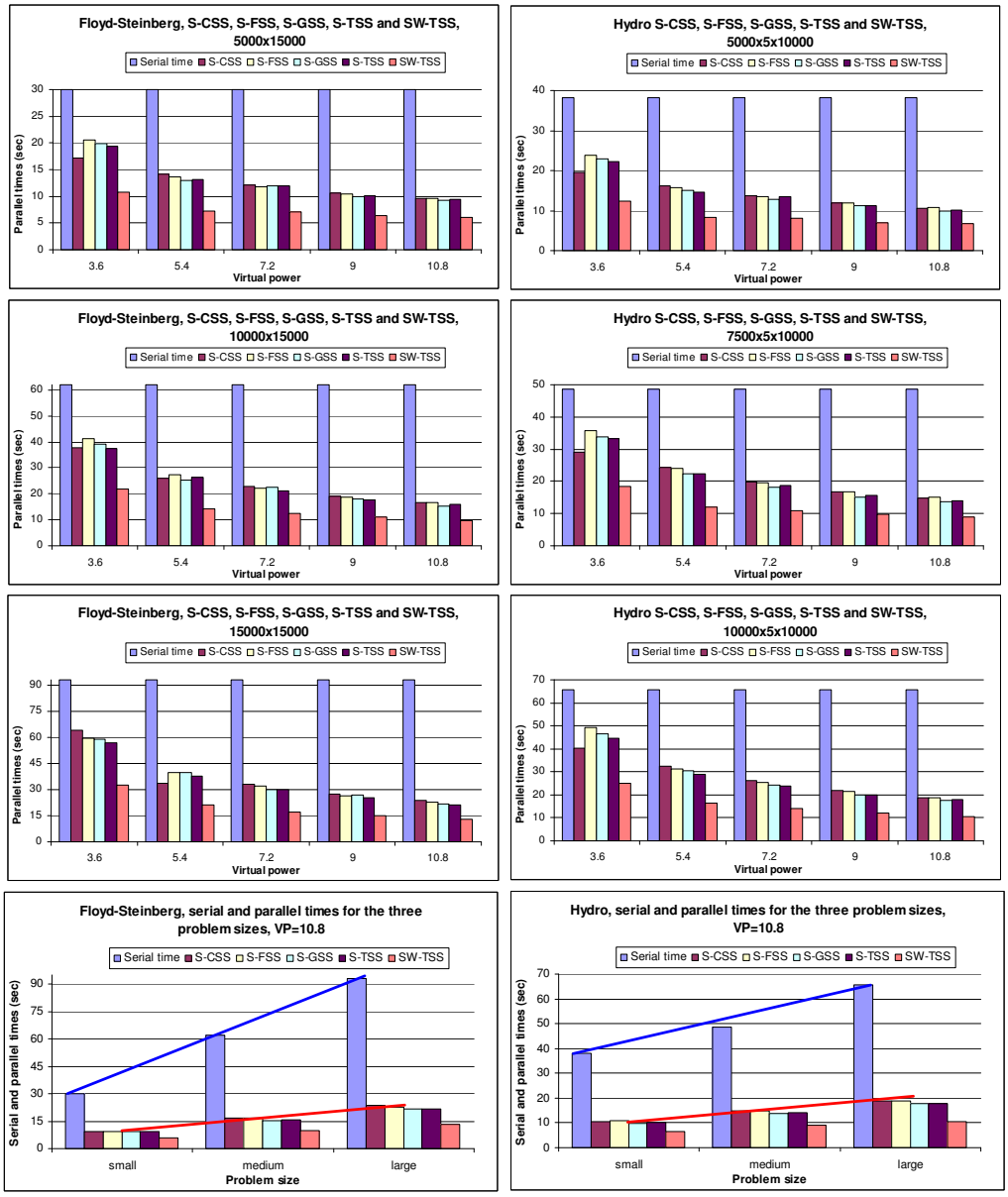
Τα αποτελέσματα αποδεικνύουν ότι ο μηχανισμός συγχρονισμού μπορεί να εφαρμοστεί σε όλους τους υπάρχοντες αυτοδρομολογούμενους αλγόριθμους, δημιουργώντας τις συγχρονισμένες εκδοχές τους, οι οποίες μπορούν να δρομολογήσουν αποδοτικά πραγματικές εφαρμογές που περιέχουν βρόχους με εξαρτήσεις. Μπορούμε να παρατηρήσουμε ότι οι αλγόριθμοι $\mathcal{S} - CSS$, $\mathcal{S} - FSS$, $\mathcal{S} - GSS$, και $\mathcal{S} - TSS$ δίνουν

Πίνακας 6.4: Χώροι επαναλήψεων για τις εφαρμογές Floyd-Steinberg και Hydro.

Μέγεθος χώρου	μικρό	μεσαίο	μεγάλο
Floyd-Steinberg	5000×15000	10000×15000	15000×15000
Upper/lower threshold	500/10	750/10	1000/10
Hydro	$5000 \times 5 \times 10000$	$7500 \times 5 \times 10000$	$10000 \times 5 \times 10000$
Upper/lower threshold	500/10	750/10	1000/10

Πίνακας 6.5: Επιτάχυνση των εφαρμογών Floyd-Steinberg και Hydro.

Εφαρμογή	VP	$\mathcal{S}-CSS$	$\mathcal{S}-FSS$	$\mathcal{S}-GSS$	$\mathcal{S}-TSS$	$\mathcal{SW}-TSS$
Floyd-Steinberg	**3.6**	1.45	1.57	1.59	1.63	2.86
	5.4	2.76	2.35	2.33	2.47	4.35
	7.2	2.81	2.92	3.09	3.10	5.39
	9	3.41	3.50	3.49	3.70	6.27
	10.8	3.95	4.07	4.27	4.34	7.09
Hydro	**3.6**	1.64	1.33	1.42	1.47	2.61
	5.4	2.02	2.10	2.16	2.28	4.04
	7.2	2.53	2.57	2.72	2.75	4.73
	9	3.01	3.07	3.33	3.29	5.43
	10.8	3.49	3.49	3.72	3.69	6.16

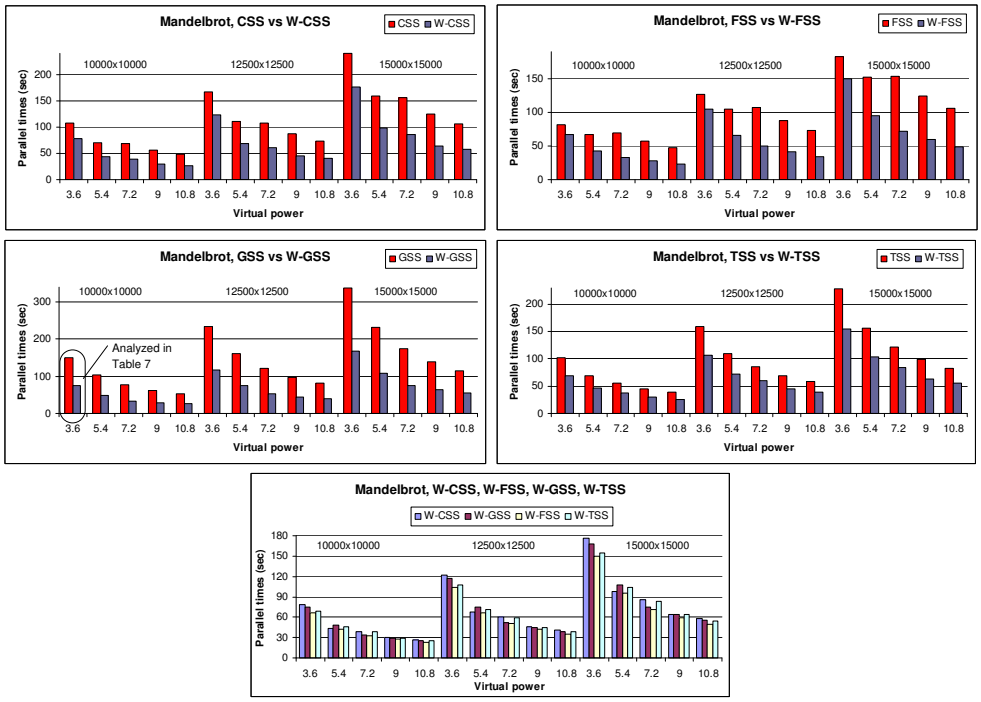


Σχήμα 6.8: Παράλληλοι χρόνοι εκτέλεσης των συγχρονισμένων αλγορίθμων των εφαρμογών Floyd-Steinberg και Hydro.

ικανοποιητική επιτάχυνση σε σχέση με την σειριακή εκτέλεση, αποδεικνύοντας έτσι την αξία των συγχρονισμένων αλγορίθμων, ωστόσο, σε κάθε περίπτωση ο αλγόριθμος $\mathcal{SW}-TSS$ υπερτερεί έναντι όλων των άλλων αλγορίθμων καθώς παρέχει πάντα μεγαλύτερη επιτάχυνση. Αυτό εξηγείται από το γεγονός ότι ο αλγόριθμος $\mathcal{SW}-TSS$ είναι ο μόνος από τους παραπάνω που προσαρμόζεται στην ανομοιογένεια του συστήματος. Τα δύο γραφήματα στο Σχήμα 6.8 απεικονίζουν τους σειριακούς και τους παράλληλους χρόνους εκτέλεσης των δύο εφαρμογών σε 12 εργάτες. Ο σειριακός χρόνος εκτέλεσης αυξάνεται με γρηγορότερο ρυθμό με την αύξηση του χώρου επαναλήψεων, από ότι ο παράλληλος χρόνος. Αυτό δείχνει ότι οι παράλληλοι αλγόριθμοι χειρίζονται ικανοποιητικά την κλιμάκωσης του μεγέθους των εφαρμογών. Όσο αυξάνεται το μέγεθος του προβλήματος, τόσοι περισσότεροι εργάτες μπορούν να χρησιμοποιηθούν αποδοτικά στην παράλληλη εκτέλεση.

Δεύτερη σειρά πειραμάτων. Στην δεύτερη σειρά πειραμάτων, το παράδειγμα που χρησιμοποιήθηκε είναι ο υπολογισμός των σημείων του συνόλου Mandelbrot. Όπως είπαμε νωρίτερα, ο υπολογισμός αυτός αποτελείται από ένα παράλληλο βρόχο με ακανόνιστο υπολογιστικό φόρτο, οι διαστάσεις του οποίου κατά την διάρκεια των πειραμάτων ήταν 7500×10000, 10000×10000 και 12500×12500 επαναλήψεις στον μιγαδικό χώρο $\mathcal{C} = \{\alpha + \beta i : -2.0 \leq \alpha \leq 1.25, -1.25 \leq \beta \leq 1.25\}$. Δρομολογούμε τον υπολογισμό με τους αλγόριθμους CSS, FSS, GSS, TSS και συγκρίνουμε την απόδοση τους, σε όρους παράλληλου χρόνου, με αυτή των σταθμισμένων εκδοχών τους $\mathcal{W}-CSS$, $\mathcal{W}-FSS$, $\mathcal{W}-GSS$ και $\mathcal{W}-TSS$. Σε αυτή τη σειρά πειραμάτων, περιμένουμε ότι ο μηχανισμός σταθμίσματος θα διευκολύνει τους αλγόριθμους δρομολόγησης να κατανείμουν καλύτερα το υπολογιστικό φόρτο, βελτιώνοντας με αυτό τον τρόπο την συνολική απόδοση. Οι παραπάνω εκτίμηση επιβεβαιώθηκε από τα πειραματικά αποτελέσματα τα οποία δίνονται στο Σχήμα 6.9, όπου μπορούμε να δούμε ότι οι σταθμισμένοι αλγόριθμοι υπερτερούν πάντα σε απόδοση έναντι των μη-σταθμισμένων, αρχικών αλγορίθμων.

Η **βελτίωση** της απόδοσης των σταθμισμένων έναντι των αρχικών αλγορίθμων δίνεται και στον Πίνακα 6.6. Σε αυτό τον πίνακα η βελτίωση υπολογίζεται ως $\frac{T_{\mathcal{A}}-T_{\mathcal{W}-\mathcal{A}}}{T_{\mathcal{A}}}$, όπου $T_{\mathcal{A}}$ είναι ο παράλληλος χρόνος εκτέλεσης του αρχικού, μη-σταθμισμένου αλγορίθμου, ενώ $\mathcal{W}$-$\mathcal{A}$ είναι ο παράλληλος χρόνος της αντίστοιχης σταθμισμένης εκδοχής του. Μας ενδιαφέρει να εξακριβώσουμε το διάστημα εμπιστοσύνης των βελτιώσεων για όλες τις παραπάνω περιπτώσεις. Όπως συνηθίζεται, υπολογίσαμε το 95% διάστημα εμπιστοσύνης. Ο αλγόριθμος με τον ελάχιστο παράλληλο χρόνο εκτέλεσης ήταν ο $\mathcal{SW}-FSS$ για όλους τους χώρους επαναλήψεων. Η βελτίωση της απόδοσης του $\mathcal{SW}-FSS$ σε σχέση με τον FSS κυμαίνεται στο διάστημα 18% με 53%. Με πιθανότητα 0.95, η βελτίωση της απόδοσης βρίσκεται στο διάστημα $42 \pm 8\%$. Ωστόσο, ο αλγόριθμος που ωφελήθηκε περισσότερο από την εφαρμογή του μηχανισμού $\mathcal{W}$, ήταν ο GSS, με ποσοστό βελτίωσης που κυμαίνεται από 50% ως 57%. Με πιθανότατα 0.95, η βελτίωση στην περίπτωση αυτή βρίσκεται στο διάστημα $53 \pm 6\%$. Όπως δίνεται στο Σχήμα 6.9, η διαφορά στην απόδοση μεταξύ των σταθμισμένων αλγορίθμων είναι σημαντικά μικρότερη από αυτή των αρχικών αλγορίθμων.



Σχήμα 6.9: Παράλληλοι χρόνοι των αρχικών και των σταθμισμένων αλγορίθμων για την εφαρμογή Mandelbrot.

Ο Πίνακας 6.7 δίνει τον συνολικό χρόνο υπολογισμού και τον συνολικό αριθμό επαναλήψεων που ανατέθηκε κάθε εργάτη ξεχωριστά (για $m = 4$) κατά την διάρκεια της παράλληλης εκτέλεσης του υπολογισμού Mandelbrot για την περίπτωση των αλγορίθμων GSS και WGSS. Ο στόχος του πίνακα αυτού είναι να επιβεβαιώσουμε ότι ο μηχανισμός $\mathcal{W}$ βελτιώνει την απόδοση μέσω της καλύτερης κατανομής υπολογιστικού φόρτου στους διαθέσιμους εργάτες. Ο συνολικός χρόνος υπολογισμού κάθε εργάτη είναι ο χρόνος τον οποίο ο εργάτης ξόδεψε εκτελώντας χρήσιμη εργασία. Στην ιδανική περίπτωση όλοι οι εργάτες θα πρέπει να έχουν ακριβός τον ίδιο χρόνο υπολογισμού, ενώ μεγάλες διαφοροποιήσεις υποδεικνύουν μεγάλη ανισορροπία στην κατανομή του υπολογιστικού φόρτου.

Τα δεδομένα στον Πίνακα 6.7 αντιστοιχούν στο Σχήμα 6.9, GSS έναντι $\mathcal{W}-GSS$ και στην περίπτωση των τεσσάρων εργατών, με συνολική $VP = 3.6$ και δίνουν την διαφορά μεταξύ την σταθμισμένης και αρχικής εκδοχής του αλγόριθμου. Μπορούμε να δούμε ότι ο συνολικός παράλληλος χρόνος εκτέλεσης στο Σχήμα 6.9 είναι πολύ κοντά στον μέγιστο χρόνο υπολογισμού των εργατών. Είναι προφανές ότι στην αρχική, μη-σταθμισμένη εκδοχή του GSS, ο χρόνος υπολογισμού των εργατών παρουσιάζει μια διακύμανση ανάλογη με την υπολογιστική δύναμη και το εξωτερικό φόρτο του εργάτη, δηλαδή αργοί και φορτωμένοι εργάτες παρουσιάζουν μεγαλύτερους χρόνους υπολογισμού. Στην περίπτωση των σταθμισμένων αλγορίθμων, η διακύμανση αυτή μειώνεται σημαντικά. Πιο συγκεκριμένα, στην περίπτωση του GSS αν και οι εργάτες

Πίνακας 6.6: Βελτίωση των σταθμισμένων έναντι των μη σταθμισμένων αλγορίθμων για την εφαρμογή Mandelbrot.

Εφαρμογή	Μέγεθος χώρου επαναλήψεων	VP	$\mathcal{S}-CSS$ *vs* $\mathcal{SW}-CSS$	$\mathcal{S}-GSS$ *vs* $\mathcal{SW}-GSS$	$\mathcal{S}-FSS$ *vs* $\mathcal{SW}-FSS$	$\mathcal{S}-TSS$ *vs* $\mathcal{SW}-TSS$
Mandelbrot	10000×10000	**3.6**	27%	50%	18%	33%
		5.4	38%	54%	37%	34%
		7.2	43%	57%	52%	32%
		9	48%	53%	52%	35%
		10.8	43%	52%	52%	34%
	12500×12500	**3.6**	27%	50%	18%	33%
		5.4	38%	54%	37%	34%
		7.2	44%	57%	53%	30%
		9	47%	54%	52%	35%
		10.8	44%	52%	53%	34%
	15000×15000	**3.6**	27%	50%	18%	33%
		5.4	38%	54%	37%	34%
		7.2	45%	57%	53%	31%
		9	49%	54%	52%	35%
		10.8	46%	52%	54%	33%
Διάστημα εμπιστοσύνης (95%)	συνολικό 42 ± 3 %		40 ± 6 %	53 ± 6 %	42 ± 8 %	33 ± 4 %

Πίνακας 6.7: Κατανομή φόρτου εκφραζόμενη σε συνολικό αριθμό επαναλήψεων ανά εργάτη χρόνο υπολογισμού ανά εργάτη, GSS έναντι $\mathcal{W}-GSS$ (Mandelbrot test case).

Εργάτης	GSS # Επαναλήψεων (10^6)	GSS Χρόνος υπολογισμού (sec)	$\mathcal{W}-GSS$ # Επαναλήψεων (10^6)	$\mathcal{W}-GSS$ Χρόνος υπολογισμού (sec)
$twin2$	56.434	34.63	55.494	62.54
$kid1$	18.738	138.40	15.528	62.12
$twin3$	10.528	39.37	15.178	74.63
$kid2$	14.048	150.23	13.448	61.92

kid2 και twin2 υπολόγισαν 14.048×10^6 και 56.434×10^6 επαναλήψεις αντίστοιχα, ο twin2 χρειάσθηκε 34.63 sec, ενώ ο kid2 150.23 sec, ο οποίος είναι περίπου ο συνολικός παράλληλος χρόνος. Αυτή η άνιση κατανομή υπολογιστικού φόρτου είναι υπεύθυνη για την κακή απόδοση του αλγορίθμου δρομολόγησης. Αντιθέτως, στην περίπτωση του $\mathcal{W}-GSS$ οι χρόνοι υπολογισμού είναι αρκετά ισορροπημένοι. Ασφαλώς, αποκλίσεις στους χρόνους υπολογισμού οφείλονται στην ακανόνιστη φύση της υπό εξέτασης εφαρμογής.

Τρίτη σειρά πειραμάτων. Για την τρίτη σειρά πειραμάτων, επαναλάβαμε την πρώτη σειρά, εφαρμόζοντας αυτή την φορά και τον μηχανισμό $\mathcal{W}$ στους συγχρονισμένους αλγόριθμους, δημιουργώντας έτσι τους αλγόριθμους $\mathcal{SW}-CSS$, $\mathcal{SW}-FSS$, $\mathcal{SW}-GSS$ και $\mathcal{SW}-TSS$. Οι εφαρμογές που εκτελέσθηκαν ήταν πάλι ο αλγόριθμος Floyd-Steinberg και η εφαρμογή υδροδυναμικής. Τα αποτελέσματα που παρουσιάζονται στα Σχήματα 6.10 και 6.11 δείχνουν ότι σε όλες τις περιπτώσεις οι συγχρονισμένοι και σταθμισμένοι αλγόριθμοι παρέχουν καλύτερη απόδοση, δηλαδή μικρότερους παράλληλους χρόνους εκτέλεσης από τους συγχρονισμένουσ-μόνο αλγόριθμους. Μπορεί επίσης να διαπιστωθεί ότι όλοι οι $\mathcal{SW}-A$ αλγόριθμοι δίνουν συγκρίσιμους παράλληλους χρόνους.

Τα αποτελέσματα αυτά δίνονται επίσης στον Πίνακα 6.8, ο οποίος δίνει την βελτίωση των αλγορίθμων $\mathcal{SW}$-$\mathcal{A}$ έναντι των $\mathcal{S}$-$\mathcal{A}$, που υπολογίζεται ως $\frac{T_{\mathcal{S}-\mathcal{A}}-T_{\mathcal{SW}-\mathcal{A}}}{T_{\mathcal{S}-\mathcal{A}}}$, όπου $T_{\mathcal{S}-\mathcal{A}}$ είναι ο παράλληλος χρόνος εκτέλεσης του συγχρονισμένου-μόνο αλγορίθμου $\mathcal{A}$, ενώ $T_{\mathcal{SW}-\mathcal{A}}$ είναι ο παράλληλος χρόνος εκτέλεσης του αντίστοιχου συγχονισμένου και σταθμισμένου αλγόριθμου. Στον πίνακα αυτό δίνονται επίσης τα διαστήματα εμπιστοσύνης κάθε αλγορίθμου χωριστά αλλά και συνολικά, όπως και στην προηγούμενη υποένοτητα.

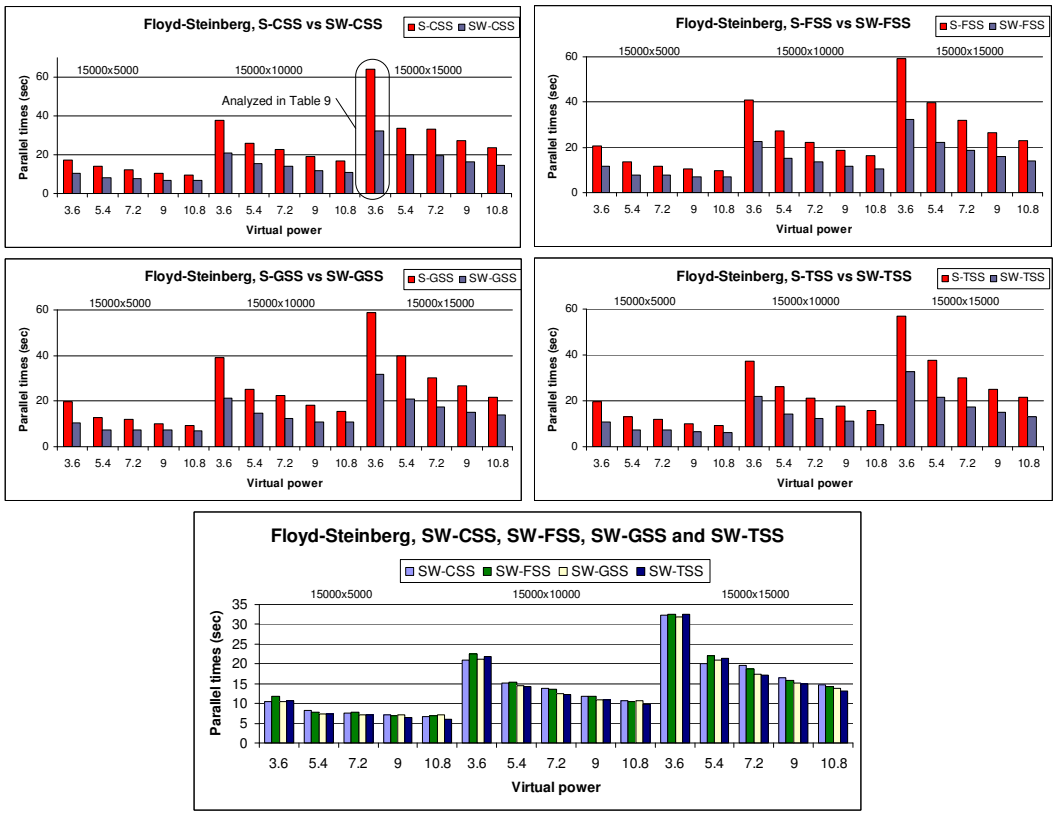
Στον Πίνακα 6.8 δίνονται οι περιπτώσεις των αλγορίθμων με την μεγαλύτερη βελτίωση για κάθε εφαρμογή. Στην συνέχεια, στον Πίνακα 6.9 αναλύουμε την βελτίωση των $\mathcal{SW}-CSS$ έναντι του $\mathcal{S}-CSS$ για τον υπολογισμό του Floyd-Steinberg και την βελτίωση του $\mathcal{SW}-GSS$ έναντι του $\mathcal{S}-GSS$ για την εφαρμογή της υδροδυναμικής. Και σε αυτές τις δύο περιπτώσεις η βελτίωση της απόδοσης οφείλεται στην καλύτερη κατανομή του υπολογιστικού φόρτου από τους σταθμισμένους αλγόριθμους. Πιο συγκεκριμένα, οι αλγόριθμοι $\mathcal{S}-CSS$ και $\mathcal{S}-GSS$ αποδίδουν περίπου τον ίδιο αριθμό επαναλήψεων σε όλους τους εργάτες που συμμετέχουν στην παράλληλη εκτέλεση, και αυτό προκαλεί ανισότητα στους συνολικούς χρόνους υπολογισμού ανά εργάτη, αφού υπάρχουν μεγάλες διαφορές στις υπολογιστικές τους δυνατότητες. Με τους αλγόριθμους $\mathcal{SW}-CSS$ και $\mathcal{SW}-GSS$, βλέπουμε τις διαφορές στους χρόνους υπολογισμού να μειώνονται στο ελάχιστο γιατί ο αριθμός των επαναλήψεων που δίνεται σε κάθε εργάτη προσαρμόζεται ανάλογα με την υπολογιστική του ικανότητα.

6.6 Συμπεράσματα

Αυτό το κεφάλαιο ασχολείται με την εφαρμογή γενικών μηχανισμών σε δυναμικούς αλγόριθμους δρομολόγησης. Ο στόχος των μηχανισμών είναι η εφαρμογή των δυναμικών αλγορίθμων σε βρόχους με εξαρτήσεις που πριν δεν ήταν δυνατό. Στοχεύουν

Πίνακας 6.8: Βελτίωση των συγχρονισμένων-στατθμισμένων έναντι των συγχρονισμένων-μόνο αλγορίθμων για τις εφαρμογές Floyd-Steinberg και Hydro.

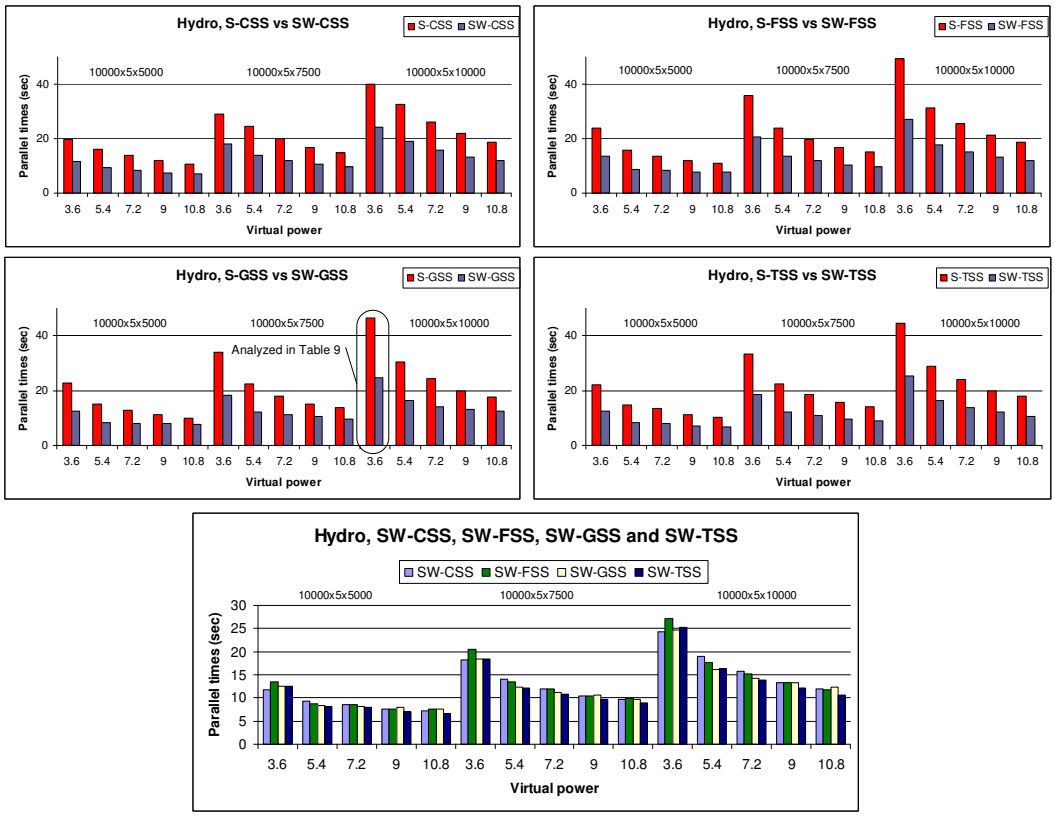
Εφαρμογή	Μέγεθος χώρου επαναλήψεων	VP	$\mathcal{S}-CSS$ *vs* $\mathcal{SW}-CSS$	$\mathcal{S}-GSS$ *vs* $\mathcal{SW}-GSS$	$\mathcal{S}-FSS$ *vs* $\mathcal{SW}-FSS$	$\mathcal{S}-TSS$ *vs* $\mathcal{SW}-TSS$
Floyd-Steinberg	15000 × 5000	**3.6**	39%	47%	43%	45%
		5.4	42%	43%	44%	44%
		7.2	37%	40%	35%	40%
		9	34%	27%	34%	36%
		10.8	31%	23%	28%	35%
	15000 × 7500	**3.6**	44%	46%	45%	42%
		5.4	41%	42%	44%	45%
		7.2	39%	45%	38%	42%
		9	37%	40%	37%	38%
		10.8	36%	30%	36%	38%
	15000 × 10000	**3.6**	**50%**	46%	45%	43%
		5.4	41%	48%	44%	43%
		7.2	41%	42%	41%	42%
		9	39%	43%	40%	41%
		10.8	38%	36%	38%	39%
Διάστημα εμπιστοσύνης (95%)	συνολικό 40 ± 1 %		39 ± 2 %	40 ± 3 %	40 ± 2 %	41 ± 2 %
Hydro	10000 × 5000 ×5	**3.6**	40%	46%	43%	44%
		5.4	43%	44%	44%	44%
		7.2	39%	37%	37%	41%
		9	37%	29%	36%	38%
		10.8	32%	23%	29%	34%
	10000 × 7500 ×5	**3.6**	38%	46%	43%	44%
		5.4	43%	45%	44%	46%
		7.2	40%	37%	39%	42%
		9	37%	29%	38%	38%
		10.8	35%	30%	35%	36%
	10000 × 10000 ×5	**3.6**	40%	**47%**	45%	44%
		5.4	42%	47%	44%	43%
		7.2	40%	41%	41%	42%
		9	39%	33%	38%	39%
		10.8	37%	30%	37%	40%
Διάστημα εμπιστοσύνης (95%)	συνολικό 39 ± 1 %		39 ± 2 %	38 ± 4 %	40 ± 2 %	41 ± 2 %



Σχήμα 6.10: Παράλληλοι χρόνοι των σταθμισμένων-συχρονισμένων και των συγχρονισμένων-μόνο αλγορίθμων για την εφαρμογή Floyd-Steinberg, για χώρους επαναλήψεων: 15000×5000, 15000×10000 και 15000×15000 σημείων.

επίσης και στην βελτίωση της απόδοσης τους σε ετερογενή και μη-αφοσιωμένα περιβάλλοντα. Το μοντέλο επεξεργασίας είναι αυτό του συντονιστή-εργάτη, το οποίο τροποποιείται καταλλήλως με την εφαρμογή του μηχανισμού συντονισμού που επιτρέπει την ανταλλαγή δεδομένων, η οποία είναι απαραίτητη σε προβλήματα που περιέχουν εξαρτήσεις. Στην συνέχεια προτείνεται ο μηχανισμός στάθμισης, που βελτιώνει την εξισορρόπηση της κατανομής φορτίου στην περίπτωση των ετερογενών, μη αφοσιωμένων συστημάτων Οι μηχανισμοί αυτοί είναι ανεξάρτητοι και μπορούν να εφαρμοστούν ταυτόχρονα σε οποιοδήποτε δυναμικό αλγόριθμο.

Η εφαρμοσιμότητα και η αποδοτικότητα των δύο μηχανισμών αποδείχτηκε πειραματικά και με την χρήση τριών πραγματικών βρόχων εφαρμογών, με ή και χωρίς εξαρτήσεις. Αν και οι εξαρτήσεις που περιείχαν οι εφαρμογές ήταν ομοιόμορφες και σταθερές τα αποτελέσματα που παρουσιάζονται θα μπορούσαν να γενικευθούν και σε βρόχους με μη ομοιόμορφες εξαρτήσεις.



Σχήμα 6.11: Παράλληλοι χρόνοι των σταθμισμένων-συχρονισμένων και των συγχρονισμένων-μόνο αλγορίθμων για την εφαρμογή Hydro, για χώρους επαναλήψεων: $10000 \times 5 \times 5000$, $10000 \times 5 \times 7500$ ανδ $10000 \times 5 \times 10000$ σημείων.

Πίνακας 6.9: Κατανομή φόρτου εκφραζόμενη σε συνολικό αριθμό επαναλήψεων ανά εργάτη χρόνο υπολογισμού ανά εργάτη, $\mathcal{S}-FSS$ έναντι $\mathcal{SW}-FSS$.

Εφαρμογή	Εργάτης	Αριθμός επαναλήψεων (10^6)	Χρόνος υπολογισμού (sec)	Αριθμός επαναλήψεων (10^6)	Χρόνος υπολογισμού (sec)
		$\mathcal{S}-CSS$	$\mathcal{S}-CSS$	$\mathcal{SW}-CSS$	$\mathcal{SW}-CSS$
	$twin2$	59.93	19.25	89.90	28.88
Floyd-	$kid1$	59.93	62.22	29.92	30.86
Steinberg	$twin3$	59.93	19.24	74.92	24.06
	$kid2$	44.95	46.30	29.92	29.08
		$\mathcal{S}-GSS$	$\mathcal{S}-GSS$	$\mathcal{SW}-GSS$	$\mathcal{SW}-GSS$
	$twin2$	84.50	15.32	117.94	21.39
Hydro	$kid1$	78.38	42.60	38.03	22.49
	$twin3$	62.69	17.44	106.48	20.75
	$kid2$	73.58	33.72	36.41	19.46

Κεφάλαιο 7

Βέλτιστη συχνότητα συγχρονισμού δυναμικών αλγορίθμων

Οι αυτοδρομολογούμενοι αλγόριθμοι μπορούν πια να χειριστούν βρόχους με ή χωρίς εξαρτήσεις σε ετερογενείς συστοιχίες υπολογιστών. Η ύπαρξη εξαρτήσεων κάνει απαραίτητη την επικοινωνία μεταξύ των επεξεργαστών και αυτό επιτυγχάνεται με την εφαρμογή του μηχανισμού συγχρονισμού που περιγράφηκε στην προηγούμενη ενότητα. Η συχνότητα του συγχρονισμού πρέπει να επιλεχθεί σωστά έτσι ώστε να ελαχιστοποιηθεί το κόστος επικοινωνίας, διατηρώντας ταυτόχρονα υψηλά επίπεδα παραλληλισμού. Στα προηγούμενα κεφάλαια, η συχνότητα συγχρονισμού επιλεγόταν εμπειρικά, σε αυτό το κεφάλαιο δίνεται ένα θεωρητικό μοντέλο για τον καθορισμό της συχνότητας που οδηγεί στον ελάχιστο παράλληλο χρόνο για την κατηγορία των αυτοδρομολογούμενων αλγορίθμων. Το μοντέλο αυτό προσεγγίζει τον παράλληλο χρόνο εκτέλεσης εφαρμογών που περιέχουν βρόχους με εξαρτήσεις που χρονοδρομολογούνται σε ετερογενή συστήματα. Χρησιμοποιώντας αυτό το μοντέλο μπορούμε να καθορίσουμε την συχνότητα συγχρονισμού που ελαχιστοποιεί τον εκτιμώμενο παράλληλο χρόνο. Η ακρίβεια του αλγορίθμου αποδεικνύεται από πειραματικά αποτελέσματα στα οποία η συχνότητα συγχρονισμού, που προέκυψε από τον αλγόριθμο, είναι πολύ κοντά σε αυτή που βρέθηκε πειραματικά.

7.1 Εισαγωγή

Στην βιβλιογραφία υπάρχει σημαντικός όγκος δουλειάς για τον καθορισμό του βέλτιστου μεγέθους ομάδας επαναλήψεων (tile size, block size, grain size) κατά την δρομολόγηση φωλιασμένων βρόχων σε ομοιογενή κατανεμημένα συστήματα ([30], [76], [50], [49], [65] και στις αναφορές που περιέχουν).Οι Ponnusamy et al [77] χρησιμοποίησαν ένα μοντέλο επιθεωρητή/εκτελεστή (inspector/executor model) για να προβλέψουν τις επικοινωνιακές απαιτήσεις του κώδικα της εφαρμογής και να βελτιστοποιήσουν

τις επικοινωνίες κατά την διάρκεια του χρόνου εκτέλεσης για να αντιμετωπίσουν ακανόνιστα προβλήματα (irregular problems). Οι Huang et al [88] παρουσίασαν ένα χρονοδρομολογητή που χωρίζει ένα μερικά παράλληλο βρόχο σε ανεξάρτητα μέτωπα (wavefronts) και στην συνέχεια εκτελεί τις επαναλήψεις σε κάθε μέτωπο παράλληλα. Οι Lowenthal et al [24] προτείνουν μια μέθοδο για την ακριβή επιλογή του μεγέθους ομάδας κατά τον χρόνο εκτέλεσης σε προγράμματα που εκτελούνται σε σωληνώσεις (pipelined computations). Στο [24] οι υπολογισμοί που αναθέτονται σε κάθε επεξεργαστή εκτελούνται κατά ομάδες. Το μέγεθος της κάθε ομάδας επιλέγεται κατά την διάρκεια του χρόνου εκτέλεσης ανάλογα με τις μεταβολές που παρατηρούνται στον φόρτο εργασίας, λόγο της ακανόνιστης φύσης του βρόχου της εφαρμογής.

Το πρόβλημα της αναζήτησης του βέλτιστου μεγέθους ομάδας σε ετερογενή κατανεμημένα συστήματα δεν έχει μελετηθεί στον ίδιο βαθμό. Οι Boulet et al [71] ήταν οι πρώτοι που εφάρμοσαν την μέθοδο tiling σε ετερογενή συστήματα. Στην δουλεία τους χωρίζουν πλήρως μεταθέσιμους βρόχους σε ομάδες επαναλήψεων ίδιου μεγέθους. Στην συνέχεια συνθέτουν από αυτές τις ομάδες μεγαλύτερες ομάδες που ανταποκρίνονται στις δυνάμεις των διαθέσιμων επεξεργαστών, δίνοντας περισσότερη δουλειά στους γρηγορότερους επεξεργαστές. Η μέθοδος τους χρησιμοποιεί έναν αλγόριθμο που διαλέγει την καλύτερη δυνατή ανάθεση ομάδων βασιζόμενος σε μια συνάρτηση κόστους. Επιλέξανε να αποφύγουν την δημιουργία ομάδων μεταβλητού μεγέθους για να απλοποιήσουν στην δημιουργία κώδικα. Οι Chen και Xue [81] αναθέτουν στατικά ομάδες επαναλήψεων διαφορετικού μεγέθους στους διαθέσιμους επεξεργαστές βασιζόμενοι σε μια συνάρτηση που ελαχιστοποιεί τον παράλληλο χρόνο σύμφωνα με ένα μοντέλο κόστους υπολογισμών και ένα μοντέλο κόστους επικοινωνιών. Στο μοντέλο υπολογισμών, οι ικανότητα υπολογισμών των επεξεργαστών μοντελοποιείται με βάση την ταχύτητα τους και το μέγεθος μνήμης ενώ το μοντέλο επικοινωνιών περιλαμβάνει το κόστος αποστολής και λήψης καθώς επίσης και το επίπεδο συμφόρησης του δικτύου.

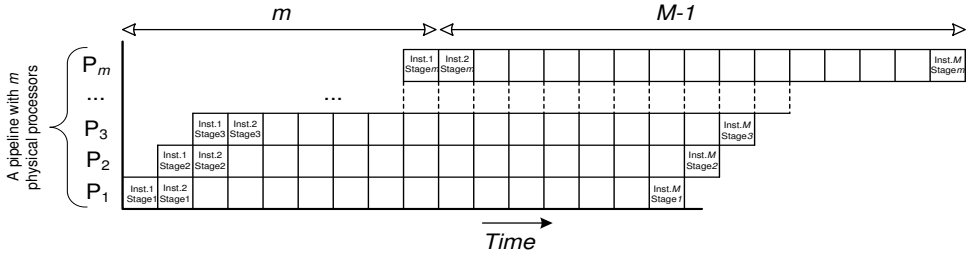
Καμία από τις παραπάνω εργασίες δεν αφορά την περίπτωση του δυναμικού υπολογισμού και ανάθεσης ομάδων επαναλήψεων στους διαθέσιμους επεξεργαστές. Στους αλγόριθμους που παρουσιάστηκαν στα [31] και [102] οι αυτοδρομολογούμενοι αλγόριθμοι επεκτάθηκαν μέσω του μηχανισμού συγχρονισμού για εφαρμογή σε βρόχους με εξαρτήσεις, ωστόσο, η επιλογή της κατάλληλη συχνότητας συγχρονισμού δεν είχε μελετηθεί σε βάθος. Σε αυτό το κεφάλαιο παρουσιάζεται μια μέθοδος που επιτρέπει την εξεύρεση της συχνότητας συγχρονισμού η οποία ελαχιστοποιεί τον χρόνο της παράλληλης εκτέλεσης των παραπάνω αυτοδρομολογούμενων αλγορίθμων σε ετερογενή και μη-αφοσιωμένα κατανεμημένα συστήματα.

7.2 Εξαρτήσεις και υπολογισμοί σωληνώσεων

Όπως είδαμε στους αλγορίθμους που παρουσιάζοντα στα προηγούμενα κεφάλαια η ύπαρξη των σημείων συγχρονισμού εξαναγκάζει τη παράλληλη εκτέλεση προχωρήσει σε διακριτά βήματα, υπό την μορφή εκτέλεσης σωλήνωσης [31, 8] (pipelined computations).

Γενικά, θεωρούμε ότι κάθε γύρος αναθέσεων ομάδων επαναλήψεων στους εργάτες

αντιστοιχεί σε μια ξεχωριστή σωλήνωση με αριθμό σταδίων (stages) ίσο με τον αριθμό των διαθέσιμων εργατών m (βλέπε Σχήμα 9.1). Ο αριθμός των γύρων αναθέσεων, αντιστοιχεί στον αριθμό των σωληνώσεων p.



Σχήμα 7.1: Χωρικό διάγραμμα μιας σωλήνωσης χονδρού-κόκκου με m στάδια, M στιγμιότυπα και $m + (M - 1)$ βήματα.

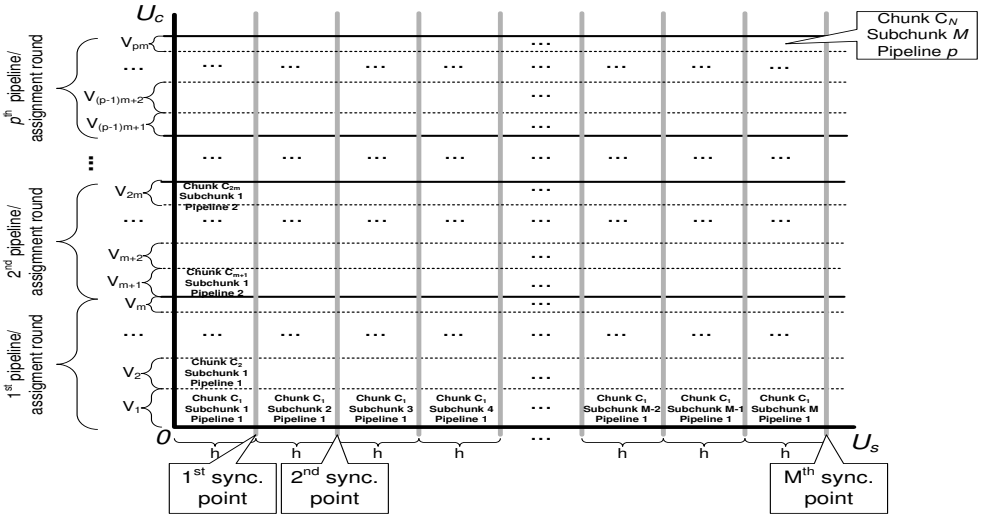
Μέσα σε μια σωλήνωση κάθε εργάτης συγχρονίζεται με τους γειτονικούς του εργάτες στα προκαθορισμένα σημεία συγχρονισμού. Υπάρχουν συνολικά M σημεία συγχρονισμού, ομοιόμορφα κατανεμημένα κατά μήκος της διάστασης συγχρονισμού U_s, η οποία είναι η και πιο μεγάλη από τις n διαστάσεις του φωλιασμένου βρόχου (βλέπε Σχήμα7.2). Μεταξύ δυο σημείων συγχρονισμού περιέχονται h επαναλήψεις στην διάσταση συγχρονισμού U_s.

Για την ολοκλήρωση της εκτέλεσης κάθε σωλήνωσης απαιτούνται $m + (M - 1)$ βήματα όπως φαίνεται στο Σχήμα 9.1, όπου m είναι ο αριθμός των εργατών που αντιστοιχεί στην αρχική καθυστέρηση εκτέλεσης της σωλήνωσης (αριθμός βημάτων που απαιτείται για να γεμίσει η σωλήνωση). Το M αντιστοιχεί στον αριθμό των στιγμιότυπων της σωλήνωσης, δηλαδή στον αριθμό των βημάτων από την στιγμή που θα γεμίσει η σωλήνωση μέχρι το τέλος της εκτέλεσης. Τα δεδομένα που παράγονται στο τέλος ενός σταδίου της σωλήνωσης τροφοδοτούνται στο επόμενο στάδιο. Στο Σχήμα 9.1, κάθε ομάδα υπολογισμών που ανατίθεται σε ένα εργάτη αποτελεί ένα ξεχωριστό στάδιο της σωλήνωσης, ενώ ένα σημείο συγχρονισμού διαχωρίζει δυο διαφορετικά στιγμιότυπα της σωλήνωσης.

Το μήκος των διαστημάτων συγχρονισμού καθορίζει το πλήθος των υπολογισμών πριν την αποστολή ενός μηνύματος. Είναι προφανές ότι η επιλογή του διαστήματος συγχρονισμού παίζει σημαντικό ρόλο στον συνολικό παράλληλο χρόνο. Μικρό διάστημα σημαίνει συχνούς συγχρονισμούς και άρα πυκνές επικοινωνίες, ένα μεγάλο διάστημα σημαίνει μικρή συχνότητα συγχρονισμών, γεγονός που μπορεί να περιορίσει τον διαθέσιμο παραλληλισμό.

7.2.1 Μοντέλα κόστους υπολογισμών και επικοινωνιών

Στην προσέγγιση που ακολουθούμε σε αυτό το κεφάλαιο, για να εφαρμόσουμε ένα αυτοδρομολογούμενο αλγόριθμο σε ένα ετερογενές και μη αφοσιωμένο σύστημα αρχίζουμε με ένα μετασχηματισμό που μετατρέπει τους m πραγματικούς επεξεργαστές σε ένα πίνακα από τους αντίστοιχους A ομοιογενείς και αφοσιωμένους εικονικούς επεξεργαστές, όπως παρουσιάστηκε αρχικά από τον αλγόριθμο DTSS (βλέπε ενότητα



Σχήμα 7.2: Καταμερισμός ενός χώρου 2 διαστάσεων σε ομάδες υπολογισμών και τοποθέτηση σημείων συγχρονισμού.

5.1.1). Κάθε ένας από αυτούς τους εικονικούς επεξεργαστές έχει μοναδιαία διαθέσιμη υπολογιστική ισχύ. Ένας πραγματικός εργάτης k μοντελοποιείται σαν ένα σύνολο από A_k εικονικούς εργάτες με διαθέσιμη υπολογιστική ισχύ 1, όπου A_k είναι και η διαθέσιμη υπολογιστική ισχύ του πραγματικού εργάτη k, με $k = 1, \ldots, m$. Ο συνολικός αριθμός των εικονικών επεξεργαστών είναι $A = \sum_{k=1}^{m} A_k$. Ο λόγος $\frac{m}{A}$ είναι μια ένδειξη για την ανομοιογένεια του συστήματος.

Μετά τον μετασχηματισμό, η κάθε σωλήνωση αποτελείται από A (αντί για m στάδια) και από M στιγμιότυπα. Στην συνέχεια της ενότητας, όλοι οι αυτοδρομολογούμενοι αλγόριθμοι εφαρμόζονται στους A εικονικούς εργάτες και καλούνται ως DCSS, DGSS, DTSS, DFSS. Ο μετασχηματισμός αυτός βελτιώνει την απόδοση των αυτοδρομολογούμεων αλγορίθμων αφού ένας εργάτης με μεγάλη υπολογιστική δύναμη θα χωριστεί σε πολλούς εικονικούς εργάτες και άρα θα αναλάβει μεγάλο μέρος της δουλειάς.

Μοντέλο κόστους υπολογισμών: Το κόστος υπολογισμών είναι μια γραμμική συνάρτηση του αριθμού των επαναλήψεων επί του κόστους υπολογισμού της κάθε επανάληψης. Θεωρούμε ότι ο κάθε εικονικός εργάτης έχει υπολογιστική ισχύ ίση με την υπολογιστική ισχύ του πιο αργού φυσικού εργάτη του συστήματος. Το κόστος υπολογισμού μιας υποομάδας επαναλήψεων είναι:

$$t_p = hV_i c_p \tag{7.1}$$

όπου V_i είναι το μέγεθος της ομάδας επαναλήψεων στην διάσταση δρομολόγησης U_s και c_p είναι ο χρόνος που απαιτείται από τον εικονικό εργάτη για την εκτέλεση μίας επανάληψης. Προφανώς το παραπάνω ισχύει για βρόχους δυο διαστάσεων, σε άλλη περίπτωση το t_p θα πρέπει να πολλαπλασιαστεί και με το μέγεθος των υπόλοιπων διαστάσεων. Στη συνέχεια η ανάλυση θα περιοριστεί σε βρόχους δυο διαστάσεων.

Μοντέλο κόστους επικοινωνιών: Υποθέτουμε ότι το κόστος αποστολής ενός μηνύματος (t_s) είναι ίδιο με το κόστος λήψης ενός μηνύματος (t_r). Η ακρίβεια αυτής της απόδοσης δεν μας απασχολεί ιδιαίτερα αφού οι επικοινωνίες συνήθως εκτελούνται σε ζεύγη αποστολών-λήψεων, και άρα μας απασχολεί περισσότερο ο συνολικός χρόνος αποστολής-λήψης. Το κόστος για την ανταλλαγή ενός μηνύματος μεγέθους h (Σχήμα 7.2, 7.3 και 7.4) μεταξύ δυο εργατών είναι:

$$t_c = c_d + hc_c \tag{7.2}$$

Όπου c_d είναι το κόστος αρχικοποίησης (ο χρόνος για την αποστολή ενός μηνύματος μηδενικού μεγέθους) και c_c είναι η ρυθμοαπόδοση (throughput) του δικτύου, η οποία ορίζεται ως 1 /(σταθερό εύρος ζώνης), όπου το σταθερό εύρος ζώνης (sustained bandwidth) είναι το πλήθος των δεδομένων που μπορούν να ανταλλαγούν σε μια μονάδα χρόνου, το μετράται σε επίπεδο εφαρμογής.

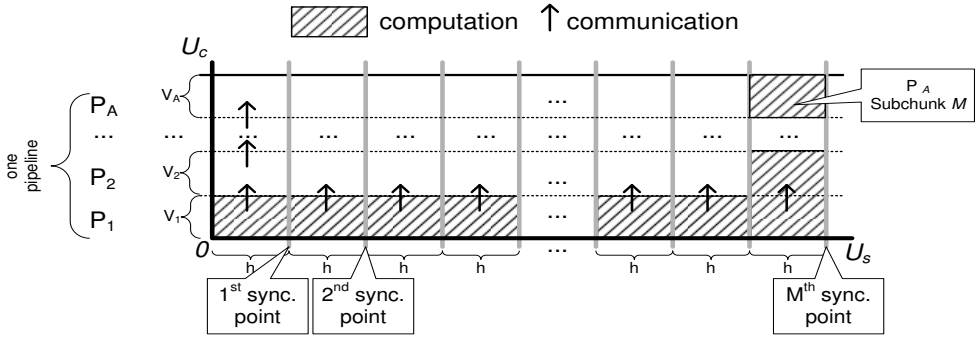
Σημείωση 1: Σε ένα ετερογενές σύστημα, οι εργάτες μπορεί να επικοινωνούν σε διαφορετικές ταχύτητες, ακόμα και αν το δίκτυο διασύνδεσης είναι ομοιογενές. Ακόμα, όταν πολλοί εργάτες στέλνουν μηνύματα στο δίκτυο την ίδια στιγμή, το επίπεδο συμφόρησης του δικτύου αυξάνεται. Για να λάβουμε υπόψη μας τα παραπάνω, πραγματοποιούμε πολλαπλές δοκιμές αποστολής-λήψης (ping-pong test) μεταξύ πολλών ζευγών εργατών για να βρούμε τις μέσες τιμές c_c και c_d.
Σημείωση 2: Επικοινωνίες έχουμε μόνο μεταξύ των m πραγματικών εργατών και όχι μεταξύ των vm εικονικών εργατών.

7.3 Καθορισμός της συχνότητας συγχρονισμού για ετερογενή, μη- αφοσιωμένα συστήματα

Στην συνέχεια θα αναλύσουμε δύο περιπτώσεις αναλογίας του αριθμού εικονικών εργατών και του αριθμού σταδίων της σωλήνωσης (αριθμού διαθέσιμων ομάδων επαναλήψεων): όταν ο αριθμός εικονικών εργατών ισούται με τα αριθμό σταδίων της σωλήνωσης, δηλαδή όταν έχουμε $A = N$ και όταν ο αριθμός εικονικών εργατών είναι μικρότερος του αριθμού σταδίων της σωλήνωσης, δηλαδή όταν $A < N$. Η περίπτωση $A > N$ δεν έχει αναλυθεί γιατί μπορεί να υποβαθμισθεί στην περίπτωση $A = N$ με την χρησιμοποίηση μόνο των N εκ των A εργατών.

7.3.1 Περίπτωση $A = N$

Σε αυτή την περίπτωση ο αριθμός των εικονικών εργατών είναι ίσος με τον συνολικό αριθμό των ομάδων επαναλήψεων, δηλαδή κάθε εικονικός εργάτης θα αναλάβει την εκτέλεση ακριβώς μιας ομάδας επαναλήψεων. Αυτή είναι και η στατική περίπτωση. Για λόγους απλότητας, υποθέτουμε ότι χρησιμοποιείται ο αλγόριθμος DCSS στην διάσταση U_c την οποία και μοιράζει σε N ομάδες επαναλήψεων του ίδιου μεγέθους $V_i = \frac{U_c}{A}$. Το Σχήμα 7.3 απεικονίζει τον διαμερισμό του χώρου επαναλήψεων σε N ίσες ομάδες (οριζόντια τμήματα) οι οποίες έχουν ανατεθεί σε $A = N$ εικονικούς εργάτες. Στο ίδιο σχήμα, μπορούμε επίσης να δούμε τα M σημεία συγχρονισμού τα οποία είναι κατανεμημένα ομοιόμορφα στην διάσταση U_s.



Σχήμα 7.3: Περίπτωση $A = N$: Μοτίβο επικοινωνιών και υπολογισμών για ένα χώρο επαναλήψεων που αντιστοιχεί σε μία σωλήνωση με A στάδια, M στιγμιότυπα και αριθμό βημάτων $A + (M - 1)$.

Ο συνολικός παράλληλος χρόνος σε αυτή την περίπτωση, είναι ο χρόνος υπολογισμού των $A+(M-1)$ βημάτων της σωλήνωσης. Κάθε βήμα αποτελείται, στην γενική περίπτωση, από τις λειτουργίες *λήψης*, *υπολογισμού*, *αποστολής*. Τα βήματα που εκτελούνται από τον πρώτο εργάτη, δηλαδή τον P_1, αποτελούνται μόνο από λειτουργίες *υπολογισμού* και *αποστολής*, αφού δεν χρειάζεται να λάβει δεδομένα από κάποιον προηγούμενο εργάτη. Παρομοίως, τα βήματα που εκτελούνται από τον τελευταίο εργάτη, δηλαδή τον P_A, αποτελούνται μόνο από λειτουργίες λήψης και υπολογισμού.

Ας ορίσουμε ότι το h δηλώνει το διάστημα συγχρονισμού, U_c είναι το μέγεθος της διάστασης δρομολόγησης, U_s το μέγεθος της διάστασης συγχρονισμού και $M = \frac{U_s}{h}$ είναι ο αριθμός των σημείων συγχρονισμού. Επίσης, συμβολίζουμε με c_p τον χρόνο υπολογισμού μιας επανάληψης για τον εικονικό εργάτη, με N τον αριθμό των βημάτων δρομολόγησης, που ισούται με τον αριθμό των ομάδων επαναλήψεων αλλά και με τον αριθμό των εικονικών εργατών A. Τέλος, το μέγεθος των ομάδων επαναλήψεων σύμφωνα με τον αλγόριθμο DCSS είναι $V_i = \frac{U_c}{A}$. Ο συνολικός παράλληλος χρόνος δίνεται από:

$$T_{comp}^{A=N} = \frac{m}{A} hU_c c_p + (hV_i c_p)(M-1) \tag{7.3}$$

Παρατήρηση 4.1 (i) Ο πρώτος όρος της εξίσωσης (7.3) είναι ο χρόνος υπολογισμού όλων των τελευταίων υποομάδων του προβλήματος. Αυτό απεικονίζεται από τα κάθετα σκιασμένα ορθογώνια του Σχήματος 7.3. Αφού βασιζόμαστε στον χρόνο υπολογισμού ανά επανάληψη του εικονικού εργάτη c_p, ο οποίος αντιστοιχεί στον χρόνο υπολογισμού του πιο αργού εργάτη του συστήματος, πρέπει να σταθμίσουμε αυτόν τον πρώτο όρο με $\frac{m}{A}$, έτσι ώστε να η υπολογιστική δύναμη των A εικονικών εργατών να αντιστοιχεί σε αυτή των m πραγματικών. (ii) Παρομοίως, ο δεύτερος όρος της εξίσωσης (7.3) αντιπροσωπεύει τον χρόνο υπολογισμού των $M-1$ στιγμιότυπων της σωλήνωσης, όπως απεικονίζεται από τα οριζόντια σκιασμένα ορθογώνια του Σχήματος 7.3. Είναι προφανές ότι ο συνολικός χρόνος υπολογισμού είναι το άθροισμα όλων των κάθετα και οριζόντια σκιασμένων ορθογωνίων, δηλαδή του συνολικού αριθμού

βημάτων της σωλήνωσης (βλέπε Σχήμα 9.1 και 7.3).

Όπως αναφέρθηκε νωρίτερα, ανταλλαγές δεδομένων λαμβάνουν χώρα μόνο μεταξύ των πραγματικών εργατών. Συμβολίζουμε με t_c τον χρόνο αποστολής ή λήψης ενός μηνύματος, ο οποίος ορίζεται στην εξίσωση (7.2). Ο χρόνος για την ολοκλήρωση όλων των αποστολών και λήψεων δίνεται από:

$$T_{comm}^{A=N} = \quad (m-2)(2t_c) + (M-1)(2t_c) \tag{7.4}$$

Παρατήρηση 4.2 (i) Για ένα χώρο επαναλήψεων συγκεκριμένου μεγέθους και για την περίπτωση που εξετάζουμε, ο χρόνος επικοινωνιών αυξάνεται γραμμικά σε σχέση με το m. Ο πρώτος όρος της εξίσωσης (7.4) αντιπροσωπεύει τον χρόνο επικοινωνιών που σχετίζεται με τον υπολογισμό όλων των πρώτων υποομάδων του προβλήματος, δηλαδή του πρώτου όρου της εξίσωσης (7.3).(ii) Παρομοίως, Ο δεύτερος όρος της εξίσωσης (7.4) είναι ο χρόνος επικοινωνιών που σχετίζεται με τον υπολογισμό όλων των $M-1$ στιγμιότυπων της σωλήνωσης (δηλαδή με τον δεύτερο όρο της εξίσωσης (7.3).

Πρέπει να προσθέσουμε και τον χρόνο ανάθεσης εργασίας, αφού έχουμε μοντέλο συντονιστή-εργάτη, ο οποίος είναι $T_{wa} = 2t_{ms} + c_{sch}$, όπου $t_{ms} = c_d + Vc_c$. Σε αυτό τον χρόνο περιέχεται ο χρόνος για την ανταλλαγή πληροφοριών μεταξύ του εργάτη και του συντονιστή, καθώς και ο χρόνος c_{sch} που ξοδεύει ο συντονιστής για να υπολογίσει το μέγεθος της επόμενης ομάδας επαναλήψεων, που καλούμε κόστος δρομολόγησης (scheduling overhead).

Επομένως, ο συνολικός παράλληλος χρόνος για την περίπτωση $A = N$ είναι:

$$T_{par}^{A=N} \quad = T_{comp}^{A=N} + T_{comm}^{A=N} + T_{wa} \tag{7.5}$$

Ο χρόνος T_{wa} λαμβάνεται μόνο μια φορά γιατί οι χρόνοι για τις επόμενες αναθέσεις επικαλύπτονται από άλλες λειτουργίες υπολογισμών ή επικοινωνιών.

Στην συνέχεια ορίζουμε τον παράλληλο χρόνο συναρτήσει του διαστήματος συγχρονισμού h το οποίο και μορφοποιούμε σαν την Πρόταση 1. Για τις τιμές του N και V_i που δίνονται από έναν αλγόριθμο δρομολόγησης (βλέπε Πίνακα 7.1), ο ελάχιστος παράλληλος χρόνος μπορεί να βρεθεί παραγωγίζοντας τον $T_{par}^{A=N}$ και επιβεβαιώνοντας ότι η δεύτερη παράγωγος είναι θετική.

Πρόταση 1. *Ο παράλληλος χρόνος $T_{par}^{A=N}$ σαν συνάρτηση του h έχει ελάχιστη τιμή στο σημείο:*

$$h_{opt}^{A=N} \quad = \quad \sqrt{\frac{2U_s A c_d}{c_p(V_i A - U_c m) - 2c_c A(m-2)}} \tag{7.6}$$

Απόδειξη. Αφού μας ενδιαφέρει να βρούμε το διάστημα συγχρονισμού που ελαχιστοποιεί τον $T_{par}^{A=N}$ παραγωγίζουμε $T_{par}^{A=N}$ ως προς το h. Αυτό μας δίνει:

$$\frac{d}{dh}T_{par}^{A=N} \quad = \quad 2c_c(m-3) + c_p(U_c\frac{m}{A} - V_i) - \frac{2U_s c_d}{h^2} \tag{7.7}$$

και η λύση του $\frac{d}{dh}T_{par}^{A=N} = 0$ δίνεται από την εξίσωση (7.6). Με την χρήση του πακέτου Mathcad [106] υπολογίζουμε την δεύτερη παράγωγο του $T_{par}^{A=N}$ στο σημείο $h_{opt}^{A=N}$, και διαπιστώνουμε ότι η $\frac{d^2}{d(h_{opt}^{A=N})^2}T_{par}^{A=N}$ είναι θετική. Αυτό μας καθορίζει ότι η λύση της εξίσωσης (7.6) είναι ένα τοπικό ελάχιστο του παράλληλου χρόνου $T_{par}^{A=N}$.

7.3.2 Περίπτωση $A < N$

Αυτή είναι η γενική περίπτωση στην οποία οι ομάδες υπολογισμών είναι περισσότερες από τους διαθέσιμους εργάτες και άρα κάθε εργάτης θα αναλάβει περισσότερες από μια ομάδες. Κάθε νέος γύρος αναθέσεων αντιστοιχεί σε μια ξεχωριστή σωλήνωση. Για λόγους απλότητας υποθέτουμε ότι ο αριθμός των ομάδων επαναλήψεων είναι ακέραιο πολλαπλάσιο του αριθμού των εργατών, και άρα έχουμε ακριβώς p γύρους ανάθεσης και p σωληνώσεις. Στην πράξη είναι πιθανών να μην χρησιμοποιηθούν όλοι οι εργάτες στην τελευταία σωλήνωση. Η έξοδος κάθε σωλήνωσης τροφοδοτεί την είσοδο της επόμενης. Το Σχήμα 7.4 απεικονίζει μια συλλογή από p επαναλήψεις του Σχήματος 9.1. Ο θεωρητικός παράλληλος χρόνος σε αυτή την περίπτωση είναι το άθροισμα των χρόνων ολοκλήρωσης όλων των p σωληνώσεων και του χρόνου μεταφοράς των δεδομένων μεταξύ των σωληνώσεων. Στην πραγματικότητα, ο συνολικός παράλληλος χρόνος ορίζεται από τον χρόνο που θα ολοκληρωθεί η τελευταία υποομάδα του προβλήματος, η οποία απεικονίζεται στο Σχήμα 7.4 σαν 'P_A, *υποομάδα* M, *σωλήνωση* p'. Κάθε σωλήνωση μπορεί να ολοκληρωθεί σε $A + (M-1)$ βήματα, τα οποία, όπως αναφέρθηκε προηγουμένως απαρτίζονται από τις λειτουργίες λήψης, υπολογισμού και αποστολής (βλέπε Σχήμα 9.1 και 7.3). Και σε αυτή την περίπτωση, τα βήματα που εκτελούνται από τον πρώτο εικονικό εργάτη δεν περιλαμβάνουν λήψης δεδομένων και αυτά που εκτελούνται από τον τελευταίο δεν περιλαμβάνουν αποστολές δεδομένων.

Οπότε, ο συνολικός χρόνος των υπολογισμών για όλες τις σωληνώσεις είναι p φορές ο χρόνος μίας σωλήνωσης, δηλαδή θα έχουμε $p \times [A + (M-1)]$ βήματα υπολογισμού, που ολοκληρώνονται σε χρόνο:

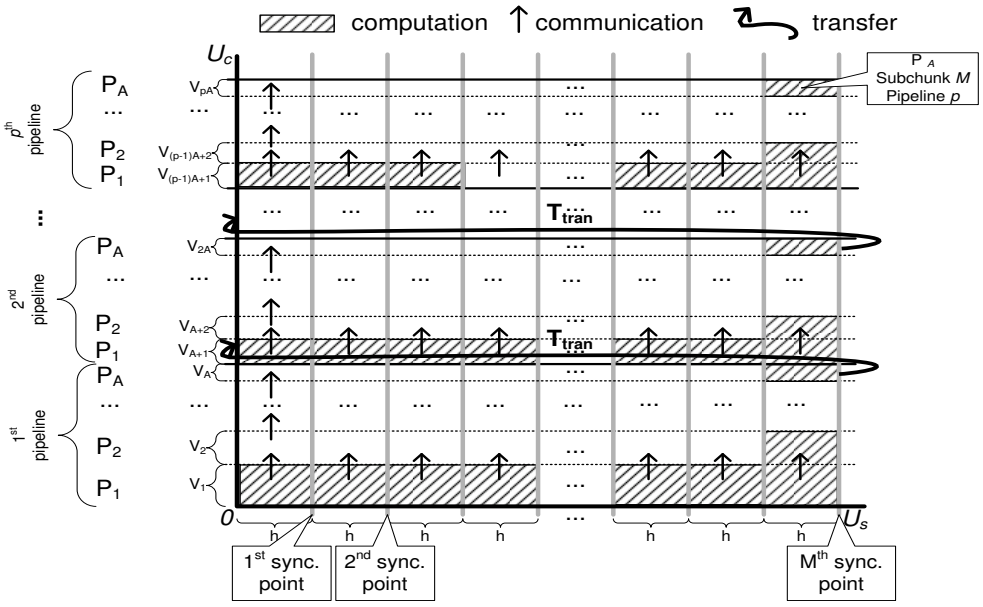
$$T_{comp}^{A<N} = \frac{m}{A}c_p h U_c + c_p h (M-1) \sum_{j=0}^{p-1} V_{jA+1} \tag{7.8}$$

Όπου V_{jA+1} είναι το μέγεθος υπολογισμού της πρώτης ομάδας επαναλήψεων κάθε σωλήνωσης.

Παρατήρηση 4.3 (*i*) Ο πρώτος όρος της εξίσωσης (7.8) μας δίνει τον χρόνο υπολογισμού όλων των τελευταίων υποομάδων του προβλήματος, τα οποία αντιστοιχούν σε $p \times A$ βήματα υπολογισμών, τα οποία απεικονίζονται από τα κάθετα σκιασμένα ορθογώνια του σχήματος 7.4. Τον όρο αυτό πρέπει να τον σταθμίσουμε πάλι με τον λόγο $\frac{m}{A}$, όπως στην περίπτωση $N = A$. (*ii*) Ο δεύτερος όρος της εξίσωσης (7.8) αναφέρεται στον χρόνο υπολογισμού των $M-1$ στιγμιότυπων κάθε σωλήνωσης, τα οποία ολοκληρώνονται σε $p(M-1)$ βήματα υπολογισμού και απεικονίζονται στο Σχήμα 7.4 από τα οριζόντια σκιασμένα ορθογώνια.

Ο συνολικός χρόνος των υπολογισμών δίνεται από το άθροισμα των κάθετων και οριζόντιων σκιασμένων ορθογωνίων του Σχήματος 7.4.

Ο συνολικός αριθμός βημάτων αποστολών και λήψης για όλες τις σωληνώσεις είναι: $p[(m-2) + (M-1)] + (p-1)$. Οπότε ο χρόνος για την πραγματοποίηση



Σχήμα 7.4: Περίπτωση $A < N$: Μοτίβο επικοινωνιών και υπολογισμών για ένα χώρο επαναλήψεων που αντιστοιχεί σε p σωληνώσεις, κάθε μια με A στάδια, M στιγμιότυπα και αριθμό βημάτων $A + (M - 1)$.

όλων των επικοινωνιακών λειτουργιών, συμπεριλαμβανομένου και του χρόνου για την μεταφορά δεδομένων μεταξύ των σωληνώσεων είναι ο ακόλουθος:

$$T_{comm}^{A<N} = \quad p(m-2)(2t_c) + p(M-1)(2t_c) + (p-1)T_{tr} \tag{7.9}$$

Παρατήρηση 4.4 (i) Για ένα χώρο επαναλήψεων συγκεκριμένου μεγέθους και για την περίπτωση που εξετάζουμε, ο χρόνος επικοινωνιών αυξάνεται γραμμικά σε σχέση με το m και το p. Ο πρώτος όρος της εξίσωσης (7.9) αντιπροσωπεύει τον χρόνο επικοινωνιών που σχετίζεται με τον υπολογισμό όλων των πρώτων υποομάδων του προβλήματος, δηλαδή του πρώτου όρου της εξίσωσης (7.8), που αντιστοιχεί στα $p(m-2)$ βήματα επικοινωνιών. (ii) Παρομοίως, Ο δεύτερος όρος της εξίσωσης (7.9) είναι ο χρόνος επικοινωνιών που σχετίζεται με τον υπολογισμό όλων των $M-1$ στιγμιότυπων της σωλήνωσης, δηλαδή με τον δεύτερο όρο της εξίσωσης (7.8), που αντιστοιχεί στα $p(M-1)$ βήματα επικοινωνιών. (iii) T_{tr} είναι ο χρόνος μεταφοράς των δεδομένων από μια σωλήνωση στην επόμενη και υπολογίζεται ως $2\ (c_d + U_s c_c)$ και επαναλαμβάνεται $p-1$ φορές (βλέπε Σχήμα 7.4).

Πρέπει τέλος, να προσθέσουμε και τον χρόνο ανάθεσης εργασίας, ο οποίος είναι υπολογίζεται πάλι όπως στην προηγούμενη περίπτωση σαν $T_{wa} = 2t_{ms} + c_{sch}$, όπου $t_{ms} = c_d + Vc_c$. Σε αυτό τον χρόνο περιέχεται ο χρόνος για την ανταλλαγή πληροφοριών μεταξύ του εργάτη και του συντονιστή, καθώς και ο χρόνος c_{sch} που ξοδεύει ο συντονιστής για να υπολογίσει το μέγεθος της επόμενης ομάδας επαναλήψεων. Ο χρόνος αυτός υπολογίζεται πάλι μόνο για την πρώτη ομάδα επαναλήψεων

Πίνακας 7.1: Αριθμός βημάτων δρομολόγησης και μεγέθη ομάδων επαναλήψεων με κατώφλι t $(= L)$.

	N	V_i
$DCSS$	$\frac{U_c}{V_i}$	V_i
$DGSS$	$\frac{1}{\ln\lceil\frac{A}{A-1}\rceil}(\ln\lceil\frac{U_c}{A}\rceil - \ln\ t)$	$(1-\frac{1}{A})^i\frac{U_c}{A}$
$DTSS$	$\frac{2U_c}{F+L}$	$F-(i-1)D$
$DFSS$	$A(1.44\ln\lceil\frac{U_c}{A}\rceil - \ln\ t)$	$(\frac{1}{\alpha})^{i+1}\frac{U_c}{A}$
$DTFSS$	$\frac{2U_c}{F+L}$	$[\sum_{i=k}^{k+A}(F-(i-1)D)]/A$

του προβλήματος.

Οπότε ο συνολικός παράλληλος χρόνος για την περίπτωση $A < N$ είναι:

$$T_{par}^{A<N} = T_{comp}^{A<N} + T_{comm}^{A<N} + T_{wa} \tag{7.10}$$

Στην συνέχεια ορίζουμε τον παράλληλο χρόνο συναρτήσει του διαστήματος συγχρονισμού h τον οποίο και μορφοποιούμε σαν την Πρόταση 2. Για τις τιμές του N και V_i που δίνονται από έναν αλγόριθμο δρομολόγησησ(βλέπε Πίνακα 7.1), ο ελάχιστος παράλληλος χρόνος μπορεί να βρεθεί παραγωγίζοντας τον $T_{par}^{A<N}$ και επιβεβαιώνοντας ότι η δεύτερη παράγωγος είναι θετική.

Πρόταση 2. *Ο παράλληλος χρόνος $T_{par}^{A<N}$ σαν συνάρτηση του h έχει ελάχιστη τιμή στο σημείο:*

$$h_{opt}^{A<N} = \sqrt{\frac{2\ (U_s\ p\ A\ c_d)}{(2m-6)Ac_cp+U_cc_pm-c_pA\sum_{j=0}^{p}V_{jA+1}}} \tag{7.11}$$

Απόδειξη. Αφού μας ενδιαφέρει να βρούμε το διάστημα συγχρονισμού που ελαχιστοποιεί τον $T_{par}^{A<N}$ παραγωγίζουμε $T_{par}^{A<N}$ ως προς το h. Αυτό μας δίνει:

$$\frac{d}{dh}T_{par}^{A<N} = U_cc_p\frac{m}{A} - c_p\sum_{j=0}^{p-1}V_{jA+1} + (2m-6)c_cp - 2\frac{U_sc_d}{h^2} \tag{7.12}$$

και η λύση του $\frac{d}{dh}T_{par}^{A<N} = 0$ δίνεται από την εξίσωση (7.11). Με την χρήση του πακέτου $Mathcad$ [106] υπολογίζουμε την δεύτερη παράγωγο του $T_{par}^{A<N}$ στο σημείο $h_{opt}^{A<N}$, και διαπιστώνουμε ότι η $\frac{d^2}{d(h_{opt}^{A<N})^2}T_{par}^{A<N}$ είναι θετική. Αυτό μας καθορίζει ότι η λύση της εξίσωσης (7.11) είναι ένα τοπικό ελάχιστο του παράλληλου χρόνου $T_{par}^{A=N}$.

7.4 Πειραματική αξιολόγηση

Πειραματική διάταξη. Τα πειράματα διεξάχθηκαν σε μια συστοιχία υπολογιστών (Intel Xeons, 2.8GHz, 2GB RAM), διασυνδεδεμένων με δίκτυο GigaBit Ethernet στην οποία χρησιμοποιήθηκαν 17 διεργασίες MPI (1 συντονιστής και m=16 εργάτες). Ο κώδικας υλοποιήθηκε σε γλώσσα C, με την χρήση του gcc 4.2.3. Ταυτόχρονα με

τα πειράματα εκτελέσθηκαν διεργασίες στο παρασκήνιο (background jobs) των $m/2$ εργατών, προκειμένου να δημιουργηθεί τεχνητή ανομοιογένεια στο σύστημα. Τα αποτελέσματα που παρουσιάζονται αποτελούν τον μέσο όρο 10 εκτελέσεων. Η εφαρμογή που εκτελείται είναι ο αλγόριθμος εξομάλυνσης σφάλματος Floyd-Steinberg (F-S), ο οποίος περιγράφεται στην ενότητα 5.3.1. Σε όλες τις περιπτώσεις χρησιμοποιήθηκε σαν ελάχιστο μέγεθος ομάδας οι $t = 20$ επαναλήψεις κατά μήκος της διάστασης Uc.

Προσδιορισμός παραμέτρων. Για να προσδιορισθούν οι τιμές των παραμέτρων επικοινωνιών, αναπτύχθηκε ένα μετροπρόγραμμα το οποίο εξομοιώνει ένα μικρής κλίμακας μοντέλο συντονιστή-εργάτη. Πραγματοποιεί αποστολές και λήψης μηνυμάτων διαφορετικού μεγέθους, μεταξύ όλων των ζευγών εργατών προκειμένου να εξομοιώσει επίπεδα συμφόρηση του δικτύου παρόμοια με αυτά των πραγματικών εφαρμογών. Το μετροπρόγραμμα αυτό μετράει τον μέσο χρόνο αποστολής-λήψης (round-trip time) και τον διαιρεί με το 2, υποθέτοντας ότι ο χρόνος αποστολής και λήψης είναι ίδιοι. Αυτή η μέτρηση επιτρέπει τον προσδιορισμό της χρόνου αρχικοποίησης $c_d = 8 \times 10^{-5}s$ και του ρυθμοαπόδοσης $c_c = 6.55 \times 10^{-7}s$. Το κόστος δρομολόγησης για κάθε αλγόριθμο βρέθηκε ως: DCSS $c_{sch} = 3.2 \times 10^{-5}s$, DTSS $c_{sch} = 3.4 \times 10^{-5}s$, DFSS $c_{sch} = 7 \times 10^{-5}s$, DGSS $c_{sch} = 8.5 \times 10^{-5}s$ και DTFSS $c_{sch} = 3.7 \times 10^{-5}s$.

Για να προσδιορισθούν οι τιμές των παραμέτρων υπολογισμών, εκτελέσθηκαν εκδόσεις μικρής κλίμακας της εφαρμογής και στους απλούς αλλά και στους υπερφορτωμένους εργάτες. Με αυτόν τον τρόπο, προσδιορίσθηκαν οι τιμές του c_p, οι οποίες για τους κανονικούς εργάτες είναι $c_p^{fast} = 3.3 \times 10^{-8}s$, ενώ για τους υπερφορτωμένους $c_p^{slow} = 2c_p^{fast}$. Με βάση τα παραπάνω, έχουμε για τους υπερφορτωμένους εργάτες $VP^{slow} = 1$ και για τους απλούς εργάτες $VP^{fast} = 2$. Ο Πίνακας 2 συνοψίζει τις τιμές όλων των παραπάνω παραμέτρων.

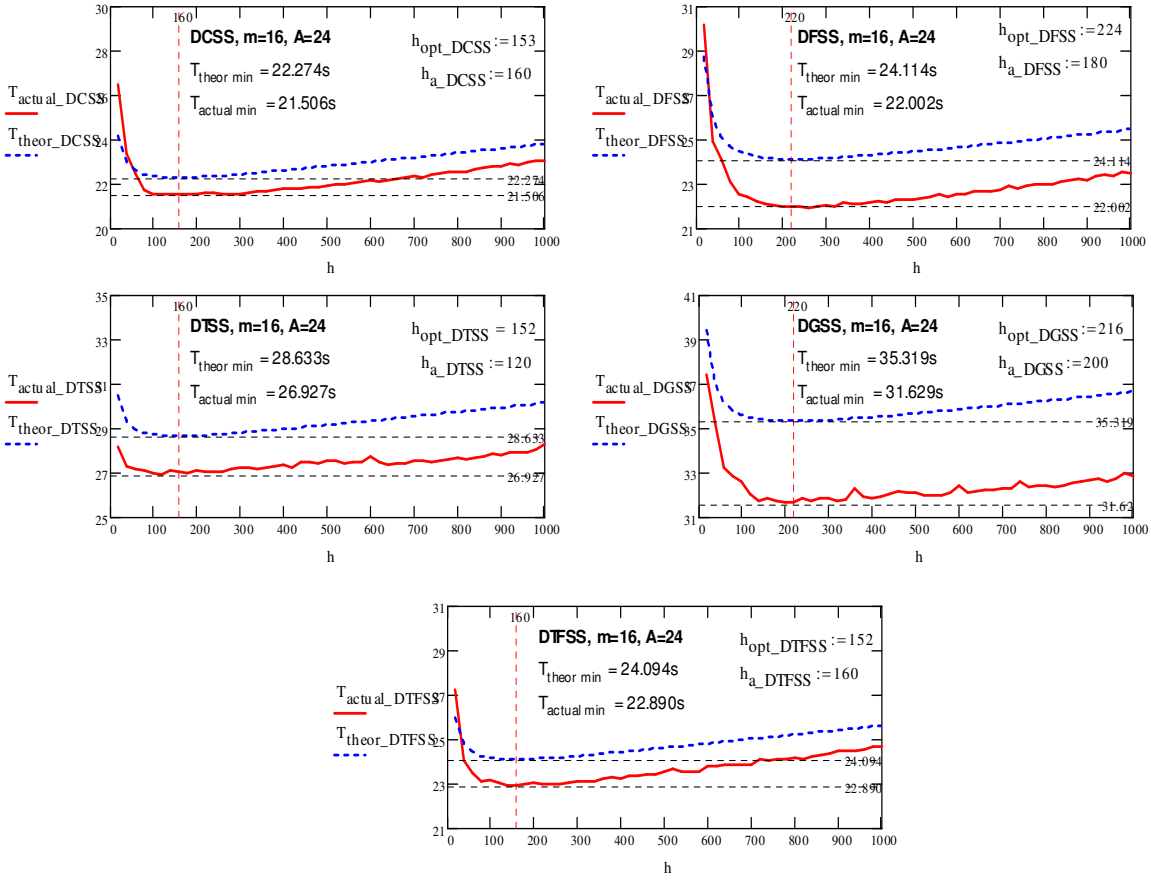
Πίνακας 7.2: Εκτιμώμενες τιμές παραμέτρων για τα μοντέλα υπολογισμών-επικοινωνιών.

$F-S$	c_d	c_c	c_p^{slow}	VP^{slow}	VP^{fast}
	$8 \times 10^{-5}s$	$6.55 \times 10^{-7}s$	$6.6 \times 10^{-8}s$	1	2
c_{sch}	$DCSS$	$DTSS$	$DFSS$	$DGSS$	$DTFSS$
	$3.2 \times 10^{-5}s$	$3.4 \times 10^{-5}s$	$7 \times 10^{-8}s$	$8.5 \times 10^{-8}s$	$3.7 \times 10^{-8}s$

7.4.1 Αποτελέσματα

Κατά την διάρκεια των πειραμάτων, υπερφορτώσαμε με εξωτερικά φορτία τους μισούς φυσικούς επεξεργαστές ($m/2$). Για $m = 16$ εργάτες, η συνολική δύναμη ήταν $A = 24$. Εκτελέσαμε την εφαρμογή $F - S$ με τους αλγόριθμους δρομολόγησης DCSS, DTSS, DFSS, DGSS και DTFSS σε ένα πλαίσιο εικόνας 50000×150000 σημείων, για διαστήματα συγχρονισμού $20, 40, \ldots, 1000$ σημείων και μετρήσαμε τον πραγματικό παράλληλο χρόνο. Απεικονίσαμε αυτό τον χρόνο στο διάγραμμα του Σχήματος 7.5. Στο ίδιο διάγραμμα περιλαμβάνονται τα αποτελέσματα για τον θεωρητικά εκτιμώμενο παράλληλο χρόνο, που συγκεντρώθηκαν με την χρήση της συνάρτησης εχ. (10.1). Καθορίσαμε επίσης θεωρητικά το βέλτιστο διάστημα συγχρονισμού $h_{opt}^{A<N}$ με την

χρήση της Συνάρτησης (7.11) για κάθε αλγόριθμο, για 24 εικονικούς εργάτες. Τα αποτελέσματα δίνονται στον Πίνακα 7.4.1



Σχήμα 7.5: Θεωρητικοί χρόνοι έναντι των πραγματικών, όπως επίσης βέλτιστες θεωρητικές και πραγματικές τιμές του διαστήματος συγχρονισμού h για τις διάφορες μεθόδους ($h = 20, 40, \ldots, 1000$).

Μπορούμε να δούμε ότι η μέθοδος μας περιγράφει με ακρίβεια την συμπεριφορά του συστήματος αφού η καμπύλη του θεωρητικού παράλληλου χρόνου ακολουθεί σε όλες τις περιπτώσεις την καμπύλη των πραγματικών παράλληλων χρόνων. Πιο συγκεκριμένα, και οι δύο καμπύλες έχουν ελάχιστη τιμή για το ίδιο περίπου διάστημα συγχρονισμού h. Τα αποτελέσματα δείχνουν ότι το βέλτιστο $h_{opt}^{A<N}$ που δίνεται από την Εξίσωση (7.11) είναι πολύ κοντά στο πραγματικό h_a που δίνει τον ελάχιστο παράλληλο χρόνο στην πράξη. Η απόσταση μεταξύ της θεωρητικής και της πρακτικής καμπύλης οφείλεται κυρίως στις αποκλίσεις των μετρήσεων των χαρακτηριστικών των μοντέλων επικοινωνιών και υπολογισμών και στην αδυναμία ακριβούς προσδιορισμού των βημάτων δρομολόγησης N, των αλγορίθμων δρομολόγησης που αναφέρονται στην ενότητα 2.3.2. Ωστόσο η απόσταση αυτή δεν επηρεάζει τις επιδόσεις του μοντέλου

στον προσδιορισμό του βέλτιστου διαστήματος συγχρονισμού.

Σε όλες τις περιπτώσεις, η διαφορά μεταξύ των δύο διαστημάτων συγχρονισμού, $h_{opt}^{A<N}$ και h_a, όπως και η διαφορές μεταξύ των ελάχιστων θεωρητικών και πρακτικών παράλληλων χρόνων για κάθε ένα από αυτά τα διαστήματα δίνονται στο Πίνακα 7.4.1. Το συμπέρασμα που μπορούμε να βγάλουμε από τον πίνακα αυτόν είναι ότι χρησιμοποιώντας το θεωρητικά υπολογισμένο βέλτιστο διάστημα συγχρονισμού $h_{opt}^{A<N}$, μπορούμε να επιτύχουμε ένα παράλληλο χρόνο που είναι πολύ κοντά στον πραγματικό ελάχιστο παράλληλο χρόνο.

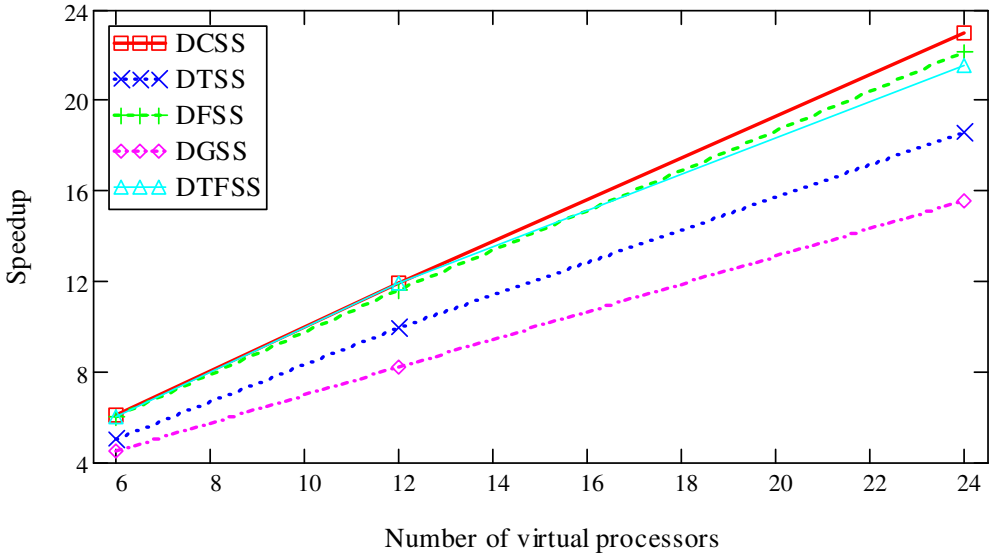
Πίνακας 7.3: Βέλτιστες θεωρητικές και πραγματικές τιμές του διαστήματος συγχρονισμού, και οι αντίστοιχοι παράλληλοι χρόνοι.

$Floyd-Steinberg\ A=24$					
Alg.	$h_{opt}^{A<N}$	h_a	$T_{t_{min}}$	$T_{a_{min}}$	$\lvert T_{t_{min}} - T_{a_{min}}\rvert$
DCSS	153	160	22.274	21.506	0.768
DTSS	152	120	28.633	26.927	1.706
DFSS	224	180	24.114	22.002	2.112
DGSS	216	200	35.319	31.629	3.690
DTFSS	152	160	24.094	22.890	1.204
Alg.	$\lvert T_{t_{min}} - T_{a_{min}}\rvert / T_{a_{min}}$	$\lvert h_{opt}^{A<N} - h_a\rvert$	$\frac{\lvert h_{opt}^{A<N} - h_a\rvert}{h_a} \cdot 100$		
DCSS	0.0357%	7	0.0437%		
DTSS	0.0633%	8	0.0666%		
DFSS	0.0959%	44	0.2444%		
DGSS	0.1166%	16	0.0800%		
DTFSS	0.0525%	8	0.0500%		

Επίσης μετρήσαμε την επιτάχυνση σε σχέση με την σειριακή εφαρμογή, για όλους τους αλγόριθμους δρομολόγησης και για $m = 4, 8, 16$ επεξεργαστές. Η ακόλουθη διάταξη χρησιμοποιήθηκε: για $m = 4$, είχαμε 2 γρήγορους και 2 αργούς επεξεργαστές, για $m = 8$, 4 γρήγορους και 4 αργούς επεξεργαστές, και τέλος για $m = 16$, 8 γρήγορους και 8 αργούς επεξεργαστές. Επομένως, είχαμε πίνακες των $A = 6, 12, 24$ εικονικών επεξεργαστών αντιστοίχως. Σε κάθε περίπτωση χρησιμοποιήσαμε το βέλτιστο θεωρητικό διάστημα συγχρονισμού (βλέπε Σχήμα 7.5). Ο σειριακός χρόνος μετρήθηκε σε ένα εικονικό επεξεργαστή και τα αποτελέσματα δίνονται στο Σχήμα 7.6.

7.5 Συμπεράσματα

Σε αυτή την ενότητα παρουσιάστηκε ένα θεωρητικό μοντέλο που επιτρέπει τον καθορισμό του βέλτιστου διαστήματος συγχρονισμού για εφαρμογές που περιέχουν βρόχους με εξαρτήσεις (DOACROSS loops) σε ετερογενή συστήματα υπολογιστών. Το μοντέλο που προτείνεται είναι αρκετά γενικό ώστε να καλύπτει ολόκληρη την τάξη των αυτοδρομολογούμενων αλγορίθμων. Η χρησιμότητα της μεθοδολογίας οφείλεται στο γεγονός ότι μια κακή επιλογή διαστήματος συγχρονισμού μπορεί να οδηγήσει σε μείω-



Σχήμα 7.6: Επιτάχυνση που επιτυγχάνουν οι διαφορές μέθοδοι και τα βέλτιστα διαστήματα συγχρονισμού.

ση της απόδοσης και το κόστος του καθορισμού του βέλτιστου διαστήματος συγχρονισμού μέσω εξαντλητικής αναζήτησης είναι απαγορευτικό. Το διάστημα συγχρονισμού που καθορίζεται βάση του μοντέλου είναι πολύ κοντά σε αυτό που διαπιστώνεται στην πράξη. Τέλος, τα πειραματικά αποτελέσματα αποδεικνύουν ότι χρησιμοποιώντας αυτό το διάστημα συγχρονισμού μπορεί να επιτευχθεί πολύ καλή επιτάχυνση σε σχέση με τον σειριακό κώδικα.

Κεφάλαιο 8

Αυτό-προσαρμοζόμενη δρομολόγηση σε στοχαστικά περιβάλλοντα

Αυτό το κεφάλαιο εξετάζει δυναμικούς αλγόριθμους εξισορρόπησης φορτίων για μη-αφοσιωμένες και ετερογενείς συστοιχίες υπολογιστών. Παρουσιάζεται ένας νέος αλγόριθμος που ονομάζεται 'αλγόριθμος αυτό-προσαρμοζόμενης δρομολόγησης ' (Self-Adapting Scheduling ή για συντομία SAS). Ο Αλγόριθμος αυτός στοχεύει στην δυναμική δρομολόγηση φωλιασμένων βρόχων, με ομοιόμορφες εξαρτήσεις σε στοχαστικά περιβάλλοντα. Το φορτίο που απασχολεί το σύστημα και δεν ανήκει στην παράλληλη εφαρμογή (εξωγενές φορτίο), ακολουθεί μια απρόβλεπτη συμπεριφορά, η οποία μπορεί να μοντελοποιηθεί σαν στοχαστική διεργασία.

Στην ενότητα 5.2 παρουσιάστηκε ο DMPS(DTSS), ο οποίος σύμφωνα με τα πειραματικά αποτελέσματα είναι ο πιο αποδοτικός αλγόριθμος για την δυναμική δρομολόγηση φωλιασμένων βρόχων με εξαρτήσεις, σε ετερογενείς συστοιχίες υπολογιστών. Σε αυτό το κεφάλαιο συγκρίνουμε την απόδοση του SAS με αυτήν του DMPS(DTSS) κάνοντας την υπόθεση ότι ο χρόνος ζωής και το μεσοδιάστημα άφιξης των εισερχομένων εργασιών, που συνθέτουν το φορτίο της συστοιχίας, ακολουθούν μια εκθετική κατανομή. Τα πειραματικά αποτελέσματα δείχνουν ότι ο αλγόριθμος SAS υπερισχύει σημαντικά του DMPS(DTSS), ειδικά σε περιπτώσεις με γρήγορα μεταβαλλόμενα φορτία.

8.1 Εισαγωγή

Η πρώτη προσπάθεια προσαρμογής των αυτόδρομολογούμενων αλγορίθμων σε ετερογενή συστήματα ήταν το Weighted Factoring (WF) που παρουσιάζεται στο [82]. Η διαφορά του WF από τον απλό FSS είναι ότι στον πρώτο, τα μεγέθη των ομάδων επαναλήψεων σταθμίζονται με κάποια βάρη (weights) που προκύπτουν από την σχετική υπολογιστική ισχύ των εργατών. Ωστόσο, στον WF τα βάρη των εργατών

παραμένουν σταθερά σε όλη την διάρκεια εκτέλεσης του παράλληλου προγράμματος. Αυτό έχει σαν αποτέλεσμα την καλή απόδοση σε ετερογενή και αφοσιωμένα συστήματα αλλά την μείωση της απόδοσης σε μη-αφοσιωμένα συστήματα. Οι Banicescu & Liu πρότειναν στο [41] μια μέθοδο που ονομάζεται "Adaptive Factoring (AF)", η οποία προσαρμόζει τα βάρη αυτά κατά την διάρκεια του χρόνου εκτέλεσης, χρησιμοποιώντας μετρήσεις χρόνων που αντικατοπτρίζουν τις μεταβολές της υπολογιστικής ισχύος των εργατών. Η μέθοδος αυτή είχε σχεδιαστεί για εφαρμογές που διεξάγονται σε χρονικά βήματα(time-stepping), όπου οι μετρήσεις που διεξάγονται στα προηγούμενα χρονικά βήματα μπορούν να χρησιμοποιηθούν για να βελτιώσουν την απόδοση στα επόμενα χρονικά βήματα. Οι Chronopoulos et al. παρουσίασαν στο [4] μια επέκταση του αλγόριθμου TSS, τον Distributed TSS (DTSS), ο οποίος περιγράφεται στην ενότητα 5.1.1. Στον DTSS τα μεγέθη των ομάδων επαναλήψεων μεταβάλλονται ανάλογα με την επεξεργαστική ισχύ των εργατών αλλά και τον αριθμό των διεργασιών που βρίσκονται σε κάθε βήμα δρομολόγησης στην ουρά εκτέλεσης του κάθε εργάτη, δηλαδή ανάλογα με την διαθέσιμη επεξεργαστική ισχύ. Εδώ συνδυάζουμε τις αρετές του DTSS (πληροφορίες από την ουρά εκτέλεσης) και του AF(μετρήσεις χρόνων), και παρουσιάζουμε τον SAS, ένα αλγόριθμο που εξισορροπεί δυναμικά το υπολογιστικό φορτίο της παράλληλης εφαρμογής, κάτω από την παρουσία μεταβαλλόμενου εξωγενούς φορτίου. Ο SAS αναθέτει ομάδες επαναλήψεων στους εργάτες σύμφωνα με:

1. Το ιστορικό όλων των προηγούμενων χρονικών μετρήσεων για την συγκεκριμένη εφαρμογή
2. Το ιστορικό των εισερχόμενων εργασιών
3. τον αριθμό των διεργασιών στην ουρά εκτέλεσης των εργατών στο τρέχων βήμα δρομολόγησης

Πρέπει να σημειώσουμε ότι οι αλγόριθμοι που αναφέρονται παραπάνω έχουν σχεδιαστεί για βρόχους που δεν περιέχουν εξαρτήσεις, αντίθετα ο SAS στοχεύει σε βρόχους με εξαρτήσεις. Για την ανταλλαγή δεδομένων χρησιμοποιείται μηχανισμός συγχρονισμού που αναλύθηκε στην ενότητα 6.2. Ο υπολογισμός εκτελείται πάλι σε μια μορφή σωλήνωσης (pipeline).

8.2 Ο αλγόριθμος Self Adaptive Scheduling

Σε αυτή την ενότητα περιγράφεται ο αλγόριθμος SAS, μια προσπάθεια για δυναμική εξισορρόπηση φορτίου κάτω από την παρουσία στοχαστικού εξωγενούς φορτίου. ο SAS είναι ένας αυτοδρομολογούμενος αλγόριθμος, που αναθέτει ομάδες επαναλήψεων στους εργάτες συνδυάζοντας το ιστορικό των χρόνων υπολογισμού των προηγούμενων ομάδων επαναλήψεων που τους είχαν ανατεθεί, το ιστορικό των εργασιών που περιέχονται στην ουρά εκτέλεσης τους και τον αριθμό των εργασιών στην ουρά εκτέλεσης τους στο τρέχον βήμα δρομολόγησης. Ξεκινάει ταξινομώντας τους εργάτες σε φθίνουσα σειρά, σύμφωνα με την ικανότητα υπολογισμών τους. Χωρίς βλάβη της γενικότητας υποθέτουμε ότι $VP_1 \geq VP_2 \geq \ldots \geq VP_m$ και ότι $VP_1 = 1$. Ο SAS

υπολογίζει και αναθέτει τις πρώτες ομάδες επαναλήψεων στους εργάτες ανάλογα με την ικανότητα υπολογισμών τους. Η πρώτη ομάδα επαναλήψεων που ανατίθεται στον P_k υπολογίζεται σύμφωνα με τον παρακάτω τύπο:

$$\left\lceil \frac{u_c \times VP_k}{2 \times \sum_{i=1}^{m} VP_i} \right\rceil \tag{8.1}$$

Θεωρούμε ότι έχουμε ικανοποιητική εξισορρόπηση φορτίου όταν όλοι οι εργάτες ολοκληρώνουν τον υπολογισμό τις ομάδας που τους ανατέθηκε μέσα στον ίδιο περίπου χρόνο. Για αυτό τον λόγο ο SAS χρησιμοποιεί την έννοια του χρόνου αναφοράς (reference time), που συμβολίζεται ως t_{ref}, υπό την έννοια ότι περιμένουμε όλοι ο εργάτες να ολοκληρώσουν τον υπολογισμό της κάθε ομάδας επαναλήψεων μέσα σε αυτό τον χρόνο αναφοράς. Ο χρόνος αυτός λαμβάνεται ως ο χρόνος υπολογισμού της πρώτης ομάδας από τον πρώτο εργάτη. Ο SAS κατανέμει τους υπολογισμούς έτσι ώστε όλοι οι εργάτες να ολοκληρώσουν τις ομάδες επαναλήψεων τους μέσα σε αυτόν τον ενδεικτικό χρόνο. Αυτό το επιτυγχάνει κάνοντας προβλέψεις για τον χρόνο υπολογισμού μιας ομάδας, βασιζόμενος σε μετρήσεις χρόνων υπολογισμού προηγούμενων ομάδων επαναλήψεων και σε ιστορικά στοιχεία των ουρών εκτέλεσης των εργατών.

Επιπλέον Ορολογία.

- V_1^k, ..., V_j^k είναι τα μεγέθη των πρώτων ομάδων επαναλήψεων που έχουν ανατεθεί στον P_k και $t_1^k, \ldots, t_j^k$ είναι οι πραγματικοί χρόνοι υπολογισμού τους.
- $q_1^k, \ldots, q_j^k$ είναι ο αριθμός εργασιών στην ουρά εκτέλεσης του P_k κατά την ανάθεση των j πρώτων ομάδων επαναλήψεων.
- Ο μέσος χρόνος ανά επανάληψη και ανά δουλειά για τις πρώτες j ομάδες επαναλήψεων του εργάτη P_k, που συμβολίζεται ως $\overline{\mu}_j^k$, δίνεται από τον τύπο:

  $$\overline{\mu}_j^k = \frac{\sum_{l=1}^{j} \frac{t_l^k}{q_l^k \times V_l^k}}{j}.$$

  Αυτός ο τύπος λαμβάνει υπόψη του τον αριθμό των εργασιών στην ουρά εκτέλεσης του P_k αφού έχουμε υποθέσει ότι ο αριθμός εργασιών επιφέρει γραμμική επιβράδυνση στην υπολογιστική ισχύ.
- $\widehat{t}_{j+1}^k = \overline{\mu}_j^k \times q_{j+1}^k \times V_j^k$ είναι ο *προβλεπόμενος χρόνος* του P_k για τον υπολογισμό της ομάδας επαναλήψεων αν το μέγεθος της είναι V_j και υπάρχουν q_{j+1}^k εργασίες στην ουρά εκτέλεσης του.

Επομένως, ο SAS υπολογίζει το μέγεθος της επόμενης ομάδας που αναθέτει στον P_k με βάση τον *ενδεικτικό* χρόνο και το *μέγεθος* και τον *χρόνο υπολογισμού* της προηγούμενης ομάδας του P_k ως εξής:

$$V_{j+1}^k = \frac{t_{Ref}}{\widehat{t}_{j+1}^k} \times V_j^k \tag{8.2}$$

Ο αλγόριθμος SAS περιγράφεται στον ακόλουθο ψευδοκώδικα:

Συντονιστής:

Αρχικοποίηση:
(α) Εγγραφή εργατών. Οι εργάτες αναφέρουν τα VP_k και q_1^k.
(β) Ταξινόμηση των εργατών σε φθίνουσα σειρά της VP_k και ανάθεση της πρώτης ομάδας επαναλήψεων (chunk) σε κάθε εργάτη.

1. Όσο υπάρχουν διαθέσιμες επαναλήψεις:

   (α′) Λήψη αίτησης από τον εργάτη P_k, καταγραφή των q_{j+1}^k και t_j^k.
   (β′) Υπολογισμός του V_{j+1}^k σύμφωνα με τον τύπο 8.2, και ανάθεση στον P_k.

Εργάτης P_k:

Αρχικοποίηση:
(α) Εγγραφή με τον συντονιστή και αναφορά των VP_k και q_1^k.

1. Αποστολή αίτησης στον συντονιστή, αναφορά του q_k και του χρόνου υπολογισμού της προηγούμενης ομάδας t_j^k.
2. Αναμονή απάντησης: Αν δεν υπάρχουν άλλες διαθέσιμες ομάδες επαναλήψεων, τερματισμός. Αλλιώς λήψη της ομάδας με μέγεθος (V_{j+1}^k) και υπολογισμός.
3. Ανταλλαγή δεδομένων στο *SPs* όπως περιγράφεται στο προηγούμενο κεφάλαιο.
4. Μέτρηση του χρόνου υπολογισμού t_{j+1}^k της ομάδας επαναλήψεων V_{j+1}^k.
5. Επιστροφή στο βήμα 1.

Όταν οι εργάτες κάνουν αίτηση για δουλειά στον συντονιστή, στέλνουν και της μετρήσεις που έχουν κάνει για την προηγούμενη ομάδα επαναλήψεων, έτσι ώστε ο συντονιστής να μπορεί να τις χρησιμοποιήσει στο επόμενο βήμα δρομολόγησης. Αυτό δεν επηρεάζει την απόδοση του συστήματος αφού οι πληροφορίες από τις μετρήσεις δεν έχουν μεγάλο όγκο.

8.3 Στοχαστική μοντελοποίηση του περιβάλλοντος - Κατανομή του εξωγενούς φορτίου

Οι κλασικοί αυτοδρομολογούμενοι (self-scheduling) αλγόριθμοι παράγουν ομάδες επαναλήψεων των οποίων το μέγεθος είναι είτε σταθερό, είτε μειώνεται με σταθερό τρόπο. Οι αλγόριθμοι αυτοί αποδίδουν σχετικά καλά σε αφοσιωμένα συστήματα αλλά

η απόδοση τους είναι περιορισμένη σε μη-αφοσιωμένα συστήματα. Οι προσαρμοστικοί (adaptive) αλγόριθμοι, όπως οι AF και DTSS, αντισταθμίζουν την διακύμανση του εξωγενούς φορτίου χρησιμοποιώντας μεταβαλλόμενα βάρη κατά τον υπολογισμό της δουλειάς που στέλνουν σε κάθε εργάτη. Οι αλγόριθμοι αυτοί πρέπει να αποδίδουν καλά σε πραγματικά μη-αφοσιωμένα συστήματα. Αφού η διακύμανση του φορτίου σε τέτοια συστήματα δεν είναι ντετερμινιστική, μπορεί να μοντελοποιηθεί σαν μια στοχαστική διεργασία. Για αυτό τον λόγο η απόδοση τον προσαρμοστικών αλγορίθμων πρέπει να ελεγχθεί σε ένα στοχαστικό περιβάλλον.

Υποθέτουμε ότι ο χρόνος ανάμεσα σε διαδοχικές αφίξεις νέων εργασιών στο σύστημα (interarrival time) είναι μια εκθετικά κατανεμημένη τυχαία μεταβλητή, δηλαδή η διαδικασία άφιξης των εργασιών ακολουθεί την κατανομή Poisson. Αν ο ρυθμός αφίξεων είναι λ, τότε η πιθανότητα ότι η επόμενη εργασία θα έρθει μετά από χρόνο x δίνεται ως:

$$1 - e^{-\lambda x} \tag{8.3}$$

όπου $x \geq 0$ [33]. Υποθέτουμε επίσης ότι ο χρόνος ζωής (lifetime) κάθε εργασίας ακολουθεί της ίδια κατανομή, με ρυθμό εξυπηρέτησης μ. Δίνοντας διαφορετικές τιμές στα λ και μ, μπορεί να εξομοιωθεί ένα γρήγορα ή αργά μεταβαλλόμενο φορτίο. Σε γενικές γραμμές όσο μεγαλύτερος είναι ο ρυθμός αφίξεων και ο χρόνος ζωής των εργασιών, τόσο περισσότερες εργασίες συσσωρεύονται στο σύστημα, ενώ χαμηλός ρυθμός αφίξεων και μικρός χρόνος ζωής οδηγεί σε μικρότερο αριθμό εργασιών και άρα χαμηλό εξωγενές φορτίο.

Για να εξομοιώσουμε το φορτίο στο σύστημα εκτελούμε σε κάθε εργάτη μια γεννήτρια τυχαίων διεργασιών, δηλαδή ένα πρόγραμμα που δημιουργεί νέες διεργασίες σύμφωνα με τα παραπάνω.

8.4 Πειραματική αξιολόγηση

Πειραματική διάταξη. Η υλοποίηση του SAS στηρίζεται στο πλαίσιο κατανεμημένου προγραμματισμού που προσφέρεται από την έκδοση `mpich.1.2.6` της βιβλιοθήκης ανταλλαγής μηνυμάτων (MPI) (Pachecho, 1997), και έγινε σε γλώσσα C με την χρήση της έκδοση 1.2.6 του μεταγλωττιστή `gcc`.

Τα πειράματα διεξάχθηκαν σε ένα ετερογενές κατανεμημένο σύστημα που αποτελείτε από 7 πολυεπεξεργαστικούς κόμβους (2 επεξεργαστικά στοιχεία ανά κόμβο), με έναν από αυτούς να αναλαμβάνει τον ρόλο του συντονιστή. Πιο συγκεκριμένα χρησιμοποιήσαμε: (α) 4 4 Intel Pentiums III 800MHz με 256MB RAM που ονομάζονται twins, και τα οποία διαθέτουν $VP_k = 0.5$ (ένα από αυτά στον ρόλο του συντονιστή)· (β) 3 Intel Pentiums III 1266MHz με 1GB RAM τα οποία ονομάζονται zealots, με $VP_k = 1$· και (γ) 5 Intel Pentiums III 500MHz με 512MB RAM που ονομάζονται kids και έχουν $VP_k = 0.33$. Η εικονική δύναμη του κάθε τύπου υπολογιστή καθορίσθηκε από την αναλογία των χρόνων εκτέλεσης ενός μετροπρογράμματος σε κάθε τύπο υπολογιστή σε σχέση με τους υπόλοιπους. Το δίκτυο διασύνδεσης ήταν 100Mbit fast Ethernet(switched). Στα πειράματα χρησιμοποιήθηκε η ακόλουθη διάταξη (machinefile): *zealot1, twin2, twin2, zealot1, zealot2, twin3, twin3, zealot2,*

Πίνακας 8.1: ρυθμοί άφιξης και χρόνων ζωής

	arrival rate λ	lifetime rate μ
γρήγορο	10	10
μέσο	3.33	3.33
αργό	2	2

Πίνακας 8.2: Ποσοστιαία διαφορά χρόνων εκτέλεσης (βελτίωση) του SAS προς τον $DTSS$ για τον υπολογισμό του Floyd-Steinberg

m	4	8	12
γρήγορο	26.60%	20.35%	10.23%
μέσο	12.24%	24.54%	15.38%
αργό	25.80%	30.17%	14.00%

zealot4, twin4, twin4, zealot4. Τα ονόματα των κόμβων δίνονται δυο φορές αφού έχουμε δυο επεξεργαστές ανά κόμβο. Οι μετρήσεις δίνονται για $m = 4, 8$, και 12 εργάτες. Αν παρατηρήσουμε την παραπάνω διάταξη, με αυτό το βήμα στην αύξηση του αριθμού των επεξεργαστών περιμένουμε ένα σχετικά ομαλό γράφημα αφού αυξάνουμε σταθερά κάθε φορά την υπολογιστική ισχύ στο σύστημα.

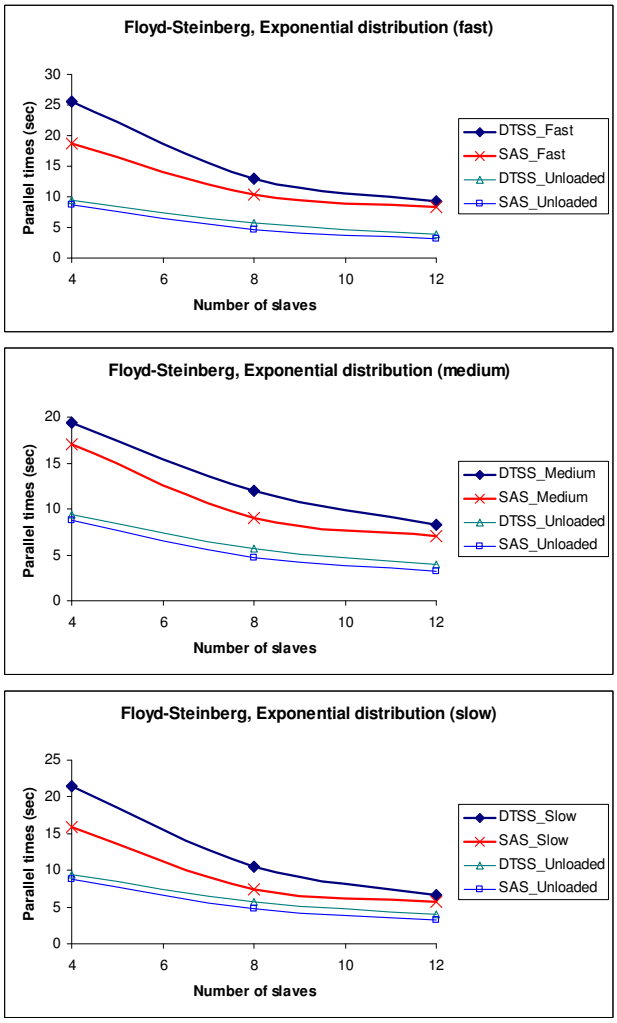
Όπως έχουμε αναφέρει νωρίτερα υποθέτουμε ότι το εξωγενές φορτίο ακολουθεί μια εκθετική κατανομή. Επιλέγουμε χρόνους άφιξης και ζωής των εξωτερικών εργασιών, έτσι ώστε να εξομοιώσουμε γρήγορα, μέσα και αργά μεταβαλλόμενο φορτίο στο σύστημα. Όταν λέμε γρήγορα μεταβαλλόμενο φορτίο εννοούμε ότι θα αλλάξει αρκετές φορές τιμή μέσα σε ένα βήμα δρομολόγησης. Οι ρυθμοί αυτοί δίνονται στον πίνακα 8.1.

8.4.1 Αποτελέσματα.

Συγκρίναμε την απόδοση του SAS με αυτή του $DMPS(DTSS)$ για την εφαρμογή Floyd-Steinberg, χωρίς εξωγενές φορτίο (αφοσιωμένο σύστημα) και στην συνέχεια για αργά, μεσαία και γρήγορα μεταβαλλόμενο φορτίο, σύμφωνα με τους ρυθμούς που δίνονται στον πίνακα 8.1. Η σύγκριση με τον $DMPS(DTSS)$ γίνεται γιατί ο $DMPS(DTSS)$ μπορεί να χειρισθεί με επιτυχία, προβλήματα με εξαρτήσεις σε μη-αφοσιωμένα συστήματα αφού και ο ίδιος είναι ένας προσαρμοστικός αλγόριθμος.

Στο σχήμα 8.1 δίνονται οι μέσοι παράλληλοι χρόνοι εκτέλεσης των δυο αλγορίθμων, για τους τρεις τύπους εξωγενούς φορτίου. Σε κάθε περίπτωση δίνεται και ο χρόνος παράλληλης εκτέλεσης των δυο αλγορίθμων στο αφοσιωμένο σύστημα, σαν ένδειξη του ορίου της απόδοσης που μπορεί να επιτύχει κάθε αλγόριθμος. Σε κάθε περίπτωση και για τους δυο αλγόριθμους, το διάστημα συγχρονισμού είναι 100 σημεία ($H = 100$). Η ποσοστιαία βελτίωση του SAS προς τον $DMPS(DTSS)$ συνοψίζεται στον πίνακα 8.2.

Για της επιλογές στον ρυθμό μεταβολής του εξωγενούς φορτίου που δοκιμάστη-



Σχήμα 8.1: Παράλληλοι χρόνοι των SAS και $DTSS$ για την εκτέλεση του υπολογισμού Floyd-Steinberg

καν, τα πειραματικά αποτελέσματα δείχνουν ότι ο SAS πάντα αποδίδει καλύτερα από τον $DMPS(DTSS)$. Ο SAS είναι μια πρώτη προσπάθεια προσαρμογής δυναμικού αλγορίθμου, για βρόχους με εξαρτήσεις, σε στοχαστικά περιβάλλοντα. Τα αποτελέσματα που συγκεντρώθηκαν δεν αποκάλυψαν την ακριβή επίδραση ανάμεσα στον ρυθμό μεταβολής του φορτίου και της απόδοσης του αλγορίθμου, είναι πάντως εμφανές ότι λαμβάνοντας υπόψη τους χρόνους υπολογισμού των προηγούμενων εργασιών μπορούμε να κάνουμε καλύτερες εκτιμήσεις για την διακύμανση του φορτίου στο σύστημα, δίνοντας έτσι την δυνατότητα για καλύτερη εξισορρόπηση του υπολογιστικού φορτίου. Ακόμα ο SAS υπερτερεί του $DMPS(DTSS)$ και στην περίπτωση του αφοσιωμένου συστήματος. Αυτό οφείλεται στο γεγονός ότι οι εκτιμήσεις για την εικονική υπολογιστική ισχύ των επεξεργαστών του συστήματος μπορεί να μην είναι

απόλυτα ακριβής, ο SAS από την άλλη μεριά χρησιμοποιεί τις χρονικές μετρήσεις για να περιορίσει αυτό το πιθανό σφάλμα εκτίμησης.

Οι δοκιμές έγιναν σε ένα περιβάλλον με λίγους επεξεργαστές, αποκάλυψαν ωστόσο ότι ο SAS είναι ένας πολλά υποσχόμενος αλγόριθμος. Βλέπουμε ότι και σε αυτόν τον αλγόριθμο, όπως άλλωστε και στους αλγορίθμους που περιγράφονται στα προηγούμενα κεφάλαια, η κλιμάκωση (scalability) του συστήματος είναι περιορισμένη, αφού η αύξηση της απόδοσης από ένα αριθμό επεξεργαστών και πάνω, δεν ακολουθεί την αύξηση του αριθμού των επεξεργαστών. Αυτό οφείλεται εν μέρη στον περιορισμό που θέτει το μοντέλο εργάτη συντονιστή το οποίο ακολουθούν όλοι οι αλγόριθμοι.

8.5 Συμπεράσματα

Σε αυτό το κεφάλαιο παρουσιάστηκε ένας νέος δυναμικός αλγόριθμος δρομολόγησης (SAS)για μη-αφοσιωμένες συστοιχίες υπολογιστών. Το μοντέλο εφαρμογών είναι και σε αυτή την περίπτωση οι φωλιασμένοι βρόχοι που περιέχουν ομοιόμορφες εξαρτήσεις. Η απόδοση του SAS συγκρίθηκε με τον αλγόριθμο $DMPS(DTSS)$ κάτω από στοχαστικά μεταβαλλόμενο εξωγενές φορτίο. Ο χρόνος άφιξης και ζωής του εξωγενούς φορτίου μοντελοποιήθηκε με την χρήση της εκθετικής κατανομής. Ο αλγόριθμος αυτού του κεφαλαίου παρουσιάζει βελτιωμένη απόδοση λόγο της ικανότητας του προσαρμογής στις μεταβολές του εξωγενούς φορτίου. Τα πειραματικά αποτελέσματα αποδεικνύουν ότι ο νέος αυτός αλγόριθμος είναι πιο αποδοτικός από αυτούς που περιγράφονται στην βιβλιογραφία. Ωστόσο έγιναν εμφανείς οι περιορισμοί στην κλιμάκωση της απόδοση του συστήματος σε σχέση με την αύξηση του αριθμού των εργατών, γεγονός που σχετίζεται με το μοντέλο συντονιστή-εργάτη. Τον περιορισμό αυτόν θα μελετήσουμε στο επόμενο κεφάλαιο όπου παρουσιάζεται ένας κατανεμημένος δυναμικός αλγόριθμος.

Κεφάλαιο 9

Κατανεμημένο σχήμα δυναμικής δρομολόγησης

Η ανισότητα στην κατανομή του υπολογιστικού φορτίου είτε στα ομοιογενή ή στα ετερογενή κατανεμημένα συστήματα μπορεί να βλάψει σημαντικά την απόδοση. Οι δυναμικές μέθοδοι εξισορρόπησης φορτίου που έχουμε παρουσιάσει στα προηγούμενα κεφάλαια είναι βασισμένα στο μοντέλο συντονιστή εργάτη, στο οποίο ένας μοναδικός κόμβος συντονιστής λαμβάνει όλες τις αποφάσεις και καταρτίζει το σχέδιο δρομολόγησης, βασισμένος συνήθως σε πληροφορίες που του παρέχουν οι εργάτες. Το μοντέλο συντονιστή-εργάτη έχει δύο μειονεκτήματα: (1) οι πληροφορίες στις οποίες στηρίζει τις αποφάσεις του ο συντονιστής μπορεί να μην είναι έγκυρες, ειδικά στην περίπτωση που τα χαρακτηριστικά του συστήματος μεταβάλλονται γρήγορα, (2) η δυνατότητα κλιμάκωσης του συστήματος, με την αύξηση του αριθμού εργατών είναι περιορισμένη. Σε αυτό το κεφάλαιο παρουσιάζεται μια κατανεμημένη αλγοριθμική προσέγγιση, στην οποία υπάρχουν μόνο εργάτες, οι οποίοι είναι ικανοί να λάβουν μόνοι τους τις αποφάσεις σχετικά με την δρομολόγησης των εργασιών τους, εξαλείφοντας έτσι τα παραπάνω μειονεκτήματα.

9.1 Εισαγωγή

Όλες οι μέθοδοι δρομολόγησης πού περιγράφονται στα προηγούμενα κεφάλαια βασίζονται στο μοντέλο συντονιστή-εργάτη, στο οποίο ο συντονιστής είναι αποκλειστικά υπεύθυνος για την δρομολόγηση των εργασιών, βασιζόμενος συνήθως σε πληροφορίες που λαμβάνει κατά διαστήματα από τους εργάτες. Οι πληροφορίες αυτές αποστέλλονται από τους εργάτες μετά την ολοκλήρωση της εργασίας που τους έχει ανατεθεί και αξιοποιούνται από τον συντονιστή στα επόμενα βήματα δρομολόγησης. Σε συστήματα με ταχέως μεταβαλλόμενα χαρακτηριστικά, όπως για παράδειγμα σε συστήματα με μεταβαλλόμενο φορτίο, υπάρχει ο κίνδυνος οι πληροφορίες που επιστρέφουν οι εργάτες να χάνουν την εγκυρότητα τους πριν χρησιμοποιηθούν από τον συντονιστή. Ο κίνδυνος αυτός είναι μεγαλύτερος σε μεγάλα συστήματα τα οποία μπορούν να περιλαμβάνουν εκατοντάδες ή χιλιάδες εργάτες, όπως για παράδειγμα σε συστοιχίες

πολυεπεξεργαστών (SMP clusters) ή σε εφαρμογές του υπολογιστικού πλέγματος (Grid sites).

Επιπροσθέτως, το σχήμα συντονιστή-εργάτη είναι γνωστό μπορεί να παρουσιάσει περιορισμούς στην δυνατότητα κλιμάκωσης. Όσο αυξάνεται ο αριθμός των εργατών, η ταυτόχρονη αύξηση των απαιτήσεων πρόσβασης στον κεντρικό κόμβο-συντονιστή συνήθως αποτελεί ανασχετικό παράγοντα στην κλιμάκωση της απόδοσης. Μια φυσική επέκταση είναι το ιεραρχικό μοντέλο συντονιστή-εργάτη [5], το οποίο βελτιώνει την δυνατότητα κλιμάκωσης χρησιμοποιώντας περισσότερους συντονιστές. Ωστόσο, είναι σαφώς προτιμότερη η χρησιμοποίηση μιας κατανεμημένης προσέγγισης στην οποία δεν υπάρχει η ανάγκη για ένα ή περισσότερους συντονιστές. Σε αυτή την προσέγγιση, οι αποφάσεις δρομολόγησης λαμβάνονται από τους ίδιους τους εργάτες. Γίνεται χρήση των πιο πρόσφατων και έγκυρων πληροφοριών σχετικά με την κατάσταση του συστήματος οδηγώντας έτσι σε βελτίωση της συνολικής απόδοσης σε σχέση με τις προσεγγίσεις που βασίζεται στο μοντέλο συντονιστή εργάτη.

Σε αυτό το κεφάλαιο περιγράφεται ένας πλήρως κατανεμημένος δυναμικός αλγόριθμος δρομολόγησης για προβλήματα που περιέχουν βρόχους με εξαρτήσεις. Ο αλγόριθμος αυτός έχει αρκετές ομοιότητες με τον αλγόριθμο SAS ο οποίος περιγράφηκε στο προηγούμενο κεφάλαιο. Πιο συγκεκριμένα χρησιμοποιεί πληροφορίες σχετικά με την ουρά εκτέλεσης, όπως επίσης και χρονικές μετρήσεις κατά την διάρκεια της χρονοδρομολόγησης. Ωστόσο διαφέρει από τον SAS στο ότι κάθε εργάτης είναι υπεύθυνος για τις δικές του αποφάσεις δρομολόγησης, καταργώντας την ανάγκη ύπαρξης ενός κεντρικού κόμβου συντονιστή. Ο αλγόριθμος αυτός ονομάζεται DAS (Distributed Adaptive Scheduling ή Κατανεμημένη Προσαρμοστική Δρομολόγηση).

Για να αξιολογήσουμε την απόδοση του κατανεμημένου αλγορίθμου, τον συγκρίνουμε με δύο αλγορίθμους που βασίζονται στο σχήμα συντονιστη-εργάτη: τον SAS και τον $\mathcal{S}-AWF$. Ο τελευταίος είναι ένας νέος αλγόριθμος, ο οποίος βασίζεται στον αλγόριθμο AWF ([42]) και στον οποίο έχει εφαρμοστεί ο μηχανισμός συγχρονισμού $\mathcal{S}$, όπως περιγράφεται στο Κεφάλαιο 6. Σαν παράδειγμα εφαρμογής χρησιμοποιείται ένας αλγόριθμος στοίχισης DNA ([103]). Πραγματοποιούμε στοίχιση δύο ακολουθιών μήκους 1024000 στοιχείων, σε μια συστοιχία πολυεπεξεργαστών SMP cluster με 128 επεξεργαστές (16 8-way nodes).

Η ανάλυση των αποτελεσμάτων επιβεβαιώνει την υπεροχή της κατανεμημένης προσέγγισης έναντι των δύο συγκεντρωτικών για τις συνθήκες του πειράματος, σε όρους απόδοσης δηλαδή παραλλήλων χρόνων, αλλά και σχετικά με την ικανότητα κλιμάκωσης της απόδοσης όσο αυξάνεται ο αριθμός των εργατών.

9.2 Εκτελέσεις σωλήνωσης και εξισορρόπηση υπολογιστικού φορτίου

Όπως έχει ήδη αναφερθεί σε προηγούμενες ενότητες, όταν οι εφαρμογές που περιέχουν βρόχους με εξαρτήσεις δρομολογούνται με την χρήση δυναμικών αλγορίθμων, οδηγούνται σε εκτέλεση σωλήνωσης (pipelined execution). Ένα ακόμα παράδειγμα διαμερισμού ενός χώρου επαναλήψεων, όπως επίσης και της σωλήνωσης που προκύπτει από αυτό τον διαμερισμό παρουσιάζεται στο σχήμα 9.1. Θεωρούμε ότι η σωλήνωση

αυτή εκτελείται σε ένα σύστημα που αποτελείται από ετερογενείς επεξεργαστές, οι οποίοι είναι συνδεδεμένοι μέσω ενός ομοιογενούς δικτύου. Θεωρούμε επίσης ότι η ανομοιογένεια των επεξεργαστών οφείλεται στο διαφορετικό πρότυπο (pattern) πού ακολουθεί το φορτίο του κάθε επεξεργαστή. Έτσι καταλήγουμε σε ένα σύστημα με επεξεργαστές που διαθέτουν διαφορετική διαθέσιμη υπολογιστική ισχύ.

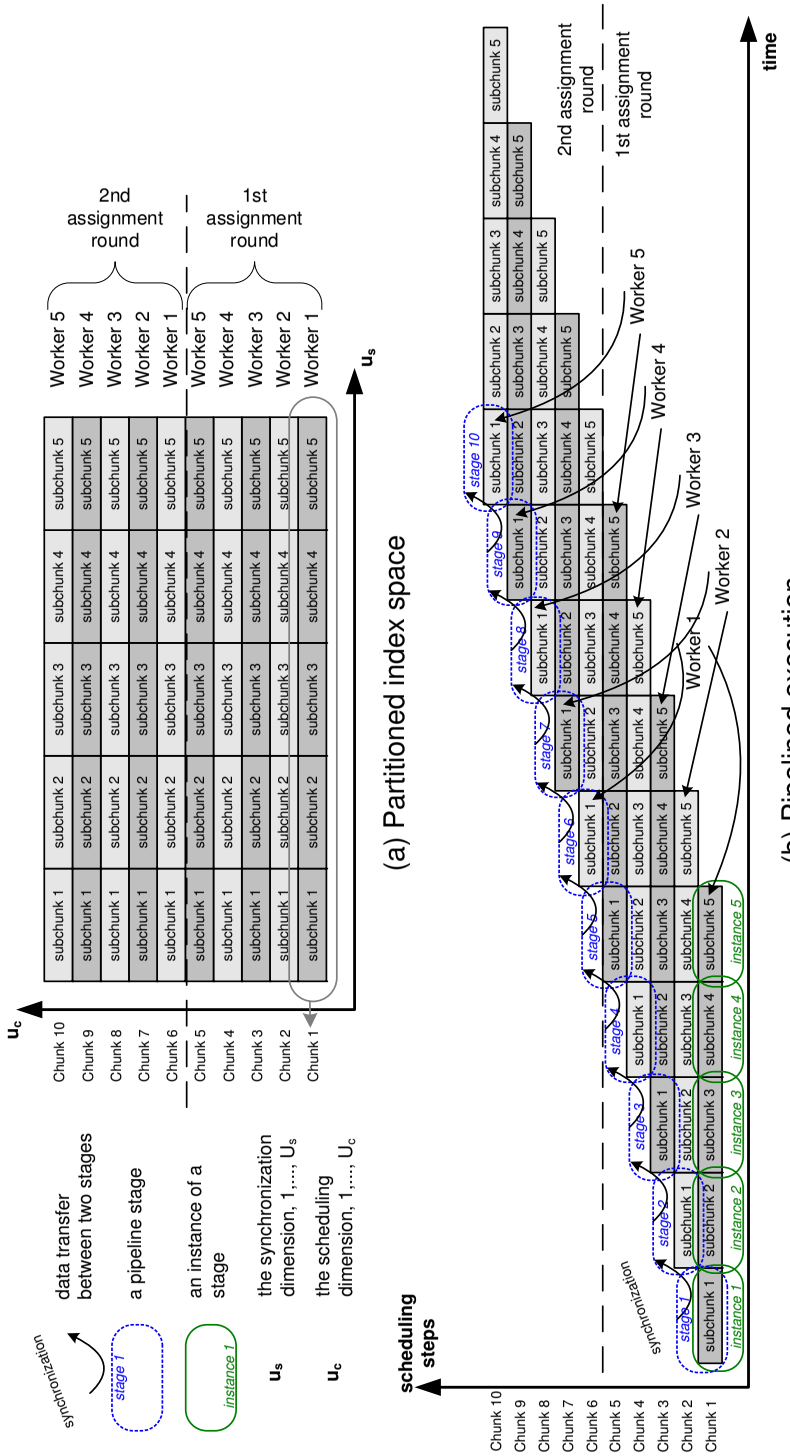
Η κατανομή του υπολογιστικού φορτίου είναι ιδανική, δηλαδή ανταποκρίνεται απόλυτα στα χαρακτηριστικά (διαθέσιμη ισχύ) των επεξεργαστών, όταν οι επεξεργαστές ολοκληρώνουν την εκτέλεση των εργασιών που έχουν αναλάβει την ίδια χρονική στιγμή. Σε αυτή την περίπτωση οι επεξεργαστές μπορούν να ανταλλάξουν άμεσα δεδομένα και να προχωρήσουν στην εκτέλεση νέων εργασιών. Κάθε απόκλιση από την ιδανική αυτή περίπτωση προκαλεί καθυστερήσεις, δηλαδή άεργους χρόνους (idle times), μέσα στους οποίους οι επεξεργαστές περιμένουν για δεδομένα χωρίς να εκτελούν κάποια χρήσιμη εργασία (βλέπε Σχήμα 9.2). Είναι προφανές, ότι για να ελαχιστοποιηθεί ο συνολικός χρόνος εκτέλεσης της σωλήνωσης πρέπει να ελαχιστοποιηθεί ο συνολικός άεργος χρόνος, ο οποίος δημιουργείται μεταξύ διαδοχικών σταδίων ή στιγμιότυπων της σωλήνωσης.

Το Σχήμα 9.2 απεικονίζει ένα στιγμιότυπο εκτέλεσης δύο περιπτώσεων, στις οποίες δύο ομάδες επαναλήψεων έχουν ανατεθεί σε δύο επεξεργαστές και η εκτέλεση εξελίσσεται (α) χωρίς άεργο χρόνο, (β) με άεργο χρόνο. Στην πρώτη περίπτωση, και οι δύο επεξεργαστές μπορούν να εκτελέσουν τον ίδιο αριθμό επαναλήψεων στον ίδιο χρόνο. Ο επεξεργαστής 1 ολοκληρώνει τη υποομάδα 1 την χρονική στιγμή t_1 και στέλνει τα ενδιάμεσα αποτελέσματα στον επεξεργαστή 2. Ο επεξεργαστής 1 ξεκινάει την εκτέλεση της υποομάδας 2 την ίδια χρονική στιγμή που ο επεξεργαστής 2 ξεκινάει τον υπολογισμό της δικής του υποομάδας 1. Αφού οι δύο επεξεργαστές έχουν ακριβός την ίδια υπολογιστική ισχύ θα ολοκληρώσουν την εκτέλεση την ίδια χρονική στιγμή. Αυτή η απόλυτα συγχρονισμένη εκτέλεση δεν προκαλεί καμία καθυστέρηση στον επεξεργαστή 2. Στην δεύτερη περίπτωση, ο επεξεργαστής 2 είναι πιο γρήγορος από τον επεξεργαστή 1. Αν ξεκινήσουν την εκτέλεση των αντίστοιχων υποομάδων τους την ίδια χρονική στιγμή ο επεξεργαστής 2 θα ολοκληρώσει την υποομάδα 1 νωρίτερα από ότι ο επεξεργαστής 1 την υποομάδα 2 και έτσι θα αναγκαστεί περιμένει ανενεργός.

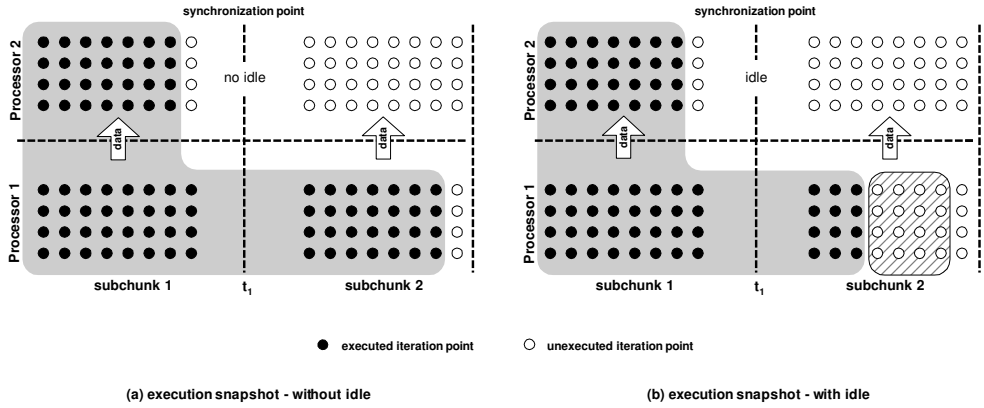
Ο άεργος χρόνος της δεύτερης περίπτωσης μπορεί να αποφευχθεί με την ανάθεση λιγότερων επαναλήψεων στον επεξεργαστή 1 ή περισσότερων επαναλήψεων στον επεξεργαστή 2 αντίστοιχα. Στην περίπτωση ενός στατικού συστήματος μια ικανοποιητική κατανομή του φορτίου είναι εφικτή. Σε συστήματα με μεταβαλλόμενο φορτίο, η απόδοση του αλγορίθμου δρομολόγησης εξαρτάται από την ικανότητα του να προσαρμόζεται στις μεταβολές, όπως είδαμε και στο προηγούμενο κεφάλαιο με τον αλγόριθμο SAS.

9.3 Μοντέλο Συντονιστή-εργάτη και το κατανεμημένο μοντέλο

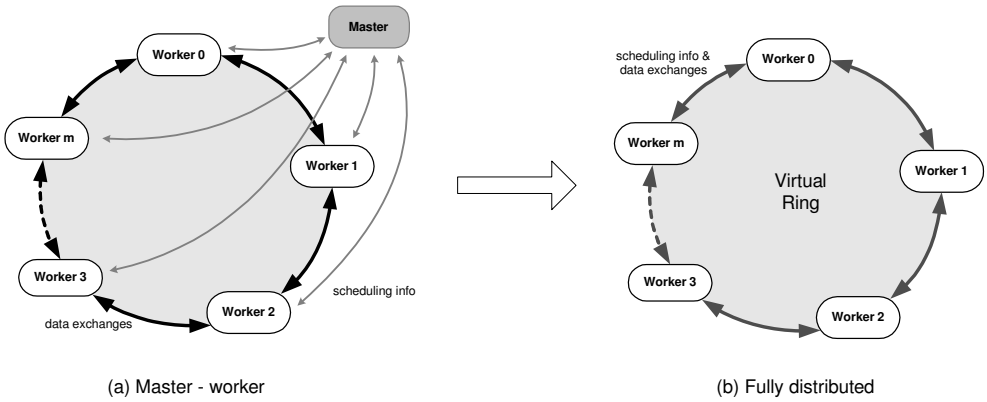
Το σχήμα 9.3 (α) απεικονίζει το μοντέλο συντονιστή-εργάτη, όπως αυτό προτάθηκε στο [102] για εφαρμογές που περιέχουν φωλιασμένους βρόχους με εξαρτήσεις. Σε αυτό το μοντέλο μπορούμε να διακρίνουμε δύο τύπους μηνυμάτων: αυτά που περιέ-



Σχήμα 9.1: *(α) Ένας χώρος δεικτών χωρισμένος σε 10 ομάδες επαναλήψεων που αποτελούνται από 5 υποομάδες. Η ανάθεση γίνεται σε 5 εργάτες μέσα σε 2 γύρους. (β) Η αντίστοιχη χοδρόκοκκη σωλήνωση που προκύπτει από τον κατα-μερισμό του χώρου δεικτών. Η σωλήνωση έχει 10 στάδια και 5 στιγμιότυπα.*



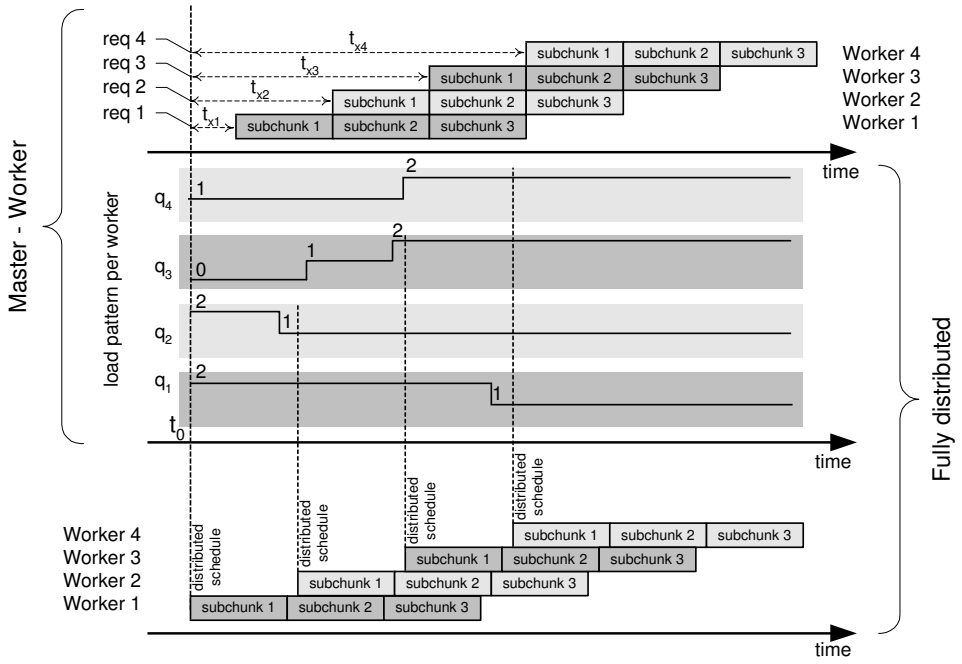
Σχήμα 9.2: *Η ανισοκατανομή του υπολογιστικού φορτίου προκαλεί άεργους χρόνους κατά την εκτέλεση δύο διαδοχικών στιγμιοτύπων (υποομάδων).*



Σχήμα 9.3: *Το μοντέλο εργάτη-συντονιστή και το κατανεμημένο μοντέλο.*

χουν *πληροφορίες δρομολόγησης* τα οποία ανταλλάσσονται μεταξύ του συντονιστή και των εργατών, και αυτά που περιέχουν δεδομένα (μερικά αποτελέσματα), τα οποία ανταλλάσσονται μεταξύ των εργατών.

Σε αυτό το μοντέλο, οι εργάτες στέλνουν αιτήσεις για εργασία στον συντονιστή ο οποίος είναι υπεύθυνος για την κατανομή των εργασιών. Από την στιγμή που ο εργάτης αποστέλλει την αίτηση στον συντονιστή, μέχρι την στιγμή που αρχίζει τον υπολογισμό της ομάδας επαναλήψεων μεσολαβεί ένα χρονικό διάστημα, το οποίο συμβολίζουμε t_{xk} (βλέπετε Σχήμα 9.4). Κατά την διάρκεια του χρόνου t_{xk}, ο εργάτης k περιμένει για την απάντηση του συντονιστή, καθώς επίσης και για την λήψη μερικών αποτελεσμάτων από τον προηγούμενο εργάτη $k-1$. Σε ένα σύστημα με μεταβλητό φορτίο, υπάρχει πιθανότητα το φορτίο να μεταβληθεί μέσα σε αυτό το διάστημα. Το γεγονός αυτό προκαλεί μείωση της απόδοσης αφού ο συντονιστής βασίζεται σε άκυρες πληροφορίες προκειμένου να δρομολογήσει τις εργασίες. Ο κίνδυνος αυτός αυξάνεται



Σχήμα 9.4: *Σύγκριση μεταξύ του σχεδίου δρομολόγησης που προκύπτει από την προσέγγιση εργάτη-συντονιστή και από την πλήρως κατανεμημένη προσέγγιση.*

με τον αριθμό των εργατών, αφού ο χρόνος t_{xk} είναι ανάλογος του αριθμού των εργατών.

Η λύση του παραπάνω προβλήματος βρίσκεται στην εφαρμογή ενός πλήρως αποκεντρωμένου μοντέλου όπως αυτού που περιγράφεται στην συνέχεια. Στο πλήρως αποκεντρωμένο μοντέλο που παρουσιάζεται στα σχήματα 9.3 (b) και 9.4 δεν υπάρχει συντονιστής αλλά μόνο εργάτες. Στις κατηγορίες εφαρμογών που εξετάζουμε, η ύπαρξη των εξαρτήσεων δεδομένων υπαγορεύει τον σχηματισμό ενός εικονικού δακτυλίου στο οποίο είναι διατεταγμένοι οι εργάτες όπως φαίνεται στο σχήμα 9.3 (a). Η διάταξη αυτή παραμένει αναλλοίωτη καθ όλη την διάρκεια της εκτέλεσης, καθώς κάθε εργάτης προωθεί μερικά αποτελέσματα στον επόμενο προκειμένου να ικανοποιηθούν οι εξαρτήσεις δεδομένων. Οι εργάτες μπορούν να χρησιμοποιούν την διάταξη αυτή για να ανταλλάσσουν πληροφορίες δρομολόγησης εξαλείφοντας τον λόγο ύπαρξης του κεντρικού κόμβου συντονιστή. Έτσι στο κατανεμημένο μοντέλο, οι εργάτες είναι υπεύθυνοι και για τους πραγματικούς υπολογισμούς αλλά και για την δρομολόγηση.

Σε αυτό το μοντέλο, μόλις ένας εργάτης είναι έτοιμος, λαμβάνει τις πληροφορίες δρομολόγησης καθώς και τα μερικά δεδομένα από τον προηγούμενο εργάτη χωρίς να μεσολαβήσει κάποιος χρόνος t_{xk} όπως στην περίπτωση του μοντέλου συντονιστή-εργάτη. Επομένως ο εργάτης μπορεί να αποφασίσει σχετικά με το μέγεθος της ομάδας επαναλήψεων που θα αναλάβει χρησιμοποιώντας τις πιο πρόσφατες πληροφορίες και

να ξεκινήσει αμέσως τον υπολογισμό. Αυτό το βήμα καλούμε κατανεμημένη δρομολόγηση.

Το σχήμα 9.4 απεικονίζει τα πιθανά οφέλη της κατανεμημένης δρομολόγησης έναντι στο μοντέλο συντονιστή-εργάτη, για μια συγκεκριμένη ακολουθία μεταβολής του φορτίου του συστήματος. Στο μοντέλο συντονιστή-εργάτη, ο συντονιστής λαμβάνει τις πληροφορίες σχετικά με το φορτίο από τους εργάτες την χρονική στιγμή t_0. Ο συντονιστής παράγει το σχέδιο δρομολόγησης βασισμένος σε αυτές τις πληροφορίες και αναθέτει ομάδες επαναλήψεων στους εργάτες. Στην περίπτωση του εργάτη 1, το μέγεθος της ομάδας που έχει αναλάβει ανταποκρίνεται στο πραγματικό φορτίο ωστόσο, στην περίπτωση των εργατών 2,3 και 4 οι πληροφορίες σχετικά με το φορτίο δεν είναι πια έγκυρες και έτσι τα μεγέθη των ομάδων επαναλήψεων που έχουν αναλάβει δεν οδηγούν σε ισοσταθμισμένη εκτέλεση. Αντιθέτως, στο κατανεμημένο μοντέλο, οι εργάτες αποφασίζουν σχετικά με την εργασία που θα εκτελέσουν βασιζόμενοι στις πιο πρόσφατες πληροφορίες.

Το παραπάνω παράδειγμα παρουσιάζει απλά την αρχή λειτουργίας του κατανεμημένου μοντέλου, Στην πραγματικότητα ο αλγόριθμος που περιγράφεται στις επόμενες ενότητες είναι πιο πολύπλοκος καθώς επιτρέπει στους εργάτες, να αναπροσαρμόσουν ως ένα βαθμό το μέγεθος της ομάδας επαναλήψεων κατά την διάρκεια εκτέλεσης του. Με αυτό τον τρόπο επιτυγχάνεται η καλύτερη δυνατή κατανομή φορτίου. Περισσότερες πληροφορίες δίνονται στην ενότητα 9.5.

9.4 Αλγόριθμοι που βασίζονται στο μοντέλο συντονιστή-εργάτη

Στο προηγούμενο κεφάλαιο παρουσιάστηκε ο αλγόριθμος SAS, ο οποίος αποδείχθηκε πολύ αποτελεσματικός για την δρομολόγηση εφαρμογών που περιέχουν εξαρτήσεις σε συστήματα με μεταβαλλόμενο φορτίο. Στην συνέχεια περιγράφουμε συνοπτικά τον αλγόριθμο Synchronized Adaptive Weighting Factoring ($\mathcal{S}$-AWF). Ο αλγόριθμος αυτός βασίζεται στον αλγόριθμο AWF ο οποίος είχε αρχικά προταθεί στο [42], για την δρομολόγηση βρόχων χωρίς εξαρτήσεις οι οποίοι έχουν ακανόνιστους χρόνους εκτέλεσης. Στον αλγόριθμο AWF προστέθηκε ο μηχανισμός συγχρονισμού $\mathcal{S}$ που περιγράφεται στην ενότητα 6.2, προκειμένου να μπορεί να εφαρμοστεί σε προβλήματα που περιέχουν εξαρτήσεις. Και ο αλγόριθμος SAS αλλά και ο $\mathcal{S}$-AWF βασίζονται στο μοντέλο συντονιστή-εργάτη και θα χρησιμοποιηθούν στην συνέχεια στην αξιολόγηση της απόδοσης του κατανεμημένου μοντέλου σε σχέση αυτό του συντονιστή-εργάτη.

Ο αρχικός αλγόριθμος AWF αναθέτει σειρές των m ομάδων επαναλήψεων σε m εργάτες. Το μέγεθος V_i της i-στη σειράς είναι μια σταθερή αναλογία των υπολειπόμενων επαναλήψεων. Το μέγεθος ομάδας V_i^k, που ανήκει στην σειρά i, και ανατίθεται στον εργάτη P_k, σταθμίζεται σύμφωνα με την υπολογιστική του δύναμη. Οι τιμές των βαρών των εργατών ανανεώνονται μετά την ανάθεση κάθε σειράς ομάδων επαναλήψεων και είναι βασισμένα στην συνολική απόδοση των εργατών σε όλες τις προηγούμενες αναθέσεις. Μετά τον υπολογισμό κάθε ομάδας επαναλήψεων, οι εργάτες στέλνουν στον συντονιστή τους χρόνους εκτέλεσης και αυτός υπολογίζει τα βάρη για όλους τους εργάτες όπως φαίνεται παρακάτω.

Η σταθμισμένη μέση απόδοση (WAP) όλων των αναθέσεων υπολογίζονται από τον συντονιστή για κάθε εργάτη k ως:

$$WAP_k = \frac{\sum_{i=1}^{j} t_i^k \times i}{\sum_{i=1}^{j} V_i^k \times i} \tag{9.1}$$

όπου

$$V_{i+1}^k = V_{i+1} \times WP_k \tag{9.2}$$

$$V_{i+1} = \frac{R_{i+1}}{2m} \tag{9.3}$$

ο αριθμός των υπολειπόμενων σημείων είναι: $R_{i+1} = R_i - \sum_{k=1}^{m} V_i^k$, $R_0 = U_c$, $i = 1, \ldots, j$. t_i^k είναι ο χρόνος υπολογισμού της ομάδας επαναλήψεων V_i^k από τον εργάτη P_k.

Ο μέσος χρόνος υπολογισμού ($AWAP$) και το βάρος αναφοράς όλων των εργατών είναι:

$$AWAP = \frac{\sum_{k=1}^{m} WAP_k}{m} \tag{9.4}$$

$$RWP_k = \frac{AWAP}{WAP_k} \tag{9.5}$$

Το RWP_k πρέπει να κανονικοποιηθεί σύμφωνα με τον αριθμό των εργατών m για να παράξει το πραγματικό βάρος, γιατί το άθροισμα όλων των βαρών των εργατών πρέπει να είναι:

$$\sum_{k=1}^{m} WP_k = m \tag{9.6}$$

Το συνολικό βάρος αναφοράς (TRW) δίνεται από:

$$TRW = \sum_{k=1}^{m} RWP_k \tag{9.7}$$

και το πραγματικό βάρος του εργάτη P_k υπολογίζεται μετά την κανονικοποίηση του βάρους αναφοράς του προς το συνολικό βάρος αναφοράς όλων των εργατών:

$$WP_k = \frac{RWP_k \times m}{TRW} \tag{9.8}$$

Ο λειτουργία του αλγορίθμου $\mathcal{S} - AWF$ για εφαρμογές με εξαρτήσεις περιγράφεται παρακάτω.

Συντονιστής:

1) Εγγραφή εργατών.

2) Υπολογισμός του μεγέθους της πρώτης σειράς ομάδων επαναλήψεων V_1 με την χρήση του τύπου 9.3.

3) **Όσο** υπάρχουν διαθέσιμες επαναλήψεις: {

Λήψη αίτησης από τον εργάτη P_k.

Αύξηση του αριθμού αιτήσεων $\#reqs$.

Αν(i **!= 1**){ * *Δεν είμαστε στην πρώτη σειρά ομάδων υπολογισμού* *\

λήψη της σταθμισμένης μέσης απόδοσης WAP_k του εργάτη P_k.

Αν($\#reqs \ mod \ m = 0$){ * *Αρχή νέας σειράς* *\

Υπολόγισε:

- το μέγεθος V_i, της νέας σειράς, με την χρήση του τύπου 9.3.
- την μέση ταχύτητα εκτέλεσης $AWAP$.
- το βάρος αναφοράς RWP_k για κάθε εργάτη.
- το συνολικό βάρος αναφοράς TRW για όλους τους εργάτες.

}

Υπολόγισε το πραγματικό βάρος WP_k για τον εργάτη P_k.

Υπολόγισε το μέγεθος της σταθμισμένης ομάδας επαναλήψεων V_i^k του P_k για την τρέχουσα σειρά με την χρήση του τύπου 9.2.

} **Αλλιώς** {

Ανάθεση ομάδας μεγέθους V_1 στον P_k.

Αν($\#reqs = m$) * *Τέλος της πρώτης σειράς* *\

Αύξηση του i

}

Ενημέρωση του αριθμού των διαθέσιμων επαναλήψεων R_{i+1}.

} 4) Τερματισμός

Εργάτης P_k :

1) Εγγραφή με τον συντονιστή

2) Αποστολή αίτησης εργασίας στον συντονιστή και αναφορά της WAP_k αν είναι διαθέσιμη.

3) Αναμονή απάντησης.

4) **Αν (δεν υπάρχουν άλλες ομάδες επαναλήψεων)** {

Τερματισμός

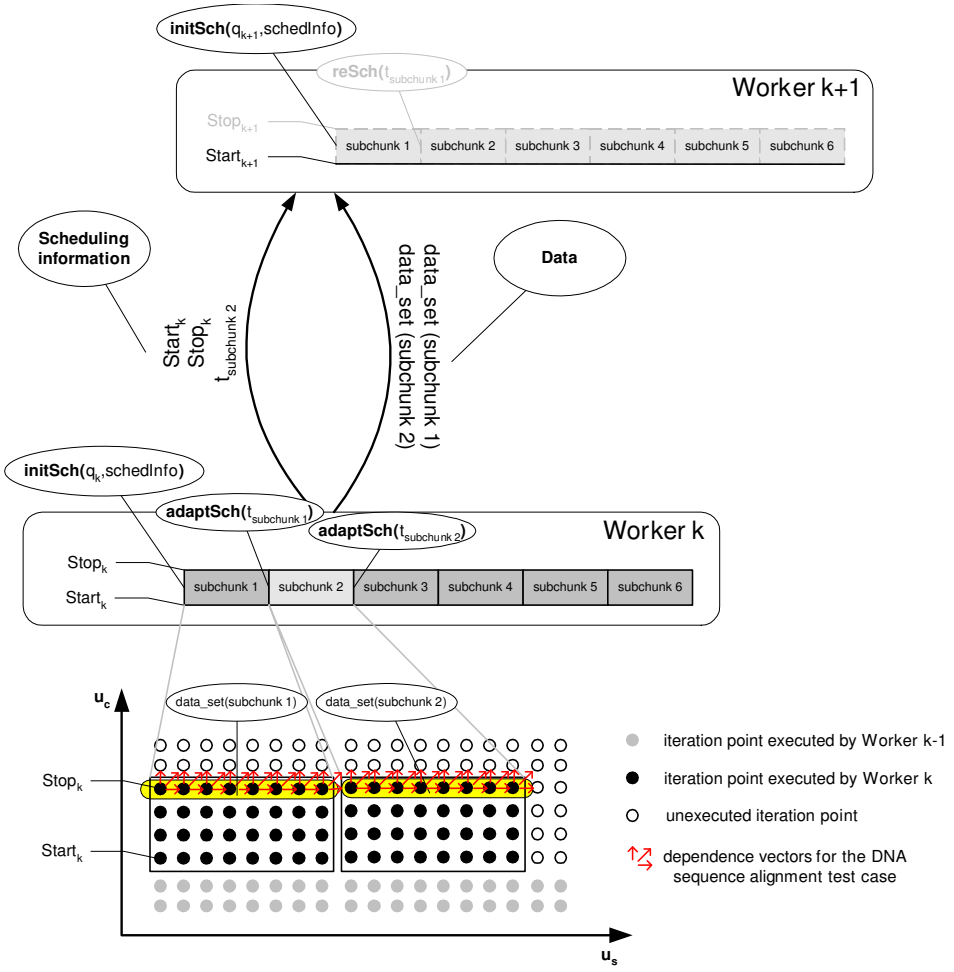
} **Αλλιώς** {

Λήψη και εκτέλεση της επόμενης ομάδας εργασιών μεγέθους V_i^k

Ανταλλαγή δεδομένων σε κάθε SPs * *Συγχρονισμός* *\

} 5) Υπολογισμός της WAP_k.

6) επιστροφή στο βήμα 2.



Σχήμα 9.5: *Ανταλλαγή μηνυμάτων μεταξύ δύο γειτονικών εργατών.*

9.5 Κατανεμημένος αλγόριθμος δυναμικής δρομολόγησης

Σε αυτή την ενότητα περιγράφεται ο προτεινόμενος κατανεμημένος δυναμικός αλγόριθμος, ο οποίος ονομάζεται αλγόριθμος Κατανεμημένης Προσαρμοστικής Δρομολόγησης (Distributed Adaptive Scheduling (DAS). Στον DAS, λόγο της απουσίας του συντονιστή, οι m εργάτες αρχίζουν σχηματίζοντας ένα εικονικό δακτύλιο (σχήμα 9.3). Σε αυτόν τον δακτύλιο οι εργάτες ανταλλάσσουν δύο τύπους μηνυμάτων: μηνύματα *δρομολόγησης* και μηνύματα *δεδομένων*. Αυτή η ανταλλαγή μηνυμάτων απεικονίζεται στο σχήμα 9.5.

Τα μηνύματα δρομολόγησης περιέχουν το αρχικό ($Start_k$) και τερματικό ($Stop_k$) σημείο, τα οποία περιγράφουν την ομάδα επαναλήψεων κατά μήκος της διάστασης U_c που εκτελέστηκε από τον εργάτη k, όπως επίσης και τον χρόνο υπολογισμού

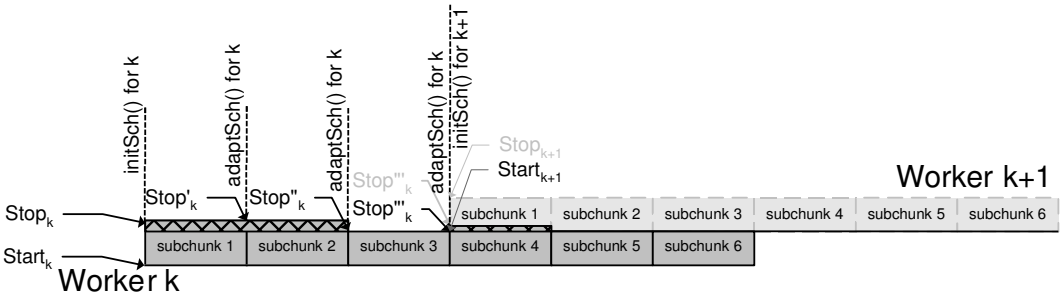
της τελευταίας υποομάδας $t^k_{lastSbchnk}$. Τα μηνύματα δεδομένων περιέχουν τα μερικά αποτελέσματα έχει υπολογίσει ο εργάτης k και χρειάζεται ο εργάτης $k+1$ για να αρχίσει τον υπολογισμό.

Η διαδικασία αρχικοποιείται από τον εργάτη με αριθμό 0. Για τον σχηματισμό του εικονικού δακτυλίου, ο εργάτης 0 παίζει τον ρόλο του προσωρινού συντονιστή. Ο εργάτης 0 δέχεται αιτήσεις εγγραφής από τους υπόλοιπους εργάτες και σχηματίζει τον δακτύλιο βασισμένος στην σειρά λήψης των αιτήσεων. Στέλνει τις πληροφορίες που αφορούν τον δακτύλιο σε κάθε εργάτη, δηλαδή την ταυτότητα του εργάτη που είναι αριστερά (προηγούμενος εργάτης) και δεξιά (επόμενος εργάτης) στον δακτύλιο από τον τρέχον εργάτη (γείτονες). Αφού σχηματιστεί ο δακτύλιος, ο προσωρινός συντονιστής δεν είναι πλέον απαραίτητος και έτσι ο εργάτης 0 τοποθετείται στον δακτύλιο μαζί με τους υπόλοιπους εργάτες. Για λόγους απλότητας υποθέτουμε ότι οι εργάτες είναι παρατεταγμένοι με τον ακόλουθο τρόπο: 1,2,3, ..., m, 0 και ότι για τον εργάτη 1, ο προηγούμενος εργάτης (στα αριστερά) είναι ο εργάτης 0, ενώ ο επόμενος είναι ο εργάτης 2. Το χρονικό κόστος για τον σχηματισμό του δακτυλίου εξαρτάται από τον αριθμό των εργατών, θεωρείται πάντως αμελητέο. Ο εργάτης 0 πυροδοτεί την έναρξη της εκτέλεσης στέλνοντας ένα αρχικό πακέτο δρομολόγησης στον επόμενο εργάτη του δακτυλίου, δηλαδή στον εργάτη 1.

Ο εργάτης 1 λαμβάνει αυτό το πακέτο και καθορίζει το μέγεθος της πρώτης ομάδας επαναλήψεων χρησιμοποιώντας την συνάρτηση **initSch(schedInfo)** πού περιγράφεται παρακάτω. Για τον πρώτο γύρο αναθέσεων ($i=1$), χρησιμοποιεί της τρέχουσες πληροφορίες σχετικά με το εξωτερικό φορτίο του (q_1), προκείμενου να καθορίσει το μέγεθος της ομάδας επαναλήψεων (V^1_1). Για όλες τις επόμενες αναθέσεις, η συνάρτηση **initSch(schedInfo)** χρησιμοποιεί το μέγεθος της προηγούμενης ομάδας που είχε ανατεθεί στον ίδιο εργάτη V^k_{i-1}, τον χρόνο εκτέλεσης της τελευταίας υποομάδας του προηγούμενου εργάτη $t^{k-1}_{lastSbchk}$ καθώς και τον χρόνο εκτέλεσης της τελευταίας υποομάδας του τρέχοντος εργάτη $t^k_{lastSbchk}$. Αφού ο εργάτης 1 είναι αυτός που ξεκινά την εκτέλεση δεν χρειάζεται να περιμένει για μερικά αποτελέσματα από τον εργάτη 0.

Προκείμενου να αποφευχθούν οι άεργοι χρόνοι όπως περιγράφεται στο Σχήμα 9.2 (a), πρέπει να εξασφαλιστεί ότι χρόνος υπολογισμού κάθε υποομάδας του προβλήματος είναι ο ίδιος. Αφού στοχεύουμε σε ένα σε σύστημα με μεταβλητό φορτίο, δεν μπορούμε να εξασφαλίσουμε ότι ο χρόνος υπολογισμού όλων των υποομάδων του προβλήματος θα είναι ο ίδιος, προσπαθούμε ωστόσο να πετύχουμε την σύγκληση των χρόνων εκτέλεσης των υποομάδων που εκτελούνται από δύο διαδοχικούς εργάτες P_{k-1} και P_k. Αυτή την χρονική σύγκλιση προσπαθεί να πετύχει η συνάρτηση **adaptSch(...)**, η οποία περιγράφεται παρακάτω.

Ο εργάτης P_k λαμβάνει από τον εργάτη P_{k-1} τις πληροφορίες δρομολόγησης (**schedInfo**), οι οποίες αποτελούνται από τον χρόνο υπολογισμού της τελευταίας υποομάδας που έχει υπολογίσει ο P_{k-1} ($t^{k-1}_{lastSbchnk}$) και τις τιμές των $Start_{k-1}$ και $Stop_{k-1}$. Από αυτό το σημείο και ύστερα, ο εργάτης P_{k-1} δεν έχει το δικαίωμα να αναπροσαρμόσει το μέγεθος της ομάδας επαναλήψεων που έχει αναλάβει, και διατηρεί την τιμή του $Stop_{k-1}$ μέχρι την ολοκλήρωση του υπολογισμού της τρέχουσας ομάδας επαναλήψεων. Αυτό γίνεται για να ξέρει ο επόμενος εργάτης P_k το σημείο από το οποίο θα αρχίσει να υπολογίζει. Θα ήταν θεμιτό να μπορεί κάποιος εργάτης να προσαρμόζει το μέγεθος της ομάδας επαναλήψεων ανά πάσα στιγμή αντιληφθεί αλλαγή



Σχήμα 9.6: Οι λειτουργίες **initSch()** και **adaptSch()**

στο εξωτερικό φορτίο του, ωστόσο αυτή η προσέγγιση θα προκαλούσε πολύ μεγάλο κόστος συντονισμού και ανωμαλίες στην εκτέλεση, αφού θα έπρεπε να ειδοποιήσει όλους τους τους επόμενους εργάτες, που με την σειρά τους θα έπρεπε να προσαρμόσουν αναλόγως την δρομολόγηση.

Ο εργάτης P_k χρησιμοποιεί το **schedInfo** που έχει λάβει από τον εργάτη P_{k-1} και εκτελεί την συνάρτηση **initSch(schedInfo)** για να καθορίσει την τιμή του V_i^k, έτσι ώστε $t^k_{subchunk1} \approx t^{k-1}_{lastSbchnk}$. Στην συνέχεια λαμβάνει τα μερικά αποτελέσματα από τον εργάτη P_{k-1}. Υπολογίζει την πρώτη υποομάδα επαναλήψεων (subchunk 1) μετρώντας παράλληλα τον χρόνο υπολογισμού $t^k_{lastSbchk}$. Στην συνέχεια εκτελεί την συνάρτηση **adaptSch**($t^{k-1}_{lastSbchk}$, $t^k_{subchunk1}$, $Start_k$, $Stop_k$) (Σχήμα 9.5) προκειμένου να ελαχιστοποιήσει την διαφορά μεταξύ του δικού του χρόνου υπολογισμού ($t^k_{subchunk1}$) και αυτού του προηγούμενου εργάτη $t^{k-1}_{lastSbchnk}$.

Αν το μέγεθος της ομάδας $V_i^{'k}$ που προκύπτει από την συνάρτηση *adaptSch(...)* είναι μεγαλύτερο από το αρχικό μέγεθος V_i^k (βλέπε Σχήμα 9.6, το γκρι $Stop_k^{'''}$), τότε το $V_i^{'k}$ γίνεται ίσο με το αρχικό μέγεθος V_i^k. Δεν επιτρέπεται στο μέγεθος της αναπροσαρμοσμένης ομάδας να γίνει μεγαλύτερη της αρχικής για να διατηρηθεί η απλότητα του κοψίματος του χώρου επαναλήψεων και της δρομολόγησης. Σε αντίθετη περίπτωση θα υπήρχαν σημεία του χώρου επαναλήψεων πού δεν έχουν εκτελεστεί και τα οποία θα ήταν δύσκολο να ανατεθούν σε κάποιο εργάτη. Ωστόσο, στην περίπτωση που το $V_i^{'k}$ είναι μικρότερο του V_i^k (βλέπε Σχήμα 9.6, $Stop_k^{''}$), ο εργάτης P_k θα ενημερώσει την τιμή του $Stop_k$. Έτσι βέβαια όταν ο επόμενος εργάτης P_{k+1} ενημερωθεί για την τιμή του $Stop_k$ θα υπολογίσει ξανά κάποια σημεία που είχε υπολογίσει προηγουμένως ο εργάτης P_k, ωστόσο αυτό δεν επηρεάζει σημαντικά την συνολική απόδοση.

Πρέπει να ξεκαθαρίσουμε ότι η συνάρτηση **adaptSch(...)**, μπορεί να εκτελεστεί μόνο μέχρι την στιγμή που εμφανίζεται ο επόμενος εργάτης (στον παρακάτω ψευδοκώδικα $nextReady = TRUE$). Από την στιγμή αυτή και πέρα ο εργάτης δεν μπορεί να προσαρμόσει άλλο το μέγεθος της ομάδας επαναλήψεων του και στέλνει το πακέτο δρομολόγησης στον επόμενο εργάτη.

Μέχρι να εμφανιστεί ο επόμενος εργάτης P_{k+1}, ο εργάτης P_k αποθηκεύει τα μερικά αποτελέσματα που πρέπει κάποια στιγμή να προωθήσει (μεταβλητή $offset$ του ψευδοκώδικα). Μόλις εμφανιστεί ο επόμενος εργάτης του αποστέλλει όλα τα

αποθηκευμένα αποτελέσματα.

Μόλις η τιμή του $Stop_k$ σταθεροποιείται, μετά την τελευταία εκτέλεση της συνάρτησης **adaptSch(...)**, η εκτέλεση προχωράει ακολουθώντας την ακόλουθη σειρά: *λήψη μερικών αποτελεσμάτων* → *υπολογισμός υποομάδος* → *αποστολή μερικών αποτελεσμάτων*.

Ο αλγόριθμος DAS περιγράφεται από τον ακόλουθο ψευδοκώδικα.

Αρχικοποίηση:

Αν ο αριθμός του εργάτη = 0

Δέξου αιτήσεις εγγραφής από τους υπόλοιπους εργάτες

Σχημάτισε τον δακτύλιο.

Στείλε τις πληροφορίες σχετικά με τους γειτονικούς κόμβους του δακτυλίου σε όλους τους εργάτες.

Στείλε το αρχικό πακέτο δρομολόγησης για να αρχίσει η εκτέλεση.

Αλλιώς

Εγγράψου με τον εργάτη με αριθμό ταυτότητας 0

Λάβε τις πληροφορίες σχετικά με τους γειτονικούς εργάτες.

Εκτέλεση:

Περίμενε για πακέτο δρομολόγησης

Αν δεν υπάρχουν διαθέσιμες επαναλήψεις

Τερματισμός

Καθορισμός αρχικού μεγέθους ομάδας επαναλήψεων $V_i^k = initSch(schedInfo)$

Για κάθε υποομάδα επαναλήψεων:

Αν ο αριθμός των υπολειπόμενων υποομάδων > 0

Λάβε μερικά αποτελέσματα από τον προηγούμενο εργάτη στον δακτύλιο.

Καθόρισε τον αριθμό των υποομάδων $\#sc\_count$ στα οποία αντιστοιχούν τα ληφθέντα μερικά αποτελέσματα.

Μείωσε τον αριθμό των υπολειπόμενων υποομάδων κατά $\#sc\_count$.

Άρχισε το χρονόμετρο.

Υπολόγισε την τρέχουσα υποομάδα επαναλήψεων.

Σταμάτα το χρονόμετρο.

Αν ο επόμενος εργάτης στον δακτύλιο δεν είναι έτοιμος

Αν δεν έχει ξεπεραστεί το όριο αναπροσαρμογής του μεγέθους της ομάδας επαναλήψεων ($adaptCount \leq adaptLimit$) Προσάρμοσε το μέγεθος της ομάδας επαναλήψεων, $V_i'^k = adaptSch(\ldots)$, με $V_i'^k \leq V_i^k$.

Εξέτασε αν ο επόμενος εργάτης είναι έτοιμος.

Αν δεν είναι

Αύξησε τον μετρητή $Offset$ (πόσες υποομάδες προηγείται ο τρέχον εργάτης του επομένου).

Αλλιώς

Στείλε το πακέτο δρομολόγησης.

Μηδένισε το $adaptCount$.

Αλλιώς

Στείλε τα μερικά αποτελέσματα που αντιστοιχούν στις $Offset$ υποομάδες.

$offset = 1$

---

* Αρχικό σχέδιο δρομολόγησης *\

Συνάρτηση **initSch**(schedInfo) * Αρχικό σχέδιο δρομολόγησης *\

if$(i = 1)$ * Πρώτος γύρος *\

$V_i^k = \frac{V_i}{q_k}$

else

$V_i^k = \frac{V_{i-1}^k \times t_{lastSbchk}^{k-1}}{t_{lastSbchk}^k}$

endIf

return V_i^k

---

* Αναπροσαρμογή σχεδίου δρομολόγησης *\

Συνάρτηση **adaptSch**($t_{lastSbchk}^{k-1}$, $t_{lastSbchk}^k$, $Start_k$, $Stop_k$)

$V_i'^k = \frac{V_{i-1}^k \times t_{lastSbchk}^{k-1}}{t_{lastSbchk}^k}$ * *υπολόγισε το νέο μέγεθος* $V_i'^k$ *\

if$(V_i'^k > V_i^k)$

$V_i'^k = V_i^k$ * το νέο μέγεθος δεν μπορεί να είναι μεγαλύτερο του αρχικού *\

else

ανανέωσε το $Stop_k$, έτσι ώστε $Stop_k - Start_k = V_i'^k$.

endIf

return $V_i'^k$

9.6 Πειραματική αξιολόγηση

Σε αυτή την ενότητα περιγράφουμε το παράδειγμα εφαρμογής, πού είναι μια μέθοδος ευθυγράμμισης ακολουθιών DNA και παρουσιάζουμε τα πειραματικά αποτελέσματα που επιδεικνύουν την αποδοτικότητα του προτεινόμενου αλγόριθμου.

9.6.1 Ευθυγράμμιση ακολουθίας DNA

Μια από τις πιο αποτελεσματικές μεθόδους για να συμπεράνουμε σχετικά με την λειτουργία ενός γονιδίου είναι αυτή της αναζήτησης ομοιοτήτων μεταξύ των ακολουθιών DNA που περιέχονται μέσα σε βάσεις δεδομένων. Οι μέθοδοι σύγκρισης ακολουθιών που βασίζονται στον δυναμικό προγραμματισμό όπως οι Needleman-Wunsch [103] και Smith-Waterman [104], αν και παράγουν βέλτιστες λύσεις, είναι αρκετά απαιτητικές υπολογιστικά. Για αυτό τον λόγο, οι περισσότερες μέθοδοι σύγκρισης ακολουθιών που χρησιμοποιούνται στην πράξη βασίζονται σε εμπειρικές ευριστικές μεθόδους οι οποίες είναι μεν γρηγορότερες,παράγουν ωστόσο λύσεις οι οποίες δεν είναι βέλτιστες. ο αλγόριθμος Needleman-Wunsch επιλέχθηκε σαν παράδειγμα εφαρμογής, προκειμένου να αξιολογηθεί η απόδοση του αλγορίθμου DAS επειδή είναι και υπολογιστικά απαιτητικός και παράγει βέλτιστες λύσεις.

Ο αλγόριθμος Needleman-Wunsch αποτελείται από δύο μέρη: (1) την σύγκριση δυο ακολουθιών DNA με την παραγωγή του πίνακα βαθμολόγησης, ο οποίος δείχνει τον βαθμό ομοιότητας και (2) την αναγνώριση της ευθυγράμμισης που οδήγησε σε αυτή την βαθμολογία. Άν το τελευταίο στοιχείο του πίνακα βαθμολόγησης περιέχει μια υψηλή θετική βαθμολογία τότε οι δύο ακολουθίες έχουν μεγάλο βαθμό ομοιότητας. Στην παρούσα προσέγγιση παραλληλοποιούμε το πρώτο μέρος του αλγορίθμου, δηλαδή την δημιουργία του πίνακα, το οποίο έχει και μεγαλύτερο υπολογιστικό κόστος.

Οι δύο ακολουθίες τοποθετούνται κατά μήκος των αριστερών οριακών στοιχείων (X) και των οριακών στοιχείων της κορυφής (Υ) του πίνακα βαθμολόγησης. Ο πίνακας βαθμολόγησης αρχικοποιείται με φθίνουσες τιμές (0, -1, -2, -3, ...) κατά μήκος της πρώτης γραμμής και πρώτης στήλης με στόχο να τιμωρήσει ο αλγόριθμος την ύπαρξη διαδοχικών κενών. Τα υπόλοιπα στοιχεία του πίνακα βαθμολόγησης $SM[n_1, n_2]$ υπολογίζονται με την χρήση της παρακάτω επαναληπτικής εξίσωσης:

$$SM[i,j] = \max \begin{cases} SM[i,j-1] + gp \\ SM[i-1,j-1] + ss \\ SM[i-1,j] + gp \end{cases}$$

Όπου gp είναι η ποινή του κενού και ss είναι η βαθμολογία αντικατάστασης, στα αποτελέσματα που παρουσιάζονται στην συνέχεια έχουμε $gp = -2$, και $ss = 1$ αν τα στοιχεία ταιριάζουν ή αλλιώς $ss = 0$.

Η παραπάνω επαναληπτική διαδικασία μπορεί να παραλληλοποιηθεί θεωρώντας μια εκτέλεση που ακολουθεί ένα αντιδιαγώνιο μέτωπο κύματος, δηλαδή, είναι δυνατός ο ταυτόχρονος υπολογισμός των σημείων που βρίσκονται πάνω στην ίδια αντιδιαγώνιο σε κάθε βήμα εκτέλεσης. Φυσικά, μια προσέγγιση παραλληλισμού λεπτού κόκκου θα προκαλούσε μεγάλο κόστος επικοινωνιών και άρα ακολουθούμε την συνήθη προσέγγιση του χονδρόκοκκου παραλληλισμού.

Πειραματική διάταξη. Με την καθιέρωση της τεχνολογίας των πολλαπλών επεξεργαστικών πυρήνων σε ένα ολοκληρωμένο κύκλωμα, οι σημερινές συστοιχίες υπολογιστών αποτελούνται από πολύεπεξεργαστικούς κόμβους (SMP clusters). Τα παρακάτω πειράματα εκτελέστηκαν σε μία συστοιχία πολυεπεξεργαστών που αποτελείται από 16 κόμβους linux (πυρήνα 2.6.24.2) που ονομάζονται clones. Κάθε κόμβος

περιλαμβάνει δύο τετραπύρηνους επεξεργαστές (quad-core Xeon chips) βασισμένους στην αρχιτεκτονική Intel Core 2 (E5405, @2.00 GHz). Δύο πυρήνες σε κάθε πακέτο μοιράζονται μια 4MB L2 cache. Συνολικά το σύστημα μπορεί να προσφέρει 128 υπολογιστικούς πυρήνες. Το δίκτυο διασύνδεσης είναι το Gigabit Ethernet.

Όλοι οι αλγόριθμοι δρομολόγησης υλοποιήθηκαν σε γλώσσα *C* με την χρήση της βιβλιοθήκης MPI (mpich 1.2.6) για την ανταλλαγή των μηνυμάτων. Ο αλγόριθμος Needleman-Wunsch ενσωματώθηκε στον κώδικα που εκτελείται από κάθε εργάτη. Το μήκος κάθε ακολουθίας που δοκιμάστηκε ήταν 1024000 στοιχεία. Τέλος, κάθε πείραμα εκτελέστηκε 10 φορές και τα αποτελέσματα που παρουσιάζονται αποτελούν τους μέσους όρους αυτών των εκτελέσεων.

Για να μελετήσουμε την συμπεριφορά του κατανεμημένου αλγόριθμου σε σχέση με αυτούς που στηρίζονται στο μοντέλο συντονιστή-εργάτη επινοήσαμε δύο σενάρια μεταβολής του φορτίου του συστήματος: (α) αργή μεταβολή και (β) γρήγορη μεταβολή. Με αυτό τον τρόπο μπορούμε να εξομοιώσουμε ένα μη-αφοσιωμένο σύστημα πολλαπλών χρηστών, στο οποίο το φορτίο μεταβάλλεται, όπως και στα πειράματα του προηγούμενου κεφαλαίου.

Και στα δύο σενάρια χρησιμοποιήθηκαν τρεις τύποι κόμβων:

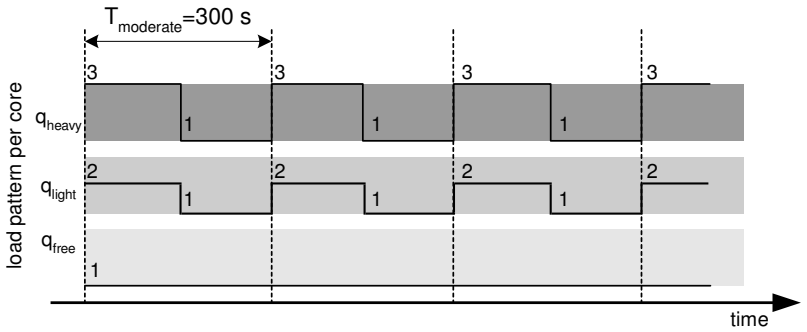
ελεύθερος κόμβος - Κάθε πυρήνας αυτού του κόμβου ήταν αφοσιωμένος στην εφαρμογή ευθυγράμμισης DNA. Ο αριθμός διεργασιών στην ουρά εκτέλεσης κάθε πυρήνα ήταν πάντα 1.

ελαφρά φορτωμένος κόμβος - Κάθε πυρήνας του κόμβου μοιράζεται με *μια* εξωτερική εφαρμογή για σταθερά χρονικά διαστήματα. Ο αριθμός των διεργασιών στην ουρά εκτέλεσης κάθε κόμβου μπορεί να είναι είτε 1 είτε 2.

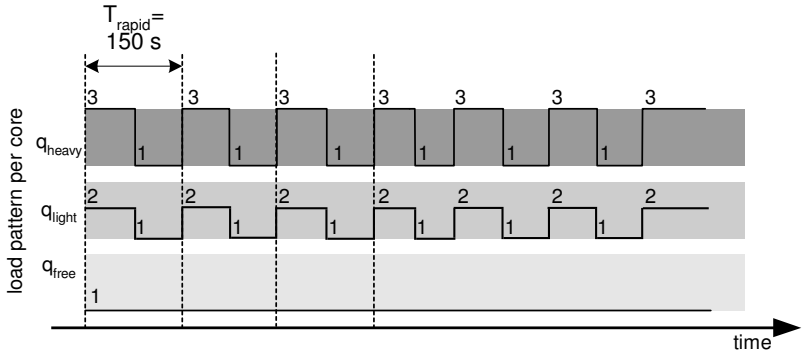
βαριά φορτωμένος κόμβος - Κάθε πυρήνας του κόμβου μοιράζεται με *δύο* εξωτερικές εφαρμογές για σταθερά χρονικά διαστήματα. Ο αριθμός των διεργασιών στην ουρά εκτέλεσης κάθε κόμβου μπορεί να είναι είτε 1 είτε 3.

Στους υπερφορτωμένους κόμβους, κάθε πυρήνας μοιράζεται μεταξύ της εφαρμογής ευθυγράμμισης και εξωτερικές εφαρμογές για ένα σταθερό χρονικό διάστημα και στην συνέχεια, για το ίδιο σταθερό χρονικό διάστημα αφοσιώνεται στην αποκλειστική εκτέλεση της εφαρμογής ευθυγράμμισης. Ο κύκλος αυτός επαναλαμβάνεται μέχρι την περάτωση του κάθε πειράματος, δηλαδή μέχρι να ολοκληρωθεί η εκτέλεση της εφαρμογής. Η διαφορά στα δύο σενάρια που αναφέρθηκαν παραπάνω είναι στην χρονική περίοδο που διαρκεί κάθε κύκλος: Η περίοδος για το σενάριο αργής μεταβολής είναι $300s$ ενώ για την γρήγορη μεταβολή είναι $150s$. Η επιλογή της περιόδου δεν είναι τυχαία. Ο χρόνος των $300s$ αντιστοιχεί στον μέσο χρόνο εκτέλεσης των αρχικών ομάδων επαναλήψεων που προκύπτουν από τους τρεις αλγόριθμους: SAS, $\mathcal{S}-AWF$ και DAS, όταν αυτοί εκτελούνται σε στο αφοσιωμένο σύστημα από 64 εργάτες. Οι κύκλοι φόρτωσης απεικονίζονται στο Σχήμα 9.7.

Στα πειράματα που πραγματοποιήθηκαν δόθηκε βάση στον ρυθμό μεταβολής του φορτίου, αφού η απόδοση κάθε αλγόριθμου εξισορρόπησης φορτίου εξαρτάται από την ικανότητα του να προσαρμόζεται στης μεταβολές του φορτίου του συστήματος.



(a) Moderate load variation scenario



(b) Rapid load variation scenario

Σχήμα 9.7: *Σχηματική περιγραφή της αργής και γρήγορης μεταβολής του φορτίου του συστήματος. Το πρώτο φορτίο στην ουρά εκτέλεσης αντιστοιχεί πάντα στην δική μας εφαρμογή.*

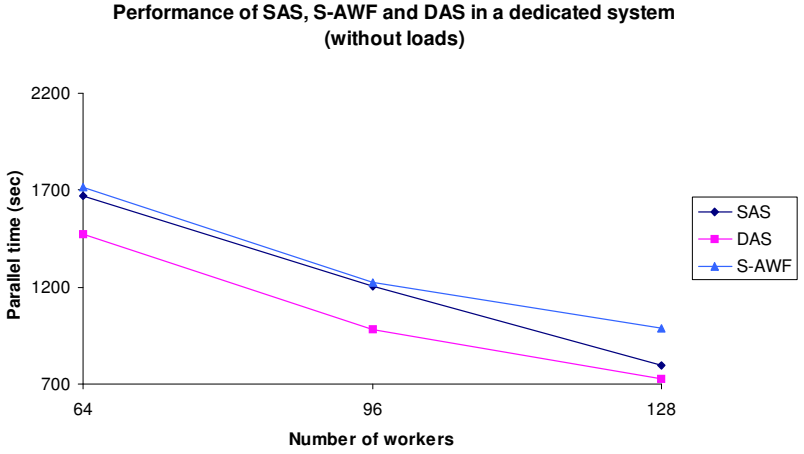
9.6.2 Αποτελέσματα

Αποτελέσματα απόδοσης. Εδώ παρουσιάζουμε την απόδοση των αλγορίθμων SAS, $\mathcal{S}$-AWF και DAS σε ένα σύστημα 64, 96 και 128 εργατών. Οι 64 εργάτες προήλθαν από 8 κόμβους από τους οποίους χρησιμοποιήθηκαν και οι 8 πυρήνες ($8x8$), οι 96 εργάτες από 12 κόμβους ($12x8$) και τέλος οι 128 από το σύνολο των 16 κόμβων ($16x8$). Η κατανομή των *ελεύθερων*, *ελαφρά φορτωμένων* και *βαριά φορτωμένων* κόμβων για κάθε μια από της παραπάνω περιπτώσεις δίνεται στον Πίνακα 9.1.

Αρχικά στο Σχήμα 9.8 παρουσιάζεται η απόδοση των τριών αλγορίθμων στο αφοσιωμένο σύστημα, δηλαδή στο σύστημα στο οποίο δεν εκτελούνται εξωτερικές εφαρμογές. Αυτό είναι και ένα μέτρο της βέλτιστης απόδοσης του κάθε αλγόριθμου κάτω από τις τρέχουσες συνθήκες. Γίνεται αντιληπτό ότι για στην περίπτωση των 128 εργατών, η απόδοση του αλγόριθμου $\mathcal{S} - AWF$ υποφέρει από μεγάλο αριθμό αιτήσεων προς τον συντονιστή, εξαιτίας του μικρού μεγέθους των ομάδων επαναλήψεων προς το τέλος της παράλληλης εκτέλεσης. Όπως φαίνεται, οι άλλοι δύο αλγόριθμοι δεν παρουσιάζουν το ίδιο πρόβλημα. Ο αλγόριθμος DAS παρουσιάζει την καλύτερη απόδοση από τους τρεις αλγορίθμους λόγο της αποκεντρωμένης φύσης του.

Πίνακας 9.1: Οι ελεύθεροι, ελαφρά και βαριά φορτωμένοι κόμβοι ανάλογα με τον αριθμό των εργατών στο σύστημα.

Αριθμός εργατών και κόμβων	ελεύθεροι	ελαφριά φορτωμένοι	βαριά φορτωμένοι
64 εργάτες (8 κόμβοι)	1 4 7	2 5 8	3 6
96 εργάτες (12 κόμβοι)	1 4 7 10	2 5 8 11	3 6 9 12
128 εργάτες (16 κόμβοι)	1 4 7 10 13 16	2 5 8 11 14	3 6 9 12 15



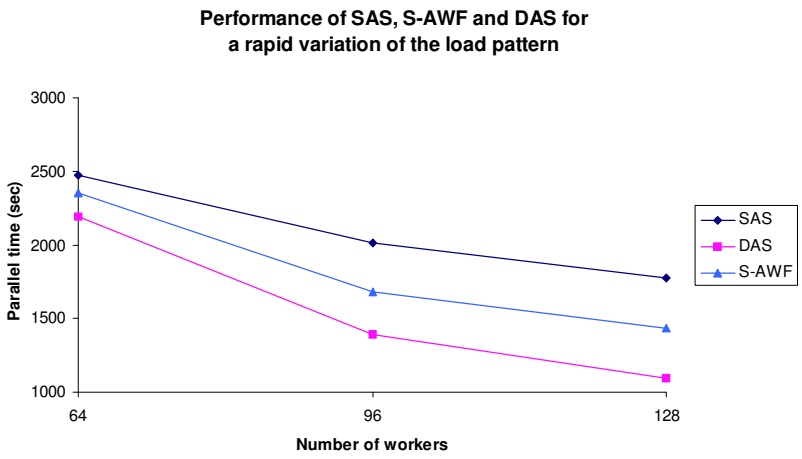
Σχήμα 9.8: *Απόδοση των SAS, $\mathcal{S}-AWF$ και DAS στο αφοσιωμένο σύστημα.*

Σχήμα 9.9: *Απόδοση των SAS, $\mathcal{S}-AWF$ και DAS στο σύστημα με την αργή μεταβολή φορτίου.*

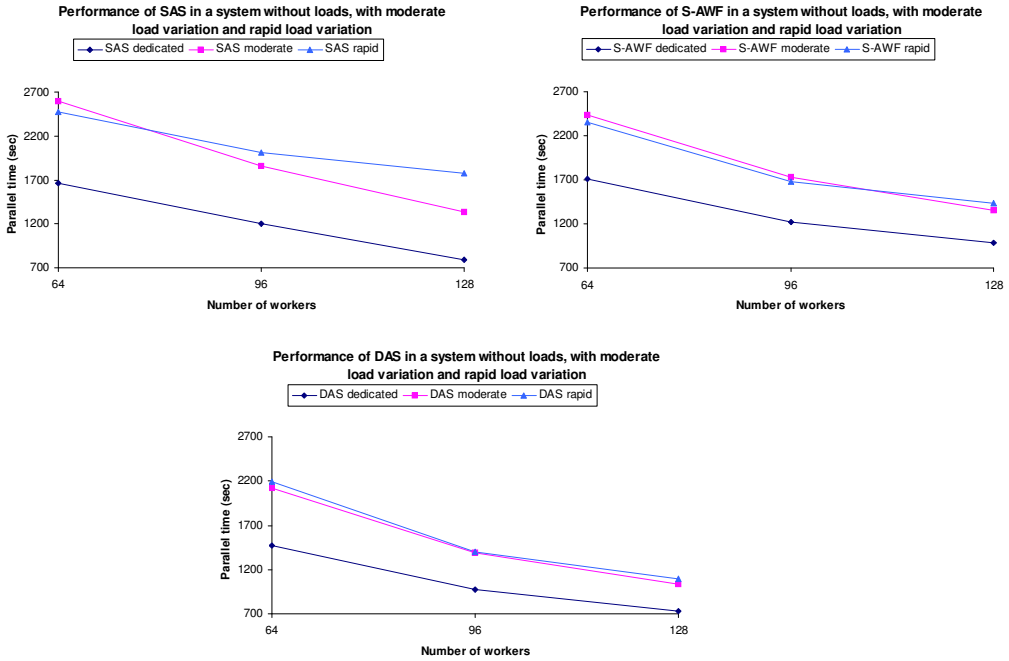
Στα επόμενα σχήματα 9.9 και 9.10 παρουσιάζεται η απόδοση των τριών αλγορίθμων για τις περιπτώσεις του συστήματος με αργό και γρήγορα μεταβαλλόμενο φορτίο. Ο αλγόριθμος $\mathcal{S}-AWF$ δείχνει να προσαρμόζεται καλύτερα στις μεταβολές του εξωγενούς φορτίου από ότι ο SAS. Αν και στην αρχή της εκτέλεσης ο $\mathcal{S}-AWF$ αναθέτει μεγάλες ομάδες επαναλήψεων, πράγμα που μπορεί να προκαλέσει ανισότητα στην κατανομή του φορτίου, στην συνέχεια και κυρίως προς το τέλος της εκτέλεσης αναθέτει πολύ μικρότερες ομάδες επαναλήψεων που εξισορροπούν την κατανομή του υπολογιστικού φορτίου. Από την άλλη μεριά, ο SAS προσπαθεί να διατηρήσει σταθερούς τους χρόνους υπολογισμού όλων των ομάδων επαναλήψεων της εφαρμογής, με αποτέλεσμα να διατηρεί το μέγεθος των ομάδων επαναλήψεων σε σταθερά μεγάλο αριθμό σημείων. Έτσι με τον SAS υπάρχει μεγάλος κίνδυνος να βασιστεί η δρομολόγηση σε παρωχημένες πληροφορίες σχετικά με το εξωτερικό φορτίο. Ο αλγόριθμος DAS παρουσιάζει την καλύτερη απόδοση και από τους τρεις αλγορίθμους του πειράματος και αυτό οφείλεται στο ότι κατέχει πάντα πιο πρόσφατες πληροφορίες από ότι οι δύο άλλοι αλγόριθμοι και έτσι μπορεί να παίρνει καλύτερες αποφάσεις για την κατανομή του υπολογιστικού φορτίου.

Ευαισθησία στις μεταβολές του φορτίου. Για να εξακριβώσουμε τον βαθμό στον οποίο επηρεάζεται η απόδοση του κάθε αλγορίθμου από τον φόρτο του συστήματος, καταρτίσαμε ένα διάγραμμα για κάθε αλγόριθμο, στο οποίο εμφανίζεται ο μέσος χρόνος εκτέλεσης για τις περιπτώσεις του αφοσιωμένου, του αργά μεταβαλλόμενου και του γρήγορα μεταβαλλόμενου συστήματος αντίστοιχα (βλέπε Σχήμα 9.11). Είναι ξεκάθαρο ότι ο αλγόριθμος DAS επηρεάζεται λιγότερο από τις μεταβολές του εξωτερικού φορτίου από ότι οι αλγόριθμοι SAS και $\mathcal{S}-AWF$.

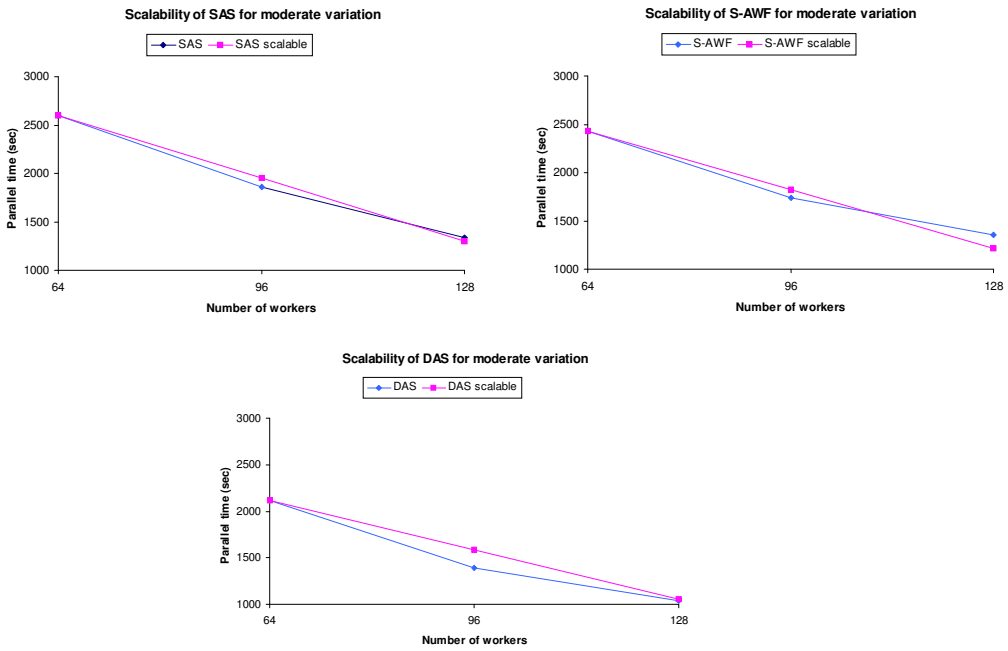
Δυνατότητα κλιμάκωσης. Οι σημερινές συστοιχίες υπολογιστών περιλαμβάνουν εκατοντάδες ή ακόμα και χιλιάδες επεξεργαστές. Το γεγονός αυτό επιβάλει την ανάπτυξη αλγορίθμων που μπορούν να χειριστούν αποδοτικά τόσο μεγάλα πλήθη επεξεργαστών. Στην συνέχεια δίνεται μια σύντομη αξιολόγηση των δυνατοτήτων κλι-



Σχήμα 9.10: *Απόδοση των SAS, $\mathcal{S}-AWF$ και DAS στο σύστημα με την γρήγορη μεταβολή φορτίου.*



Σχήμα 9.11: *Ατομική απόδοση κάθε αλγορίθμου.*

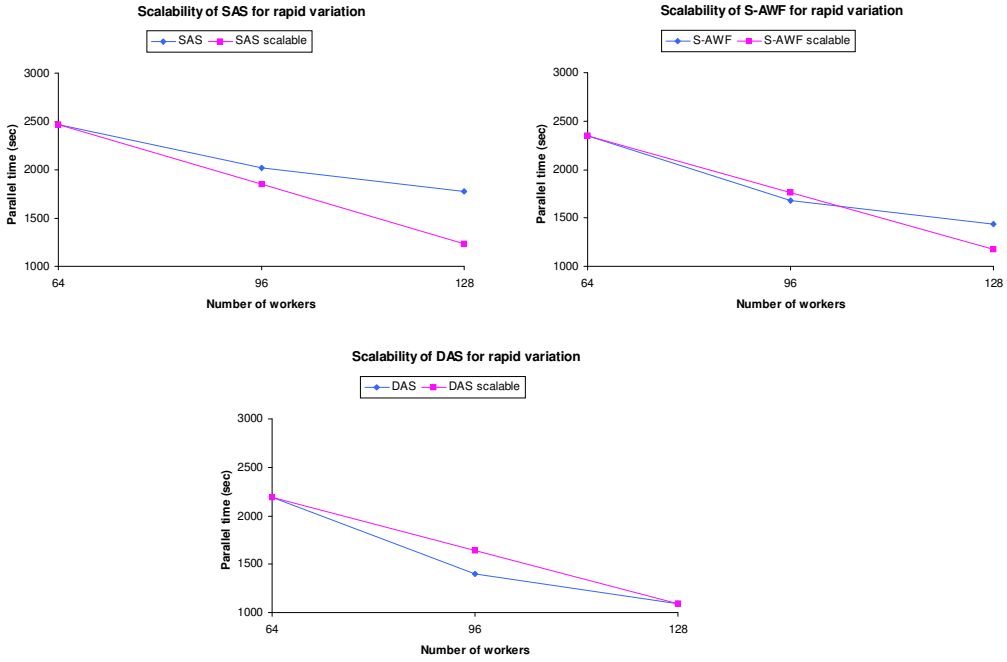


Σχήμα 9.12: *Κλιμάκωση της απόδοσης των SAS, $\mathcal{S}-AWF$ και DAS σε ένα σύστημα με αργή μεταβολή φορτίου.*

μάκωσης των τριών αλγορίθμων, καθώς ο αριθμός των εργατών που περιλαμβάνει το σύστημα αυξάνεται από 64 σε 96 και 128 εργάτες.

Τα αποτελέσματα τις αξιολόγησης παρουσιάζονται στα Σχήματα 9.12 και 9.13. Και στα δύο σχήματα η ευθεία γραμμή αντιστοιχεί στην ιδανική απόδοση στους 64, 96 και 128 επεξεργαστές, αν πάρουμε σαν απόδοση αναφοράς αυτή των 64 επεξεργαστών. Όπως μπορούμε να δούμε στο Σχήμα 9.12, η αύξηση της απόδοσης καθώς αυξάνεται ο αριθμός των επεξεργαστών είναι πολύ κοντά στην ιδανική περίπτωση και για τους τρεις αλγορίθμους. Ωστόσο, για την περίπτωση του γρήγορα μεταβαλλόμενου φορτίου, οι αλγόριθμοι SAS και $\mathcal{S}-AWF$ δεν έχουν την ίδια ικανότητα κλιμάκωσης όπως ο DAS, ο οποίος μπορεί να αντιληφθεί πιο άμεσα τις μεταβολές του φορτίου και να προσαρμοστεί πιο άμεσα από τους δύο άλλους αλγορίθμους (Σχήμα 9.13).

Διαστήματα εμπιστοσύνης. Στους Πίνακες 9.2 και 9.3 δίνονται οι μέσοι χρόνοι εκτέλεσης των αλγορίθμων DAS, SAS και $\mathcal{S}-AWF$, για το σύστημα με το αργό και το γρήγορα μεταβαλλόμενο φορτίο αντίστοιχα. Σε κάθε πίνακα δίνεται επίσης η βελτίωση που παρουσιάζει η απόδοση του DAS σε σχέση με τους συγκεντρωτικούς αλγορίθμους SAS και $\mathcal{S}-AWF$. Μας ενδιαφέρει να προσδιορίσουμε το επίπεδο εμπιστοσύνης στην βελτίωση που παρουσιάζει ο κατανεμημένος αλγόριθμος σε σχέση με τους συγκεντρωτικούς. Όπως συνηθίζεται στους πίνακες 9.2 και 9.3 δίνεται το διάστημα εμπιστοσύνης 95%. Με πιθανότητα 0.95, η βελτίωση του DAS σε σχέση με τον SAS κυμαίνεται στο διάστημα $24 \pm 3.21\%$ για την περίπτωση της αργής μεταβολής και $21.67 \pm 4.66\%$ για την περίπτωση της γρήγορης μεταβολής. Αντίστοιχα,



Σχήμα 9.13: *Κλιμάκωση της απόδοσης των SAS, $\mathcal{S}-AWF$ και DAS σε ένα σύστημα με γρήγορη μεταβολή φορτίου.*

Πίνακας 9.2: Μέσοι χρόνοι εκτέλεσης και κέρδος του κατανεμημένου ως προς τους αλγορίθμους εργάτη-συντονιστή σε ένα σύστημα με αργή μεταβολή εξωτερικού φορτίου

Αριθμός εργατών	DAS (sec)	SAS (sec)	κέρδος του DAS προς τον SAS	DAS (sec)	$\mathcal{S}-AWF$ (sec)	κέρδος του DAS προς τον $\mathcal{S}-AWF$
64	2116.73	2598.82	18.55%	2116.73	2432.96	12.99%
96	1391.27	1861.04	**25.24%**	1391.27	1734.60	19.79%
128	1034.62	1335.99	22.55%	1034.62	1357.59	**23.78%**
διάστημα εμπιστοσύνης (95%)			$\mathbf{24 \pm 3.21\%}$	**διάστημα εμπιστοσύνης (95%)**		$\mathbf{16.68 \pm 4.13\%}$

Πίνακας 9.3: Μέσοι χρόνοι εκτέλεσης και κέρδος του κατανεμημένου ως προς τους αλγορίθμους εργάτη-συντονιστή σε ένα σύστημα με γρήγορη μεταβολή εξωτερικού φορτίου

Αριθμός εργατών	DAS (sec)	SAS (sec)	κέρδος DAS προς SAS	DAS (sec)	$\mathcal{S}-AWF$ (sec)	κέρδος DAS προς $\mathcal{S}-AWF$
64	2191.78	2471.59	11.32%	2191.78	2349.64	06.71%
96	1395.63	2016.75	30.79%	1395.63	1680.86	16.96%
128	1094.73	1775.43	**38.34**%	1094.73	1433.89	**23.65**%
(95%) **διάστημα εμπιστοσύνης**			$\mathbf{21.67 \pm 4.66}$%	(95%) **διάστημα εμπιστοσύνης**		$\mathbf{13.48 \pm 3.27}$%

με πιθανότητα 0.95, ο DAS παρουσιάζει βελτίωση σε σχέση με τον $\mathcal{S}-AWF$ στο διάστημα $16.68 \pm 4.13\%$ για το αργά μεταβαλλόμενο σύστημα και $13.48 \pm 3.27\%$ για το γρήγορα μεταβαλλόμενο σύστημα.

9.7 Συμπεράσματα

Σε αυτό το κεφάλαιο παρουσιάστηκε μια νέα κατανεμημένη προσέγγιση στην δυναμική δρομολόγηση φωλιασμένων βρόχων, ο αλγόριθμος DAS. Ο κατανεμημένος αλγόριθμος εφαρμόστηκε σε μία υπολογιστικά απαιτητική εφαρμογή και η απόδοση του συγκρίθηκε με αυτή δύο συγκεντρωτικών αλγορίθμων (SAS και $\mathcal{S}-AWF$) σε μια συστοιχία πολυεπεξεργαστών με αργά ή γρήγορα μεταβαλλόμενο εξωτερικό φορτίο. Σε όλα τα πειράματα, η κατανεμημένη προσέγγιση ξεπέρασε τις συγκεντρωτικές και σε απόδοση (απόλυτοι χρόνοι) αλλά και σε δυνατότητα κλιμάκωσης. Το συμπέρασμα είναι ότι για μεγάλα συστήματα που αποτελούνται από εκατοντάδες ή και χιλιάδες επεξεργαστικά στοιχεία, η κατανεμημένες προσεγγίσεις μπορούν να προσφέρουν καλύτερη απόδοση. Ένα επιπλέον στοιχείο πού κάνει τις κατανεμημένες μεθόδους πιο ελκυστικές είναι ότι η έλλειψη ενός κεντρικού σημείου ελέγχου τις κάνει και πιο ανθεκτικές στα σφάλματα του συστήματος. Αυτή είναι και μια κατεύθυνση που πρέπει να μας απασχολήσει στο μέλλον.

Κεφάλαιο 10

Υλοποίηση δυναμικών αλγορίθμων σε επανάπρογραμματιζόμενο υλικό

Οι δυναμικοί αλγόριθμοι έχουν χρησιμοποιηθεί για την δρομολόγηση παράλληλων υπολογισμών σε παραδοσιακές πλατφόρμες αλλά και σε συστοιχίες παράλληλων υπολογιστών. Σε αυτό το κεφάλαιο παρουσιάζεται μια νέα αρχιτεκτονική που υλοποιεί παραλληλισμό χονδρού κόκκου, με την χρήσης δυναμικών αλγορίθμων σε πλατφόρμες επαναπρογραμματιζόμενου υλικού. Παρουσιάζεται επίσης ένα αναλυτικό μοντέλο και πειραματικά αποτελέσματα που αξιολογούν την απόδοση αυτής της αρχιτεκτονικής. Η βελτίωση που παρατηρείται σε σχέση με προγενέστερες προσεγγίσεις οφείλεται στην πιο αποδοτική χρήση της μνήμης και των διαθέσιμων υπολογιστικών στοιχείων.

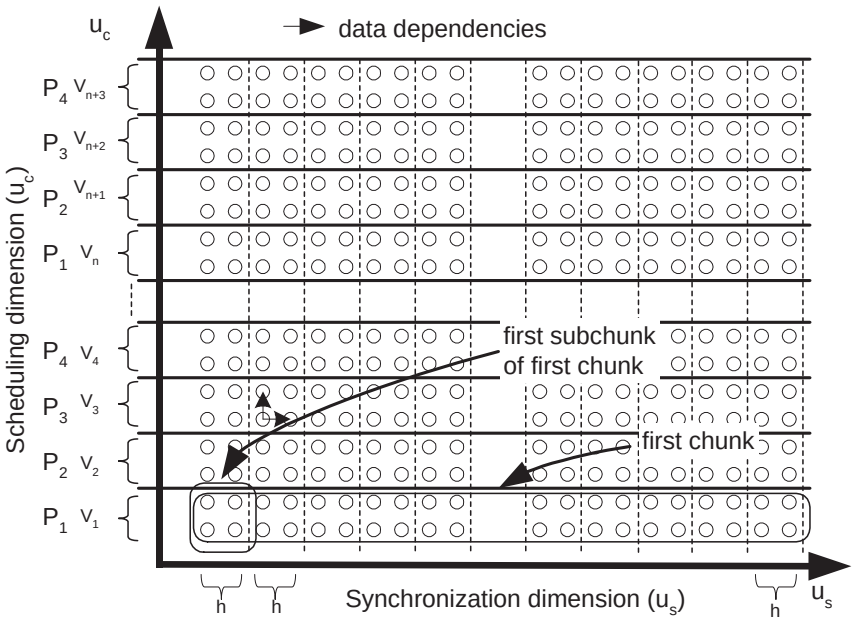
10.1 Εισαγωγή

Η συ-σχεδίαση υλικού-λογισμικού είναι μια εναλλακτική και αποτελεσματική προσέγγιση στον τομέα των συστημάτων υψηλής απόδοσης. Σε αυτή την προσέγγιση ο πηγαίος κώδικας της εφαρμογής χωρίζεται σε μέρη υλικού και λογισμικού. Στην συνέχεια το μέρος του λογισμικού εκτελείται σε ένα επεξεργαστή γενικής χρήσης, ενώ το υλικό μέρος τοποθετείται σε κάποια πλατφόρμα επαναπρογραμματιζόμενου υλικού, όπως για παράδειγμα σε ένα FPGA. Έτσι, έχουμε και την ευελιξία που προσφέρει μια υλοποίηση σε λογισμικό αλλά και την υψηλή απόδοση που παρέχει η υλοποίηση σε υλικό Έχουν υλοποιηθεί στο παρελθόν διάφορες τέτοιες προσεγγίσεις όπως οι αυτές που παρουσιάζονται στα [1], [98], [94], [47], [45]. Μιας και οι φωλιασμένοι βρόχοι είναι από τα πιο απαιτητικά υπολογιστικά τμήματα του πηγαίου κώδικα ενός προγράμματος, αποτελούν ιδανική περίπτωση για υλοποίηση σε υλικό. Επίσης, έχουν προταθεί πολλές μεθοδολογίες για την παραλληλοποίηση τον φωλιασμένων βρόχων πριν την τοποθέ-

τηση τους στο υλικό [44], [93], [58], με στόχο την περαιτέρω αύξηση της απόδοσης. Ωστόσο, όλες αυτές οι μεθοδολογίες χρησιμοποιούν παραλληλισμό λεπτού κόκκου. Από την άλλη, όπως έχουμε δει σε συστήματα υπολογιστών κατανεμημένης μνήμης, οι μεθοδολογίες που χρησιμοποιούν παραλληλισμό χονδρού κόκκου, όπως αυτές που παρουσιάζονται στα [51], [34], [48] προσφέρουν καλύτερη τοπικότητα δεδομένων και πιο αποτελεσματική χρήση της μνήμης σε σχέση με αυτές που χρησιμοποιούν παραλληλισμό λεπτού κόκκου. Η χρήση χονδρόκοκκου παραλληλισμού μπορεί προσφέρει βελτίωση της απόδοσης και στην περίπτωση των ενσωματωμένων συστημάτων.

Οι παραπάνω μεθοδολογίες είναι στατικές, δηλαδή η ανάθεση επαναλήψεων σε επεξεργαστικά στοιχεία έχει αποφασιστεί πριν την έναρξη την εκτέλεσης, κατά την φάση της μεταγλώττισης. Αυτή η προσέγγιση απαιτεί την κατάληψη χώρου στην μνήμη για την αποθήκευση του χάρτη εκτέλεσης, δηλαδή της ανάθεσης των επαναλήψεων στα διαθέσιμα επεξεργαστικά στοιχεία. Αν αυτή η ανάθεση γίνεται βάση ενός αλγορίθμου που παρέχει παραλληλισμό λεπτού κόκκου, ο χώρος μνήμης που απαιτείται είναι σημαντικός. Αντίθετα οι δυναμικοί αλγόριθμοι, εκτός από τα γνωστά πλεονεκτήματα που προσφέρουν σχετικά με την αποτελεσματικότερη κατανομή φορτίου έναντι των στατικών αλγορίθμων, έχουν ακόμα το πλεονέκτημα ότι δημιουργούν τον χάρτη εκτέλεσης δυναμικά κατά την διάρκεια της εκτέλεσης. Αυτό σημαίνει ότι δεν χρειάζεται να σπαταλήσουν πολλή μνήμη για να τον αποθηκεύσουν. Το πρόβλημα της παραλληλοποίησης φωλιασμένων βρόχων, με την χρήση δυναμικών αλγορίθμων σε πλατφόρμες επαναπρογραμματιζόμενου υλικού παραμένει ανοιχτό ερευνητικό θέμα. Η επιτυχία των αυτοδρομολογούμενων αλγορίθμων [6], οι οποίοι μπορούν να προσφέρουν δυναμική δρομολόγηση χονδρού κόκκου, στις πλατφόρμες κατανεμημένης μνήμης κάνει την μεταφορά τους στα ενσωματωμένα συστήματα πολύ ελκυστική, στοχεύοντας στην δημιουργία ενός αποδοτικού παράλληλου περιβάλλοντος.

Σε αυτό το κεφάλαιο παρουσιάζεται ο αλγόριθμος Δυναμικής Δρομολόγησης Πολλαπλών φάσεων σε Υλικό (Hardware Dynamic-Multiphase scheduling ή για συντομία H-DMPS). Ο αλγόριθμος H-DMPS είναι ένας νέος αλγόριθμος για την δυναμική δρομολόγηση φωλιασμένων βρόχων σε υλικό, ο οποίος βασίζεται στον αλγόριθμο DMPS ([31]), ο οποίος παρουσιάστηκε στο κεφάλαιο 5. Ο Αλγόριθμος H-DMPS είναι ο πρώτος που συνδυάζει δυναμική δρομολόγηση, παραλληλισμό χονδρού κόκκου για βρόχους με εξαρτήσεις σε μια πλατφόρμα επαναπρογραμματιζόμενου υλικού. Τα πειραματικά αποτελέσματα δείχνουν ότι ο H-DMPS επιτυγχάνει ικανοποιητικά αποτελέσματα συγκρινόμενος με μια αντίστοιχη μέθοδο, που προσφέρει παραλληλισμό λεπτού κόκκου, και η οποία παρουσιάζεται στο [44]. Το βασικό μειονεκτήματα αυτής της μεθόδου, ήταν ο περιορισμός στον αριθμό των επεξεργαστικών στοιχείων που μπορούσε να χρησιμοποιήσει, λόγο της συμφόρησης της μνήμης. Στην μέθοδο που παρουσιάζεται σε αυτή την ενότητα, η χρήση του παραλληλισμού χονδρού κόκκου, μειώνει την συμφόρηση της μνήμης και επιτρέπει την αποδοτική χρησιμοποίηση περισσότερων επεξεργαστικών στοιχείων.

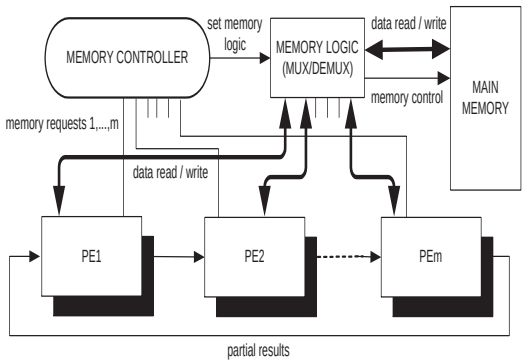


Σχήμα 10.1: Αποδόμηση του χώρου επαναλήψεων και οι διαστάσεις συγχρονισμού και δρομολόγησης ενός βρόχου 2 διαστάσεων.

10.1.1 Αυτοδρομολογούμενοι Αλγόριθμοι για βρόχους με εξαρτήσεις

Οι αυτοδρομολογούμενοι αλγόριθμοι χωρίζουν τον χώρο επαναλήψεων σε ομάδες επαναλήψεων V_i σημείων (chunks), κατά μήκος της διάστασης δρομολόγησης (u_c), δημιουργώντας μια δεξαμενή εργασιών οι οποίες αναθέτονται δυναμικά στους διαθέσιμους επεξεργαστές σύμφωνα με το μοντέλο συντονιστή-εργάτη. Στο [31] προστέθηκαν κατά μήκος μιας δεύτερης διάστασης (διάσταση συγχρονισμού u_s), σημεία συγχρονισμού (SPs) στο βασικό μηχανισμό των αυτοδρομολογούμενων αλγορίθμων, έτσι ώστε να γίνει εφικτή η χρήση τους με προβλήματα που περιέχουν εξαρτήσεις. Έτσι προέκυψε ο αλγόριθμος DMPS. Τα σημεία αυτά είναι ομοιόμορφα κατανεμημένα σε σταθερές αποστάσεις h σημείων. Στο σχήμα 10.1 παρουσιάζεται ο διαμερισμός ενός χώρου δυο διαστάσεων και περιέχονται οι παραπάνω διατυπώσεις. Στο ίδιο σχήμα διευκρινίζονται και οι έννοιες των ομάδων και υποομάδων επαναλήψεων (chunks-subchunks). Το κάτω όριο μιας ομάδας επαναλήψεων είναι το σημείο με τον μικρότερο δείκτη στην διάσταση δρομολόγησης το οποίο ανήκει στην ομάδα επαναλήψεων, ενώ το άνω όριο είναι το το σημείο με τον μεγαλύτερο δείκτη αντίστοιχα. Η ίδια βασική ορολογία θα χρησιμοποιηθεί και για την περιγραφή του αλγορίθμου $H-DMPS$.

Στον πιο βασικό αλγόριθμο self-scheduling (simple self-scheduling)[72] κάθε επεξεργαστής αναλαμβάνει να εκτελέσει μια επανάληψη του βρόχου μόλις γίνει διαθέσιμος, δηλαδή όταν τελειώσει την εκτέλεση της προηγούμενης επανάληψης που είχε



Σχήμα 10.2: Σχηματικό διάγραμμα του H-DMPS

αναλάβει. Αυτός ο αλγόριθμος επιτυγχάνει πολύ καλή εξισορρόπηση φορτίου (load-balance) αφού κάθε επεξεργαστής μπορεί να τελειώσει την εκτέλεση το πολύ με διαφορά μιας επανάληψης. Ωστόσο είναι φανερό ότι το κόστος δρομολόγησης (scheduling overhead), δηλαδή ο χρόνος που απαιτείται για την ανάθεση των επαναλήψεων στους επεξεργαστές είναι πολύ υψηλό. Για να μειωθεί το κόστος δρομολόγησης επινοήθηκε ο αλγόριθμος Chunk-scheduling (CSS)[12], στον οποίο κάθε επεξεργαστής αναλαμβάνει μια ομάδα επαναλήψεων (chunk) την φορά αντί για μια και μόνο επανάληψη. Σε γενικές γραμμές όσο μεγαλύτερο είναι το μέγεθος του chunk τόσο μειώνεται το κόστος δρομολόγησης, μειώνεται όμως ταυτόχρονα και ο διαθέσιμος παραλληλισμός. Σε αυτήν την ενότητα θα ασχοληθούμε μόνο με τον αλγόριθμο CSS.

10.2 Μεταφορά σε υλικό, ο αλγόριθμος H-DMPS

Η μεταφορά του αλγορίθμου DMPS σε πλατφόρμα υλικού ακολουθεί την προσέγγιση συντονιστή-εργάτη. Οι εργάτες υλοποιούνται σαν μια ομάδα επεξεργαστικών στοιχείων (PEs) ειδικού σκοπού, τα οποία έχουν πρόσβαση σε μια κοινή-μοιραζόμενη μνήμη, στην οποία αποθηκεύονται όλα τα δεδομένα της εφαρμογής. Τον ρόλο του συντονιστή αναλαμβάνει ένας ελεγκτής, οποίος ελέγχει την πρόσβαση των επεξεργαστικών στοιχείων στην μνήμη. Μόνο ένα PE μπορεί να προσπελάσει την μνήμη μια δεδομένη χρονική στιγμή.

Τα Επεξεργαστικά Στοιχεία διασυνδέονται σε διάταξη δακτυλίου (ring). Σε αυτή την διάταξη κάθε PE περνάει δεδομένα στο επόμενο, υλοποιώντας έτσι τον μηχανισμό συγχρονισμού η ύπαρξη του οποίου είναι απαραίτητη για να να ικανοποιηθούν οι εξαρτήσεις δεδομένων. Κάθε Επεξεργαστικό Στοιχείο εκτελεί τα παρακάτω βασικά βήματα: διαβάζει δεδομένα από την κοινή μνήμη, λαμβάνει δεδομένα (μερικά αποτελέσματα) από το προηγούμενο PE σύμφωνα με τις εξαρτήσεις, επεξεργάζεται μια υποομάδα επαναλήψεων, στέλνει μερικά αποτελέσματα στο επόμενο PE και γράφει τα επεξεργασμένα δεδομένα πίσω στην κοινή μνήμη. Προηγούμενο PE είναι αυτό που βρίσκεται στα αριστερά του τρέχοντος PE στον δακτύλιο, ενώ επόμενο στοιχείο είναι αυτό που βρίσκεται στα δεξιά του τρέχοντος στοιχείου στον δακτύλιο.

Στο σχήμα 10.2 μπορούμε να δούμε το γενικό διάγραμμα του αλγορίθμου (H-DMPS), το οποίο περιλαμβάνει τον δακτύλιο των PE, καθώς και την σύνδεση των PEs με τον ελεγκτή και με την κοινή μνήμη.

Ο αλγόριθμος $H-DMPS$ περιγράφεται συνοπτικά στην παρακάτω αλγοριθμική περιγραφή, ενώ περισσότερες πληροφορίες για την δομή και λειτουργία του αλγορίθμου δίνονται στις υποενότητες 10.2.1, 10.2.2, 10.2.3.

Από την πλευρά του Ελεγκτή μνήμης:

1. Έλεγξε αν υπάρχουν αιτήσεις (1 ως m) για πρόσβαση στην Κεντρική Μνήμη στην είσοδο, τα οποία αντιστοιχούν στα m PEs ($PE_1 \ldots PE_m$).
2. Από αυτές τις αιτήσεις, διάλεξε την αίτηση k που έχει την μεγαλύτερη προτεραιότητα.
3. Δώσε πρόσβαση στο PE_k
4. **Όσο το PE_k διατηρεί ενεργή την αίτηση του**

   Δώσε αποκλειστική πρόσβαση στην Μνήμη στο PE_k
5. Πήγαινε στο βήμα 1

Απο την πλευρά του Επεξεργαστικού στοιχείου PE_k, όπου $k = 1, \ldots, m$:

Έναρξη εκτέλεσης νέας ομάδας επαναλήψεων:

1. Υπολόγισε τις παραμέτρους της ομάδας επαναλήψεων (άνω και κάτω όρια)
2. **Για όλες τις υποομάδες της τρέχουσας ομάδας επαναλήψεων**
   - Ζήτα πρόσβαση στην Κεντρική Μνήμη από τον Ελεγκτή μνήμης
   - Περίμενε μέχρι να εγκριθεί η αίτηση:
     - Διάβασε τα δεδομένα της τρέχουσας υποομάδας από την Κεντρική Μνήμη
     - Ελευθέρωσε την Κεντρική Μνήμη
   - Διάβασε τα μερικά αποτελέσματα από τον προηγούμενο Επεξεργαστικό στοιχείο (PE_{k-1})
   - Εκτέλεσε τους υπολογισμούς της τρέχουσας υποομάδας υπολογισμών.
   - Γράψε τα μερικά αποτελέσματα στον επόμενο Επεξεργαστικό στοιχείο (PE_{k+1})
   - Ζήτα πρόσβαση στην Κεντρική Μνήμη από τον Ελεγκτή μνήμης
   - Περίμενε μέχρι να εγκριθεί η αίτηση:
     - Γράψε τα δεδομένα της τρέχουσας υποομάδας επαναλήψεων στην Κεντρική Μνήμη

– Ελευθέρωσε την Κεντρική Μνήμη

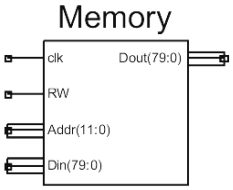
3. **Αν υπάρχουν και άλλες διαθέσιμες ομάδες επαναλήψεων**

   Τότε Πήγαινε στο βήμα 1

   Αλλιώς Πήγαινε στον τερματισμό

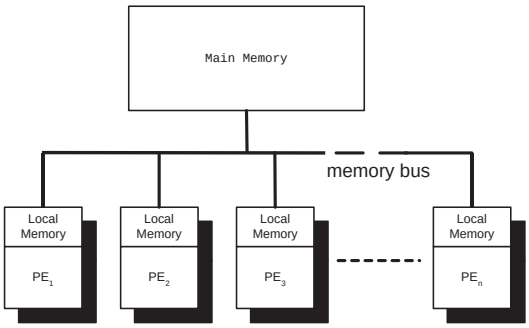
10.2.1 Αρχιτεκτονική μνήμης

Η μνήμη περιέχει τα αρχικά δεδομένα και συγκεντρώνει τα αποτελέσματα μετά το τέλος της εκτέλεσης. Κάθε PE διαβάζει ένα κομμάτι μνήμης, το επεξεργάζεται και εγγράφει πίσω αποτελέσματα πάνω στις ίδιες θέσεις. Έτσι η μνήμη περιέχει τα ταυτόχρονα αρχικά δεδομένα αλλά και τελικά αποτελέσματα, μέχρι το σημείο που έχει προχωρήσει ο υπολογισμός.

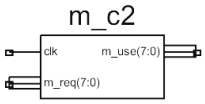


Σχήμα 10.3: Το σύμβολο της μνήμης

Η οργάνωση της μνήμης του συστήματος παρουσιάζεται στο σχήμα 10.4. Η κοινή μνήμη μπορεί να προσπελαστεί από όλα τα PE με την χρήση του διαύλου μνήμης. Παράλληλα με την κοινή μνήμη, κάθε PE διαθέτει και τοπική μνήμη. Η τοπική μνήμη είναι αρκετά μεγάλη για να αποθηκεύσει τα δεδομένα μιας υποομάδας επαναλήψεων. Πριν από την επεξεργασία μιας υποομάδας κάθε PE μεταφέρει τα δεδομένα που απαιτούνται από την κοινή στην τοπική του μνήμη. Με αυτό τον τρόπο μειώνεται ο συνολικός αριθμός προσπελάσεων στην κοινή μνήμη. Ένα άλλο πλεονέκτημα της αρχιτεκτονικής χονδρού κόκκου είναι ότι προσφέρει πολύ καλύτερη τοπικότητα δεδομένων δε σχέση με την προσέγγιση του παραλληλισμού λεπτού κόκκου. Τα δεδομένα που πρέπει να μεταφερθούν από την κοινή μνήμη σε κάποια τοπική μνήμη για την επεξεργασία μιας υποομάδας επαναλήψεων, είναι αποθηκευμένα σε διαδοχικές θέσεις μνήμης. Οι κύκλοι ρολογιού που απαιτούνται για την μεταφορά των δεδομένων είναι ανάλογος του μήκους του διαύλου μνήμης. Αν ο δίαυλος έχει αρκετό πλάτος, τότε τα δεδομένα μιας υποομάδας μπορούν να μεταφερθούν ακόμα και μέσα σε ένα κύκλο ρολογιού. Από αυτό συμπεραίνουμε ότι υπάρχει μια σχέση μεταξύ της επιφάνειας που μπορούμε να διαθέσουμε για την δημιουργία του διαύλου της μνήμης και του κόστους μεταφοράς δεδομένων. Για παράδειγμα, μια τα δεδομένα μιας υποομάδας, μεγέθους 100 λέξεων μπορούν να μεταφερθούν σε ένα κύκλο χρησιμοποιώντας έναν δίαυλο πλάτους 100 λέξεων, σε 10 κύκλους με ένα δίαυλο πλάτους 10 λέξεων ή σε 100 κύκλους σε ένα δίαυλο πλάτους μιας λέξης.



Σχήμα 10.4: Η αρχιτεκτονική της Μνήμης.

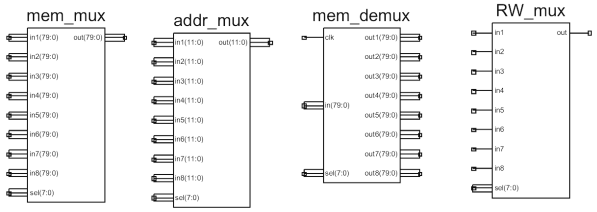


Σχήμα 10.5: Ο ελεγκτής Μνήμης.

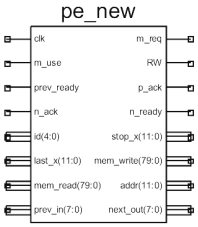
10.2.2 Ο Ελεγκτής μνήμης

Ο ελεγκτής μνήμης (ΕΜ) συντονίζει την πρόσβαση των PEs στην κεντρική-κοινή μνήμη. Ο ΕΜ έχει μια είσοδο και μία έξοδο, όπως φαίνεται στο σχήμα 10.5. Στην είσοδο *m_req* λαμβάνει αιτήσεις για πρόσβαση στην μνήμη από τα PEs. Ο ΕΜ αποφασίζει σε πιο PE θα δώσει πρόσβαση σύμφωνα με ένα απλή μέθοδο προτεραιοτήτων. Η απόφαση του εγγράφεται στην έξοδο *m_use*. Αν μόνο ένα PE ζητάει πρόσβαση, τότε ο ΕΜ αποφασίζει να επιτρέψει σε αυτό το PE να προσπελάσει την μνήμη. Αν ο αριθμός των PEs είναι μεγαλύτερος, τότε ο ΕΜ επιλέγει το PE με τον μικρότερο αριθμό σειράς (id). Το επιλεγμένο PE στην συνέχεια διαθέτει αποκλειστική πρόσβαση στην μνήμη για ανάγνωση ή εγγραφή. Η υλοποίηση του παραπάνω σχήματος είναι αρκετά απλή. Η είσοδος και η έξοδος του PE έχουν μήκος m bits, όπου m είναι ο συνολικός αριθμός των PEs. Κάθε bit αντιστοιχεί σε ένα PE, με το πρώτο bit να αντιστοιχεί στο PE με αριθμό 1 (PE_1), το δεύτερο bit στο PE με αριθμό 2 (PE_2), μέχρι το bit m που αντιστοιχεί στο PE με αριθμό m (PE_m). Ο ΕΜ αντιλαμβάνεται ένα λογικό 1 σε κάποιο bit της εισόδου του σαν μια αίτηση για πρόσβαση στην μνήμη και την θέση του bit αυτού σαν τον αριθμό του αιτούμενου PE. Ο ΕΜ επιλέγει το λιγότερο σημαντικό bit εισόδου που είναι σε λογικό 1 και ενεργοποιεί το αντίστοιχο bit στην έξοδο.

Η έξοδος από τον ελεγκτή μνήμης τροφοδοτεί και την σειρά από πολυπλέκτες/αποπλέκτες που φαίνονται στο σχήμα 10.6. Πιο συγκεκριμένα, χρησιμοποιείται ένας πολυπλέκτης που συγκεντρώνει τα σήματα επιλογής ανάγνωσης/εγγραφής από τα επεξεργαστικά στοιχεία προς την μνήμη (RW_mux), ένας πολυπλέκτης από τους εργάτες προς τον δίαυλο διευθύνσεων της μνήμης (ADDR_mux), ένας πολυπλέκτης (MEM_mux) και ένας αποπλέκτης (MEM_demux) από τους διαύλους εισόδου και εξόδου δεδομένων



Σχήμα 10.6: Υπομονάδες πρόσβασης στη μνήμη.



Σχήμα 10.7: Σύμβολο του επεξεργαστικού στοιχείου.

των PEs προς την είσοδο και έξοδο της μνήμης.

Όλοι οι πολυπλέκτες έχουν αριθμό εισόδων ίσο με τον αριθμό των διαθέσιμων PEs και μια μοναδική έξοδο. Η συσχέτιση της εξόδου με την κατάλληλη είσοδο γίνεται με την χρήση της εισόδου επιλογής (sel), η οποία τροφοδοτείται από την έξοδο m_use του ΕΜ. Ο αποπλέκτης της μνήμης έχει μια είσοδο και αριθμό εξόδων ίσο με τον αριθμό των διαθέσιμων PEs. Η συσχέτιση της εισόδου με την κατάλληλη έξοδο γίνεται πάλι βάση της εξόδου m_use του ΕΜ.

10.2.3 Τα Επεξεργαστικά Στοιχεία

Το πιο σημαντικό μέρος του συστήματος είναι τα επεξεργαστικά στοιχεία (PEs), τα οποία είναι ειδικά σχεδιασμένα για να εκτελούν το σώμα του βρόχου της εκάστοτε εφαρμογής. Το PE στηρίζει την λειτουργία του σε μια μηχανή πεπερασμένων καταστάσεων (finite state machine) (σχ. 10.8) και στην τοπική μνήμη που περιγράφεται στην υποενότητα 10.2.1. Τα PEs έχουν ένα αριθμό ταυτότητας από 1 έως m, όπου m είναι το πλήθος των PEs. Το PE_1 θεωρούμε ότι βρίσκεται στην αρχή του δακτυλίου ενώ το PE_m είναι στο τέλος του. Το στοιχείο 1 αρχίζει την επεξεργασία και ενεργοποιεί το στοιχείο 2 το οποίο αρχίζει την επεξεργασία και ενεργοποιεί με την σειρά του το στοιχείο 3, κτλ μέχρι να φτάσουμε στο στοιχείο m. Από εκεί και πέρα όλα τα στοιχεία θα δουλεύουν παράλληλα σε μορφή σωλήνωσης.

Στην συνέχεια θα δούμε περιληπτικά τις λειτουργίες που εκτελούνται σε κάθε κατάσταση του διαγράμματος καταστάσεων του σχήματος 10.8. Η παράλληλη εκτέλεση ξεκινάει, με όλα τα PEs να βρίσκονται στην κατάσταση $s0$. Σε αυτή την κατάσταση γίνεται η δρομολόγηση, κατά την οποία κάθε PE υπολογίζει τα όρια και το μέγεθος των επόμενων ομάδων επαναλήψεων που θα εκτελέσουν στην συνέχεια. Πρόκειται

για μια κατανεμημένη μέθοδο δρομολόγησης. Αρχικά το PE_1, ξεκινάει από το σημείο 0 της διάστασης δρομολόγησης u_c και προσθέτει σε αυτό το μέγεθος της ομάδας επαναλήψεων V, σύμφωνα με τον επιλεγμένο αλγόριθμο δρομολόγησης, που στην περίπτωση μας είναι ο CSS. Έτσι βρίσκει το πάνω όριο της επόμενης ομάδας επαναλήψεων. Στην συνέχεια το PE_1 γράφει το άνω όριο που υπολόγισε στην πόρτα εξόδου $stop_x$ και προχωράει στην κατάσταση $s1$. Το επόμενο PE, το οποίο είναι το PE_2 διαβάζει την τιμή αυτή στην είσοδο $last_x$ και ορίζει αυτή την τιμή σαν το κάτω άκρο της επόμενης ομάδας επαναλήψεων που θα υπολογίσει στην διάσταση u_c. Προσθέτει στο κάτω άκρο την τιμή V για να βρει το άνω άκρο της ομάδας επαναλήψεων του, το οποίο γράφει με την σειρά του στην έξοδο $stop_x$ και προχωράει στη κατάσταση $s1$. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να φτάσουμε στο PE_m. Οι καταστάσεις $s1$ με $s3$ σχετίζονται με την μνήμη. Στην $s1$, τα PEs υπολογίζουν την διεύθυνση της μνήμης στην οποία είναι αποθηκευμένα τα δεδομένα τα οποία πρέπει να μεταφερθούν στην τοπική μνήμη των PEs, η διεύθυνση αυτή τοποθετείται στον δίαυλο διευθύνσεων και προχωρούν στην κατάσταση $s2$. Σε αυτή την κατάσταση, κάθε PE κάνει μια αίτηση για πρόσβαση στην μνήμη, θέτοντας σε λογικό 1 το bit της πόρτας εξόδου τους m_req. Στην συνέχεια περιμένουν στην κατάσταση αυτή μέχρι να τους δοθεί άδεια για πρόσβαση στην μνήμη από τον EM. Όταν γίνει αυτό, προχωρούν στην κατάσταση $s3$. Σε αυτή την κατάσταση πραγματοποιείται η μεταφορά δεδομένων από την κοινή στην τοπική μνήμη του PE που διαθέτει την αποκλειστική πρόσβαση στην μνήμη. Όταν η μεταφορά δεδομένων ολοκληρωθεί, το PE περνάει στην κατάσταση $s4$, αφού πρώτα θέσει το bit της πόρτας m_req σε λογικό 0, για να δηλώσει ότι δεν επιθυμεί πια πρόσβαση στην κοινή μνήμη.

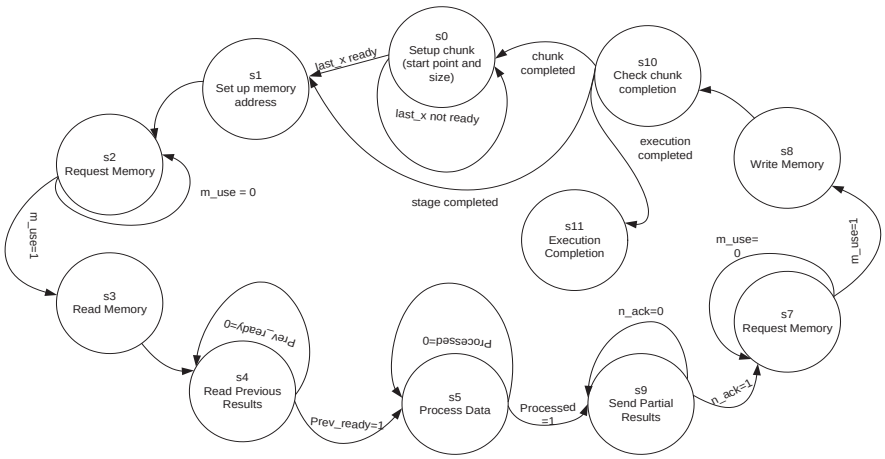
Στη κατάσταση $s4$ λαμβάνει χώρα η πρώτη φάση του συγχρονισμού. Το τρέχων PE περιμένει για το σήμα $prev\_ready$. Η λήψη του σήματος αυτού σημαίνει ότι έχει ολοκληρωθεί η επεξεργασία μιας υποομάδας επαναλήψεων από το προηγούμενο PE. Το τρέχων PE διαβάζει στην είσοδο $prev\_in$ τα μερικά αποτελέσματα, δηλαδή τα επεξεργασμένα δεδομένα που χρειάζεται από το προηγούμενο PE. Για να ειδοποιήσει το προηγούμενο PE ότι διάβασε τα δεδομένα στέλνει το σήμα p_ack και προχωράει στη κατάσταση $s5$. Σε αυτό το σημείο το PE έχει όλα τα δεδομένα που χρειάζεται για να επεξεργαστεί μια υποομάδα επαναλήψεων, η $s5$ είναι η κατάσταση επεξεργασίας. Το δεύτερο μέρος του συγχρονισμού πραγματοποιείται στην κατάσταση $s6$, στην οποία το PE γράφει τα μερικά αποτελέσματα του στην πόρτα εξόδου $next\_out$ και στέλνει το σήμα n_ready, έτσι ώστε να ειδοποιήσει το επόμενο PE να τα διαβάσει. Στην συνέχεια περιμένει στην κατάσταση αυτή μέχρι να να λάβει την βεβαίωση λήψης μέσω του σήματος n_ack. Στις καταστάσεις $s7$ και $s8$ τα PE κάνουν αιτήσεις για πρόσβαση στην μνήμη προκειμένου να γράψουν τα αποτελέσματα της επεξεργασίας της υποομάδας επαναλήψεων, με τρόπο παρόμοιο με αυτό των καταστάσεων $s2$ και $s3$. Σε αυτό το σημείο, στην κατάσταση $s10$, η εκτέλεση μιας υποομάδας επαναλήψεων έχει ολοκληρωθεί και τα PEs πρέπει να οδηγηθούν σε μια από τις παρακάτω καταστάσεις:

- αν βρίσκονται στο τέλος της τρέχουσας ομάδας επαναλήψεων, η επόμενη κατάσταση πρέπει να είναι η $s0$, για να ξεκινήσει την εκτέλεση μιας νέας ομάδας επαναλήψεων.

- αν δεν βρίσκονται στο τέλος μιας ομάδας επαναλήψεων, η επόμενη κατάσταση

πρέπει να είναι η $s1$, έτσι ώστε να ξεκινήσει την εκτέλεση μια νέας υποομάδας επαναλήψεων.

- Αν αυτό είναι το τέλος της ομάδας επαναλήψεων και δεν υπάρχουν άλλες διαθέσιμες υποομάδες, τότε βρισκόμαστε στο τέλος της παράλληλης εκτέλεσης και η επόμενη κατάσταση πρέπει να είναι η $s11$ η οποία είναι και η τελική κατάσταση.



Σχήμα 10.8: Το διάγραμμα μετάβασης καταστάσεων των επεξεργαστικών στοιχείων.

10.3 Ανάλυση

Σε αυτή την ενότητα αναλύουμε σύντομα τον αλγόριθμο $H-DMPS$. Καταλήγουμε σε εκτίμηση του συνολικού παράλληλου χρόνου εκτέλεσης και αξιολογείται η επίδραση του μεγέθους του διαύλου μνήμης στην συνολική επίδοση του αλγορίθμου δρομολόγησης.

Στην συνέχεια θα χρησιμοποιηθούν οι παρακάτω συμβολισμοί:

• t_{sr} είναι το κόστος αποστολής -λήψης ανά λέξη, το οποίο σχετίζεται με μήκος των διανυσμάτων εξάρτησης (για μοναδιαίες εξαρτήσεις $t_{sr} = 1$).

• $T_s = T_r = t_{sr} \times h$, οι κύκλοι ρολογιού που απαιτούνται για την αποστολή/ λήψη των μερικών δεδομένων μιας υποομάδας επαναλήψεων από το προηγούμενο PE.

• W_{bus} είναι το μήκος του διαύλου της μνήμης σε λέξεις.

• t_{mr} και t_{mw} είναι το κόστος ανάγνωσης/εγγραφής στην μνήμη (ο αριθμός των λέξεων που πρέπει να μεταφερθούν από την μνήμη) για κάθε επανάληψη.

• $T_{mr} = t_{mr} \times V \times h/W_{bus}$ και $T_{mw} = t_{mw} \times V \times h/W_{bus}$, οι κύκλοι ρολογιού για την εγγραφή/ανάγνωση των δεδομένων μιας υποομάδας επαναλήψεων από/προς την μνήμη.

• T_{sch} Το κόστος δρομολόγησης σε κύκλου ρολογιού.

• $T_p = t_p \times V \times h$ είναι ο χρόνος επεξεργασίας μια υποομάδας επαναλήψεων σε κύ-

κλους ρολογιού, όπου t_p είναι το κόστος επεξεργασίας ανά επανάληψη.

Στο Σχήμα 10.9 δίνεται η χρονοδρομολόγηση 12 ομάδων επαναλήψεων σε 4 PEs. Λόγω των εξαρτήσεων δεν μπορούν όλες οι υποομάδες να εκτελεστούν ταυτόχρονα, έχουμε αντίθετα εκτέλεση σωλήνωσης. Οι αριθμοί μέσα στα κουτάκια του Σχήματος 10.9 προσδιορίζει το χρονικό βήμα κατά το οποίο η αντίστοιχη υποομάδα υπολογισμών μπορεί να εκτελεστεί. Όπως βλέπουμε, σχηματίζονται 3 διακριτές σωληνώσεις και η εκτέλεση ολοκληρώνεται σε 27 χρονικά βήματα. Στο τέλος κάθε χρονικού βήματος, κάθε Επεξεργαστικό Στοιχείο ανταλλάσσει δεδομένα (μερικά αποτελέσματα) με τα γειτονικά του. Ο αριθμός των χρονικών βημάτων είναι ανάλογος του πλήθους των Επεξεργαστικών Στοιχείων και του μεγέθους των ομάδων επαναλήψεων και δίνεται από την σχέση:

$$N = (m-1) + k \times U_s/h$$

Όπου m είναι το πλήθος των Επεξεργαστικών Στοιχείων και $k = U_c/(V \times m)$ είναι ο αριθμός των σωληνώσεων. Σε κάθε χρονικό βήμα εκτελείται ο υπολογισμός μιας υποομάδας επαναλήψεων σε χρόνο $T_{sbch} = T_{mr} + T_r + T_p + T_s + T_{mw} + T_{sch}$, ο οποίος περιλαμβάνει τον χρόνο υπολογισμού, τον χρόνο ανάγνωσης και εγγραφής στην μνήμη, τον χρόνο ανταλλαγής μερικών αποτελεσμάτων, καθώς και το το κόστος δρομολόγησης. Με βάση τον αριθμό των χρονικών βημάτων μπορούμε να προσεγγίσουμε τον συνολικό παράλληλο χρόνο. Οπότε ο συνολικός παράλληλος δίνεται από την σχέση:

$$T_{par} = N \times T_{sbch}$$

Στην παραπάνω σχέση, υποθέσαμε ότι τα επεξεργαστικά στοιχεία δεν μένουν ανενεργά, περιμένοντας για να αποκτήσουν πρόσβαση στην μνήμη. Για να ισχύσει αυτό θα πρέπει τα Επεξεργαστικά στοιχεία να ξοδεύουν περισσότερο χρόνο σε υπολογισμούς παρά σε μεταφορές δεδομένων από και προς την μνήμη, δηλαδή όταν $T_p \geq T_{mr} + T_{mw}$ ή αλλιώς, ο αριθμός των Επεξεργαστικών στοιχείων θα πρέπει να είναι $m \leq \lceil T_p/(T_{mr}+T_{mw}) \rceil$. Αν αντικαταστήσουμε τις τιμές των T_{mr} και T_{mw} στην παραπάνω σχέση έχουμε :

$$m \ \leq \ \lceil \frac{t_p}{(t_{mr} + t_{mw})} \times W_{bus} \rceil$$

Από αυτή την σχέση, μπορούμε να δούμε ότι ο αριθμός των Επεξεργαστικών Στοιχείων που μπορούν να χρησιμοποιηθούν αποδοτικά από τον αλγόριθμο $H - DMPS$ είναι ευθέως ανάλογος μεγέθους του διαύλου της μνήμης.

pipeline 3	PE_3	20	21	22	23	24	25	26	27
	PE_2	19	20	21	22	23	24	25	26
	PE_2	18	19	20	21	22	23	24	25
	PE_1	17	18	19	20	21	22	23	24
pipeline 2	PE_3	12	13	14	15	16	17	18	19
	PE_2	11	12	13	14	15	16	17	18
	PE_2	10	11	12	13	14	15	16	17
	PE_1	9	10	11	12	13	14	15	16
pipeline 1	PE_3	4	5	6	7	8	9	10	11
	PE_2	3	4	5	6	7	8	9	10
	PE_2	2	3	4	5	6	7	8	9
	PE_1	1	2	3	4	5	6	7	8

Σχήμα 10.9: Δρομολόγηση 12 ομάδων επαναλήψεων σε 4 επεξεργαστικά στοιχεία.

10.4 Πειραματική αξιολόγηση

Πειραματική διάταξη. Προκειμένου να εκτιμήσουμε την απόδοση της μεθόδου που περιγράφεται σε αυτό το κεφάλαιο, ο αλγόριθμος H-DMPS μοντελοποιήθηκε σε γλώσσας περιγραφής υλικού *verilog*, με την χρήση του πακέτου εξομοίωσης *modelsim*. Οι μετρήσεις δίνονται για 2,4,6, και 8 PEs. Επίσης χρησιμοποιήθηκαν δύο διαφορετικά μήκη διαύλου μνήμης. Στην πρώτη περίπτωση ο δίαυλος είχε μήκος μιας λέξης, οπότε μόνο μια λέξη μπορούσε να μεταφερθεί σε κάθε κύκλο ρολογιού από την κοινή στην τοπική μνήμη, ενώ στην δεύτερη περίπτωση 10 λέξεις μπορούσαν να μεταφερθούν ταυτόχρονα, αφού ο δίαυλος της μνήμης είχε μήκος 10 λέξεων. Με αυτόν τον τρόπο θέλουμε να διερευνήσουμε την σχέση μεταξύ των αυξημένων απαιτήσεων σε επιφάνεια υλικού (πύλες) για την υλοποίηση μεγαλύτερων διαύλων και την αύξηση της επιτάχυνσης. Το διάστημα συγχρονισμού ήταν σε κάθε περίπτωση μια επανάληψη ($H = 1$). Στην συνέχεια συγκρίναμε τα αποτελέσματα που συγκεντρώσαμε με αυτά που δίνονται από τον αλγόριθμο που περιγράφεται στο [44].

Εφαρμογή. Ο φωλιασμένος βρόχος που χρησιμοποιήθηκε στα πειράματα είναι ένα κλασσικό παράδειγμα μερικής διαφορικής εξίσωσης, που επιλύεται με την χρήση επαναληπτικών μεθόδων. Ο ψευδοκώδικας που παραλληλοποιήθηκε δίνεται παρακάτω και είναι μέρος μιας εφαρμογής που υπολογίζει την πίεση που παρατηρείται σε σημεία μιας ομοιόμορφης πλάκας όταν συγκεκριμένη πίεση ασκείται σε μια από τις γωνίες της.

```
for (i=0;i<160;i++)
 for (j=0;j<63;j++)  {
     P=0.5*Pm[i-1,j-1]+0.5*Pm[i-2,j-2];
     if (s[i,j]<zmax) {
         s[i,j]+=k[i,j]*a[i,j]*P;
Pm[i,j]=(1-a[i,j]*P);
 } else
        Pm[i,j]=P;
```

Πίνακας 10.1: Αποτελέσματα εξομοίωσης. Χρόνος εκτέλεσης (σε κύκλους ρολογιού) για 2,4,6 και 8 PEs

PE	1 λέξη/κύκλο	10 λέξεις/κύκλο
2	181730	154514
4	92100	78474
6	61015	54211
8	58271	40812

```
          }
         }
LastVertexPressure=Pm[159,62];
```

Στον παραπάνω ψευδοκώδικα, επιλέγουμε τον εξωτερικό βρόχο (δείκτης i) για διάσταση δρομολόγησης (u_c), και τον εσωτερικό βρόχο (δείκτης j) για διάσταση συγχρονισμού (u_s).

Όπως μπορούμε να δούμε από τα όρια του φωλιασμένου βρόχου, το σώμα του βρόχου επαναλαμβάνεται $160 \times 63 = 10080$ φορές.

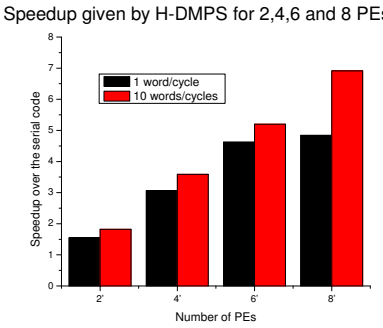
Για να μπορέσουμε να αξιολογήσουμε την απόδοση του αλγορίθμου $H-DMPS$, κάνουμε τις παρακάτω υποθέσεις. Υποθέτουμε, ότι όλες οι δηλώσεις μπορούν να εκτελεστούν μέσα σε ένα κύκλο ρολογιού, εκτός από πολλαπλασιασμούς και διαιρέσεις, που απαιτούν 5 κύκλους. Αυτή η υπόθεση μπορεί να μην είναι απόλυτα ακριβής, όμως στην πράξη μας βοηθάει στην καλύτερη αξιολόγηση της προτεινόμενης μεθόδου. Επίσης, κατά την υλοποίηση, δεν έχει γίνει καμία βελτιστοποίηση, η μετασχηματισμός του πηγαίου κώδικα της εφαρμογής.

Ο παραπάνω βρόχος εκτελείται κατά μέσω όρο σε 28 κύκλους οπότε στην καλύτερη περίπτωση μία σειριακή εκτέλεση θα ολοκληρωνόταν σε $10080 \times 28 = 282240$ κατά μέσω όρο. Θα χρησιμοποιήσουμε αυτή την τιμή για να εκτιμήσουμε την αποδοτικότητα του αλγόριθμου και να υπολογίσουμε τα μεγέθη των επιταχύνσεων στην συνέχεια.

10.4.1 Αποτελέσματα.

Στον Πίνακα 10.4.1 παρουσιάζουμε τα αποτελέσματα της εξομοίωσης, τα οποία συγκεντρώθηκαν βάση των παραπάνω υποθέσεων με την χρήση της λειτουργίας εξομοίωσης συμπεριφοράς (behavioral simulation) του εξομοιωτή *modelsim* .

Στον Πίνακα 10.4.1 δίνεται ο αριθμός κύκλων ρολογιού για την εκτέλεση του παράλληλου αλγόριθμου σε 2,4, 6 και 8 Επεξεργαστικών Στοιχείων και για την περίπτωση της ταυτόχρονης μεταφοράς 1 ή 10 λέξεων από την κοινή στις τοπικές μνήμες. Η επιτάχυνση από την σειριακή εκτέλεση, η οποία ολοκληρώνεται σε 282240, όπως περιγράφηκε παραπάνω παρουσιάζεται στο Γράφημα 10.10. Όπως μπορούμε να δούμε, σε όλες τις περιπτώσεις ο παράλληλος αλγόριθμος προσφέρει σημαντική επιτάχυνση σε σχέση με την σειριακό εκτέλεση. Ακόμα, μπορούμε να δούμε ότι οι περιπτώσεις



Σχήμα 10.10: Επιτάχυνση του H-DMPS, (2,4,6,8 PEs)

Πίνακας 10.2: Χρόνος εκτέλεσης της λεπτόκοκκης μεθόδου

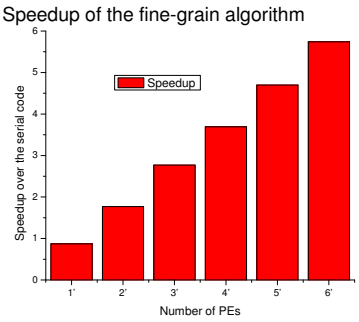
PE	Χρόνος εκτέλεσης (κύκλοι ρολογιού)	επιτάχυνση
1	323598	0.87
2	158697	1.77
3	103796	2.77
4	76395	3.69
5	59994	4.70
6	49093	5.74

στις οποίες έχουμε μεγαλύτερο μήκος διαύλου μνήμης, δηλαδή ταυτόχρονη μεταφορά 10 λέξεων, έχουμε βελτίωση 11% ως 29% σε σχέση με τις αντίστοιχες περιπτώσεις στις οποίες όμως το μήκος του διαύλου είναι μια μόνο λέξη.

Το επόμενο βήμα είναι να συγκρίνουμε την απόδοση του $H-DMPS$ με αυτή άλλων αλγορίθμων της βιβλιογραφίας. Αφού δεν αναφέρονται άλλοι δυναμικοί και χονδρόκοκκοι αλγόριθμοι, για την σύγκριση θα χρησιμοποιήσουμε ένα πρόσφατο δυναμικό αλγόριθμο που ακολουθεί την προσέγγιση λεπτού κόκκου. Η σύγκριση είναι έγκυρη, αφού και οι δύο αλγόριθμοι προορίζονται και για την ίδια αρχιτεκτονική, αυτή του επαναπρογραμματιζόμενου υλικού και για τις ίδιες εφαρμογές, αυτές που περιέχουν φωλιασμένους βρόχους με ομοιόμορφες εξαρτήσεις. Η σύγκριση αυτή δίνει επίσης την δυνατότητα να διαφανούν τα τυχόν πλεονεκτήματα και μειονεκτήματα της κάθε προσέγγισης.

Ο Πίνακας 10.4.1 συγκεντρώνει τα αποτελέσματα που παρέχονται από τον αλγόριθμο λεπτού-κόκκου που περιγράφεται στο [44] για το ίδιο κώδικα εφαρμογής που περιγράφηκε παραπάνω. Ο βέλτιστος αριθμός PEs, σύμφωνα με τις εξισώσεις που περιέχονται στο [44] είναι $m_{opt} = 6$, πέρα από αυτό τον αριθμό η απόδοση δεν παρουσιάζει βελτίωση. Η επιτάχυνση που προσφέρει αυτή η μέθοδος για 1 έως 6 Επεξεργαστικά Στοιχεία δίνεται στο γράφημα 10.11.

Όπως μπορούμε να δούμε στον Πίνακα 10.4.1, η προσέγγιση λεπτού κόκκου επιτυγχάνει σχεδόν ιδανική επιτάχυνση αφού χρησιμοποιώντας 6 Επεξεργαστικά Στοι-



Σχήμα 10.11: Επιτάχυνση της λεπτόκοκκης μεθόδου, (1 ως 6 PEs)

χεία ο παράλληλος αλγόριθμος είναι 5.74 φορές πιο γρήγορος από τον σειριακό. Στον ίδιο αριθμό Επεξεργαστικών Στοιχείων, ο H-DMPS παρουσιάζει μια μεγαλύτερη απόκλιση από την ιδανική περίπτωση αφού η επιτάχυνση που προσφέρει είναι 5.20 σε σχέση με τον σειριακό αλγόριθμο, στην περίπτωση του διαύλου μνήμης των 10 λέξεων. Ωστόσο, το πλεονέκτημα του χονδρόκοκκου αλγόριθμου είναι ότι προκαλεί μικρότερη συμφόρηση στην μνήμη και έτσι μπορούν να χρησιμοποιηθούν περισσότερα Επεξεργαστικά Στοιχεία. Με την προσθήκη δυο περισσοτέρων Επεξεργαστικών Στοιχείων, δηλαδή για $m = 8$, Ο αλγόριθμος H-DMPS πετυχαίνει επιτάχυνση 6.91, δηλαδή παρέχει μια βελτίωση 17% σε σχέση με την καλύτερη απόδοση του λεπτόκοκκου αλγόριθμου.

10.5 Συμπεράσματα

Σε αυτή την ενότητα παρουσιάστηκε ένας νέος δυναμικός αλγόριθμος για την δρομολόγηση φωλιασμένων βρόχων με εξαρτήσεις, σε επαναπρογραμματιζόμενο υλικό. Ο αλγόριθμος αυτός προσφέρει ένα χονδρόκοκκο τεμαχισμό του φωλιασμένου βρόχου που παρουσιάζεται αποδοτικότερος από τον λεπτόκοκκο τεμαχισμό, σε όρους χρόνου εκτέλεσης, τοπικότητας δεδομένων, και διαχείρισης μνήμης. Ο παραπάνω ισχυρισμός επιβεβαιώνεται από τα πειραματικά αποτελέσματα στα οποία ο αλγόριθμος H-DMPS παρέχει σημαντική επιτάχυνση σε σχέση και με τον αντίστοιχο σειριακό αλγόριθμο, αλλά και σε σχέση με ένα πρόσφατο δυναμικό αλγόριθμο που προσφέρει παραλληλισμό λεπτού κόκκου.

Κεφάλαιο 11

Επίλογος

Στα προηγούμενα κεφάλαια παρουσιάστηκε μια σειρά από νέους δυναμικούς αλγορίθμους δρομολόγησης. Οι αλγόριθμοι αυτοί σχεδιάστηκαν για να γεφυρώσουν ένα σημαντικό κενό της βιβλιογραφίας, αυτό της δυναμικής δρομολόγησης εφαρμογών που περιέχουν φωλιασμένους βρόχους με εξαρτήσεις. Αυτού του είδους οι εφαρμογές δρομολογούνταν στο παρελθόν αποκλειστικά με την χρήση στατικών μεθόδων. Οι δυναμικοί αλγόριθμοι έρχονται να βελτιώσουν την απόδοση των στατικών αλγορίθμων σε κατανεμημένα συστήματα υπολογιστών. Τα σύγχρονα, κατανεμημένα υπολογιστικά συστήματα συχνά περιλαμβάνουν ετερογενείς πόρους ή άλλους παράγοντες ανομοιογένειας και σε αυτά τα συστήματα η χρήση δυναμικών αλγορίθμων μπορεί να αποδειχθεί πολύ αποδοτική.

Σύνοψη.

Αρχικά δείχνουμε ότι οι υπάρχοντες δυναμικοί αλγόριθμοι μπορούν να βελτιώσουν την απόδοση εφαρμογών χωρίς εξαρτήσεις, και ότι η βελτίωση είναι πιο σημαντική όσο αυξάνονται οι παράγοντες ανομοιογένειας. Στην συνέχεια παρουσιάζουμε μια σειρά από νέους και καινοτόμους δυναμικούς αλγορίθμους που απευθύνονται σε εφαρμογές που περιέχουν εξαρτήσεις. Κάνουμε την υπόθεση ότι το στοιχείο που επηρεάζει περισσότερο την απόδοση ενός παράλληλου αλγορίθμου είναι η ισοκατανομή του υπολογιστικού φορτίου. Σταδιακά, με διαδοχικούς αλγορίθμους προσπαθούμε να βελτιώσουμε την κατανομή του υπολογιστικού φορτίου στους διαθέσιμους επεξεργαστές ανάλογα με την σχετική υπολογιστική τους δύναμη, δηλαδή ανάλογα με την ταχύτητα με την οποία μπορούν να εκτελέσουν υπολογισμούς. Στην συνέχεια ασχολούμαστε με την μεταβολή του εξωτερικού φορτίου του συστήματος, δηλαδή με το φορτίο που δημιουργείται από άλλους χρήστες, στην περίπτωση μη-αφοσιωμένων συστημάτων, και παρουσιάζουμε αλγορίθμους που μπορούν να προσαρμοστούν σε αυτή την μεταβολή. Όλοι οι παραπάνω αλγόριθμοι βασίζονται στο μοντέλο συντονιστή-εργάτη το οποίο ακολουθείται αρκετά συχνά σε αρχιτεκτονικές κατανεμημένης μνήμης. Παρουσιάζουμε επίσης έναν κατανεμημένο αλγόριθμο ο οποίος βελτιώνει την απόδοση του συστήματος από άποψη ικανότητας κλιμάκωσης αλλά όπως αποδεικνύεται μπορεί να προσφέρει και μικρότερο παράλληλο χρόνο, επειδή μπορεί να προσαρμοστεί καλύτερα

στις μεταβολές τους εξωτερικού φορτίου του συστήματος. Τέλος πειραματιζόμαστε με την υλοποίηση δυναμικών μεθόδων σε επαναπρογραμματιζόμενο υλικό, στο οποίο αν και οι παράγοντες τις ανομοιογένειας δεν είναι παρόντες, ωστόσο η απόδοση μπορεί να βελτιωθεί λόγο της καλύτερης χρήσης της μνήμης.

Το γενικό συμπέρασμα είναι ότι οι δυναμικοί αλγόριθμοι είναι μια ασφαλής και πολύ πρακτική επιλογή για την παραλληλοποίηση εφαρμογών που περιέχουν φωλιασμένους βρόχους, είτε με ή χωρίς εξαρτήσεις. Οι αλγόριθμοι πού προτείνουμε είναι αποδοτικοί και εύκολοι στην υλοποίηση τους και καλύπτουν μια αρκετά μεγάλη κατηγορία εφαρμογών. Δεν εγγυώνται την βέλτιστη απόδοση, αλλά υπόσχονται ότι θα κάνουν το καλύτερο δυνατόν, από πλευράς απόδοσης, ανάλογα με τις συνθήκες εκτέλεσης που επικρατούν και με τα χαρακτηριστικά της εφαρμογής.

Μελλοντικές κατευθύνσεις.

Ένα πολύ ενεργό πεδίο έρευνας πού συνδέεται άμεσα με τις ιδέες πού αναπτύχθηκαν και παρουσιάστηκαν στην παρούσα διατριβή είναι αυτό του υπολογιστικού πλέγματος (the Grid). Το υπολογιστικό πλέγμα έχει αρχίσει να ωριμάζει σαν πλατφόρμα εκτέλεσης παραλλήλων εφαρμογών. Το υπολογιστικό πλέγμα είναι στην πραγματικότητα ένα ευρείας έκτασης κατανεμημένο σύστημα, και αποτελεί μια μεγάλη συλλογή από ετερογενείς υπολογιστικούς, αποθηκευτικούς και άλλους πόρους. Τα χαρακτηριστικά του υπολογιστικού πλέγματος ταιριάζουν σε μεγάλο βαθμό με τα χαρακτηριστικά των δυναμικών αλγορίθμων. Ο χρήστης του πλέγματος έχει μια γενική περιγραφή των κόμβων στους οποίους έχει πρόσβαση, ωστόσο μπορεί να ωφεληθεί από την ικανότητα προσαρμογής στις πραγματικές συνθήκες εκτέλεσης που προσφέρουν οι δυναμικοί αλγόριθμοι. Υπάρχουν αρκετές αναφορές στην βιβλιογραφία για εφαρμογή δυναμικών αλγορίθμων στο υπολογιστικό πλέγμα, ωστόσο αυτές στο σύνολο τους αφορούν παράλληλους βρόχους χωρίς εξαρτήσεις. Οι αλγόριθμοι που παρουσιάσαμε μπορούν να εφαρμοστούν με μικρές επεμβάσεις στο υπολογιστικό πλέγμα και η εφαρμογή τους θα παρουσίαζε αρκετό ερευνητικό ενδιαφέρων.

Ένα δεύτερο πεδίο έρευνας θα ήταν αυτό της αντιμετώπισης (αντοχής) σφαλμάτων (fault tolerance). Όσο αυξάνεται ο αριθμός των επεξεργαστών και κυρίως όσο μεγαλώνει η γεωγραφική κατανομή του, όπως στην περίπτωση του υπολογιστικού πλέγματος που αναφέρεται παραπάνω, τόσο μεγαλώνει η πιθανότητα να υπάρξουν σφάλματα. Τα σφάλματα αυτά μπορεί να περιλαμβάνουν απώλεια υπολογιστικών κόμβων, δηλαδή επεξεργαστών, απώλεια επικοινωνιακών συνδέσμων, δηλαδή βλάβες ή συμφόρηση του δικτύου κτλ. Ένας αλγόριθμος δρομολόγησης πρέπει να είναι ικανός να αντιμετωπίζει τέτοιες καταστάσεις για πολλούς διαφορετικούς λόγους: για παράδειγμα μπορεί να μην είναι εύκολο να εξασφαλίσει κάποιος τους υπολογιστικούς πόρους και να μην είναι δυνατόν να επαναλάβει την εκτέλεση σε περίπτωση σφάλματος, ακόμα μπορεί το αποτέλεσμα της εκτέλεσης μπορεί να χρειάζεται να γίνει γνωστό πριν από την λήξη κάποιας προθεσμίας, όπως στην περίπτωση των συστημάτων πραγματικού χρόνου (realtime systems). Από την φύση τους οι δυναμικοί αλγόριθμοι παρουσιάζουν κάποια χαρακτηριστικά αντοχής σε σφάλματα, το πιο εμφανές χαρακτηριστικό είναι ότι οι επεξεργαστές δεν μοιράζονται όλο το πρόβλημα από την αρχή της εκτέλεσης, αλλά η εκτέλεση προχωράει σε μικρά βήματα. Αυτό δίνει κάποια δυνατότητα στον αλγό-

ριθμο να απομονώσει και να αποκλείσει προβληματικούς επεξεργαστές. Οι εγγενείς αυτές δυνατότητες θα πρέπει να ενισχυθούν ώστε να καταλήξουμε με αξιόπιστους και αποδοτικούς αλγορίθμους που δεν θα σπαταλάνε άσκοπα την διαθέσιμη υπολογιστική ισχύ. Στην βιβλιογραφία αναφέρεται αρκετή δουλειά σχετικά με τεχνικές αντιμετώπισης σφαλμάτων και θα ήταν ιδιαίτερα ενδιαφέρον να ενσωματώσουμε τις τεχνικές αυτές στους αλγορίθμους που παρουσιάστηκαν σε αυτή την διατριβή.

Ένα τελευταίο πεδίο έρευνας είναι αυτό της δυναμικής δρομολόγησης εφαρμογών που περιέχουν βρόχους με μη-ομοιόμορφες εξαρτήσεις. Η δρομολόγηση των εφαρμογών αυτών, ακόμα και με στατικές μεθόδους παρουσιάζει αρκετές δυσκολίες, ωστόσο η προσπάθεια παραλληλοποίησης με την χρήση δυναμικών μεθόδων, ενός υποσυνόλου των περιπτώσεων εφαρμογών που περιέχουν μη-ομοιόμορφες εξαρτήσεις θα ήταν ίσως εφικτή, μέσω της απεικόνισης των μη-ομοιόμορφων σε ομοιόμορφες εξαρτήσεις. Με τον τρόπο αυτό θα μεγάλωνε το εύρος εφαρμογής, και κατά συνέπεια η χρησιμότητα των δυναμικών αλγορίθμων.

Αντιστοιχία Αγγλικών–Ελληνικών όρων

A

adaptive– προσαρμοστικός
agglomeration– συσσωμάτωση
anti-dependencies– Αντί-εξαρτήσεις

B

balanced load– εξισορροπημένη κατανομή φορτίου
behavioral simulation– εξομοίωση συμπεριφοράς
benchmark– μετροπρόγραμμα

C

centralized – συγκεντρωτικός
chunk of iterations – ομάδα επαναλήψεων
chunk weighting – στάθμιση ομάδας υπολογισμών
coarse-grain parallelism – παραλληλισμός χονδρού κόκκου
collective communication – συγκεντρωτική επικοινωνία
communication cost – κόστος επικοινωνίας
communication set – σύνολο επικοινωνιών
compile-time – χρόνος μεταγλώττισης
computation cost – κόστος υπολογισμού
computation to communication ratio – λόγος υπολογισμών - επικοινωνιών
computational grid – υπολογιστικό πλέγμα
computer Cluster – συστοιχία υπολογιστών
conditional statement – δήλωση διακλάδωσης
conectionless – ασυνδεσμικό
control dependencies – Εξαρτήσεις ελέγχου

D

data parallel – παραλληλισμός δεδομένων
decentralized – αποκεντρωμένος
dedicated system – αφοσιωμένο σύστημα
delay – καθυστέρηση
demultiplexer – αποπλέκτης
dependence loop – βρόχος με εξαρτήσεις
distributed memory – κατανεμημένη μνήμη
distributed shared memory – κατανεμημένη/κοινή μνήμη
distributed system – κατανεμημένο σύστημα
DOACROSS loop – βρόχος με εξαρτήσεις
DOALL Loop – βρόχος χωρίς εξαρτήσεις

E

exogenous workload – εξωγενές φορτίο

F

fault tolerance – αντοχή σε σφάλματα
fine grain parallelism – παραλληλισμός λεπτού κόκκου
finite state machine – μηχανή πεπερασμένων καταστάσεων
flow dependencies – Εξαρτήσεις ροής
fully distributed – πλήρως αποκεντρωμένος

G

geomagnetically-induced current – γεωμαγνητικά-επαγόμενο ρεύμα
granularity – μέγεθος κόκκου

H

heterogeneity – ανομοιογένεια
heterogeneous – ετερογενής
homogeneity – ομοιογένεια
homogeneous – ομοιογενής

I

idle time – άεργος χρόνος
idle – άεργος
imbalanced load – μη-εξισορροπημένη κατανομή φορτίου
index space – χώρος δεικτών
inspector/executor model μοντέλο επιθεωρητή/εκτελεστή
interarrival time – χρόνος άφιξης
iteration space – χώρος επαναλήψεων

L

lifetime – χρόνος ζωής
linearly decreasing workload – γραμμικά μειούμενο φορτίο
linearly increasing workload – γραμμικά αυξανόμενο φορτίο
load balancing – εξισορρόπηση φορτίου
loosely coupled – χαλαρά συνδεδεμένο

M

master-worker model – μοντέλο συντονιστή-εργάτη
message passing – ανταλλαγή μηνυμάτων
multiplexer – πολυπλέκτης

N

nested loop – φωλιασμένος βρόχος
Network of Computers – δίκτυο υπολογιστών
non-dedicated system – μη-αφοσιωμένο
non-uniform dependencies – μη-ομοιόμορφες εξαρτήσεις
non-uniform loop – μη ομοιόμορφος βρόχος

O

output dependencies – Εξαρτήσεις εξόδου
overhead – κόστος

P

parallel loop – βρόχος χωρίς εξαρτήσεις
pattern – πρότυπο
ping-pong test – δοκιμές αποστολής-λήψης
pipeline instance – στιγμιότυπο της σωλήνωσης
pipeline stage – στάδιο της σωλήνωσης
pipeline – σωλήνωση
pipelining – εκτέλεση σωλήνωσης
point-to-point – σημείο-προσ-σημείο

R

random workload – ακανόνιστο φορτίο
realtime system – σύστημα πραγματικού χρόνου
reconfigurable hardware – επαναπρογραμματιζόμενο υλικό
reducing chunk size algorithms – αλγόριθμοι με φθίνων μέγεθος ομάδας
reference time – χρόνος αναφοράς
reliability – Αξιοπιστία
request – αίτηση

run-queue – ουρά εκτέλεσης
run-time – χρόνος εκτέλεσης

S

scalability – Δυνατότητα κλιμάκωσης
scheduling overhead – κόστος δρομολόγησης
scheduling – δρομολόγηση
Self-Adapting Scheduling – αυτό-προσαρμοζόμενη δρομολόγηση
self-scheduling algorithm – αυτόδρομολογούμενος αλγόριθμος
server – εξυπηρετητής
service rate – ρυθμός εξυπηρέτησης
shared-memory – μοιραζόμενη/κοινή μνήμη
shelf-scheduling – αυτοδρομολόγηση
speedup – επιτάχυνση
startup cost – κόστος αρχικοποίησης
stochastic process – στοχαστική ανέλιξη
sustained bandwidth – σταθερό εύρος ζώνης

T

task parallel – παραλληλισμός εργασιών
task pools – δεξαμενές εργασιών
task – εργασία
throughput – ρυθμοαπόδοση
tightly coupled – στενά συνδεδεμένο
tiling – ομαδοποίηση
time scheduling – χρονοδρομολόγηση

U

uniform dependencies – ομοιόμορφες εξαρτήσεις
virtual computing power – εικονική δύναμη υπολογισμών

W

wavefront – μέτωπο κύματος
weighting στάθμιση
work stealing – κλέψιμο εργασιών
work-queue – ουρά εργασιών
workload variation – διακύμανση φορτίου
workstation – σταθμός εργασίας

A

ανομοιογένεια – heterogeneity
άεργος – idle
άεργος χρόνος – idle time
αίτηση – request
ακανόνιστο φορτίο – random workload
αλγόριθμοι με φθίνων μέγεθος ομάδας – reducing chunk size algorithms
ανταλλαγή μηνυμάτων – message passing
Αντί-εξαρτήσεις – anti-dependencies
αντοχή σε σφάλματα – fault tolerance
αξιοπιστία – reliability
αποκεντρωμένος – decentralized
αποπλέκτης – demultiplexer
ασυνδεσμικό – conectionless
αυτό-προσαρμοζόμενη δρομολόγηση – Self-Adapting Scheduling
αυτοδρομολόγηση – shelf-scheduling

αυτόδρομολογούμενος αλγόριθμος – self-scheduling algorithm
αφοσιωμένο σύστημα – dedicated system

Β

βρόχος με εξαρτήσεις – DOACROSS loop
βρόχος με εξαρτήσεις – dependence loop
βρόχος χωρίς εξαρτήσεις – parallel loop
βρόχος χωρίς εξαρτήσεις – DOALL Loop

Γ

γεωμαγνητικά-επαγόμενο ρεύμα – geomagnetically-induced current
γραμμικά αυξανόμενο φορτίο – linearly increasing workload
γραμμικά μειούμενο φορτίο – linearly decreasing workload

Δ

δεξαμενή εργασιών – task pool
δήλωση διακλάδωσης – conditional statement
διακύμανση φορτίου – workload variation
δίκτυο υπολογιστών – Network of Computers
δοκιμές αποστολής-λήψης – ping-pong test
δρομολόγηση – scheduling
δυνατότητα κλιμάκωσης – scalability

Ε

εικονική δύναμη υπολογισμών – virtual computing power
εκτέλεση σωλήνωσης – pipelining
Εξαρτήσεις ελέγχου – control dependencies
Εξαρτήσεις εξόδου – output dependencies
εξαρτήσεις ροής – flow dependencies
εξισορροπημένη κατανομή φορτίου – balanced load
εξισορρόπηση φορτίου – load balancing
εξομοίωση συμπεριφοράς – behavioral simulation
εξυπηρετητής – server
εξωγενές φορτίο – exogenous workload
επαναπρογραμματιζόμενο υλικό – reconfigurable hardware
επιτάχυνση – speedup
εργασία – task
ετερογενής – heterogeneous

Κ

καθυστέρηση – delay
κατανεμημένη μνήμη – distributed memory
κατανεμημένη/κοινή μνήμη – distributed shared memory
κατανεμημένο σύστημα –distributed system
κλέψιμο εργασιών – work stealing
κόστος αρχικοποίησης – startup cost
κόστος δρομολόγησης – scheduling overhead
κόστος επικοινωνίας – communication cost
κόστος υπολογισμού – computation cost

Λ

λόγος υπολογισμών - επικοινωνιών – computation to communication ratio

Μ

μέγεθος κόκκου – granularity
μετροπρόγραμμα – benchmark

μέτωπο κύματος – wavefront
μη ομοιόμορφος βρόχος – non-uniform loop
μη-αφοσιωμένο – non-dedicated system
μη-εξισορροπημένη κατανομή φορτίου – imbalanced load
μη-ομοιόμορφες εξαρτήσεις – non-uniform dependencies
μηχανή πεπερασμένων καταστάσεων – finite state machine
μοιραζόμενη/κοινή μνήμη – shared-memory
μοντέλο επιθεωρητή/εκτελεστή – inspector/executor model
μοντέλο συντονιστή-εργάτη – master-worker model

Ο

ομάδα επαναλήψεων – chunk of iterations
ομαδοποίηση – tiling
ομοιογένεια – homogeneity
ομοιογενής – homogeneous
ομοιόμορφες εξαρτήσεις – uniform dependencies
ουρά εκτέλεσης – run-queue
ουρά εργασιών – work-queue

Π

παραλληλισμός δεδομένων – data parallel
παραλληλισμός εργασιών – task parallel
παραλληλισμός λεπτού κόκκου – fine grain parallelism
παραλληλισμός χονδρού κόκκου – coarse-grain parallelism
πλήρως αποκεντρωμένος – fully distributed
πολυπλέκτης – multiplexer
προσαρμοστικός – adaptive
πρότυπο – pattern

Ρ

ρυθμοαπόδοση – throughput
ρυθμός εξυπηρέτησης – service rate

Σ

σημείο-προσ-σημείο – point-to-point
στάδιο της σωλήνωσης – pipeline stage
σταθερό εύρος ζώνης – sustained bandwidth
στάθμιση – weighting
στάθμιση ομάδας υπολογισμών – chunk weighting
σταθμός εργασίας – workstation
στενά συνδεδεμένο – tightly coupled
στιγμιότυπο της σωλήνωσης – pipeline instance
στοχαστική ανέλιξη – stochastic process
συγκεντρωτική επικοινωνία – collective communication
συγκεντρωτικός – centralized
σύνολο επικοινωνιών – communication set
συσσωμάτωση – agglomeration
σύστημα πραγματικού χρόνου – realtime system
συστοιχία υπολογιστών – Computer Cluster
σωλήνωση – pipeline

Υ

υπολογιστικό πλέγμα – computational grid

Φ

φωλιασμένος βρόχος – nested loop

χ

χαλαρά συνδεδεμένο – loosely coupled
χρονοδρομολόγηση – time scheduling
χρόνος αναφοράς – reference time
χρόνος άφιξης – interarrival time
χρόνος εκτέλεσης – run-time
χρόνος ζωής – lifetime
χρόνος μεταγλώττισης – compile-time
χώρος δεικτών – index space
χώρος επαναλήψεων – iteration space

Βιβλιογραφία

[1] A. Sangiovanni-Vincentelli and G. Martin. Platform-Based Design and Software Design Methodology for Embedded Systems, *IEEE Transactions on CAD of Integrated Circuits and Systems*, 2002, December, Vol. 19, No 12, pp. 1523-1533

[2] A. T. Chronopoulos and M. Benche and D. Grosu and R. Andonie, A Class of Loop Self-Scheduling for Heterogeneous Clusters, *CLUSTER '01: Proceedings of the 3rd IEEE International Conference on Cluster Computing*, 282, 2001, Washington, DC, USA.

[3] A. T. Chronopoulos, S. Penmatsa, J. Xu and S. Ali. Distributed Loop Scheduling Schemes for Heterogeneous Computer Systems, *Concurrency and Computation: Practice and Experience*, 18(7), 771–785, 2006.

[4] A.T. Chronopoulos, R. Andonie, M. Benche and D. Grosu. A Class of Distributed Self-Scheduling Schemes for Heterogeneous Clusters, *Proc. of the 3rd IEEE Int. Conf. on Cluster Computing (CLUSTER 2001)*, Newport Beach, CA USA, 2001.

[5] A.T. Chronopoulos, S. Penmatsa, N. Yu, Scalable Loop Self-Scheduling Schemes for Heterogeneous Clusters, *Proc. of the 4nd IEEE cond. on Cluster Computing (CLUSTER 2002)*, pp. 353-359, 2002

[6] AR Hurson, JT Lim, KK Kavi, B Lee. Parallelization of DOALL and DOACROSS LoopsAdvances in computers: A Survey. *In Advances in Computer*, volume 45, 1997.

[7] B. B. Mandelbrot. Fractal Geometry of Nature. W. H. Freeman & Co, August, 1988.

[8] B. Wilkinson and M. Allen. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. 2nd ed., Prentice Hall 2005.

[9] Boulet, P., Dongarra, J., Rastello, F., Robert, Y., & Vivien, F. Algorithmic Issues on Heterogeneous Computing Platforms. *Parallel Processing Letters*, 1998, v. 9(2), pp. 197–213.

[10] C. D. Polychronopoulos and D. J. Kuck, Guided self-scheduling: a practical scheduling scheme for parallel supercomputers, *In IEEE Transactions on Computers*, September 1992.

[11] C. F. Kennel, Convection and Substorms - Paradigms of magnetospheric phenomenology, *Oxford University Press, New York*, 1995.

[12] C. Kruskal, A. Weiss. Allocating Independent Subtasks on Parallel Processors. *IEEE Transactions on Software Engineering*, SE-11 (10), pp. 1001-1016, 1985

[13] C.D. Polychronopoulos and D.J. Kuck, Guided self-scheduling: A practical self-scheduling scheme for parallel supercomputers, *IEEE Trans. on Computer*, C-36(12), 1425•1439, 1987.

[14] A. Kejariwal, A. Nicolau, C. D. Polychronopoulos. History-aware Self-Scheduling, *Proc. of the 2006 Int'l Conference on Parallel Processing (ICPP 2006)*, pp. 185–192, Columbus, Ohio USA, 2006.

[15] C.-J. Hou and K.G. Shin. Load sharing with consideration of future task arrivals in heterogeneous distributed real-time systems. *IEEE Trans. Comput.*, 49(9):1076–1090, 1994.

[16] C.T. Yang and S.C. Chang. A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters, *Proc. of Intl Conf. on Computational Science*, Melbourne, Australia and St. Petersburg, Russia, 2003, pp. 1079–1088.

[17] C.T. Yang, K.W. Cheng, An Enhanced Parallel Loop Self-Scheduling Scheme for Cluster Environments, *The Journal of Supercomputing*, Vol, 34,pp. 315-335, 2005.

[18] D. L. Eager and E. D. Lazowska and J. Zahorjan, Adaptive load sharing in homogeneous distributed systems, *IEEE Trans. Softw. Eng.*, 12, 5, 662•675, 1986.

[19] D. N. Baker and T. I. Pulkkinen and V. Angelopoulos and W. Baumjohann and R. L. McPherron, Neutral line model of substorms: Past results and present view, *Journal of Geophysical Research*, 101, 12975–13010, 1996.

[20] D.C. Delcourt and J.-A. Sauvaud and A. Pedersen, Dynamics of single-particle orbits during substorm expansion phase, *Journal of Geophysical Research*, 95, 20853–20865, 1990.

[21] D.C. Delcourt, Particle acceleration by inductive electric fields in the inner magnetosphere, *Journal of Atmospheric and Solar-Terrestrial Physics*, 64, 551–559, 2002

[22] D.J. Hancock, J.M. Bull, R.W. Ford and T.L. Freeman. An Investigation of Feedback Guided Dynamic Scheduling of Nested Loops *Proc. of the IEEE International Workshops on Parallel Processing*, 21-24 Aug. 2000, ed. P. Sadayappan, pp. 315–321.

[23] D.J. Lilja, Exploiting the Parallelism Available in Loops, *IEEE Computer* , Vol. 27, No. 2, pp. 13-26, February 1994

[24] D.K. Lowenthal. Accurately Selecting Block Size at Run Time in Pipelined Parallel Programs, *Int. J. of Parallel Programming*, 28(3):245-274, 2000.

[25] D.P. Stern, The motion of a proton in the equatorial magnetosphere, *Journal of Geophysical Research*, 80, 595–599, 1975.

[26] E. de S. e Silvia and M. Gerla, Queueing network models for load balancing in distributed systems, *Journal of Parallel Distributed Computing*, 12, 1, 24•38, 1991.

[27] E. P. Markatos , T. J. LeBlanc Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessors, *IEEE Transactions on Parallel and Distributed Systems*, Volume 5 , Issue 4, pp 379 - 400, April 1994

[28] E. Silva and M. Gerla. Queueing network models for load balancing in distributed systems. *Journal of Parallel and Distributed Computing*, 12:24–38, 1991.

[29] E.P. Markatos and T.J. LeBlanc. Using Processor Affinity in Loop Scheduling on Shared-Memory Multiprocessors, *IEEE Transactions on Parallel and Distributed systems*, 5(4):379–400, April 1994.

[30] F. Desprez, P. Ramet and J. Roman. Optimal Grain Size Computation for Pipelined Algorithms, *Euro-Par*, 1:165–172, 1996.

[31] F. M. Ciorba, T. Andronikos, I. Riakiotakis, A. T. Chronopoulos and G. Papakonstantinou. Dynamic Multi Phase Scheduling for Heterogeneous Clusters, *Proc. of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006)*, Rhodes, Greece, 2006.

[32] F.M. Ciorba, T. Andronikos, I. Drositis and G. Papakonstantinou. Reducing Communication via Chain Pattern Scheduling. *Proc. of the 4th IEEE International Symposium on Network Computing and Applications (IEEE NCA05)*, Cambridge, MA USA, July 27-29 2005.

[33] G. Bolch, S. Greiner, H. de Meer and K.S. Trivedi. Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. *John Wiley & Sons, Inc.*, 1998.

[34] G. Goumas, N. Drosinos, M.Athanasaki, N. Koziris. Compiling Tiled Iteration Spaces for Clusters, *IEEE International Conference on Cluster Computing*, p 360, September 2002

[35] G. Papakonstantinou, I. Riakiotakis, T. Andronikos, F. M. Ciorba and A. T. Chronopoulos. Dynamic Scheduling for Dependence Loops on Heterogeneous Clusters. *Submitted to the Journal of Neural, Parallel & Scientific Computing*, Jan. 2006.

[36] H. Volland, Semi-empirical model of large-scale magnetospheric electric field, *Journal of Geophysical Research*, 78, 171–180, 1973.

[37] Hancock, Bull, Ford, Freeman, Feedback Guided Dynamic Scheduling of Nested Loops, *in Proceedings of PARA2000, Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 2000

[38] Hennessy, J. and Patterson D. Computer Architecture A Quantitative Approach Third Edition, Morgan Kaufmann, May, 2002.

[39] I. A. Daglis and Y. Kamide, The role of substorms in storm-time particle acceleration, in Disturbances in Geospace: The Storm-Substorm Relationship, *Geophysical Monograph Series, American Geophysical Union, Washington, DC*, 142, 119–129, 2003.

[40] I. Ahmad. A. Ghafoor and K. Mehrotra. Performance prediction of distributed load balancing on multicomputer systems, *In Proc. Supercomputing '91*, pp. 830–839, 1991.

[41] I. Banicescu and Z. Liu. Adaptive Factoring: A Dynamic Scheduling Method Tuned to the Rate of Weight Changes, *Proc. of the High Performance Computing Symposium 2000*, Washington, USA, 2000, pp. 122–129.

[42] I. Banicescu et al, 2003 Banicescu, I., Velusamy, V., & Devaprasad, J. (2003). On the Scalability of Dynamic Scheduling Scientific Applications with Adaptive Weighted Factoring, *Cluster Computing: The Journal of Networks, Software Tools and Applications -Kluwer Academic Publishers*, 6(3): 215–226.

[43] I. Banicescu, V. Velusamy and J. Devaprasad. On the Scalability of Dynamic Scheduling Scientific Applications with Adaptive Weighted Factoring, *Cluster Computing 6*, 2003, pp. 215–226.

[44] I. Panagopoulos, G. Manis and G. Papakonstantinou. Flexible General-Purpose Parallelizing Architecture for Nested Loops in Reconfigurable Platforms, *PATMOS 2007*, Sweeden, 2007.

[45] I. Panagopoulos. A. Dimopoulos, G.Manis, G. Papakonstantinou. AMPLE:Automatic mapping of algorithms for embedded systems, *PCI2007*, Patra, Greece, 2007.

[46] J. Barbosa, J. Tavares and A. J. Padilha. Linear Algebra Algorithms in a Heterogeneous Cluster of Personal Computers, *Proc. of the 9th Heterogeneous Computing Workshop (HCW 2000)*, Cancun, Mexico, 2000, pp. 147–159.

[47] J. Phillips, M. Areno, C. Rogers, A. Dasu, and B. Eames. A Reconfigurable Load Balancing Architecture for Molecular Dynamics, *In Proc. 14th Reconfigurable Architectures Workshop*, 2007.

[48] J. Ramanujam, P. Sadayappan. Tiling of Iteration Spaces for Multicomputers, *International Conference on Parallel Processing*, Vol. II, pp. 179-186, 1990

[49] J. Xue and W. Cai. Time-minimal Tiling when Rise is Larger than Zero, *Par. Comp.*, 28(6):915–939, 2002.

[50] J. Xue. Loop Tiling for Parallelism. *Kluwer Academic Publishers*, August 2000 (280 pages).

[51] J. Xue. On Tiling as a Loop Transformation, *Parallel Processing Letters*, vol.7, no.4, pp. 409–424, 1997.

[52] J.-L. Wang, L.-T. Lee and Y.-J. Hunag. Load balancing policies in heterogeneous distributed systems. *In Proc. of 26th Southeastern Symposium System Theory*, 1994.

[53] J.S. Bagla, TreePM: A Code for Cosmological N-Body Simulations, *J.Astrophysics*, Volume 23,pp.185-196, 2002.

[54] K. K. Yue, D. J. Lijla. Parallel Loop Scheduling for High-Performance Computers, *Tehcnical Report No. HPPC-94-12*, Dept. of Computer Science, Univ. of Minnessota, 1994.

[55] Karniadakis G. Em., & Kirby, R.M., Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and their Implementation, 2002, Cambridge University Press.

[56] Kwok, Y.K. & Ahmad, I. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, *ACM Computing Surveys*, 1999, v. 31(4), pp. 406•471.

[57] M. Adler, P. Berenbrink and K. Schroder. Analyzing an infinite parallel job allocation process. *In Proc. of 6th European Symposium on Algorithms* (ESA), 417–428, 1998.

[58] M. Bednara and J. Teich. Automatic Synthesis of FPGA Processor Arrays from Loop Algorithms, *The Journal of Supercomputing*, vol. 26, pp. 149-165, Kluwer Academic Publishers, 2003

[59] M. Cierniak, W. Li and M. J. Zaki. Loop Scheduling for Heterogeneity, *Proc. of the 4th IEEE Intl. Symp. on High Performance Distributed Computing*, Washington, DC, 1995, pp. 78–85.

[60] M. Gonzalez, E. Ayugade, X. Martorell and J. Labarta. Defining and Supporting Pipelined Executions in OpenMP. *Proc. of the Int'l Workshop on OpenMP Applications and Tools: OpenMP Shared Memory Parallel Programming (WOMPAT 2001)*, 2001, pp. 155–169.

[61] M. Gonzalez, E. Ayugade, X. Martorell and J. Labarta. Exploting Pipelined Executions in OpenMP. *Proc. of the Int'l Conference on Parallel Processing (ICPP 2003)*, 2003, pp. 153–160.

[62] M. Harchol-Balter and A. B. Downey, Exploiting process lifetime distributions for dynamic load balancing, *ACM Trans. Comput. Syst.*,15, 3, 253•285, 1997.

[63] M. Wolfe. High Performance Compilers for Parallel Computing, *Addison-Wesley Publication Co.*, 1996.

[64] M.-C. Fok and T. E. Moore and D. C. Delcourt, Modeling of inner plasma sheet and ring current during substorms, *Oxford University Press, New York*, 104, no. A7, 14557–14569, 1999.

[65] M.M. Strout, L. Carter, J. Ferrante and B. Kreaseck. Sparse Tiling for Stationary Iterative Methods, *Int'l J. of High Perf. Computing Applications*, 18(1):95–113, 2004.

[66] M.W. Liemohn and J.U Kozyra, Assessing the importance of convective and inductive electric fields in forming the stormtime ring current, *R.L. Winglee (ed.), Sixth International Conference on Substorms*, 103, 456-•462, Univ. Washington, Seattle, 2002.

[67] McMahon, F. H. The Livermore Fortran Kernels: A Computer Test Of The Numerical Performance Range, Lawrence Livermore National Laboratory, Livermore, California, UCRL-53745, 1986.

[68] N. A. Tsyganenko, A magnetospheric magnetic field model with a warped tail current sheet, *Planetary and Space Science*, 37, 5–20, 1989.

[69] N. Manjikian and T. S. Abdelrahman, Exploiting Wavefront Parallelism on Large-Scale Shared-Memory Multiprocessors, *IEEE Trans. Parallel Distrib. Syst.*, 12, 3, 259•271, 2001.

[70] N. Manjikian and T.S. Abdelrahman. Exploiting Wavefront Parallelism on Large-Scale Shared-Memory Multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 12(3):259–271, 2001.

[71] P. Boulet, J. Dongarra, F. Rastello, Y. Robert and F. Vivien. Algorithmic Issues on Heterogeneous Computing Platforms. *Parallel Processing Letters*, 9(2): 197–213, 1998.

[72] P. Tang, P.C. Yew. Processor Self-Scheduling for Multiple-Nested Parallel Loops. *Proceedings International Conference on Parallel Processing*, pp. 528-535, 1986

[73] Peter Pachecho. Parallel Programming with MPI, *Morgan Kauffman* 1997.

[74] Petkov, D. Harr, R. Amarasinghe, S. Efficient pipelining of nested loops: unroll-and-squash *Proc. of the 16th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2002)*, Ft. Lauderdale, FL, USA, 2002.

[75] Q. Lu, S.-M. Lau and K.-S. Leung. Dynamic load distribution using antitasks and load state vectors, *Concurrency: Practice and Experience*, 10(14):1251–1269, 1998.

[76] R. Andonov and S. Rajopadhye. Optimal Orthogonal Tiling of 2-D Iterations, *Int. J. of Parallel and Distrib.Computing*, 45(2):159–165, 1997.

[77] R. Ponnusamy, J. Saltz, A. Choudhary, Y.-S. Hwang, G. Fox. Runtime Support and Compilation Methods for User-Specified Irregular Data Distributions. *IEEE Trans. on Par. and Dist. Systems*, 6:8, pp. 815–831, 1995.

[78] R.E. Lopez and A.T.E. Lui and D.G. Sibeck and K. Takahashi, R.W. McEntire and L.J. Zanetti and S.M. Krimigis , On the relationship between the energetic particle flux morphology and the change in the magnetic field magnitude during substorms, *Journal of Geophysical Research*, 94, 17105–17119, 1989.

[79] R.W. Floyd and L. Steinberg. An adaptive algorithm for spatial grey scale. *Proc. Soc. Inf. Display*, 17:75-•77, 1976.

[80] Rastello, F., Rao, A., & Pande, S. Optimal Task Scheduling to minimize Inter-Tile Latencies. *Parallel Computing*, v. 29(2), pp. 209–239.

[81] S. Chen and J. Xue. Partitioning and scheduling loops on NOWs, *Comp. Comm. J.*, 22:1017–1033, 1999.

[82] S. F. Hummel and J. Schmidt and R. N. Uma and J. Wein, Load-sharing in heterogeneous systems via weighted factoring, *SPAA '96: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, 318–328,Padua, Italy, 1996.

[83] S. F. Hummel, E. Schonberg, L.E. Flynn. Factoring: A Method for Scheduling Parallel Loops, *Communications of the ACM*, 35(8):90–101, 1992.

[84] S.Chen, J.Xue, Partitioning and scheduling of loops on NOWs, *Computer Communications*, vol. 22, pp. 1017-1033, 1999

[85] Scott Whitman , Dynamic Load Balancing for Parallel Polygon Rendering, *IEEE Computer Graphics and Applications archive*, Volume 14 , Issue 4, pp. 41-48, 1994

[86] T. Andronikos, F.M. Ciorba, P. Theodoropoulos, D. Kamenopoulos and G. Papakonstantinou. Code Generation for General Loops Using Methods from Computational Geometry. *Proc. of the IASTED Parallel and Distributed Computing and Systems Conference (PCDS 2004)*, Cambrige, MA USA, November 9-11, 2004, pp. 348–353.

[87] T. Kunz. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. *IEEE Trans. on Soft. Eng.* pp. 725–730, 1991.

[88] T.-C. Huang, P.-H. Hsu, T.-N. Sheng. Efficient Run-time Scheduling for Parallelizing Partially Parallel Loop. *Proc. of the 3rd Int'l Conf. on Algorithms and Architectures for Parallel Processing.* pp. 397–403, 1997.

[89] T.H. Kim and J.M. Purtilo. Load Balancing for Parallel Loops in Workstation Clusters, *Proc. of Intl. Conference on Parallel Processing*, Bloomingdale, IL USA, 3:182–190, 1996.

[90] T.H. Tzen and L.M. Ni. Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers, *IEEE Trans. on Parallel and Distributed Systems*, 4(1):87–98, Jan. 1993.

[91] Teebu Philip, Increasing chunk size loop scheduling algorithms for data independent loops, Master Thesis in Computer Engineering, Pennsylvania State University. 1995

[92] Thanalapati T., & Dandamudi, S. An Efficient Adaptive Scheduling Scheme for Distributed Memory Multicomputers. *IEEE Transactions on Parallel and Distributed Systems*, 2001, v. 12(7), pp. 758–768.

[93] U. Bondhugula, J. Ramanujam, P. Sadayappan, Automatic mapping of nested loops to FPGAs, *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'07)*, San Jose California, USA, 2007.

[94] X. Wang, S. Ziavras. A configurable multiprocessor and dynamic load balancing for parallel LU factorization, *In Proc.18th International Parallel and Distributed Processing Symposium*, 2004.

[95] Y. K. Kwok and I. Ahmad, Benchmarking and Comparison of the Task Graph Scheduling Algorithms, *Journal of Parallel and Distributed Computing*, 59, 3, 381–422, 1999.

[96] Y. K. Kwok and I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, *ACM Comput. Surv.*, 31, 4, 406–471, 1999.

[97] Y. Kamide and et al., Current understanding of magnetic storms: Storm/ substorm relationships, *Journal of Geophysical Research*, 103, 17705–17728,1998.

[98] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood. Hardware-software co-design of embedded reconfigurable architectures, *In Proc. Design Automation Conf.*, 2000, pp. 507–512.

[99] Y. Yan, C. Jin and X. Zhang. Adaptively Scheduling Parallel Loops in Distributed Shared-Memory Systems, *IEEE Trans. on Parallel and Distributed Systems*, 8(1):70–81, Jan. 1997.

[100] Y.W. Fann, C.T. Yang, S.S. Tseng and C.J. Tsai. An Intelligent Parallel Loop Scheduling for Parallelizing Compilers, *Journal of Information Science and Engineering*, 16:69–200, 2000.

[101] C-T Yang, K-W Cheng and K-C Li. An Efficient Load Balancing Scheme for Grid-based High Performance Scientific Computing, *Proc. of the 19th Int. Conf. on Advanced Information Networking and Applications (AINA'05)*, Tamkang University, Taiwan, 2005.

[102] F. M. Ciorba, I. Riakiotakis, T. Andronikos, G. Papakonstantinou, and A. T. Chronopoulos. Enhancing self-scheduling algorithms via synchronization and weighting. *Journal of Parallel Distributed Computing*, 68(2): 246–264, 2008.

[103] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two sequences. *Journal of Molecular Biology*, 48:443–453, 1970.

[104] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195-197, 1981.

[105] F. M. Ciorba, I. Riakiotakis, T. Andronikos, A. T. Chronopoulos and G. Papakonstantinou. Optimal Synchronization Frequency for Dynamic Pipelined Computations on Heterogeneous Systems. *in Procs. of IEEE Int. Conf. on Cluster Computing (CLUSTER 2007)*, Austin, TX USA, 17-20 September, 2007.

[106] Mathcad. http://www.mathcad.com

[107] I. Riakiotakis, G. Goumas, N. Koziris, F.A. Metallinou, I. Daglis. Evaluation of Dynamic Scheduling Methods in Simulations of Storm-time Ion Acceleration, *In Proc. of the 22th IEEE International Parallel and Distributed Processing Symposium (IPDPS '08)*, Miami, Florida, USA, April 18, 2008.

Κατάλογος Σχημάτων

Κατάλογος Πινάκων