ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πρόβλεψη επίδοσης εφαρμογών σε παράλληλες αρχιτεκτονικές μοιραζόμενης μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του
Ευστράτιου Μ. Καραπαναγιώτη

Επιβλέπων: Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2023

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πρόβλεψη επίδοσης εφαρμογών σε παράλληλες αρχιτεκτονικές μοιραζόμενης μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ευστράτιου Μ. Καραπαναγιώτη

Επιβλέπων: Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ..... .

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος
Αναπληρωτής Καθηγητής, Ε.Μ.Π.	Καθηγητής, Ε.Μ.Π.	Καθηγητής, Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2023

(Υπογραφή)

.....
Ευστράτιος Μ. Καραπαναγιώτης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Η διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων (Cslab) της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών. Θα ήθελα να ευχαριστήσω τα άτομα του εργαστηρίου που μου προσέφεραν τις κατάλληλες κατευθύνσεις στα εμπόδια της παρούσας εργασίας.

Στον επιβλέποντα καθηγητή μου, Γεώργιο Γκούμα, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες για τη δυνατότητα που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα, για τη συνεχόμενη παρουσία του και την καθοδήγησή του σε όλα τα στάδια της διπλωματικής εργασίας μου.

Τέλος, θα ήθελα να ευχαριστήσω πολύ την οικογένεια μου που συνέχισε να αποτελεί μια σημαντική βοήθεια και σε αυτό το στάδιο της ζωής μου.

Περίληψη

Τα παράλληλα συστήματα είναι πανταχού παρόντα στη ζωή μας, από τα κινητά τηλέφωνά μας, τους προσωπικούς υπολογιστές μας μέχρι και τα υπολογιστικά νέφη πανεπιστημίων, επιστημονικών κέντρων, κρατών και πολυεθνικών εταιρειών. Το κάθε ένα από αυτά σχεδιάζεται με διαφορετικά χαρακτηριστικά, ανάλογα με τις ανάγκες του πεδίου εφαρμογής που τα χρησιμοποιεί. Υπάρχουν πολλές παράλληλες αρχιτεκτονικές και μοντέλα παράλληλου προγραμματισμού που υλοποιούν διαφορετικά παράλληλα συστήματα. Επίσης, από ένα ακολουθιακό πρόγραμμα προκύπτουν πολλά πιθανά παράλληλα προγράμματα, λόγω των διαφορετικών προγραμματιστικών διεπαφών και βιβλιοθηκών που υπάρχουν. Η ερώτηση που προσπαθεί να απαντηθεί στη διπλωματική εργασία είναι, αν έχοντας γνώση της σειριακής εκδοχής ενός κώδικα και μερικών χαρακτηριστικών των μηχανημάτων στα οποία θα εκτελεστεί, είναι δυνατόν να γίνει γνωστή η επίδοση της παράλληλης εκδοχής του πριν την υλοποίησή της. Το όραμα μας είναι να δοθεί απάντηση για όλα τα συστήματα παράλληλης επεξεργασίας. Στην εργασία αυτή όμως, στόχος είναι να κατασκευαστεί ένα μοντέλο πρόβλεψης επίδοσης για τις παράλληλες αρχιτεκτονικές μοιραζόμενης μνήμης και μοντέλα προγραμματισμού κοινού χώρου διευθύνσεων. Για την επίτευξή του στόχου, αναλύονται οι νόμοι της κλιμακωσιμότητας - νόμος του Amdahl, Gustafson και Ισοαποδοτικότητας - για την καλή θεωρητική και μαθηματική θεμελίωση του μοντέλου μας. Παρουσιάζονται τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν για τη διεξαγωγή πειραμάτων και την καταγραφή των απαραίτητων μετρήσεων για τα μηχανήματα και το μοντέλο μας. Γίνεται εκτενής περιγραφή της κατασκευής του μοντέλου μας και της επίδοσης των προβλέψεων του για ρεαλιστικά προγράμματα. Στο τέλος εξάγονται συμπεράσματα, αναφέρονται τα αδύναμα σημεία του και προτείνονται δυνατές κατευθύνσεις για μελλοντική έρευνα.

Λέξεις-κλειδιά: κλιμακωσιμότητα, παράλληλα συστήματα επεξεργασίας, πολυεπεξεργαστές, πολυπύρηνα συστήματα, αρχιτεκτονικές μοιραζόμενης μνήμης, μοντέλο πρόβλεψης επίδοσης, μοντέλο Roofline, νόμος Amdahl, νόμος Gustafson, ισοαποδοτικότητα, πυρήνας Jacobi, OpenMP, μετροπρογράμματα

Abstract

More than ever, parallel processing systems are intertwined with our daily necessities. They are the backbones of our mobile devices and personal computers and the foundations of the cloud services that are provided by universities, research centers, nations, and multinational companies. Every single one of them is built with different specifications that better accommodate the needs of the specific field that uses them. As such, there are various parallel architectures and models for parallel programming. For the engineer, having this much variety makes it difficult to choose the best option. The problem gets harder when someone takes into account the fact that there are different parallel programs that stem from a single serial program, because of the APIs and libraries that exist. The question asked in this diploma thesis is, if there is a way to deduce the performance of a parallel program prior to its implementation if the only known facts are the execution time of the serial version of the program and some necessary specifications of the computational systems on which it will be executed. Our long term aim is to provide a solution for all the attainable parallel processing systems. However, in this particular thesis, the main focus is to create a forecasting model for parallel systems of shared memory architecture and shared address space programming models. In light of that, at first, the scalability laws that govern the functionality of the parallel systems - Amdahl's law, Gustafson's law, Isoefficiency - are analyzed. They are used as theoretical and mathematical foundations of our model. The necessary tools for experimenting and obtaining information about the systems and our model, are also presented. The whole process to which our model came to be and how it performs in industry-standard benchmarks is described extensively in the last chapters. In the end, conclusions about the predictions' accuracy are drawn and limitations of the model as well as the future research are also discussed.

Keywords: scalability, parallel processing systems, multiprocessors, multicore systems, shared memory architectures, performance prediction model, Roofline model, Amdahl's law, Gustafson's law, isoefficiency, Jacobi kernel, OpenMP, benchmarks

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Στη σημερινή εποχή, η ανάγκη μας για επίλυση προβλημάτων με μεγάλες υπολογιστικές απαιτήσεις έστρεψε το ενδιαφέρον των μηχανικών στα παράλληλα συστήματα επεξεργασίας. Καταλυτικός παράγοντας ήταν η αδυναμία να αυξηθεί η επίδοσή των μονοεπεξεργαστών χωρίς να αυξηθεί το κόστος κατασκευής τους και η κατανάλωση ισχύος για τη λειτουργία τους. Σύμφωνα με το νόμο του Moore κάθε χρόνο, η τεχνολογία επιτρέπει να διπλασιάζεται ο αριθμός των transistors σε ένα ολοκληρωμένο κύκλωμα ενώ μειώνεται παράλληλα το μέγεθός τους. Προκειμένου όμως η συνολική κατανάλωση ισχύος του ολοκληρωμένου κυκλώματος να μην είναι μεγαλύτερη από αυτή του προηγούμενου του πριν την κλιμάκωσή του, η τάση ενεργοποίησης των transistors του μειώνεται και αυτή. Βρισκόμαστε όμως πλέον στο στάδιο όπου η τάση με την οποία τροφοδοτούνται δεν μπορεί να μειωθεί περισσότερο. Ως αποτέλεσμα ο μοναδικός τρόπος να μειωθεί η κατανάλωση είναι να γίνει μικρότερη η συχνότητα μεταγωγής αυτών. Αυτό το συμπέρασμα προκύπτει εύκολα μελετώντας την εξίσωση φυσικής για την ισχύ ενός transistor,

$$P = C \times V^2 \times f$$

Αυτή η απόφαση θα δημιουργήσει όμως μικροεπεξεργαστές μικρής ταχύτητας παρόλο που η αρχιτεκτονική τους θα μπορεί να αντιμετωπίσει πιο πολύπλοκα προβλήματα. Από την άλλη προσπαθώντας να αυξήσουμε τη συχνότητα λειτουργίας των transistors διατηρώντας σταθερό το πλήθος των μικροεπεξεργαστών σημαίνει και αύξηση της καταναλισκόμενης ισχύος. Επομένως, ο μοναδικός δρόμος που απέμεινε για την αύξηση της επίδοσης ενός υπολογιστικού συστήματος είναι η χρήση πολλών μικροεπεξεργαστών στο ίδιο σύστημα.

Ένα πολυπύρηνο σύστημα καταναλώνει λιγότερη ισχύ για να πετύχει την ίδια επίδοση με ένα σύστημα με έναν μικροεπεξεργαστή στο οποίο έχει αυξηθεί η συχνότητα. Έτσι, μπορούν να δημιουργηθούν πολύ ισχυρές υπολογιστικές συσκευές για την προσωπική χρήση ενός μέσου καταναλωτή αλλά και μεγάλα συστήματα μαζικής επεξεργασίας δεδομένων για κράτη και εταιρείες με πολύ μικρή κατανάλωση. Η αλλαγή φιλοσοφίας στη σχεδίαση των υπολογιστικών συστημάτων έφερε και νέα ερωτήματα. Όπως το πώς πρέπει να σχεδιαστεί ένας παράλληλος επεξεργαστής, ποια παράλληλη αρχιτεκτονική είναι προτιμότερη και ποιό προγραμματιστικό μοντέλο παράλληλου προγραμματισμού είναι το καλύτερο.

Έχοντας στη διάθεσή μας εργαλεία για τη μελέτη αυτών των παράλληλων συστημάτων, σκοπός αυτής της διπλωματικής εργασίας είναι η δημιουργία ενός μοντέλου πρόβλεψης της επίδοσης ενός παράλληλου προγράμματος πριν αυτό παραλληλοποιηθεί, για αρχιτεκτονικές μοιραζόμενης μνήμης και προγραμματιστικά μοντέλα κοινού χώρου διευθύνσεων. Συγκεκριμένα, θα θεμελιώσουμε ένα μαθηματικό μοντέλο μέσω των νόμων που διέπουν τα παράλληλα συστήματα. Θα δείξουμε ποιά χαρακτηριστικά από μια αρχιτεκτονική και μια εφαρμογή χρειαζόμαστε και πώς τα μετράμε. Θα ελέγξουμε την εγκυρότητα των προβλέψεων του μοντέλου μας χρησιμοποιώντας μετροπρογράμματα της βιομηχανίας. Τέλος, θα συζητηθούν το ποιές είναι οι αδυναμίες του και πώς μπορεί να επεκταθεί η συζήτηση αυτή σε μελλοντικές έρευνες.

Η κατασκευή ενός τέτοιο μοντέλου θα αποτελέσει σημαντική βοήθεια για έναν μηχανικό στην εξοικονόμηση χρόνου και χρημάτων κατα την εργασία του, διότι θα γνωρίζει εξαρχής αν αξίζει η υλοποίηση του παράλληλου προγράμματος πάνω σε συγκεκριμένη αρχιτεκτονική.

Κεφάλαιο 2

Παράλληλα Συστήματα Επεξεργασίας

Προκειμένου να δημιουργηθεί ένα μοντέλο πρόβλεψης για ένα παράλληλο σύστημα συγκεκριμένης αρχιτεκτονικής και προγραμματιστικού μοντέλου, χρειάζεται να οριστεί το τι είναι ένα παράλληλο σύστημα.

Ορισμός 1 *Παράλληλο σύστημα είναι ο συνδυασμός μιας παράλληλης αρχιτεκτονικής και ενός παράλληλου προγράμματος.*

Ο λόγος για τον οποίο το μοντέλο μας συμπεριλαμβάνει και την αρχιτεκτονική ενός υπολογιστή στις προβλέψεις του είναι επειδή η συμπεριφορά ενός παράλληλου προγράμματος διαφέρει από αρχιτεκτονική σε αρχιτεκτονική. Για ένα ακολουθιακό πρόγραμμα, το καλύτερο πρόγραμμα είναι αυτό που πετυχαίνει τον βέλτιστο χρόνο εκτέλεσης. Αν η αρχιτεκτονική στην οποία εκτελείται αλλάξει πάλι ο μικρότερος χρόνος εκτέλεσης θα προκύπτει από τη βέλτιστη εκδοχή του ακολουθιακού προγράμματος μιας και οι αρχιτεκτονικές των ακολουθιακών υπολογιστών ακολουθούν το ίδιο μοντέλο υπολογιστή (Von Neumann). Αντιθέτως, υπάρχουν παράλληλες αρχιτεκτονικές που κατασκευάζονται από διαφορετικά υπολογιστικά μοντέλα. Επίσης, υπάρχουν παράλληλες αρχιτεκτονικές με ίδιο υπολογιστικό μοντέλο αλλά διαφορετικό πλήθος επεξεργαστικών μονάδων. Έτσι, ένα παράλληλο πρόγραμμα δε γίνεται να αποκαλείται βέλτιστο αν δε γίνεται αναφορά και στην παράλληλη αρχιτεκτονική της πλατφόρμας στην οποία εκτελείται.

2.1 Παράλληλες Αρχιτεκτονικές

Για να εξηγήσουμε τα συστήματα για τα οποία το μοντέλο μας θα εξάγει προβλέψεις για την επίδοση των προγραμμάτων που θα εκτελούνται σε αυτά, θα γίνει αρχικά μια ανάλυση των διαφορετικών παράλληλων αρχιτεκτονικών που υπάρχουν.

Από το μοντέλο αρχιτεκτονικής Von Neumann, ένας ακολουθιακός υπολογιστής αποτελείται από μία κεντρική μονάδα επεξεργασίας και μία μνήμη στην οποία είναι αποθηκευμένα οι εντολές του προγράμματος προς εκτέλεση και τα δεδομένα του. Υπάρχει δίαυλος επικοινωνίας μεταξύ της κεντρικής μονάδας επεξεργασίας και της μνήμης για τη μεταφορά των εντολών και δεδομένων.

Εστιάζοντας περισσότερο στη λειτουργία αυτού του μοντέλου, διαπιστώνει κανείς πως υπάρχει μια ροή εντολών και μια ροή δεδομένων μεταξύ επεξεργαστή και μνήμης. Χρησιμοποιώντας ως αφετηρία το μοντέλο αρχιτεκτονικής Von Neumann και γενικεύοντας το σε όσο πλήθος κεντρικών μονάδων επεξεργασίας και μνημών, δημιουργείται μια ιδέα για μοντέλο παράλληλης αρχιτεκτονικής[1][2][3]. Σύμφωνα με το μοντέλο αυτό θα υπάρχουν πολλές ροές εντολών αλλά και πολλές ροές δεδομένων, ανάλογα με την ανάγκη της κάθε εφαρμογής. Αυτή η ελευθερία επιλογών δεν οδηγεί σε καμία περιγραφή των ικανοτήτων των παράλληλων αρχιτεκτονικών προκειμένου για κάθε περίσταση να επιλεχθεί η καλύτερη αρχιτεκτονική.

Σύμφωνα με την ταξινόμηση κατά Flynn όλα τα μοντέλα αρχιτεκτονικών για τους υπολογιστές είτε ακολουθιακά είτε παράλληλα μπορούν να κατηγοριοποιηθούν στις επόμενες τέσσερις ομάδες, ανάλογα με το πλήθος των ροών εντολών και δεδομένων[4].

1. **Μοναδική Ροή Εντολών, Μοναδική Ροή Δεδομένων (SISD)**, όπου στις αρχιτεκτονικές αυτές οι εντολές εκτελούνται ακολουθιακά μία προς μία από μια μονάδα επεξεργασίας και έχοντας πρόσβαση στα δεδομένα βήμα προς βήμα κατά την εκτέλεση. Στην κατηγορία αυτή ανήκουν οι μονοί επεξεργαστές.

2. **Μοναδική Ροή Εντολών, Πολλαπλές Ροές Δεδομένων (SIMD)**, όπου στις αρχιτεκτονικές αυτές ίδιες εντολές εκτελούνται ακολουθιακά μία προς μία από πολλές μονάδες επεξεργασίας και εφαρμόζονται σε διαφορετικές ροές δεδομένων συγχρονισμένα. Το μοντέλο αυτό αξιοποιεί τον παραλληλισμό σε επίπεδο δεδομένων. Στην κατηγορία αυτή ανήκουν οι διανυσματικοί επεξεργαστές, όπου υπάρχει μία κεντρική μονάδα ελέγχου που διαβιβάζει στις μονάδες επεξεργασίας την ίδια εντολή προς εκτέλεση στα δεδομένα τους. Οι μονάδες επεξεργασίας διαθέτουν ξεχωριστή μνήμη που ισοδυναμεί με ξεχωριστή ροή δεδομένων.

3. **Πολλαπλές Ροές Εντολών, Μοναδική Ροή Δεδομένων (MISD)**, όπου στις αρχιτεκτονικές αυτές εκτελούνται πολλές εντολές σε ίδια δεδομένα. Στην κατηγορία αυτή δεν υπάρχουν εμπορικοί επεξεργαστές.

4. **Πολλαπλές Ροές Εντολών, Πολλαπλές Ροές Δεδομένων (MIMD)**, όπου στις αρχιτεκτονικές αυτές κάθε μονάδα επεξεργασίας είναι ανεξάρτητη να εκτελέσει τις δικές της εντολές με τα δικά της δεδομένα. Το μοντέλο αυτό αξιοποιεί τον παραλληλισμό σε επίπεδο νημάτων. Στην κατηγορία αυτή ανήκουν οι πολυεπεξεργαστές γενικού σκοπού.

Το πιο διαδεδομένο μοντέλο είναι το MIMD, διότι παρέχει τη μεγαλύτερη προγραμματιστική ευελιξία. Οι εργασίες ενός προγράμματος μπορούν να ανατεθούν σε διαφορετικές μονάδες επεξεργασίας για την παράλληλη εκτέλεσή τους. Το μοντέλο αυτό, μπορεί επίσης να προσομοιώσει τα υπόλοιπα τρία μοντέλα.

Αν και η ταξινόμηση κατά Flynn δίνει μια καλή αρχική εικόνα για τις παράλληλες αρχιτεκτονικές, δεν παύει να είναι μια αρκετά αφαιρετική προσέγγιση στο θέμα αυτό, με αποτέλεσμα να μην μπορεί να κατηγοριοποιήσει όλες τις ειδικές περιπτώσεις. Ταυτόχρονα, καθώς η τεχνολογία εξελίσσεται αλλάζουν και οι τεχνικές παραλληλισμού στις αρχιτεκτονικές των υπολογιστικών

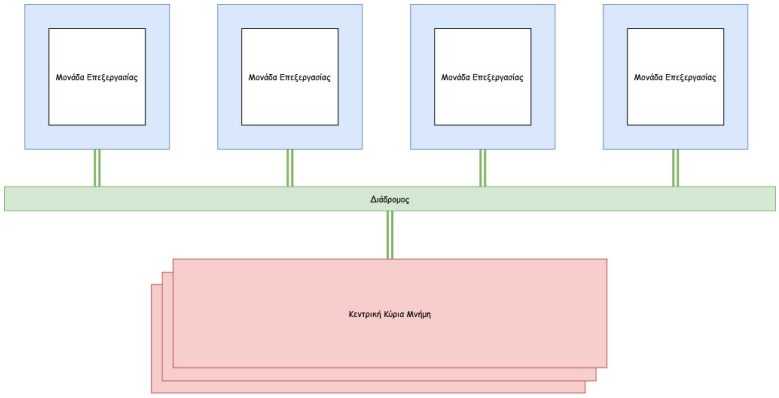
συστημάτων[5]. Έτσι, και βιβλιογραφικά, οι παράλληλες αρχιτεκτονικές είναι προτιμότερο να αναλύονται ως προς το σύστημα μνήμης τους και πως συνδέονται οι μονάδες επεξεργασίας σε αυτό.

Η διασύνδεση των μονάδων επεξεργασίας μεταξύ τους και με τη μνήμη μπορεί να γίνει με τρεις διαφορετικούς τρόπους. Ο πρώτος τρόπος είναι οι επεξεργαστές να μοιράζονται μια κοινή φυσική μνήμη στην οποία να συνδέονται με μεταγωγείς ή με διάδρομο. Ο δεύτερος τρόπος είναι ο κάθε επεξεργαστής να έχει τη δικιά του μνήμη αλλά όλες οι φυσικές μνήμες να αντιμετωπίζονται από τα προγράμματα ως μια ενιαία φυσική μνήμη. Δηλαδή οι λογικές διευθύνσεις των προγραμμάτων που εκτελούνται σε κάθε επεξεργαστή να ταυτίζονται με τις φυσικές της εικονικής ενιαίας φυσικής μνήμης. Οι μονάδες επεξεργασίας θα συνδέονται μεταξύ τους μέσω ενός διαδρόμου. Ο τρίτος τρόπος είναι ο κάθε επεξεργαστής να έχει τη δική του μνήμη, για την οποία όμως οι διευθύνσεις της να είναι ιδιωτικές και προσβάσιμες μόνο από τον επεξεργαστή με τον οποίο συνδέεται. Η επικοινωνία μεταξύ των επεξεργαστών σε αυτήν την αρχιτεκτονική θα γίνεται μέσω ενός δικτύου διασύνδεσης.

Οι τρεις προηγούμενοι τρόποι που περιέγραψαν τον τρόπο με τον οποίο το σύστημα μνήμης διασυνδέεται με τους επεξεργαστές, αντιστοιχούν σε ονοματισμένες αρχιτεκτονικές στη βιβλιογραφία. Ο πρώτος τρόπος περιγράφει τη Συγκεντρωτική Αρχιτεκτονική Μοιραζόμενης Μνήμης. Ο δεύτερος τρόπος διασύνδεσης περιγράφει την Κατανεμημένη Αρχιτεκτονική Μοιραζόμενης Μνήμης, ενώ ο τρίτος τρόπος απλά την Κατανεμημένη Αρχιτεκτονική.

2.1.1 Συγκεντρωτική Αρχιτεκτονική Μοιραζόμενης Μνήμης

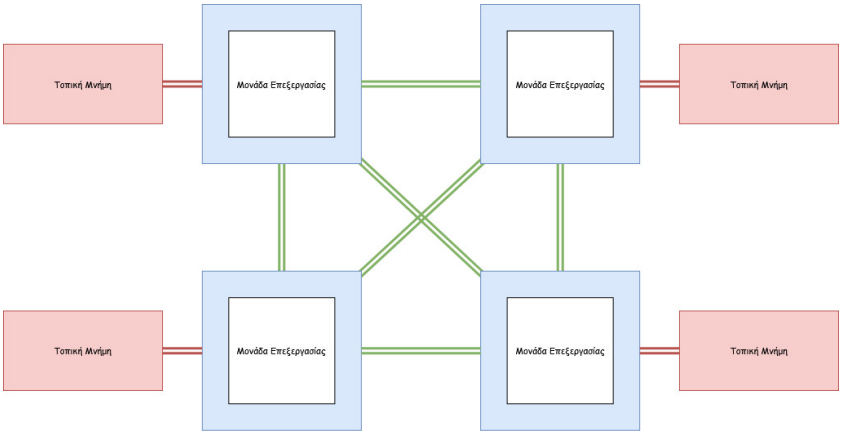
Η αρχιτεκτονική αυτής της μορφής προτιμάται όταν το σύστημα προς κατασκευή δεν έχει μεγάλο πλήθος μονάδων επεξεργασίας. Ο λόγος είναι ότι όταν το πλήθος των επεξεργαστών αυξηθεί αρκετά το εύρος ζώνης της κεντρικής μνήμης θα περιορίσει την ταχύτητα απόκρισης της σε αιτήματα αυτών. Έτσι, πολλοί επεξεργαστές θα παραμένουν ανενεργοί εως ότου το αίτημα τους εκπληρωθεί και μπορέσουν να συνεχίσουν τους υπολογισμούς τους. Οι επεξεργαστές συνδέονται στην κύρια μνήμη είτε σημείο προς σημείο είτε μέσω ενός μεταγωγέα. Επειδή η σύνδεση είναι άμεση και υπάρχει μία μόνο κεντρική μνήμη, ο χρόνος προσπέλασης των δεδομένων είναι ομοιόμορφος. Εξαιτίας αυτής της ιδιότητας, η αρχιτεκτονική είναι γνωστή βιβλιογραφικά και ως αρχιτεκτονική ομοιόμορφης προσπέλασης μνήμης (uniform memory access (UMA)). Οι πολυεπεξεργαστές που σχεδιάζονται με βάση αυτή την αρχιτεκτονική, λόγω της συμμετρικής σχέσης μεταξύ των επεξεργαστών και της κεντρικής μνήμης, ονομάζονται συμμετρικοί πολυπεξεργαστές μοιραζόμενης μνήμης (symmetric shared-memory multiprocessors)[6].



Σχήμα 2.1: Διάγραμμα μπλοκ για την συγκεντρωτική αρχιτεκτονική μοιραζόμενης μνήμης.

2.1.2 Κατανεμημένη Αρχιτεκτονική Μοιραζόμενης Μνήμης

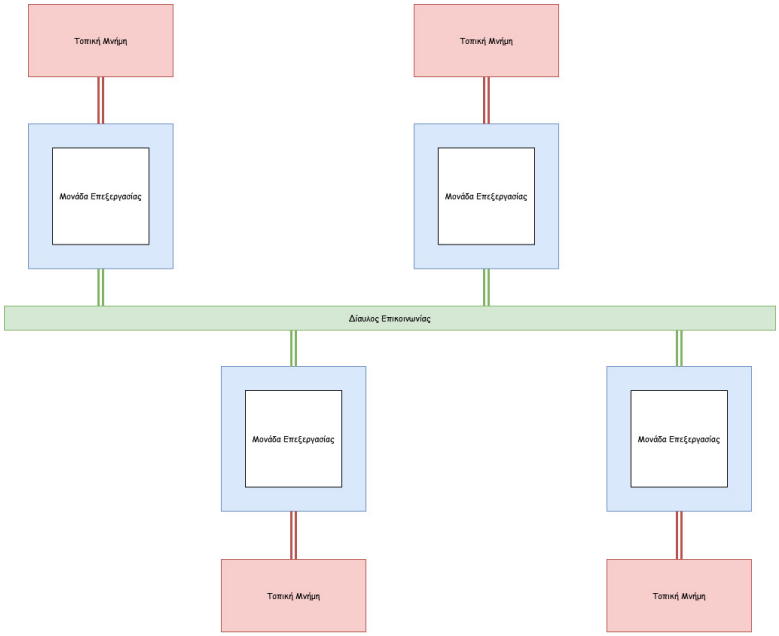
Στην περίπτωση αυτή η κάθε μονάδα επεξεργασίας έχει τη δική της ιεραρχία μνήμης που αποτελείται από τις κρυφές μνήμες και την κύρια μνήμη. Όλες οι κατανεμημένες φυσικές μνήμες όμως λειτουργούν ενιαία και παρέχουν κοινό χώρο διευθύνσεων για κάθε πρόγραμμα που εκτελείται. Παρουσιάζεται έτσι η εικόνα μιας μεγάλης κεντρικής φυσικής μνήμης και οι λογικές διευθύνσεις ταυτίζονται με τις φυσικές διευθύνσεις. Η κατανεμημένη αρχιτεκτονική μοιραζόμενης μνήμης κλιμακώνει πολύ καλά για μεγάλο αριθμό μονάδων επεξεργασίας σε σύγκριση με τη συγκεντρωτική αρχιτεκτονική μοιραζόμενης μνήμης και λόγω αυτού είναι πιο διαδεδομένη στα σύγχρονα συστήματα. Επειδή, στην αρχιτεκτονική αυτή δεν υπάρχει κεντρική μνήμη ώστε οι επεξεργαστές να συνδέονται άμεσα πάνω σε αυτή (σημείο προς σημείο), ο χρόνος προσπέλασης δεδομένων για κάθε επεξεργαστή διαφέρει. Αυτό συμβαίνει επειδή ο χρόνος πρόσβασης τοπικών δεδομένων σε σύγκριση με το χρόνο πρόσβασης για απομακρυσμένα δεδομένα δεν είναι ίδιος και εξαρτάται από το δίαυλο που διασυνδέει τις μονάδες επεξεργασίας. Για παράδειγμα, αν κάποια δεδομένα τα οποία χρειάζεται ένας επεξεργαστής είναι αποθηκευμένα σε διευθύνσεις μνήμης ενός άλλου επεξεργαστή τότε ο χρόνος πρόσβασης σε αυτά εξαρτάται από την απόσταση των δύο επεξεργαστών στο δίαυλο. Αν είναι κοντά τότε ο χρόνος θα είναι ίδιος με το χρόνο προσπέλασης αν τα δεδομένα ήταν αποθηκευμένα τοπικά. Αν όμως η απόσταση τους είναι μεγάλη τότε ο χρόνος προσπέλασης των δεδομένων αυτών θα είναι μεγαλύτερος από μια τοπική προσπέλαση. Αυτός ο μη ομοιόμορφος τρόπος προσπέλασης των δεδομένων της συνολικής μνήμης οδήγησε στο να ονομαστεί η αρχιτεκτονική αυτή και ως αρχιτεκτονική μη ομοιόμορφης προσπέλασης μνήμης (non uniform memory access (NUMA)) [7][8].



Σχήμα 2.2: Διάγραμμα μπλοκ για την κατανεμημένη αρχιτεκτονική μοιραζόμενης μνήμης.

2.1.3 Αρχιτεκτονική Κατανεμημένης Μνήμης

Στην αρχιτεκτονική κατανεμημένης μνήμης, οι τοπικές κύριες μνήμες των επεξεργαστών δε «συνεργάζονται» προκειμένου να δημιουργήσουν έναν κοινό χώρος διευθύνσεων. Όπως και προηγουμένως, η κάθε μονάδα επεξεργασίας έχει μια τοπική ιεραρχία μνήμης, με κρυφές μνήμες και μια τοπική κύρια μνήμη. Οι φυσικές διευθύνσεις της κύριας μνήμης είναι ιδιωτικές και δεν ταυτίζεται με καμία φυσική διεύθυνση των υπόλοιπων μνημών. Οι επεξεργαστές συνδέονται μεταξύ τους μέσω ενός διαύλου επικοινωνίας μεγάλου εύρους ζώνης και ταχύτητας, με αποτέλεσμα να μπορούν να κατασκευαστούν συστήματα αρκετά μεγάλης κλίμακας. Επειδή, δεν υπάρχει κοινός χώρος διευθύνσεων στον οποίο να έχουν πρόσβαση οι επεξεργαστές, η μεταφορά δεδομένων μεταξύ τους μπορεί να γίνει μόνο μέσω ανταλλαγής μηνυμάτων στο δίαυλο επικοινωνίας. Επομένως, η αρχιτεκτονική αυτή δίνει πολύ μεγάλη σημασία στην επικοινωνία μεταξύ των στοιχείων του συστήματος καθώς μπορεί να μετατραπεί στο πιο χρονοβόρο αίτιο. Παράδειγμα τέτοιων υπολογιστικών συστημάτων αποτελούν οι συστάδες (clusters).



Σχήμα 2.3: Διάγραμμα μπλοκ για την αρχιτεκτονική κατανεμημένης μνήμης.

2.2 Προγραμματιστικά Μοντέλα

Η σχεδίαση ενός παράλληλου προγράμματος είναι περίπλοκη υπόθεση διότι υπάρχουν διαφορετικές παράλληλες αρχιτεκτονικές. Δεν μπορεί να δημιουργηθεί ένα παράλληλο πρόγραμμα το οποίο να εκτελείται σε όλες τις παράλληλες αρχιτεκτονικές. Το βασικό πρόβλημα είναι πως υπάρχουν αρχιτεκτονικές που παρουσιάζουν μια ενιαία μια ενιαία μνήμη και άλλες που την παρουσιάζουν κατανεμημένη. Ως αποτέλεσμα, έχουν δημιουργηθεί προγραμματιστικά μοντέλα που υιοθετούν την μια ή την άλλη εικόνα για τη κύρια μνήμη.

2.2.1 Προγραμματιστικό Μοντέλο Κοινού Χώρου Διευθύνσεων

Στο μοντέλο αυτό η μνήμη ή συγκεκριμένα όλες οι διευθύνσεις της θεωρούνται ότι είναι διαθέσιμες στον προγραμματιστή κατά τη σχεδίαση ενός παράλληλου προγράμματος. Οι αρχιτεκτονικές μοιραζόμενης μνήμης, UMA και NUMA, ενέπνευσαν αυτό το προγραμματιστικό μοντέλο διότι δημιουργούν ένα κοινό χώρο διευθύνσεων για την κύρια μνήμη είτε αυτή είναι κεντρική είτε κατανεμημένη. Χάρη σε αυτή την ιδέα ο προγραμματισμός απλοποιείται αρκετά και η δομή του κώδικα θυμίζει πολύ τη δομή ενός ακολουθιακού προγράμματος. Επίσης, λόγω του κοινού χώρου διευθύνσεων μια περιοχή στη μνήμη μπορεί να χρησιμοποιηθεί ως σημείο επικοινωνίας για την ανταλλαγή δεδομένων μεταξύ των μονάδων επεξεργασίας. Το πρόβλημα όμως είναι ότι δημιουργούνται συνθήκες ανταγωνισμού μεταξύ των επεξεργαστών για τα δεδομένα που είναι αποθηκευμένα

στην κοινόχρηστη περιοχή με κίνδυνο αυτά να αλλοιωθούν. Για το λόγο αυτό στο προγραμματιστικό μοντέλο κοινού χώρου διευθύνσεων ορίζονται και δομές οι οποίες χρησιμοποιούνται για το συγχρονισμό της πρόσβασης των επεξεργαστών σε αυτές τις περιοχές της μνήμης. Αυτό το προγραμματιστικό μοντέλο δεν μπορεί να υλοποιηθεί εύκολα για καθαρές αρχιτεκτονικές κατανεμημένης μνήμης εξαιτίας της τελείως διαφορετικής φιλοσοφίας στη διευθυνσιοδότηση της μνήμης του συστήματος. Παρόλα αυτά υπάρχουν προγραμματιστικά μοντέλα, όπως το μοντέλο PGAS, τα οποία υλοποιούνται με βάση τις απαιτήσεις μιας κατανεμημένης αρχιτεκτονικής, αλλά παρέχουν μια εικόνα ενός παγκόσμιου κοινού χώρου διευθύνσεων στον προγραμματιστή.

2.2.2 Προγραμματιστικό Μοντέλο Ανταλλαγής Μηνυμάτων

Στις αρχιτεκτονικές κατανεμημένης μνήμης, ο κάθε επεξεργαστής έχει τη δική του τοπική μνήμη. Όλες οι λογικές διευθύνσεις που παράγει αναφέρονται στις φυσικές διευθύνσεις της τοπικής του μνήμης. Δηλαδή, δεν υπάρχει καμία επικοινωνία μεταξύ των επεξεργαστών μέσω μιας περιοχής κοινών διευθύνσεων. Αυτό σημαίνει πως δεν υπάρχουν συνθήκες ανταγωνισμού μεταξύ των επεξεργαστών και πως δεν είναι ανάγκη ο συγχρονισμός των εργασιών τους. Για την ανταλλαγή δεδομένων, οι μονάδες επεξεργασίας επικοινωνούν μέσω του δικτύου διασύνδεσής του. Η επικοινωνία γίνεται μέσω μηνυμάτων, τα οποία παρέχουν τις κατάλληλες πληροφορίες για το ποιός είναι ο αποστολέας, ποιός είναι ο παραλήπτης και ποιό είναι το αίτημα προς εκπλήρωση. Το προγραμματιστικό μοντέλο ανταλλαγής μηνυμάτων επικεντρώνεται στο πως θα διεκπεραιωθεί η επικοινωνία μεταξύ των επεξεργαστών και στην μείωση των σφαλμάτων κατά την ανταλλαγή μηνυμάτων. Αυτό σημαίνει πως μεγάλο μέρος του προγραμματισμού ενός παράλληλου προγράμματος αφιερώνεται στα θέματα επικοινωνίας με αποτέλεσμα η δουλειά ενός προγραμματιστή να γίνεται πιο περίπλοκη. Παρόλα αυτά, αυτό το μοντέλο προγραμματισμού είναι πιο γενικό από το μοντέλο κοινού χώρου διευθύνσεων διότι μπορεί να υλοποιηθεί εύκολα και σε αρχιτεκτονικές μοιραζόμενης μνήμης, χωρίς αυτό όμως να σημαίνει πως το παράλληλο πρόγραμμα θα φτάσει σε επίδοση ένα αντίστοιχο πρόγραμμα αλλά δομημένο με τη τεχνική του κοινού χώρου διευθύνσεων.

Κεφάλαιο 3

Βιβλιοθήκη **OpenMP**

Όπως αναφέρθηκε ο κύριος στόχος της διπλωματικής εργασίας είναι η δημιουργία ενός μοντέλου πρόβλεψης επίδοσης προγραμμάτων τα οποία εκτελούνται σε αρχιτεκτονικές μοιραζόμενης μνήμης με προγραμματιστικό μοντέλο κοινού χώρου διευθύνσεων. Η βιβλιοθήκη OpenMP αποτελεί μια υλοποιημένη διεπαφή του συγκεκριμένου προγραμματιστικού μοντέλου, η οποία έχει τυποποιηθεί σε πολλές εκδόσεις και είναι αρκετά διαδεδομένη στη βιομηχανία. Η βιβλιοθήκη OpenMP διαθέτει όλα απαραίτητα τα εργαλεία για τον παραλληλισμό ενός κώδικα, καθώς υποστηρίζει το διαμοιρασμό των παράλληλων εργασιών σε νήματα και το συγχρονισμό τους σε κοινό χώρο διευθύνσεων. Για τη γλώσσα C η εφαρμογή των παράλληλων δομών γίνεται μέσω εντολών προς τον προεπεξεργαστή της γλώσσας, ενώ οι δομές αυτές μπορούν να ελεγχθούν κατά την εκτέλεση του προγράμματος μέσω περιβαλλοντικών μεταβλητών. Έτσι, η βιβλιοθήκη OpenMP είναι πολύ εύκολη στη χρήση της για την παραλληλοποίηση ενός προγράμματος και τον έλεγχό του[9].

Κεφάλαιο 4

Ο πυρήνας Jacobi

Για τη δημιουργία του μοντέλου μας, χρειάστηκε να έχουμε στη διάθεσή μας έναν πυρήνα προγράμματος ο οποίος είναι εύκολα παραλληλοποιήσιμος και η δομή του είναι απλή για τον έλεγχο πιθανών λαθών. Η βασική ιδέα είναι να τον χρησιμοποιήσουμε ως σημείο αναφοράς κατά την παραγωγή του μοντέλου μας, για να εξάγουμε τα κατάλληλα χαρακτηριστικά που χρειάζονται για τις προβλέψεις του. Επιλέγεται ο πυρήνας Jacobi ο οποίος ανήκει στην οικογένεια των προγραμμάτων stencils[10]. Τα προγράμματα stencils χρησιμοποιούνται κατά κύριο λόγο στο επιστημονικό πεδίο της αριθμητικής ανάλυσης. Τα δεδομένα τους είναι αποθηκευμένα σε πίνακες, πολλών διαστάσεων. Στα προγράμματα αυτά, η τιμή των στοιχείων του πίνακα ανανεώνεται ανά επανάληψη χρησιμοποιώντας τις τιμές των γειτονικών τους στοιχείων. Για παράδειγμα, στον πυρήνα Jacobi η νέα τιμή ενός στοιχείου είναι ο μέσος όρος των τεσσάρων στοιχείων που βρίσκονται πάνω, κάτω, δεξιά και αριστερά από αυτό. Ο πυρήνας Jacobi χρησιμοποιείται για την αριθμητική επίλυση μερικών διαφορικών εξισώσεων, όπως είναι το φαινόμενο διάδοσης της θερμότητας εντός ενός χώρου.

Η υλοποίησή του στη γλώσσα C παρουσιάζεται παρακάτω.

```
1  void Jacobi(double ** u_previous, double ** u_current,
2              int X_min, int X_max, int Y_min, int Y_max)
3  {
4
5      int i,j;
6
7      for (i=X_min;i<X_max;i++) {
8          for (j=Y_min;j<Y_max;j++) {
9              u_current[i][j]=(u_previous[i-1][j] + u_previous[i+1][j] +
10                              u_previous[i][j-1] + u_previous[i][j+1]) / 4.0;
11         }
12     }
13 }
```

Ο πίνακας u_previous περιέχει τις τιμές των στοιχείων της προηγούμενης επανάληψης, ενώ ο πίνακας u_current της τωρινής επανάληψης στην οποία ανανεώνονται τα στοιχεία του με νέες τιμές τους μέσους όρους των τιμών που είχαν τα γειτονικά στοιχεία του στην προηγούμενη επανάληψη.

Αν οι πίνακες προσομοιώνουν ένα δισδιάστατο χώρο, τότε ο πυρήνας Jacobi προσομοιώνει το πως μεταδίδεται η θερμότητα στο χώρο αυτό με το πέρασμα του χρόνου. Όπου ο χρόνος είναι οι επαναλήψεις.

Κεφάλαιο 5

Κλιμακωσιμότητα

Κατά τη μετατροπή ενός ακολουθιακού προγράμματος σε παράλληλο, προκύπτουν πολλές παράλληλες εκδοχές. Στόχος είναι να βρεθεί η καλύτερη παράλληλη εκδοχή η οποία αξιοποιεί βέλτιστα τα χαρακτηριστικά της παράλληλης αρχιτεκτονικής πάνω στην οποία εκτελείται. Η κλιμακωσιμότητα ενός παράλληλου συστήματος χρησιμοποιείται ως έννοια σύγκρισης μεταξύ των παράλληλων εκδοχών του σειριακού προγράμματος. Ορίζεται με βάση το πλήθος των μονάδων επεξεργασίας που προσφέρει μια παράλληλη αρχιτεκτονική. Η παράλληλη εκδοχή που κλιμακώνεται καλύτερα από τις υπόλοιπες είναι και η βέλτιστη εκδοχή.

Ορισμός 2 *Κλιμακωσιμότητα είναι η ικανότητα ενός παράλληλου αλγορίθμου να μπορεί να αξιοποιεί αποδοτικά όσους νέους πυρήνες του προσφέρει ένα υπολογιστικό σύστημα παράλληλης αρχιτεκτονικής.*

Η κλιμακωσιμότητα θεμελιώνεται μαθηματικά μέσω δύο μεγεθών, της επιτάχυνσης S και της αποδοτικότητας E. Τα δύο μεγέθη είναι συμπληρωματικά και παρουσιάζουν δύο διαφορετικές οπτικές γωνίες για την έννοια της κλιμακωσιμότητας ενός παράλληλου συστήματος. Συγκεκριμένα, όταν η επιτάχυνση ενός παράλληλου συστήματος βελτιώνεται σημαίνει πως ο χρόνος λειτουργίας των μονάδων επεξεργασίας της αρχιτεκτονικής του μειώνεται. Όταν η αποδοτικότητα ενός παράλληλου συστήματος βελτιώνεται αυτό μεταφράζεται σε καλύτερο διαμοιρασμό του φόρτου εργασίας στις μονάδες επεξεργασίας της αρχιτεκτονικής. Φορμαλιστικά, η επιτάχυνση διατυπώνεται ως

$$S = \frac{T_s}{T_p} \tag{5.1}$$

όπου το T_s είναι ο χρόνος εκτέλεσης της βέλτιστης σειριακής εκδοχής ενός προγράμματος και T_p είναι ο χρόνος εκτέλεσης μιας παράλληλης εκδοχής του αντίστοιχου προγράμματος για p στο πλήθος μονάδες επεξεργασίας. Η αποδοτικότητα ορίζεται με βάση την επιτάχυνση ως εξής.

$$E = \frac{S}{p} \tag{5.2}$$

Σε όλη την ιστορία των παράλληλων συστημάτων επεξεργασίας τα δύο αυτά μαθηματικά μεγέθη χρησιμοποιήθηκαν για τον ορισμό των νόμων που διέπουν τη λειτουργία τους. Στα επόμενα κεφάλαια αναλύονται αυτοί οι νόμοι, τα συμπεράσματά τους και οι εφαρμογές τους.

5.1 Νόμος του Amdahl

Η ιδέα του νόμου του Amdahl είναι απλή. Αν βελτιωθεί ένα μέρος ενός συστήματος με καλύτερα υλικά ή χρησιμοποιηθεί μια καλύτερη τεχνική για την αξιοποιήσει των πόρων του, τότε η βελτίωση της επίδοσης του συνολικού συστήματος περιορίζεται από το ποσοστό συμμετοχής στη συνολική λειτουργία του μέρους που βελτιώθηκε. Δηλαδή αν το μέρος αυτό επηρεάζει κατά 80% τη συνολική λειτουργία όλου του συστήματος τότε η βελτίωση επίδοσής του θα είναι αισθητή στην αύξηση της επίδοσης του συνολικού συστήματος. Αν όμως επηρεάζει το 10% της συνολικής λειτουργίας τότε δεν είναι αισθητή στη συνολική επίδοση όλου του συστήματος.

Ο νόμος του Amdahl μπορεί να εφαρμοσθεί τόσο σε ζητήματα υλισμικού όσο και λογισμικού. Στη μελέτη των παράλληλων συστημάτων επεξεργασίας εξετάζεται το πόσο συνεισφέρει η παραλληλοποίηση ενός μέρους ενός προγράμματος στη συνολική επίδοση όλου του προγράμματος. Τα μεγέθη τα οποία χρησιμοποιούνται για τη μαθηματική διατύπωση του νόμου του Amdahl, είναι το μέγεθος της επιτάχυνσης (Εξίσωση (5.1)), το σειριακό (ή ακολουθιακό) μέρος του προγράμματος που συμβολίζεται ως s και το παραλληλοποιήσιμο μέρος του προγράμματος που συμβολίζεται ως f. Το σειριακό μέρος του προγράμματος είναι αυτό το οποίο δεν μπορεί να παραλληλοποιηθεί. Το παραλληλοποιήσιμο μέρος είναι το μέρος που επιδέχεται παραλληλοποίηση. Δηλαδή, οι εργασίες που εκτελούνται σε αυτό το μέρος του προγράμματος μπορούν να μοιραστούν στις μονάδες επεξεργασίας ενός παράλληλου συστήματος για ταυτόχρονη διεκπεραίωσή τους. Επειδή, τα s και f διαχωρίζουν το συνολικό πρόγραμμα σε μερίδες, ορίζονται ως ποσοστά του συνολικού κώδικα. Συνολικά, $s + f = 1$ [13]. Στη μοντέρνα του μορφή, υπό αυτό το πλαίσιο, ο νόμος του Amdahl εκφράζει το χρόνο εκτέλεσης μιας παράλληλης εκδοχής ενός προγράμματος ως άθροισμα δύο χρόνων[14]. Του χρόνου εκτέλεσης για το μέρος του προγράμματος που δεν παραλληλοποιείται και του χρόνου εκτέλεσης για το μέρος του προγράμματος που παραλληλοποιείται ως,

$$T_p = s \times T_s + f \times \frac{T_s}{p} = T_s(s + \frac{f}{p}) \tag{5.3}$$

Αντικαθιστώντας το T_p στην εξίσωση επιτάχυνσης (Εξίσωση (5.1)), έχουμε την τελική μορφή του νόμου του Amdahl

$$S = \frac{T_s}{T_p} = \frac{1}{s + \frac{f}{p}} \tag{5.4}$$

Μελετώντας ποιοτικά αυτή την εξίσωση μπορούν να εξαχθούν σημαντικά διαισθητικά συμπεράσματα για τη λειτουργία των παράλληλων συστημάτων. Αρχικά, όταν το πλήθος των πυρήνων p αυξάνεται προς το άπειρο, η επιτάχυνση ισούται με το αντίστροφο του σειριακού μέρους του προ-

γράμματος, δηλαδή με $\frac{1}{s}$.

$$\lim_{p\to\infty} S = \lim_{p\to\infty} \frac{1}{s + \frac{f}{p}} = \frac{1}{s}$$

Αυτό σημαίνει ότι όσο και αν είναι το πλήθος των μονάδων επεξεργασίας για μια παράλληλη αρχιτεκτονική η κλιμάκωση ενός προγράμματος θα περιορίζεται πάντα από το χρόνο εκτέλεσης του σειριακού της μέρους. Κατά συνέπεια η επίδοση ενός παράλληλου προγράμματος θα περιορίζεται πάντα από ένα ανώτατο κατώφλι λόγω του s.

Για s πολύ μικρό, να τείνει στο 0, τότε το παραλληλοποιήσιμο μέρος του προγράμματος f τείνει στο 1. Έτσι, από την εξίσωση (5.4) η επιτάχυνση θα ισούται με,

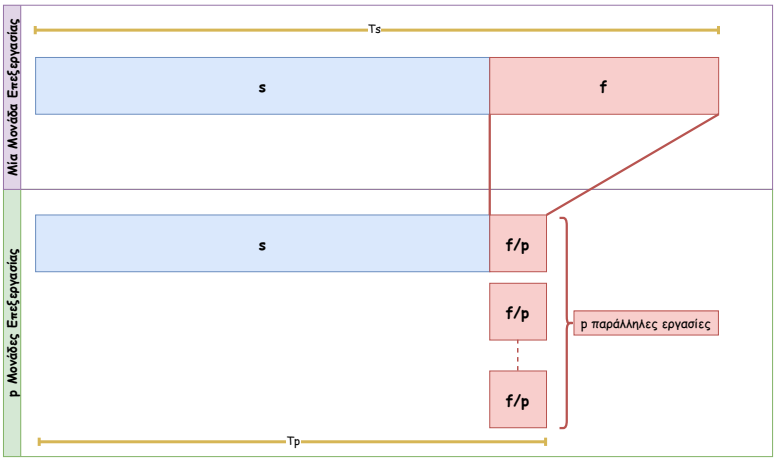
$$\lim_{s\to 0} S = \lim_{s\to 0} \frac{1}{s + \frac{f}{p}} = \frac{1}{0 + \frac{1}{p}} = p$$

Δηλαδή η επιτάχυνση ισούται με το πλήθος των μονάδων επεξεργασίας που χρησιμοποιούνται για την εκτέλεση του προγράμματος. Αυτή, η επιτάχυνση ονομάζεται **γραμμική επιτάχυνση** και εκφράζει τη βέλτιστη επίδοση που μπορεί να έχει ένα παράλληλο πρόγραμμα, καθώς αξιοποιεί στο μέγιστο τους υπολογιστικούς πόρους που προσφέρει η αρχιτεκτονική του συστήματος.

Αντιθέτως, για f πολύ μικρό, να τείνει στο 0, το s τείνει στο 1 και η επιτάχυνση ισούται με,

$$\lim_{f\to 0} S = \lim_{f\to 0} \frac{1}{s + \frac{f}{p}} = \frac{1}{s + \frac{0}{p}} = 1$$

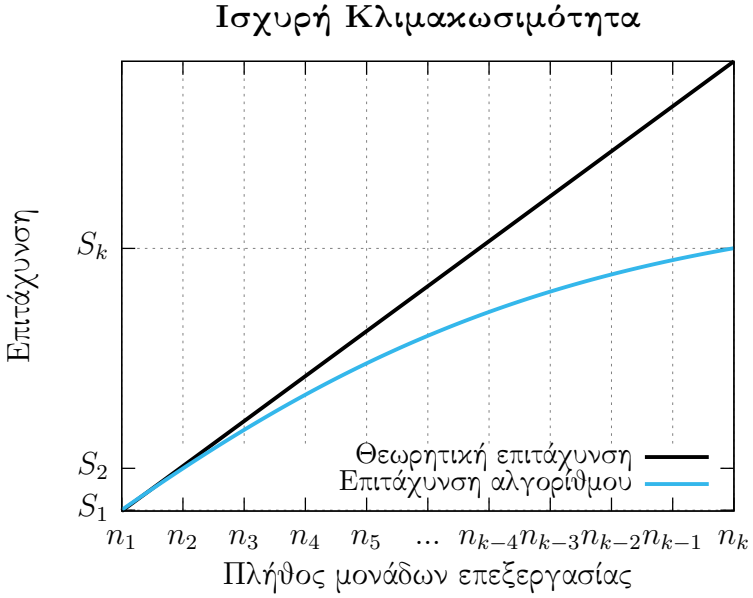
Δηλαδή, το παράλληλο πρόγραμμα δεν έχει επιταχυνθεί καθόλου από τη σειριακή εκδοχή του. Η επίδοση του ισούται με την επίδοση της σειριακής εκδοχής του.



Σχήμα 5.1: Ο νόμος του Amdahl αποδομένος σε σχήμα

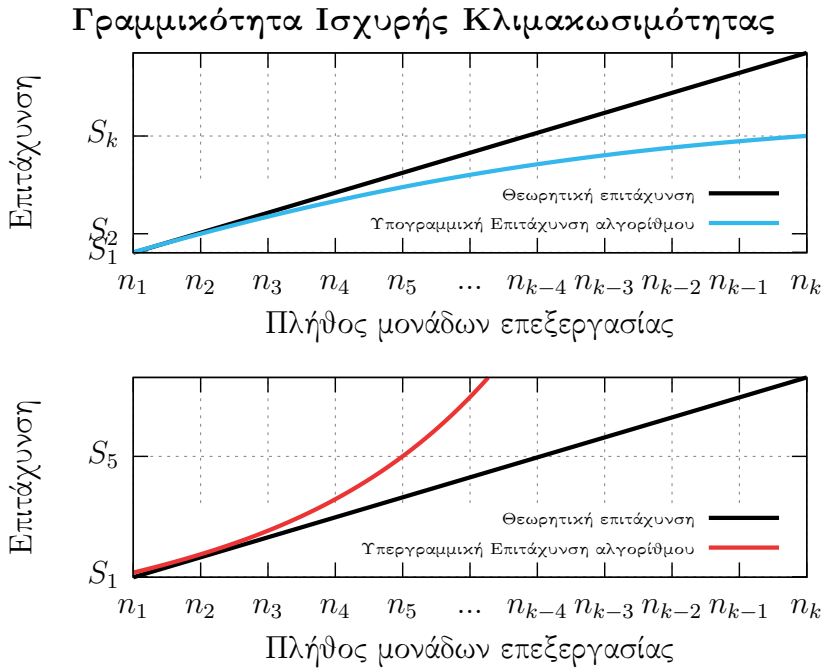
5.1.1 Ισχυρή κλιμακωσιμότητα

Η ισχυρή κλιμακωσιμότητα είναι μια έννοια που προκύπτει άμεσα από το νόμο του Amdahl. Μια υπόθεση που γίνεται αλλά δε σχολιάζεται στο νόμο του Amdahl είναι ότι κατά τη μελέτη επίδοσης ενός παράλληλου προγράμματος, αυτό και οι είσοδοί του παραμένουν ίδια. Το μέγεθος του προβλήματος δεν αλλάζει καθώς αυξάνονται οι μονάδες επεξεργασίας της αρχιτεκτονικής που χρησιμοποιούνται. Η ισχυρή εκδοχή αναγνωρίζει ως μοναδικό αίτιο για την κλιμάκωση ενός παράλληλου προγράμματος το πλήθος των μονάδων επεξεργασίας. Στα διαγράμματα ισχυρής κλιμακωσιμότητας δίνεται ενδιαφέρον η αποκλειστική απεικόνιση της επίδοσης ενός παράλληλου προγράμματος ως προς το πλήθος των μονάδων επεξεργασίας που χρησιμοποιήθηκαν κατά την εκτέλεση του. Δεν λαμβάνεται υπόψη κανένα άλλο μέγεθος. Για την κατασκευή ενός διαγραμμάτος ισχυρής κλιμακωσιμότητας χρησιμοποιείται η γενική εξίσωση της επιτάχυνσης (Εξίσωση (5.1)). Αλλάζοντας το πλήθος των μονάδων επεξεργασίας παίρνουμε διαφορετικούς χρόνους εκτέλεσης T_p και κατά συνέπεια διαφορετικές επιταχύνσεις. Αν θεωρήσουμε αύξουσα ακολουθία πλήθους μονάδων επεξεργασίας $(n_1, n_2, \ldots, n_k)$ τότε έχουμε ακολουθία διαφορετικών επιταχύνσεων $(S_{n_1}, S_{n_2}, \ldots, S_{n_k})$. Απεικονίζοντας τις τιμές αυτές σε ένα γράφημα ως σημεία, με τετμημένη το πλήθος μονάδων επεξεργασίας n_i και τεταγμένη την τιμή της επιτάχυνσης S_{n_i} δημιουργούμε το διάγραμμα ισχυρής κλιμακωσιμότητας για ένα παράλληλο πρόγραμμα (Σχήμα 5.2).



Σχήμα 5.2: Αφαιρετικό διάγραμμα ισχυρής κλιμακωσιμότητας για ένα παράλληλο πρόγραμμα

Μελετώντας το διάγραμμα αυτό ένας επιστήμονας υπολογιστών μπορεί να διαγνώσει αν το πρόγραμμα κλιμακώνει καλά ως προς το πλήθος των μονάδων επεξεργασίας. Συνήθως καθώς το πλήθος των μονάδων επεξεργασίας αυξάνεται η επιτάχυνση φτάνει σε ένα σημείο κορεσμού και από



Σχήμα 5.3: Υπογραμμική και Υπεργραμμική επιτάχυνση

εκείνο το σημείο και έπειτα είτε μένει σταθερή είτε πέφτει απότομα. Στό Σχήμα 5.2 η επιτάχυνση ενός προγράμματος απεικονίζεται ως ένα γράφημα το οποίο δεν ξεπερνάει τη γραμμική συνάρτηση. Αυτού του είδους επιτάχυνση ονομάζεται βιβλιογραφικά ως **υπογραμμική επιτάχυνση**. Η ιδανική κλιμάκωση για ένα πρόγραμμα όπως αναφέρθηκε είναι η γραμμική επιτάχυνση ως προς το πλήθος των μονάδων επεξεργασίας. Δηλαδή, αν χρησιμοποιούνται 5 μονάδες επεξεργασίας για την εκτέλεση ενός παράλληλου προγράμματος η επιτάχυνση είναι 5, αν είναι 100 μονάδες επεξεργασίας τότε η επιτάχυνση είναι 100 και ούτω καθεξής. Οι λόγοι για τους οποίους πολλά προγράμματα δε φτάνουν σε γραμμική απόκριση μελετάται σε επόμενες ενότητες. Σε πολύ σπάνιες περιπτώσεις η επιτάχυνση ενός παράλληλου προγράμματος ξεπερνάει τη γραμμική επιτάχυνση. Αυτό το φαινόμενο ονομάζεται **υπεργραμμική επιτάχυνση** και οφείλεται κυρίως στις κρυφές μνήμες των επεξεργαστών. Συγκεκριμένα, καθώς αυξάνεται το πλήθος των μονάδων επεξεργασίας που χρησιμοποιούνται για την επίλυση του προβλήματος άλλο τόσο αυξάνεται και το πλήθος των κρυφών μνημών που συμμετέχουν στη διαδικασία υπολογισμού. Αν το μέγεθος εισόδου είναι μικρό, όλο και μεγαλύτερο μέρος του αποθηκεύεται στις κρυφές μνήμες των μονάδων επεξεργασίας. Ως αποτέλεσμα επιτυγχάνονται επιταχύνσεις που ξεπερνούν τη γραμμική επιτάχυνση. Στο σχήμα 5.3 παρουσιάζονται δύο γραφήματα επιταχύνσεων για ένα πρόγραμμ υπογραμμικής επιτάχυνσης και ένα υπεργραμμικής επιτάχυνσης.

5.2 Νόμος του Gustafson

Ο νόμος του Amdahl οδήγησε σε προβληματισμούς για το κατά πόσο είναι χρήσιμα τα παράλληλα συστήματα μεγάλης κλίμακας, μιας και η επίδοση αυτών περιορίζεται από το $\frac{1}{s}$ όσο αυξάνεται το πλήθος των μονάδων επεξεργασίας. Στην πράξη, τα γραφήματα ισχυρής κλιμακωσιμότητας που δέχονται ως μοναδική μεταβλητή το πλήθος των μονάδων επεξεργασίας αποδεικνύουν πως για μεγάλο πλήθος αυτών, η επιτάχυνση φτάνει σε κορεσμό ή μειώνεται μετά από το σημείο κορεσμού.

Λύση σε αυτό το πρόβλημα δόθηκε από τον Gustafson που υπέθεσε πως όσο αυξάνονται οι μονάδες επεξεργασίας που είναι διαθέσιμες προς αξιοποίηση θα πρέπει να αυξάνεται και το μέγεθος προβλήματος του αλγορίθμου που εκτελείται[15]. Το μέγεθος προβλήματος ενός αλγορίθμου είναι το πλήθος των πράξεων που εκτελεί ο αλγόριθμος αυτός και εξαρτάται άμεσα από το μέγεθος εισόδου του. Για παράδειγμα, αν το μέγεθος εισόδου ενός προγράμματος είναι N και το πρόγραμμα υπολογίζει την πρόσθεση δύο $N \times N$ πινάκων τότε το μέγεθος προβλήματος είναι N^2. Αν το πρόγραμμα υπολογίζει τον πολλαπλασιασμό των δύο πινάκων τότε το μέγεθος προβλήματος είναι N^3.

Βασική ιδέα αυτής της υπόθεσης είναι να αυξηθεί ο φόρτος εργασίας στις μονάδες επεξεργασίας σε τέτοιο βαθμό ώστε ο χρόνος εκτέλεσης του παραλληλοποιήσιμου μέρους ενός προγράμματος, να είναι ίδιος σε όλες. Θεωρώντας ως T_p το συνολικό χρόνο εκτέλεσης του παράλληλου προγράμματος σε p μονάδες επεξεργασίας τότε η σειριακή εκδοχή του απαιτεί χρόνο,

$$T_s = s \times T_p + p \times f \times T_p = T_p(s + p \times f) \tag{5.5}$$

Η επιτάχυνση τότε εκφράζεται μέσω της εξίσωσης (5.1) ως,

$$S = \frac{T_s}{T_p} = s + p \times f \tag{5.6}$$

Αναλύοντας την εξίσωση (5.6) με μια ποιοτική ματιά, εξάγονται ενδιαφέροντα συμπεράσματα για τα παράλληλα συστήματα επεξεργασίας.

Θεωρώντας πως το πλήθος των μονάδων επεξεργασίας τείνει προς το άπειρο τότε η επιτάχυνση φαίνεται πως απειρίζει.

$$\lim_{p \to \infty} S = \lim_{p \to \infty} (s + p \times f) = \infty$$

Με βάση το νόμο του Gustafson παρατηρείται πως επιτάχυνση δεν περιορίζεται από το κατώφλι $\frac{1}{s}$ όπως υποδηλώνει ο νόμος του Amdahl. Ο νόμος του Gustafson δείχνει πως με κατάλληλη αύξηση του πλήθους των πράξεων εντός ενός προγράμματος, η επιτάχυνση θα συνεχίζει να αυξάνεται όσο αυξάνεται το πλήθος των μονάδων επεξεργασίας. Έτσι, τα συστήματα παράλληλης επεξεργασίας πάρα πολλών μονάδων επεξεργασίας συνεχίζουν να είναι χρήσιμα.

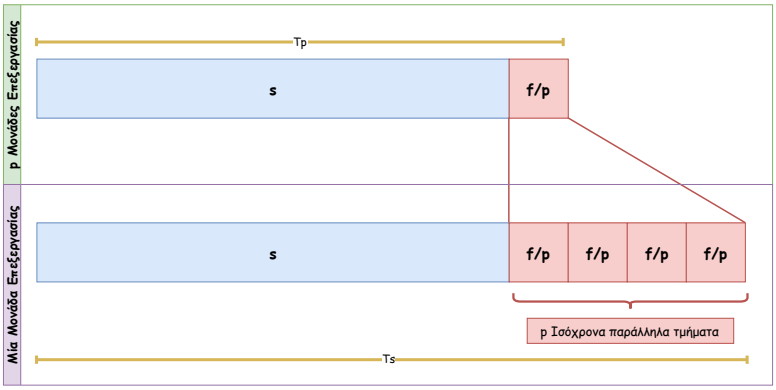
Επίσης, ο νόμος του Gustafson διατηρεί τις ιδιότητες του νόμου του Amdahl για το s και το f. Αν το μέρος του προγράμματος που δεν παραλληλοποιείται είναι πολύ μικρό σε ποσοστό, δηλαδή τείνει στο μηδέν, τότε το μέρος που παραλληλοποιείται έχει σημαντικότερο ρόλο και τείνει στο ένα. Η επιτάχυνση σε αυτή την περίπτωση ισούται με το πλήθος των μονάδων επεξεργασίας

που χρησιμοποιούνται κατά την εκτέλεση του παράλληλου προγράμματος. Αυτό εκφράζει την περίπτωση της γραμμικής επιτάχυνσης όπως και στο νόμο του Amdahl,

$$\lim_{s\to 0} S = \lim_{s\to 0}(s + p \times f) = 0 + p \times 1 = p$$

Στον αντίποδα αυτού, αν το μέρος του κώδικα που παραλληλοποιείται είναι μικρό σε μέγεθος, δηλαδή τείνει στο μηδέν, τότε το μέρος του που δεν παραλληλοποιείται τείνει στο ένα και η επιτάχυνση του παράλληλου αλγορίθμου είναι μονάδα. Δηλαδή, η επίδοσή του ισούται με την επίδοση του αντίστοιχου ακολουθιακού προγράμματος.

$$\lim_{f\to 0} S = \lim_{f\to 0}(s + p \times f) = 1 + p \times 0 = 1$$



Σχήμα 5.4: Ο νόμος του Gustafson αποδομένος σε σχήμα

5.2.1 Ισοαποδοτικότητα

Με το νόμο του Gustafson διατυπώνεται η σημαντικότητα του μεγέθους προβλήματος για μελέτη παράλληλων συστημάτων επεξεργασίας μεγάλης κλίμακας.

Οι εξισώσεις των χρόνων εκτέλεσης (5.3) και (5.5) που εκφράζουν τις υποθέσεις των νόμων Amdahl και Gustafson αντίστοιχα, μπορούν να διατυπωθούν σε μια γενικότερη μορφή μετά από μερικούς αλγεβρικούς μετασχηματισμούς. Αρχικά για την εξίσωση (5.3) και το νόμο του Amdahl έχουμε,

$$T_p = sT_s + \frac{1-s}{p}T_s \Rightarrow T_p = \frac{T_s}{p} + s(1-\frac{1}{p})T_s \tag{5.7}$$

και για την εξίσωση (5.5) και το νόμο του Gustafson έχουμε,

$$T_s = sT_p + p(1-s)T_p \Rightarrow T_p = \frac{T_s}{p} + \frac{s(p-1)}{p^2 + p(1-p)s}T_s \tag{5.8}$$

Και στις δύο εξισώσεις απομονώνεται ο όρος $\frac{T_s}{p}$ που εκφράζει την ιδανική περίπτωση της γραμμικής κλιμάκωσης του προγράμματος. Οι δευτερεύοντες όροι στα αθροίσματα της κάθε εξίσωσης είναι αλγεβρικές εκφράσεις που εκφράζουν τις πρόσθετες καθυστερήσεις στους χρόνους εκτέλεσης. Όπως φαίνεται, εξαρτώνται από το s, που είναι το ποσοστό του κώδικα που δεν μπορεί να παραλληλοποιηθεί. Δηλαδή, τα πρόσθετα κόστη στους νόμους προκύπτουν λόγω της ύπαρξης εργασιών εντός του προγράμματος που δεν είναι δυνατό να εκτελεστούν παράλληλα και εκτελούνται σειριακά. Επίσης, οι όροι αυτοί εξαρτώνται και από το πλήθος των μονάδων επεξεργασίας. Αυτή η διαπίστωση για το s και το p είναι πολύ σημαντική όταν θα γίνει μελέτη σε επόμενο κεφάλαιο για τα πρόσθετα κόστη που περιορίζουν την επίδοση ενός παράλληλου προγράμματος.

Παρατηρώντας τους δύο νόμους στις μορφές (5.7) και (5.8) οι δευτερεύοντες όροι μπορούν να αντικατασταθούν από μια γενική συνάρτηση $\mathcal{O}(s,p)$ που εξαρτάται από το s και το πλήθος των μονάδων επεξεργασίας της αρχιτεκτονικής p. Ο χρόνος $\mathcal{O}$ προκύπτει από την αγγλική λέξη overhead.

$$T_p = \frac{T_s}{p} + \mathcal{O} \tag{5.9}$$

Η εξίσωση (5.9) μπορεί να χρησιμοποιηθεί για τον ορισμό της γενικευμένης επιτάχυνσης για τα παράλληλα συστήματα επεξεργασίας,

$$S = \frac{T_s}{T_p} = \frac{T_s}{\frac{T_s}{p} + \mathcal{O}} = \frac{p}{1 + p\frac{\mathcal{O}}{T_s}} \tag{5.10}$$

Εξετάζεται η περίπτωση κατά την οποία το πλήθος των μονάδων επεξεργασίας αυξάνεται πολύ τείνοντας στο άπειρο.

$$\lim_{p\to\infty} S = \lim_{p\to\infty} \frac{T_s}{\frac{T_s}{p} + \mathcal{O}} = \frac{T_s}{\lim_{p\to\infty} \mathcal{O}}$$

Γνωρίζουμε ήδη πως το $\mathcal{O}$ είναι συνάρτηση που εξαρτάται από το πλήθος των μονάδων επεξεργασίας αλλά δεν ξέρουμε το πως εξαρτάται από αυτό. Διακρίνονται οι επόμενες περιπτώσεις,

- Στην περίπτωση που το $\mathcal{O}$ είναι ανάλογο του πλήθους των μονάδων επεξεργασίας p, θα έχουμε

  $$\lim_{p\to\infty} S = \frac{T_s}{\lim_{p\to\infty} \mathcal{O}} = 0$$

  δηλαδή η επιτάχυνση θα τείνει στο μηδέν. Αυτό στην πράξη δε γίνεται αλλά μεταφράζεται ως το γεγονός που το $\mathcal{O}$ θα περιορίζει τόσο πολύ το παραλληλισμό του προγράμματος που ουσιαστικά η επίδοσή του θα πλησιάζει την επίδοση της ακολουθιακής εκδοχής του.

- Στην περίπτωση που το $\mathcal{O}$ είναι αντιστρόφως ανάλογο του πλήθους των μονάδων επεξεργασίας p, θα έχουμε

  $$\lim_{p\to\infty} S = \frac{T_s}{\lim_{p\to\infty} \mathcal{O}} = \infty$$

  δηλαδή θα έχουμε την περίπτωση για την οποία εξηγεί ο νόμος του Gustafson.

- Στην περίπτωση που το $\mathcal{O}$ είναι ανεξάρτητο του πλήθους των μονάδων επεξεργασίας p, θα έχουμε

$$\lim_{p\to\infty} S = \frac{T_s}{\mathcal{O}(s)}$$

δηλαδή μια γενικότερη περίπτωση του νόμου του Amdahl. Αν το $\mathcal{O} = sT_s$, τότε $S = \frac{1}{s}$.

Οι V. Kumar, A. Grama και A. Gupta για να μελετήσουν την κλιμακωσιμότητα ενός παράλληλου συστήματος υιοθέτησαν ως μετρική την αποδοτικότητα αντί της επιτάχυνσης όπως στους νόμους Amdahl και Gustafson. Χρησιμοποιώντας την εξίσωση (5.10) όρισαν την αποδοτικότητα ως

$$E = \frac{S}{p} = \frac{1}{1 + p\frac{\mathcal{O}}{T_s}} \tag{5.11}$$

και την εξίσωση αυτή την ονόμασαν «συνάρτηση ισοαποδοτικότητας» [16][17].

Μετά προχώρησαν στη λογική υπόθεση ότι όσο το μέγεθος του προβλήματος ενός προγράμματος αυξάνεται θα αυξάνεται και ο χρόνος εκτέλεσης του. Για πολύ μεγάλο μέγεθος προβλήματος ο χρόνος εκτέλεσης της σειριακής εκδοχής του θα είναι και αυτός πολύ μεγάλος. Τελικά, για χρόνο εκτέλεσης να τείνει στο άπειρο η αποδοτικότητα του παράλληλου συστήματος ισούται με

$$\lim_{T_s\to\infty} E = \lim_{T_s\to\infty} \frac{1}{1 + p\frac{\mathcal{O}}{T_s}} = 1$$

Η αποδοτικότητα του παράλληλου συστήματος δηλαδή πλησιάζει τη μονάδα όταν αυξάνεται πολύ το μέγεθος προβλήματος του. Αυτό συνεπάγεται ότι το παράλληλο σύστημα κλιμακώνει γραμμικά.

Αντιθέτως, όταν οι καθυστερήσεις αυξάνονται πιο γρήγορα από το μέγεθος προβλήματος τότε έχουμε,

$$\lim_{\mathcal{O}\to\infty} E = \lim_{\mathcal{O}\to\infty} \frac{1}{1 + p\frac{\mathcal{O}}{T_s}} = 0$$

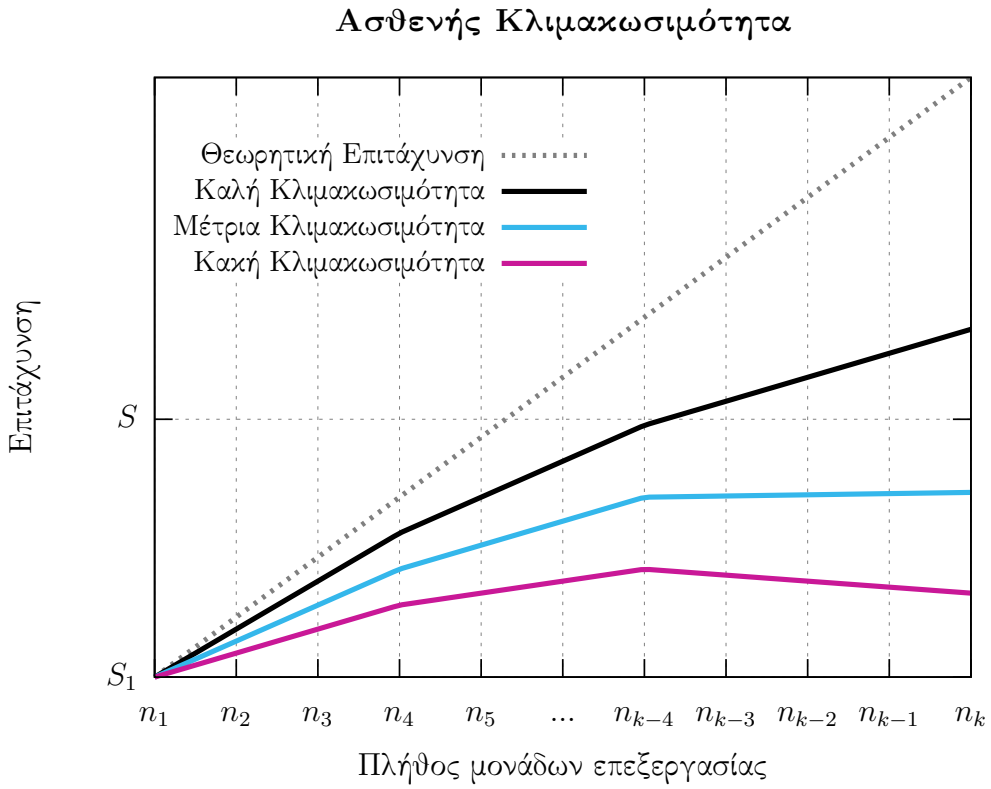
Οι πρόσθετες καθυστερήσεις οδηγούν την αποδοτικότητα ενός παράλληλου συστήματος στο μηδέν.

Από την παραπάνω ποιοτική ανάλυση κατανοούμε πως το πλήθος μονάδων επεξεργασίας και το μέγεθος προβλήματος είναι παράγοντες που αντιμάχονται για την αποδοτικότητα ενός παράλληλου συστήματος. Το καλύτερο ενδεχόμενο είναι για κάθε προσθήκη μίας νέας μονάδας επεξεργασίας να αυξάνεται και το μέγεθος προβλήματος με τέτοιο τρόπο ώστε η αποδοτικότητα του παράλληλου συστήματος να διατηρείται σταθερή. Αυτός ο συλλογισμός γέννησε την έννοια της ισοαποδοτικότητας και για να επιτευχθεί αυτό ένας μηχανικός ή προγραμματιστής χρειάζεται να μελετήσει τη συνάρτηση ισοαποδοτικότητας (Εξίσωση (5.11)).

Ορισμός 3 *Ισοαποδοτικότητα (Isoefficiency) είναι η ιδέα ότι για να κλιμακώσει αποδοτικά ένα παράλληλο σύστημα θα πρέπει να διατηρείται σταθερή η αποδοτικότητα του όσο αυξάνεται το πλήθος των μονάδων επεξεργασίας. Αυτό επιτυγχάνεται με την αύξηση του μεγέθους προβλήματος με τον καταλληλότερο τρόπο.*

5.2.2 Ασθενής κλιμακωσιμότητα

Η έννοια της ασθενούς κλιμακωσιμότητα προκύπτει από το νόμο του Gustafson και την έννοια της ισοαποδοτικότητας. Ενώ προσθέτουμε νέες μονάδες επεξεργασίας για την εκτέλεση ενός προγράμματος, αυξάνουμε το μέγεθος του προβλήματος προκειμένου να διατηρήσουμε σταθερό το φόρτο εργασίας ανά μονάδα επεξεργασίας και κατ΄ επέκταση το χρόνο εκτέλεσης ανά μονάδα επεξεργασίας. Ένα διάγραμμα ασθενούς κλιμακωσιμότητας είναι ένα διάγραμμα επιτάχυνσης όπου για το ίδιο πρόγραμμα απεικονίζονται διαφορετικές γραφικές παραστάσεις των επιταχύνσεων του. Οι διαφορετικές επιταχύνσεις του προγράμματος προκύπτουν από διαφορετικά μεγέθη προβλήματος.



Σχήμα 5.5: Διάγραμμα για διαφορετικές αποκρίσεις ασθενούς κλιμακωσιμότητας

Στο σχήμα 5.5 βλέπουμε τον ίδιο αλγόριθμο να εκτελείται για διαφορετικά μεγέθη προβλήματος. Η μωβ γραμμή απεικονίζει την επιτάχυνση του αλγορίθμου χωρίς καμία αλλαγή στο μέγεθος προβλήματος κατά την αύξηση του πλήθους των μονάδων επεξεργασίας. Η γαλάζια γραμμή απεικονίζει μια μέτρια κλιμάκωση του παράλληλου συστήματος. Σημαίνει ότι το μέγεθος προβλήματος αυξήθηκε αργά συγκριτικά με την αύξηση του πλήθους των μονάδων επεξεργασίας που χρησιμοποιούνται. Η μαύρη γραμμή απεικονίζει μια καλή κλιμάκωση του αλγορίθμου. Το μέγεθος του προβλήματος εισόδου αυξάνεται με καλό ρυθμό ως προς το πλήθος των μονάδων επεξεργασίας.

Από τα διαγράμματα ασθενούς κλιμακωσιμότητας, ένας μηχανικός καταλαβαίνει το αν το παράλληλο πρόγραμμά του υστερεί σε πλήθος πράξεων ως προς το πλήθος των μονάδων επεξεργασίας και επιλέγει να αυξήσει το πλήθος των πράξεων με το ρυθμό που επιστρέφει την καλύτερη επιτάχυνση.

Κεφάλαιο 6

Πρόσθετα Κόστη (**Overheads**)

Από τους νόμους του Amdahl και του Gustafson φάνηκε πως η γραμμική κλιμάκωση $\frac{T_s}{p}$ ενός παράλληλου συστήματος περιορίζεται από πρόσθετες κόστη. Τα κόστη αυτά όπως περιγράφονται από τη συνάρτηση $\mathcal{O}$ της εξίσωσης (5.9) εξαρτώνται από το πλήθος των μονάδων επεξεργασίας p της αρχιτεκτονικής του συστήματος και από το μέρος του προγράμματος που δεν παραλληλοποιείται s. Επειδή το μη-παραλληλοποίησιμο μέρος ενός κώδικα s δεν αποτελεί μέγεθος της αρχιτεκτονικής του συστήματος, θεωρείται ως μαθηματική έννοια που εκφράζει όλα τα είδη για τα πρόσθετα κόστη. Δηλαδή, ο κώδικας που προσθέτει καθυστερήσεις και δεν επιτρέπει τη γραμμική κλιμάκωση του προγράμματος είναι ακολουθιακός [18]. Επομένως, τα πρόσθετα κόστη είναι πολλά και μοντελοποιούνται ως το άθροισμα για κάθε είδους πρόσθετου κόστους [18][19]. Η συνάρτηση $\mathcal{O}$ από την (5.9) υπολογίζεται ως

$$\mathcal{O}(p) = \sum_i O_i(p) \tag{6.1}$$

όπου οι συναρτήσεις O_i είναι οι επιπλέον καθυστερήσεις στην παράλληλη υλοποίηση ενός κώδικα και η καθεμια προκύπτει από διαφορετικό αίτιο.

Τα πρόσθετα κόστη ενός παράλληλου προγράμματος διακρίνονται σε δύο βασικές κατηγορίες· τα χρονικά (temportal) τα οποία επηρεάζουν το χρόνο εκτέλεσης ενός προγράμματος και τα χωρικά (spatial) τα οποία επιδρούν αρνητικά στην κατανάλωση των πόρων της μνήμης [19]. Από την εξίσωση (5.9) το ενδιαφέρον επικεντρώνεται στα χρονικά κόστη. Οι λόγοι για τους οποίους δημιουργούνται είναι,

- **Πρόσβαση Δεδομένων**, όπου το εύρος ζώνης της μνήμης περιορίζει την ταχύτητα πρόσβασης στα δεδομένα που χρειάζεται ένα πρόγραμμα, τα πρωτόκολλα συνοχής των κρυφών μνημών και οι αστοχίες τους, ο ψευδής διαμοιρασμός πόρων (false sharing) και τα απομακρυσμένα δεδομένα εξαιτίας της παράλληλης αρχιτεκτονικής του συστήματος.

- **Συγχρονισμός νημάτων**, όπου χρησιμοποιούνται προγραμματιστικά φράγματα (barriers) τα οποία οργανώνουν το σύνολο των νημάτων για την προστασία της παράλληλης

περιοχής και τα προγραμματιστικά κλειδώματα (locks) για την προστασία κοινόχρηστων δεδομένων μεταξύ των νημάτων.

- **Προγραμματισμός νημάτων**, για την κατάλληλη ανάθεση των παράλληλων διεργασιών σε κάθε νήμα (scheduling).
- **Ανισόρροπη κατανομή φορτίου** (load imbalance), όπου ο διαμοιρασμός των εργασιών στα παράλληλα νήματα γίνεται με άνισο τρόπο.
- Ύπαρξη περιοχών που δεν παραλληλοποιούνται, εντός μιας παράλληλης περιοχής του κώδικα, οι οποίες εκτελούνται μόνο από ένα νήμα.

6.1 Πρόσβαση Δεδομένων

Το εύρος ζώνης (διαφορετικά ρυθμαπόδοση) της κύριας μνήμης ενός συστήματος αποτελεί γενικό αίτιο για την προσθήκη καθυστερήσεων στην εκτέλεση ενός προγράμματος. Είναι το μοναδικό αίτιο που εξετάζεται το οποίο επηρεάζει και τα ακολουθιακά προγράμματα. Καθώς η τεχνολογία των πολυπεξεργαστών βελτιώνεται, η αύξηση της ταχύτητας τους στην επεξεργασία των δεδομένων έχει ξεπεράσει κατά πολύ την ταχύτητα με την οποία οι μονάδες μνήμης μπορούν να ανταποκριθούν στα αιτήματά τους. Δημιουργείται έτσι ένα χάσμα ταχυτήτων το οποίο προσθέτει επιπλέον καθυστερήσεις στην εκτέλεση ενός προγράμματος και το οποίο επιδεινώνει τη συνολική επίδοση ενός υπολογιστικού συστήματος[20]. Η λύση που δόθηκε σε αυτό το πρόβλημα - που στην ξενόγλωσση βιβλιογραφία ονομάζεται ως processor-memory speed gap - ήταν η προσθήκη των κρυφών μνημών στις αρχιτεκτονικές των επεξεργαστών. Οι κρυφές μνήμες είναι μικρότερες σε μέγεθος από την κύρια μνήμη οι οποίες αποθηκεύουν τα καταλληλότερα μπλοκ της προκειμένου να μειωθεί το χάσμα ταχύτητας μεταξύ επεξεργαστή και μνήμης. Τα πιο κατάλληλα μπλοκ περιέχουν τις εντολές και τα δεδομένα ενός προγράμματος τα οποία επαναλαμβάνονται περισσότερο κατά την εκτέλεση του.

Η χρήση των κρυφών μνημών στις αρχιτεκτονικές αν και μειώνει τις καθυστερήσεις που προσθέτει το εύρος ζώνης της κύριας μνήμης δημιουργούν και αυτές επιπλέον κόστη στο χρόνο εκτέλεσης. Προκειμένου, στις κρυφές μνήμες να είναι αποθηκευμένα κάθε χρονική στιγμή τα σωστά μπλοκ μνήμης με τις σωστές τιμές για τα δεδομένα, χρειάζεται να υπάρξει μια πολιτική που να ρυθμίζει τη συνοχή των περιεχομένων τους. Ο επιπλέον αυτός έλεγχος στη λειτουργία τους προσθέτει και αυτός εν τέλει κόστος στην επίδοση ενός συστήματος. Οι επιπλέον αυτές καθυστερήσεις είναι τυχαίες κατά την εκτέλεση ενός προγράμματος.

Ένα φαινόμενο το οποίο σχετίζεται με τις κρυφές μνήμες και το πρωτόκολλο συνοχής τους είναι το ψευδές μόιρασμα πόρων της μνήμης (false sharing). Στην περίπτωση που δύο η περισσότερα νήματα έχουν αποθηκευμένα τα ιδιωτικά τους δεδομένα στο ίδιο μπλοκ μνήμης (cache block ή cache line) και ένα από αυτά αλλάξει τα δεδομένα του, τότε το μπλοκ αυτό σημειώνεται ως τροποποιημένο και εκτελείται το πρωτόκολλο συνοχής μνήμης. Δηλαδή, τα ιδιωτικά δεδομένα των υπόλοιπων νημάτων ανανεώνονται για την τήρηση της συνοχής μεταξύ των κρυφών μνημών ακόμα και αν δεν έχουν υποστεί αλλαγή. Αυτή η επιπλέον ενέργεια προσθέτει καθυστέρηση κατά την εκτέλεση της

παράλληλης περιοχής του κώδικα [21][22]. Για να αποφευχθεί ή να μειωθεί η καθυστέρηση αυτή ένας τρόπος είναι να απομονωθούν τα δεδομένα των νημάτων σε αποκλειστικά δικά τους μπλοκ κρυφής μνήμης· δηλαδή να αυξηθεί η τοπικότητα αναφορών των νημάτων [21][23]. Επίσης, σε παράλληλες αρχιτεκτονικές που σχεδιάζονται με τεχνολογίες του είδους hyper-threading μια λύση είναι να μη χρησιμοποιούνται οι επιπλέον λογικοί πυρήνες που παρέχουν και να λειτουργούν ως αυτοτελείς φυσικοί πυρήνες, δηλαδή ως μονοεπεξεργαστές. Αποφεύγεται έτσι ο διαμοιρασμός των μονάδων αριθμητικής και λογικής επεξεργασίας μεταξύ των νημάτων υλισμικού εντός του ίδιου επεξεργαστή, με αποτέλεσμα να αποφεύγεται και η γειτονική διευθυνσιοδότηση των δεδομένων τους - γίνεται για λόγους βελτίωσης της επίδοσης ενός προγράμματος από τους μεταγλωττιστές - που οδηγεί σε αποθήκευση αυτών στα ίδια μπλοκ κρυφής μνήμης.

Η πρόσβαση δεδομένων σε απομακρυσμένες διευθύνσεις παρατηρούνται στις αρχιτεκτονικές κατανεμημένης μνήμης. Συγκεκριμένα, σε αυτές τις αρχιτεκτονικές ο σημαντικότερος λόγος καθυστερήσεων είναι η προσπέλαση από έναν επεξεργαστή, δεδομένων που δεν είναι αποθηκευμένα στην τοπική του μνήμη αλλά σε μια απομακρυσμένη μνήμη ενός άλλου επεξεργαστή [24]. Στην κατανεμημένη αρχιτεκτονική μοιραζόμενης μνήμης (NUMA), η απόσταση της διαδρομής μεταξύ δύο επεξεργαστών προσδιορίζει την καθυστέρηση που προσθέτει η ενέργεια πρόσβασης σε απομακρυσμένα δεδομένα, ακόμα και αν ο χώρος διευθύνσεων προγραμματιστικά είναι κοινός [25]. Σύμφωνα με το [19] στα παράλληλα συστήματα με αρχιτεκτονικές NUMA οι καθυστερήσεις αυτές για μικρής κλίμακας συστήματα είναι αμελητέες. Στην περίπτωση των συστημάτων αυστηρής κατανεμημένης μνήμης, οι πρόσθετες καθυστερήσεις προκύπτουν από την επικοινωνία μεταξύ των επεξεργαστών στο δίαυλο που τους ενώνει. Οι καθυστερήσεις αυτές επηρεάζονται τόσο από το εύρος ζώνης τους διαύλου επικοινωνίας όσο και από το μέγεθος των μηνυμάτων που ανταλλάσσονται[26].

6.2 Συγχρονισμός Νημάτων

Κατά την εκτέλεση ενός παράλληλου προγράμματος, οι εργασίες που εκτελούνται εντός μιας παράλληλης περιοχής του διαμερίζονται σε νήματα. Κατά την έξοδο από μια παράλληλη περιοχή το κάθε νήμα περιμένει να τελειώσουν και τα υπόλοιπα νήματα τα οποία βρίσκονταν εντός αυτής και τελούν ακόμα μια συγκεκριμένη δουλειά. Αυτό επιτυγχάνεται προγραμματιστικά με τη δημιουργία ενός φράγματος μετά το πέρας της παράλληλης περιοχής στο οποίο τα νήματα αδρανοποιούνται. Η οργανωμένη εκτέλεση της παράλληλης περιοχής, από την αρχικοποίηση των νημάτων μέχρι και την έξοδό τους από αυτήν προσθέτει χρονική καθυστέρηση στο συνολικό χρόνο εκτέλεσης. Επίσης, αν εντός του παράλληλου μέρους του κώδικα υπάρχουν δεδομένα τα οποία είναι κοινόχρηστα, θα πρέπει να ελεγχθεί η σειρά με την οποία τα προσπελάσουν τα νήματα. Αν δεν υπάρχει ο κατάλληλος έλεγχος, δημιουργούνται συνθήκες ανταγωνισμού (race conditions) μεταξύ των νημάτων για το διάβασμα ή το γράψιμο των κοινόχρηστων δεδομένων· οι τιμές τους θα είναι απροσδιόριστες κατά την εκτέλεση και η έξοδος του παράλληλου προγράμματος λάθος. Για το λόγο αυτό θα γίνεται έλεγχος των νημάτων και αυτή η ενέργεια ως αποτέλεσμα δημιουργεί πρόσθετες καθυστερήσεις στο συνολικό χρόνο εκτέλεσης.

Στην προγραμματιστική διεπαφή OpenMP οι δομές που επιβάλουν συγχρονισμό στα νήματα

για την οργανωμένη εκτέλεση τους εντός μιας περιοχής παράλληλων εργασιών (work-sharing) είναι οι επόμενες,

- **omp parallel**, είναι η γενικότερη δομή για την παραλληλοποίηση μιας περιοχής εντός του προγράμματος.
- **omp parallel for**, χρησιμοποιείται για την παραλληλοποίηση επαναληπτικών δομών εντός του κώδικα στην οποία δεν υπάρχουν εξαρτήσεις μεταξύ πιθανών εμφωλευμένων επαναλήψεων.
- **omp barrier**, είναι δομή που χρησιμοποιείται κατά την έξοδο από μιας παράλληλη περιοχή εντός του κώδικα, ως προγραμματιστικό φράγμα για την οργάνωση των νημάτων. Συνήθως, η παράλληλη περιοχή που έχει προηγηθεί δεν επιβάλλει συγχρονισμό νημάτων, με αποτέλεσμα κατά την έξοδό τους να μην περιμένουν την περάτωση των υπόλοιπων. Αυτό επιτυγχάνεται στο OpenMP με την προσθήκη της αναφοράς **nowait** σε μια δομή συγχρονισμού.

Η οργάνωση των νημάτων μπορεί να οριστεί είτε χειροκίνητα από τον χρήστη είτε αυτόματα από το περιβάλλον εκτέλεσης (runtime) του OpenMP. Αξίζει να σημειωθεί πως η κάθε δομή που αναφέρθηκε επιβάλλει διαφορετική καθυστέρηση στο χρόνο εκτέλεσης. Αυτό το σχόλιο αναλύεται εκτενέστερα στο κεφάλαιο των πρόσθετων καθυστερήσεων κατά τη μελέτη του μοντέλου μας.

Οι δομές που επιβάλουν έλεγχο στα νήματα για την πρόσβαση τους σε κοινόχρηστα δεδομένα εντός παράλληλων περιοχών είναι οι επόμενες,

- **omp critical**
- **omp lock**
- **omp atomic**
- **omp reduction**

Και σε αυτή την περίπτωση η κάθε δομή επιβάλλει διαφορετικές καθυστέρησεις στο χρόνο εκτέλεσης μιας παράλληλης περιοχής.

6.3 Προγραμματισμός Νημάτων

Η προγραμματιστική διεπαφή OpenMP προσφέρει τη δυνατότητα ελέγχου του τρόπου με τον οποίο διαμοιράζονται οι επαναλήψεις μιας παράλληλης περιοχής **omp parallel for** στα νήματα. Ο προγραμματιστής αν θεωρήσει πως ένας διαφορετικός διαμοιρασμός θα οδηγήσει σε καλύτερη επίδοση του παράλληλου πρόγραμματος μπορεί να προσθέσει τις επόμενες προτάσεις κατά τη δημιουργία της παράλληλης περιοχής. Οι διαφορετικές αυτές προτάσεις είναι οι εξής,

- **schedule(static, n)**, όπου n είναι το πλήθος επαναλήψεων που διαχειριστεί κάθε νήμα στην παράλληλη περιοχή. Η πρόταση static, σημαίνει ότι ο τρόπος διαχωρισμού των επαναλήψεων στα νήματα της συγκεκριμένης περιοχής θα παραμείνει σταθερός ακόμα και αν το πλήθος των νημάτων σε κάθε εκτέλεσή της είναι διαφορετικό.

- **schedule(dynamic)**, όπου η πρόταση dynamic ενημερώνει το περιβάλλον του OpenMP να επιλέξει τον τρόπο με τον οποίο μοιράζονται οι επαναλήψεις στα νήματα κατά την εκτέλεση της παράλληλης περιοχής. Για διαφορετικό πλήθος επαναλήψεων και νημάτων, ο διαμοιρασμός των επαναλήψεων στα νήματα θα είναι κάθε φορά διαφορετικός. Αυτός ο προγραμματισμός νημάτων χρησιμοποιείται κυρίως όταν οι επαναλήψεις εντός μιας παράλληλης περιοχής δημιουργούν πλεονάζον υπολογιστικό κόστος για μερικά νήματα οδηγώντας σε ανισόρροπη κατανομή φορτίου.

- **schedule(guided, n)**, όπου το πλήθος των επαναλήψεων που εκτελεί το κάθε νήμα μεταβάλλεται δυναμικά ανάλογα με το πλήθος επαναλήψεων και το πλήθος των νημάτων που χρησιμοποιούνται. Αν το n οριστεί τότε το πλήθος επαναλήψεων που διαχειρίζεται κάθε νήμα δεν μπορεί να είναι μικρότερο του n.

Οι τρείς διαφορετικοί τρόποι προγραμματισμού των νημάτων προσθέτουν διαφορετικές καθυστερήσεις στο χρόνο εκτέλεσης μιας παράλληλης περιοχής. Συγκεκριμένα, η πρόταση **schedule(dynamic)** θα προσθέτει το μεγαλύτερο κόστος λόγω των επιπλέον ελέγχων που γίνονται από το περιβάλλον εκτέλεσης του OpenMP.

6.4 Ανισόρροπη Κατανομή Φορτίου (Load Imbalance)

Όταν ο διαμοιρασμός των παράλληλων εργασιών στα νήματα δε γίνεται με ομοιόμορφο τρόπο ως προς το πλήθος των υπολογισμών που εκτελεί το καθένα, τότε θα υπάρχουν νήματα στα σημεία συγχρονισμού τους τα οποία θα παραμείνουν ανενεργά για μεγάλο χρονικό διάστημα εώς ότου τερματίσουν την εκτέλεσή τους και τα υπόλοιπα. Η καθυστέρηση που προστίθεται μπορεί να υπολογιστεί αν θεωρηθεί μια παράλληλη περιοχή στην οποία στη μία περίπτωση παρουσιάζεται το φαινόμενο της ανισόρροπης κατανομής φορτίου και στην άλλη περίπτωση δεν παρουσιάζεται. Μετρώντας τους χρόνους εκτέλεσης και για τις δύο περιπτώσεις, η καθυστέρηση θα ισούται με τη διαφορά των δύο χρόνων. Παρόλο που το φαινόμενο αυτό δεν εντοπίζεται εύκολα υπάρχουν διαφορετικοί τρόποι αντιμετώπισής του, όπως φάνηκε και από την παράγραφο *Προγραμματισμός Νημάτων*.

6.5 Μη-παραλληλοποιήσιμες Περιοχές

Εντός μιας παράλληλης περιοχής μπορεί να υπάρχουν περιοχές που εκτελούνται αποκλειστικά ακολουθιακά, μόνο από ένα νήμα, και δεν σχετίζονται με το συγχρονισμό ή τον προγραμματισμό

νημάτων όπως διατυπώθηκε παραπάνω· Η λογική του προγράμματος απαιτεί την ύπαρξή τους. Όπως εξηγήθηκε τα πρόσθετα κόστη αποτελούν τμήματα του κώδικα που εκτελούνται ακολουθιακά. Επομένως, οι περιοχές αυτές προσθέτουν καθυστερήσεις στο χρόνο εκτέλεσης μιας παράλληλης περιοχής. Η καθυστέρηση ισούται με το χρόνο εκτέλεσης της παράλληλης περιοχής με την ακολουθιακή περιοχή μείον το χρόνο εκτέλεσης της παράλληλης περιοχής χωρίς την σειριακή περιοχή.

Στην προγραμματιστική διεπαφή OpenMP τέτοιες περιοχές δημιουργούνται με τις δομές,

- **omp master**, όπου η ακολουθιακή περιοχή εκτελείται αποκλειστικά από το νήμα που δημιούργησε τα νήματα.
- **omp single**, όπου η περιοχή που δεν παραλληλοποιείται εκτελείται από ένα μόνο νήμα.

Κεφάλαιο 7

Το Κλασικό Μοντέλο **Roofline**

Στην προηγούμενη παράγραφο έγινε αναφορά σε ένα κοινό περιορισμό της επίδοσης των υπολογιστικών συστημάτων, αυτόν της ταχύτητας της μνήμης συγκριτικά με την ταχύτητα των σύγχρονων πολυεπεξεργαστών. Οι νόμοι του Amdahl και Gustafson για τα παράλληλα συστήματα αγνοούν τη ρυθμαπόδοση της μνήμης όταν μοντελοποιούν την επίδοσή τους. Αντιθέτως, το μοντέλο Roofline την περιλαμβάνει. Αποτελεί έτσι μια λύση για τη συνολική εικόνα της επίδοσης ενός παράλληλου συστήματος επεξεργασίας.

Ορισμός 4 *Το μοντέλο Roofline είναι ένα μοντέλο που στηρίζεται στην τεχνική ανάλυσης «κατωφλιού και συμφόρησης» (bound-and-bottleneck)[27]. Απεικονίζει την επίδοση ενός παράλληλου συστήματος με γραφικό τρόπο χρησιμοποιώντας δύο τεμνόμενες ημιευθείες. Αυτές λειτουργούν ως οπτικές ενδείξεις για το πως περιορίζεται το παράλληλο σύστημα από τις μονάδες επεξεργασίας και τη μνήμη του[28].*

7.1 Βασικές Έννοιες

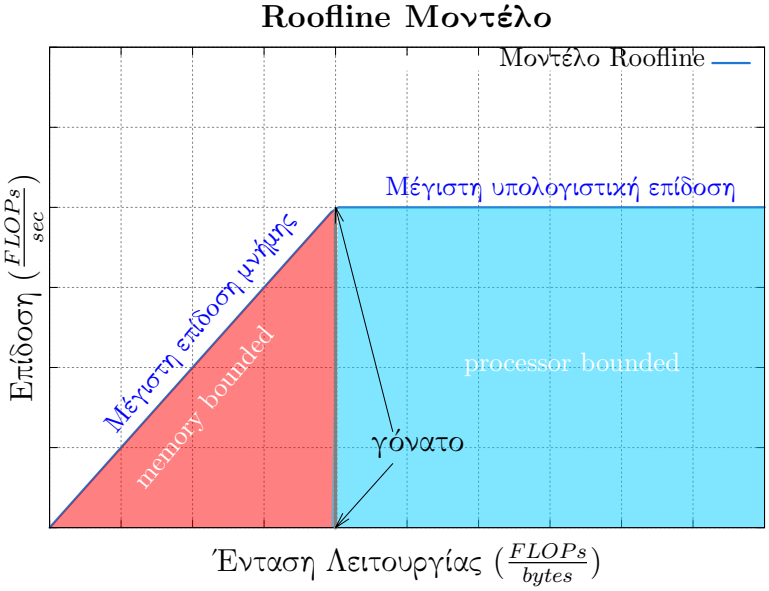
Η επίδοση ενός συστήματος ορίζεται ως ο αριθμός των πράξεων κινητής υποδιαστολής που μπορούν να εκτελεστούν σε ένα δευτερόλεπτο. Η μονάδα μέτρησης της είναι $\frac{FLOPs}{sec}$, όπου FLOPs είναι αρκτικόλεξο από τις αγγλικές λέξεις floating point operations. Στη βιβλιογραφία η μονάδα μέτρησης σημειώνεται και ως FLOPS.

Όπως σχολιάστηκε, κίνητρο για τη σύλληψη του μοντέλου Roofline αποτέλεσε το εύρος ζώνης της κύριας μνήμης ενός συστήματος [28]. Το βασικό πρόβλημα, όπως συζητήθηκε στο κεφάλαιο *Πρόσθετα Κόστη*, είναι ότι η ταχύτητα με την οποία ένας πολυεπεξεργαστής κάνει αίτημα για πρόσβαση σε δεδομένα της μνήμης αυξάνεται με γρήγορους ρυθμούς καθώς εξελίσσεται η τεχνολογία, ενώ η ταχύτητα με την οποία η κύρια μνήμη αποκρίνεται στα αιτήματα αυτά δεν αυξάνεται το ίδιο [29]. Δημιουργείται έτσι ένα χάσμα στις ταχύτητες των δύο μονάδων το οποίο συνεχίζει να μεγαλώνει καθώς βελτιώνεται η κατασκευή των ολοκληρωμένων κυκλωμάτων. Επομένως, η ρυθμαπόδοση της κύριας μνήμης περιορίζει την επίδοση ενός προγράμματος ιδιαίτερα αν για ο περισσότερος χρόνος εκτέλεσής του δαπανάται στην μεταφορά δεδομένων. Δηλαδή, το **εύρος**

ζώνης της κύριας μνήμης αποτελεί τη μετρική για τη συμφόρηση ενός συστήματος στο Roofline μοντέλο. Η μονάδα μέτρησης του είναι $\frac{bytes}{sec}$.

Βασικό μέγεθος για την ανάλυση ενός προγράμματος είναι η **ένταση λειτουργίας** του (operational intensity). Ορίζεται ως το πλήθος των πράξεων κινητής υποδιαστολής που εκτελεί δια το μέγεθος το σύνολο των δεδομένων που μεταφέρθηκαν από και προς την κύρια μνήμη, Σε bytes. Η ένταση λειτουργίας μπορεί να οριστεί γενικότερα ως το πλήθος ενεργειών που εκτέλεσε ένα πρόγραμμα προς το σύνολο των δεδομένων που μεταφέρθηκαν σε bytes, όπου ενέργειες μπορεί να είναι πράξεις κινητής υποδιαστολής, πράξεις ακεραίων κτλ. Διατηρώντας τη σημασιολογία των μεγεθών του κλασικού μοντέλου Roofline η μονάδα μέτρησης της είανι $\frac{FLOPs}{bytes}$.

Το μοντέλο Roofline απεικονίζεται σε ένα ορθοκανονικό σύστημα δύο αξόνων. Ο οριζόντιος άξονας είναι η ένταση λειτουργίας του προγράμματος ενώ ο κατακόρυφος άξονας είναι η επίδοσή του συστήματος. Με αρχή το σημείο (0,0) σχηματίζουμε ημιευθεία με κλίση ίση με το εύρος ζώνης της κύριας μνήμης του συστήματος. Επίσης, σχηματίζουμε οριζόντια ημιευθεία από το 0 του οριζόντιου άξονα με τιμή για την τεταγμένη ίση με τη μέγιστη δυνατή επίδοση του συστήματος όταν αυτό δεν περιορίζεται από το εύρος ζώνης της μνήμης. Το σημείο τομής των δύο ημιευθειών ονομάζεται **γόνατο**. Η επίδοση ενός προγράμματος θα βρίσκεται πάντα κάτω από τις δύο ημιευθείες. Οι δύο ημιευθείες σχηματίζουν οπτικά μια οροφή εξού και το όνομα Roofline.



Σχήμα 7.1: Το μοντέλο Roofline για ένα θεωρητικό υπολογιστικό σύστημα

7.2 Προβλέψεις μέσω Roofline

Η συνολική λειτουργία ενός προγράμματος εκφράζεται στο Roofline μοντέλο ως ένα διατεταγμένο ζεύγος (x, y), όπου x είναι η ένταση λειτουργίας του και y είναι η επίδοσή του. Με βάση το x

γνωρίζουμε ότι ένα πρόγραμμα περιορίζεται από τη μνήμη (**memory bounded**) αν $x <$ γόνατο. Αν $x >$ γόνατο τότε θα περιορίζεται από τη μέγιστη υπολογιστική ικανότητά του συστήματος (**compute bounded**). Με βάση το y ο σχεδιαστής του προγράμματος μπορεί να διαπιστώσει αν δεν είναι επαρκώς βελτιστοποιημένο, συγκρίνοντας την απόσταση της επίδοσής του από τις δύο ημιευθείες.

Η επίδοση που μπορεί να φτάσει ένα πρόγραμμα σύμφωνα με το μοντέλο Roofline προκύπτει μέσω του τύπου,

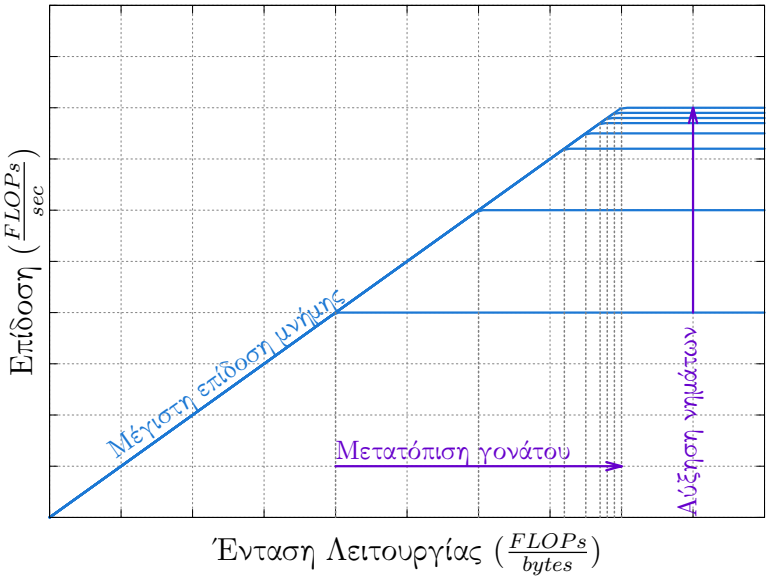
$$\text{AttainableFLOPS} = min(\text{MaximumFLOPS}, \text{Bandwidth} \times \text{OperationalIntensity}) \tag{7.1}$$

όπου AttainableFLOPS είναι η εφικτή επίδοση που μπορεί να φτάσει το πρόγραμμα αν βελτιστοποιηθεί επαρκώς, MaximumFLOPS είναι η μέγιστη υπολογιστική επίδοση του παράλληλου συστήματος, Bandwidth είναι το εύρος ζώνης της κύριας μνήμης του και OperationalIntensity είναι η ένταση λειτουργίας του προγράμματος.

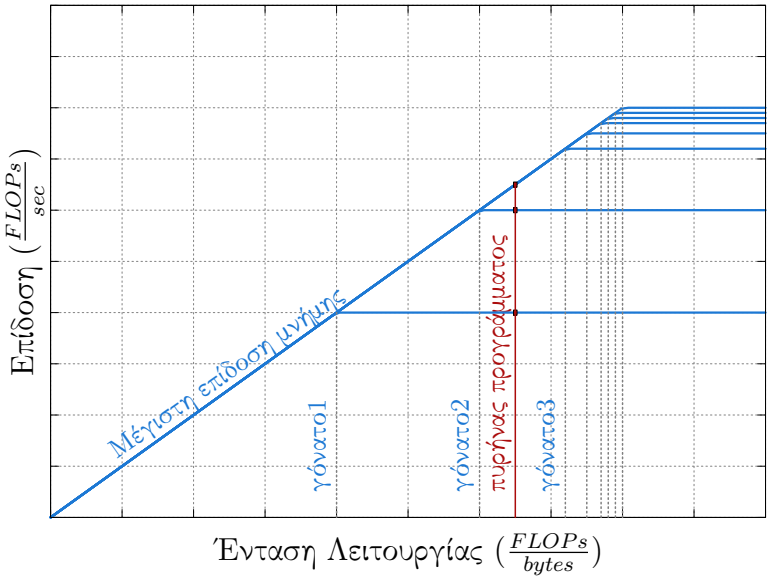
7.3 Σχέση με τους νόμους Amdahl και Gustafson

Το μοντέλο Roofline παρόλο που είναι απλό, διαισθητικά ενσωματώνει τα γενικά συμπεράσματα των νόμων του Amdahl και Gustafson.

Όσον αφορά το νόμο του Amdahl μπορούμε να το σκεφτούμε ως εξής. Ενόσω το πλήθος μονάδων επεξεργασίας ενός παράλληλου συστήματος αυξάνεται, στο μοντέλο Roofline θα μετατοπίζεται η οριζόντια ημιευθεία, που είναι η μέγιστη υπολογιστική επίδοσή του, προς τα πάνω. Καθώς αυξάνεται το οριζόντιο κατώφλι, το γόνατο του μοντέλου μετατοπίζεται όλο και δεξιότερα σε μεγαλύτερες εντάσεις λειτουργίας. Η ένταση λειτουργίας ενός προγράμματος θα φαίνεται ότι μετατοπίζεται αριστερότερα και θα κατευθύνεται προς την περιοχή που περιορίζεται από το μέγιστο εύρος ζώνης της κύριας μνήμης. Αυτό σημαίνει πως το πρόγραμμα θα περιορίζεται όλο και περισσότερο από το s, την περιοχή του δηλαδή που δεν παραλληλοποιείται. Μοντελοποιείται έτσι το συμπέρασμα του νόμου του Amdahl πως όσο αυξάνονται οι μονάδες επεξεργασίας που χρησιμοποιούνται σε ένα παράλληλο σύστημα, η επίδοση ενός προγράμματος οδηγείται κάποια στιγμή σε κορεσμό, από το αντίστροφο μέρος της περιοχής που δεν παραλληλοποιείται.



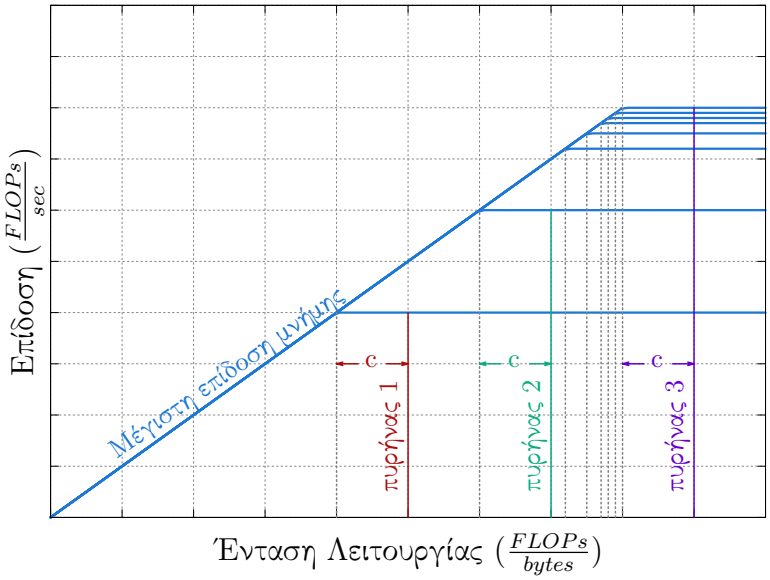
Σχήμα 7.2: Όσο αυξάνεται το πλήθος των νημάτων του παράλληλου συστήματος τόσο μετατοπίζεται προς τα δεξιά το γόνατο του Roofline μοντέλου.



Σχήμα 7.3: Η ένταση λειτουργίας του πυρήνα είναι μεγαλύτερη από τα γόνατα 1 και 2, διατηρώντας τον πυρήνα στην περιοχή που περιορίζεται από τη μέγιστη υπολογιστική επίδοση. Το σύστημα δηλαδή κλιμακώνει καλά κατά τη μετάβασή του από την οροφή με γόνατο το 1 στην οροφή με γόνατο το 2. Εν αντιθέσει, κατά την οροφή με γόνατο 3 το σύστημα βρίσκεται στην περιοχή που περιορίζεται από την μέγιστη επίδοση μνήμης. Αυτό δείχνει πως το σύστημα δεν κλιμακώνει καλά από το 1 στο 3 αποδεικνύοντας οπτικά πως ο νόμος του Amdahl ισχύει. Δηλαδή ότι καθώς αυξάνεται το πλήθος των νημάτων το σύστημα δεν κλιμακώνει καλά.

Ο νόμος του Gustafson ενσωματώνεται στην έννοια της έντασης λειτουργίας ενός προγράμματος. Συγκεκριμένα, ο διαιρέτης της έντασης λειτουργίας, δηλαδή το πλήθος των πράξεων κινητής υποδιαστολής, αντιστοιχίζεται με την έννοια του μεγέθους προβλήματος του νόμου του Gustafson. Αν αυξηθεί το πλήθος των πράξεων διατηρώντας σταθερό το πλήθος προσβάσεων στην κύρια μνήμη, τότε η ένταση λειτουργίας ενός προγράμματος θα μεγαλώνει όλο και περισσότερο, μετατοπίζοντας το δεξιότερα στο διάγραμμα, στην περιοχή που περιορίζεται από την υπολογιστική επίδοση του συστήματος. Ουσιαστικά, σύμφωνα και με όσα αναφέραμε για την απόδοση του νόμου του Amdahl στο μοντέλο Roofline, ο νόμος του Gustafson προτείνει να αυξηθεί το πλήθος των πράξεων κινητής υποδιαστολής ενός προγράμματος όσο αυξάνεται το πλήθος των μονάδων επεξεργασίας, προκειμένου το παράλληλο σύστημα πάντα να κλιμακώνει.

Ο νόμος της ισοαποδοτικότητας αποδίδεται στο μοντέλο Roofline με την αύξηση του πλήθους των πράξεων κινητής υποδιαστολής ενός προγράμματος σε τέτοιο βαθμό ώστε η διαφορά της νέας έντασης λειτουργίας του από το νέο γόνατο να είναι ίση με τη διαφορά που είχε πριν την αύξηση των πράξεων και των μονάδων επεξεργασίας. Μαθηματικά εκφράζεται με τον τύπο, $\forall p \in \mathbb{N} : \text{OperationalIntensity}_p - \text{γόνατο}_p = c$, όπου c είναι σταθερά.



Σχήμα 7.4: Οι πυρήνες 2 και 3 έχουν προκύψει από τον πυρήνα 1. Η διαφορά είναι πως έχουν προστεθεί σε αυτούς επιπλέον πράξεις κινητής υποδιαστολής. Βλέπουμε ότι καθώς αυξάνεται το πλήθος των νημάτων αν αυξήσουμε το πλήθος των πράξεων κινητής υποδιαστολής, δηλαδή το μέγεθος προβλήματος, ο πυρήνας 1 κλιμακώνει καλά καθώς διατηρείται ως processor bounded στα Roofline μοντέλα. Επίσης, αν η διαφορά από το κάθε νέο γόντατο διατηρηθεί σταθερή ικανοποιούνται οι συνθήκες της ισοαποδοτικότητας.

7.4 Μέτρηση Κατωφλιών και Μετρητές Επίδοσης

Για την κατασκευή του μοντέλου Roofline για ένα παράλληλο σύστημα απαιτείται να γνωρίζουμε τις τιμές για τα δύο κατώφλια, της μέγιστης υπολογιστικής επίδοσης του και του μέγιστου εύρους ζώνης της κύριας μνήμης του. Για τον σκοπό αυτό χρησιμοποιούνται είτε μετροπρογράμματα για τον υπολογισμό των τιμών είτε οι τιμές που προτείνονται σε εγχειρίδια οδηγιών των κατασκευαστών [30][31][32][33]. Επίσης, υπάρχουν και πιο θεωρητικές προσεγγίσεις με μαθηματικές εκφράσεις για αυτές τις τιμές χρησιμοποιώντας ως ορίσματα μεγέθη, τα οποία βρίσκονται επίσης στα εγχειρίδια των κατασκευαστών.

7.4.1 Μέγιστη υπολογιστική επίδοση

Για τον υπολογισμό της μέγιστης υπολογιστικής επίδοσης υπάρχει μαθηματική εξίσωση η οποία χρησιμοποιεί χαρακτηριστικά τόσο της μικροαρχιτεκτονικής των πολυεπεξεργαστών όσο και της αρχιτεκτονικής του συνολικού παράλληλου συστήματος[34]. Συγκεκριμένα,

$$\frac{FLOPS}{node} = \frac{FLOP}{operation} \times \frac{operations}{instruction} \times \frac{instructions}{cycle} \times \frac{cycles}{second} \times \frac{cores}{socket} \times \frac{sockets}{node} \quad (7.2)$$

όπου,

1. $\frac{FLOP}{operation}$: το πλήθος των πράξεων κινητής υποδιαστολής που περιέχονται σε μια αριθμητική πράξη. Για παράδειγμα στην πράξη της διαίρεσης, αν και σημασιολογικά είναι μια μαθηματική πράξη, υπολογιστικά περιέχει περισσότερες. Επίσης, πολλές νεότερες αρχιτεκτονικές περιέχουν υλισμικό για την πράξη αθροίσματος-πολλαπλασιασμού ως μια υπολογιστική μονάδα (FMA, fused multiply-add). Όλες οι πράξεις αντιμετωπίζονται ως λόγος 1/1 ενώ οι FMA πράξεις ως 2/1.

2. $\frac{operations}{instruction}$: σύνολο πράξεων που εκτελούνται ανά εντολή. Αν το σύστημα υποστηρίζει για παράδειγμα εντολές SIMD εκτελούνται 4 πράξεις για 1 εντολή, δηλαδή λόγος 4/1. Άλλα παράδειγμα αποτελούν οι εντολές AVX-256, AVX-512. Οι εντολές αυτές αποκαλούνται διανυσματικές.

3. $\frac{instructions}{cycle}$: πόσες πράξεις εκτελούνται σε έναν κύκλο. Το μέγεθος αυτό εκφράζει αν ο επεξεργαστής είναι υπερβαθμωτός (superscalar).

4. $\frac{cycles}{sec}$: θεωρείται ως η συχνότητα λειτουργίας του επεξεργαστή. Η συχνότητα αυτή στην πράξη αλλάζει με το χρόνο και το φόρτο του επεξεργαστή.

5. $\frac{cores}{socket}$: πόσοι πυρήνες υπάρχουν σε ένα επεξεργαστή. Υπολογίζονται επίσης και υλισμικά νήματα (hyperthreading, hardware multithreading).

6. $\frac{sockets}{node}$: πόσοι επεξεργαστές υπάρχουν ανά υπολογιστή. Εδώ ο κόμβος (node) απευθύνεται σε έναν υπολογιστή, ο οποίος μπορεί να είναι συνδεδεμένος σε ένα δίκτυο υπολογιστών.

Σε αυτό τον μαθηματικό τύπο γίνονται πολλές εξιδανικεύσεις, διότι οι τιμές για πολλά από τα μεγέθη του μεταβάλλονται κατά την λειτουργία του παράλληλου συστήματος. Για πιο ρεαλιστικές τιμές της μέγιστης υπολογιστικής επίδοσης χρησιμοποιούνται μετροπρογράμματα, που υπολογίζουν τα μέγιστα FLOPS αξιοποιώντας ταυτόχρονα τις βελτιστοποιήσεις των μεταγλωττιστών και τις ταχύτερες εντολές assembly. Μερικά από τα γνωστά μετροπρογράμματα που χρησιμοποιούνται στη βιομηχανία είναι τα επόμενα,

1. **Linpack**[35] [36]
2. **High Performance Linpack (HPL)**[37] [38]
3. **DGEMM**[39] [38]

7.4.2 Μέγιστο εύρος ζώνης κύριας μνήμης

Για τον θεωρητικό υπολογισμό του μέγιστου εύρους ζώνης της κύριας μνήμης υπάρχει ο επόμενος μαθηματικός τύπος,

$$\text{MaxBandWidth} = \text{BaseMemoryFrequency} \times \text{DataRate} \times \text{BusWidth} \times \text{Channels} \quad (7.3)$$

όπου,

1. BaseMemoryFrequency: η συχνότητα βασικής λειτουργίας της μνήμης χωρίς την εμπλοκή τεχνολογιών όπως διπλασιασμός ρυθμού μετάδοσης δεδομένων (double data rate - DDR).

2. DataRate: ρυθμός μετάδοσης δεδομένων. Για παράδειγμα αν η τεχνολογία μνήμης είναι της οικογένειας DDR τότε το DataRate είναι 2.

3. BusWidth: το πλάτος ενδιάμεσης μνήμης ονομάζεται και ως πλάτος γραμμής (line width) και πλάτος μπλοκ, και ισούται με το πλήθος των bytes που μεταφέρονται. Συνήθως, είναι 8 bytes.

4. Channels: είναι το πλήθος καναλιών που μεταφέρουν δεδομένα. Για παράδειγμα στην τεχνολογία DDR υπάρχουν αριθμοί που ακολουθούν την ονομασία και συμβολίζουν το πλήθος καναλιών. Έτσι, αν έχουμε κύρια μνήμη DDR4 τότε σημαίνει πως υπάρχουν 4 κανάλια.

Όπως και για τον υπολογισμό της μέγιστης υπολογιστικής επίδοσης ενός συστήματος, στη βιομηχανία υπάρχουν συγκεκριμένα μετροπρογράμματα τα οποία μετρούν το μέγιστο εύρος ζώνης της κύριας μνήμης τους. Μερικά από αυτά είναι,

1. **STREAM**[40]

2. **High-performance conjugate-gradient (HPCG)**[41]

Έχοντας πλέον τις τιμές για τα δύο κατώφλια μπορούμε να σχεδιάσουμε τις δύο ημιευθείες όπως συζητήθηκε προηγουμένως και να υπολογίσουμε το γόνατο του Roofline μοντέλου. Το γόνατο στη συνέχεια χρησιμοποιείται ως σημείο αναφοράς για την ένταση λειτουργίας ενός προγράμματος για την επίδοσή του και τον χαρακτηρισμό της λειτουργίας του.

7.4.3 Ένταση λειτουργίας προγράμματος

Η ένταση λειτουργίας ενός προγράμματος μπορεί να υπολογιστεί και θεωρητικά και πρακτικά. Θεωρητικά, αν η περιοχή ενδιαφέροντος εντός του κώδικα, είναι καλά δομημένη μπορεί κανείς να μετρήσει τις πράξεις που εκτελούνται σε αυτή και τα bytes που μεταφέρονται, με απλή μελέτη της.

Έστω ένας επαναληπτικός βρόγχος δύο επιπέδων με N_1 επαναλήψεις για τον εξωτερικό βρόγχο και N_2 επαναλήψεις για τον εσωτερικό βρόγχο. Εντός των βρόχων εκτελούνται τρεις πράξεις πρόσθεσης σε αριθμούς κινητής υποδιαστολής. Ένας πιθανός ψευδοκώδικας που περιγράφει αυτό το σενάριο είναι ο επόμενος,

Algorithm 1: Παράδειγμα ενός πυρήνα για τον υπολογισμό της έντασης λειτουργίας του

```
...;
for i = 0 to N_1 do
    for j = 0 to N_2 do
        a[i][j] ← b[i − 1][j] + c[i + 1][j − 1] + d[i][j + 1] + e[i − 1][j];
    end
end
...;
```

Συνολικά, οι πράξεις που εκτελούνται είναι $N_1 \times N_2 \times 3 = 3N_1N_2$. Το σύνολο δεδομένων που μεταφέρεται, αν θεωρήσουμε ότι οι πίνακες a, b, c, d και e είναι αρκετά μεγάλοι για να μη χωρέσουν στην μεγαλύτερη κρυφή μνήμη Last Level Cace (LLC), είναι $N_1 \times N_2 \times 5 \times 8 = 40N_1N_2$. Η ένταση λειτουργίας του πυρήνα αυτού επομένως είναι,

$$OperationalIntensity = \frac{3N_1N_2}{40N_1N_2} = \frac{3}{40} = 0.075$$

Χρησιμοποιώντας τώρα την ένταση λειτουργίας και το γόνατο μπορούμε να σχολιάσουμε αν ο πυρήνας έχει ως κατώφλι τη μέγιστη επίδοση της μνήμης ή τη μέγιστη υπολογιστική επίδοση του συστήματος. Ανάλογα, σε ποιά περιοχή από τις δύο βρίσκεται μπορούν να διατυπωθούν οι κατάλληλες βελτιστοποιήσεις για την περιοχή του κώδικα.

Στη συγκεκριμένη περίπτωση ο πυρήνας του προγράμματος έχει εύκολη δομή για μια θεωρητική μελέτη της έντασης λειτουργίας του. Στην περίπτωση όμως που ήταν πιο περίπλοκος, είναι πολύ δύσκολο κανείς να την υπολογίσει με το χέρι. Επίσης, στη θεωρητική μας μελέτη υποθέσαμε αυθαίρετα πως οι πίνακες είναι μεγαλύτεροι σε μέγεθος από το μέγεθος της μεγαλύτερης κρυφής μνήμης του πολυεπεξεργαστή. Είναι απαραίτητος επομένως ένας πιο πρακτικός τρόπος για τον υπολογισμό της έντασης λειτουργίας ενός προγράμματος. Για το λόγο αυτό οι σχεδιαστές επεξεργαστών συμπεριλαμβάνουν στο ολοκληρωμένο κύκλωμα υλικό ειδικού σκοπού. Οι μονάδες αυτού του υλικού ονομάζονται **μετρητές επίδοσης (performance counters)**.

Ορισμός 5 *Μετρητής Επίδοσης είναι ένας καταχωρητής επεξεργαστή, ειδικού σκοπού, που είτε έχει προσχεδιαστεί από το εργοστάσιο να καταγράφει ένα γεγονός που γίνεται εντός της μονάδας επεξεργασίας είτε είναι προγραμματίσιμος για να καταγράψει ένα μέγεθος του επεξεργαστή που ενδιαφέρει τον χρήστη.*

Οι μετρητές επίδοσης καταγράφουν διάφορα γεγονότα και σήματα που γίνονται εντός του επεξεργαστή κατά την εκτέλεση ενός προγράμματος. Μερικά παραδείγματα είναι η καταγραφή των αστοχιών της κάθε κρυφής μνήμης στο γράψιμο ή διάβασμα, της ισχύος που καταναλώνεται ανά εντολή και του πλήθους των διανυσματικών εντολών που εκτελέστηκαν. Σε αυτά τα γεγονότα καταγράφεται και το πλήθος των πράξεων κινητής υποδιαστολής και σύνολο των bytes που μεταφέρθηκαν από και προς την κύρια μνήμη του συστήματος. Διαδεδομένα εργαλεία που λειτουργούν ως διεπαφές για την πρόσβαση των μετρητών επίδοσης ενός επεξεργαστή είναι,

1. **Βιβλιοθήκη PAPI**[42][43]
2. **Πρόγραμμα perf**[44]

Τα προγράμματα αυτά έχουν συνήθως μια βάση δεδομένων με τους μετρητές επίδοσης της κάθε μικροαρχιτεκτονικής. Εναλλακτικά, οι μετρητές αναφέρονται και στα εγχειρίδια παραγωγής λογισμικού (Software Development Manual) του κάθε κατασκευαστή.

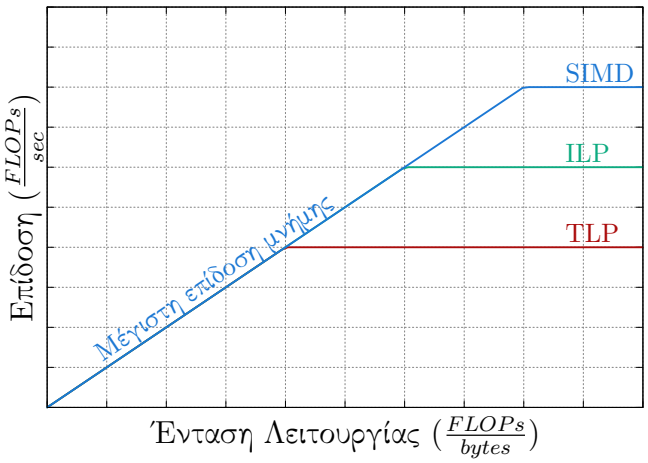
7.5 Επεκτάσεις του κλασικού μοντέλου

Στο κλασικό μοντέλο Roofline έχουν γίνει διάφορες υποθέσεις για το υλικό του υπολογιστή. Ως εύρος ζώνης μνήμης θεωρείται η ρυθμαπόδοση της κύριας μνήμης, ενώ ως μέγιστο υπολογιστικό κατώφλι θεωρείται η επίδοση του συστήματος με όλες τις δυνατές βελτιστοποιήσεις που προσφέρει σε ένα πρόγραμμα. Προκειμένου η ανάλυση ενός συστήματος να γίνει πιο διαισθητική για τον σχεδιαστή προγράμματος, γίνεται να προστεθούν επιπλέον κατώφλια και στη μέγιστη υπολογιστική επίδοση και στο μέγιστο εύρος ζώνης της μνήμης[45].

Η μέγιστη υπολογιστική επίδοση μπορεί να διαχωριστεί στα επόμενα οριζόντια κατώφλια,

1. Παραλληλισμός σε Επίπεδο Νημάτων (Threads Level Parallelism (TLP))
2. Παραλληλισμός σε Επίπεδο Εντολών (Instruction Level Parallelism (ILP))
3. Διανυσματικοποίηση Εντολών (Single Instruction Multiple Data (SIMD) και AVX καταχωρητές)

Roofline Μοντέλο Με Επιπλέον Υπολογιστικά Κατώφλια



Σχήμα 7.5: Το μοντέλο Roofline με επιπλέον κατώφλια μέγιστης υπολογιστικής επίδοσης.

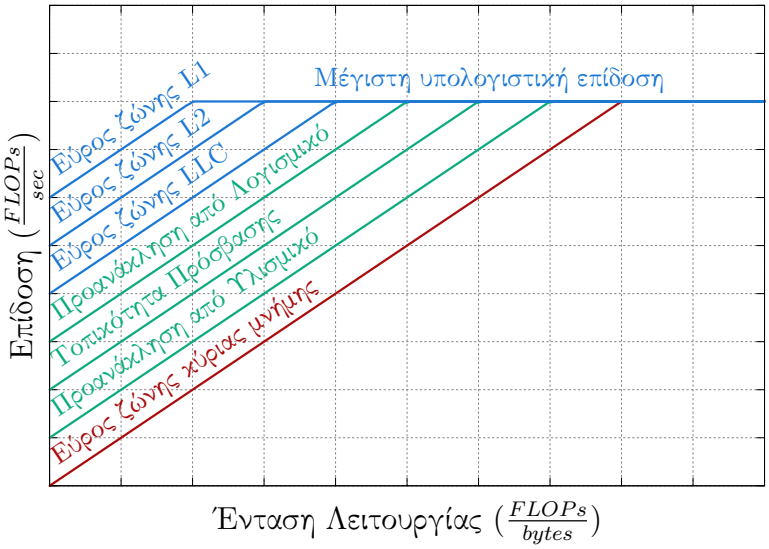
Η μέγιστη επίδοση μνήμης μπορεί να διαχωριστεί στα επόμενα κατώφλια,

1. Προανάκληση από Υλισμικό (Hardware Prefetching), με συγκεκικριμένες μονάδες υλισμικού για την πρόσβαση σε διευθύνσεις της μνήμης που γειτνιάζουν με την παρούσα πριν την ανάκλησή τους.
2. Τοπικότητα πρόσβασης δεδομένων σε συστήματα με αρχιτεκτονικές απομακρυσμένης μνήμης (για παράδειγμα NUMA)

3. Προανάκληση από Λογισμικό, όπου μεταγλωττιστές ανασκευάζουν τον κώδικα για συντομότερη πρόσβαση στα δεδομένα.

4. Πολλαπλά εύρη ζώνης για την Ιεραρχεία Μνήμης, όπως είναι η κύρια μνήμη και οι κρυφές μνήμες διαφόρων επιπέδων της ιεραρχείας [46].

Roofline Μοντέλο Με Επιπλέον Κατώφλια Μνήμης



Σχήμα 7.6: Το μοντέλο Roofline με επιπλέον κατώφλια εύρη ζώνης για το σύστημα μνήμης.

7.6 Οι περιορισμοί στις προβλέψεις

Το Roofline μοντέλο όπως έχει αναφερθεί ανήκει στην οικογένεια των bounded-and-bottleneck μοντέλων. Δεν ενδιαφέρεται για την εξαγωγή μιας ακριβούς πρόβλεψης της επίδοσης ενός συστήματος αλλά για τα ανώτατα εφικτά όρια του, όπως προκύπτει και από την εξίσωση (7.1). Το μοναδικό κόστος το οποίο μοντελοποιείται είναι οι καθυστερήσεις που προστίθενται στο χρόνο εκτέλεσης ενός προγράμματος εξαιτίας του εύρους ζώνης της κύριας μνήμης. Τα υπόλοιπα κόστη που παρουσιάζονται στα παράλληλα προγράμματα δεν προσμετρώνται και είναι δουλειά του σχεδιαστή προγράμματος να καταλάβει ποιά από αυτά περιορίζουν την επίδοση του κώδικα.

Ένας άλλος περιορισμός του μοντέλου Roofline είναι ότι ο σειριακός χρόνος εκτέλεσης του προγράμματος δεν εμπλέκεται. Ο μοναδικός χαρακτηρισμός ενός προγράμματος γίνεται μέσω της έντασης λειτουργίας. Η ένταση λειουργίας όμως δεν μπορεί να περιγράψει το πόσο βελτιστοποιημένη είναι η σειριακή εκδοχή του προγράμματος, διότι ορίζεται αποκλειστικά με το πλήθος των πράξεων και το σύνολο δεδομένων. Με αυτή την απλοποίηση, χάνεται η πληροφορία της επιτάχυνσης ενός προγράμματος, σε αντίθεση με τους νόμους του Amdahl και Gustafson.

Αν θεωρήσουμε ως ένταση λειτουργίας ενός προγράμματος το OI, η πρόβλεψη για τον χρόνο εκτέλεσης της παράλληλης εκδοχής του στο μοντέλο Roofline υπολογίζεται μέσω της επόμενης μαθηματικής εξίσωσης,

$$T = \mathbb{1}_{[0,\text{γόνατο})}(\text{OI}) \times \frac{\text{bytes}}{\text{bandwidth}} + \mathbb{1}_{[\text{γόνατο},\infty)}(\text{OI}) \times \frac{\text{FLOPs}}{\text{MaximumFLOPS}} \tag{7.4}$$

όπου bytes είναι το σύνολο των δεδομένων που μεταφέρονται από και προς τη μνήμη του συστήματος κατά την εκτέλεση του προγράμματος, FLOPs είναι το πλήθος των πράξεων που εκτελούνται στο πρόγραμμα και MaximumFLOPS είναι η μέγιστη υπολογιστική επίδοση του συστήματος.

Κεφάλαιο 8

Το μοντέλο μας

Με βάση όλα όσα έχουν διατυπωθεί, το μοντέλο μας αποτελεί μια εκδοχή του μοντέλου Roofline, τροποποιημένη έτσι ώστε να δίνει περισσότερο βάση στη λειτουργία ενός προγράμματος και στα κόστη τα οποία προσθέτει η παραλληλοποίησή του. Σκοπός του είναι, δοθέντος ενός προγράμματος με σειριακό χρόνο εκτέλεσης T_s και μιας αρχιτεκτονικής μοιραζόμενης μνήμης, με εύρος ζώνης Bandwidth και πλήθος φυσικών πυρήνων p να προβλέψει με μικρό λάθος το χρόνο εκτέλεσης της παράλληλης εκδοχής του και κατά συνέπεια και την επίδοσή του.

8.1 Οι υποθέσεις του μοντέλου

Για τη δημιουργία του μοντέλου μας υιοθετήσαμε κάποιες βασικές υποθέσεις, τις οποίες χρησιμοποιούμε καθ΄ όλη τη διάρκεια κατασκευής του. Στο τέλος θα αμφισβητήσουμε αυτές τις υποθέσεις και θα παρουσιάσουμε πόσο καλά το μοντέλο μας μπορεί να ανταποκριθεί στις νέες συνθήκες. Οι υποθέσεις έχουν ως εξής,

1. Το μέγεθος εισόδου ενός προγράμματος θα είναι αρκετά μεγάλο προκειμένου να αποφύγουμε την αποθήκευσή μέρους της στην κρυφή μνήμη τελευταίου επιπέδου (LLC) του επεξεργαστή κατά την εκτέλεση του προγράμματος. Με αυτό τον τρόπο διασφαλίζουμε το πρόγραμμά μας να μην εξαρτάται από το εύρος ζώνης της κρυφής μνήμης τελευταίου επιπέδου (LLC) αλλά από το εύρος ζώνης της κύριας μνήμης DRAM.

2. Δε χρησιμοποιούμε νήματα υλισμικού (harware threads, τεχνολογία hyperthreading) για λόγους απλοποίησης και αποφυγής τυχαίων σφαλμάτων. Όταν αναφερόμαστε σε πυρήνα επεξεργαστή, αναφερόμαστε μόνο σε έναν φυσικό πυρήνα. Δηλαδή, αποφεύγουμε τη χρήση λογικών πυρήνων. Για να το πετύχουμε αυτό χρειάζεται να μάθουμε την αρχιτεκτονική σχεδίαση του επεξεργαστή που χρησιμοποιούμε και να περιοριστούμε κατά την εκτέλεση στους φυσικούς πυρήνες. Αποφεύγουμε έτσι το διαμοιρασμό πόρων στα νήματα υλισμικού και την πιθανότητα εμφάνισης του φαινομένου false sharing.

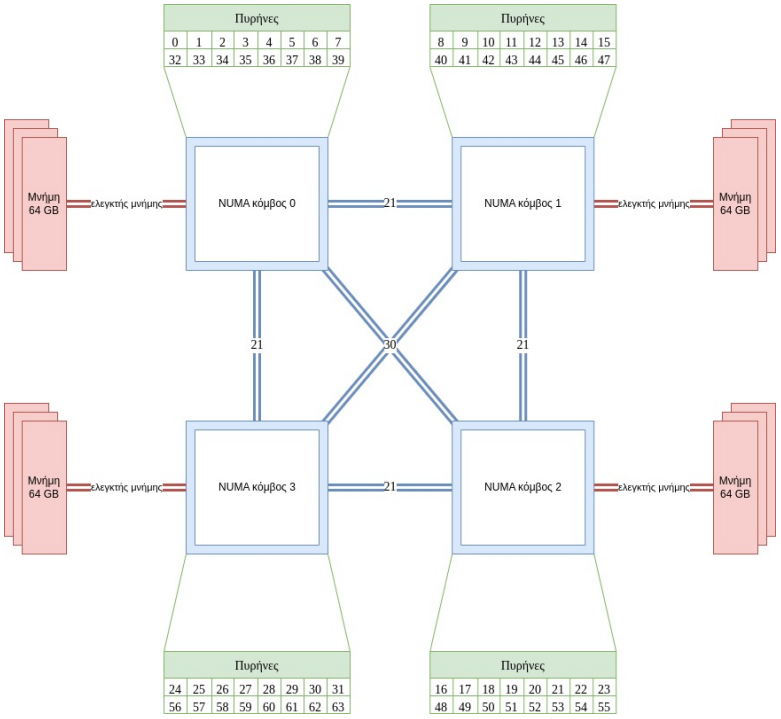
3. Παρόλο που στις αρχιτεκτονικές μοιραζόμενης μνήμης, η συνολική μνήμη φαίνεται ως ενιαία από τον προγραμματιστή, περιοριζόμαστε στις διευθύνσεις μνήμης που αφορούν τον επεξεργαστή στους πυρήνες του οποίου εκτελείται το πρόγραμμα. Για παράδειγμα αν έχουμε μια αρχιτεκτονική με δύο επεξεργαστές και δύο ελεγκτές μνήμης (memory controllers) σε καθ' ένα από αυτούς, και χρησιμοποιούμε τους πυρήνες τους δεύτερου επεξεργαστή, τότε τα δεδομένα μας θα πρέπει να είναι αποθηκευμένα στις διευθύνσεις που αποκωδικοποιεί ο ελεγκτής μνήμης του δεύτερου επεξεργαστή. Διατηρούμε έτσι την τοπικότητα δεδομένων.

4. Το μοντέλο μας δεν περιλαμβάνει τις καθυστερήσεις λόγω ανισόρροπης κατανομής φορτίου, διότι η μελέτη αυτού του φαινομένου θεωρούμε ότι ξεφεύγει από το περιεχόμενο της διπλωματικής. Θεωρείται κυρίως, ότι αποτελεί ευθύνη του προγραμματιστή να σχεδιάσει ένα παράλληλο πρόγραμμα στο οποίο δεν εμφανίζεται αυτό το φαινόμενο. Στην ενότητα *Πρόσθετα Κόστη* σχολιάστηκαν τρόποι ελαχιστοποίησης της επίδρασής του.

8.2 Τα μηχανήματα του εργαστηρίου

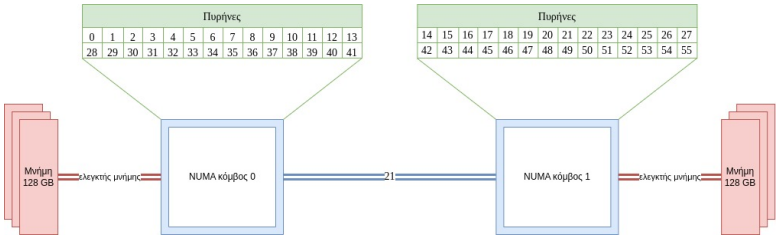
Για την εκτέλεση πειραμάτων και την εξακρίβωση των αποτελεσμάτων χρησιμοποιήθηκαν δύο μηχανήματα του εργαστηρίου. Και τα δύο μηχανήματα στηρίζονται στην αρχιτεκτονική μοιραζόμενης μνήμης. Στους δύο επόμενους πίνακες παρουσιάζονται τα βασικότερα χαρακτηριστικά τους, τα περισσότερα από τα οποία χρησιμοποιούνται για την εκτέλεση πειραμάτων και την κατασκευή του μοντέλου μας. Επίσης, παρουσιάζεται για κάθε μηχάνημα και ένα αφαιρετικό διάγραμμα της συνδεσμολογίας επεξεργαστών-μνήμης.

Μηχάνημα 1	
Μοντέλο επεξεργαστών	Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz
Μικροαρχιτεκτονική πυρήνων	Sandy Bridge-EP
Σύνολο επεξεργαστών	4
Φυσικοί πυρήνες ανά επεξεργαστή	8
Λογικοί πυρήνες ανά επεξεργαστή	16
Σύνολο πυρήνων	64
Hyperthreading	ναι
NUMA αρχιτεκτονική	ναι
Μέγεθος κύριας μνήμης	$4 \times 64GB = 256GB$
Μέγεθος L1 κρυφής μνήμης	$32KB$
Μέγεθος L2 κρυφής μνήμης	$256KB$
Μέγεθος L3 κρυφής μνήμης	$16MB$



Σχήμα 8.1: Η αρχιτεκτονική κοινής μνήμης για το Μηχάνημα 1. Κάθε επεξεργαστής είναι ένας κόμβος NUMA, στον οποίο αντιστοιχίζονται πυρήνες όπως φαίνεται με τους αντίστοιχους δείκτες. Κάθε κόμβος NUMA ελέγχει 64GB διευθύνσεων της συνολικής μνήμης.

Μηχάνημα 2	
Μοντέλο επεξεργαστών	Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz
Μικροαρχιτεκτονική πυρήνων	Skylake-SP
Σύνολο επεξεργαστών	2
Φυσικοί πυρήνες ανά επεξεργαστή	14
Λογικοί πυρήνες ανά επεξεργαστή	28
Σύνολο πυρήνων	56
Hyperthreading	ναι
NUMA αρχιτεκτονική	ναι
Μέγεθος κύριας μνήμης	$2 \times 128GB = 256\ GB$
Μέγεθος L1 κρυφής μνήμης	$32KB$
Μέγεθος L2 κρυφής μνήμης	$1MB$
Μέγεθος L3 κρυφής μνήμης	$\approx 20MB$



Σχήμα 8.2: Η αρχιτεκτονική κοινής μνήμης για το Μηχάνημα 2. Κάθε επεξεργαστής είναι ένας κόμβος NUMA ο οποίος ελέγχει 128GB διευθύνσεων της συνολικής μνήμης.

8.3 Ποια μεγέθη μας ενδιαφέρουν και πως τα μετράμε

Τα βασικά μεγέθη τα οποία χρειαζόμαστε για την εκτέλεση πειραμάτων και την κατασκευή του μοντέλου μας είναι,

1. Μέγιστη υπολογιστική επίδοση (MaximumFLOPS).
2. Μέγιστο εύρος ζώνης κύριας μνήμης.
3. Χρόνος εκτέλεσης σειριακής εκδοχής προγράμματος.
4. Χρόνος εκτέλεσης παράλληλης εκδοχής προγράμματος.
5. Πλήθος πράξεων κινητής υποδιαστολής.
6. Σύνολο bytes που μεταφέρθηκαν από και προς την κύρια μνήμη.

Οι μετρήσεις αυτών των μεγεθών και οποιουδήποτε άλλου μεγέθους που δεν καταγράφεται εδώ αλλά σχετίζεται με το μοντέλο μας, θα πρέπει να ακολουθούν τους κανόνες που αναφέρονται στις βασικές υποθέσεις.

Για την πρώτη υπόθεση αρκεί να χρησιμοποιήσουμε datasets που να έχουν μέγεθος μεγαλύτερο των 16MB για το Μηχάνημα 1 και μεγαλύτερο των 20MB για το Μηχάνημα 2.

Η δεύτερη και τρίτη υπόθεση, να εκτελεστεί ένα πρόγραμμα χρησιμοποιώντας μόνο φυσικούς πυρήνες, χωρίς hyperthreading σε ένα κόμβο NUMA και χρησιμοποιώντας αποκλειστικά τις διευθύνσεις της μνήμης που αντιστοιχούν στον κόμβο για το πρόγραμμα και τα δεδομένα του, γίνεται με τη χρήση τόσο των περιβαλλοντικών μεταβλητών της βιβλιοθήκης OpenMP όσο και με χρήση της εντολής *numactl*. Η εντολή *numactl* διαχειρίζεται την κατανομή των πόρων σε NUMA αρχιτεκτονικές. Συγκεκριμένα, μπορεί κανείς να επιλέξει για ένα πρόγραμμα σε ποιόν κόμβο NUMA να εκτελεστεί και σε ποιες διευθύνσεις μνήμης να αποθηκεύσει τις εντολές και τα δεδομένα του. Αυτό γίνεται με τις πρόσθετες ρυθμίσεις *–cpunodebind*=number και *–membind*=number αντίστοιχα. Ο αριθμός number επιλέγεται με βάση την αρχιτεκτονική NUMA του κάθε μηχανήματος, σύμφωνα με τα διαγράμματα μπλοκ αυτών που παρουσιάζονται στην ενότητα *Τα μηχανήματα του εργαστηρίου*. Για παράδειγμα, αν θέλουμε ένα πρόγραμμα να εκτελεστεί στο Μηχάνημα 1 στον κόμβο

NUMA 1 ενώ είναι αποθηκευμένο στη μνήμη του 3ου κόμβου NUMA καλούμε το *numactl* με τις επιλογές,

numactl –cpunodebind=1 –membind=3 ./program

Αν το πρόγραμμα είναι παράλληλο, γραμμένο με τη βιβλιοθήκη OpenMP και θέλουμε να χρησιμοποιήσει 5 φυσικούς πυρήνες του 1ου κόμβου NUMA του Μηχανήματος 1 τότε χρησιμοποιούμε τις ακόλουθες περιβαλλοντικές μεταβλητές της διεπαφής OpenMP· OMP_NUM_THREADS, OMP_PROC_BIND και OMP_PLACES. Αναλυτικά, θα έχουμε,

export OMP_NUM_THREADS=5
export OMP_PROC_BIND=true
export OMP_PLACES=′{8:13}′

Αν συνδιάσουμε και τα δύο, σε ένα τερματικό Linux θα έχουμε

$ export OMP_NUM_THREADS=5
$ export OMP_PROC_BIND=true
$ export OMP_PLACES=′{8:13}′
$ numactl –cpunodebind=1 –membind=3 ./program

8.3.1 Μέγιστη υπολογιστική επίδοση

Για τον υπολογισμό της μέγιστης υπολογιστικής επίδοσης χρησιμοποιήθηκε τόσο η θεωρητική τιμή (Εξίσωση (7.2)) όσο και μετροπρογράμματα όπως το LINPACK (OpenMP, Intel), HPL και το πρόγραμμα peakperf [47]. Προκειμένου να είμαστε συμβατοί με τις αρχικές υποθέσεις περιορίσαμε την εκτέλεση σε 8 φυσικούς πυρήνες για το πρώτο μηχάνημα και στους 14 φυσικούς πυρήνες για το δεύτερο. Το σύνολο δεδομένων του κάθε μετροπρογράμματος αποθηκεύτηκε στις διευθύνσεις μνήμης που αντιστοιχούν στον κόμβο NUMA που έτρεχε το αντίστοιχο μετροπρόγραμμα.

8.3.2 Μέγιστο εύρος ζώνης

Τα χαρακτηριστικά για τις μνήμες δε μας ήταν διαθέσιμα· επομένως δεν υπολογίσαμε τη θεωρητική τιμή με βάση την εξίσωση (7.3). Το μέγιστο εύρος ζώνης της κύριας μνήμης μετρήθηκε με χρήση του μετροπρογράμματος STREAM, όπου από την έξοδο του χρησιμοποιήσαμε το την τιμή για το τεστ triad. Το πείραμα triad αποτελείται από μία πρόσθεση και έναν πολλαπλασιασμό, πράξεις που περιγράφουν το μεγαλύτερο ποσοστό των προγραμμάτων που υπάρχουν. Επίσης, όπως και προηγουμένως, προκειμένου να είμαστε σύμφωνοι με τις υποθέσεις περιορίσαμε την εκτέλεση για το πρώτο μηχάνημα στους 8 φυσικούς πυρήνες και για το δεύτερο στους 14 φυσικούς πυρήνες, ενώ το σύνολο δεδομένων του STREAM αποθηκεύτηκε στις διευθύνσεις μνήμης του αντίστοιχου κόμβου εκτέλεσης.

8.3.3 Χρόνοι εκτέλεσης σειριακής και παράλληλης εκδοχής προγράμματος

Για τους χρόνους εκτέλεσης χρησιμοποιήθηκε η συνάρτηση *gettimeofday* της τυπικής βιβλιοθήκης *sys/time.h* της γλώσσας C. Επίσης, με χρήση του προγράμματος perf διαπιστώθηκε ότι η συνάρτηση αυτή μετράει τον πραγματικό χρόνο εκτέλεσης εντός ενός προγράμματος χωρίς να προσθέτει επιπλέον καθυστερήσεις εξαιτίας της προσθήκης της.

8.3.4 Πλήθος πράξεων κινητής υποδιαστολής και σύνολο bytes που μεταφέρθηκαν από και προς την κύρια μνήμη

Για τον υπολογισμό του πλήθους πράξεων κινητής υποδιαστολής και το σύνολο bytes που μεταφέρθηκαν από και προς την κύρια μνήμη χρησιμοποιήθηκε το υποπρόγραμμα *perf stat* του προγράμματος perf. Το άρθρωμα *perf stat* είναι ένα πρόγραμμα που λειτουργεί ως διεπαφή για τους μετρητές επίδοσης μιας μικροαρχιτεκτονικής. Προκειμένου να επικαλεστούμε τους κατάλληλους μετρητές επίδοσης χρειάζεται αρχικά να ανακαλύψουμε τα ονόματά τους.

Για το Μηχάνημα 1 χρησιμοποιήθηκε η σουίτα *libpfm4* [48] και το εγχειρίδιο παραγωγού λογισμικού της Intel για τη μικροαρχιτεκτονική Sandy Bridge[49]. Η σουίτα *libpfm4* περιέχει ένα πρόγραμμα για την καταγραφή των ονομάτων όλων των μετρητών και της λειτουργίας τους, σε λίστα, και ένα άλλο για την κωδικοποίησή τους σε δεκαεξαδική μορφή που καταλαβαίνει το *perf stat*. Το εγχειρίδιο παραγωγού περιέχει τις ίδιες πληροφορίες αλλά αναλύει σε μεγαλύτερο βαθμό τη λειτουργία του κάθε μετρητή. Για το Μηχάνημα 2 χρησιμοποιήθηκε η άλλη λειτουργία του προγράμματος perf, το *perf list*. Το *perf list* καταγράφει σε λίστα όλους τους μετρητές επίδοσης για μια μικροαρχιτεκτονική, αν αυτή είναι αποθηκευμένη στη βάση δεδομένων του. Στην περίπτωσή μας μόνο η μικροαρχιτεκτονική Skylake είναι αποθηκευμένη. Επίσης, αξίζει να σημειωθεί, ότι η σουίτα *libpfm4* μπορεί και εδώ να χρησιμοποιηθεί όπως και το εγχειρίδιο παραγωγού λογισμικού της μικροαρχιτεκτονικής Skylake για τα ονόματα και τις λειτουργίες των μετρητών. Στη συνέχεια παρουσιάζονται σε πίνακες οι μετρητές που χρησιμοποιούνται και η λειτουργία τους για κάθε μηχάνημα.

Μηχάνημα 1		
Όνομα	Κωδικός perf	Λειτουργία
FP_COMP_OPS_EXE:X87	r530110	Μετράει το πλήθος των πράξεων κινητής υποδιαστολής, δηλαδή των προσθέσεων, αφαιρέσεων, πολλαπλασιασμών, διαιρέσεων και ριζών.
FP_COMP_OPS_EXE: SSE_FP_PACKED_DOUBLE	r531010	Μετράει το πλήθος των έγκλειστων μικροεντολών κινητής υποδιαστολής διπλής ακρίβειας. SSE
FP_COMP_OPS_EXE: SSE_SCALAR_DOUBLE	r538010	Μετράει το πλήθος των βαθμωτών μικροεντολών κινητής υποδιαστολής απλής ακρίβειας. SSE
OFFCORE_RESPONSE_0: LLC_MISS_LOCAL_DRAM	r5301b7 r3f80408fff	Μετράει το πλήθος των αστοχιών της κρυφής μνήμης L3 για δεδομένα που βρίσκονται στην τοπική κύρια μνήμη του κόμβου.
OFFCORE_RESPONSE_0: LLC_MISS_REMOTE_DRAM	r5301bb r3f80408fff	Μετράει το πλήθος των αστοχιών της κρυφής μνήμης L3 για δεδομένα που βρίσκονται σε μια απομακρυσμένη κύρια μνήμη από αυτή του κόμβου.
OFFCORE_RESPONSE_1: LLC_MISS_LOCAL_DRAM	r5301b7 r3f80808fff	Μετράει το πλήθος των αστοχιών της κρυφής μνήμης L3 για δεδομένα που βρίσκονται στην τοπική κύρια μνήμη του κόμβου.
OFFCORE_RESPONSE_1: LLC_MISS_REMOTE_DRAM	r5301bb r3f80808fff	Μετράει το πλήθος των αστοχιών της κρυφής μνήμης L3 για δεδομένα που βρίσκονται σε μια απομακρυσμένη κύρια μνήμη από αυτή του κόμβου.

Μηχάνημα 2		
Όνομα	Κωδικός perf	Λειτουργία
fp_arith_inst_retired.scalar_double	fp_arith_inst_retired. scalar_double	Καταγράφει το πλήθος των πράξεων κινητής υποδιαστολής διπλής ακρίβειας, των προσθέσεων, αφαιρέσεων, πολλαπλασιασμών, διαιρέσεων και ριζών ως μια εντολή και των συνδιασμών μια εντολή, τη συγχώνευση πολλαπλασιασμού άθροισης (FMA) ως δύο εντολές και τη συγχώνευση πολλαπλασιασμού αφαίρεσης (FMS) ως δύο εντολές.
fp_arith_inst_retired. 128b_packed_double	fp_arith_inst_retired. 128b_packed_double	Καταγράφει το πλήθος των διανυσματικών πράξεων για διανύσματα δύο διαστάσεων που η κάθε διάσταση περιέχει έναν αριθμό κινητής υποδιαστολής διπλής ακρίβειας. Η κάθε πράξη υπολογίζεται ως δύο εντολές.
fp_arith_inst_retired. 256b_packed_double	fp_arith_inst_retired. 256b_packed_double	Καταγράφει το πλήθος των διανυσματικών πράξεων για διανύσματα τεσσάρων διαστάσεων που η κάθε διάσταση περιέχει έναν αριθμό κινητής υποδιαστολής διπλής ακρίβειας. Η κάθε πράξη υπολογίζεται ως τέσσερις εντολές.
OFFCORE_RESPONSE_0 _ANY_REQUEST_L3_MISS_LOCAL _SNP_ANY	r5301b7 r3f840085b7	Μετράει το πλήθος των αστοχιών της κρυφής μνήμης L3 για δεδομένα που βρίσκονται στην τοπική κύρια μνήμη του κόμβου.
OFFCORE_RESPONSE_0 _ANY_REQUEST _L3_MISS_REMOTE_HOP1_DRAM _SNP_ANY	r5301b7 r3f900085b7	Μετράει το πλήθος των αστοχιών της κρυφής μνήμης L3 για δεδομένα που βρίσκονται σε μια απομακρυσμένη κύρια μνήμη από αυτή του κόμβου.
OFFCORE_RESPONSE_1 _ANY_REQUEST_L3_MISS_LOCAL _SNP_ANY	r5301bb r3f840085b7	Μετράει το πλήθος των αστοχιών της κρυφής μνήμης L3 για δεδομένα που βρίσκονται στην τοπική κύρια μνήμη του κόμβου.
OFFCORE_RESPONSE_1 _ANY_REQUEST _L3_MISS_REMOTE_HOP1_DRAM _SNP_ANY	r5301b7 r3f900085b7	Μετράει το πλήθος των αστοχιών της κρυφής μνήμης L3 για δεδομένα που βρίσκονται σε μια απομακρυσμένη κύρια μνήμη από αυτή του κόμβου.

8.4 Ο παραλληλοποιημένος πυρήνας Jacobi

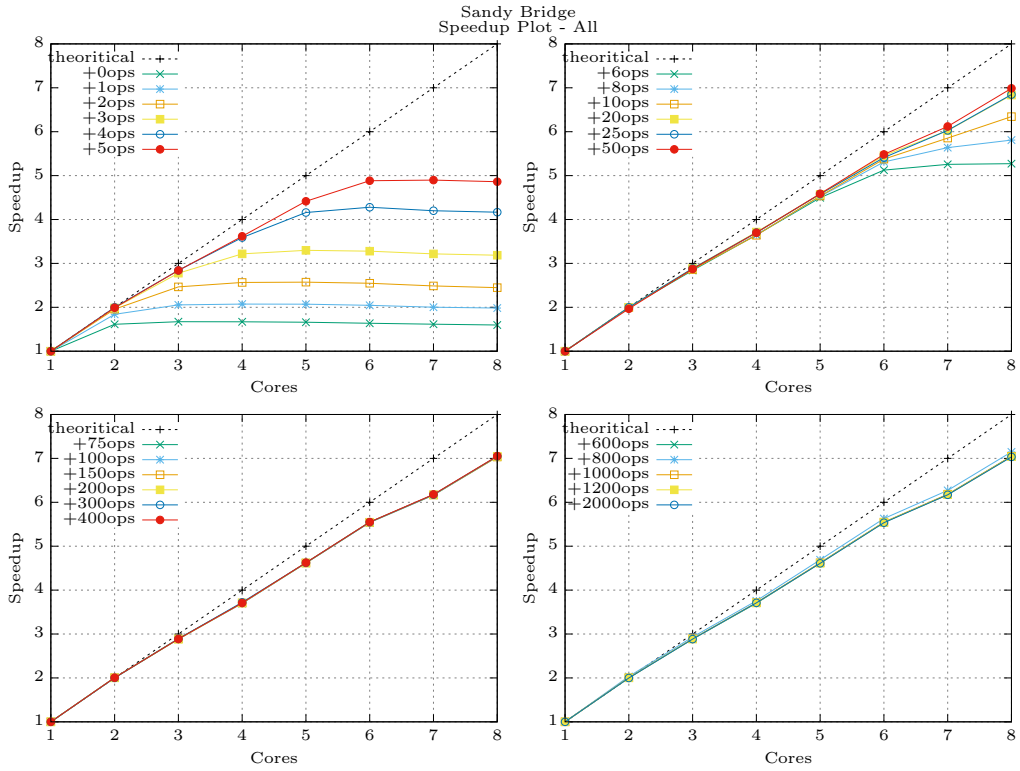
Για τη δημιουργία του μοντέλου χρειάστηκε να χρησιμοποιηθεί ένα πρόγραμμα ως βάση για τα πειράματα. Όπως σχολιάστηκε και στην εισαγωγή χρησιμοποιήθηκε ο προγραμματιστικός πυρήνας επίλυσης μερικών διαφορικών εξισώσεων κατά Jacobi - που θα αποκαλείται πυρήνας Jacobi από

εδώ και στο εξής για συντόμευση. Παρακάτω παρουσιάζεται ο πυρήνας Jacobi γραμμένος στη γλώσσα C και παραλληλοποιημένος με τη βοήθεια της βιβλιοθήκης OpenMP.

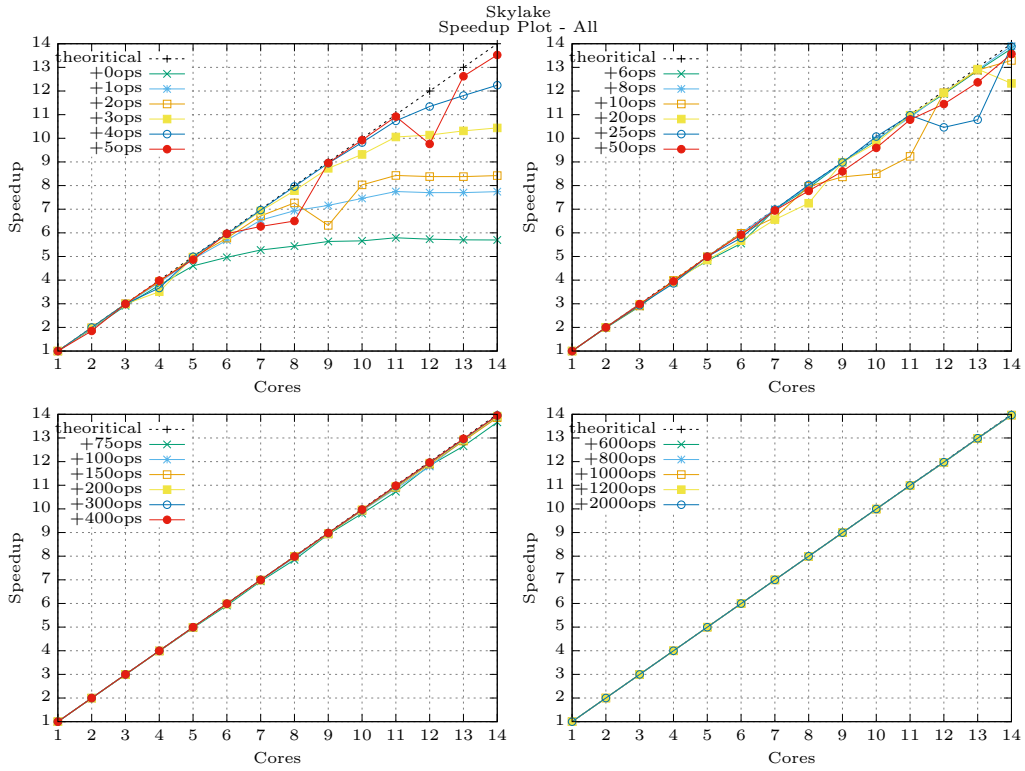
```
void Jacobi(double ** u_previous, double ** u_current,
            int X_min, int X_max, int Y_min, int Y_max)
{

    int i,j,k;

    #pragma omp parallel for private(i, j, k) shared(u_previous, u_current)
    for (i=X_min;i<X_max;i++) {
        for (j=Y_min;j<Y_max;j++) {

            #ifdef ITERS

            k = 0;
            /* If we want to perf to read the bytes transferred we should
             * force the program to read from the memory. Because the size
             * of a row is bigger in bytes than what the cache can store it
             * we call for an element in the previous row and/or next row */
            while (k < ITERS) {

                  /* We use an arithmetic progression to avoid optimization
                  from the compiler */
                  u_current[i][j] += u_previous[i-1][j-1];
                  k++;

            }

            #endif

            u_current[i][j]=(u_previous[i-1][j] + u_previous[i+1][j] +
                             u_previous[i][j-1] + u_previous[i][j+1]) / 4.0;
        }
    }
}
```

Για την παραλληλοποίηση χρησιμοποιήθηκε η εντολή προεπεξεργαστή *#pragma omp* για να ειδοποιήσει τον μεταγλωττιστή ότι πρόκειται για παράλληλη περιοχή στην οποία θα αρχικοποιηθούν νήματα από τη βιβλιοθήκη OpenMP. Σε αυτή την περιοχή τα νήματα διαχωρίζουν τη δουλειά που πρέπει να γίνει ανά γραμμή, για τους πίνακες u_previous και u_current. Κάθε νήμα αναλαμβάνει δηλαδή να επεξεργαστεί/υπολογίσει ένα πλήθος γραμμών του πίνακα u_current χωρίς να παρεμβαίνει στο πλήθος γραμμών ενός άλλου νήματος. Για το λόγο αυτό το κάθε νήμα θα πρέπει να έχει τις δικές του συντεταγμένες γραμμής και στήλης οι οποίες θα είναι προσβάσιμες μόνο από το ίδιο νήμα. Αυτές οι πρόσθετες πληροφορίες είναι εύκολο να εφαρμοστούν με τη βιβλιοθήκη OpenMP με την προσθήκη στην εντολή προεπεξεργαστή προτάσεων για τη διαχείριση των μεταβλητών εντός της παράλληλης περιοχής. Γίνεται να οριστούν το ποιες μεταβλητές είναι ιδιωτικές ανά νήμα και ποιες είναι μοιραζόμενες μεταξύ αυτών. Η γραμμή i και η στήλη j για κάθε νήμα θα

πρέπει να είναι ιδιωτικές για τους λόγους που αναφέραμε. Οι δείκτες των πινάκων εν αντιθέσει δεν απαιτείται να είναι ιδιωτικοί διότι καμία πράξη δεν επηρεάζει τη θέση τους στην κύρια μνήμη για πιθανότητα εμφάνισης race conditions αλλά ούτε ο πυρήνας Jacobi εμπεριέχει πράξεις που δημιουργούν εξαρτήσεις μεταξύ των γραμμών τους. Τέλος, εντός των δύο επαναλήψεων προστέθηκε και μια επιπλέον επανάληψη με τη μεταβλητή k, η οποία αναλόγως με το πως είναι ορισμένο το ITERS προσθέτει επιπλέον πράξεις κινητής υποδιαστολής ενώ διατηρεί το σύνολο των bytes που μεταφέρονται από και προς την κύρια μνήμη, σταθερό. Με αυτή τη δομή επανάληψης για το k είναι πολύ εύκολο θεωρώντας το ITERS ως μεταβλητή που ορίζεται κατά τη μεταγλώττιση του πυρήνα Jacobi να υλοποιηθούν πυρήνες Jacobi με διαφορετική ένταση λειτουργίας. Ο λόγος για τον οποίο κάνουμε κάτι τέτοιο είναι επειδή θέλουμε να μελετήσουμε το μοντέλο μας σε όλο το μήκος του, δηλαδή και στην περιοχή που περιορίζεται από το μέγιστο εύρος ζώνης μνήμης και στην περιοχή που περιορίζεται από τη μέγιστη υπολογιστική επίδοση του συστήματος. Στη συνέχεια παρουσιάζονται τα διαγράμματα κλιμακωσιμότητας των πυρήνων Jacobi στα δύο μηχανήματα του εργαστηρίου. Τα διαγράμματα κλιμακωσιμότητας είναι καλή ένδειξη για το τι συμπεριφορά να περιμένουμε από ένα πρόγραμμα στα Roofline μοντέλα του κάθε μηχανήματος.



Σχήμα 8.3: Διάγραμμα κλιμακωσιμότητας με τους πυρήνες Jacobi για διαφορετική ένταση λειτουργίας στο Μηχάνημα 1.



Σχήμα 8.4: Διάγραμμα κλιμακωσιμότητας με τους πυρήνες Jacobi για διαφορετική ένταση λειτουργίας στο Μηχάνημα 2.

Στα υπομνήματα οι πυρήνες Jacobi αποτυπώνονται με τον κωδικό + *number* **ops**, όπου *ops* προκύπτει από το operations και είναι οι πράξεις (εδώ κινητής υποδιαστολής) που προστίθενται στον αρχικό πυρήνα Jacobi. Το number είναι το πλήθος των πράξεων που προστίθενται ανά επανάληψη εντός του πυρήνα Jacobi.

Από τα διαγράμματα κλιμακωσιμότητας βλέπουμε ότι για χαμηλό πληθος πράξεων κινητής υποδιαστολής, οι αντίστοιχοι πυρήνες Jacobi δεν κλιμακώνουν καλά ως προς το πλήθος των πυρήνων. Ο κορεσμός τους σε χαμηλό πλήθος πυρήνων δείχνει ότι περιορίζονται από κάποιον άλλο παράγοντα και αυτός είναι το μέγιστο εύρος ζώνης της κύριας μνήμης. Δηλαδή, οι πυρήνες Jacobi που στα διαγράμματα δεν κλιμακώνουν καλά βρίσκονται αριστερά από το γόνατο του Roofline του αντίστοιχου συστήματος. Καθώς αυξάνονται οι πράξεις οι αντίστοιχοι πυρήνες απεικονίζονται ως γραφικές παραστάσεις που ¨ορθοστατούν' και γίνονται πιο γραμμικές. Αυτό ουσιαστικά σημαίνει πως η ένταση λειτουργίας μετατοπίζεται προς τα δεξία πλησιάζοντας το γόνατο αλλά μην προσπερνώντας το. Για μεγάλο πλήθος πράξεων κινητής υποδιαστολής, πλέον, οι γραφικές παραστάσεις των αντίστοιχων πυρήνων Jacobi είναι (σχεδόν) γραμμικές. Αυτό απεικονισμένο στο Roofline μοντέλο θα σήμαινε πως οι πυρήνες αυτοί βρίσκονται δεξιά από το γόνατο του μοντέλου και περιορίζονται μόνο από τη μέγιστη υπολογιστική επίδοση του κάθε συστήματος.

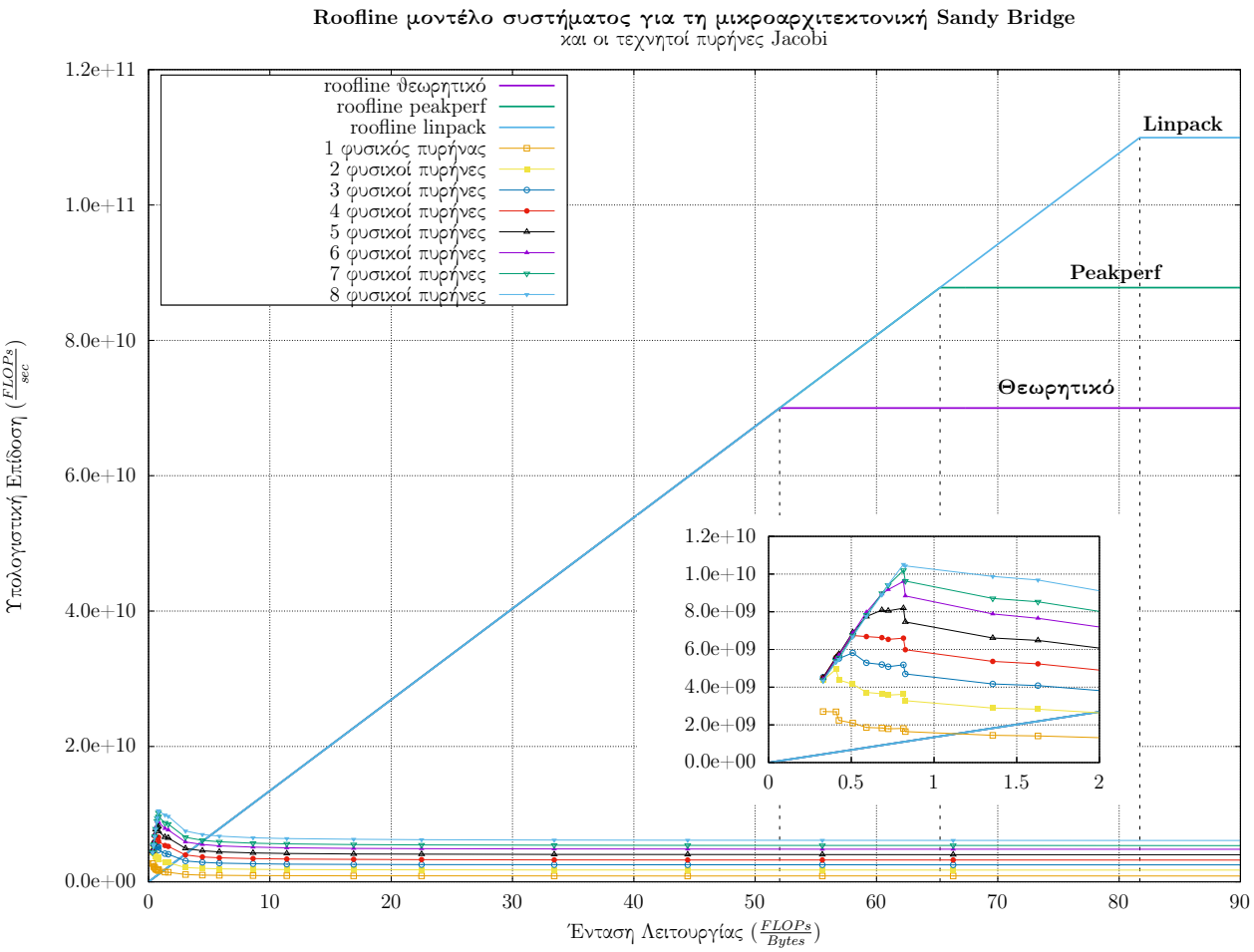
8.5 Η αδυναμία του κλασικού μοντέλου Roofline

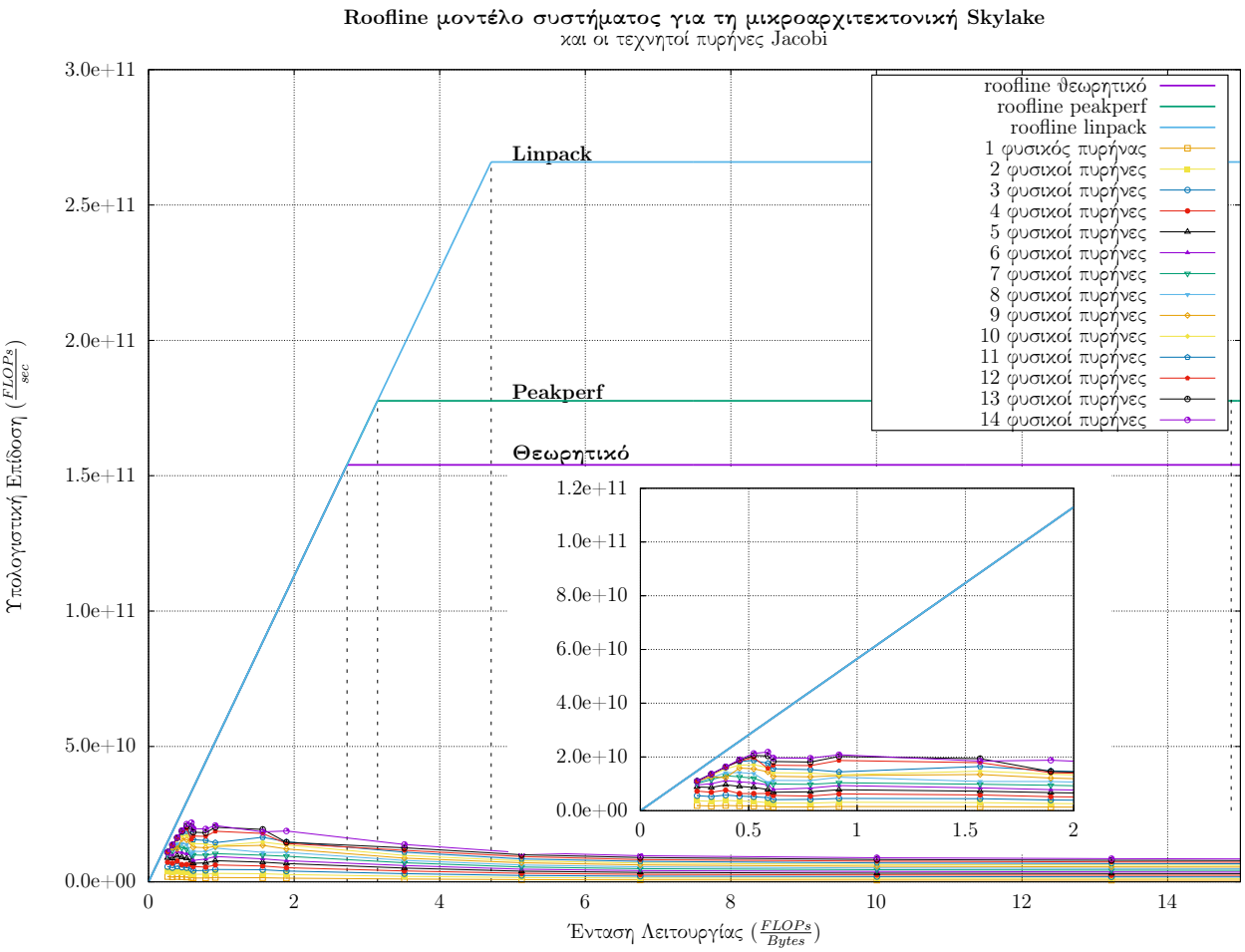
Η πρώτη προσπάθεια να δημιουργήσουμε ένα μοντέλο προβλέψεων βασίστηκε στην χρήση του κλασικού Roofline μοντέλου, αυτούσιου. Για την κατασκευή του μοντέλου Roofline χρειαζόμαστε το μέγιστο εύρος ζώνης του κάθε μηχανήματος και τη μέγιστη υπολογιστική επίδοση τους. Στον επόμενο πίνακα παρουσιάζονται οι μετρήσεις μας για κάθε μηχάνημα.

	Μηχάνημα 1	Μηχάνημα 2
Μέγιστη υπολογιστική επίδοση (Θεωρητικά)	70 GFLOPS ($1 \times 1 \times 4 \times 2.2G \times 8 \times 1$)	154 GFLOPS ($1 \times 1 \times 5 \times 2.2G \times 14 \times 1$)
Μέγιστη υπολογιστική επίδοση (Linpack)	109.95 GFLOPS	265.87 GFLOPS
Μέγιστη υπολογιστική επίδοση (peakperf)	$\frac{175.58}{2} = 87.79$ GFLOPS	$\frac{710.68}{4} = 177.67$ GFLOPS
Μέγιστο εύρος ζώνης (STREAM benchmark)	13451.8 MB/s	48559.5 MB/s

Όλες οι μετρήσεις έγιναν σύμφωνα με τις βασικές υποθέσεις του μοντέλου, όπως περιγράφεται στο κεφάλαιο *Ποια μεγέθη μας ενδιαφέρουν και πως τα μετράμε*. Είναι σημαντικό να σημειωθεί πως το μετροπρόγραμμα peakperf χρησιμοποιεί διανυσματικές εντολές για τη μέτρηση της μέγιστης υπολογιστικής επίδοσης, εξού και η διαίρεση με το αντίστοιχο πλήθος διαστάσεων των διανυσμάτων για κάθε μηχάνημα.

Στη συνέχεια, χρησιμοποιώντας τις μετρήσεις των χρόνων και των εντάσεων λειτουργίας για τον πυρήνα Jacobi του κεφαλαίου *Ο παράλληλος πυρήνας Jacobi*, κατασκευάζουμε το μοντέλο Roofline για κάθε μηχάνημα και για κάθε κατώφλι μέγιστης υπολογιστικής επίδοσης και απεικονίζουμε τους πυρήνες Jacobi σε αυτά.


Roofline μοντέλο συστήματος για τη μικροαρχιτεκτονική Sandy Bridge
και οι τεχνητοί πυρήνες Jacobi
roofline θεωρητικό
roofline peakperf
roofline linpack
1 φυσικός πυρήνας
2 φυσικοί πυρήνες
3 φυσικοί πυρήνες
4 φυσικοί πυρήνες
5 φυσικοί πυρήνες
6 φυσικοί πυρήνες
7 φυσικοί πυρήνες
8 φυσικοί πυρήνες
Linpack
Peakperf
Θεωρητικό
Υπολογιστική Επίδοση ($\frac{FLOPs}{sec}$)
Ένταση Λειτουργίας ($\frac{FLOPs}{Bytes}$)
1.2e+11
1.0e+11
8.0e+10
6.0e+10
4.0e+10
2.0e+10
0.0e+00
0
10
20
30
40
50
60
70
80
90
1.2e+10
1.0e+10
8.0e+09
6.0e+09
4.0e+09
2.0e+09
0.0e+00
0
0.5
1
1.5
2



Για την κατασκευή των διαγραμμάτων χρησιμοποιήθηκαν οι πυρήνες +0ops, +1ops, +2ops, +3ops, +4ops, +5ops, +6ops, +8ops, +10ops, +20ops, +25ops, +50ops, +75ops, +100ops, +150ops, +200ops. Όσο μεγαλύτερος είναι ο αριθμός των ops τόσο μεγαλύτερη είναι η ένταση λειτουργίας ενός πυρήνα.

Από τα διαγράμματα φαίνεται πως οι πυρήνες παρουσιάζουν υπολογιστικές επιδόσεις πολύ μικρές συγκριτικά με τα κατώφλια των Roofline μοντέλων για κάθε μηχάνημα. Μια πρώτη υπόθεση θα μπορούσε να είναι ότι υπάρχουν κόστη στην παραλληλοποίηση του κώδικα τα οποία δεν έχουν εξαλειφθεί. Αναλύοντας τον πυρήνα Jacobi όμως δεν υπάρχει κάτι το οποίο προσθέτει σημαντικό κόστος. Η μοναδική καθυστέρηση που προστίθεται είναι ο συγχρονισμός των νημάτων, η οποία όμως είναι μικρή καθώς επαναλαμβάνεται μόνο 256 φορές εντός του προγράμματος. Επομένως, δεν υπάρχει κάποια πρόσθετη καθυστέρηση η οποία να μειώνει τόσο πολύ την επίδοση των πυρήνων. Μελετώντας τα διαγράμματα, παρατηρούμε πως σε χαμηλές εντάσεις λειτουργίας οι γραφικές παραστάσεις των πυρήνων αυξάνονται, κυρίως με γραμμικό χαρακτήρα, μέχρι την περιοχή 0.5 με 1, στην οποία παρουσιάζουν ένα γόνατο. Από αυτό το γόνατο και έπειτα η επίδοσή τους παραμένει

σταθερή. Η γραφική απεικόνιση των πυρήνων θυμίζει ένα μοντέλο Roofline το οποίο έχει γόνατο μια τιμή κοντά στο 0.5 και στα δύο μηχανήματα. Δηλαδή, αυτό το μοντέλο Roofline που προκύπτει από τους πυρήνες Jacobi φαίνεται πως είναι ανεξάρτητο από το σύστημα στο οποίο εκτελούνται. Αξίζει να σημειωθεί επίσης, ότι στο διάγραμμα του Μηχανήματος 1 στις χαμηλές εντάσεις λειτουργίας η επίδοση των πυρήνων φαίνεται πως ξεπερνάει την επίδοση του μηχανήματος.

Συμπερασματικά, τα μοντέλα Roofline για τα δύο μηχανήματα, αν και συνοψίζουν την ικανότητα των αρχιτεκτονικών τους, δεν μπορούν να χρησιμοποιηθούν για προβλέψεις της επίδοσης των πυρήνων Jacobi καθώς το σφάλμα στην πρόβλεψη θα ήταν υπερβολικά μεγάλο.

8.6 Η αρχική μορφή του μοντέλου μας

Όπως φάνηκε στην προηγούμενη ενότητα με το κλασικό Roofline μοντέλο είναι αδύνατο να γίνουν βάσιμες προβλέψεις. Είδαμε πως δεν υπάρχει στον πυρήνα Jacobi κάποιο πρόσθετο κόστος το οποίο να μειώνει την επίδοσή του. Παρατηρώντας το γόνατο το οποίο σχηματίζουν οι πυρήνες Jacobi συμπεραίνουμε πως το ανώτατο κατώφλι μέγιστης υπολογιστικής επίδοσης θα πρέπει να είναι χαμηλότερο προκειμένου το γόνατο που σχηματίζει το Roofline μοντέλο συστήματος να μετακινηθεί αριστερά. Για το λόγο αυτό προχωρούμε σε μια επιπλέον υπόθεση. Θεωρούμε στο μοντέλο μας ότι η μέγιστη υπολογιστική επίδοση ισούται με τη γραμμική κλιμάκωση του παράλληλου προγράμματος που μελετούμε για το μέγιστο πλήθος φυσικών πυρήνων που του διαθέτουμε. Έτσι, το μοντέλο Roofline από μοντέλο αρχιτεκτονικής μετατρέπεται σε μοντέλο προγράμματος, καθώς για διαφορετικά προγράμματα θα υπάρχει και διαφορετικό μοντέλο Roofline. Το μοντέλο μας εκφράζει τον παράλληλο χρόνο εκτέλεσης με την επόμενη μαθηματική μορφή,

$$T(p, OI) = \mathbb{1}_{[0,\text{γόνατο})}(OI) \times \frac{bytes_{program}}{bandwidth} + \mathbb{1}_{[\text{γόνατο},\infty)}(OI) \times \frac{T_s}{p} \tag{8.1}$$

όπου p είναι το πλήθος φυσικών πυρήνων που διαθέτουμε στο παράλληλο πρόγραμμα, OI είναι η ένταση λειτουργίας αυτού, T_s είναι ο σειριακός χρόνος εκτέλεσής του, $bytes_{program}$ είναι το πλήθος των bytes που μεταφέρθηκαν από και προς την κύρια μνήμη του συστήματος κατά την εκτέλεση του προγράμματος και $bandwidth$ είναι το μέγιστο εύρος ζώνης του συστήματος.

Το γόνατο του μοντέλου μας όμως σε αυτό το στάδιο δεν είναι σαφώς ορισμένο διότι προσδιορίσαμε με δική μας τιμή τη μέγιστη υπολογιστική επίδοση του Roofline μοντέλου. Διατηρώντας όμως το σκεπτικό του κλασικού μοντέλου Roofline μπορούμε να το υπολογίσουμε. Θεωρώντας ως ανώτατο κατώφλι υπολογιστικής επίδοσης την επίδοση του παράλληλου προγράμματος για γραμμική κλιμάκωση στο μέγιστο πλήθος πυρήνων που του διαθέτουμε, έχουμε,

$$bandwidth \times \text{γόνατο} = \frac{FLOPs_{program}}{\frac{T_s}{p}} \Rightarrow \text{γόνατο} = \frac{FLOPs_{program} \times p}{T_s \times bandwidth} \tag{8.2}$$

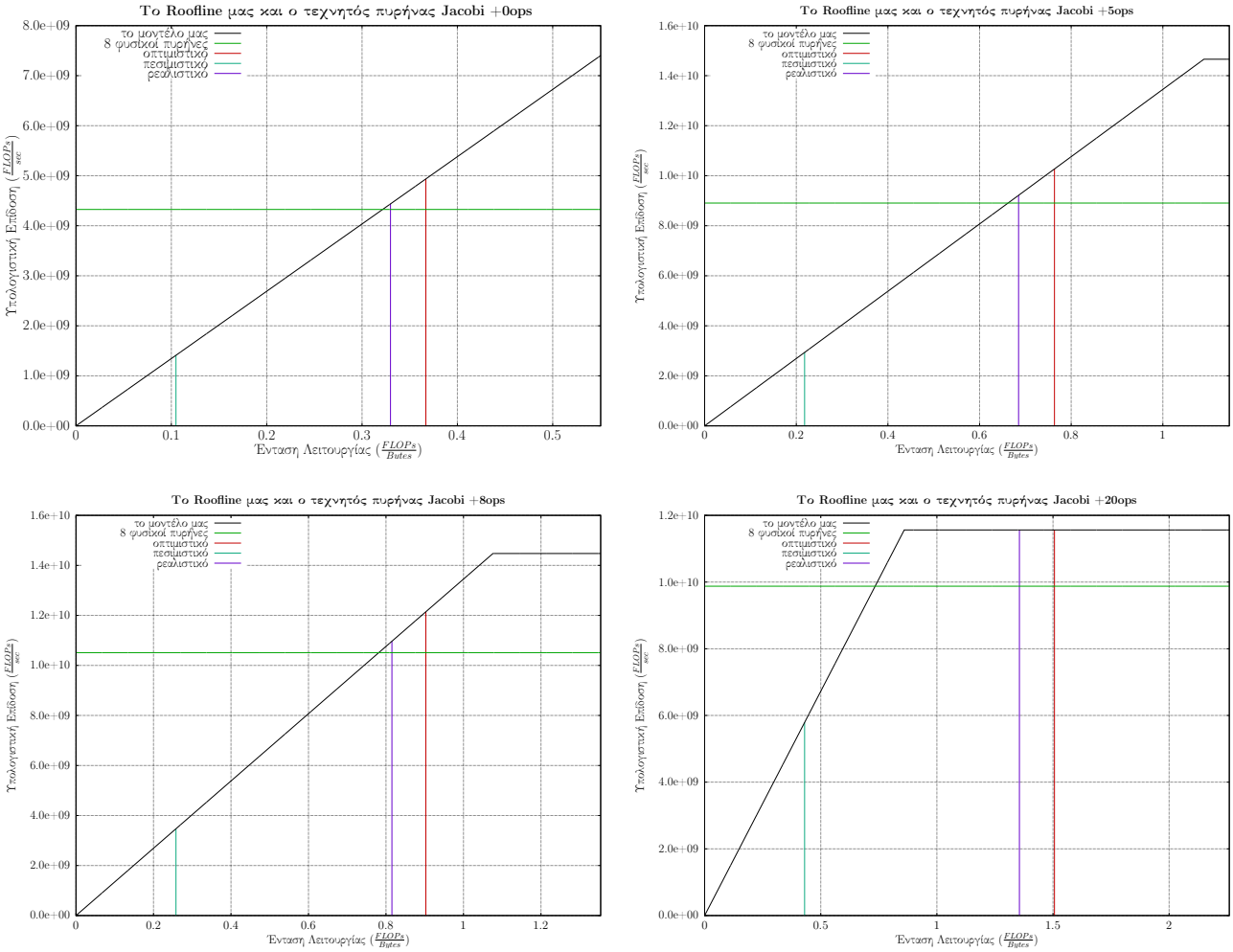
με $FLOPs_{program}$ το πλήθος των πράξεων κινητής υποδιαστολής που εκτελεί το πρόγραμμα.

8.6.1 Τα τρία μοντέλα έντασης λειουργίας ενός προγράμματος

Πριν προχωρήσουμε στην παρουσίαση της αποδοτικότητας του μοντέλου μας στις προβλέψεις του, γίνεται έρευνα στο κατά πόσο η χρήση των μετρητών επίδοσης επιστρέφουν σωστά την ένταση λειτουργίας ενός προγράμματος. Ο λόγος για τον οποίο γίνεται αυτή η μελέτη είναι επειδή υπάρχει πιθανότητα η επίδοση των πυρήνων Jacobi να αλλοιώνεται από την ένταση λειτουργίας τους. Επίσης, στο μοντέλο μας το κατώφλι της μέγιστης υπολογιστικής επίδοσης και το γόνατο εξαρτώνται από το πλήθος των πράξεων κινητής υποδιαστολής που μετρούν οι μετρητές επίδοσης για κάθε πρόγραμμα.

Για το λόγο αυτό δημιουργήθηκαν τρία μοντέλα τα οποία υπολογίζουν τρεις διαφορετικές εντάσεις λειτουργίας για ένα πρόγραμμα. Για την εξαγωγή συμπερασμάτων βασιζόμαστε για ακόμα μια φορά στον παραλληλοποιημένο πυρήνα Jacobi.

Δύο μοντέλα είναι θεωρητικά. Η διαφορά τους έγκειται στο πως αντιμετωπίζουν την πρόσβαση δεδομένων στη μνήμη. Το πρώτο θεωρεί πως η πρόσβαση στα δεδομένα των δύο πινάκων του πυρήνα Jacobi ισοδυναμεί απλώς με τη μεταφορά δύο πινάκων στον πυρήνα Jacobi. Το μοντέλο αυτό το ονομάζουμε **οπτιμιστικό**, διότι θεωρεί χαμηλό πλήθος προσβάσεων. Το δεύτερο θεωρητικό μοντέλο, εξετάζει τον πυρήνα Jacobi και θεωρεί κάθε κάλεσμα ενός στοιχείου των πινάκων ως πρόσβαση στη μνήμη. Το μοντέλο αυτό το ονομάζουμε **πεσιμιστικό**, διότι θεωρεί μια πολυάσχολη μνήμη από την οποία για κάθε επανάληψη εντός του πυρήνα Jacobi ζητούνται ή αποθηκεύονται δεδομένα. Το τρίτο μοντέλο είναι το πιο πρακτικό και αυτό που μας ενδιαφέρει περισσότερο, διότι χρησιμοποιούνται οι μετρητές επίδοσης. Το μοντέλο αυτό το ονομάζουμε **ρεαλιστικό**, για να δηλώσουμε ότι δεν προκύπτει από τη θεωρητική μελέτη της δομής του πυρήνα Jacobi. Στη συνέχεια παρουσιάζονται σε διαγράμματα τα τρία μοντέλα έντασης λειτουργίας για επιλεγμένους τεχνητούς πυρήνες Jacobi στα δύο μηχανήματα του εργαστηρίου. Στα διαγράμματα αυτά απεικονίζεται η πραγματική επίδοση του κάθε τεχνητού πυρήνα Jacobi ως σταθερή συνάρτηση, διότι μας ενδιαφέρει να δούμε σε ποιο σημείο τέμνει τη γραφική παράσταση του μοντέλου μας. Η τετμημένη αυτού του σημείου τομής είναι η ένταση λειτουργίας του τεχνητού πυρήνα Jacobi. Ένα μοντέλο υπολογίζει καλύτερα την ένταση λειτουργίας ενός πυρήνα αν η τιμή που εξάγει βρίσκεται πλησιέστερα στην τιμή της τετμημένης της τομής. Με τα διαγράμματα αυτά, μας ενδιαφέρει να δούμε ποιο μοντέλο προβλέπει καλύτερα την ένταση λειτουργίας του κάθε τεχνητού πυρήνα.

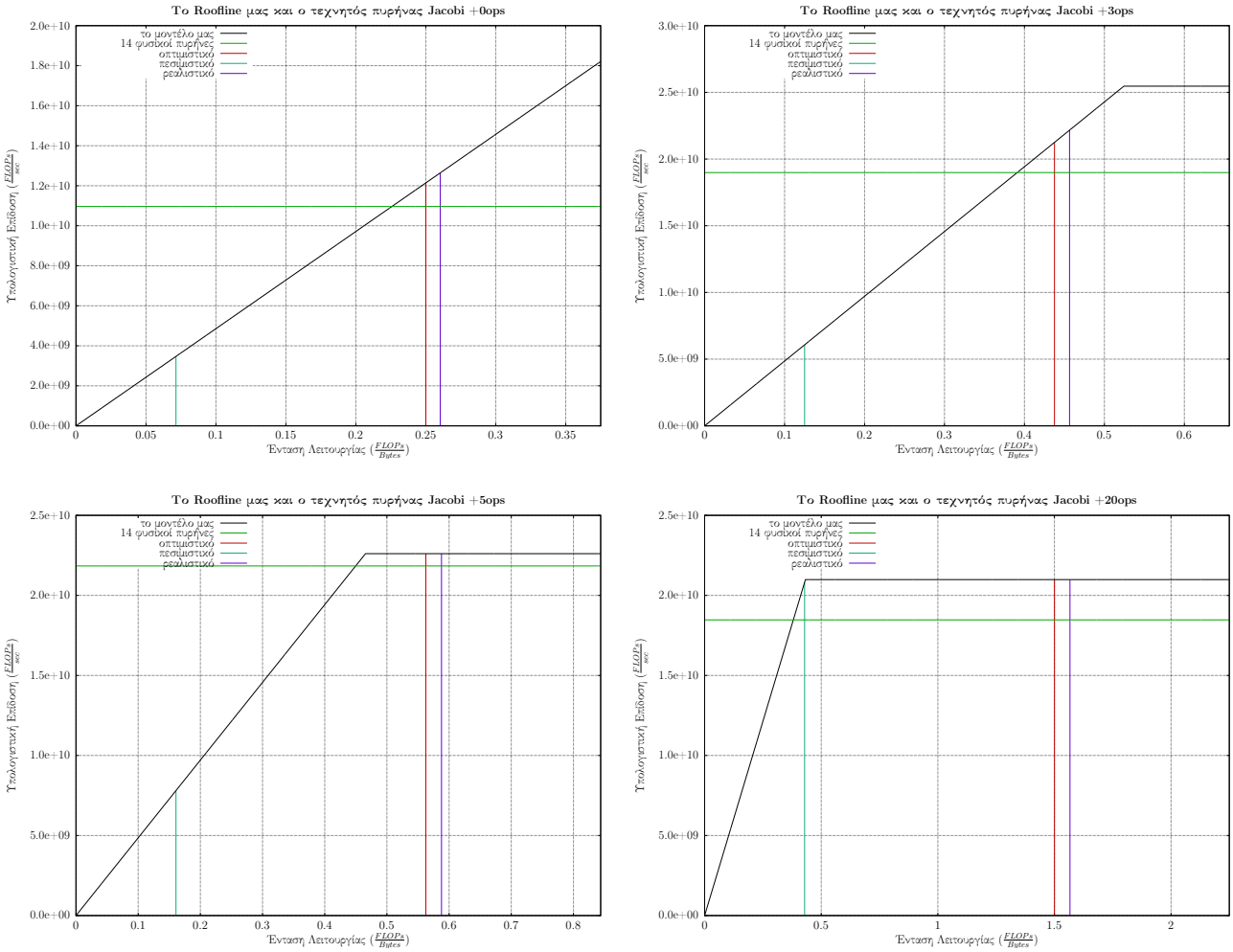


Σχήμα 8.5: Τα τρία μοντέλα έντασης λειτουργίας στο Μηχάνημα 1. Έλεγχος με τους τεχνητούς πυρήνες +0ops, +5ops, +8ops, +20ops

Στο Μηχάνημα 1 παρατηρούμε πως για τους τεχνητούς πυρήνες +0ops, +5ops, +8ops το ρεαλιστικό μοντέλο αποδίδει πολύ καλύτερα από τα άλλα δύο μοντέλα καθώς είναι πιο κοντά στο σημείο τομής της γραφικής παράστασης του μοντέλου μας με τη σταθερή συνάρτηση της πραγματικής επίδοσης του κάθε πυρήνα. Για τον τεχνητό πυρήνα Jacobi +20ops η ερμηνεία περιπλέκεται. Η πραγματική επίδοση του πυρήνα τέμνει τη διαγώνιο του μοντέλου μας και το πεσιμιστικό μοντέλο είναι πιο κοντά στο σημείο τομής. Αντιθέτως, το ρεαλιστικό και οπτιμιστικό μοντέλο βρίσκονται δεξιά του γόνατος του μοντέλου μας, στην περιοχή που περιορίζεται από την επίδοση γραμμικής κλιμακωσιμότητας του πυρήνα. Από τα διαγράμματα των μοντέλων Roofline για τα μηχανήματα, που παρουσιάστηκαν στην ενότητα *Η αδυναμία του κλασικού μοντέλου Roofline* γνωρίζουμε ότι για τους μεγαλύτερους τεχνητούς πυρήνες Jacobi η επίδοση παραμένει σταθερή. Επομένως, η

σταθερή συνάρτηση της πραγματικής επίδοσης των μεγάλων πυρήνων θα συνεχίζει να τέμνει τη διαγώνιο των Roofline μοντέλων τους. Αντιθέτως, τα τρία μοντέλα έντασης λειτουργίας των πυρήνων θα έχουν μετατοπιστεί όλα δεξιά του κάθε γόνατος διότι το πλήθος των πράξεων κινητής υποδιαστολής θα έχει αυξηθεί. Συμπερασματικά, και στον τεχνητό πυρήνα Jacobi +20ops το ρεαλιστικό μοντέλο προβλέπει την ένταση λειτουργίας καλύτερα διότι όπως αποδείξαμε ο πυρήνας σε αυτή την περίπτωση είναι όντως processor bounded.

Για όλους τους πυρήνες το πεσιμιστικό μοντέλο προβλέπει χαμηλή την ένταση λειτουργίας τους, συγκριτικά με τα άλλα δύο μοντέλα. Αυτό είναι αναμενόμενο διότι σύμφωνα με τον ορισμό του το σύνολο των bytes που θεωρεί ότι μεταφέρονται από και προς την κύρια μνήμη είναι μεγαλύτερο από τα άλλα δύο μοντέλα. Το οπτιμιστικό μοντέλο προβλέπει την ένταση λειτουργίας για κάθε τεχνητού πυρήνα πιο ψηλά από το ρεαλιστικό μοντέλο, επειδή θεωρεί πως οι πίνακες έχουν αποθηκευτεί στις κρυφές μνήμες και δεν απαιτείται ξανά προσπέλαση των δεδομένων τους από την κύρια μνήμη.



Σχήμα 8.6: Τα τρία μοντέλα έντασης λειτουργίας στο Μηχάνημα 2. Έλεγχος με τους τεχνητούς πυρήνες +0ops, +3ops, +5ops, +20ops

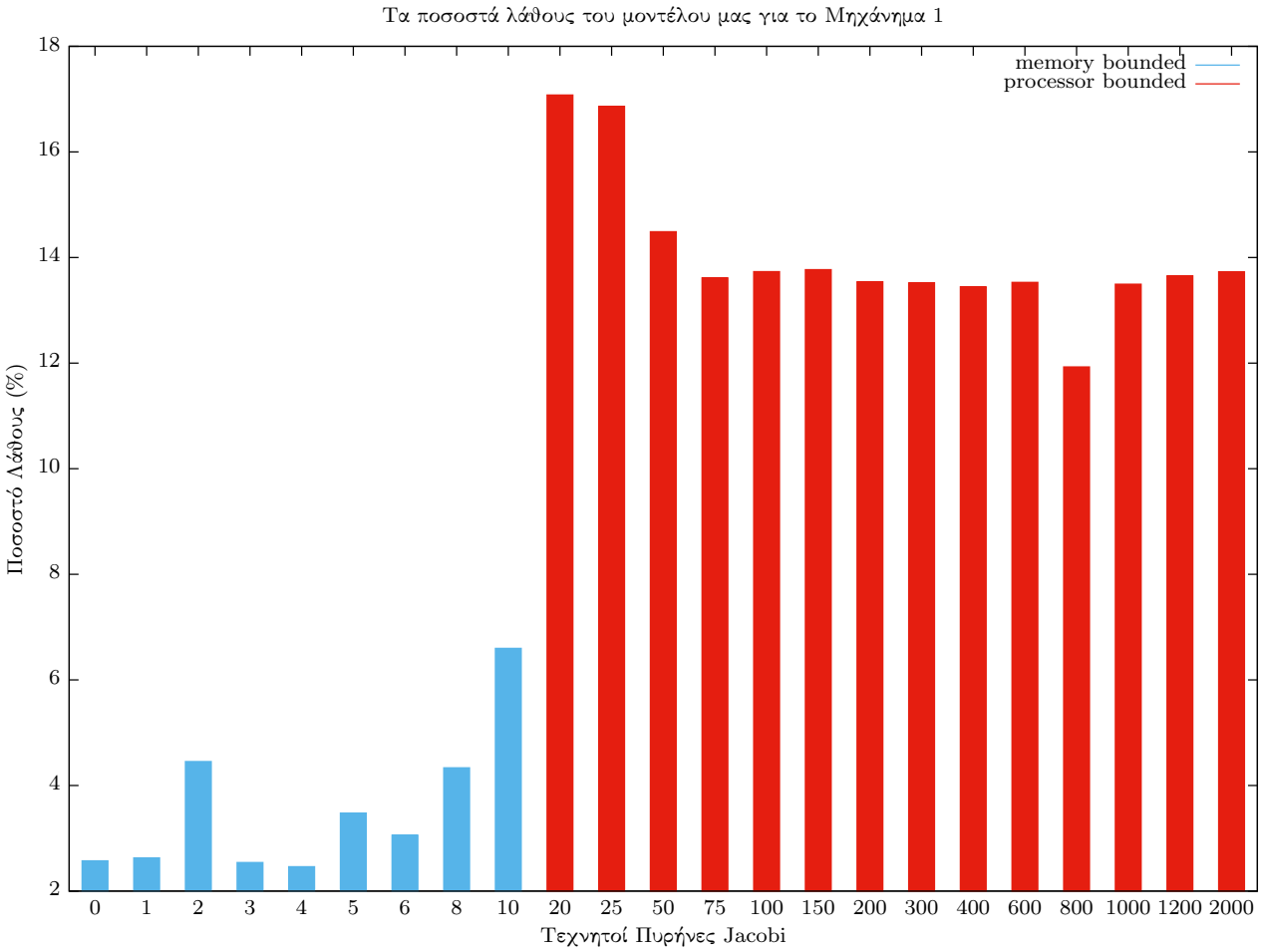
Τα ίδια συμπεράσματα για το Μηχάνημα 2 είναι τα ίδια με το Μηχάνημα 1. Το ρεαλιστικό μοντέλο φαίνεται πως αντιλαμβάνεται καλύτερα για το που βρίσκεται η ένταση λειτουργίας του κάθε τεχνητού πυρήνα από τα υπόλοιπα δύο μοντέλα.

Συνολικά, το ρεαλιστικό μοντέλο, δηλαδή η χρήση των μετρητών επίδοσης που διατίθενται από τους επεξεργαστές κάθε μηχανήματος, για τον υπολογισμό της έντασης λειτουργίας είναι καλή λύση με μικρά λάθη, συγκριτικά με τα άλλα δύο μοντέλα. Όμως και τα δύο θεωρητικά μοντέλα έχουν τη χρήση τους. Μπορούν να χρησιμοποιηθούν ως ανώτατα και κατώτατα όρια για την ένταση λειτουργίας ενός προγράμματος ή να θεωρηθούν ως όρια για την απασχόληση της μνήμης από τον επεξεργαστή. Για παράδειγμα, αν το ρεαλιστικό μοντέλο τείνει πιο κοντά στο πεσιμιστικό σημαίνει ότι κατά την εκτέλεση του προγράμματος η μνήμη θα είναι απασχολημένη. Εν αντιθέσει, αν το

ρεαλιστικό μοντέλο είναι πιο κοντά στο οπτιμιστικό μοντέλο τότε σημαίνει πως το πρόγραμμα δεν απασχολεί τόσο πολύ τη μνήμη του συστήματος.

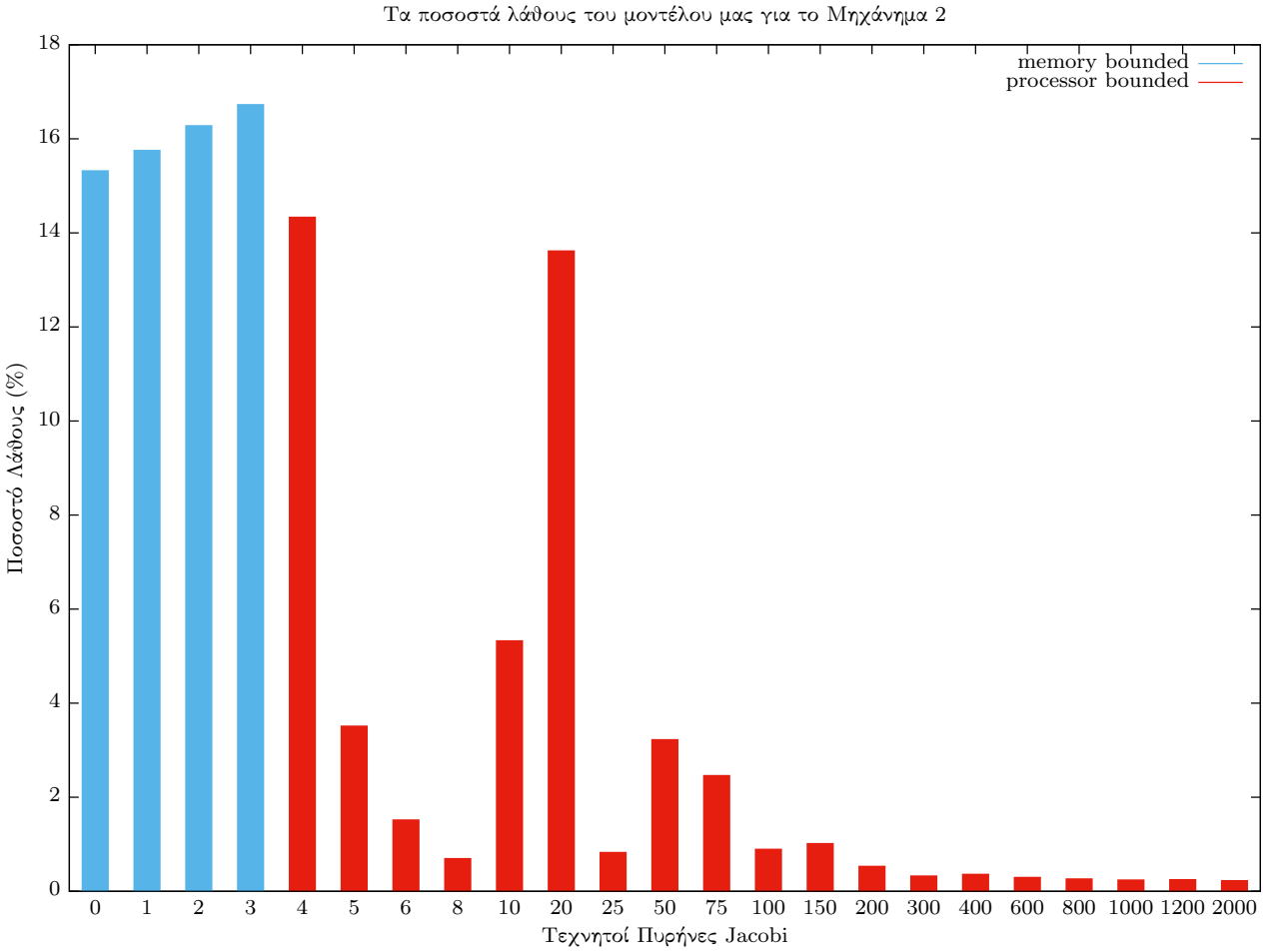
8.6.2 Οι προβλέψεις του αρχικού μοντέλου μας

Αφού επιβεβαιώθηκε πως η χρήση των μετρητών επίδοσης που προσφέρουν οι επεξεργαστές επιστρέφουν ικανοποιητικές μετρήσεις για την ένταση λειτουργίας των τεχνητών πυρήνων, απομένει να ελεγχθεί η ικανότητα του μοντέλου μας στην πρόβλεψη της επίδοσης των τεχνητών πυρήνων Jacobi. Στη συνέχεια, παρουσιάζονται τα διαγράμματα με το ποσοστό λάθους στην πρόβλεψη για κάθε τεχνητό πυρήνα και στα δύο μηχανήματα και με πλήθος πυρήνων το μέγιστο πλήθος φυσικών πυρήνων που μπορεί να διαθέσει ένας κόμβος NUMA για κάθε μηχάνημα.



Σχήμα 8.7: Τα ποσοστιαία σφάλματα των προβλέψεων του μοντέλου μας για κάθε τεχνητό πυρήνα Jacobi στο Μηχάνημα 1 με 8 φυσικούς πυρήνες.

Όπως φαίνεται από το διάγραμμα το μοντέλο μας σημείωσε ικανοποιητικά ποσοστά λάθους για το Μηχάνημα 1, με μέγιστη τιμή λάθους $\approx 17\%$ στον πυρήνα +20ops. Η τιμή αυτή είναι πολύ μικρή για θεωρηθεί σημαντικό σφάλμα στην ικανότητα πρόβλεψης του ως μοντέλο. Τα μεγαλύτερα ποσοστά λάθους σημειώθηκαν στους πυρήνες που βρίσκονται δεξιά του γονάτου του μοντέλου μας, στην περιοχή που περιορίζεται από τη μέγιστη υπολογιστική επίδοση (processor bounded).



Σχήμα 8.8: Τα ποσοστιαία σφάλματα των προβλέψεων του μοντέλου μας για κάθε τεχνητό πυρήνα Jacobi στο Μηχάνημα 2 με 14 φυσικούς πυρήνες.

Από το διάγραμμα διαπιστώνουμε πως το μοντέλο μας απέδωσε τόσο καλά στο Μηχάνημα 1 όσο στο Μηχάνημα 2. Και για τα δύο μηχανήματα το μέγιστο ποσοστιαίο σφάλμα που σημειώνεται ισούται περίπου με 17%.

Στο Μηχάνημα 1 παρατηρούμε πως τα μεγαλύτερα σφάλματα παρουσιάζονται στην περιοχή που περιορίζεται από το μέγιστο υπολογιστικό κατώφλι. Αυτό σημαίνει πως υπάρχουν κόστη τα

οποία δεν επιτρέπουν την πλήρη γραμμική κλιμάκωση των μεγαλύτερων τεχνητών πυρήνων Jacobi. Παρατηρώντας, τα διαγράμματα κλιμακωσιμότητας (Σχήμα 8.3) για το Μηχάνημα 1, όντως για τους μεγαλύτερους τεχνητούς πυρήνες η επιτάχυνσή τους ως προς το πλήθος των φυσικών πυρήνων πλησιάζει τη γραμμική αλλά δε γίνεται ποτέ. Στην πραγματικότητα δηλαδή η επιτάχυνση για 8 φυσικούς πυρήνες δεν είναι 8. Επομένως, το μοντέλο μας θεωρώντας ως υπολογιστικό κατώφλι την επίδοση των τεχνητών πυρήνων για γραμμική κλιμάκωση αυτών στους 8 πυρήνες, παρουσιάζει τα σφάλματα στις προβλέψεις του που σχολιάστηκαν.

Στο Μηχάνημα 2, αντιθέτως, τα μεγαλύτερα σφάλματα προκύπτουν στην περιοχή που περιορίζεται από το μέγιστο εύρος ζώνης της μνήμης. Μια αιτία θα μπορούσε να είναι πως το μέγιστο εύρος ζώνης μνήμης που μετρήθηκε για το δεύτερο μηχάνημα είναι μεγαλύτερο σε τιμή από το πραγματικό εύρος ζώνης της μνήμης του. Εκτός από την περιοχή αυτή σημαντικά σφάλματα παρουσιάζονται και στους τεχνητούς πυρήνες +4ops και +20ops για 14 φυσικούς πυρήνες στον NUMA κόμβο. Από το Σχήμα 8.4 διαπιστώνουμε ότι όντως για αυτούς τους τεχνητούς πυρήνες Jacobi η επιτάχυνση για μικρούς πυρήνες δεν είναι ομαλή και πως για τους πυρήνες +4ops, +20ops δεν είναι γραμμική στους 14 φυσικούς πυρήνες.

8.7 Τα πρόσθετα κόστη του OpenMP

Στην προηγούμενη ενότητα αλλά και στην ενότητα *Πρόσθετα Κόστη* παρουσιάστηκαν διάφοροι λόγοι για τους οποίους η κλιμακωσιμότητα ενός παράλληλου συστήματος περιορίζεται. Στην ενότητα αυτή θα μελετήσουμε τα βασικά κόστη τα οποία προσθέτει η βιβλιοθήκη OpenMP που χρησιμοποιούμε για την παραλληλοποίηση των προγραμμάτων μας και το πως μπορούμε να τα εφαρμόσουμε στις προβλέψεις του μοντέλου μας. Για τη μέτρηση των πρόσθετων καθυστερήσεων κατασκευάστηκαν κατάλληλα μετροπρογράμματα με βάση το [50]. Η ιδέα είναι να χρησιμοποιηθεί ως βάση ένας ακολουθιακός πυρήνας από τον οποίο κατασκευάζονται παράλληλες εκδοχές του με τις δομές OpenMP που μας ενδιαφέρουν. Ο χρόνος εκτέλεσης του σειριακού πυρήνα θα χρησιμοποιηθεί ως χρόνος αναφοράς για κάθε καθυστέρηση που προσθέτουν οι δομές. Η μεταβλητή *innerreps* είναι το πλήθος δοκιμών για τον υπολογισμό του χρόνου εκτέλεσης της κάθε εκδοχής του πυρήνα. Για κάθε πείραμα το πλήθος δοκιμών ισούται με δέκα εκατομμύρια. Στο τέλος χρησιμοποιούνται οι χρόνοι εκτέλεσης για τον υπολογισμό της πρόσθετης καθυστέρησης μιας δομής. Η διαφορά του χρόνου εκτέλεσης μιας παράλληλης εκδοχής του πυρήνα με το χρόνο αναφοράς, διαιρείται με το πλήθος δοκιμών. Επομένως, τα αποτελέσματα είναι αμερόληπτα ως προς τυχόν διαταραχές του συστήματος στο οποίο γίνονται οι μετρήσεις.

Ο συγχρονισμός των νημάτων αποτελεί πρόσθετο κόστος κατά την παραλληλοποίηση ενός προγράμματος. Όπως αναφέρθηκε η χρονοδρομολόγηση νημάτων κατά την έναρξη μιας παράλληλης περιοχής και ο έλεγχος πρόσβασης στα δεδομένα εντός αυτής είναι τα βασικά αίτια για επιπλέον καθυστερήσεις. Οι εντολές προεπεξεργαστή από τη βιβλιοθήκη OpenMP που μελετούνται είναι

- **#pragma omp parallel**
- **#pragma omp parallel for**

- **#pragma omp barrier**
- **#pragma omp single**

```
...
for (i = 0; i < interreps; i++)
  delay(&delaytime);
...
```

Listing 8.1: Σειριακός πυρήνας για τον υπολογισμό του χρόνου αναφοράς

Ο βασικός πυρήνας για το συγκεκριμένο πείραμα είναι η συνάρτηση *delay*, που χρησιμοποιεί τη συνάρτηση *usleep* ως βάση για την κατασκευή της. Ουσιαστικά, ο πυρήνας βάζει σε αδράνεια ένα νήμα όταν την εκτελεί για χρόνο *delaytime*.

Προκειμένου η μέτρηση του χρόνου πρόσθετων καθυστερήσεων για το συγχρονισμό νημάτων να είναι σωστή σε κάθε παράλληλη εκδοχή για κάθε δομή OpenMP θα πρέπει να προσμετρηθεί και η συνεισφορά του πλήθους νημάτων στο χρόνο εκτέλεσης. Για να μην επηρεάζεται ο χρόνος εκτέλεσης από το πλήθος των νημάτων η ιδέα είναι να διαμοιραστεί σε κάθε νήμα το ίδιο φορτίο, ίσο με το φορτίο του σειριακού πυρήνα, ώστε η παράλληλη εκδοχή να λειτουργεί όπως η σειριακή. Με αυτό τον τρόπο ανεξαρτητοποιείται ο χρόνος εκτέλεσης των παράλληλων εκδοχών από το πλήθος νημάτων και η διαφορά του χρόνου εκτέλεσής της είναι η πραγματική καθυστέρηση που προσθέτουν οι δομές OpenMP.

```
...
for (i = 0; i < interreps; i++) {
  #pragma omp parallel firstprivate(i) shared(delaytime)
  {
    #pragma omp for
    for (j = 0; j < omp_get_num_threads(); j++)
      delay(&delaytime);
  }
}
...
```

Listing 8.2: Πυρήνας για τη μέτρηση του πρόσθετου χρόνου της δομής omp parallel

```
...
for (i = 0; i < interreps; i++) {
  #pragma omp parallel for shared(delaytime)
  for (j = 0; j < omp_get_num_threads(); j++)
    delay(&delaytime);
}
...
```

Listing 8.3: Πυρήνας για τη μέτρηση του πρόσθετου χρόνου της δομής omp parallel for

```
...
for (i = 0; i < interreps; i++) {
```

```
  #pragma omp parallel firstprivate(i) shared(delaytime)
  {
    #pragma omp for nowait
    for (j = 0; j < omp_get_num_threads(); j++)
      delay(&delaytime);
    #pragma omp barrier
  }
}
...
```

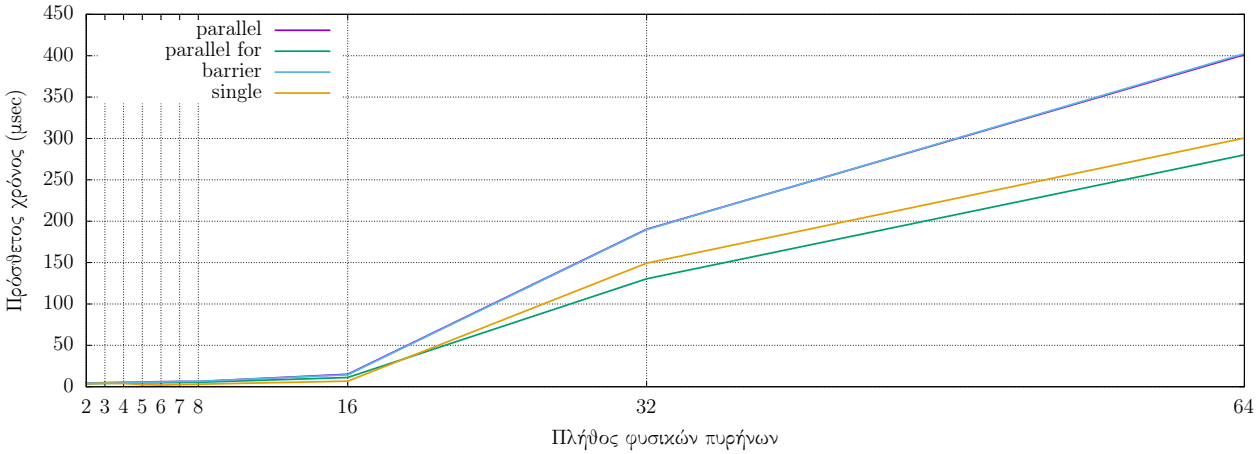
Listing 8.4: Πυρήνας για τη μέτρηση του πρόσθετου χρόνου της δομής omp barrier

```
...
for (i = 0; i < interreps; i++) {
  #pragma omp parallel firstprivate(i) shared(delaytime)
  {
    for (j = 0; j < omp_get_num_threads(); j++) {
      #pragma omp single nowait
      {
        delay(&delaytime);
      }

    }
  }
}
...
```

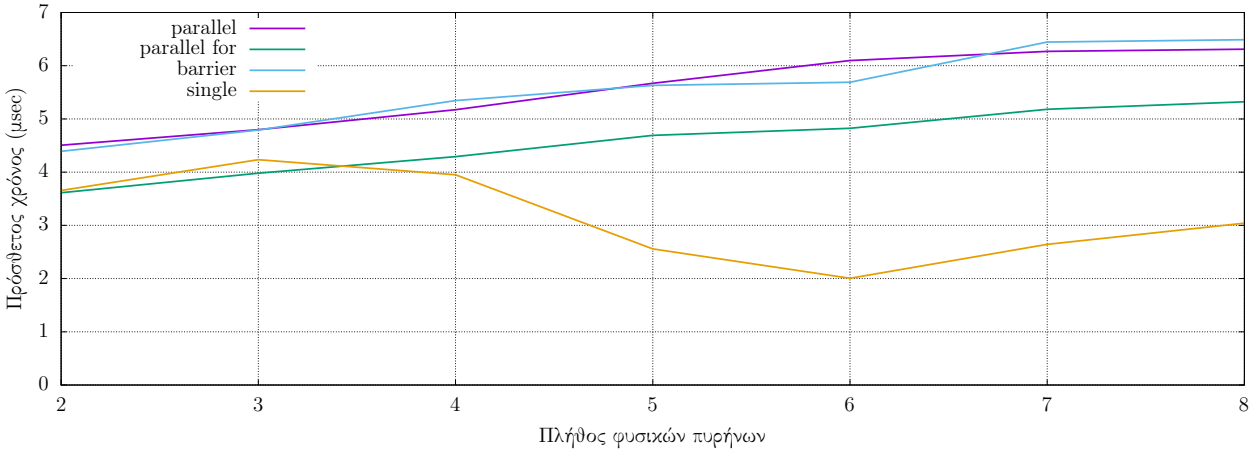
Listing 8.5: Πυρήνας για τη μέτρηση του πρόσθετου χρόνου της δομής omp single

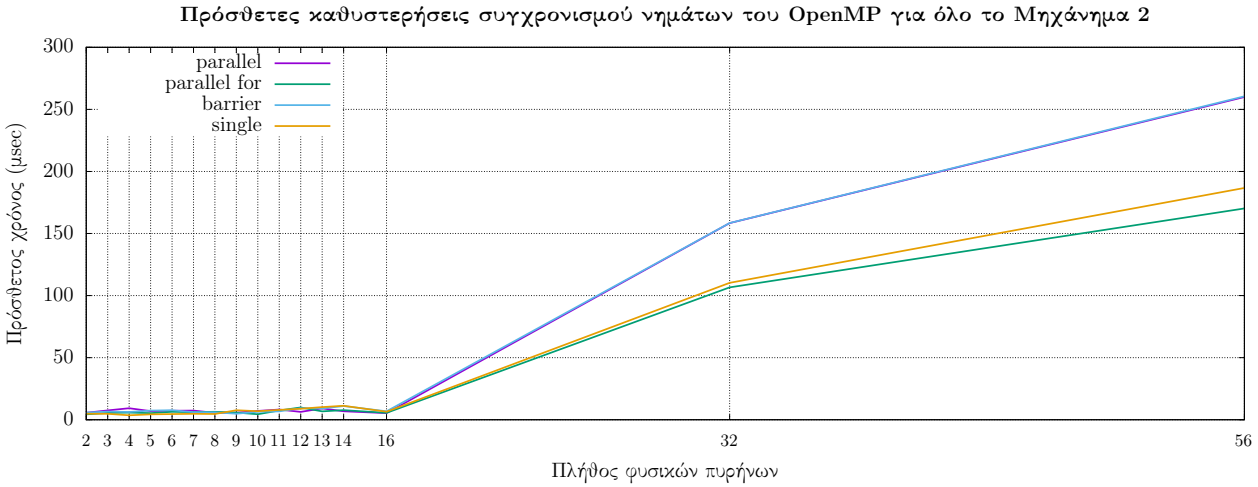
Στη συνέχεια για κάθε μηχάνημα παρουσιάζεται το κόστος συγχρονισμού για κάθε πλήθος φυσικών πυρήνων ενός κόμβου NUMA.

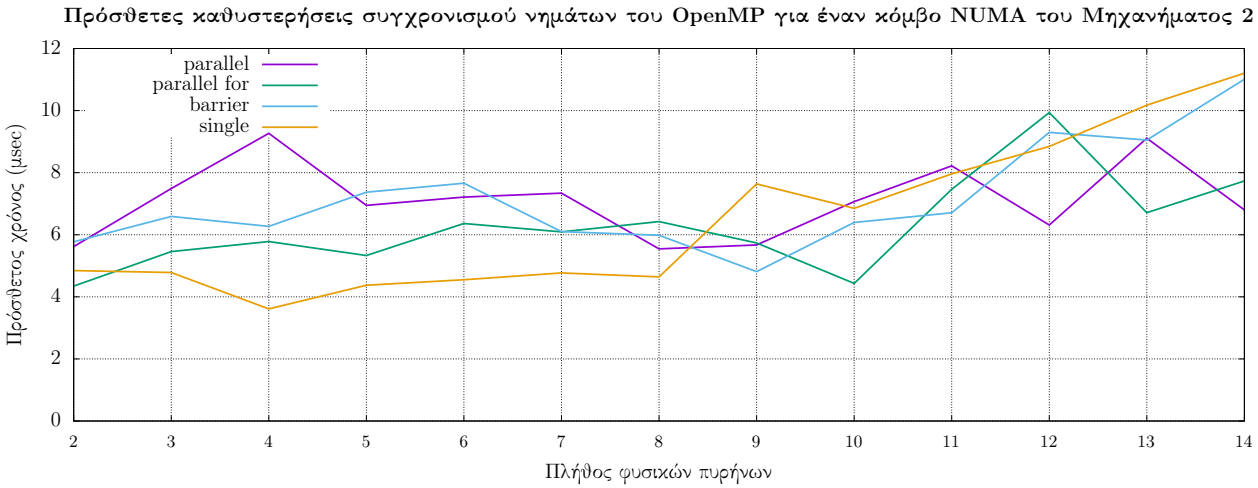
Πρόσθετες καθυστερήσεις συγχρονισμού νημάτων του OpenMP για όλο το Μηχάνημα 1



Πρόσθετες καθυστερήσεις συγχρονισμού νημάτων του OpenMP για έναν κόμβο NUMA του Μηχανήματος 1







Από τα διαγράμματα μπορούμε να εξάγουμε πως η δομή **omp parallel for** προσθέτει τη μικρότερη καθυστέρηση σε ένα παράλληλο πρόγραμμα. Εν αντιθέσει η δομή **omp parallel** προσθέτει τη μεγαλύτερη καθυστέρηση. Αυτό προκύπτει ενδεχομένως επειδή η δομή αυτή είναι η γενικότερη δομή παραλληλοποίησης της βιβλιοθήκης OpenMP. Μπορεί να διαχειριστεί μια work-sharing περιοχή και με άλλες δομές περαν της δομής **omp for**, όπως τη δομή **omp sections**, οπότε και θα πραγματοποιούνται περισσότεροι έλεγχοι συγχρονισμού.

Ο έλεγχος πρόσβασης σε κοινόχρηστα δεδομένα αποτελεί επίσης ένα σημαντικό παράγοντα προσθήκης καθυστερήσεων σε ένα παράλληλο πρόγραμμα. Οι δομές της βιβλιοθήκης OpenMP που χρησιμοποιούνται για την προστασία των δεδομένων είναι

- **#pragma omp critical**
- **#pragma omp lock**
- **#pragma omp atomic**

- **#pragma omp reduction**

```
...
for ( i = 0; i < innerreps; i++ ) {
  svar = 0;
  for (j = 0; j < ARRAY_SIZE; j++)
    svar += arr[j];
}
...
```

Listing 8.6: Σειριακός πυρήνας για την μέτρηση του χρόνου αναφοράς

Για τη μέτρηση των καθυστερήσεων των δομών OpenMP που χρησιμοποιούνται σε αμοιβαίως αποκλειόμενες περιοχές, χρησιμοποιείται ως βάση ο παραπάνω σειριακός πυρήνας. Αποτελείται από μια αρχικοποίηση της μεταβλητής *svar* στην οποία προστίθονται όλα τα στοιχεία του πίνακα *arr*. Ο πίνακας *arr* αρχικοποιείται πριν τον πυρήνα και έχει μέγεθος *ARRAY_SIZE*.

Ο λόγος για τον οποίο κατασκευάσαμε αυτό τον πυρήνα αντί να γίνει χρήση του προηγούμενου, ήταν επειδή χρειαζόμαστε ένα κοινόχρηστο δεδομένο μεταξύ των νημάτων. Το ρόλο αυτόν αναλαμβάνει η κοινόχρηστη μεταβλητή *svar*, στην οποία προστίθενται συνεχώς τα στοιχεία του πίνακα. Μια απροσεξία στη σειρά εκτέλεσης των αθροίσεων θα οδηγήσει σε εσφαλμένο αποτέλεσμα. Ο λόγος για τον οποίο χρησιμοποιούνται στο άθροισμα στοιχεία ενός πίνακα είναι επειδή μελετάται η δομή **omp reduction** της βιβλιοθήκης OpenMP.

```
...
for ( i = 0; i < innerreps; i++ ) {
  svar = 0;
  #pragma omp parallel for shared(svar,arr)
  for (j = 0; j < ARRAY_SIZE; j++)
    #pragma omp critical
    svar += arr[j];
}
...
```

Listing 8.7: Πυρήνας για τη μέτρηση του πρόσθετου χρόνου της δομής omp critical

```
...
omp_init_lock(&lock);
for ( i = 0; i < innerreps; i++ ) {
  svar = 0;
  #pragma omp parallel for shared(svar,arr)
  for (j = 0; j < ARRAY_SIZE; j++) {
    omp_set_lock(&lock);
    svar += arr[j];
    omp_unset_lock(&lock);
  }
}
...
```

Listing 8.8: Πυρήνας για τη μέτρηση του πρόσθετου χρόνου της δομής omp lock

```
...
for ( i = 0; i < innerreps; i++ ) {
  svar = 0;
  #pragma omp parallel for shared(svar,arr)
  for (j = 0; j < ARRAY_SIZE; j++)
    #pragma omp atomic
    svar += arr[j];
}
...
```

Listing 8.9: Πυρήνας για τη μέτρηση του πρόσθετου χρόνου της δομής omp atomic

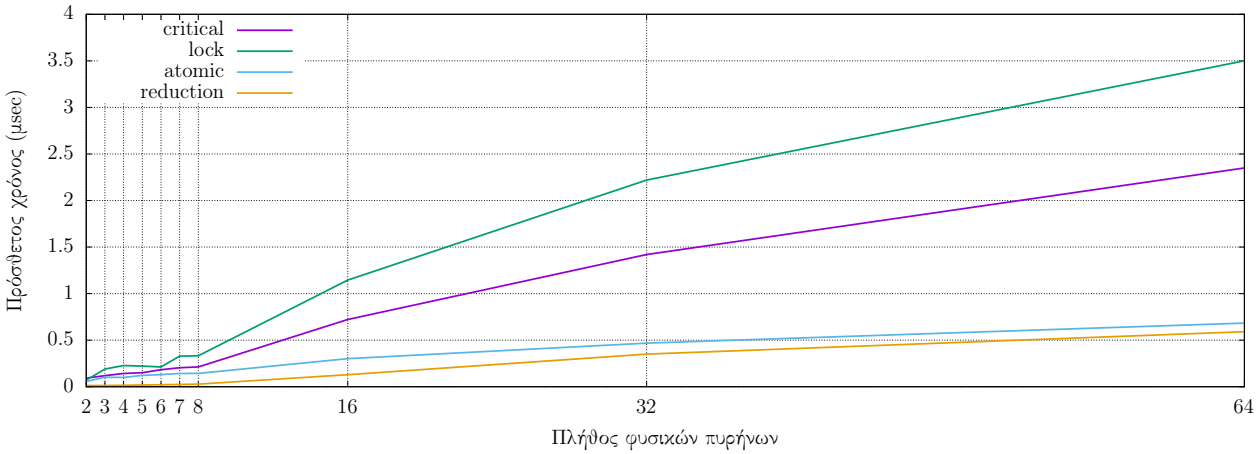
```
...
for ( i = 0; i < innerreps; i++ ) {
  svar = 0;
  #pragma omp parallel for shared(arr) reduction(+:svar)
  for (j = 0; j < ARRAY_SIZE; j++)
    svar += arr[j];
}
...
```

Listing 8.10: Πυρήνας για τη μέτρηση του πρόσθετου χρόνου της δομής omp reduction

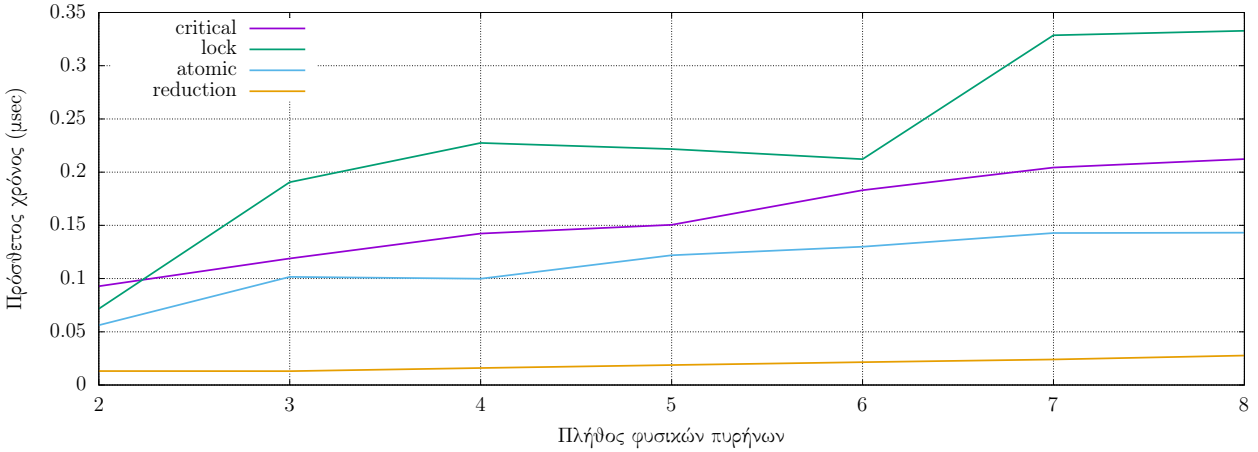
Τέλος, για τον υπολογισμό της καθυστέρησης που προσθέτει η κάθε δομή, διαιρούμε με το πλήθος επαναλήψεων που είναι το πλήθος δοκιμών *innerreps* επί το μέγεθος *ARRAY_SIZE* του πίνακα *arr*.

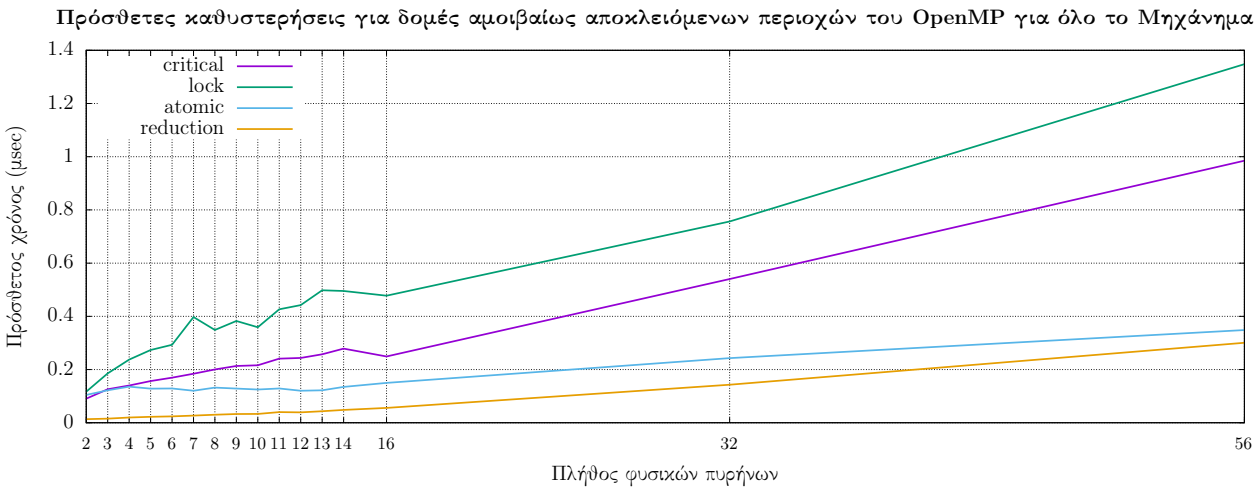
Στα επόμενα διαγράμματα απεικονίζεται ο πρόσθετος χρόνος όταν χρησιμοποιούνται οι δομές αυτές ως προς το πλήθος των φυσικών πυρήνων για τα δύο μηχανήματα του εργαστηρίου.

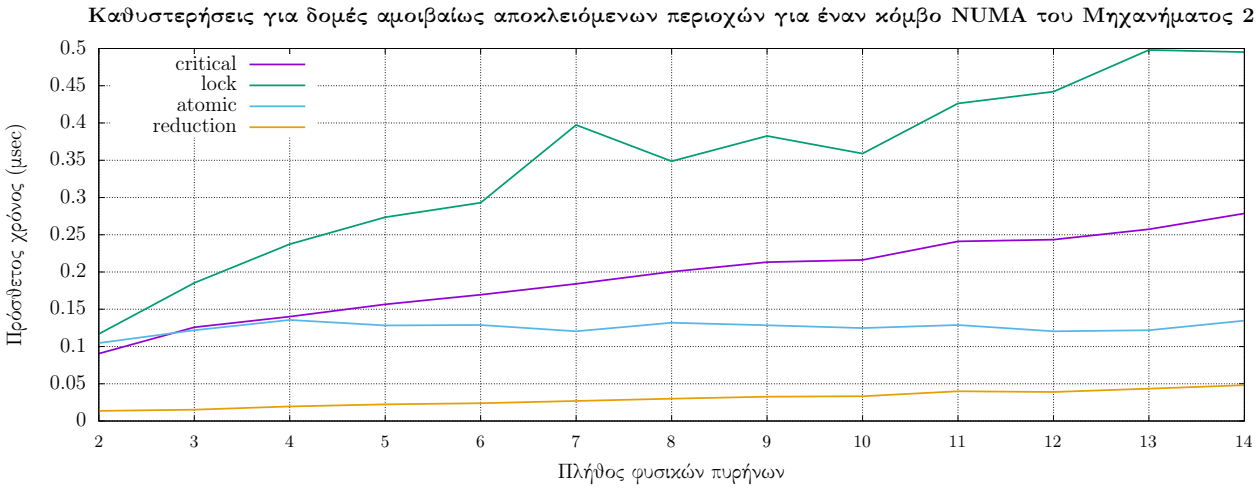
Πρόσθετες καθυστερήσεις για δομές αμοιβαίως αποκλειόμενων περιοχών του OpenMP για όλο το Μηχάνημα 1



Καθυστερήσεις για δομές αμοιβαίως αποκλειόμενων περιοχών για έναν κόμβο NUMA του Μηχανήματος 1







Από τα διαγράμματα συμπεραίνουμε πως η δομή **lock** της βιβλιοθήκης OpenMP προσθέτει τη μεγαλύτερη καθυστέρηση στο χρόνο εκτέλεσης ενός παράλληλου προγράμματος και για τα δύο μηχανήματα. Αντιθέτως, η δομή **omp reduction** επιστρέφει το μικρότερο κόστος.

Με βάση την εκτενή ανάλυση των πρόσθετων καθυστερήσεων της διεπαφής OpenMP μπορεί κανείς να χρησιμοποιήσει τα νούμερα και τα συμπεράσματα για την καλύτερη δόμηση του κώδικά του. Το ερώτημα που παραμένει είναι το πως μπορούν οι τιμές αυτές να εφαρμοστούν στο μοντέλο μας. Μελετώντας τις εξισώσεις (8.1) και (5.9) παρατηρούμε πως οι νόμοι, στους οποίους στηρίχτηκαν τα μοντέλα που αναλύθηκαν, προσθέτουν τις επιπλέον καθυστερήσεις στο τελικό χρόνο εκτέλεσης που υπολογίζεται. Το μοντέλο μας, η κατασκευή του οποίου στηρίζεται σε αυτούς τους νόμους, ακολουθεί την ίδια τακτική.

8.8 Το τελικό μοντέλο

Στο κεφάλαιο αυτό παρουσιάζεται το τελικό μοντέλο για την πρόβλεψη του χρόνου εκτέλεσης ή της επίδοσης ενός παράλληλου συστήματος. Όπως αναφέρθηκε και στην εισαγωγή ένα παράλληλο σύστημα αποτελείται από ένα παράλληλο πρόγραμμα και μια παράλληλη αρχιτεκτονική. Το να κατασκευαστεί ένα μοντέλο πρόβλεψης το οποίο θα βασίζεται στα χαρακτηριστικά του ενός από τα δύο στοιχείων από τα οποία αποτελείται θα οδηγήσει σε λάθος εκτιμήσεις. Για το λόγο αυτό, το κλασικό μοντέλο Roofline το οποίο κατασκευάζεται αποκλειστικά με τα χαρακτηριστικά της αρχιτεκτονικής του συστήματος, δεν μπορεί να χρησιμοποιηθεί για προβλέψεις. Εν αντιθέσει οι νόμοι Amdahl και Gustafson χρησιμοποιούν τα χαρακτηριστικά και των δύο στοιχείων ενός παράλληλου συστήματος και μπορούν να χρησιμοποιηθούν ως μοντέλα πρόβλεψης. Το πρόβλημα είναι ότι υστερούν σε πολυπλοκότητα, καθώς όλες οι πρόσθετες καθυστερήσεις του παράλληλου προγράμματος μοντελοποιούνται με το μη-παραλληλοποιήσιμο μέρος s του κώδικα, ενώ ταυτόχρονα αγνοούν την επίδραση της μνήμης στη συνολική επίδοση. Επίσης, οι νόμοι αυτοί προϋποθέτουν γνώση της δομής του παράλληλου προγράμματος για να εξαχθούν οι τιμές των αριθμών s και f.

Το μοντέλο μας συνδυάζει τα δυνατά σημεία των μοντέλων αυτών. Αρχικά, η βάση του μοντέλου μας είναι το κλασικό μοντέλο Roofline. Το μοντέλο Roofline έχει την ικανότητα με πολύ ευφυή τρόπο να μοντελοποιήσει την καθυστέρηση που προσθέτει το εύρος ζώνης της μνήμης στην επίδοση ενός προγράμματος. Επιπλέον, οι νόμοι Amdahl και Gustafson γενικεύονται στην εξίσωση (5.9) η οποία επιδεικνύει ότι οι επιπλέον καθυστερήσεις, πέραν του εύρους ζώνης της μνήμης, μπορούν απλά να προστεθούν στην τελική πρόβλεψη. Στην περίπτωση της βιβλιοθήκης OpenMP οι καθυστερήσεις αυτές υπολογίστηκαν στην προηγούμενη ενότητα. Προκειμένου κανείς να μπορεί να κατασκευάσει το μοντέλο μας χρειάζεται να ξέρει τέσσερα μεγέθη. Το πρώτο είναι το μέγιστο εύρος ζώνης της μνήμης του συστήματος. Το δεύτερο και τρίτο είναι το πλήθος πράξεων κινητής υποδιαστολής που έγιναν κατά την εκτέλεση του προγράμματος και το σύνολο σε bytes των δεδομένων που μεταφέρθηκαν από και προς τη μνήμη. Από αυτά τα δύο προκύπτει η ένταση λειτουργίας του προγράμματος που εκτελείται. Το τέταρτο μέγεθος είναι ο χρόνος εκτέλεσης της σειριακής εκδοχής του προγράμματος που εκτελείται. Το πλήθος των φυσικών πυρήνων που χρησιμοποιούνται κατά την εκτέλεση της παράλληλης εκδοχής του κώδικα είναι μεταβλητή. Αυτά τα μεγέθη μπορούν να μετρηθούν για ένα πρόγραμμα χωρίς τη χρήση πολλαπλών φυσικών πυρήνων. Τελικά, το μοντέλο μας εκφράζει το χρόνο εκτέλεσης της παράλληλης εκδοχής του προγράμματος με την επόμενη μαθηματική εξίσωση,

$$T(p) = \mathbb{1}_{[0,\text{γόνατο})}(OI) \times \frac{bytes}{bandwidth} + \mathbb{1}_{[\text{γόνατο},\infty)}(OI) \times \frac{T_s}{p} + \mathcal{O}(p) \tag{8.3}$$

όπου $O(p)$ είναι το άθροισμα των καθυστερήσεων για όλα τα πρόσθετα κόστη όταν χρησιμοποιούνται p φυσικοί πυρήνες για την εκτέλεση του προγράμματος. Αναλυτικά,

$$\mathcal{O}(p) = \mathcal{O}_{synchronization}(p) + \mathcal{O}_{mutualexclusion}(p) + \mathcal{O}_{scheduling}(p) \tag{8.4}$$

Επειδή πλέον στο μοντέλο έχουν προστεθεί καθυστερήσεις, ο χαμηλότερος χρόνος εκτέλεσης ενός προγράμματος το οποίο περιορίζεται από το μέγιστο υπολογιστικό κατώφλι θα είναι $\frac{T_s}{p} + \mathcal{O}(p)$.

Έτσι, και το γόνατο του μοντέλου μας θα αλλάξει. Η εξίσωση (8.2) μετατρέπεται στην,

$$\text{γόνατο} = \frac{\text{FLOPs}_{\text{program}}}{\text{bandwidth} \times (\frac{T_s}{p} + \mathcal{O}(p))} \tag{8.5}$$

Στη συνέχεια, δοθέντος ενός παράλληλου συστήματος, αρχιτεκτονικής και προγράμματος, παρουσιάζεται αλγόριθμος για την πρόβλεψη του χρόνου εκτέλεσης της παράλληλης εκδοχής του προγράμματος στην αρχιτεκτονική ή/και της επίδοσης της παράλληλης εκδοχής, χρησιμοποιώντας το μοντέλο μας.

Αλγόριθμος πρόβλεψης του χρόνου εκτέλεσης ή επίδοσης με βάση το μοντέλο μας

Μετρούμε με τη βοήθεια του μετροπρογράμματος STREAM το μέγιστο εύρος ζώνης της κύριας μνήμης, για το μέγιστο πλήθος φυσικών πυρήνων που διαθέτει ένας κόμβος NUMA.

Εκτελούμε τη σειριακή εκδοχή του προγράμματος και μετρούμε το χρόνο εκτέλεσής του, το πλήθος των πράξεων κινητής υποδιαστολής που έγιναν και το σύνολο των δεδομένων σε bytes που μεταφέρθηκαν από και προς την κύρια μνήμη.

Διαιρούμε το πλήθος των πράξεων κινητής υποδιαστολής με το σύνολο των bytes που μεταφέρθηκαν για να υπολογίσουμε την ένταση λειτουργίας του προγράμματος.

Γνωρίζοντας σε ποιά σημεία θα μπούν οι κατάλληλες εντολές ξέρουμε και το πόσες φορές θα εκτελεστεί μια δομή OpenMP. Για μία δομή OpenMP που θα χρησιμοποιηθεί βρίσκουμε την καθυστέρηση που προσθέτει μια κλήση της (όπως παρουσιάστηκε στο κεφάλαιο *Τα πρόσθετα κόστη του OpenMP*) και την πολλαπλασιάζουμε με το πόσες φορές θα εκτελεστεί. Αυτό το κάνουμε για κάθε δομή που θα υπάρξει στην παράλληλη εκδοχή του κώδικα. Αθροίζουμε όλα τα γινόμενα που προκύπτουν και έτσι έχουμε το μέγεθος $\mathcal{O}(p)$.

Χρησιμοποιώντας την εξίσωση (8.5) υπολογίζουμε το γόνατο του μοντέλου μας.

Αν η ένταση λειτουργίας του προγράμματός μας είναι μικρότερη του γονάτου, τότε ο χρόνος εκτέλεσης της παράλληλης εκδοχής θα ισούται με

$$T(p) = \frac{\text{bytes}}{\text{bandwidth}} + \mathcal{O}(p)$$

Διαφορετικά, ο χρόνος εκτέλεσης της παράλληλης εκδοχής θα είναι ίσος με

$$T(p) = \frac{T_s}{p} + \mathcal{O}(p)$$

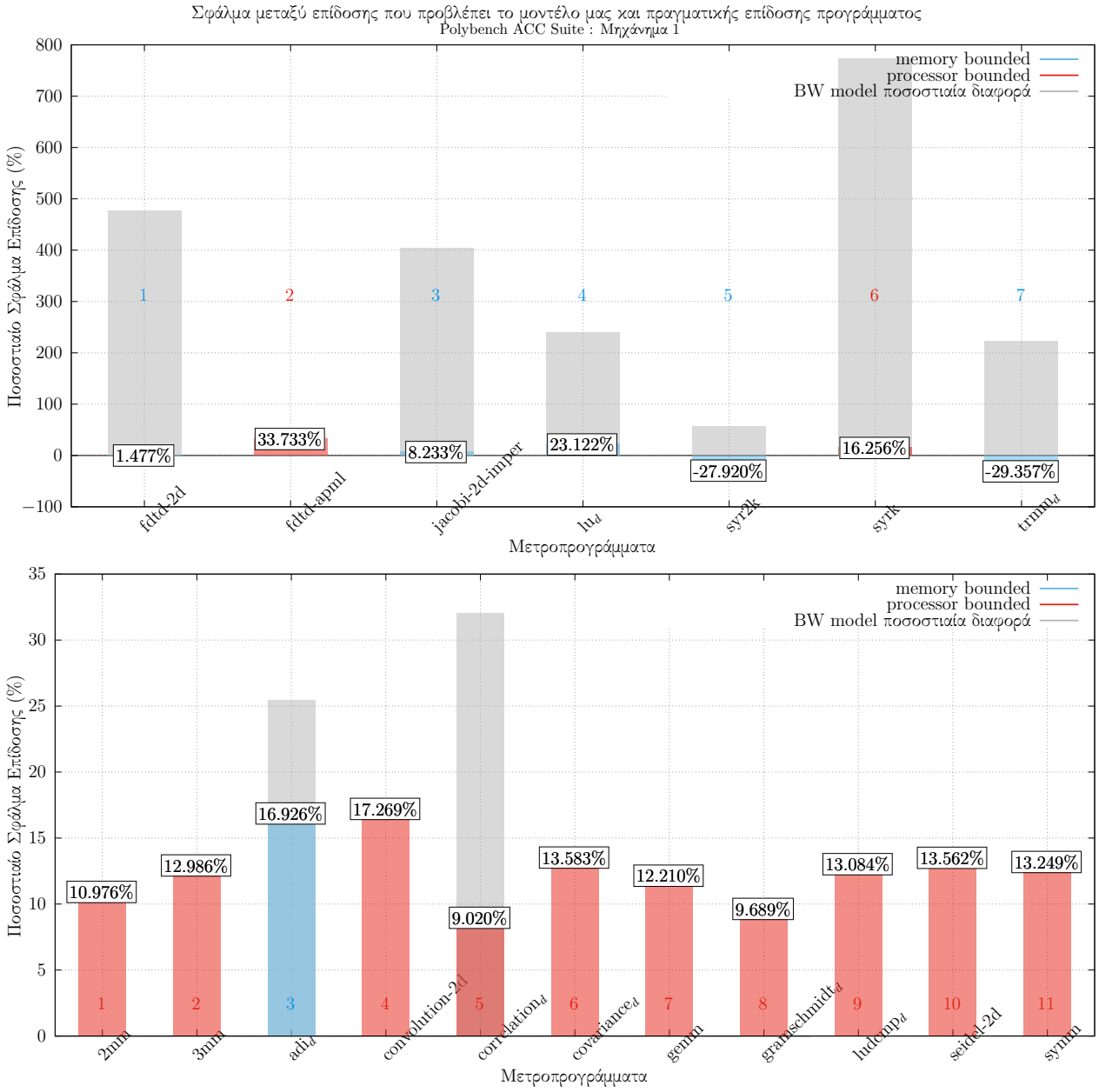
Αν θέλουμε να υπολογίσουμε την επίδοση της παράλληλης εκδοχής διαιρούμε το πλήθος των πράξεων κινητής υποδιαστολής με το χρόνο εκτέλεσης της παράλληλης περιοχής που μετρήσαμε στο αμέσως προηγούμενο βήμα.

Θα εφαρμόσουμε τον αλγόριθμο πρόβλεψης στους τεχνητούς πυρήνες Jacobi. Μελετώντας τη δομή του προγράμματος παρατηρούμε πως η δομή συγχρονισμού **parallel for** επαναλαμβάνεται

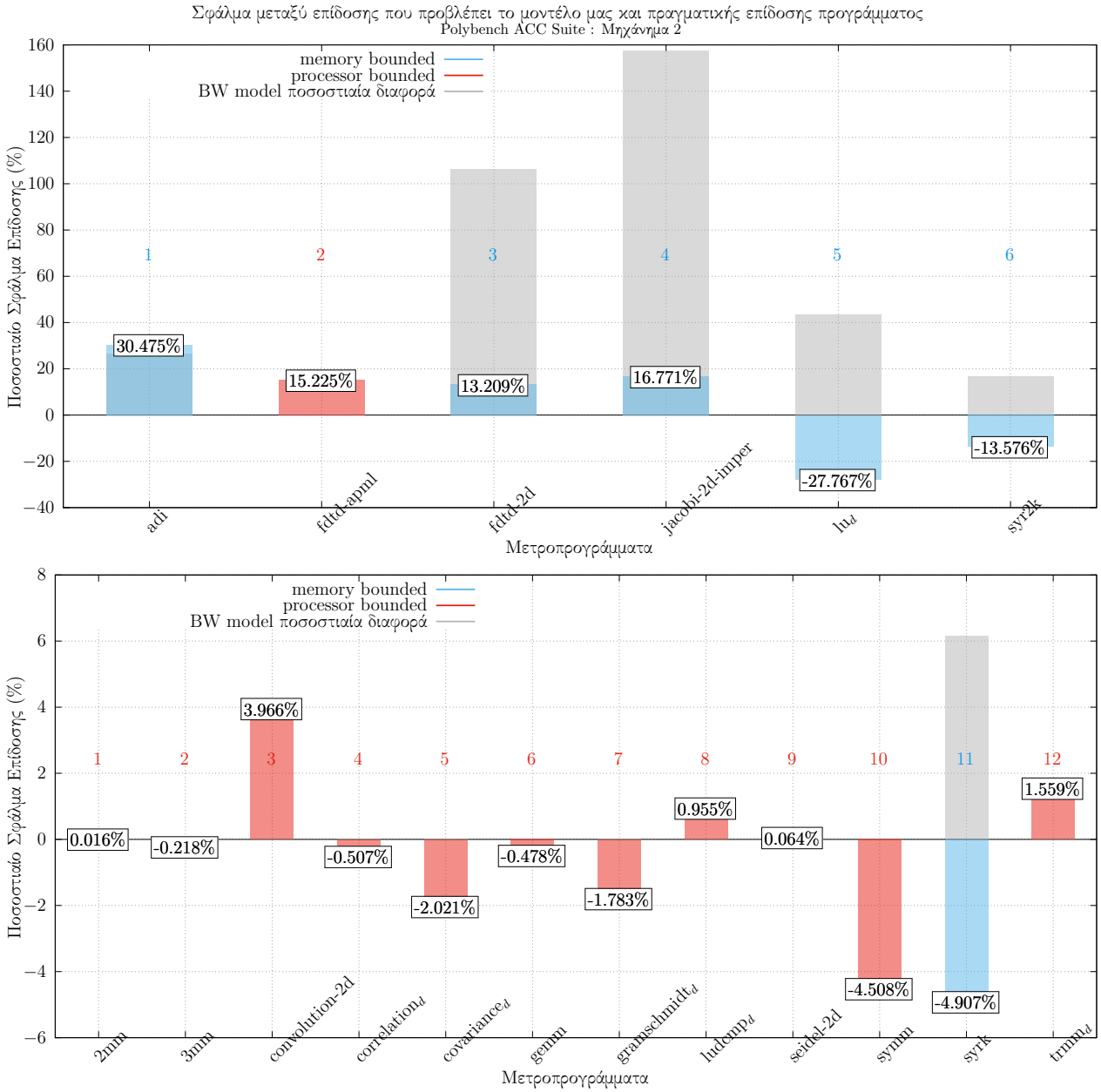
256 φορές. Οι επαναλήψεις της δομής είναι πολύ λίγες προκειμένου να παρατηρηθούν σημαντικές αλλαγές στις προβλέψεις του τελικού μοντέλου σε σύγκριση με το αρχικό μοντέλο. Δηλαδή, τα συνολικά πρόσθετα κόστη είναι $\mathcal{O}(p) = 256 \times o_{\text{parallel for}}(p)$, όπου $o_{\text{parallel for}}(p)$, είναι ο χρόνος που προσθέτει η κλήση της δομής **parallel for** μόνο μία φορά.

8.8.1 Η αποτελεσματικότητα σε προβλέψεις του μοντέλου μας

Έχοντας πλέον έναν αλγόριθμο που κατασκευάζει το μοντέλο μας, θα πρέπει να μελετηθεί το πόσο ικανοποιητικές είναι οι προβλέψεις του και σε άλλα προγράμματα. Για το λόγο αυτό θα χρησιμοποιήσουμε τα μετροπρογράμματα της σουίτας PolyBench-ACC. Για όλα τα μετροπρογράμματα εμφανίζονται μόνο δομές OpenMP για το συγχρονισμό των νημάτων. Επομένως, τα συνολικά πρόσθετα κόστη θα έχουν τη μορφή $\mathcal{O}(p) = \mathcal{O}_{\text{synchronization}}(p)$. Στη συνέχεια παρουσιάζονται σε διαγράμματα τα ποσοστά σφάλματος για το χρόνο εκτέλεσης που προβλέπει το μοντέλο μας ως προς τον πραγματικό χρόνο εκτέλεσης του κάθε μετροπρογράμματος. Και σε αυτή τη μελέτη μας ενδιαφέρει να ικανοποιηθούν οι βασικές υποθέσεις του μοντέλου μας. Τα datasets που χρησιμοποιούνται έχουν μέγεθος πολύ μεγαλύτερο του μεγέθους της κρυφής μνήμης τελευταίου επιπέδου (LLC) σε κάθε κόμβο NUMA για τα δύο μηχανήματα του εργαστηρίου.



Σχήμα 8.9: Τα ποσοστιαία σφάλματα πρόβλεψης του μοντέλου μας για το Μηχάνημα 1 σε 8 φυσικούς πυρήνες και με πολύ μεγάλο μέγεθος dataset



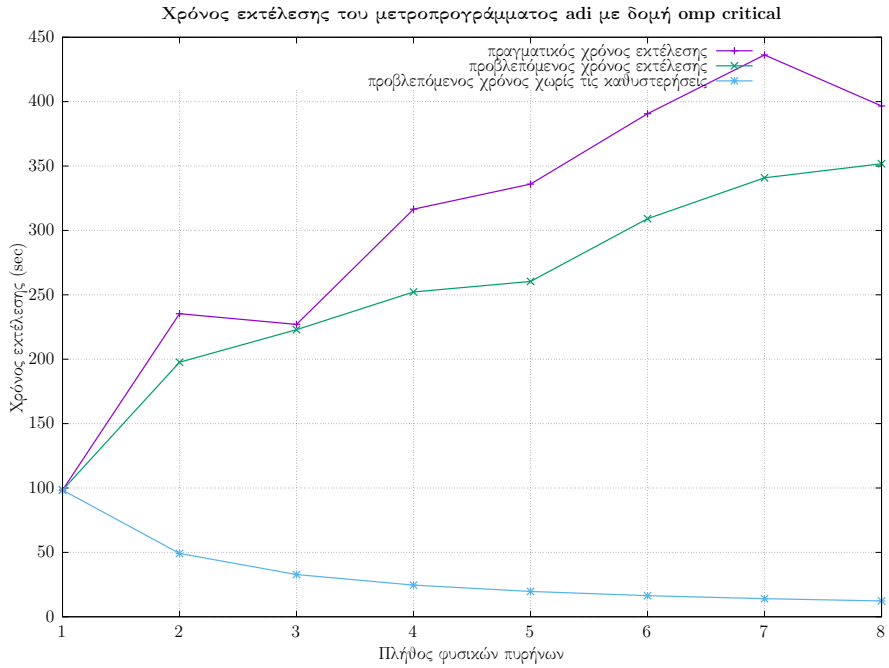
Σχήμα 8.10: Τα ποσοστιαία σφάλματα πρόβλεψης του μοντέλου μας για το Μηχάνημα 2 σε 14 φυσικούς πυρήνες και με πολύ μεγάλο μέγεθος dataset

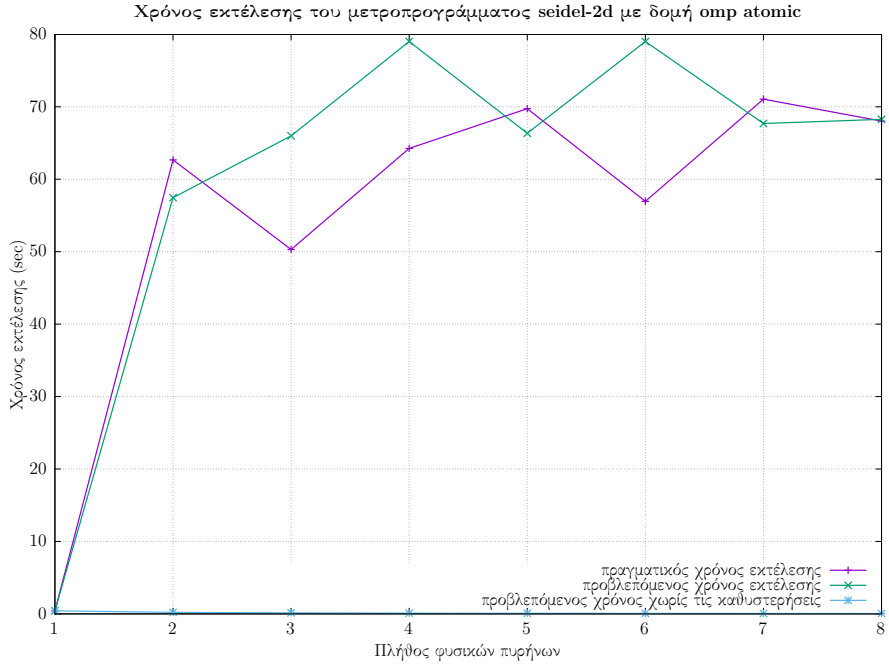
Στα διαγράμματα το ποσοστιαίο σφάλμα πρόβλεψης του μοντέλου μας για κάθε μετροπρόγραμμα απεικονίζεται ως μια χρωματισμένη μπάρα με ύψος όσο και το ποσοστιαίο λάθος. Επίσης, καταγράφονται τα σφάλματα με ετικέτες για μια καλύτερη ιδέα της τιμής τους. Οι μπάρες είναι χρωματισμένες σύμφωνα με το σε ποια περιοχή λειτουργεί το κάθε μετροπρόγραμμα. Αν ένα

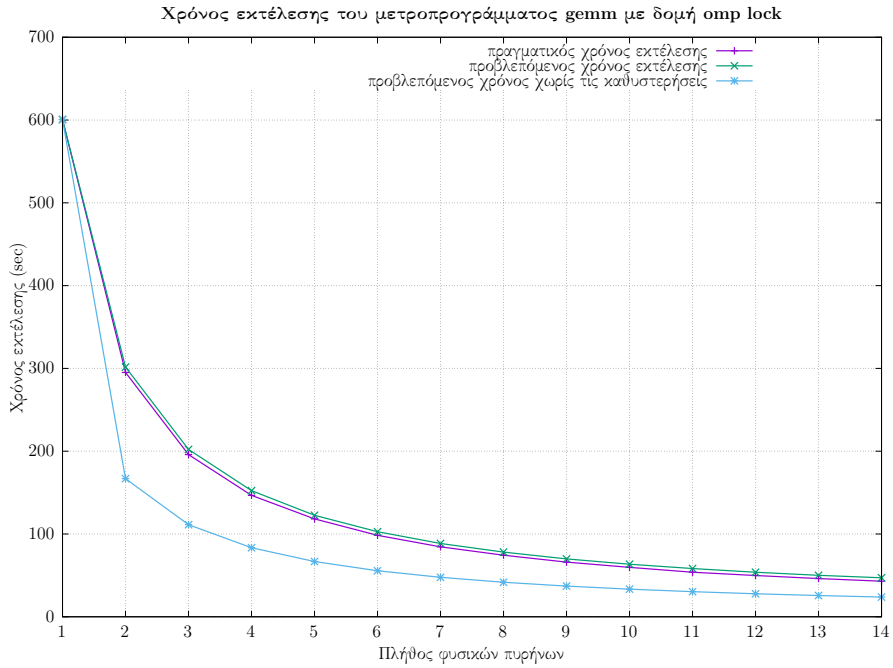
μετροπρόγραμμα λειτουργεί στην περιοχή η οποία περιορίζεται από το κατώφλι της μέγιστης υπολογιστικής επίδοσης τότε η μπάρα του θα έχει ανοιχτό κόκκινο χρώμα. Αν λειτουργεί στην περιοχή που περιορίζεται από το μέγιστο εύρος ζώνης τότε το χρώμα της μπάρας θα είναι γαλάζιο. Επίσης, αν η μπάρα δεν είναι εμφανής λόγο του πολύ μικρού σφάλματος υπάρχουν και οι αριθμοί που είναι χρωματισμένοι και λειτουργούν ως δείκτες για κάθε μετροπρόγραμμα. Το χρώμα κάθε αριθμού ακολουθεί τον ίδιο κανόνα με το χρώμα κάθε μπάρας.

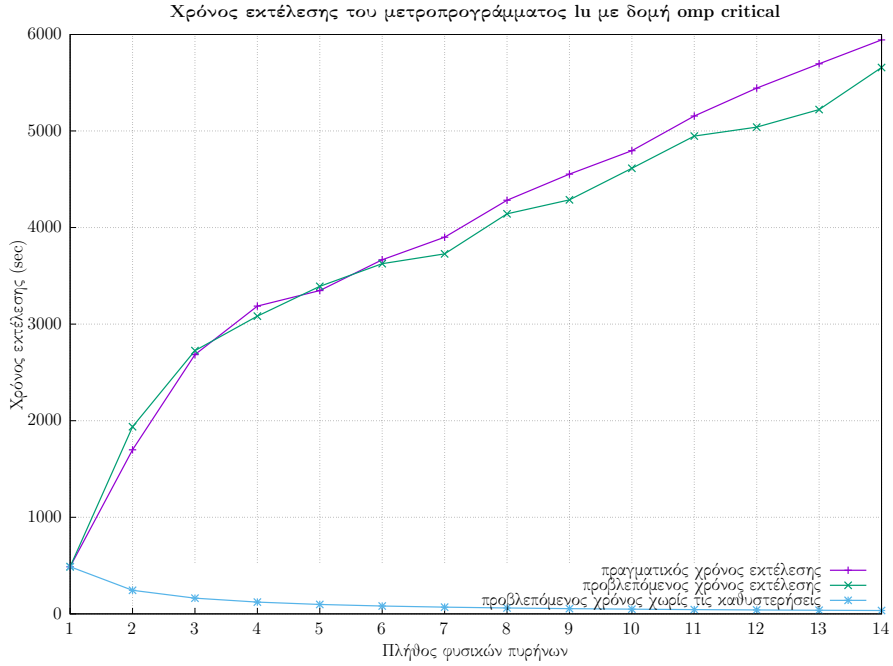
Από τα διαγράμματα συμπεραίνουμε πως το μοντέλο μας για όλα τα μετροπρογράμματα επιστρέφει πολύ ικανοποιητικές προβλέψεις για την επίδοσή τους με μικρά σφάλματα. Για την εξακρίβωση της αξιοπιστίας των αποτελεσμάτων στα διαγράμματα παρουσιάζεται με τις γκρι μπάρες το μοντέλο εύρους ζώνης. Το μοντέλο εύρους ζώνης υπολογίζει την επιτάχυνση ενός παράλληλου προγράμματος με τη χρήση του ενεργού εύρους ζώνης (effective bandwidth). Το ενεργό εύρος ζώνης ενός προγράμματος ισούται με το πλήθος των bytes που μεταφέρθηκαν από και προς την κύρια μνήμη σε ένα δευτερόλεπτο. Η επιτάχυνση ισούται με τη διαίρεση του μέγιστου εύρους ζώνης για έναν κόμβο NUMA που μετρήθηκε με το μετροπρόγραμμα STREAM προς το ενεργό εύρος ζώνης. Το ύψος μιας γκρι μπάρας ισούται με την ποσοστιαία διαφορά της επίδοσης που προβλέπει το μοντέλο εύρους ζώνης και της επίδοσης που προβλέπει το μοντέλο μας. Τελικά, χρησιμοποιώντας και ως αναφορά το μοντέλο εύρους ζώνης μνήμης συμπεραίνουμε ότι οι προβλέψεις του μοντέλου μας είναι αξιόπιστες.

Στη συνέχεια προσθέτουμε σε τέσσερα μετροπρογράμματα της σουίτας PolyBench-ACC δομές της βιβλιοθήκης OpenMP για τον έλεγχο της πρόσβασης κοινόχρηστων δεδομένων. Θέλουμε να ελέγξουμε την αποδοτικότητα του μοντέλου μας και σε περιπτώσεις με άλλα πρόσθετα κόστη. Η πρόβλεψη του χρόνου εκτέλεσης μαζί με τον πραγματικό χρόνο εκτέλεσης απεικονίζεται στα επόμενα διαγράμματα.


Χρόνος εκτέλεσης του μετροπρογράμματος adi με δομή omp critical
πραγματικός χρόνος εκτέλεσης
προβλεπόμενος χρόνος εκτέλεσης
προβλεπόμενος χρόνος χωρίς τις καθυστερήσεις
Χρόνος εκτέλεσης (sec)
Πλήθος φυσικών πυρήνων
0
50
100
150
200
250
300
350
400
450
1
2
3
4
5
6
7
8


Χρόνος εκτέλεσης του μετροπρογράμματος seidel-2d με δομή omp atomic
πραγματικός χρόνος εκτέλεσης
προβλεπόμενος χρόνος εκτέλεσης
προβλεπόμενος χρόνος χωρίς τις καθυστερήσεις
Χρόνος εκτέλεσης (sec)
Πλήθος φυσικών πυρήνων
0
10
20
30
40
50
60
70
80
1
2
3
4
5
6
7
8

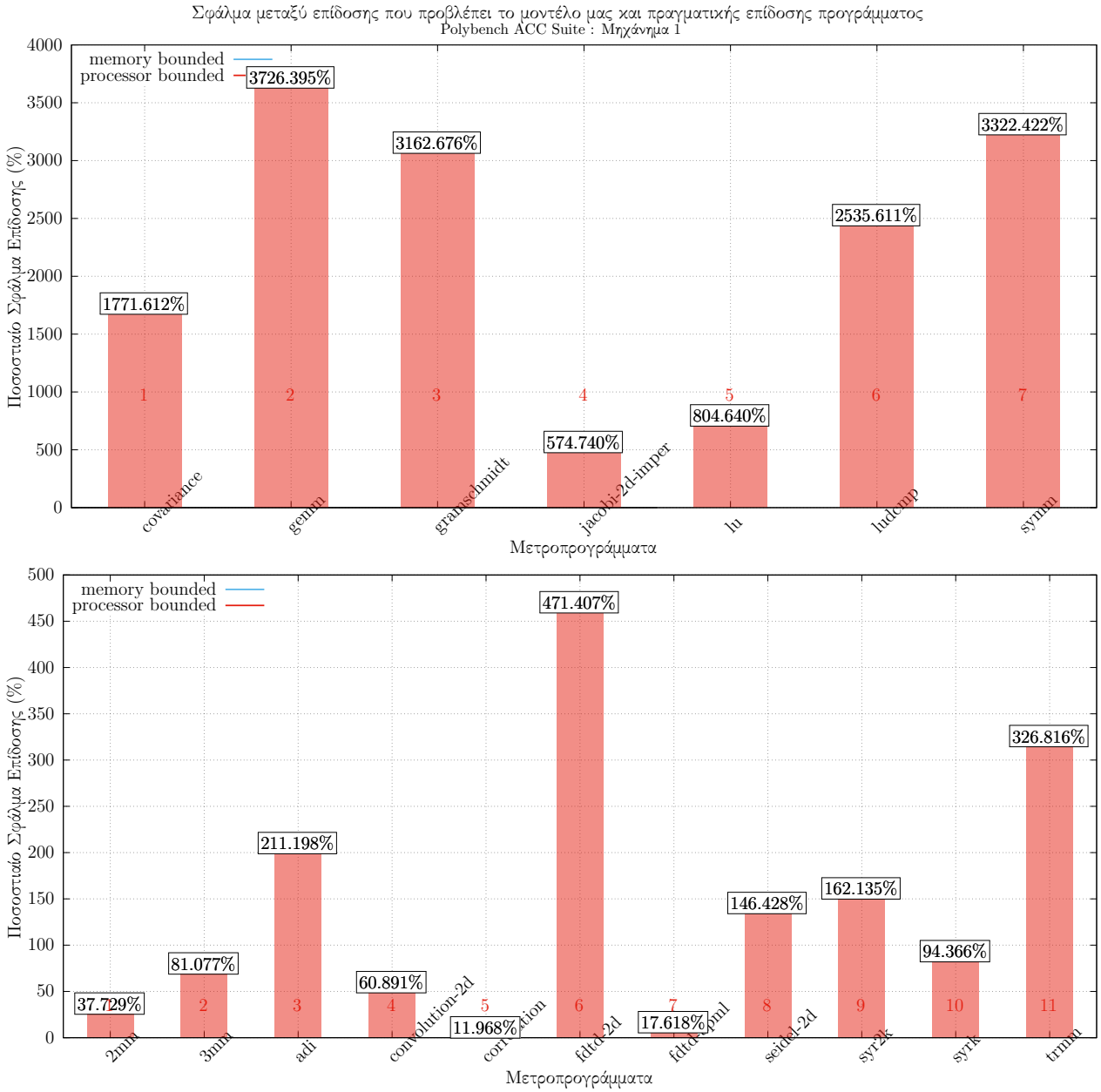

Χρόνος εκτέλεσης του μετροπρογράμματος gemm με δομή omp lock
πραγματικός χρόνος εκτέλεσης
προβλεπόμενος χρόνος εκτέλεσης
προβλεπόμενος χρόνος χωρίς τις καθυστερήσεις
Χρόνος εκτέλεσης (sec)
Πλήθος φυσικών πυρήνων
0
100
200
300
400
500
600
700
1
2
3
4
5
6
7
8
9
10
11
12
13
14


Χρόνος εκτέλεσης του μετροπρογράμματος lu με δομή omp critical
πραγματικός χρόνος εκτέλεσης
προβλεπόμενος χρόνος εκτέλεσης
προβλεπόμενος χρόνος χωρίς τις καθυστερήσεις
Χρόνος εκτέλεσης (sec)
Πλήθος φυσικών πυρήνων
0
1000
2000
3000
4000
5000
6000
1
2
3
4
5
6
7
8
9
10
11
12
13
14

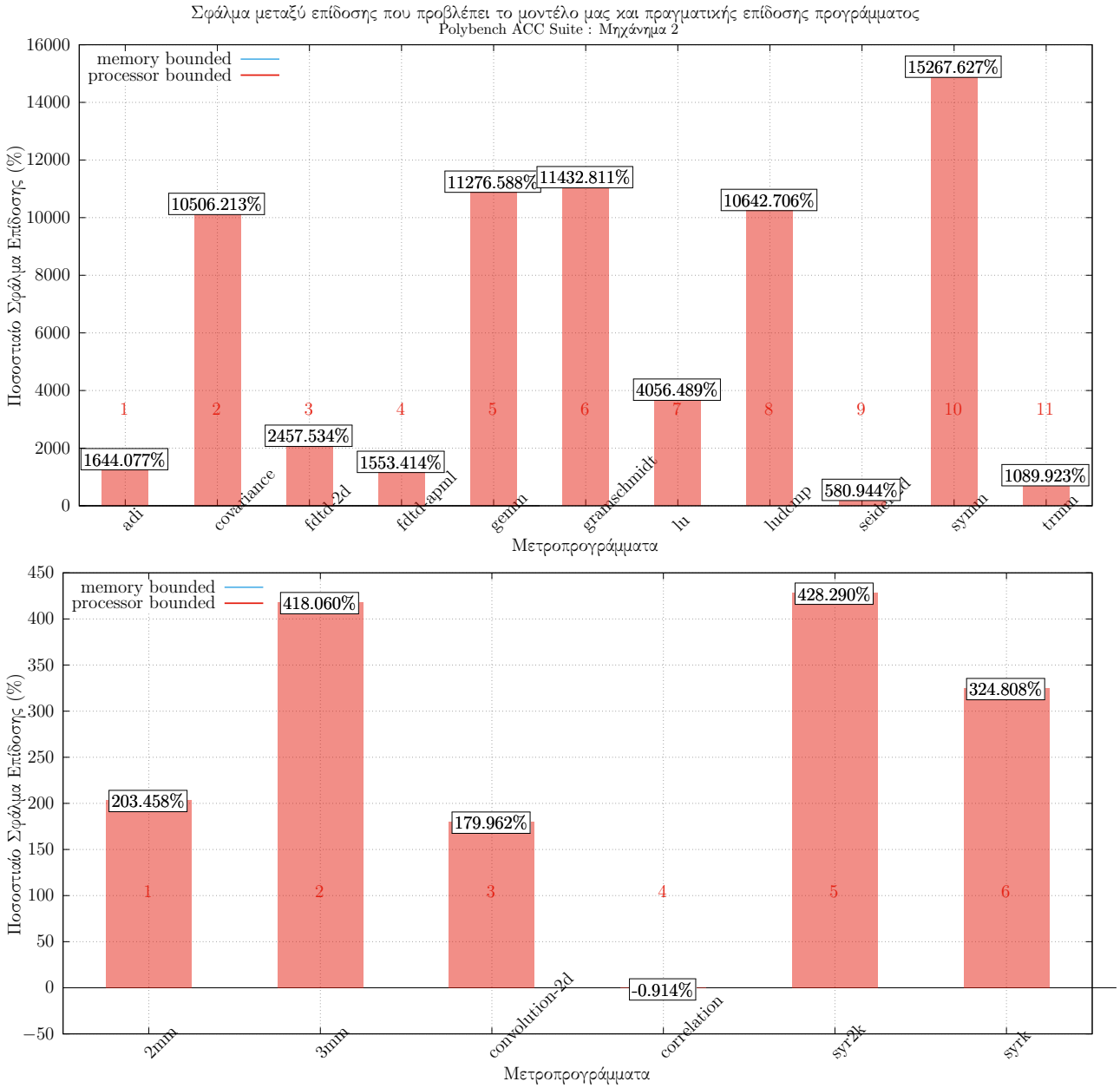
8.9 Αδυναμίες του μοντέλου μας

Το τελικό μας μοντέλο αν και αποδίδει ικανοποιητικά στις προβλέψεις του, έχει αδυναμίες. Αυτές προκύπτουν κυρίως από τους περιορισμούς που τέθηκαν ως βασικές υποθέσεις του μοντέλου. Στο κεφάλαιο αυτό θα μελετηθούν οι αδυναμίες αυτές και πιθανές λύσεις τους για μελλοντική έρευνα.

1. Τα μικρού μεγέθους datasets. Με βάση την πρώτη υπόθεση του μοντέλου μας περιοριζόμαστε στην πρόβλεψη επίδοσης ενός παράλληλου προγράμματος για μεγάλου μεγέθους datasets προκειμένου να μην αποθηκευτεί μέρος αυτών στις κρυφές μνήμες ενός κόμβου NUMA. Οι κρυφές μνήμες με τα πρωτόκολλα συνοχής αυξάνουν την πολυπλοκότητα για το πως και που αποθηκεύονται τα δεδομένα ενός προγράμματος, πράγμα που καθιστά πολύ δύσκολη τη μοντελοποίηση των προσβάσεων δεδομένων στην ιεραρχία μνήμης. Στη συνέχεια παρουσιάζονται σε διάγραμμα ξανά τα μετροπρογράμματα της σουίτας PolyBench-ACC αλλά αυτή τη φορά με μικρού μεγέθους datasets. Όπως βλέπουμε το μοντέλο μας αστοχεί στις προβλέψεις του σχεδόν για όλα τα μετροπρογράμματα και στα δύο μηχανήματα και με μεγάλο σφάλμα. Η κακή επίδοση του μοντέλου προκύπτει από τα πρωτόκολλα συνοχής των κρυφών μνημών. Επειδή το μέγεθος εισόδου είναι αρκετά μικρό, τα δεδομένα του προγράμματος μπορούν να αποθηκευτούν στις κρυφές μνήμες χαμηλού επιπέδου του κάθε φυσικού πυρήνα. Αυτό οδηγεί σε επιτάχυνση του χρόνου εκτέλεσης της σειριακής εκδοχής του κώδικα. Από την άλλη όμως, οδηγεί σε πρόσθετες καθυστερήσεις συνοχής κρυφών μνημών για την παράλληλη εκδοχή, διότι στον υπολογισμό συμμετέχουν παραπάνω από έναν πυρήνα. Όσο αυξάνεται το πλήθος των φυσικών πυρήνων που συμμετέχουν στον υπολογισμό θα αυξάνονται και οι καθυστερήσεις.



Σχήμα 8.11: Τα ποσοστιαία σφάλματα στις προβλέψεις του μοντέλου μας για μικρού μεγέθους datasets και 8 φυσικούς πυρήνες για το Μηχάνημα 1.

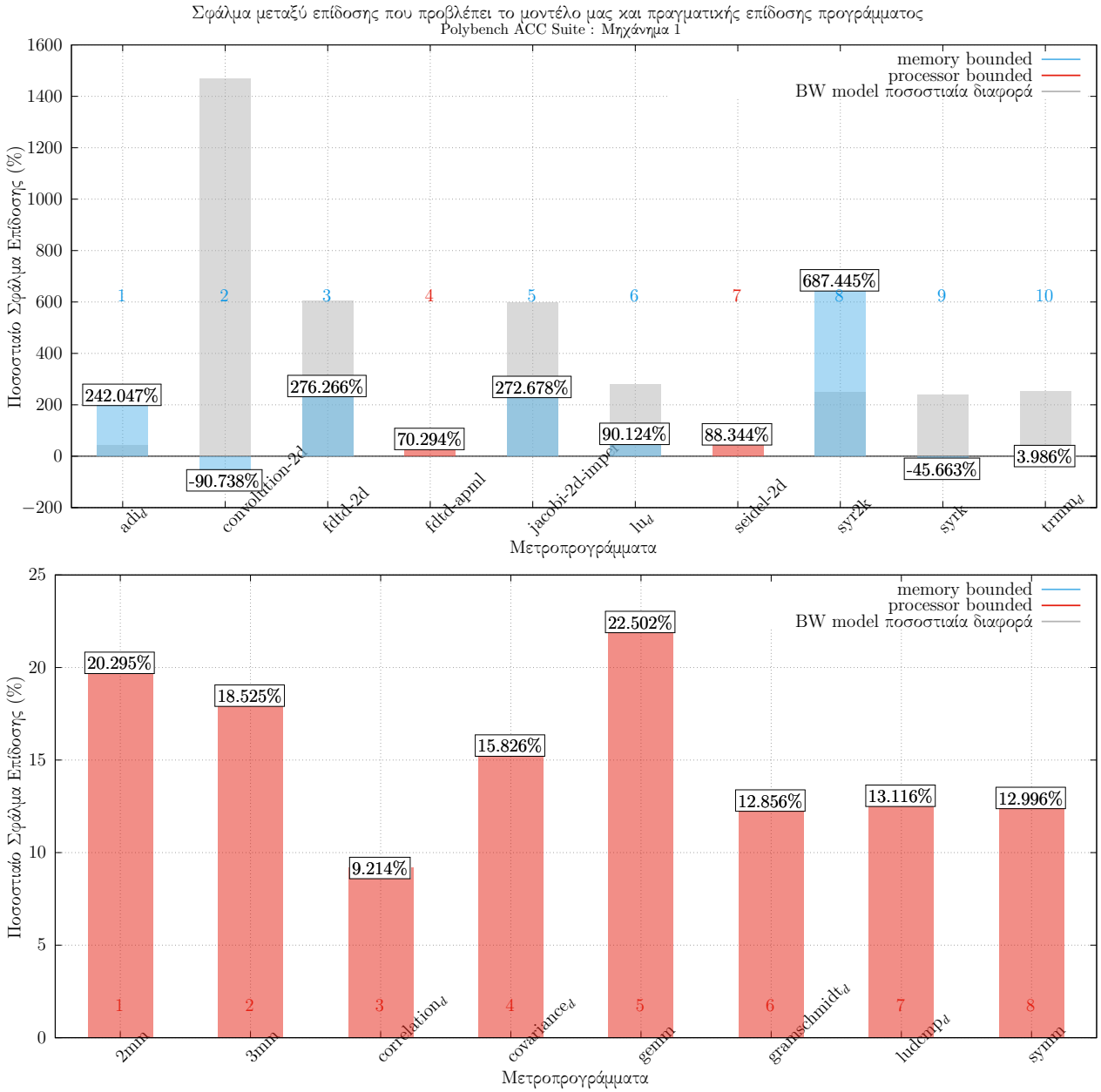


Σχήμα 8.12: Τα ποσοστιαία σφάλματα στις προβλέψεις του μοντέλου μας για μικρού μεγέθους datasets και 14 φυσικούς πυρήνες για το Μηχάνημα 2.

2. Σύμφωνα με τη δεύτερη υπόθεση του μοντέλου μας περιοριστήκαμε στους φυσικούς πυρήνες ενός επεξεργαστή. Αποφεύχθηκαν τα νήματα υλισμικού. Ο λόγος για τον οποίο έγινε αυτό είναι επειδή η χρήση τεχνολογιών, όπως Hyperthreading, αυξάνουν την πιθανότητα εμφάνισης του φαινομένου false sharing μεταξύ των νημάτων του ίδιου πυρήνα και την εμφάνιση

race conditions για τους κοινόχρηστους αριθμητικούς καταχωρητές τους όταν χρειάζεται να υπολογιστεί κάτι. Και τα δύο φαινόμενα είναι τυχαία και ο βαθμός που επηρεάζουν ένα παράλληλο πρόγραμμα αλλάζει σε κάθε εκτέλεσή του. Το μοντέλο μας επομένως αδυνατεί να προβλέψει σωστά όταν εμφανίζονται αυτά τα δύο φαινόμενα σε ένα παράλληλο σύστημα. Μια πιθανή λύση είναι η ανάπτυξη ενός στοχαστικού μοντέλου πρόβλεψης της επίδοσης ενός παράλληλου συστήματος το οποίο θα έχει ως θεμέλιο το μοντέλο μας και στο οποίο θα προστίθενται ως τυχαίες μεταβλητές καθυστέρησης.

3. Μια άλλη αδυναμία που έχει το μοντέλο μας είναι ότι έχει κατασκευαστεί με βάση μόνο έναν κόμβο NUMA. Αυτό φαίνεται εμμέσως από την τρίτη βασική υπόθεση του μοντέλου μας όπου χρησιμοποιούνται κατά την εκτέλεση ενός προγράμματος μόνο οι διευθύνσεις μνήμης στις οποίες έχει τοπική πρόσβαση ο κόμβος NUMA στον οποίο εκτελείται το πρόγραμμα. Η χρήση και των υπόλοιπων κόμβων NUMA για την εκτέλεση του προγράμματος προσθέτει επιπλέον καθυστερήσεις για την επικοινωνία μεταξύ τους και αυξάνει την πιθανότητα για ανισόρροπης κατανομή φορτίου. Όπως φαίνεται από το επόμενο διάγραμμα το μοντέλο μας στο Μηχάνημα 1 για δύο κόμβους NUMA δεν αποδίδει το ίδιο καλά.



Σχήμα 8.13: Τα ποσοστιαία σφάλματα στις προβλέψεις του μοντέλου μας για μεγάλου μεγέθους datasets και 16 φυσικούς πυρήνες για το Μηχάνημα 1 στους κόμβους NUMA 0 και 2.

4. Στο μοντέλο μας το μέγιστο υπολογιστικό κατώφλι είναι η επίδοση ενός προγράμματος μετά από τη γραμμική κλιμάκωσή του στο μέγιστο πλήθος φυσικών πυρήνων που του διαθέτουμε. Συνεπώς, το μοντέλο μας δεν μπορεί να καλύψει την πιθανότητα υπεργραμμικής κλιμάκωσης ενός προγράμματος. Για τέτοιου είδους παράλληλα προγράμματα οι προβλέψεις από το

μοντέλο μας για τους χρόνους εκτέλεσης θα είναι μεγαλύτερες του πραγματικών χρόνων εκτέλεσης.

5. Το μοντέλο μας έχει κατασκευαστεί με βάση τον πυρήνα Jacobi, που υλοποιείται στον παράλληλο κώδικα με τη χρήση της δομής **omp parallel for**. Τα μετροπρογράμματα που χρησιμοποιήθηκαν δεν έχουν την ίδια δομή με αυτή του πυρήνα αλλά ο κοινός χώρος εργασίας των νημάτων αποτελείται από μία ή πολλές επαναλήψεις for. Σε πιο περίπλοκους γράφους παράλληλης εκτέλεσης που υλοποιούνται με τη δομή **omp parallel sections** το μοντέλο μας μπορεί να μην αποδίδει το ίδιο καλά.

Κεφάλαιο 9

Συμπεράσματα

Στην παρούσα διπλωματική εργασία ο στόχος ήταν να δημιουργηθεί ένα μοντέλο πρόβλεψης το οποίο μπορεί να συμβουλευτεί ένας μηχανικός ανά πάσα στιγμή προκειμένου να μάθει το πως θα αποδώσει ένα πρόγραμμα αν παραλληλοποιηθεί και εκτελεστεί σε ένα μηχάνημα της επιλογής του. Για την κατασκευή του μοντέλου μελετήθηκαν εκτενώς οι νόμοι κλιμακωσιμότητας των παράλληλων συστημάτων επεξεργασίας. Επίσης, αναλύθηκαν τα αίτια που προσθέτουν καθυστερήσεις στο χρόνο εκτέλεσης ενός παράλληλου προγράμματος. Με βάση όλα αυτά και χρησιμοποιώντας ως αρχή το κλασικό μοντέλο Roofline, θεμελιώσαμε μαθηματικά το μοντέλο μας. Με τα πειράματα αποδείχθηκε η αξιοπιστία του να κάνει καλές προβλέψεις όταν τηρούνται οι βασικές του υποθέσεις. Όταν δεν τηρούνται οι αρχικές υποθέσεις του, το μοντέλο μας αδυνατεί να βγάλει έγκυρα συμπεράσματα. Παρόλα αυτά η μαθηματική του διατύπωση και ο αλγόριθμος κατασκευής του αποτελούν καλά θεμέλια για την επέκτασή του προς τις κατευθύνσεις που αδυνατεί να αποδώσει καλά.

Μια σημαντική διαπίστωση για το μοντέλο μας προκύπτει από την εξίσωση (8.3). Αν και η εξίσωση αυτή χρησιμοποιήθηκε σε όλη τη διάρκεια της διπλωματικής ως συνάρτηση για τον υπολογισμό του χρόνου εκτέλεσης του παράλληλου κώδικα για p πλήθος φυσικών πυρήνων, μπορεί να χρησιμοποιηθεί για τον υπολογισμό οποιουδήποτε μεγέθους μας ενδιαφέρει. Δηλαδή, μπορεί κανείς να προβλέψει τα χαρακτηριστικά που απαιτούνται για μια παράλληλη αρχιτεκτονική ώστε ένα πρόγραμμα να πετύχει συγκεκριμένο χρόνο εκτέλεσης ή επίδοση. Ταυτόχρονα, με την εξίσωση αυτή, μπορούν να συγκριθούν διάφορα παράλληλα συστήματα ως προς την επίδοση ενός προγράμματος και να επιλεχθεί το καλύτερο.

Τέλος, το μοντέλο μας είναι ένα σημαντικό εργαλείο για ένα μηχανικό προκειμένου να μειώσει το χρόνο εργασίας του, ανεξαρτήτως του πως θα χρησιμοποιηθεί. Ο επιπλέον χρόνος μπορεί να αφιερωθεί στην ανάπτυξη ποιοτικότερου παράλληλου κώδικα ή σε άλλα μέρη ενός μεγάλου project λογισμικού. Για τέτοια έργα, αυτό σημαίνει πως τα χρηματικά κέρδη θα είναι μεγαλύτερα. Παράλληλα, αν σε ένα έργο απαιτείται η κατασκευή ενός παράλληλου συστήματος που θα πετυχαίνει συγκεκριμένη επίδοση, το μοντέλο μας μπορεί να χρησιμοποιηθεί ως μοντέλο σύγκρισης για τις διαφορετικές παράλληλες αρχιτεκτονικές που μελετώνται προς κατασκευή. Η αρχιτεκτονική που πλησιάζει τις προδιαγραφές του έργου με βάση το μοντέλο μας είναι και η καταλληλότερη. Με αυτό τον τρόπο μειώνεται το κόστος κατασκευής ενός παράλληλου συστήματος επεξεργασίας.

Βιβλιογραφία

[1] L. VALIANT, "Chapter 18 - general purpose parallel architectures", in *Algorithms and Complexity*, ser. Handbook of Theoretical Computer Science, J. VAN LEEUWEN, Ed., Amsterdam: Elsevier, 1990, pp. 943–971, ISBN: 978-0-444-88071-0. DOI: https://doi.org/10.1016/B978-0-444-88071-0.50023-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444880710500230.

[2] S. Fortune and J. Wyllie, "Parallelism in random access machines", *Proceedings of the tenth annual ACM symposium on Theory of computing*, 1978.

[3] P. B. Gibbons, "A more practical pram model", in *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, 1989, pp. 158–168.

[4] M. J. Flynn and K. W. Rudd, "Parallel architectures", *ACM computing surveys (CSUR)*, vol. 28, no. 1, pp. 67–70, 1996.

[5] R. Duncan, "A survey of parallel computer architectures", *Computer*, vol. 23, pp. 5–16, 1990.

[6] A. W. Wilson, "Hierarchical cache/bus architecture for shared memory multiprocessors", in *International Symposium on Computer Architecture*, 1987.

[7] B. Nitzberg and V. M. Lo, "Distributed shared memory: A survey of issues and algorithms", *Computer*, vol. 24, pp. 52–60, 1991.

[8] J. Protic, M. Tomasevic, and V. M. Milutinovic, "Distributed shared memory: Concepts and systems", *IEEE Parallel Distributed Technol. Syst. Appl.*, vol. 4, pp. 63–71, 1997.

[9] L. Dagum and R. Menon, "Openmp: An industry standard api for shared-memory programming", *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[10] K. Datta, M. Murphy, V. Volkov, *et al.*, "Stencil computation optimization and autotuning on state-of-the-art multicore architectures", *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2008.

[11] J. M. Paul and B. H. Meyer, "Amdahl's law revisited for single chip systems", *International Journal of Parallel Programming*, vol. 35, pp. 101–123, 2007.

[12] X.-h. Sun and Y. Chen, "Reevaluating amdahl's law in the multicore era", *J. Parallel Distributed Comput.*, vol. 70, pp. 183–188, 2010.

[13] A. H. Karp and H. P. Flatt, "Measuring parallel processor performance", *Commun. ACM*, vol. 33, pp. 539–543, 1990.

[14] M. D. Hill, "Amdahl's law in the multicore era", *Computer*, vol. 41, 2008.

[15] J. L. Gustafson, "Reevaluating amdahl's law", *Commun. ACM*, vol. 31, pp. 532–533, 1988.

[16] A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the scalability of parallel algorithms and architectures", *IEEE Parallel & Distributed Technology: Systems & Applications*, vol. 1, pp. 12–21, 1993.

[17] V. P. Kumar and A. Gupta, "Analyzing scalability of parallel algorithms and architectures", *J. Parallel Distributed Comput.*, vol. 22, pp. 379–391, 1994.

[18] M. K. Bane and G. D. Riley, "Extended overhead analysis for openmp (research note)", in *Euro-Par*, 2002.

[19] J. M. Bull, "A hierarchical classification of overheads in parallel programs", in *Software Engineering for Parallel and Distributed Systems*, 1996.

[20] V. Cuppu, B. Jacob, B. T. Davis, and T. N. Mudge, "A performance comparison of contemporary dram architectures", *Proceedings of the 26th International Symposium on Computer Architecture (Cat. No.99CB36367)*, pp. 222–233, 1999.

[21] J. Torrellas, M. S. Lam, and J. L. Hennessy, "False sharing and spatial locality in multiprocessor caches", *IEEE Trans. Computers*, vol. 43, pp. 651–663, 1994.

[22] W. J. Bolosky and M. L. Scott, "False sharing and its effect on shared memory performance", 1993.

[23] J.-H. Chow and V. Sarkar, "False sharing elimination by selection of runtime scheduling parameters", *Proceedings of the 1997 International Conference on Parallel Processing (Cat. No.97TB100162)*, pp. 396–403, 1997.

[24] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova, "A case for numa-aware contention management on multicore systems", *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 557–558, 2010.

[25] S. Williams, L. Ionkov, and M. Lang, "Numa distance for heterogeneous memory", in *Proceedings of the Workshop on Memory Centric Programming for HPC*, ser. MCHPC'17, Denver, CO, USA: Association for Computing Machinery, 2017, pp. 30–34, ISBN: 9781450351317. DOI: 10.1145/3145617.3145620. [Online]. Available: https://doi.org/10.1145/3145617.3145620.

[26] N. S. Sattar and S. M. Arifuzzaman, "Overcoming mpi communication overhead for distributed community detection", *Communications in Computer and Information Science*, 2018.

[27] S. Balsamo, L. Donatiello, and N. Van Dijk, "Bound performance models of heterogeneous parallel processing systems", *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 10, pp. 1041–1056, 1998. DOI: 10.1109/71.730531.

[28] S. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insightful visual performance model for multicore architectures", *Commun. ACM*, vol. 52, pp. 65–76, 2009.

[29] D. Burger, J. R. Goodman, and A. Kägi, "Memory bandwidth limitations of future microprocessors", *23rd Annual International Symposium on Computer Architecture (ISCA'96)*, pp. 78–78, 1996.

[30] M. B. Radulovic, K. Asifuzzaman, P. M. Carpenter, P. Radojkovic, and E. Ayguadé, "Hpc benchmarking: Scaling right and looking beyond the average", in *European Conference on Parallel Processing*, 2018.

[31] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments", *2010 39th International Conference on Parallel Processing Workshops*, pp. 207–216, 2010.

[32] S. Che, M. Boyer, J. Meng, *et al.*, "Rodinia: A benchmark suite for heterogeneous computing", *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 44–54, 2009.

[33] P. Luszczek, J. J. Dongarra, D. Koester, *et al.*, "Introduction to the hpc challenge benchmark suite", Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2005.

[34] R. Dolbeau, "Theoretical peak flops per instruction set: A tutorial", *The Journal of Supercomputing*, vol. 74, pp. 1341–1377, 2018.

[35] J. J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: Past, present and future", *Concurrency and Computation: Practice and Experience*, vol. 15, 2003.

[36] J. J. Dongarra and P. Luszczek, "Linpack benchmark", in *Encyclopedia of Parallel Computing*, 2011.

[37] A. Community, "Optimization for the high performance linpack benchmark", in *The Student Supercomputer Challenge Guide: From Supercomputing Competition to the Next HPC Generation*. Singapore: Springer Singapore, 2018, pp. 181–191, ISBN: 978-981-10-3731-3. DOI: doi:10.1007/978-981-10-3731-3_11. [Online]. Available: https://doi.org/10.1007/978-981-10-3731-3%5C_11.

[38] J. D. McCalpin, "Hpl and dgemm performance variability on the xeon platinum 8160 processor", *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 225–237, 2018.

[39] K. Goto and R. A. van de Geijn, "Anatomy of high-performance matrix multiplication", *ACM Trans. Math. Softw.*, vol. 34, 12:1–12:25, 2008.

[40] J. McCalpin, *Stream: Sustainable memory bandwidth in high performance computers*, Sep. 1991.

[41] J. J. Dongarra, M. Heroux, and P. Luszczek, "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems", *The International Journal of High Performance Computing Applications*, vol. 30, pp. 10–3, 2016.

[42] S. Browne, J. J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors", *The International Journal of High Performance Computing Applications*, vol. 14, pp. 189–204, 2000.

[43] J. Dongarra, K. London, S. Moore, P. Mucci, and D. Terpstra, "Using papi for hardware performance monitoring on linux systems", in *Conference on Linux Clusters: The HPC Revolution*. Urbana, Illinois: Linux Clusters Institute, Jun. 2001.

[44] A. C. De Melo, "The new linux'perf'tools", in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.

[45] J. R. Goodman, "Using cache memory to reduce processor-memory traffic", in *International Symposium on Computer Architecture*, 1983.

[46] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware roofline model: Upgrading the loft", *IEEE Computer Architecture Letters*, vol. 13, pp. 21–24, 2014.

[47] *Microbenchmark to achieve peak performance on x86_64 cpus and nvidia gpus*, `https://github.com/Dr-Noob/peakperf`, 2021.

[48] *Libpfm-4.x: A helper library to program the performance monitoring events*, `https://github.com/wcohen/libpfm4`, 2008.

[49] P. Guide, "Intel® 64 and ia-32 architectures software developer's manual", *Volume 3B: System programming Guide, Part*, vol. 2, no. 11, 2011.

[50] J. M. Bull and J. C. Maxwell, "Measuring synchronisation and scheduling overheads in openmp", *Proceedings of First European Workshop on OpenMP*, Feb. 2002.

[51] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[52] R. Nelson, D. Towsley, and A. Tantawi, "Performance analysis of parallel processing systems", *IEEE Transactions on Software Engineering*, vol. 14, no. 4, pp. 532–540, 1988. DOI: 10.1109/32.4676.

[53] J. Protic, M. Tomasevic, and V. M. Milutinovic, "A survey of distributed shared memory systems", *Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences*, vol. 1, 74–84 vol.1, 1995.

[54] P. Woodbury, A. W. Wilson, B. Shein, *et al.*, "Shared memory multiprocessors: The right approach to parallel processing", *Digest of Papers. COMPCON Spring 89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage*, pp. 72–80, 1989.

[55] K. Li and P. Hudak, "Memory coherence in shared virtual memory systems", *ACM Trans. Comput. Syst.*, vol. 7, pp. 321–359, 1989.

[56] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.