Ατομική Διπλωματική Εργασία

ΧΑΡΑΚΤΗΡΙΣΜΟΣ ΕΠΙΔΟΣΗΣ ΣΥΣΤΗΜΑΤΟΣ ΟΚΤΩ ΠΥΡΗΝΩΝ ΜΕ ΤΗΝ ΧΡΗΣΗ ΤΗΣ ΟΜΑΔΑΣ ΠΡΟΓΡΑΜΜΑΤΩΝ ΣΥΓΚΡΙΤΙΚΗΣ ΕΠΙΔΟΣΕΩΣ PARSEC

Ορέστης Αγαθοκλέους

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2009

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Χαρακτηρισμός επίδοσης συστήματος οκτώ πυρήνων με την χρήση της ομάδας προγραμμάτων συγκριτικής αποδόσεως PARSEC

Ορέστης Αγαθοκλέους

Επιβλέπων Καθηγητής

Γιαννάκης Σαζεϊδης

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2009

Περίληψη

Σκοπός της παρούσας εργασίας είναι η μελέτη της αρχιτεκτονικής ενός σύγχρονου, παράλληλου υπολογιστικού συστήματος. Παρουσιάζεται η οργάνωση του επεξεργαστή και οι διάφορες τεχνολογίες που χρησιμοποιούνται σε αυτόν. Γίνεται ανάλυση του πρωτοκόλλου συνοχής μνήμης που χρησιμοποιείται μέσα στο σύστημα, και πώς αυτό επηρεάζει την απόδοση του συστήματος.

Η αρχιτεκτονική επηρεάζει σημαντικά την απόδοση των εφαρμογών, γιαυτό και υπάρχει μια συνεχή ανάπτυξη στα συστήματα με σκοπό να αυξήσουν την επίδοση των παράλληλων εφαρμογών. Επομένως βλέπουμε τα συστήματα να απομακρύνονται από τους “κοινούς πόρους”, και η νέες τάσεις της τεχνολογίας να μεταφέρονται σε αρχιτεκτονικές όπου οι πόροι του συστήματος είναι “κατανεμημένοι”

Μέσα από πείραμα που διεξάγεται βλέπουμε ότι το πρωτόκολλο συνοχής μνήμης επηρεάζει σημαντικά τον χρόνο εκτέλεσης των προγραμμάτων, ενώ η απουσία ανάγκης για συνοχής μνήμης μειώνει τον χρόνο εκτέλεσης της εφαρμογής στο επίπεδο του 7.

Μέσα από την χρήση προγραμμάτων της ομάδας PARSEC, βλέπουμε την συμπεριφορά τους όταν αξιολογούνται στην εν λόγω μηχανή. Προγράμματα που δεν υλοποιούν επικοινωνία και συγχρονισμό μεταξύ των νημάτων εκτέλεσης, μπορούν να φτάσουν σε speedup το οποίο είναι στο επίπεδο του θεωρητικού μεγίστου. Αντίθετα, εφαρμογές όπου υλοποιείται επικοινωνία των νημάτων εκτέλεσης με κοινούς πόρους μνήμης, μειώνει σημαντικά το speedup που μπορούν να επιτύχουν οι εφαρμογές.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Παραλληλισμός

Ο παραλληλισμός είναι σημαντικό χαρακτηριστικό των συστημάτων στις μέρες μας. Η τάση της τεχνολογίας των υπολογιστικών συστημάτων δείχνει ότι με την πάροδο του χρόνου, παρουσιάζεται αύξηση στις υπολογιστικές μονάδες που αποτελούν τα συστήματα από γενιά σε γενιά. Το γεγονός αυτό, παρουσιάζεται σε όλες τις μορφές των υπολογιστικών συστημάτων, και δεν περιορίζεται μόνο στα συστήματα γενικής χρήσης. Ακόμη και τα πιο μικρά συστήματα, όπως κινητά τηλέφωνα, παιχνιδομηχανές, κάρτες γραφικών κτλ αποτελούνται από πληθώρα υπολογιστικών μονάδων, όπου σε συνεργασία εκτελούν τους υπολογισμούς παράλληλα.

Ο παραλληλισμός όμως δεν είναι χαρακτηριστικό που παρουσιάζεται μόνο στα σύγχρονα συστήματα. Η έννοια αυτή υπάρχει στα συστήματα από πολύ παλιά, όπου με την πάροδο του χρόνου η νέες ανακαλύψεις στον κλάδο της πληροφορικής, και συγκεκριμένα στην αρχιτεκτονική συστημάτων, βρίσκουν νέους τρόπους έτσι ώστε να αξιοποιήσουν την παράλληλη επεξεργασία. Βλέπουμε αρχικά τα συστήματα να χρησιμοποιούν pipeline μοντέλο υπολογισμού, όπου η εκτέλεση μιας εντολής διαχωρίζεται σε στάδια. Με τη χρήση του μοντέλου αυτού, σε κάθε κύκλο μηχανής εκτελούνται παράλληλα διαφορετικά κομμάτια εντολών, με αποτέλεσμα να αυξάνεται

η ρυθμοαπόδοση της εκτέλεσης των εντολών των προγραμμάτων. Με την πάροδο του χρόνου τα συστήματα προσπαθούν να αξιοποιήσουν τις δυνατότητες παραλληλισμού μεταξύ των εντολών (Instruction Level Parallelism).

Το κλασσικό pipeline επεκτίνεται σε διακλαδώσεις, όπου διαφορετικές εντολές μπορούν να εκτελεστούν παράλληλα. Η παράλληλη εκτέλεση εντολών στο μοντέλο αυτό όμως περιορίζεται από τις εξαρτήσεις που έχουν μεταξύ τους οι εφαρμογές, με αποτέλεσμα να περιορίζει τον βαθμό παραλληλισμού που μπορεί να επιτευχθεί. Το γεγονός αυτό καλείται να επιλύσει το speculative execution, έτσι ώστε να αξιοποιούνται καλύτερα οι δομές μέσα στους επεξεργαστές, αυξάνοντας έτσι τον παραλληλισμό που μπορεί να επιτευχθεί. Ο παραλληλισμός που μπορεί να επιτευχθεί από το ILP έχει φτάσει στον μέγιστο βαθμό, αφού για να γίνει σωστή χρονοδρομολόγηση των εντολών ο επεξεργαστής πρέπει να έχει την ικανότητα να ψάχνει πολλές εντολές εκ των προτέρων έτσι ώστε να βρει εντολές που μπορούν να εκτελεστούν [2]. Ακόμη και αν έχει την δυνατότητα να πράξει το γεγονός αυτό, η πολυπλοκότητα και το overhead της λειτουργίας αυτής είναι πολύ μεγάλη, και επομένως όχι ρεαλιστική. Επίσης, οι διάφορες μορφές prediction που χρησιμοποιούνται στο speculative execution δεν είναι αλάνθαστες, με αποτέλεσμα να μειώνεται σημαντικά ο χρόνος εκτέλεσης των προγραμμάτων στην περίπτωση όπου γίνουν λάθη.

Για τον λόγω αυτό, οι επεξεργαστές να αξιοποιούν τον παραλληλισμό στο επίπεδο των νημάτων εκτέλεσης. Κάποιες από τις δομές του επεξεργαστή πολλαπλασιάζονται, ανάλογα με το επίπεδο παραλληλισμού που χρειάζεται να επιτευχθεί. Δομές όπως το αρχείο καταχωρητών, μετρητής εντολών προγράμματος και πίνακας σελίδων υπάρχουν για κάθε ένα από τα νήματα, ενώ δομές όπως TLB και L1 κρυφή μνήμη είναι κοινές.

Όλες οι ποιο πάνω επιλογές έχουν γίνει με τον στόχο να αυξηθεί η απόδοση των προγραμμάτων στους επεξεργαστές, μέσω της παράλληλης επεξεργασίας. Κάθε μια από αυτές, αυξάνει την πολυπλοκότητα των επεξεργαστών και επομένως την ανάγκη για περισσότερους transistiors μέσα στα συστήματα. Το γεγονός αυτό έχει γίνει δυνατό με την σταθερή μείωση του μεγέθους των transistors. Σύμφωνα με τον νόμο

του Moore, κάθε 18 μήνες διπλασιάζεται η χωρητικότητα των transistors πάνω στα chips, αφού οι εξελίξεις τις τεχνολογίας κάνουν δυνατή την μείωση του μεγέθους τους. Το γεγονός αυτό, επιτρέπει και την μείωση του κύκλου ρολογιού, που σε συνδυασμό με τους μηχανισμούς αυτούς έχουν σαν αποτέλεσμα την αύξηση της επίδοσης των συστημάτων κάθε γενιάς.

Παρόλα αυτά, η αύξηση της πολυπλοκότητας των επεξεργαστών, η μείωση του μεγέθους των transistors, καθώς επίσης και η συνεχής μείωση του κύκλου ρολογιού, έχουν παρουσιάζει ένα νέο πρόβλημα στην αρχιτεκτονική υπολογιστών, όπου η κατανάλωση ενέργειας που παρουσιάζεται στα συστήματα είναι πολύ μεγάλη. Η ενέργεια αυτή μετατρέπεται σε θερμότητα. Ψηλά επίπεδα θερμότητας επηρεάζουν την αξιοπιστία των υπολογισμών, καθώς επίσης δημιουργεί φθορές στις μονάδες του συστήματος. Οι δύο μορφές ισχύος που επηρεάζουν την κατανάλωση ενέργειας στα συστήματα είναι η δυναμική και η στατική, η κάθε μια από τις οποίες υπολογίζεται με τους τύπους που ακολουθούν.

$$\text{Power dynamic} = 1/2 * \text{Capacitive Load} * \text{Voltage}^2 * \text{Frequency}$$

$$\text{Power static} = \text{Current}_{\text{static}} * \text{Voltage}$$

Με την αύξηση της συχνότητας ρολογιού, παρουσιάζεται γραμμική αύξηση της δυναμικής ισχύος. Επίσης, η αύξηση του αριθμού των transistors μέσα στους επεξεργαστές, έχει αυξήσει και την στατική ισχύ που καταναλώνεται.

Για τον λόγω αυτό, οι σύγχρονες τάσεις στην αρχιτεκτονική υπολογιστών κάνουν χρήση πιο απλών επεξεργαστών, όπου η συχνότητα ρολογιού τους είναι χαμηλότερη από αυτούς της προηγούμενης γενιάς. Η μείωση του μεγέθους των transistors επιτρέπει την ύπαρξη πολλών τέτοιων υπολογιστικών μονάδων μέσα στα συστήματα, με αποτέλεσμα να διατηρηθεί ο παράλληλος υπολογισμός. Επομένως, οι νέοι επεξεργαστές επιτυγχάνουν την επίδοση των προγραμμάτων μέσα από τον παράλληλο υπολογισμό, και όχι με την αυξημένη συχνότητα ρολογιού.

1.2 Στόχος διπλωματικής εργασίας

Στόχος της παρούσας εργασίας είναι η μελέτη ενός σύγχρονου παράλληλου συστήματος, το οποίο αποτελείται από δύο τετραπύρηνους επεξεργαστές. Σκοπός μας είναι η μελέτη της αρχιτεκτονικής οργάνωσης του, και πώς αυτή επηρεάζει τον παραλληλισμό των προγραμμάτων.

Η παρούσα εργασία οργανώνεται με τον ακόλουθο τρόπο:
Στο κεφάλαιο 2 παρουσιάζεται η περιγραφή του συστήματος που εξετάζεται και οι διάφορες τεχνολογίες που χρησιμοποιούνται. Το κεφάλαιο 3 παρουσιάζεται ο μηχανισμός που υλοποιείται για την συνοχή της μνήμης στο σύστημα. Το κεφάλαιο 4 παρουσιάζει την οργάνωση των μελλοντικών συστημάτων της εταιρίας Intel με την τεχνολογία Quickpath. Στο κεφάλαιο 5 παρουσιάζεται πείραμα που έχει σαν στόχο την εξεύρεση του χρόνου επικοινωνίας μεταξύ των πυρήνων του συστήματος, και πώς το πρωτόκολλο συνοχής μνήμης επηρεάζει τον χρόνο αυτό. Τέλος, στο κεφάλαιο 6 παρουσιάζονται τα πειράματα που έχουν γίνει για τρεις από τις εφαρμογές που αποτελούνται στην ομάδα προγραμμάτων Parsec.

Κεφάλαιο 2

Περιγραφή Συστήματος

Το σύστημα που αξιολογούμε αποτελείται από επεξεργαστές Xeon E5405 της εταιρίας Intel και έχει την ικανότητα να υποστηρίζει δύο από τους εν λόγω επεξεργαστές. Διαθέτει δώδεκα εισδοχές μνήμης τύπου PC2-5300 DDR2 SD-RAM Fully Buffered DIMMS δίνοντας έτσι την δυνατότητα υποστήριξης 48GB κύριας μνήμης για μέγιστη χωρητικότητα.

2.1 Οργάνωση Επεξεργαστών

Όπως έχει αναφερθεί, το σύστημα χρησιμοποιεί επεξεργαστές Xeon E5405. Κάθε επεξεργαστής αποτελείται από δύο διπύρηνους επεξεργαστές, οι οποίοι λειτουργούν σε μέγιστη συχνότητα των 2GHz. Υποστηρίζεται αρχιτεκτονική x86-64 η οποία είναι

υπερσύνολο της αρχιτεκτονικής συνόλου εντολών x86, παρέχοντας την δυνατότητα εκτέλεσης κώδικα και των δύο αρχιτεκτονικών.

Η επικοινωνία του επεξεργαστή με το chipset επιτυγχάνεται με την χρήση ενός 64bit Front Side Bus (FSB) το οποία εξυπηρετεί και τα δύο ζευγάρια πυρήνων στον επεξεργαστή. Το FSB υποστηρίζει τέσσερις παράλληλες συναλλαγές ανά κύκλο οι οποίες εκτελούνται στη συχνότητα των 333MHz, παρέχοντας έτσι 1333MT το δευτερόλεπτο. Συνολικά από κάθε δίαυλο μπορούν να μεταφερθούν 10.4GB δεδομένα το δευτερόλεπτο. Η δυνατότητα του συστήματος να παρέχει υποστήριξη για δύο τέτοιους επεξεργαστές το καθιστά δυνατό να παρέχει συνολικά οκτώ λογικούς επεξεργαστές.

2.2 Δίαυλος επικοινωνίας - Front Side Bus

Ο δίαυλος επικοινωνίας (FSB) είναι ένα αξιοσημείωτο χαρακτηριστικό ενός επεξεργαστή το οποίο μπορεί να επηρεάσει δραματικά την απόδοση του συστήματος. Στόχος του διαύλου αυτού είναι η επικοινωνία του επεξεργαστή με τα επιμέρους υποσυστήματα του υπολογιστή καθώς επίσης και την ενδοεπικοινωνία μεταξύ των επεξεργαστών που υπάρχουν σε αυτό. Επιπλέον λειτουργία που πραγματοποιείται μέσω του διαύλου είναι το coherency protocol, μηχανισμός ο οποίος εξασφαλίζει την συνοχή των δεδομένων που είναι κοινά μεταξύ διαφορετικών πυρήνων και κατ΄ επέκταση επεξεργαστών του συστήματος έτσι ώστε να διατηρείται η ορθότητα της εκτέλεσης των προγραμμάτων.

Παρατηρώντας την πληθώρα των δεδομένων που διακινούνται μέσα από το δίαυλο αυτό, είναι φανερό ότι το FSB μπορεί να αποτελέσει παράγοντα συμφόρισης μέσα στο σύστημα επιφέροντας έτσι αρνητικές επιπτώσεις στην απόδοση των προγραμμάτων που εκτελούνται από το σύστημα.

Οι τελευταίες τάσης της τεχνολογίας σχεδιασμού επεξεργαστών είναι η αντικατάσταση του κοινού FSB με point-to-point συνδέσεις για την επικοινωνία μεταξύ των επεξεργαστών. Η τάση αυτή έχει αποδειχτεί ιδανικότερη και αποτελεσματικότερη, αφού με την αύξηση των πυρήνων πάνω στους επεξεργαστές καθώς επίσης και την

παρουσία πολλαπλών επεξεργαστών πάνω σε ένα σύστημα, οι ανάγκες του εύρους του καναλιού επικοινωνίας αυξάνονται με μεγάλους ρυθμούς. Το φαινόμενο αυτό καθιστά την υλοποίηση της ενδοεπικοινωνίας των επεξεργαστών με FSB μη αποτελεσματική αφού ο δίαυλος αυτός δεν μπορεί να αντεπεξέλθει στις ανάγκες του συστήματος.

Η ύπαρξη δύο FSB, όπου το κάθε ένα είναι υπεύθυνο να εξυπηρετεί αποκλειστικά διαφορετικό επεξεργαστή έρχεται να αντιμετωπίσει το πρόβλημα αυτό, αφού με την διαρρύθμιση αυτή διαμοιράζεται η μεταφορά των δεδομένων μεταξύ των δύο διαύλων αυξάνοντας έτσι και το συνολικό εύρος της επικοινωνίας που μπορεί να υποστηριχτεί από το σύστημα.

2.3 Κρυφή Μνήμη

Η κρυφή μνήμη αποτελεί σημαντικό παράγοντα για την επίδοση ενός επεξεργαστή, η οποία αναμένεται να επηρεάσει θετικά ή αρνητικά τον χρόνο εκτέλεσης του benchmark που χρησιμοποιείται για την αξιολόγηση της μηχανής. Μέσα σε κάθε πυρήνα παρουσιάζεται η ύπαρξη κρυφής μνήμης 32KB L1 για εντολές καθώς επίσης και 32KB L1 write back για δεδομένα.

Σημαντικό χαρακτηριστικό είναι η ύπαρξη μεγάλης L2 κρυφής μνήμης, το συνολικό μέγεθος της οποίας μέσα σε ένα επεξεργαστή φτάνει στα 12MB και διαμοιράζεται σε κάθε ζευγάρι πυρήνων. Η διαρρύθμιση αυτή παρέχει σε κάθε ζευγάρι πυρήνων 6MB μέγεθος κρυφής μνήμης το οποίο μοιράζονται. Το γεγονός αυτό οδηγεί σε καλύτερη χρησιμοποίηση της κρυφής μνήμης, αφού επιτρέπεται η ανισοζήγιστη κατανομή της στους εν λόγω πυρήνες όταν οι ανάγκες τους για μνήμη δεν είναι οι ίδιες.

2.4 Τεχνολογία Κύριας Μνήμης – Fully Buffered DIMMS

Η χρήση της τεχνολογίας των Fully Buffered DIMMS (FB-DIMMS) είναι μια επιλογή που έχει κάνει η εταιρία Intel για τους επεξεργαστές των σειρών Xeon 5000/5100. Η απόδοση της συγκεκριμένης τεχνολογίας είναι αμφιλεγόμενη, αφού προσπαθώντας να επιλύσει τα προβλήματα που προκύπτουν από την κλασσική μνήμη DDR,

παρουσιάζει αύξηση του χρόνου που χρειάζεται ο Memory Controller για την πρόσβαση του στην μνήμη. Αυτό αναμένεται να επηρεάζει αρνητικά τα προγράμματα τα οποία έχουν μεγάλες απαιτήσεις σε επικοινωνία με την μνήμη.

2.4.1 Προβλήματα Κλασσικής Μνήμης DDR

Σημαντικό πρόβλημα που παρουσιάζεται με την κλασσική μνήμη DDR είναι η γραμμική σχέση του χρόνου πρόσβασης της με την αύξηση των μονάδων μνήμης που προσθέτονται πάνω στο σύστημα. Το πρόβλημα αυτό προκύπτει από την χρήση ενός κοινού διαύλου επικοινωνίας μεταξύ του Memory Controller και τις διάφορες μονάδες μνήμης DDR. Με την αύξηση των μονάδων πάνω στον δίαυλο παρουσιάζεται μείωση της ποιότητας και της ταχύτητας του σήματος επικοινωνίας. Το γεγονός αυτό αποτελεί σημαντικό περιορισμό στο συνολικό μέγεθος μνήμης που μπορεί να υποστηρίζει το σύστημα.

2.4.2 Αρχιτεκτονική της Fully Buffered τεχνολογίας μνήμης

Στόχος της Fully Buffered τεχνολογίας μνήμης είναι η αντιμετώπιση των προβλημάτων που αναφέρονται πιο πάνω με την χρήση μιας νέας αρχιτεκτονικής, η οποία επιτρέπει μεγαλύτερη συνολική χωρητικότητα και μεγαλύτερη απόδοση σε σχέση με την κλασσική μνήμη DDR.

Σε αντίθεση με ενός κοινού διαύλου επικοινωνίας, η καινούρια αρχιτεκτονική κάνει χρήση ενός buffer (Advanced Memory Buffer) στο κέντρο κάθε μονάδα μνήμης. Η διασύνδεση με τον Memory Controller επιτυγχάνεται με την ύπαρξη τεσσάρων καναλιών όπου πραγματοποιείται point-to-point σύνδεση μεταξύ του Memory Controller και τα AMB τεσσάρων από τις μονάδες μνήμης. Επιπλέον μονάδες μπορούν να τοποθετηθούν σειριακά μεταξύ των τεσσάρων μονάδων αυτών όπου πραγματοποιείται σύνδεση μεταξύ των AMB.

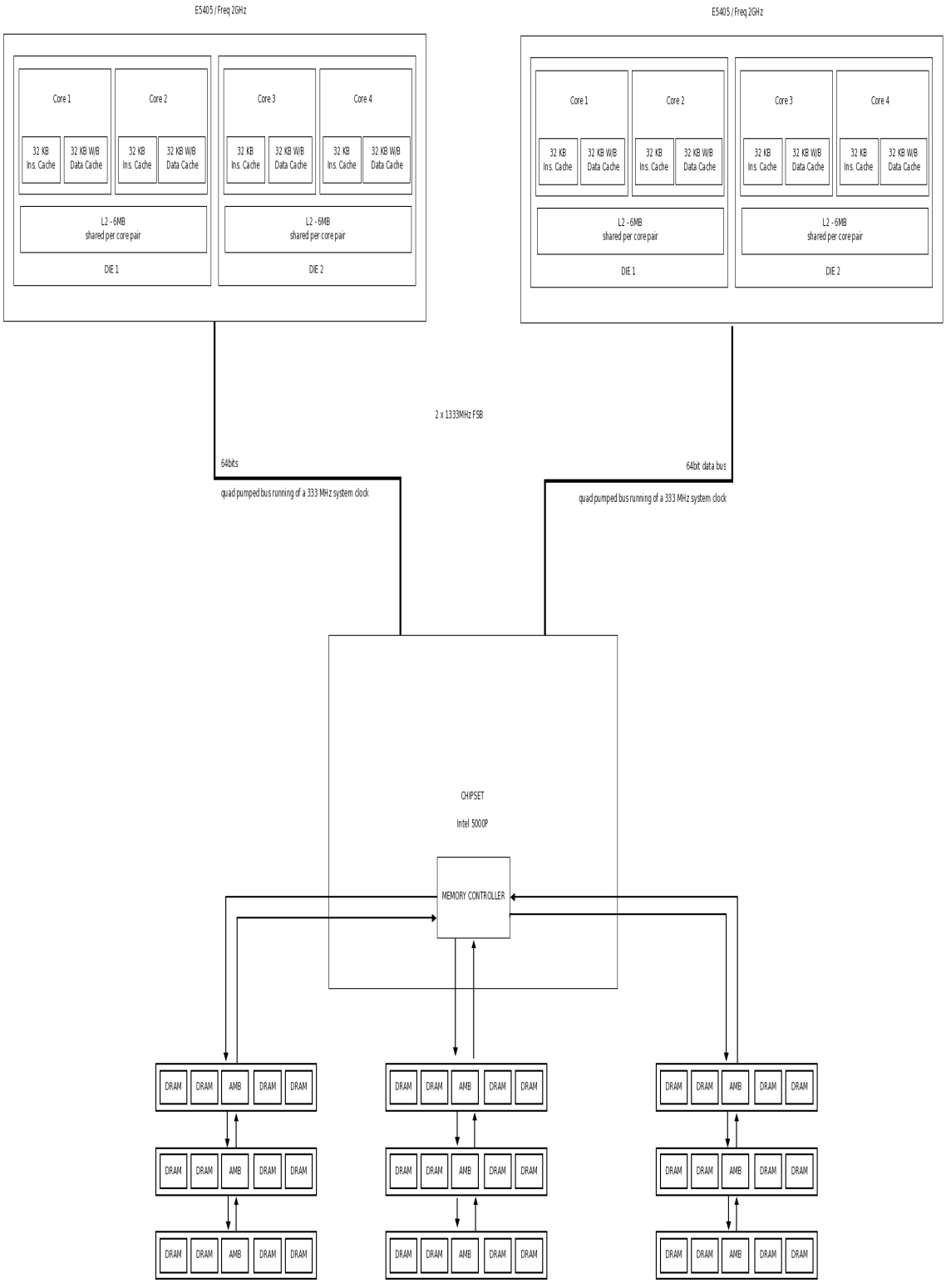
Με αυτή την αρχιτεκτονική επιτυγχάνεται μείωση του αριθμού των pins που χρειάζεται κάθε μονάδα μνήμης, αφού από 240 pins που χρειάζεται μια κλασσική μονάδα DDR μειώνονται στα 69. Ο αριθμός αυτός αποτελείται από 20 pins για

μεταφορά δεδομένων από τον Memory Controller στην μονάδα, 28 pins για δεδομένα από την μονάδα στον Memory Controller, 6 pins για ισχύ και 12 για γείωση. Κάθε κανάλι έχει την ικανότητα να υποστηρίξει μέχρι και οκτώ μονάδες μνήμης. Αξιοσημείωτο χαρακτηριστικό της αρχιτεκτονικής είναι ο διαχωρισμός των καναλιών που απευθύνονται στο διάβασμα και στην εγγραφή δεδομένων από και προς την μνήμη. Το γεγονός αυτό δίνει την δυνατότητα παράλληλης εκτέλεσης των δύο λειτουργιών, όταν αυτές εξυπηρετούν διαφορετικά DIMMS, μειώνοντας έτσι τον συνολικό χρόνο που απαιτείται για των ολοκλήρωση των λειτουργιών αυτών.

2.4.3 Προβλήματα της νέας αρχιτεκτονικής

Όπως αναμένεται, η σειριακή διαρρύθμιση των μονάδων σε κάθε κανάλι δημιουργεί πρόβλημα αύξησης χρόνου για τις μονάδες οι οποίες βρίσκονται στο τέλος του καναλιού. Κατ' επέκταση του προβλήματος αυτού, παρουσιάζεται και αύξηση χρόνου στα σήματα μετάδοσης με την πρόσθεση νέα μονάδας μνήμης πάνω σε κάποιο από τα κανάλια.

Για το πρώτο πρόβλημα, η εταιρία Intel υποστηρίζει ότι με την αύξηση της συχνότητας λειτουργίας των μονάδων μνήμης, η αύξηση χρόνου θα μειωθεί επομένως η σειριακή διαρρύθμιση δεν αποτελεί μεγάλο πρόβλημα στην τεχνολογία. Όσον αφορά την αύξηση του χρόνου μετάδοσης με την πρόσθεση επιπλέον μανάδων μνήμης στα κανάλια, τα buffers που χρησιμοποιούνται σε κάθε μονάδα χρησιμοποιούν ιδικό pass through κανάλι, σε αντίθεση με μια store and forward αρχιτεκτονική μειώνοντας έτσι τον χρόνο μετάδοσης που χρειάζονται τα σήματα για να καταφθάσουν στη σωστή μονάδα μνήμης. Επιπλέον, η παράλληλη εκτέλεση της αποστολή και παραλαβή μηνυμάτων με τον διαχωρισμό των καναλιών που είναι υπεύθυνα για κάθε λειτουργία, αναμένεται να μειώσει τον χρόνο πρόσβασης στην μνήμη.



Σχημα 2.1 Σχεδιάγραμμα οργάνωσης συστήματος

Κεφάλαιο 3

Πρωτόκολλο Συνοχής Μνήμης

Ένα κοινό πρόβλημα που παρουσιάζεται με την χρήση κρυφών μνήμων σε συστήματα πολυεπεξεργαστών και πολυπηρύνων είναι το πρόβλημα της συνοχής της μνήμης. Ενώ η χρήση της κρυφής μνήμης, και κατ' επέκταση της ιεραρχικής διαρρύθμισης της μνήμης στο σύστημα καλείται να μειώσει το χάσμα μεταξύ της απόδοσης του επεξεργαστή με αυτή της κύριας μνήμης εντούτοις στα συστήματα με πολυπεξεργαστές, όπου ο κάθε ένας έχει αποκλειστικά την δική του κρυφή μνήμη, δημιουργούνται προβλήματα στα δεδομένα που είναι κοινά μεταξύ των επεξεργαστών. Εξ' ορισμού, η παρουσία της κρυφής μνήμης περιορίζει την όψη της μνήμης από τους επεξεργαστές έτσι ώστε να βλέπουν την κατάσταση της μνήμης μέσα από την αποκλειστική κρυφή μνήμη του κάθε ενός, γεγονός το οποίο μπορεί να προκαλέσει λανθασμένη εκτέλεση όταν τα δεδομένα μιας κοινής διεύθυνσης της κύριας μνήμης βρίσκονται σε πολλές κρυφές μνήμες. Ο Πίνακας 3.1 που ακολουθεί παρουσιάζει το πρόβλημα, όπου δύο επεξεργαστές μπορεί να βλέπουν διαφορετική τιμή για μια συγκεκριμένη διεύθυνση.

Χρόνος	Γεγονός	Περιεχόμενα κρυφής μνήμης επεξεργαστή Α	Περιεχόμενα κρυφής μνήμης επεξεργαστή Β	Περιεχόμενα κύριας μνήμης για την μεταβλητή Ι
0				1
1	Ο Α διαβάζει την μεταβλητή Ι	1		1
2	Ο Β διαβάζει την μεταβλητή Ι	1	1	1
3	Ο Α τροποποιεί το περιεχόμενα της Ι στην τιμή 2	2	1	2

Πίνακας 3.1 Πρόβλημα συνοχής μνήμης για μια διεύθυνση της κύριας μνήμης (Ι) όταν αυτή είναι κοινή μεταξύ των επεξεργαστών Α και Β.

Το ποιο πάνω σενάριο παρουσιάζει το πρόβλημα της συνοχής για κοινά δεδομένα της κύριας μνήμης. Υποθέτουμε ότι η αρχική τιμή της μεταβλητής Ι στην κύρια μνήμη είναι 1, και αρχικά δεν βρίσκεται σε καμία από τις κρυφές μνήμες των δύο επεξεργαστών. Διαβάζοντας την μεταβλητή ο επεξεργαστής Α, την αντιγράφει από την κύρια μνήμη και τοποθετεί την τιμή αυτή στην κρυφή του μνήμη. Η ίδια πράξη γίνεται και από τον επεξεργαστή Β όταν διαβάζει για πρώτη φορά την τιμή από την κύρια μνήμη. Ακολούθως ο επεξεργαστής Α τροποποιεί την τιμή της μεταβλητής Ι από 1 σε 3. Η τιμή επομένως ανανεώνεται στην κρυφή του μνήμη, και υποθέτοντας Write Through κρυφή μνήμη τότε ενημερώνεται και η κύρια μνήμη με την αλλαγή. Το πρόβλημα παρουσιάζεται μεταξύ των δύο επεξεργαστών, αφού για την ίδια μεταβλητή βλέπουν διαφορετική τιμή.

3.1 Πρωτόκολλα Συνοχής Μνήμης

Για την επίλυση του πιο πάνω προβλήματος, έχουν δημιουργηθεί πρωτόκολλα τα οποία καθορίζουν κανόνες οι οποίοι εγγυούνται την συνοχή της μνήμης μεταξύ των επεξεργαστών έτσι ώστε να επιλύονται αμφιβολίες σχετικά με την τιμή κοινών μεταβλητών. Υπάρχει πληθώρα πρωτοκόλλων που έχουν υλοποιηθεί και καλούνται να επιλύσουν το πρόβλημα, όμως με βάση την λειτουργία τους μπορούν να διαχωριστούν σε δύο κατηγορίες. Έτσι υπάρχουν τα Snooping Based πρωτόκολλα και τα Directory Based.

Τα Snooping πρωτόκολλα βασίζονται στην ύπαρξη ενός κοινού διαύλου επικοινωνίας μεταξύ των επεξεργαστών (συνήθως γίνεται χρήση του FSB) στον οποίο εκδίδονται τα διάφορα μηνύματα του πρωτοκόλλου. Κάθε ένας από τους επεξεργαστές, είναι υπεύθυνος να γνωρίζει την κατάσταση των περιεχομένων της κρυφής μνήμης του, να εκδίδει μηνύματα πάνω στον δίαυλο για τις αλλαγές που κάνει πάνω στα δεδομένα της κρυφής του μνήμης καθώς επίσης και να εξετάζει τα μηνύματα που εκδίδονται πάνω στον δίαυλο από τους άλλους επεξεργαστές έτσι ώστε να πάρει τις ανάλογες αποφάσεις για τυχόν αλλαγές που πρέπει να γίνουν στα τοπικά του δεδομένα.

Αντίθετα, στα Directory Based πρωτόκολλα, γίνεται η χρήση του Directory, ένας κοινός χώρος ο οποίος είναι υπεύθυνος για την κατάσταση των δεδομένων της μνήμης. Στον χώρο αυτό βρίσκονται πληροφορίες κατά πόσο ένα block της μνήμης βρίσκεται σε κάποια από τις κρυφές μνήμες των επεξεργαστών ή όχι. Στην περίπτωση όπου το block αυτό χρησιμοποιείται από τους επεξεργαστές, επιπλέον πληροφορίες μέσα στο Directory καθορίζουν τους επεξεργαστές αυτούς. Με βάση αυτή την διαρρύθμιση, οι επεξεργαστές επικοινωνούν μόνο με το Directory, όπου στέλλουν σε αυτό μηνύματα για τις αλλαγές που κάνουν πάνω στα δεδομένα της κρυφής τους μνήμης. Ακολούθως, το Directory είναι υπεύθυνο, σε περίπτωση που οι αλλαγές έχουν γίνει σε κοινή διεύθυνση της μνήμης να μεταβιβάσει τα ανάλογα μηνύματα στους επηρεασθείς επεξεργαστές έτσι ώστε να ικανοποιηθεί η συνοχή των δεδομένων.

3.2 Πρωτόκολλο Συνοχής MESI

Στο σύστημα που εξετάζεται χρησιμοποιείται το πρωτόκολλο MESI για την διασφάλιση της συνοχής των κοινών δεδομένων της μνήμης από τους διάφορους επεξεργαστές. Το πρωτόκολλο βασίζεται στην κατηγορία των Snooping Based πρωτοκόλλων και επομένως οι πληροφορίες για τα κοινά blocks της μνήμης βρίσκονται στις κρυφές μνήμες των επεξεργαστών που τα διαχειρίζονται. Κάθε block της μνήμης που παρουσιάζεται σε κάποια από τις L2 κρυφές μνήμες μπορεί να χαρακτηρίζεται από μία από τις πιο κάτω καταστάσεις:

- Modified : Το block βρίσκεται μόνο μέσα στην κρυφή μνήμη του παρόντος επεξεργαστή και έχει δεχτεί τροποποιήσεις στα δεδομένα του. Χρειάζεται να

ενημερωθεί η κύρια μνήμη (Write Back) σε περίπτωση που το ζητήσει κάποιος άλλος επεξεργαστής.

- Exclusive : Το block βρίσκεται μόνο μέσα στην κρυφή μνήμη του παρόντος επεξεργαστή, αλλά δεν έχει δεχτεί οποιαδήποτε τροποποίηση. Τα δεδομένα που περιέχονται σε αυτό είναι τα ίδια με αυτά της κύριας μνήμης.
- Shared : Το block είναι κοινό μεταξύ επεξεργαστών και επομένως βρίσκεται σε διάφορες κρυφές μνήμες. Τα δεδομένα του όμως δεν έχουν δεχτεί τροποποίηση και είναι τα ίδια μεταξύ των κρυφών μνήμων και της κύριας μνήμης.
- Invalid : Τα δεδομένα του block δεν είναι τα πιο πρόσφατα. Μεταγενέστερη πρόσβαση στο block θα προκαλέσει ενημέρωση του block από τη μνήμη.

3.2.1 Λειτουργία Πρωτοκόλλου

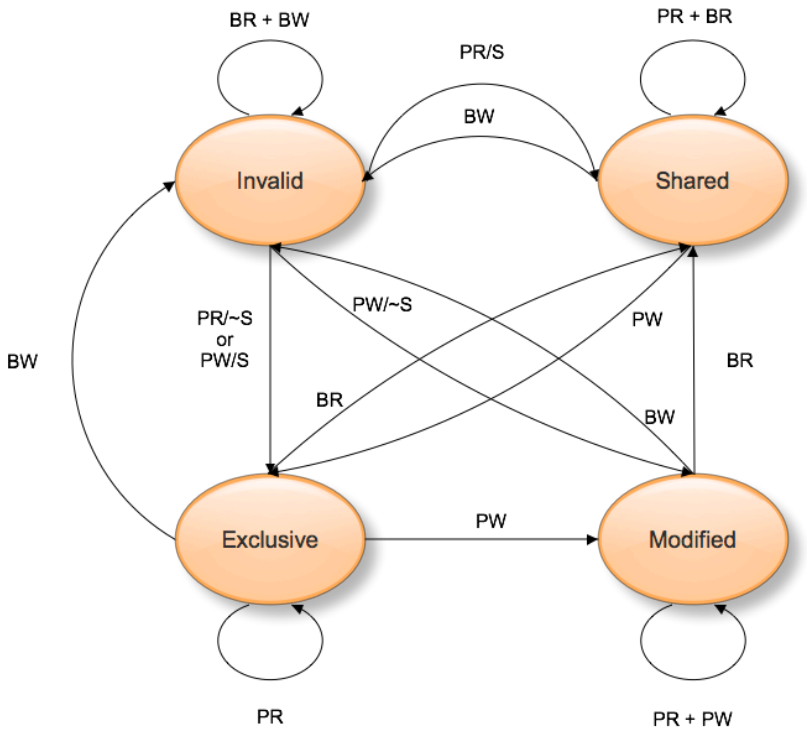
Αρχικά κανένα block της μνήμης δεν βρίσκεται στις κρυφές μνήμες των επεξεργαστών. Όταν κάποιος επεξεργαστής ζητήσει κάποιο block τότε εκδίδει μήνυμα πάνω στον δίαυλο και ο memory controller του στέλλει τα δεδομένα από την κύρια μνήμη. Στην συνέχεια, αποθηκεύεται το block στην κρυφή μνήμη του επεξεργαστή και χαρακτηρίζεται από την Exclusive κατάσταση. Σε αυτή την κατάσταση, επιπλέον διαβάσματα δεδομένων του block από τον επεξεργαστή δεν θα αλλάξουν την κατάσταση του. Ζητώντας όμως κάποιος άλλος επεξεργαστής το block, τότε μεταφέρεται από την κρυφή μνήμη του επεξεργαστή που έχει τα δεδομένα στην κρυφή μνήμη του επεξεργαστή που τα έχει ζητήσει και το block μέσα στις δύο κρυφές μνήμες χαρακτηρίζεται από την κατάσταση Shared. Επιπλέον διαβάσματα των blocks από τους επεξεργαστές δεν θα αλλάξουν την κατάσταση του.

Στην περίπτωση όπου κάποιος επεξεργαστής είναι Exclusive κάποιου block και τροποποιήσει τα δεδομένα του, τότε η κατάσταση του block μετατρέπεται σε Modified. Επιπλέον τροποποιήσεις ή διαβάσματα από τον ίδιο επεξεργαστή διατηρούν την Modified κατάσταση. Όταν όμως το block βρίσκεται σε Shared κατάσταση, και δεχτεί τροποποίηση από κάποιο από τους επεξεργαστές που το έχει στην κρυφή του μνήμη τότε ο εν λόγω επεξεργαστής εκδίδει μήνυμα ακύρωσης πάνω στον δίαυλο και ενημερώνοντας τα δεδομένα του block στην κρυφή του μνήμη τροποποιεί την κατάσταση του block σε Modified. Οι υπόλοιποι επεξεργαστές που

έχουν το block στην κρυφή τους μνήμη, λαμβάνοντας το μήνυμα από τον δίαυλο ακυρώνουν το block στην κρυφή τους μνήμη μεταφέροντας το στην Invalid κατάσταση.

Στην περίπτωση όπου το block βρίσκεται μόνο μέσα σε μια κρυφή μνήμη αλλά έχει δεχτεί τροποποίηση, και επομένως χαρακτηρίζεται από την Modified κατάσταση, και το ζητήσει κάποιος άλλος επεξεργαστής, τότε ο κατέχων επεξεργαστής το μεταβιβάζει στον επεξεργαστή που το έχει ζητήσει, και η κατάσταση του block και στις δύο κρυφές μνήμες αλλάζει σε Shared. Ο Memory Controller έχει την ευθύνη να ανιχνεύσει την αλλαγή αυτή, και ενημερώνει την μνήμη με τα ποιο πρόσφατα δεδομένα. [3]

Η πιο πάνω διαδικασία μπορεί να περιγραφεί μέσα από ένα διάγραμμα καταστάσεων (Σχήμα 3.2) το οποίο παρουσιάζει τις αλλαγές του block για κάθε γεγονός που παρουσιάζεται. Χαρακτηριστικά της κάθε κατάστασης παρουσιάζονται από τον Πίνακα 3.2.



PR	Διάβασμα δεδομένων από επεξεργαστή (Processor Read)
PW	Τροποποίηση δεδομένων από επεξεργαστή (Processor Write)
S/~S	Κοινό/Μή κοινό block (Shared/~Shared)
BR	Έκδοση μηνύματος για διάβασμα πάνω στον δίαυλο από άλλο επεξεργαστή (Bus Read)
BW	Έκδοση μηνύματος τροποποίησης πάνω στον δίαυλο από άλλο επεξεργαστή (Bus Write)

Σχήμα 3.1 Διάγραμμα καταστάσεων πρωτοκόλλου MESI για κάποιο block στην κρυφή μνήμη

Κατάσταση Block	Modified	Exclusive	Shared	Invalid
Είναι έγκυρο;	Ναι	Ναι	Ναι	Όχι
Τα δεδομένα της μνήμης..	Δεν είναι ενημερωμένα	Έγκυρα	Έγκυρα	-
Υπάρχουν αντίγραφα σε άλλες κρυφές μνήμες;	Όχι	Όχι	Μπορεί	Μπορεί
Τροποποίηση του block	Δεν εκδίδεται στον δίαυλο επικοινωνίας	Δεν εκδίδεται στον δίαυλο επικοινωνίας	Ο επεξεργαστής έχει αποκλειστική ιδιοκτησία του block	Εκδίδεται στον δίαυλο επικοινωνίας

Πίνακας 3.2 Χαρακτηριστικά των καταστάσεων του πρωτοκόλλου MESI [3]

3.3 Inclusive και Exclusive κρυφές μνήμες

Τα συστήματα με πολλαπλά επίπεδα κρυφών μνήμων παρουσιάζουν κάποιες αποφάσεις σχεδιασμού όσον αφορά την σχέση των δεδομένων που βρίσκονται ανάμεσα σε δύο επίπεδα των μνήμων αυτών. Υπάρχουν συστήματα τα οποία εγγυώνται ότι η παρουσία δεδομένων σε κάποιο επίπεδο κρυφής μνήμης πρέπει να υπάρχει και στο αμέσως επόμενο επίπεδο κρυφής μνήμης, γεγονός το οποίο χαρακτηρίζεται από τον όρο strictly inclusive. Από την άλλη, παρουσιάζονται συστήματα στα οποία η ύπαρξη δεδομένων μεταξύ δύο διαδοχικών επιπέδων κρυφής μνήμης βρίσκεται μόνο σε ένα από τα επίπεδα αυτά, συμπεριφορά η οποία χαρακτηρίζει τις μνήμες αυτές σαν exclusive. Παραλλαγές των χαρακτηριστικών αυτών αποτελούν τα συστήματα της εταιρίας Intel, στα οποία υλοποιείται ένα πιο χαλαρό μοντέλο στην σχέση των κρυφών μνήμων. Σε τέτοια συστήματα η ύπαρξη δεδομένων σε κάποιο επίπεδο κρυφής μνήμης δεν εγγυάται, ούτε αποκλείει την ύπαρξη τους στο αμέσως χαμηλότερο επίπεδο της κρυφής μνήμης, στα οποία χρησιμοποιείται ο όρος mainly inclusive για να χαρακτηρίσει την συμπεριφορά αυτή.

Πλεονέκτημα της χρήσης exclusive cache, είναι η μείωση της καθυστέρησης των δεδομένων από την μνήμη σε κάποιον επεξεργαστή, αφού σε αντίθεση με τις inclusive κρυφές μνήμες αποθηκεύονται περισσότερα δεδομένα στις κρυφές μνήμες. Παρόλα αυτά, η διαδικασία ανταλλαγής κάποιου block από την L1 στην L2 είναι πιο πολύπλοκη

από αυτή των inclusive κρυφών μνήμων, με αποτέλεσμα να υπάρχει σημαντική καθυστέρηση.

Πλεονεκτήματα χρήσης inclusive κρυφών μνήμων είναι η απλοποίηση του μηχανισμού συνοχής των δεδομένων. Στην περίπτωση όπου παρουσιάζονται μηνύματα του πρωτοκόλλου συνοχής πάνω στον δίαυλο επικοινωνίας, υλοποιείται tag match μόνο στην L2 κρυφή μνήμη για να εξακριβωθεί κατά πόσο ο επεξεργαστής παραλήπτης χρειάζεται να ενεργήσει στο μήνυμα. Σαν αποτέλεσμα, η L2 αποκρύβει την L1 από τα διάφορα μηνύματα, μη παρ ενοχλώντας έτσι την λειτουργία του επεξεργαστή.

Επιπλέον βελτιστοποιήσεις που γίνονται στις κρυφές μνήμες, ανεξαρτήτως του inclusiveness τους είναι η παρουσία των duplicated tags, όπου τα tag arrays των κρυφών μνήμων έρχονται εις διπλού. Με αυτό τον μηχανισμό, τα tag matches γίνονται πάνω στα αντιγραμμένα tag arrays, χωρίς να επηρεάζεται η απόδοση των κρυφών μνήμων.

Κεφάλαιο 4

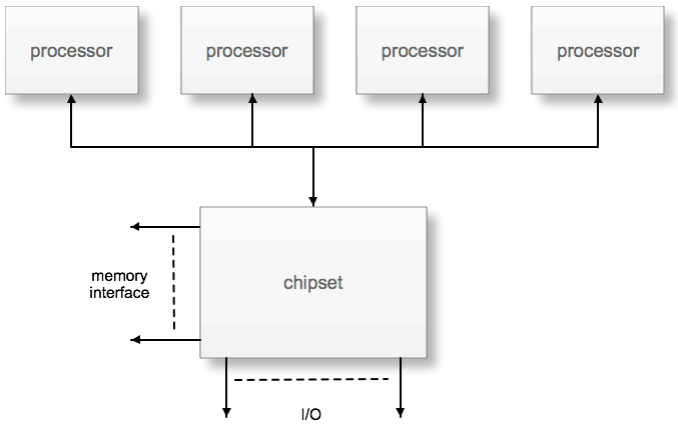
Αναδρομή και Μελλοντικές Υλοποιήσεις

4.1 Εξέλιξη του μέσου επικοινωνίας των επεξεργαστών

Μέσα στην πάροδο του χρόνου, παρουσιάστηκαν πολλές εξελίξεις στην αρχιτεκτονική των συστημάτων με πολυεπεξεργαστές. Για την επίτευξη της επικοινωνίας μεταξύ των επεξεργαστών, καθώς επίσης και με τις επιμέρους μονάδες του συστήματος (chipset) γινόταν μέχρι τώρα η χρήση διαύλου επικοινωνίας.

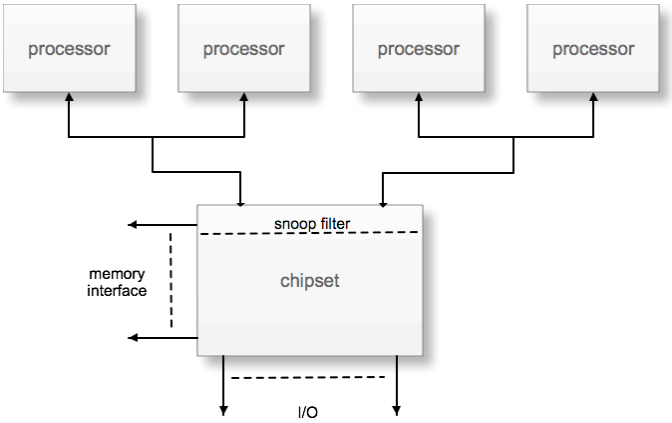
Σε παλαιότερα συστήματα με κοινό δίαυλο επικοινωνίας (Σχήμα 4.1) η αποστολή των δεδομένων μεταξύ των επεξεργαστών και του chipset γινόταν εξολοκλήρου ενός κοινού, αμφίδρομου διαύλου κοινός γνωστού με το όνομα Front Side Bus. Λόγω της φύσης του διαύλου να επιτρέπει μόνο σύγχρονη αποστολή δεδομένων, η αύξηση του αριθμού των επεξεργαστών και των πυρήνων στο σύστημα προκαλούσε δυσχέρεια στην κίνηση των δεδομένων μέσα σε αυτό. Επιπλέον πρόβλημα που παρουσιάστηκε

ήταν οι ηλεκτρονικοί περιορισμοί που έπρεπε να αντιμετωπιστούν με την αύξηση της συχνότητας λειτουργίας των διαύλων.



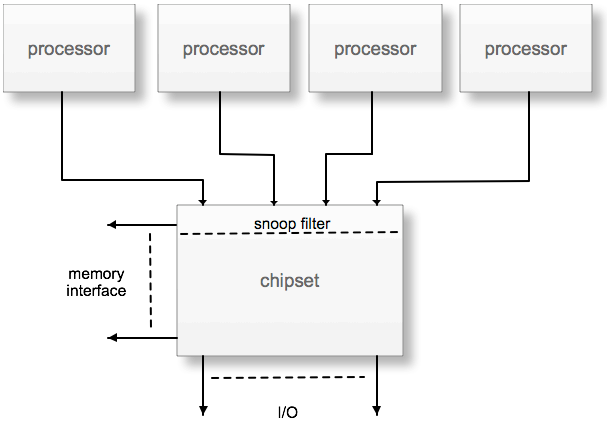
Σχήμα 4.1 Κοινός δίαυλος δεδομένων σε συστήματα μέχρι το 2004

Η επιπλέον αύξηση του εύρους ζώνης του διαύλου έχει επιτευχθεί με την χρήση δύο ανεξάρτητων διαύλων (Σχήμα 4.2). Με την διαρρύθμιση αυτή, διπλασιάζεται και το διαθέσιμο εύρος ζώνης στο σύστημα. Παρόλα αυτά, τα μνήματα του πρωτοκόλλου συνοχής μνήμης που χρησιμοποιούνται στα συστήματα (MESI) χρειάζεται να μεταδίδονται σε όλους τους επεξεργαστές ανεξάρτητα αν είναι αναγκαία ή όχι, με αποτέλεσμα την πιθανή μείωση του εύρους ζώνης. Για τον λόγω αυτό, εισάγεται μια νέα αρχιτεκτονική δομή στο σύστημα (Snoop Filter), με σκοπό να γνωρίζει τους επεξεργαστές που διαμοιράζονται δεδομένα από την μνήμη. Με αυτό τον τρόπο, η δομή αυτή είναι σε θέση να φιλτράρει μηνύματα του πρωτοκόλλου που αποστέλλονται μεταξύ των δύο διαύλων επικοινωνίας όταν αυτά είναι αχρείαστα, μειώνοντας έτσι τον φόρτο δεδομένων που μεταφέρονται σε αυτούς.



Σχήμα 4.2 Δύο ανεξάρτητοι δίαυλοι δεδομένων σε συστήματα από το 2005

Η πιο πάνω προσέγγιση έχει εξελιχθεί στην λογική επέκταση της αρχιτεκτονικής με τη χρήσης αποκλειστικών διαύλων επικοινωνίας, όπου κάθε επεξεργαστής επικοινωνεί με το chipset μέσα από τον δικό του δίαυλο (Σχήμα 4.3). Για τους ίδιους λόγους, η χρήση του snoop filter είναι απαραίτητη έτσι ώστε να μειωθεί ανεπιθύμητη επικοινωνία των μηνυμάτων του πρωτοκόλλου συνοχής μνήμης μεταξύ των επεξεργαστών.



Σχήμα 4.3 Αποκλειστικοί δίαυλοι δεδομένων σε συστήματα του 2007

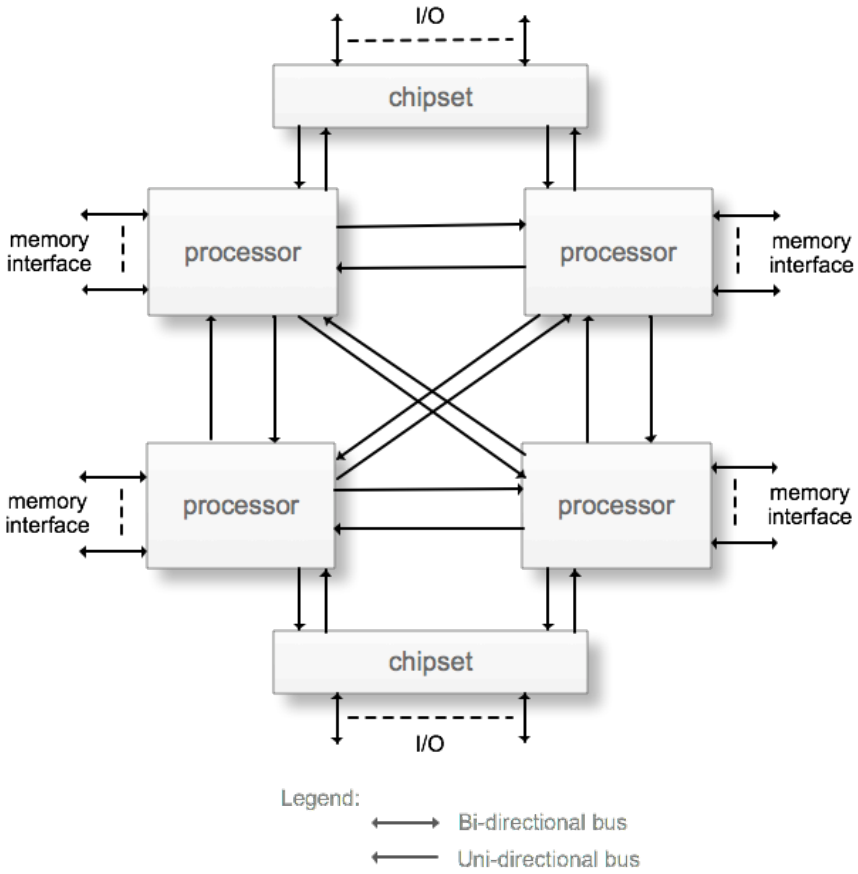
4.2 Τεχνολογία Intel Quickpath

Με την νέα τεχνολογία επεξεργαστών, παρουσιάζεται καινούρια διαρρύθμιση στην διασύνδεση των επεξεργαστών με την χρήση της τεχνολογίας Intel Quickpath (Σχήμα 4.4). Η τεχνολογία αυτή υλοποιεί συνδέσεις point-to-point, όπου κάθε επεξεργαστής

του συστήματος είναι απευθείας συνδεδεμένος με τους υπόλοιπους επεξεργαστές όπως επίσης και με το chipset. Κάθε σύνδεση μεταξύ δύο επεξεργαστών χρησιμοποιεί δύο διαύλους επικοινωνίας μονής κατεύθυνσης, επιτυγχάνοντας παράλληλη αποστολή και παραλαβή δεδομένων. Παρόμοια, η διασύνδεση κάθε επεξεργαστή με το chipset υλοποιείται επίσης από αποκλειστικό δίαυλο δεδομένων, με την διαφορά ότι γίνεται χρήση ενός διαύλου διπλής κατεύθυνσης. Με αυτό τον τρόπο ο ένας χρησιμοποιείται για την αποστολή δεδομένων από τον επεξεργαστή στο chipset, ενώ ο άλλος δίαυλος χρησιμοποιείται για την αποστολή δεδομένων από την αντίθετη κατεύθυνση, επιτυγχάνοντας έτσι παράλληλη αποστολή δεδομένων από και προς τις μονάδες εισόδου/εξόδου.

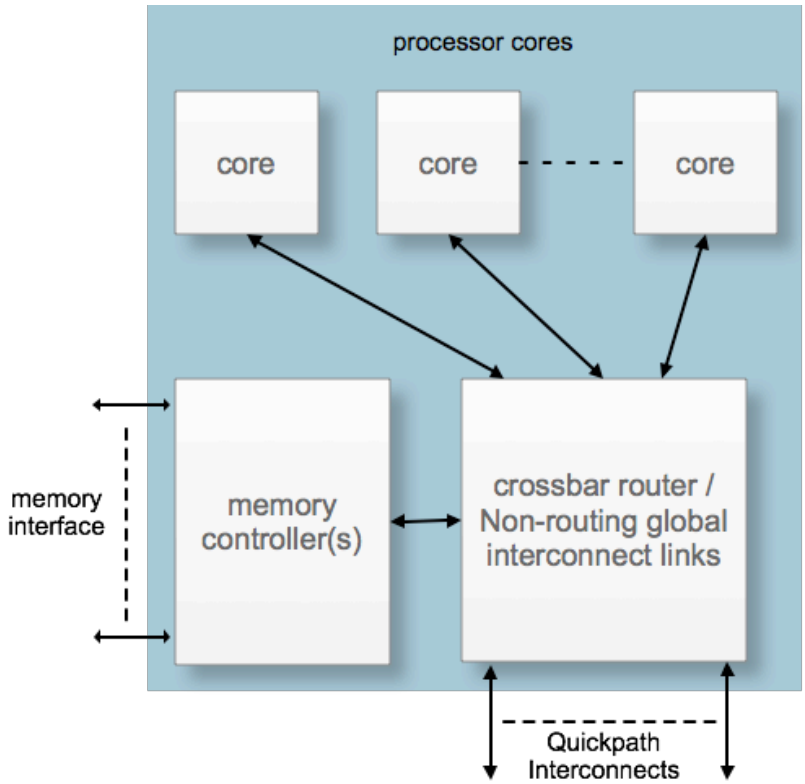
Σημαντική διαφορά παρουσιάζεται στην διαρρύθμιση της μνήμης. Η μέχρι τώρα παρουσία ενός κοινού memory controller μέσα στο chipset καταργείται, δίνοντας έμφαση σε μικροαρχιτεκτονική κατανεμημένης μνήμης. Ο memory controller μεταφέρεται από το chipset μέσα σε κάθε ένα από τους επεξεργαστές. Ανάλογα με της ανάγκες του συστήματος, κάθε επεξεργαστής μπορεί να έχει πάνω από ένα memory controller αυξάνοντας έτσι το εύρος ζώνης μεταφοράς δεδομένων από και προς την κύρια μνήμη.

Η διαρρύθμιση αυτή καθιστά δυνατή και την κατάργηση του κοινού chipset που χρησιμοποιείται στα παρόντα συστήματα. Με την νέα αρχιτεκτονική, παρουσιάζονται δύο chipsets μέσα στο σύστημα, ένα για κάθε ζευγάρι επεξεργαστών. Με αυτό τον τρόπο μειώνεται η συμφόρηση με την επικοινωνία των μονάδων εισόδου/εξόδου.



Σχήμα 4.4 Αρχιτεκτονική διασύνδεσης Intel Quickpath

Στο σχήμα 4.5 φαίνεται η σημασιολογία ενός επεξεργαστή μέσα σε συστήματα που χρησιμοποιούν την τεχνολογία αυτή. Κάθε επεξεργαστής μπορεί να αποτελείται από πολλαπλούς πυρήνες. Στην περίπτωση όπου υπάρχει παρουσία πολλών πυρήνων τότε ανάλογα με το σύστημα μπορεί αυτοί να έχουν κοινές ή ανεξάρτητες κρυφές μνήμες. Όπως έχει αναφερθεί, ο επεξεργαστής αποτελείται από ένα ή περισσότερους memory controllers. Ανάλογα με το επίπεδο επεκτασιμότητας του συστήματος, μέσα στον επεξεργαστή μπορεί να υπάρχει crossbar δρομολογητής και μία ή περισσότερες πόρτες τεχνολογίας Quickpath. Κάθε πόρτα αποτελείται από δύο διαύλους μονής κατεύθυνσης που χρησιμοποιούνται για την επικοινωνία με ένα από τους επεξεργαστές του συστήματος.



Σχήμα 4.5 Επεξεργαστής σε συστήματα που κάνουν χρήση την τεχνολογία Quickpath

4.2 Πρωτόκολλο συνοχής μνήμης

Όπως έχει αναφερθεί σε προηγούμενο κεφάλαιο, το πρωτόκολλο συνοχής μνήμης που χρησιμοποιείται στα συστήματα της εταιρίας Intel είναι το MESI. Η επιλογή της εταιρίας για το συγκεκριμένο πρωτόκολλο έχει παραμείνει σταθερή με την πάροδο του χρόνου για όλα τα συστήματα πολυεπεξεργαστών που παράγει, με μικρές βελτιστοποιήσεις του πρωτοκόλλου έτσι ώστε να εκμεταλλεύεται περιπτώσεις που παρέχουν δυνατότητα βελτιστοποίησης του.

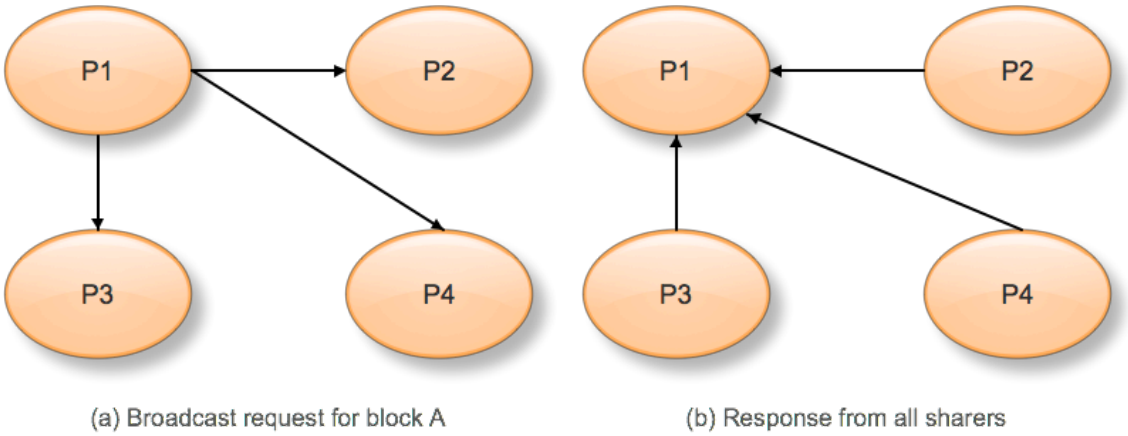
Το πρωτόκολλο αυτό χρησιμοποιείται και για τα συστήματα που χρησιμοποιούν την νέα τεχνολογία διασύνδεσης των επεξεργαστών, όπου προαιρετικά εισάγεται και μια νέα Forward (F) κατάσταση η οποία χρησιμοποιείται για cache-to-cache μεταφορά δεδομένων. Τα χαρακτηριστικά των καταστάσεων του πρωτοκόλλου συνοψίζονται στον Πίνακα 4.1. Όπως φαίνεται από τον πίνακα, με την ένταξη της νέας κατάστασης στο πρωτόκολλο αλλάζει ο ρόλος της κατάστασης Shared.

Κατάσταση	Τροποποιημένο Block;	Μπορεί να τροποποιήσει το block;	Το block μπορεί να μεταφερθεί;	Νέες Καταστάσεις
Modified	Ναι	Ναι	Ναι	-
Exclusive	Οχι	Ναι	Ναι	MSIF
Shared	Οχι	Οχι	Οχι	I
Invalid	-	Οχι	Οχι	-
Forward	Ναι	Οχι	Ναι	SI

Πίνακα 4.1 Χαρακτηριστικά καταστάσεων πρωτοκόλλου MESIF

Η επιλογή της νέας κατάστασης έχει γίνει έτσι ώστε να αποφευχθεί ένα σοβαρό πρόβλημα που υπάρχει με το κλασσικό πρωτόκολλο. Το πρόβλημα αυτό προκύπτει από τη φύση του πρωτοκόλλου, με το οποίο κάθε κόμβος που λαμβάνει μέρος σε αυτό δεν γνωρίζει για τους υπόλοιπους.

Το Σχήμα 4.6 παρουσιάζει σενάριο στο οποίο τίθεται η νέα αυτή κατάσταση να αποφύγει. Θεωρούμε ότι έχουμε σύστημα με τέσσερις επεξεργαστές, όπου οι τρεις από αυτούς διαμοιράζονται ένα κοινό block (A), το οποίο βρίσκεται στην κρυφή μνήμη και των τριών στην Shared κατάσταση. Στην συνέχεια, ο τέταρτος επεξεργαστής, θέλοντας να διαβάσει το block αυτό, εκδίδει μήνυμα πάνω στον δίαυλο επικοινωνίας. Το μήνυμα αυτό αποστέλλεται στους υπόλοιπους επεξεργαστές. Ο κάθε ένας από αυτούς, ελέγχοντας την κρυφή του μνήμη βρίσκει το block με επιτυχία και επομένως αποστέλλει το block στον επεξεργαστή που το έχει ζητήσει. Το πρόβλημα που παρουσιάζεται από αυτή την συμπεριφορά, είναι η πολλαπλή αποστολή του ίδιου block στον επεξεργαστή που το αναζητά, γεγονός που είναι αχρείαστο και ανεπιθύμητο, αφού αυξάνει την μεταφορά δεδομένων πάνω στον δίαυλο επικοινωνίας.



Σχήμα 4.6 Σενάριο προβλήματος πρωτοκόλλου MESI

Με το νέο πρωτόκολλο, μόνο η κατάσταση F χρησιμοποιείται για την αποστολή κάποιου block δεδομένων από την κρυφή μνήμη του επεξεργαστή που έχει το block στην κατάσταση αυτή, σε κάποιον άλλο επεξεργαστή που ζητά να διαβάσει το block αυτό [5]. Μόνο ένας από τους επεξεργαστές που διαμοιράζονται το block μπορεί να βρίσκεται στην F κατάσταση, ενώ οι υπόλοιποι έχουν το block στην κατάσταση Shared. Η απαγόρευση των επεξεργαστών που έχουν το block στην Shared κατάσταση να αποστείλουν δεδομένα σε επεξεργαστή που ζητά το συγκεκριμένο block, επιτυγχάνει την μη ύπαρξη πολλαπλής αποστολής των ιδίων δεδομένων. Αφού γίνει αποστολή του block στον νέο επεξεργαστή, τότε ο προηγούμενος μεταβιβάζεται στην Shared κατάσταση και ο νέος επεξεργαστής αποθηκεύει το block στην κρυφή του μνήμη σε κατάσταση F.

Για την ορθή λειτουργία του πρωτοκόλλου, η τεχνολογία Quickpath καθορίζει συγκεκριμένους ρόλους στης οντότητες που λαμβάνουν μέρος στο πρωτόκολλο συνοχής μνήμης. Συγκεκριμένα υπάρχουν δύο διακριτές ομάδες ρόλων οι οποίες καθορίζονται ως caching agents και home agents.

Caching Agent είναι οποιαδήποτε οντότητα η οποία μπορεί να ξεκινήσει αιτήματα του πρωτοκόλλου συνοχής. Επίσης, είναι οποιαδήποτε οντότητα η οποία μπορεί να έχει δεδομένα στην κρυφή της μνήμη σε οποιαδήποτε από τις καταστάσεις του πρωτοκόλλου, τα οποία μπορεί να είναι σε θέση να τα αποστείλουν σε οποιοδήποτε άλλο επεξεργαστή τα ζητήσει.

Home Agent είναι η οντότητα η οποία είναι υπεύθυνη να εξυπηρετήσει μια συναλλαγή του πρωτοκόλλου. Μία οντότητα που υπάγεται σε αυτή την ομάδα είναι υπεύθυνη για ένα κομμάτι της μνήμης στην οποία υλοποιείται το πρωτόκολλο συνοχής.

Επομένως, αιτήματα που εκδίδονται στον δίαυλο από κάποιο cache agent λαμβάνονται από τον home agent[4]. Ο home agent είναι υπεύθυνος να ελέγξει την δομή που περιέχει πληροφορίες για την μνήμη που είναι υπεύθυνος, έτσι ώστε να αποφασίσει τον κατάλληλο cache agent που έχει τα δεδομένα αυτά στην F κατάσταση. Αφού βρεθεί ο agent, αποστέλλεται μήνυμα από τον home agent σε αυτόν. Στην συνέχεια, ο cache agent με την F κατάσταση αποστέλλει τα δεδομένα στον cache agent που έχει κάνει την αίτηση, μεταφέροντας την κατάσταση του block σε αυτήν του Shared. Ο παραλήπτης λαμβάνει τα δεδομένα και τα αποθηκεύει στην F κατάσταση.
Στο τέλος, ο παραλήπτης αποστέλλει δεύτερο μήνυμα στον home agent ενημερώνοντας τον για την επιτυχία της συναλλαγής.

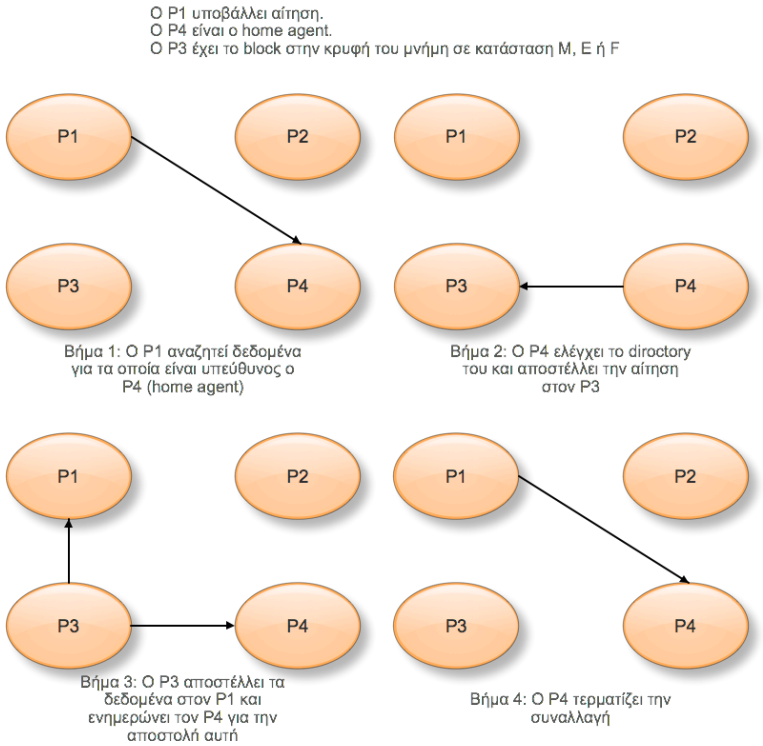
Στην περίπτωση όπου τα δεδομένα που ζητούνται δεν βρίσκονται σε καμία από τις κρυφές μνήμες των άλλων cache agents, ή αυτά παρουσιάζονται μόνο στην Shared κατάσταση των agents αυτών, τότε το δεύτερο μήνυμα που αποστέλλεται από τον cache agent που ξεκινά την αίτηση επιβεβαιώνει το πρώτο. Μέχρι να φτάσει το δεύτερο μήνυμα, ο home agent μπορεί να έχει φέρει τα δεδομένα στην κρυφή του μνήμη επιτρέποντας του να ολοκληρώσει την αίτηση.

4.3 Είδος των Snoops που υποστηρίζονται

Υπάρχουν δύο είδη Snoops που υποστηρίζονται με τις προδιαγραφές της τεχνολογίας Quickpath. Ποιο από τα δύο είδη χρησιμοποιείται μέσα σε ένα σύστημα, καθορίζεται από την αρχιτεκτονική των επεξεργαστών του και τις ανάγκες του τομέα βελτιστοποίησης που είναι επιθυμητό να επιτευχθεί μέσα στο σύστημα. Τα είδη αυτά είναι το Home Snooping και το Source Snooping, τα οποία περιγράφονται στο κείμενο που ακολουθεί.

4.3.1 Home Snooping

Η Home Snooping συμπεριφορά συνοχής μνήμης καθορίζει τον Home Agent υπεύθυνο για τα snoopings που υλοποιούνται από τους Cache Agents του συστήματος. Με αυτή την συμπεριφορά, ο Cache Agent που αναζητεί δεδομένα από κάποιο block της μνήμης, αποστέλει κατευθείαν την αίτηση του στον Home Agent. Ο Home Agent με την σειρά του, κάνοντας χρήση του directory που έχει, γνωρίζει τους Cache Agents που μπορεί να έχουν το block αυτό στην κρυφή τους μνήμη. Αφού βρει τον κατάλληλο Cache Agent, αποστέλλει σε αυτόν την αίτηση. Στην συνέχεια, ο Cache Agent αυτός αποστέλλει τα δεδομένα αυτά σε αυτόν που έχει δημιουργήσει την αίτηση. Επιπλέον, ενημερώνει τον Home Agent ότι έχει αποστείλει τα δεδομένα. Τέλος, ο Home Agent αποστέλλει μήνυμα τερματισμού της συναλλαγής στον αρχικό Cache Agent. Στο Σχήμα 4.7, φαίνεται διαγραμματικά το σενάριο που έχει περιγραφεί.



Σχήμα 4.7: Συμπεριφορά Home Snoop

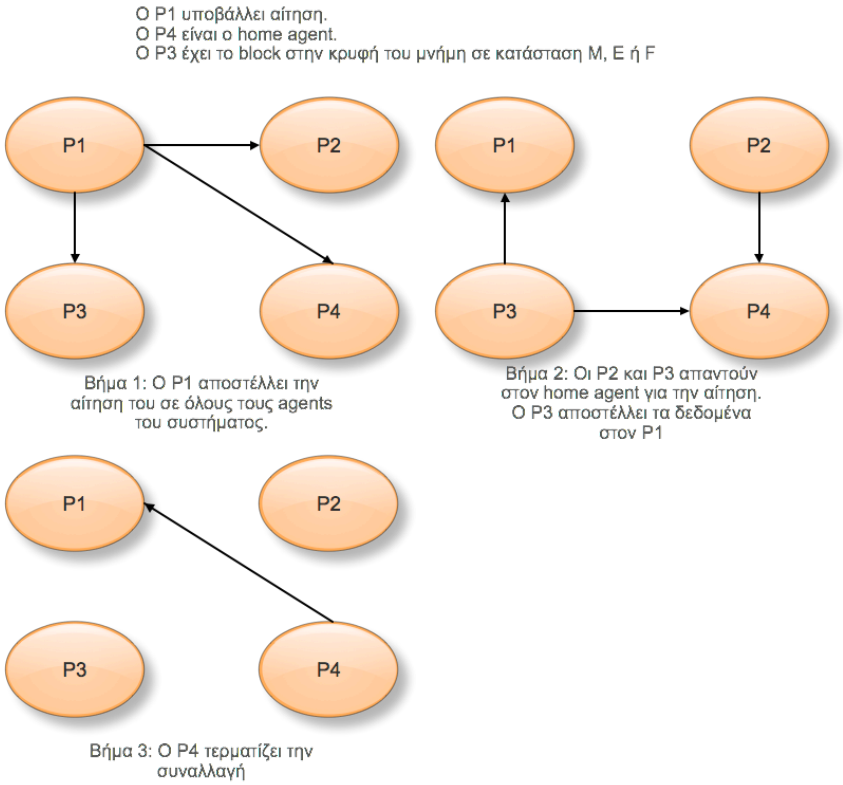
Όπως έχει αναφερθεί, η υλοποίηση αυτή συνδυάζει το snooping πρωτόκολλο MESI(F) με μια προσέγγιση η οποία βασίζεται σε directory, το οποίο βρίσκεται σε κάθε Home Agents έτσι ώστε να γνωρίζει την κατάσταση των blocks που είναι κοινά μεταξύ των

επεξεργαστών του συστήματος. Με αυτό τον τρόπο, μειώνονται τα snoops και οι απαντήσεις τους, με αποτέλεσμα να υπάρχει λιγότερος φόρτος μηνυμάτων που διακινούνται στους διαύλους τους συστήματος. Παρόλα αυτά, η μέθοδος αυτή αυξάνει το κόστος της καθυστέρησης αφού χρειάζονται τέσσερα μηνύματα για να ολοκληρωθεί μια τυπική συναλλαγή. Επίσης με αυτή την μέθοδο αυξάνεται και η πολυπλοκότητα της υλοποίησης της.

Επομένως, το Home Snooping χρησιμοποιείται σε συστήματα που έχουν μεγάλο αριθμό επεξεργαστών (Cache Agents), όπου η πιθανότητα για συμφόρηση στους διαύλους δεδομένων από τα μνήματα του πρωτοκόλλου είναι μεγάλη.

4.3.1 Source Snooping

Η διαδικασία του Source Snooping είναι παρόμοια με αυτή των κλασσικών snooping πρωτοκόλλων που χρησιμοποιούνται στα παρόντα συστήματα. Η αίτηση δεδομένων από κάποιον Cache Agent, αποστέλλεται σε όλους τους Agents του συστήματος. Ακολούθως, οι υπόλοιποι Cache Agents του συστήματος απαντούν στον Home Agent για την κατάσταση του block στην κρυφή τους μνήμη. Αν κάποιος από αυτούς έχει το ζητούμενο block, τότε το αποστέλλει στον Cache Agent που έχει υποβάλει την αίτηση, στέλλοντας επίσης το ανάλογο μήνυμα στον Home Agent. Τέλος, ο Home Agent στέλλει μήνυμα τερματισμού της συναλλαγής στον Cache Agent που έχει λάβει τα δεδομένα. Χρησιμοποιώντας το ίδιο σενάριο με προηγουμένως, η διαδικασία αυτή φαίνεται διαγραμματικά στο Σχήμα 4.8



Σχήμα 4.8: Συμπεριφορά Source Snoop

Σε αντίθεση με την προηγούμενη προσέγγιση, μια τυπική αίτηση της παρούσας προσέγγισης ολοκληρώνεται σε τρία βήματα, γεγονός που την καθιστά γρηγορότερη. Παρόλα αυτά, ο αυξημένος αριθμός μηνυμάτων του πρωτοκόλλου που δημιουργούνται αυξάνει και την πιθανότητα παρουσίας συμφόρησης μέσα στους διαύλους επικοινωνίας. Επομένως, η υλοποίηση αυτή απευθύνεται σε συστήματα όπου ο αριθμός των Cache Agents είναι μικρός.

Κεφάλαιο 5

Χαρακτηρισμός Συστήματος

Ένας από τους παράγοντες που τίθενται να αντιμετωπίσουν οι αρχιτέκτονες συστημάτων με πολυεπεξεργαστές, είναι ο σχεδιασμός συστημάτων τα οποία προσφέρουν το κατάλληλο περιβάλλον σε παράλληλα προγράμματα, υποστηρίζοντας τον μέγιστο δυνατό βαθμό απόδοσης στην εκτέλεση τους. Όπως είναι γνωστό, η αρχιτεκτονική ενός συστήματος επηρεάζει σημαντικά την απόδοση των προγραμμάτων, αφού διάφοροι μηχανισμοί που υλοποιούνται για την εξασφάλιση της ορθότητας των προγραμμάτων αυτών, μπορεί να παρουσιάσουν αύξηση στον χρόνο εκτέλεσης τους. Σημαντικότεροι παράγοντες που προκαλούν αυτού του είδους συμπεριφορά είναι το μέσο επικοινωνίας μεταξύ των επεξεργαστών, και κατ' επέκταση ο χρόνος επικοινωνίας που χρειάζεται μεταξύ των επεξεργαστών, καθώς

επίσης και το πρωτόκολλο που διεξασφαλίζει την συνοχή της κρυφής μνήμης μεταξύ επεξεργαστών που διαχειρίζονται κοινούς χώρους μνήμης.

5.1 Μετρήσεις Χρόνου Επικοινωνίας Επεξεργαστών

5.1.1 Περιγραφή Πειράματος

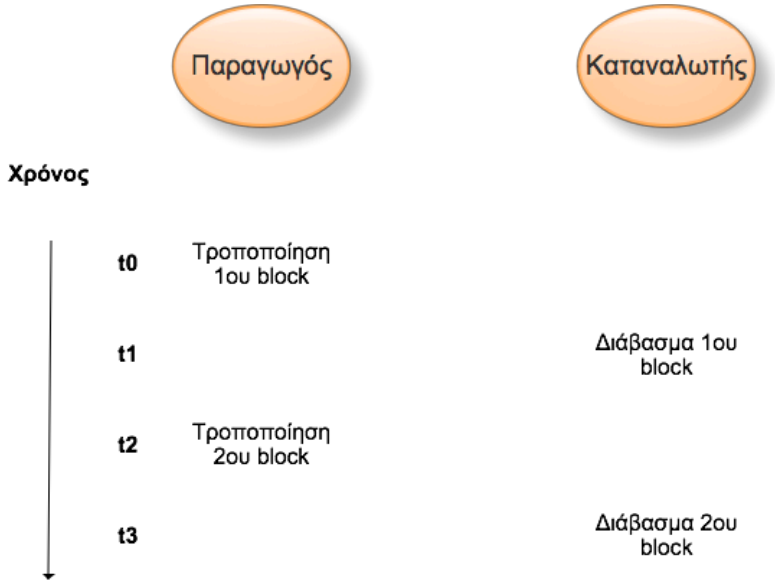
Ο έλεγχος των πιο πάνω παραμέτρων για το σύστημα που αξιολογείται γίνεται με την χρήση του προγράμματος c2cbench [8]. Το εν λόγω πρόγραμμα χρησιμοποιεί μοντέλο παραγωγού – καταναλωτή (producer-consumer), όπου ο κώδικας που απευθύνεται για τον κάθε ένα τρέχει σε διαφορετικό επεξεργαστή / πυρήνα. Υπάρχει η δυνατότητα καθορισμού του μεγέθους των δεδομένων επεξεργασίας, τα οποία είναι κοινά μεταξύ των δύο οντοτήτων, και διαχωρίζονται σε blocks. Η λειτουργία του πειράματος είναι η εξής (Σχήμα 5.1).

Για κάθε block των δεδομένων του πειράματος, ο παραγωγός κάνει τροποποίηση στα δεδομένα του block. Στην συνέχεια, περιμένει τον καταναλωτή να διαβάσει το block, και αφού τελειώσει συνεχίζει το ίδιο μοτίβο για τα υπόλοιπα blocks των δεδομένων. Με την προσπέλαση όλων των blocks, το πείραμα επαναλαμβάνεται από την αρχή. Για κάθε ένα από τα blocks σε κάθε επανάληψη, γίνονται μετρήσεις των χρόνων που χρειάζονται ο παραγωγός και ο καταναλωτής, ο πρώτος για να τροποποιήσει το block και ο δεύτερος για να το διαβάσει. Στο τέλος, παρουσιάζεται ο μέσος χρόνος όλων των επαναλήψεων που χρειάζεται ο παραγωγός για να τροποποιήσει ένα block, καθώς επίσης και ο χρόνος που χρειάζεται ο καταναλωτής για να διαβάσει ένα block.
Με την χρήση ενός τέτοιου μοντέλου, είμαστε σε θέση να μετρήσουμε τον χρόνο επικοινωνίας μεταξύ δύο επεξεργαστών όταν δεν υπάρχει συμφόρηση στον δίαυλο επικοινωνίας του συστήματος. Μέσα στον χρόνο του παραγωγού, συμπεριλαμβάνεται και ο χρόνος που χρειάζεται για να επιτευχθεί συνοχή των δεδομένων. Σύμφωνα με το πρωτόκολλο MESI, δεδομένα που είναι κοινά μεταξύ δύο επεξεργαστών τα οποία είναι είναι έγκυρα στις κρυφές μνήμες τους βρίσκονται στην Shared κατάσταση. Όταν ο παραγωγός τροποποιεί το block, τότε προηγείται το γεγονός Request For Ownership (RFO). Ο επεξεργαστής αποστέλλει μήνυμα Invalidation πάνω στον δίαυλο επικοινωνίας, με το οποίο ακυρώνει τα δεδομένα που βρίσκονται στην κρυφή μνήμη

του επεξεργαστή που βρίσκεται ο καταναλωτής. Αφού γίνει το invalidation, τα δεδομένα τροποποιούνται και βρίσκονται μέσα στην κρυφή μνήμη του επεξεργαστή που βρίσκεται ο παραγωγός, και τα οποία χαρακτηρίζονται από την Modified κατάσταση.

Ο καταναλωτής με την σειρά του εκδίδει αίτημα των δεδομένων του block πάνω στον δίαυλο αφού αυτά είναι άκυρα μέσα στην κρυφή του μνήμη. Το μήνυμα αυτό το βλέπει ο παραγωγός, και στέλλει τα δεδομένα που βρίσκονται στην κρυφή του μνήμη στον καταναλωτή. Μετά το τέλος της λειτουργίας, τα blocks βρίσκονται στην Shared κατάσταση και στις δύο κρυφές μνήμες του κάθε επεξεργαστή.

Το μέγεθος τον δεδομένων, καθώς επίσης και το μέγεθος των blocks καθορίζουν την δομή του συστήματος που θα τεθεί υπό εξέταση. Όταν το μέγεθος των δεδομένων είναι μικρό έτσι ώστε να χωρούν μέσα στην L1 ή L2 κρυφή μνήμη των επεξεργαστών, τότε οι χρόνοι που λαμβάνονται αφορούν την επικοινωνία μεταξύ των δύο επεξεργαστών, καθώς επίσης και τον χρόνο που προστίθεται από το πρωτόκολλο συνοχής της μνήμης. Στην περίπτωση όπου τα δεδομένα είναι μεγαλύτερα από την κρυφή μνήμη που υπάρχει στους επεξεργαστές, τότε μέσα στον χρόνο του παραγωγού και του καταναλωτή συμπεριλαμβάνεται ο χρόνος αποστολής δεδομένων από την κύρια μνήμη στους εν λόγω επεξεργαστές.



Σχήμα 5.1: Μοντέλο εκτέλεσης Παραγωγού - Καταναλωτή

5.1.2 Αποτελέσματα Πειράματος

Όπως έχει προαναφερθεί, ο καθορισμός των αρχιτεκτονικών δομών που τίθενται να εξεταστούν εξαρτάται από το μέγεθος των δεδομένων και του block που χρησιμοποιείται. Στο πείραμα έχουν χρησιμοποιηθεί δεδομένα μεγέθους 16KB, 4096KB και 61440KB με τα οποία ελέγχονται η L1 κρυφή μνήμη, η L2 και η Κύρια Μνήμη αντίστοιχα. Το μέγεθος κάθε block που επεξεργάζεται κάθε φορά έχει οριστεί στα 8KB, ενώ ο διασκελισμός μέσα στο block γίνεται ανά 8 words από τον παραγωγό και τον καταναλωτή αντίστοιχα, διασφαλίζοντας την τροποποίηση και το διάβασμα όλων των cache lines μέσα στο block.

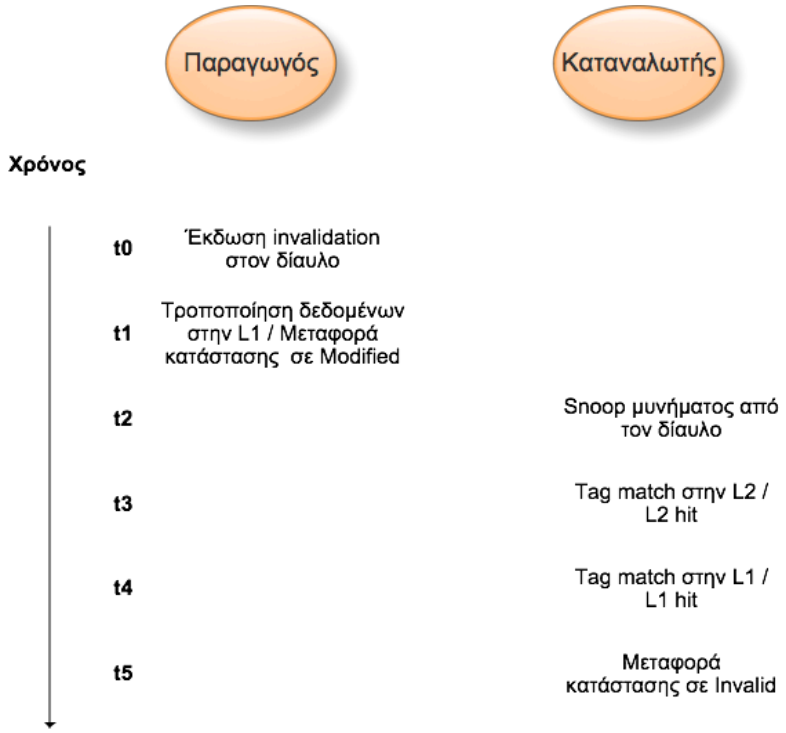
5.1.2.1 Μέγεθος Δεδομένων 16KB

CPU ID Παραγωγού	CPU ID Καταναλωτή	Χρόνος επεξεργασίας block Καταναλωτή (usecs)	Χρόνος επεξεργασίας block Παραγωγού (usecs)
0	1	1.17	2.19
0	2	14.16	2.31
0	3	14.25	2.27
0	4	14.57	2.94
0	5	14.58	2.91
0	6	14.58	2.91
0	7	14.64	2.9

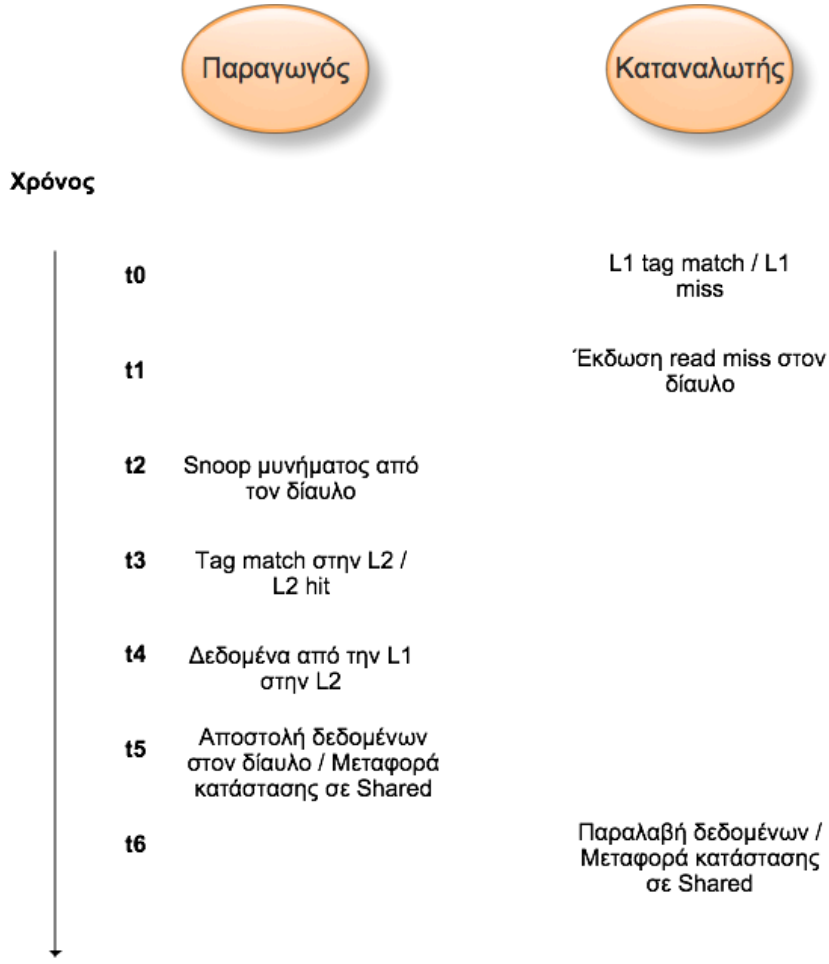
Πίνακας 5.1 Χρόνοι εκτέλεσης πειράματος με δεδομένα μεγέθους 16KB

Ο ποιο πάνω πίνακας παρουσιάζει τον μέσο χρόνο που χρειάζονται ο παραγωγός και ο καταναλωτής για την επεξεργασία ενός block. Ο παραγωγός τρέχει πάντα πάνω στον ίδιο επεξεργαστή ενώ ο καταναλωτής τρέχει σε κάθε περίπτωση σε διαφορετικό επεξεργαστή. Υποθέτοντας ότι η σχέση μεταξύ των κρυφών μνήμων L1 και L2 χαρακτηρίζεται σαν inclusive, οι χρόνοι της κάθε μιας οντότητας απαρτίζονται από τα γεγονότα όπως παρουσιάζεται διαγραμματικά στο Σχήμα 5.2. Λαμβάνοντας υπόψη το μικρό μέγεθος των δεδομένων, με την πρώτη εκτέλεση του πειράματος όλα τα δεδομένα βρίσκονται στις L1 κρυφές μνήμες των δύο επεξεργαστών και όλα τα cache lines βρίσκονται στην κατάσταση Shared. Με την εκτέλεση μεταγενέστερων επαναλήψεων, ο παραγωγός επιθυμώντας να να τροποποιήσει τα δεδομένα κάθε

cache line, εκδίδει μήνυμα invalidation πάνω στον δίαυλο επικοινωνίας και τροποποιεί τα δεδομένα στην τοπική του L1 αλλάζοντας την κατάσταση του σε Modified. Από την πλευρά του ο καταναλωτής, λαμβάνει το μήνυμα από τον δίαυλο, και εφαρμόζει tag match πάνω στην τοπική του L2. Εφόσον τα δεδομένα είναι κοινά, παρουσιάζεται L2 hit, και ακολουθεί tag match πάνω στην L1. Με την ίδια λογική, παρουσιάζεται L1 hit και επομένως το cache line μεταφέρεται στην κατάσταση Invalid. Όταν τελειώσει η διαδικασία, ο καταναλωτής προσπαθεί να διαβάσει τα cache lines που έχουν δεχτεί τροποποίηση. Εφαρμόζοντας tag match πάνω στην L1 κρυφή του μνήμη, παρουσιάζεται L1 miss αφού τα δεδομένα είναι πλέον άκυρα. Λόγω του inclusiveness, τα δεδομένα αυτά δεν υπάρχουν ούτε στην L2, επομένως εκδίδει μήνυμα read miss πάνω στον δίαυλο. Το μήνυμα αυτό λαμβάνει ο επεξεργαστής του παραγωγού, και με tag match πάνω στην δική του L2 βλέπει ότι το πιο πρόσφατο αντίγραφο των δεδομένων βρίσκεται στην τοπική του κρυφή μνήμη. Λόγω της ιδιότητας της κρυφής μνήμης L1 να είναι Write Back, τα δεδομένα στην L2 του παραγωγού δεν είναι τα πιο πρόσφατα, αφού έχουν δεχτεί τροποποίηση, επομένως πρέπει να ενημερωθούν από την L1. Στην συνέχεια, αποστέλλονται μέσω του διαύλου στον καταναλωτή, και μεταφέρονται στην κατάσταση Shared και από τους δύο επεξεργαστές.



Σχήμα 5.2 (α) Τροποποίηση ενός cache line από τον παραγωγό



Σχήμα 5.2 (β) Διάβασμα ενός cache line από τον καταναλωτή (16KB μέγεθος δεδομένων)

Παρατηρώντας τους χρόνους του πειράματος, διακρίνουμε πολύ μεγάλους χρόνους μεταξύ του παραγωγού και του καταναλωτή. Το γεγονός αυτό οφείλεται στο αντίκτυπο που έχει το πρωτόκολλο συνοχής μνήμης πάνω στον παραγωγό. Το μικρό μέγεθος των δεδομένων που χρησιμοποιείται βρίσκεται πάντα μέσα στις L1 κρυφές μνήμες των δύο επεξεργαστών, με αποτέλεσμα να προσθέτει σημαντικό χρόνο πάνω στην εκτέλεση του παραγωγού. Για κάθε τροποποίηση που γίνεται μέσα σε ένα block, το πρωτόκολλο συνοχής προκαλεί tag match πάνω στα δύο επίπεδα μνήμης του καταναλωτή. Όπως έχει περιγραφεί, η διαδικασία αυτή εκτελείται σειριακά πρώτα στο δεύτερο επίπεδο κρυφής μνήμης και μετά στο πρώτο, όπως απαιτείται από την inclusive σχέση μεταξύ της L1 και L2 μνήμης κάποιου επεξεργαστή.

Σημαντική μείωση του χρόνου εκτέλεσης του παραγωγού παρουσιάζεται όταν οι δύο οντότητες βρίσκονται στο ίδιο πακέτο πυρήνων. Εδώ φαίνεται η βελτιστοποίηση που έχει συμπεριλάβει η κατασκευαστική εταιρία πάνω στους επεξεργαστές της, όπου για δεδομένα που είναι κοινά μόνο σε πυρήνες του ιδίου πακέτου το πρωτόκολλο συνοχής εφαρμόζεται τοπικά χωρίς να εκδίδονται μηνύματα πάνω στον δίαυλο του συστήματος. Σημαντικό ρόλο στην περίπτωση αυτή έχει και η κοινή L2 που διαμοιράζονται οι δύο πυρήνες, αφού η επικοινωνία μεταξύ τους επιτυγχάνεται μέσα από αυτή, μειώνοντας σημαντικά τον χρόνο ανταλλαγής δεδομένων.

Αντίθετα, οι χρόνοι του καταναλωτή δεν παρουσιάζουν δραματικές αλλαγές αφού όπως διαφαίνεται από τον πίνακα κυμαίνονται στα ίδια πλαίσια. Η συμπεριφορά αυτή οφείλεται στο γεγονός ότι το πρωτόκολλο συνοχής δεν έχει κανένα αντίκτυπο στην λειτουργία του καταναλωτή. Μια μικρή διαφορά παρουσιάζεται στους χρόνους αυτούς όταν οι δύο οντότητες τρέχουν σε διαφορετικούς επεξεργαστές, όπου η επικοινωνία τους και η ανταλλαγή των δεδομένων γίνεται διαμέσου των δύο διαφορετικών διαύλων που ανήκουν σε κάθε επεξεργαστή, αυξάνοντας έτσι τον χρόνο μεταφοράς δεδομένων και μηνυμάτων μεταξύ τους.

5.1.2.2 Μέγεθος Δεδομένων 4096KB

CPU ID Παραγωγού	CPU ID Καταναλωτή	Χρόνος επεξεργασίας block Καταναλωτή (usecs)	Χρόνος επεξεργασίας block Παραγωγού (usecs)
0	1	0.81	2.25
0	2	7.35	2.27
0	3	7.36	2.27
0	4	7.38	2.92
0	5	7.43	2.92
0	6	7.48	2.93
0	7	7.39	2.9

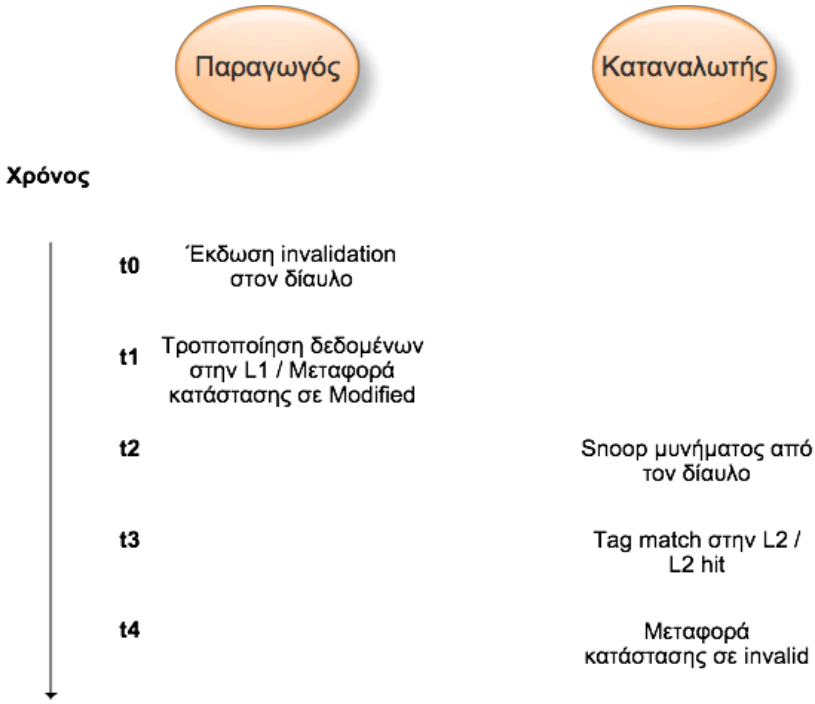
Πίνακας 5.2 Χρόνοι εκτέλεσης πειράματος με δεδομένα μεγέθους 4096KB

Στο παρόν σενάριο γίνεται έλεγχος της επικοινωνίας μεταξύ του παραγωγού και του καταναλωτή με την χρήση δεδομένων μεγέθους 4096KB. Η διαρρύθμιση αυτή

καλείται να κάνει έλεγχο της επικοινωνίας μεταξύ των L2 κρυφών μνήμων των δύο πυρήνων. Παρόμοια με το πιο πάνω πείραμα, οι χρόνοι περιλαμβάνουν γεγονότα που παρουσιάζονται από το Σχήμα 5.3.

Στην συγκεκριμένη περίπτωση, τα δεδομένα είναι πολύ μεγαλύτερα από την L1 κρυφή μνήμη των επεξεργαστών, αλλά αρκετά μικρά έτσι ώστε να βρίσκονται στην L2 κρυφή μνήμη τους. Επομένως, επίσης παρουσιάζεται invalidation κάθε cache line που τροποποιείται από τον παραγωγό. Από την πλευρά του καταναλωτή, μετά από το snoop που εκτελεί πάνω στον δίαυλο επικοινωνίας, γίνεται tag match στην L2 του το οποίο ακολουθείται από L2 hit. Σε αντίθεση με το προηγούμενο σενάριο, η L1 κρυφή μνήμη παραμένει άθικτη, αφού το cache line είναι σίγουρο ότι δεν υπάρχει σε αυτή. Επομένως ακυρώνονται μόνο τα δεδομένα της L2 του καταναλωτή.

Ο χρόνος του καταναλωτή όταν διαβάζει το block αποτελείται από τα γεγονότα όπως περιγράφονται στο προηγούμενο πείραμα, αφού η lockstep εκτέλεση των δύο οντοτήτων μαζί με το μικρό μέγεθος του κάθε block (8KB) διατηρεί την ίδια συμπεριφορά γεγονότων, ανεξάρτητα από το μέγεθος δεδομένων του πειράματος.



Σχήμα 5.3 Τροποποίηση ενός cache line από τον παραγωγό (4096KB μέγεθος δεδομένων)

Η διαφοροποίηση αυτή στα γεγονότα που προκαλούνται από το πρωτοκόλλου συνοχής μεταξύ των δύο περιπτώσεων φαίνεται να έχει σημαντική μείωση του χρόνου εκτέλεσης του παραγωγού. Συγκρίνοντας τους αντίστοιχους χρόνους των δύο πινάκων βλέπουμε ότι η αποχή του ελέγχου της L1 κρυφής μνήμης του επεξεργαστή που βρίσκεται ο καταναλωτής μετά από κάποιο invalidation που προκαλείται από τον παραγωγό, έχει σαν αποτέλεσμα την μείωση του χρόνου του παραγωγού στο μισό.
Παρόλα αυτά, εξετάζοντας τους χρόνους του Πίνακα 5.2 βλέπουμε να διατηρείται το ίδιο μοτίβο στην συμπεριφορά τους όπως και στο προηγούμενο πείραμα. Διακρίνεται και σε αυτή την περίπτωση η βελτιστοποίηση που υπάρχει στο πρωτόκολλο συνοχής για κοινά δεδομένα πυρήνων στο ίδιο πακέτο, όπου παρόμοια παρουσιάζεται και ο μικρότερος χρόνος του παραγωγού. Οι χρόνοι του καταναλωτή παραμένουν οι ίδιοι, μη επηρεαζόμενοι από την αύξηση του μεγέθους των δεδομένων.

5.1.2.3 Μέγεθος Δεδομένων 61440KB

CPU ID Παραγωγού	CPU ID Καταναλωτή	Χρόνος επεξεργασίας block Καταναλωτή (usecs)	Χρόνος επεξεργασίας block Παραγωγού (usecs)
0	1	2.94	2.04
0	2	2.96	2.08
0	3	2.96	2.08
0	4	2.95	2.48
0	5	2.95	2.48
0	6	2.93	2.47
0	7	2.92	2.47

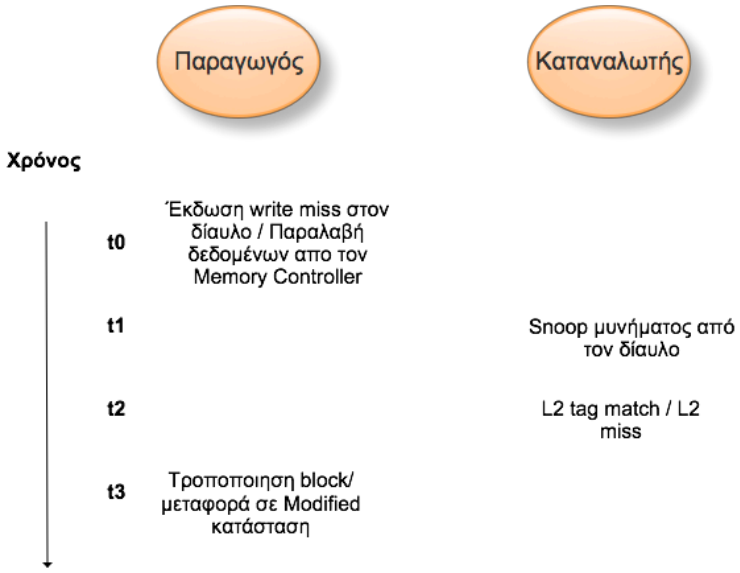
Πίνακας 5.3 Χρόνοι εκτέλεσης πειράματος με δεδομένα μεγέθους 61440KB

Ο πιο πάνω πίνακας παρουσιάζει τα αποτελέσματα της εκτέλεσης του πειράματος με μέγεθος δεδομένων 614400KB. Έχει επιλεχτεί το μέγεθος αυτό έτσι ώστε να είναι αρκετά μεγαλύτερο από την L2 κρυφή μνήμη που βρίσκεται στους επεξεργαστές του συστήματος. Σαν αποτέλεσμα, το block που τίθεται προς επεξεργασία κάθε φορά από τον παραγωγό δεν βρίσκεται σε καμιά από τις κρυφές μνήμες των δύο επεξεργαστών που λαμβάνουν μέρος στην εκτέλεση, απαιτώντας έτσι την επικοινωνία με την κύρια μνήμη του συστήματος.

Κατά την επεξεργασία κάθε block από τον παραγωγό, εφόσον δεν υπάρχει μέσα στην τοπική L2 του εκδίδεται μήνυμα write miss πάνω στον δίαυλο. Ο παραγωγός λαμβάνοντας το μήνυμα αυτό εκτελεί tag match πάνω στο tag array της τοπικής του L2, και όπως είναι αναμενόμενο ακολουθείται από L2 miss. Επομένως το block των δεδομένων αποστέλλεται στον παραγωγό από τον memory controller του συστήματος. Αφού τα παραλάβει, με την σειρά του εφαρμόζει τις τροποποιήσεις πάνω σε αυτά και επομένως τα δεδομένα βρίσκονται στην Modified κατάσταση μέσα στην L1 του παραγωγού. (Σχήμα 5.4)

Οι ενέργειες που γίνονται με το διάβασμα των τροποποιημένων δεδομένων από τον καταναλωτή είναι οι ίδιες με αυτές των προηγούμενων πειραμάτων χωρίς καμία αλλαγή.

Παρά την εμπλοκή της κύριας μνήμης, σε αυτή την περίπτωση βλέπουμε ότι η αποχή του πρωτοκόλλου συνοχής μνήμης μεταξύ των δύο επεξεργαστών έχει σαν αποτέλεσμα την μείωση του χρόνου εκτέλεσης του παραγωγού. Συγκρίνοντας τις αντίστοιχες τιμές των πινάκων του παρόντος πειράματος με το προηγούμενο, παρατηρούμε την μείωση του χρόνου του παραγωγού να γίνεται σε επίπεδα του 3. Φαίνεται λοιπόν, ότι το αντίκτυπο που έχει ο μηχανισμός συνοχής μνήμης πάνω σε κάποιο πρόγραμμα, είναι πολύ μεγαλύτερος από τον χρόνο καθυστέρησης της κύριας μνήμης.



Σχήμα 5.4 Τροποποίηση ενός cache line από τον παραγωγό (61440KB μέγεθος δεδομένων)

5.1.3 Συμπεράσματα

Το ποιο πάνω πείραμα έχει σαν στόχο την περιγραφή της επικοινωνίας που επιτυγχάνεται μεταξύ δύο επεξεργαστών του συστήματος, όταν τα δεδομένα που είναι κοινά προς αυτούς βρίσκονται στις διάφορες αρχιτεκτονικές δομές των επεξεργαστών.

Βλέπουμε ότι στην περίπτωση πολύ μικρού μεγέθους δεδομένων, αυτά μπορούν να βρίσκονται στο πιο ψηλό επίπεδο της ιεραρχίας της μνήμης. Σαν αποτέλεσμα, το πρωτόκολλο συνοχής πρέπει να περάσει από όλα τα επίπεδα ιεραρχίας μνήμης έτσι ώστε να εξασφαλίσει την συνοχή τους. Η λειτουργία αυτή γίνεται σειριακά σε κάθε επίπεδο, με αποτέλεσμα να προσθέτει μεγάλη καθυστέρηση πάνω στον χρόνο εκτέλεσης του προγράμματος.

Αυξάνοντας το μέγεθος των δεδομένων, το πρωτόκολλο συνοχής εκτελείται μόνο σε χαμηλότερα επίπεδα της ιεραρχίας, όπου και μειώνεται σημαντικά ο χρόνος εκτέλεσης του προγράμματος. Παραδόξως, από τα αποτελέσματα των τριών περιπτώσεων διακρίνεται ότι σε προγράμματα που υλοποιούν streaming συμπεριφορά, φαίνεται ότι η χρήση κοινών δεδομένων μεταξύ επεξεργαστών μειώνει

σημαντικά την απόδοση του προγράμματος αφού για κάθε τροποποίηση τους χρειάζεται να υλοποιηθεί συνοχή σε αυτά. Εντούτοις, σε μεγάλου μεγέθους δεδομένα όπου δεν μπορούν να χωρέσουν στις κρυφές μνήμες των επεξεργαστών, και επομένως παρουσιάζεται cash miss στο χαμηλότερο επίπεδο κρυφής μνήμης των επεξεργαστών μετά από κάποιο tag match, τότε η παραλαβή των δεδομένων από την μνήμη παρουσιάζει καλύτερα αποτελέσματα. Στην περίπτωση αυτή, βλέπουμε ότι και ο prefetcher του συστήματος προσφέρει θεμιτά στον χρόνο εκτέλεσης του προγράμματος. Επίσης, η έλλειψη συμφόρησης πάνω στους διαύλους επικοινωνίας του συστήματος λόγω της lockstep εκτέλεσης των δύο οντοτήτων του προγράμματος, όπως και η παρουσία fb-dimms σε αυτό φαίνεται να παίζουν σημαντικό ρόλο στον χρόνο εκτέλεσης.

Κεφάλαιο 6

Ομάδα προγραμμάτων συγκριτικής επιδόσεως PARSEC

Η αξιολόγηση του συστήματος βασίζεται στα προγράμματα που απαρτίζουν το Parsec Benchmark Suit, προϊόν του πανεπιστημίου Princeton. Σκοπός του benchmark είναι ο καθορισμός μιας ομάδας προγραμμάτων τα οποία μπορούν να χρησιμοποιηθούν για την αξιολόγηση και την κατασκευή μελλοντικών πολυεπεξεργαστών, παρέχοντας αντιπροσωπευτικές μετρήσεις για τον σκοπό αυτό.

Τον πιο πάνω στόχο το benchmark τον πετυχαίνει παρέχοντας παράλληλα προγράμματα, τα οποία μπορούν να εκμεταλλευτούν τους πολυπυρήνες των επεξεργαστών. Επιπλέον, τα προγράμματα αυτά χαρακτηρίζονται από πληθώρα φόρτου εργασίας, επικεντρώνονται σε διάφορους τομείς και χρησιμοποιούν σύγχρονες τεχνικές, δίνοντας έτσι ένα αντιπροσωπευτικό δείγμα των εργασιών που εκτελούνται από τους μοντέρνους επεξεργαστές.

Τα ποιο πάνω χαρακτηριστικά απουσιάζουν από τα υπάρχοντα benchmarks τα οποία περιορίζονται σε μερικούς τομείς προγραμμάτων (BioParallel, PhysicsBench κτλ.) ή δεν παρέχουν παράλληλα προγράμματα (SPEC CPU2006, OMP2001), και επομένως δεν μπορούν να χρησιμοποιηθούν για την αξιολόγηση παράλληλων μηχανών. Το γεγονός αυτό αποτέλεσε κίνητρο για την δημιουργία του Parsec, το οποίο έχει σαν στόχο να παρουσιάσει προγράμματα αντιπροσωπευτικά για τις σημερινές ανάγκες. Ποιο κάτω παρουσιάζεται πίνακας με τα χαρακτηριστικά των προγραμμάτων που αποτελούν το benchmark.

Program	Domain	Parallelization Model	Working Set	Data Sharing	Data Exchange
blackscholes	Financial Analysis	Data-parallel	Small	Low	Low
bodytrack	Computer Vision	Data-parallel	Medium	High	Medium
canneal	Engineering	Unstructured	Unbounded	High	High
dedup	Enterprise Storage	Pipeline	Unbounded	High	High
facesim	Animation	Data-Parallel	Large	Low	Medium
ferret	Similarity Search	Pipeline	Unbounded	High	High
fluidanimate	Animation	Data-Parallel	Large	Low	Medium
freqmine	Data Mining	Data-Parallel	Unbounded	High	Medium
streamcluster	Data Mining	Data-Parallel	Medium	Low	Medium
swaptions	Financial Analysis	Data-Parallel	Medium	Low	Low
vips	Media Processing	Data-Parallel	Medium	Low	Medium
x264	Media Processing	Pipeline	Medium	High	High

Πίνακας 6.1: Χαρακτηριστικά προγραμμάτων του Parsec Benchmark Suit

Από τον πιο πάνω πίνακα φαίνεται καθαρά η πληθώρα στους τομείς των προγραμμάτων του benchmark καθώς επίσης και το μοντέλο παραλληλισμού που χρησιμοποιούν. Σημαντικά είναι τα χαρακτηριστικά του διαμοιρασμού των δεδομένων καθώς επίσης και της επικοινωνίας που υλοποιείται σε κάθε πρόγραμμα, χαρακτηριστικά τα οποία βοηθούν στην αξιολόγηση της ιεραρχίας μνήμης καθώς επίσης και την απόδοση της επικοινωνίας που επιτυγχάνεται στους προς αξιολόγηση επεξεργαστές.

Ανάλυση για τα προγράμματα αυτά έχει γίνει από τους δημιουργούς του Benchmark Suit σε θεωρητικό επίπεδο, με την χρήση προσομοιωτή [1]. Η παρούσα εργασία τίθεται να χρησιμοποιήσει τα προγράμματα αυτά για την αξιολόγηση πραγματικού συστήματος. Για τον σκοπό αυτό, έχουν επιλεχθεί τέσσερις από τις εφαρμογές, οι οποίες περιγράφονται στο ακόλουθο μέρος της διπλωματικής εργασίας.

6.1 Μεθοδολογία

Για την ανάλυση των εφαρμογών, ακολουθούνται κάποια βήματα τα οποία καθορίζουν την μεθοδολογία που έχει ακολουθηθεί. Πρωταρχικό στάδιο της μεθοδολογίας είναι η στατική ανάλυση του κώδικα και η εξεύρεση της πολυπλοκότητας της κάθε συνάρτησης. Στο στάδιο αυτό λαμβάνονται υπόψη και οι διάφορες μορφές επικοινωνίας που παρουσιάζονται μεταξύ των νημάτων εκτέλεσης που δημιουργεί η εφαρμογή. Επίσης, σε πολύπλοκους αλγόριθμους χρησιμοποιούνται βοηθητικά έγγραφα που περιγράφουν τους αλγορίθμους, έτσι ώστε να επιτευχθεί καλύτερη κατανόηση προς αυτούς.

Στην συνέχεια, γίνεται δυναμική ανάλυση του κώδικα με τα διάφορα πειράματα. Στο στάδιο αυτό γίνονται μετρήσεις των χρόνων των διαφόρων συναρτήσεων της εφαρμογής έτσι ώστε αυτοί να συγκριθούν κατά τις διάφορες παραμέτρους των πειραμάτων.

Ο συνδυασμός της στατικής και δυναμικής ανάλυσης μας παρέχει ενδείξεις για τους παράγοντες που επηρεάζουν την εφαρμογή. Με βάση τις ενδείξεις αυτές, γίνονται τροποποιήσεις στον κώδικα, ή στα δεδομένα εισόδου έτσι ώστε να αποδειχθεί η ορθότητα της κάθε μίας. Στην περίπτωση τροποποίησης του κώδικα, ο νέος κώδικας μπορεί να είναι λανθασμένος σε σύγκριση με τον αλγόριθμο που χρησιμοποιεί η εφαρμογή. Παρόλα αυτά, τα αποτελέσματα του τροποποιημένου κώδικα μπορούν να χρησιμοποιηθούν έτσι ώστε για να εξευρεθεί ο παράγοντας που επηρεάζει την απόδοση της εφαρμογής.

Για τους σκοπούς τον μετρήσεων, χρησιμοποιείται ο συνολικός χρόνος εκτέλεσης της κάθε εφαρμογής για τα διάφορα πειράματα και στην συνέχεια υπολογίζεται το speedup της παράλληλης εφαρμογής σε σχέση με τον αντίστοιχο σειριακό κώδικα της.

Επιπλέον τροποποιήσεις που γίνονται στις σταθερές παραμέτρους μέσα στον κώδικα, καθώς επίσης και τις διάφορες βοηθητικές εντολές που παρέχει το Parsec για την μεταγλώττιση και εκτέλεση των εφαρμογών επεξηγούνται στο Παράρτημα Β.

6.2 Fluidanimate

Το πρόγραμμα Fluidanimate έχει συμπεριληφθεί στην ομάδα προγραμμάτων συγκριτικής μέτρησης επιδόσεως PARSEC (PARSEC Benchmark Suit) έτσι ώστε να αντιπροσωπεύσει τους αλγορίθμους φυσικής που χρησιμοποιούνται στα σύγχρονα ηλεκτρονικά παιχνίδια, καθώς επίσης και την ομάδα προγραμμάτων που υλοποιούν αλγορίθμους γραφικών πραγματικού χρόνου. Το αναφερόμενο πρόγραμμα είναι προϊόν της εταιρίας Intel, το οποίο χρησιμοποιείται για την αξιολόγηση των επεξεργαστών της εταιρίας. Ο αλγόριθμος που υλοποιεί το πρόγραμμα είναι μια επέκταση της μεθόδου Smooth Particle Hydrodynamics (SPH) ο οποίος προσομοιώνει την κίνηση ενός υγρού στοιχείου στα πλαίσια ενός διαδραστικού περιβάλλοντος, στοιχείο που χαρακτηρίζει τα σύγχρονα ηλεκτρονικά παιχνίδια.

6.2.1 Περιγραφή Αλγορίθμου

Η αναπαράσταση του υγρού στοιχείου γίνεται με την χρήση μορίων (particles) τα οποία αναπαριστούν την κατάσταση του υγρού σε διακριτές τοποθεσίες του όγκου που περιλαμβάνει. Η σκηνή της προσομοίωσης χαρακτηρίζεται από ένα κουτί, μέσα στο οποίο βρίσκεται το υγρό στοιχείο που τίθεται προς επεξεργασία. Η διαχείριση των συγκρούσεων που παρουσιάζονται μεταξύ των μορίων του στοιχείου γίνεται με την χρήση δυνάμεων που ασκούνται μεταξύ τους έτσι ώστε να υπολογιστεί η φορά κίνησης των μορίων. Με αυτό τον τρόπο αποφεύγεται η διαχείριση της κίνησης των μορίων με βάση την ταχύτητα τους, αφού σε κάθε στάδιο της προσομοίωσης, γνωρίζοντας την προηγούμενη και την νέα θέση των μορίων η ταχύτητα υπολογίζεται έμμεσα με interpolation μεταξύ των δύο θέσεων. Αφού υπολογιστεί η ταχύτητα, συνδυάζεται με την μάζα και την δύναμη που ασκείται πάνω στα μόρια έτσι ώστε να υπολογιστεί η επιτάχυνση τους και επομένως η νέα τους θέση.

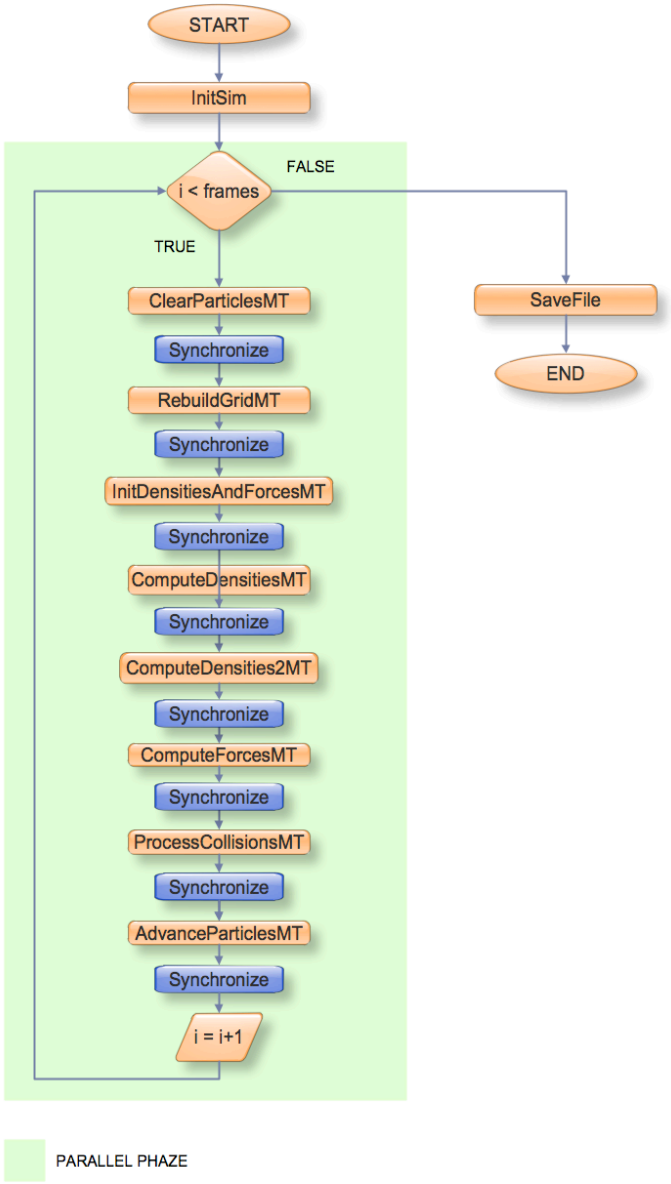
Ο μηχανισμός που χρησιμοποιείται από τον αλγόριθμο, περιορίζει τα μόρια έτσι ώστε αυτά να μπορούν να αλληλεπιδρούν με άλλα μόρια τα οποία δεν ξεπερνούν κάποια απόσταση h. Εκμεταλλευόμενος του γεγονότος, τα μόρια ομαδοποιούνται σε κελιά έτσι ώστε να μειώνεται ο έλεγχος και η επεξεργασία μορίων που αλληλεπιδρούν μεταξύ τους. Επιπλέον, ο κώδικας περιορίζει τον αριθμό των ατόμων που

ομαδοποιούνται, έτσι ώστε κάθε κελί να έχει το πολύ δεκαέξι μόρια. Ο διαμερισμός αυτός του χώρου υλοποιείται από την συνάρτηση InitSim κατά την εκκίνηση του προγράμματος, καθώς επίσης και από τις συναρτήσεις ClearParticlesMT και RebuildGridMT σε κάθε στάδιο της προσομοίωσης. Η ανάθεση των δεδομένων κάθε νήματος (thread) που δημιουργείται από το πρόγραμμα, γίνεται στο επίπεδο κελιών, διαδικασία που συμπεριλαμβάνεται στην εκτέλεση της συνάρτησης InitSim.

Πρώτο βήμα του αλγορίθμου είναι ο υπολογισμός της πυκνότητας του υγρού, στις θέσεις όπου παρουσιάζονται τα μόρια του. Σε αυτό το στάδιο πρέπει να ληφθούν υπόψη και μόρια των γειτονικών κελιών του μορίου που επεξεργάζεται ο αλγόριθμος, αφού σε περιοχές όπου τα μόρια είναι μαζεμένα τότε η πυκνότητα του εν λόγω σημείου θα είναι μεγάλη. Η λειτουργία αυτή διαχωρίζεται σε τρεις διαδοχικές διεργασίες, οι οποίες εκτελούνται από τις συναρτήσεις InitDensitiesAndForcesMT, ComputeDensitiesMT και ComputeDensities2MT αντίστοιχα.

Με τον υπολογισμό της πυκνότητας σε κάθε σημείο όπου βρίσκονται τα μόρια, μπορεί στην συνέχει να χρησιμοποιηθεί έτσι ώστε να υπολογιστούν οι δυνάμεις που ασκούνται σε κάθε ένα από αυτά. Η διαδικασία αυτή υλοποιείται με την συνάρτηση ComputeForcesMT. Στην συνέχεια, γίνεται υπολογισμός των συγκρούσεων που παρουσιάζονται μεταξύ των μορίων από την συνάρτηση ProcessCollisionsMT. Τέλος, με την χρήση των δυνάμεων που ασκούνται στα μόρια υπολογίζεται η επιτάχυνση κάθε μορίου και ενημερώνεται η νέα του θέση, λειτουργία που υλοποιείται με την συνάρτηση AdvanceParticlesMT.

Η πιο πάνω εκτέλεση εφαρμόζεται επαναληπτικά για κάθε στάδιο της προσομοίωσης, παράμετρος η οποία εκφράζεται σαν frames τα οποία εισάγονται σαν παράμετρος στο πρόγραμμα. Η παράμετρος αυτή μεταβιβάζεται στα νήματα εκτέλεσης που δημιουργούνται, και το κάθε ένα εκτελεί τις πιο πάνω συναρτήσεις για όλα τα frames, πάνω στα δεδομένα που του έχουν ανατεθεί από την συνάρτηση InitSim. Ενδιάμεσα σε κάθε μια από τις συναρτήσεις που εκτελούν τα νήματα, υλοποιείται barrier για να διεξασφαληθεί η ορθότητα της σειράς εκτέλεσης. Με αυτό τον τρόπο, όλα τα νήματα πρέπει να τελειώσουν την εκτέλεση μιας συνάρτησης, έτσι ώστε να αρχίσει η εκτέλεση της συνάρτησης από αυτά. (Σχήμα 6.1)



Σχήμα 6.1: Διάγραμμα ροής αλγορίθμου

6.2.2 Διαχωρισμός Δεδομένων

Τα μόρια της προσομοίωσης ομαδοποιούνται σε κελιά, τα οποία υπολογίζονται κατά την εκτέλεση του προγράμματος. Τα κελιά αυτά χαρακτηρίζονται από την θέση τους στις τρεις διαστάσεις του χώρου. Η ανάθεση των κελιών στα διάφορα νήματα εκτέλεσης που δημιουργεί η προσομοίωση γίνεται ανάμεσα στην X και Z διάσταση, ανάλογα με τον αριθμό των νημάτων που παίρνει σαν παράμετρο το πρόγραμμα.

Ο ποιο πάνω διαχωρισμός γίνεται με βάση την τετραγωνική ρίζα του αριθμού των νημάτων που δημιουργούνται. Συγκεκριμένα, υπολογίζεται η τετραγωνική ρίζα του αριθμού αυτού, αποκόπτοντας οποιαδήποτε δεκαδικά ψηφία προκύψουν. Η τιμή αυτή χαρακτηρίζει τον αριθμό των partitions που θα δεχτούν τα κελιά των δύο διαστάσεων. Αν το πηλίκο του αριθμού αυτού παράγει τον αριθμό των νημάτων εκτέλεσης, τότε το πρόγραμμα συνεχίζει στο επόμενο στάδιο. Αν όχι, τότε ο αριθμός αυτός διπλασιάζεται, και η νέα τιμή χαρακτηρίζει τον αριθμό των partitions που θα δεχτεί η διάσταση X. Επομένως, στην περίπτωση όπου ο αριθμός των νημάτων εκτέλεσης είναι οκτώ, τότε οι τιμή για την Z διάσταση είναι δύο, ενώ για την X είναι τέσσερα.

Αφού υπολογιστούν οι τιμές, γίνεται η ανάθεση των κελιών στα διάφορα νήματα εκτέλεσης. Στον Πίνακα 6.2 παρουσιάζεται ο διαχωρισμός των κελιών μεταξύ οκτώ νημάτων εκτέλεσης όταν το μέγεθος των διαστάσεων X και Z είναι πενήντα πέντε, ενώ της Y διάστασης εβδομήντα έξι.
Παρά τον λογικό αυτό διαχωρισμό, τα κελιά μαζί με τα μόρια που τα αποτελούν αποθηκεύονται σε ένα μονοδιάστατο πίνακα, ξεκινώντας από το κελί της διάστασης Z = 0, Y = 0, X = 0. (Σχήμα 6.2)

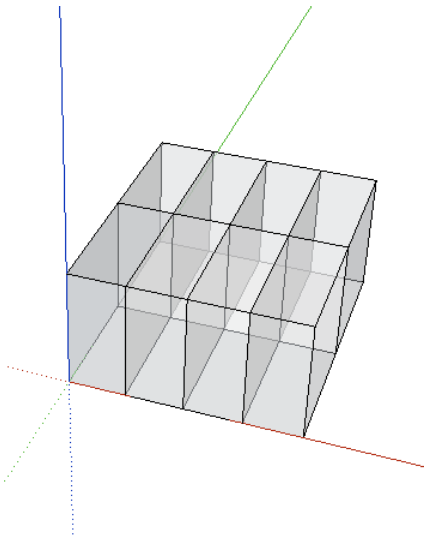
0,0,0	0,0,1	.....	0,1,0	0,1,1	.....	1,0,0	1,0,1	.....	X,Y,Z

Σχήμα 6.2: Αναπαράσταση κελιών στην μνήμη με βάση την διάσταση τους στον χώρο (X,Y,Z)

Γνωρίζοντας κάθε νήμα εκτέλεσης τις διαστάσεις των κελιών που του έχουν ανατεθεί, καθώς επίσης και το συνολικό μέγεθος κάθε διάστασης, είναι σε θέση να αντιστοιχίσει την λογική τοποθεσία ενός κελιού όπως καθορίζεται από τις τρεις διαστάσεις του στον χώρο, στην διεύθυνση που του αντιστοιχεί στον μονοδιάστατο πίνακα.
Λόγω του διαχωρισμού των κελιών σε διάφορα νήματα εκτέλεσης, και της ανάγκης του αλγορίθμου για εξεύρεση πληροφοριών από γειτονικά κελιά, τότε χρησιμοποιούνται mutexes στα μόρια των κελιών που βρίσκονται στα όρια των διαστάσεων που έχουν ανατεθεί σε κάθε νήμα εκτέλεσης.

Thread ID	X Cell Range	Y Cell Range	Z Cell Range
0	0 - 13	0 - 75	0 - 27
1	0 - 13	0 - 75	28 - 54
2	14 - 27	0 - 75	0 - 25
3	14 - 27	0 - 75	28 - 54
4	28 - 40	0 - 75	0 - 27
5	28 - 40	0 - 75	28 - 54
6	41 - 54	0 - 75	0 - 27
7	41 - 54	0 - 75	28 - 54

Πίνακας 6.2: Παράδειγμα διαχωρισμού κελιών μεταξύ οκτώ νημάτων εκτέλεσης



Σχήμα 6.3: Σχεδιάγραμμα διαχωρισμού κελιών μεταξύ οκτώ νημάτων εκτέλεσης

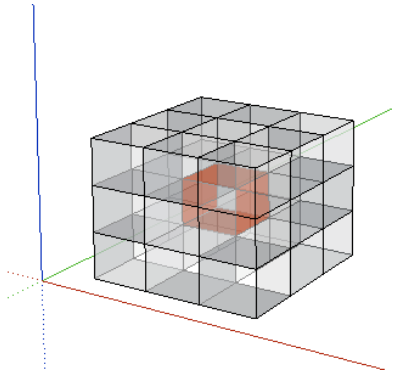
6.2.3 Πολυπλοκότητα Αλγορίθμου

Η ανάλυση του κώδικα και της πολυπλοκότητας του είναι ένα σημαντικό στάδιο που χρειάζεται να γίνεται στο χαρακτηρισμό της απόδοσης ενός προγράμματος. Κάνοντας χρήση της πολυπλοκότητας, λαμβάνοντας μαζί υπόψη τις διάφορες παραμέτρους που επηρεάζουν αρνητικά ένα παράλληλο πρόγραμμα, είμαστε σε θέση να κατανοήσουμε την συμπεριφορά του στη μηχανή που εξετάζεται.

Όνομα Συνάρτησης	Πολυπλοκότητα
ClearParticlesMT	c
RebuildGridMT	c*p*λ
InitDensitiesAndForcesMT	c*p
ComputeDensitiesMT	c*p*n*p*λ
ComputeDensities2MT	c*p
ComputeForcesMT	c*p*n*p*λ
ProcessCollisionsMT	c*p
AdvanceParticlesMT	c*p

Πινακας 6.3 : Πολυπλοκότητα Συναρτήσεων Εφαρμογής

Ο πιο πάνω πίνακας παρουσιάζει την πολυπλοκότητα των συναρτήσεων της εφαρμογής. Οι διάφορες παράμετροι που χρησιμοποιούνται για την έκφραση της πολυπλοκότητας επεξηγούνται στον Πίνακα 6.4. Παρατηρώντας την ανάλυση των συναρτήσεων βλέπουμε μια ανισότιμη συμπεριφορά στο μέγεθος των δεδομένων, και κατ' επέκταση του χρόνου εκτέλεσης της κάθε συνάρτησης. Αξιοσημείωτη είναι η συμπεριφορά των συναρτήσεων ComputeDensitiesMT και ComputeForcesMT, όπου λαμβάνονται υπόψη οι τιμές γειτονικών κελιών για κάθε κελί που δέχεται επεξεργασία. Οι άμεσοι γείτονες του κελιού χαρακτηρίζονται από την τοποθεσία τους στον τρισδιάστατο χώρο, ελέγχοντας τις συντεταγμένες x-1, x+0, x+1, y-1, y+0, y + 1, z-1, z+0, z+1, όπου x,y,z είναι οι συντεταγμένες του κελιού που γίνεται αναφορά (Σχήμα 6.4). Είναι αναμενόμενο λοιπόν να υπάρχει επικοινωνία μεταξύ επεξεργαστών που διαχειρίζονται διαφορετικά δεδομένα, στην περίπτωση της επεξεργασίας κελιών που βρίσκονται στα άκρα των διαστάσεων που έχουν ανατεθεί στα εν λόγω νήματα επεξεργασίας. Η επικοινωνία αυτή χρειάζεται προστασία, και επιτυγχάνεται με την χρήση mutex έτσι ώστε να μην επιτρέπεται η ταυτόχρονη πρόσβαση ή/και τροποποίηση μορίων που βρίσκονται στα κελιά αυτά. Ο χρόνος που χρειάζεται για την υλοποίηση της προστασίας αυτής εκφράζεται από την παράμετρο λ.



Σχήμα 6.4: Σχεδιάγραμμα γειτονικών κελιών. Στον πυρήνα του κύβου βρίσκεται το κελί αναφοράς.

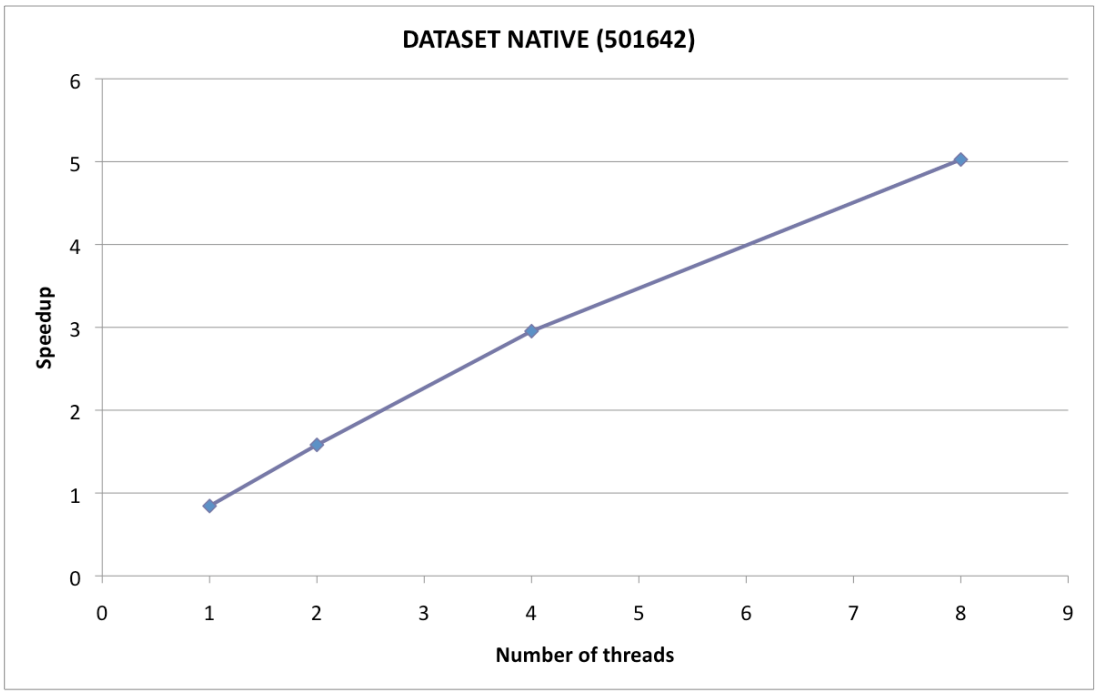
Σύμβολο	Υπολογισμός	Επεξήγηση
nx		Μέγεθος διάστασης X
ny		Μέγεθος διάστασης Y
nz		Μέγεθος διάστασης Z
XDIVS		Αριθμός partitions που εφαρμόζεται στα κελιά της X διάστασης όπως περιγράφεται στο σημείο 6.1.2
ZDIVS		Αριθμός partitions που εφαρμόζεται στα κελιά της Z διάστασης όπως περιγράφεται στο σημείο 6.1.2
c	(nx/XDIVS)*ny*(nz/ ZDIVS)	Αριθμός κελιών που ανατίθεται σε κάθε νήμα εκτέλεσης
p	0 - 16	Αριθμός μορίων που υπάρχει σε ένα κελί.
n	0 - 27	Αριθμός γειτονικών κελιών που έχει το κελί που τίθεται προς επεξεργασία
λ		Χρόνος που χρειάζεται για να ολοκληρωθεί επικοινωνία μεταξύ δεδομένων δύο επεξεργαστών.

Πινακας 6.4 : Επεξήγηση Συμβόλων Πολυπλοκότητας Συναρτήσεων

6.2.4 Αποτελέσματα πειραμάτων

Το πρώτο πείραμα που έχει διεξαχθεί, είναι η εκτέλεση του προγράμματος με βάση τα δεδομένα εισόδου που παρέχονται από το Parsec για την συγκεκριμένη εφαρμογή. Τα δεδομένα αποτελούνται από 501642 μόρια, τα οποία είναι διασκορπισμένα στα κελιά που αποτελούν τον χώρο του υγρού. Το σχήμα που ακολουθεί παρουσιάζει τα

αποτελέσματα του speedup της εφαρμογής όταν αυτή συγκρίνεται με τον χρόνο εκτέλεσης της σειριακής εκδοχής της.

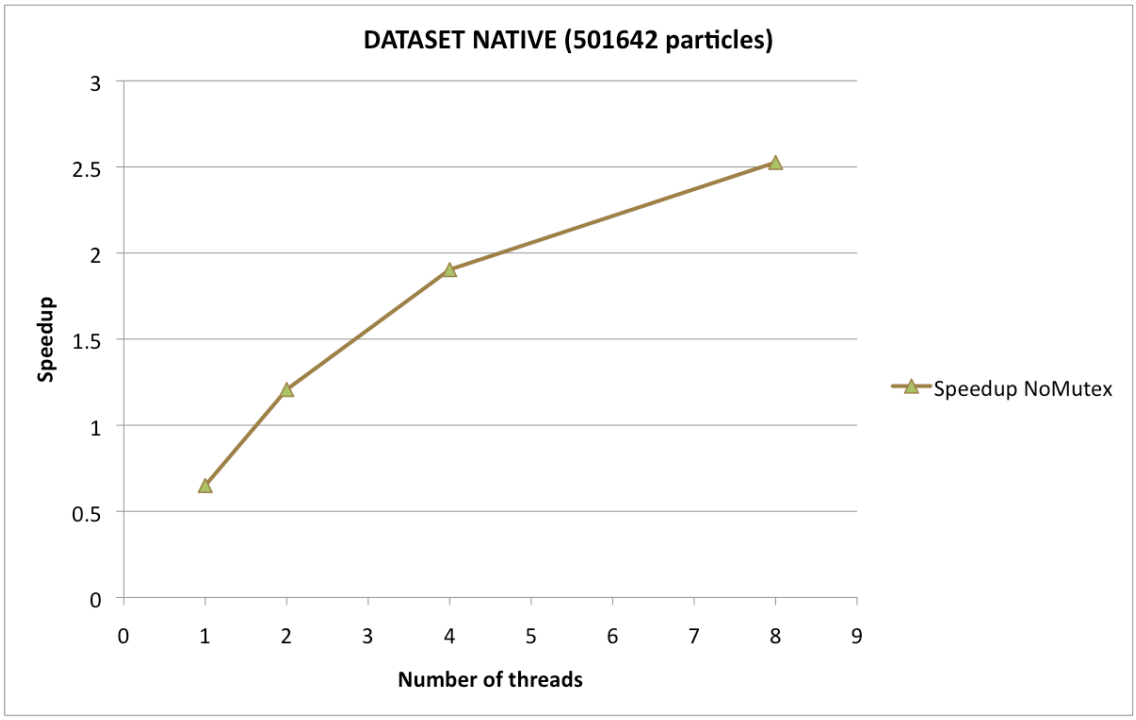


Σχήμα 6.5 Speedup εφαρμογής χρησιμοποιώντας τα δεδομένα εισόδου που παρέχονται από το Parsec

Παρατηρούμε ότι η επίδοση της εφαρμογής μειώνεται όταν αυξάνεται ο αριθμός των νημάτων εκτέλεσης της. Το γεγονός αυτό μπορεί να οφείλεται σε πολλούς παράγοντες. Για την εξακρίβωση του παράγοντα που επηρεάζει τον χρόνο εκτέλεσης της εφαρμογής, γίνονται υποθέσεις, και στην συνέχεια με αλλαγή του κώδικα ή των δεδομένων που εισάγονται στο πρόγραμμα γίνονται ξανά μετρήσεις στην επίδοση της εφαρμογής έτσι ώστε να εξακριβωθεί η ορθότητα των υποθέσεων αυτών.

Αρχικά μπορούμε να υποθέσουμε ότι η μείωση της επίδοσης του προγράμματος είναι αποτέλεσμα της χρήσης mutex στα μόρια των κελιών που βρίσκονται στα όρια μεταξύ γειτονικών νημάτων εκτέλεσης. Η παρουσία mutex, παρόλο που είναι αναγκαία, επηρεάζει σημαντικά τον χρόνο εκτέλεσης ενός παράλληλου προγράμματος, αφού ο κώδικας ο οποίος βρίσκεται μέσα στο κρίσιμο μέρος εκτελείται σειριακά από όλα τα νήματα. Στην παρούσα εφαρμογή, οι συναρτήσεις ComputeDensitiesMT και ComputeForcesMT, εφαρμόζοντας κάποιους υπολογισμούς πάνω σε κάποιο μόριο,

ενημερώνουν τις ανάλογες τιμές των μορίων που βρίσκονται στα γειτονικά κελιά. Η διαδικασία αυτή γίνεται με την χρήση mutex όταν τα γειτονικά κελιά ανήκουν σε διαφορετικό νήμα εκτέλεσης. Για την αποφυγή του κλειδώματος αυτού, έχουν τροποποιηθεί οι συναρτήσεις έτσι ώστε το κάθε μόριο να ενημερώνει μόνο τον εαυτό του, αποφεύγοντας έτσι οποιοδήποτε μηχανισμό συγχρονισμού. Το γειτονικό μόριο, που αρχικά ενημερώνεται από το παρόν μόριο, θα ενημερώσει τον εαυτό του όταν έρθει η σειρά του εκτελώντας τους ανάλογους υπολογισμούς. Στο σχήμα 6.6 φαίνεται το speedup του προγράμματος με αυτές τις τροποποιήσεις.



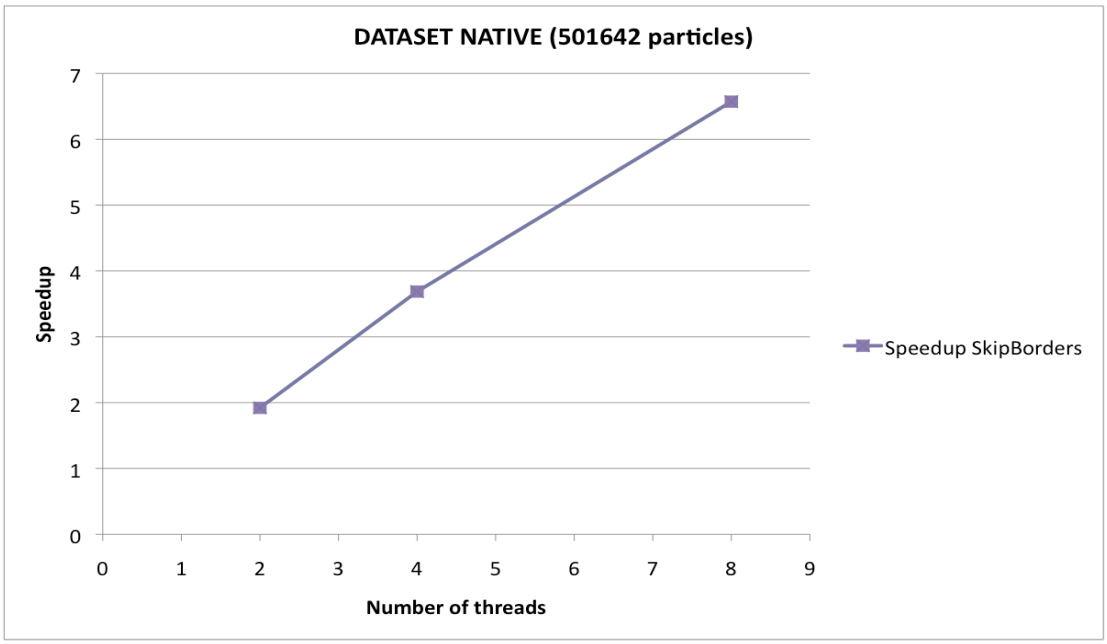
Σχήμα 6.6 Speedup εφαρμογής αποφεύγοντας την χρήση mutex

Βλέπουμε ότι το speedup που παρουσιάζεται με την τροποποίηση αυτή είναι ακόμη χαμηλότερο. Αυτό οφείλεται στο γεγονός ότι έχουν διπλασιαστεί οι υπολογισμοί που εκτελούνται από τις δύο αυτές συναρτήσεις. Συγκεκριμένα, οι συναρτήσεις αυτές εκτελούνται σε όλα τα κελιά του κάθε νήματος, οι πλειοψηφία των οποίων δεν διαμοιράζεται μεταξύ των νημάτων εκτέλεσης. Επομένως, παρόλο που αποφεύγεται η χρήση των mutex για κελιά που βρίσκονται στα όρια μεταξύ των νημάτων, εντούτοις κάθε νήμα εκτελεί διπλάσιο αριθμό υπολογισμών για τα μόρια των κελιών που του

ανήκουν . Βλέπουμε λοιπόν ότι η παρουσία των mutexes στο αρχικό πρόγραμμα δεν είναι ο λόγος από τον οποίο μειώνεται η επίδοση του.

Με βάση την ανάλυση του κώδικα, μπορούμε να διακρίνουμε τις συναρτήσεις RebuildGridMT, ComputeDensitiesMT και ComputeForcesMT να υλοποιούν επικοινωνία μεταξύ τους. Όπως έχουμε δει στην περίπτωση του πειράματος με τον παραγωγό και καταναλωτή, η επικοινωνία αυτή έχει μεγάλο κόστος, αφού τα μνήματα του πρωτοκόλλου μνήμης αυξάνουν σημαντικά τον χρόνο εκτέλεσης του προγράμματος. Επομένως, η παρουσία επικοινωνίας μεταξύ των νημάτων εκτέλεσης είναι πιθανός παράγοντας που επηρεάζει την επίδοση του προγράμματος.

Για τον έλεγχο της υπόθεσης αυτής, έχουν τροποποιηθεί οι συναρτήσεις έτσι ώστε στην περίπτωση ελέγχου γειτονικών κελιών που ανήκουν σε άλλο νήμα εκτέλεσης αυτά αγνοούνται. Με αυτό τον τρόπο, δεν υπάρχει καμία επικοινωνία μεταξύ των νημάτων εκτέλεσης, τα οποία ενεργούν μόνο πάνω στα κελιά που τους ανήκουν. Αντίθετα με την προηγούμενη τροποποίηση του αλγόριθμου, ο οποίος παράγει ορθά αποτελέσματα, ο συγκεκριμένος είναι λανθασμένος. Παρόλα αυτά, τα αποτελέσματα του νέου προγράμματος μπορούν να χρησιμοποιηθούν σαν ένδειξη ορθότητας της υπόθεσης.



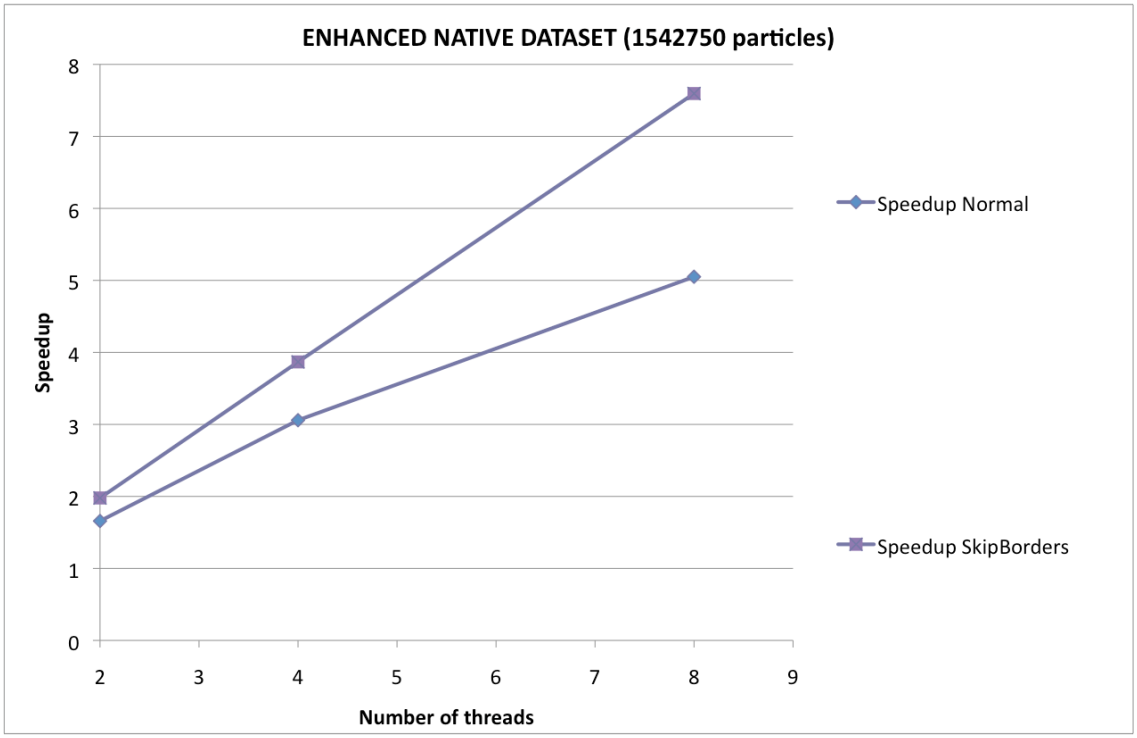
Σχήμα 6.6 Speedup εφαρμογής αγνοώντας κελιά γειτονικών νημάτων εκτέλεσης

Παρατηρούμε ότι με την αποφυγή επικοινωνίας μεταξύ των νημάτων εκτέλεσης, βλέπουμε σημαντική αύξηση του speedup της εφαρμογής. Παρόλα αυτά, και σε αυτή την περίπτωση παρουσιάζεται μείωση της όταν ο αριθμός των νημάτων εκτέλεσης της αυξάνεται. Επομένως, βλέπουμε ότι η παρουσία επικοινωνίας μεταξύ των νημάτων εκτέλεσης αποτελεί ένα από τους λόγους που προκαλούν στην μείωση της απόδοσης του προγράμματος.

Ένας κοινός παράγοντας που επηρεάζει τα παράλληλα προγράμματα είναι αυτός της μεταφοράς δεδομένων από την μνήμη. Στις περιπτώσεις όπου οι υπολογισμοί που γίνονται από τους πυρήνες του συστήματος είναι λίγοι σε σχέση με τις ανάγκες της εφαρμογής για δεδομένα, τότε το overhead της επικοινωνίας με τη μνήμη είναι μεγαλύτερο από το όφελος που παίρνουμε από τον παραλληλισμό. Ο δίαυλος επικοινωνίας που χρησιμοποιείται για την σύνδεση των πυρήνων με τις υπόλοιπες ομάδες του συστήματος, εξυπηρετεί τις αιτήσεις αυτές σειριακά. Με την αύξηση του αριθμού των πυρήνων που χρησιμοποιούνται για την εκτέλεση μιας εφαρμογής, η πιθανότητα ανάγκης για επικοινωνία με την μνήμη σε κάθε ένα από αυτούς αυξάνεται, με αποτέλεσμα να μειώνεται ακόμη περισσότερο το speedup που επιτυγχάνεται από την εφαρμογή.

Ελέγχοντας τα δεδομένα εισόδου που παρέχονται από το Parsec, έχουμε βρει ότι τα μόρια διασκορπίζονται στα κελιά του χώρου με τέτοιο τρόπο, έτσι ώστε κάθε κελί που περιέχεται από μόρια αποτελείται είτε από τέσσερα, είτε από οκτώ. Μπορούμε να αυξήσουμε τον αριθμό των υπολογισμών που υλοποιούνται από την εφαρμογή, αυξάνοντας των αριθμών μορίων που χρησιμοποιούνται σαν δεδομένα εισόδου. Επομένως, το νέο input set έχει δημιουργηθεί με τέτοιο τρόπο έτσι ώστε να εισαχθούν επιπλέον μόρια μόνο στα κελιά που δεν είναι κενά, διατηρώντας έτσι τις ιδιότητες που έχουν οι θέσεις των κελιών που βρίσκονται στα μόρια.

Η γραφική παράσταση (Σχήμα 6.7) που ακολουθεί απεικονίζει την επίδοση της εφαρμογής χρησιμοποιώντας τα δεδομένα αυτά, για τις περιπτώσεις της εφαρμογής χωρίς τροποποίηση και της εφαρμογής όπου δεν υπάρχει επικοινωνία μεταξύ των νημάτων επεξεργασίας.



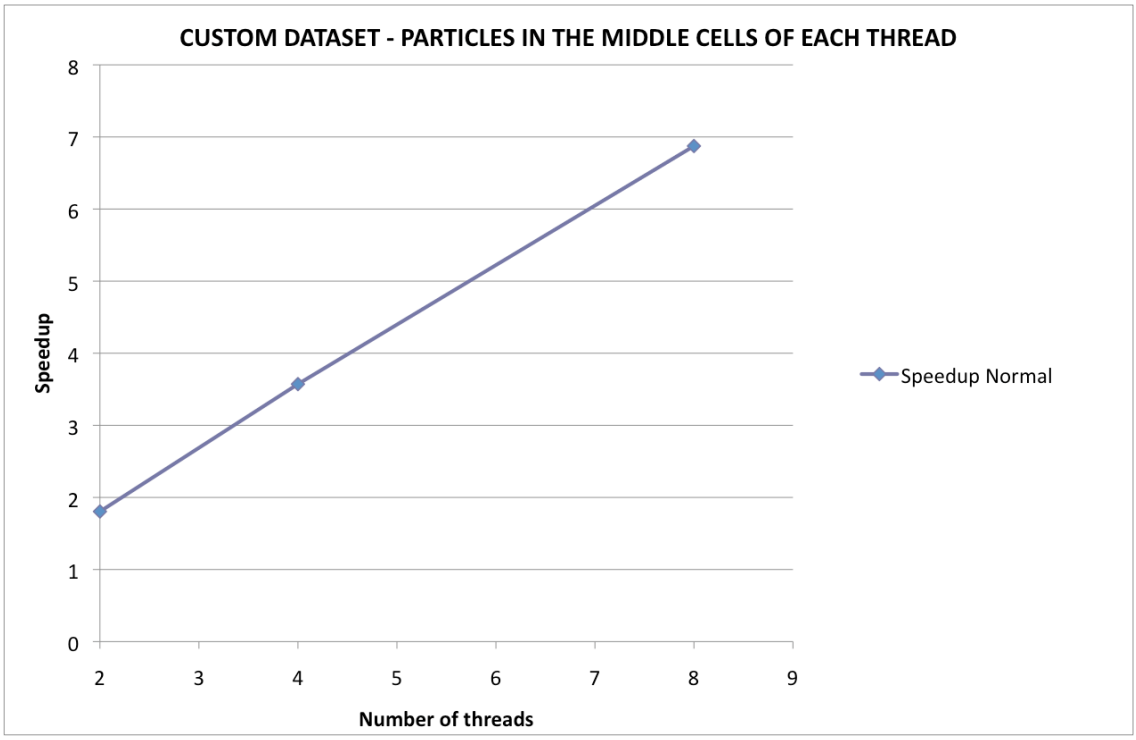
Σχήμα 6.7 Speedup εφαρμογής αυξάνοντας τον αριθμών μορίων στα κελιά που δεν είναι κενά

Από την γραφική παράσταση διακρίνουμε σημαντική αύξηση του speedup στην περίπτωση όπου δεν υπάρχει επικοινωνία μεταξύ νημάτων επεξεργασίας. Βλέπουμε ότι για κάθε αριθμό νημάτων επεξεργασίας, το speedup πλησιάζει το θεωρητικό μέγιστο που μπορεί να επιτευχθεί. Παρόλα αυτά, το speedup του αρχικού προγράμματος δεν βελτιώνεται, αφού σε αυτό υλοποιείται επικοινωνία μεταξύ των νημάτων επεξεργασίας.

Τα ποιο πάνω αποτελέσματα είναι μια καλή ένδειξη των παραγόντων που επηρεάζουν την απόδοση της εφαρμογής. Συγκεκριμένα, μας υποδηλώνουν ότι στην περίπτωση όπου υπάρχουν αρκετά δεδομένα εισόδου, όπου οι υπολογισμοί υποσκιάζουν το overhead της μεταφοράς δεδομένων από την κύρια μνήμη στους πυρήνες, και χωρίς καθόλου επικοινωνία μεταξύ των νημάτων επεξεργασίας, τότε το speedup της εφαρμογής μπορεί να φτάσει στα επίπεδα του θεωρητικού μεγίστου.

Βασιζόμενοι στα ποιο πάνω αποτελέσματα, μπορούμε να δημιουργήσουμε νέα δεδομένα εισόδου με τα οποία επιτυγχάνονται τα χαρακτηριστικά αυτά. Γνωρίζοντας

τις συντεταγμένες που διαχωρίζεται ο χώρος μεταξύ των νημάτων επεξεργασίας, δημιουργούμε νέα δεδομένα έτσι ώστε τα μόρια να βρίσκονται μαζεμένα στο κέντρο των κουτιών που ανατίθενται στα νήματα εκτέλεσης. Ο αριθμός των μορίων που χρησιμοποιούνται είναι ο ίδιος με αυτόν του προηγούμενου παραδείγματος έτσι ώστε να υπάρχουν αρκετοί υπολογισμοί μέσα στο σύστημα. Ο διαχωρισμός έχει γίνει λαμβάνοντας υπόψη την διαμοίραση του χώρα στην περίπτωση όπου η εφαρμογή εκτελείται με την χρήση οκτώ νημάτων εκτέλεσης. Η γραφική παράσταση στο Σχήμα 6.8 παρουσιάζει τα αποτελέσματα του speedup όταν η εφαρμογή, χωρίς τροποποιήσεις, εκτελείται με τα νέα δεδομένα εισόδου.

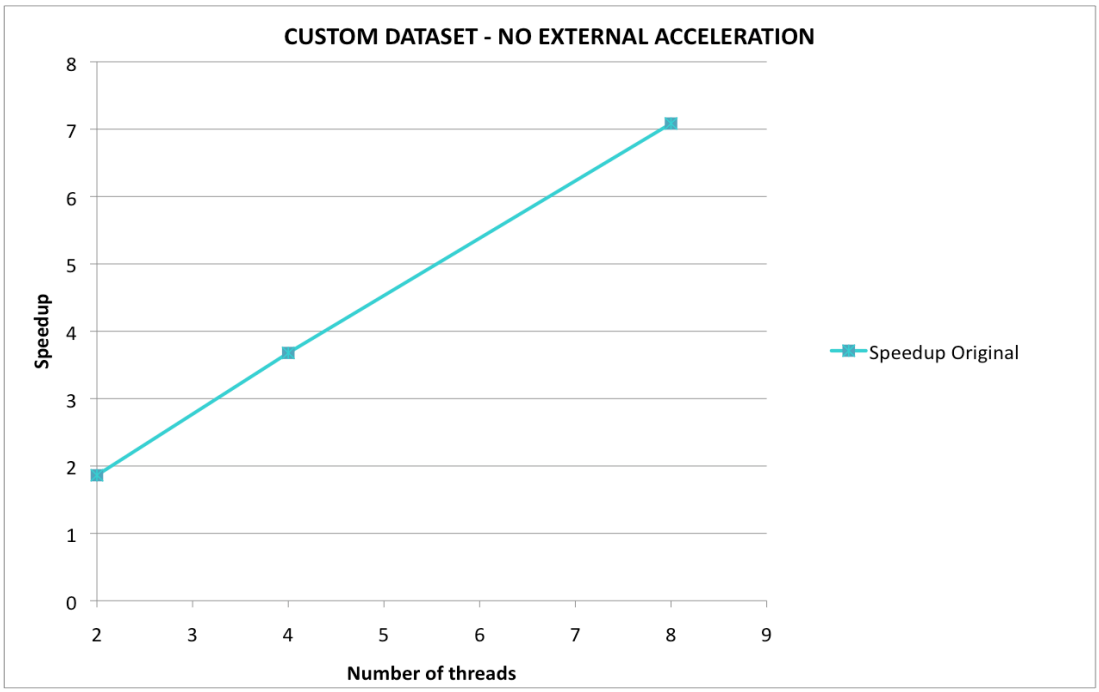


Σχήμα 6.7 Speedup εφαρμογής με τα μόρια να βρίσκονται στο κέντρο κάθε νήματος εκτέλεσης

Βλέπουμε ότι η νέα κατανομή των μορίων μέσα στον χώρο αλλάζει την συμπεριφορά της προσημείωσης. Η τοποθέτηση των μορίων μέσα στο κέντρο του κουτιού που ανατίθεται σε κάθε νήμα εκτέλεσης, μειώνει την επικοινωνία μεταξύ τους, και πλέον κάθε νήμα εκτελεί υπολογισμούς μόνο στα δεδομένα που του ανήκουν.

Το γεγονός ότι η απόδοση της εφαρμογής είναι λίγο πιο κάτω από 7, είναι αποτέλεσμα της κίνησης των μορίων μέσα στο χώρο. Σε κάποια στιγμή της προσομοίωσης, λόγω των δυνάμεων που ασκούνται μεταξύ των μορίων, αυτά θα αρχίσουν να αποκτούν επιτάχυνση με αποτέλεσμα να κινούνται όλο και πιο κοντά στα κελιά που βρίσκονται στα όρια μεταξύ των νημάτων εκτέλεσης. Αυτό θα προκαλέσει επικοινωνία μεταξύ των πυρήνων που είναι υπεύθυνοι για την εκτέλεση των εν λόγω νημάτων, με αποτέλεσμα να μειώνεται η απόδοση της εφαρμογής.

Μπορούμε όμως να μειώσουμε την επιτάχυνση των μορίων μέσα στην προσημείωση. Εκτός από τις δυνάμεις που ασκούνται μεταξύ μορίων, η εφαρμογή λαμβάνει υπόψη και τους εξωτερικούς παράγοντες της φύσης που επηρεάζουν την επιτάχυνση των μορίων σε κάθε στάδιο της προσημείωσης. Μηδενίζοντας τους παράγοντες αυτούς, τα μόρια πλέον κινούνται μόνο με τις δυνάμεις που ασκούνται μεταξύ τους. Με αυτό τον τρόπο επιβραδύνεται ο ρυθμός μετατόπισης τους, αυξάνοντας έτσι τον χρόνο που θα χρειαστεί μέχρι να υλοποιηθεί επικοινωνία μεταξύ των νημάτων εκτέλεσης. Ακολουθούν πιο κάτω τα αποτελέσματα της εφαρμογής, όπου αγνοείται ο παράγοντας της εξωτερικής επιτάχυνσης.



Σχήμα 6.7 Speedup εφαρμογής με τα μόρια να βρίσκονται στο κέντρο κάθε νήματος εκτέλεσης χωρίς εξωτερική επιτάχυνση

Από την πιο πάνω γραφική παράσταση διακρίνουμε ότι έχουμε ελάχιστη βελτίωση μειώνοντας την επιτάχυνση των μορίων μέσα στο σύστημα. Το γεγονός αυτό, οφείλεται στον περιορισμό της εφαρμογής να δέχεται το πολύ 16 μόρια σε κάθε κελί. Ο αριθμός των μορίων είναι αρκετά μεγάλος και παρά το γεγονός ότι αυτά δεν βρίσκονται σε κελιά των ορίων όπου υλοποιείται επικοινωνία, εντούτοις η απόσταση τους από τα κελιά αυτά είναι πολύ μικρή, με αποτέλεσμα η βελτιστοποίηση αυτή να μην έχει μεγάλη συνεισφορά στην απόδοση της εφαρμογής.

Το πιο πάνω πείραμα παρουσιάζει δύο πολύ συχνούς παράγοντες που επηρεάζουν την απόδοση παράλληλων προγραμμάτων. Έχουμε δει ότι όταν οι υπολογισμοί που υλοποιούνται σε σχέση με τον ρυθμό μεταφοράς δεδομένων από την μνήμη στις κρυφές μνήμες των επεξεργαστών, τότε το όφελος του παραλληλισμού δεν είναι μεγάλο. Αυξάνοντας τον αριθμό των δεδομένων, αυξάνονται οι υπολογισμοί που υλοποιούνται, με αποτέλεσμα να επισκιάζεται το overhead της επικοινωνίας με την κύρια μνήμη.

Επίσης, η επικοινωνία μεταξύ νημάτων εκτέλεσης αυξάνει σημαντικά τον χρόνο εκτέλεσης της παράλληλης εφαρμογής. Επακόλουθο του γεγονότος αυτού είναι η μείωση της απόδοσης που επιτυγχάνεται από την εφαρμογή. Στις περιπτώσεις αυτές, βλέπουμε το speedup της εφαρμογής να μειώνεται με την αύξηση των νημάτων επεξεργασίας που την εκτελούν, αφού αυξάνεται και η πιθανότητα επικοινωνίας μεταξύ τους. Η επικοινωνία αυτή υλοποιείται μέσω του διαύλου επικοινωνίας (FSB), το οποίο εξυπηρετεί τις αιτήσεις σειριακά. Επομένως, όταν ο ρυθμός των αιτήσεων για επικοινωνία αυξάνεται, τότε μειώνεται και το speedup των εφαρμογών. Με την μείωση του ρυθμού επικοινωνίας, βλέπουμε σημαντική αύξηση του speedup που επιτυγχάνεται από την εφαρμογή.

6.3 Blackscholes

Το πρόγραμμα υλοποιεί την μερική διαφορική εξίσωση Black-Scholes η οποία χρησιμοποιείται για τον υπολογισμό της τιμής Ευρωπαϊκών μετοχών. Έχει συμπεριληφθεί στην ομάδα προγραμμάτων Parsec έτσι ώστε να εκπροσωπήσει των

τομέα εφαρμογών που χρησιμοποιούν μερικές διαφορικές εξισώσεις (PDE) για την επίλυση προβλημάτων χρηματοδότησης.

6.3.1 Περιγραφή Αλγορίθμου

Η τιμή της κάθε μετοχής επιτυγχάνεται με την χρήση της πιο κάτω διαφορικής εξίσωσης.

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

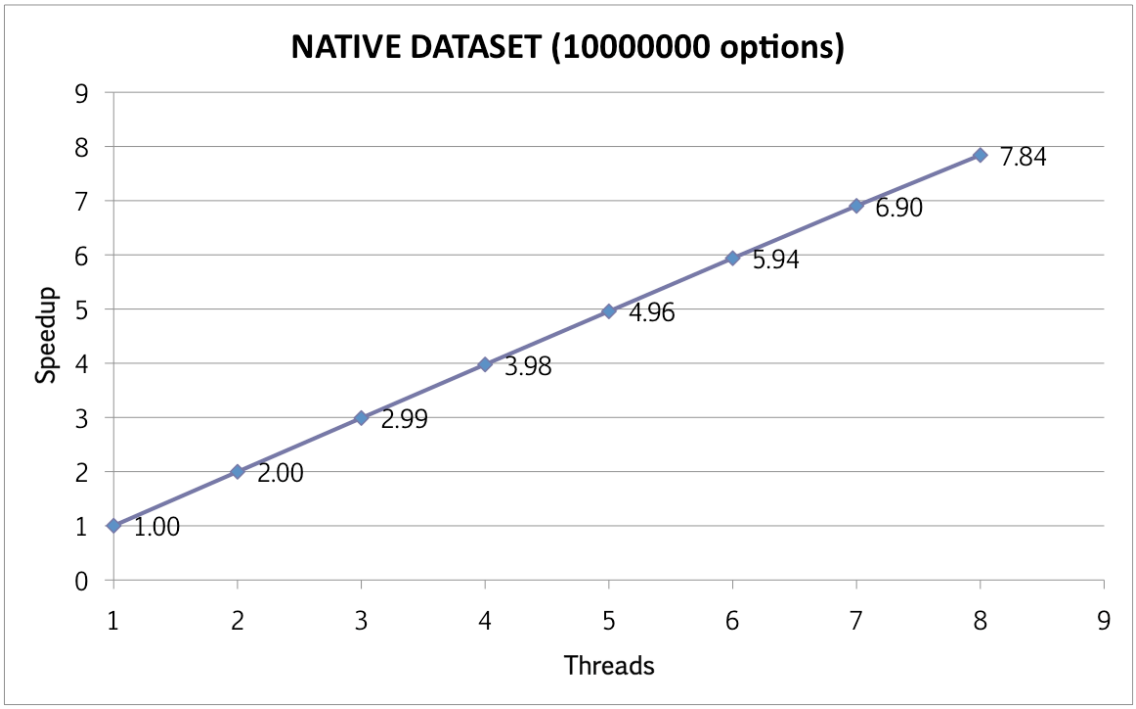
Η παράμετρος S είναι η τιμή της μετοχής, V η παράγωγος της τιμής σε σχέση με τον χρόνο t, και r το ετήσιο χωρίς ρίσκο επιτόκιο. Τα δεδομένα της εφαρμογής παρέχουν για κάθε μετοχή τις απαιτούμενες παραμέτρους. Στην συνέχεια, χρησιμοποιείται data-parallel μοντέλο παραλληλισμού, όπου οι μετοχές που τίθενται προς επεξεργασία διαμοιράζονται ομοιόμορφα στον αριθμό των νημάτων εκτέλεσης που χρησιμοποιεί η εφαρμογή, όπου και υπολογίζεται η τιμή της κάθε μετοχής με την πιο πάνω εξίσωση.

Η εξίσωση αυτή υλοποιείται από την συνάρτηση BlkSchlsEqEuroNoDiv, η οποία χρησιμοποιεί την βοηθητική συνάρτηση CNDF. Οι συναρτήσεις αυτές εκτελούν υπολογισμούς μόνο για μια μετοχή.

Τα δεδομένα του προγράμματος που παρέχονται αποτελούνται από τα χαρακτηριστικά 10000000 μετοχών. Ο υπολογισμός της τιμής κάθε μετοχής υπολογίζεται 100 φορές και λαμβάνονται μετρήσεις για τον συνολικό χρόνο εκτέλεσης του προγράμματος.

6.3.2 Αποτελέσματα πειραμάτων

Στο σχήμα 6.8 φαίνονται τα αποτελέσματα του speedup όταν η εφαρμογή τρέξει με αριθμό νημάτων εκτέλεσης από 1 μέχρι 8.



Σχήμα 6.7 Speedup εφαρμογής blackscholes χρησιμοποιώντας δεδομένα εισόδου για 10000000 μετοχές

Όπως φαίνεται από την γραφική παράσταση των μετρήσεων η επίδοση του συστήματος είναι πάρα πολύ ψηλή. Παρατηρούμε το speedup της εφαρμογής να έχει σχεδόν γραμμική τάση, το οποίο αυξάνεται με κάθε επιπρόσθετο νήμα εκτέλεσης που χρησιμοποιείται σε αυτήν, και το μέγιστο speedup να καταφθάνει στο 7.84 με την χρήση 8 threads. Αυτό είναι αναμενόμενο από το παρόν πρόγραμμα, αφού η έλλειψη επικοινωνίας μεταξύ των threads, καθώς επίσης και η έλλειψη συγχρονισμού μεταξύ τους έχει σαν αποτέλεσμα την συμπεριφορά αυτή.

Η εφαρμογή αυτή είναι ένα παράδειγμα αλγορίθμων που μπορούν να εκμεταλλευθούν τον μέγιστο παραλληλισμό που προσφέρουν τα παρόντα συστήματα. Η μη ύπαρξη επικοινωνίας και συγχρονισμού κατά την διάρκεια της εκτέλεσης τους, αφού κάθε νήμα εκτέλεσης επεξεργάζεται μόνο τα δικά του δεδομένα, καθώς επίσης και το γεγονός ότι πολύ μικρό ποσοστό τους είναι σειριακό, κάνει δυνατή την επίτευξη του μέγιστου θεωρητικού speedup που προσφέρει η μηχανή. Σύμφωνα με τον νόμο του Amhdal, το μέγιστο θεωρητικό speedup που μπορεί να προσφέρει η μηχανή που εξετάζεται είναι 8.

6.4 Swaptions

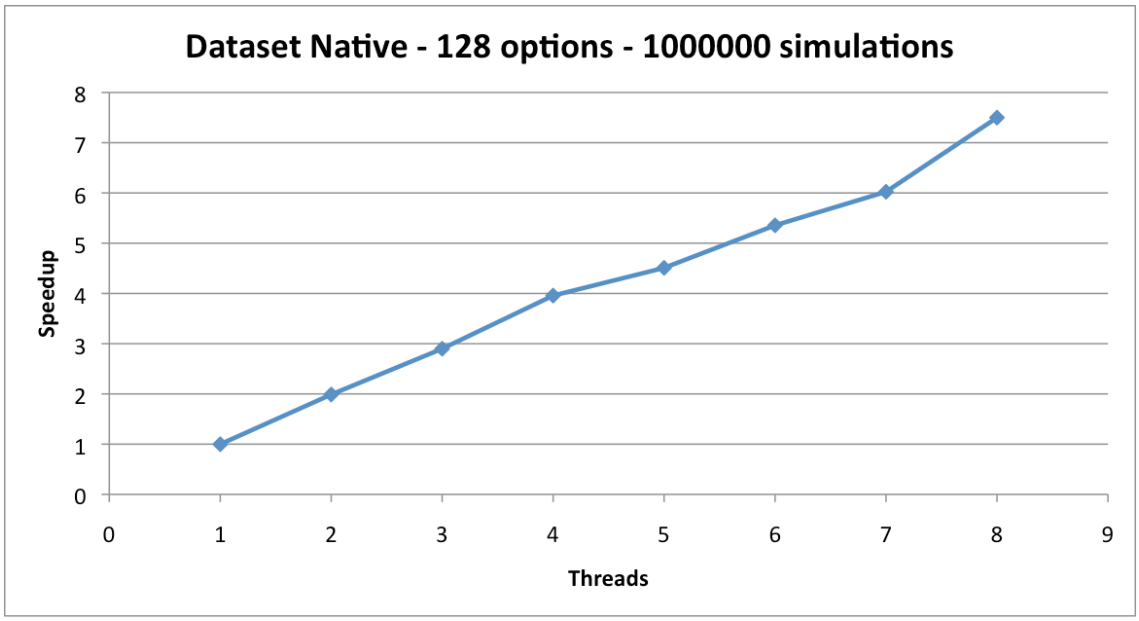
Η εφαρμογή αυτή είναι προϊόν της εταιρίας Intel, η οποία χρησιμοποιεί το πλαίσιο εργασίας Health-Jarrow-Morton έτσι ώστε να υπολογίσει την τιμή ενός συνόλου μετοχών (swaptions). Το πλαίσιο αυτό περιγράφει πώς το επιτόκιο των μετοχών εξελίσσεται, υπολογισμοί οι οποίοι χρησιμοποιούνται για διαχείριση κινδύνων από τις εταιρίες πριν αγοράσουν μετοχές. Τα μοντέλα HJM αυτά δεν μπορούν να επιλυθούν με εξισώσεις, επομένως χρησιμοποιούνται προσομοιώσεις Monte Carlo έτσι ώστε να εξευρεθεί η τιμή της κάθε μετοχής.

6.4.1 Περιγραφή Αλγορίθμου

Το πρόγραμμα κατά την εκτέλεση του δημιουργεί ένα πίνακα από μετοχές, ο οποίος διαμοιράζεται στον αριθμό των νημάτων εκτέλεσης. Κάθε κομμάτι διαμοιράζεται στο ανάλογο νήμα εκτέλεσης. Το κάθε νήμα εκτέλεσης εκτελεί προσπέλαση πάνω στο κομμάτι των μετοχών που του έχουν ανατεθεί, και εκτελεί Monte Carlo προσομοίωση για την κάθε μετοχή.

6.4.2 Αποτελέσματα πειραμάτων

Η γραφική παράσταση που ακολουθεί (Σχήμα 6.8) παρουσιάζει τα αποτελέσματα 128 μετοχών όταν αυτές υποστούν 1000000 προσομοιώσεις MC για τον υπολογισμό της τιμής τους.



Σχήμα 6.8 Speedup εφαρμογής swaptions

Όπως αναμένουμε, το speedup της εφαρμογής είναι πολύ ψηλό. Παρόμοια με την εφαρμογή blackscholes, η μη παρουσία επικοινωνίας και συγχρονισμού μεταξύ των νημάτων εκτέλεσης, καθώς επίσης και ο μικρός αριθμός που απαιτούνται για τα δεδομένα των μετοχών δεν περιορίζουν το speedup της εφαρμογής.

Η μικρή διακύμανση που παρουσιάζεται στις τιμές με διαφορετικό αριθμό νημάτων εκτέλεσης οφείλεται στο γεγονός της χρήσης ψευδοτυχαίων αριθμών στις τεχνικές Monte Carlo. Λόγω της απροσδιόριστης χρονοδρομολόγησης των νημάτων σε κάθε περίπτωση, η συμπεριφορά της γεννήτριας ψευδοτυχαίων αριθμών είναι διαφορετική σε κάθε περίπτωση, επομένως οι ίδιες μετοχές, σε διαφορετικές περιπτώσεις έχουν διαφορετικό φόρτο εργασίας.

6.5 Streamcluster

Η εφαρμογή αυτή καλείται να επιλύσει το πρόβλημα του clustering δεδομένων, τα οποία έρχονται στην εφαρμογή με streaming συμπεριφορά. Έχει συμπεριληφθεί στο Parsec, λόγω της μεγάλης χρήσης αλγορίθμων clustering σε εφαρμογές που ασχολούνται με τους τομείς του intrusion detection, pattern recognition και data mining, τα οποία απαιτούν απόκριση πραγματικού χρόνου.

6.5.1 Περιγραφή Αλγορίθμου

Το πρόβλημα του clustering προσπαθεί να ομαδοποιήσει τα δεδομένα με βάση την ομοιότητα τους. Επομένως όμοια δεδομένα ανήκουν στην ίδια ομάδα (cluster), ενώ ανόμοια δεδομένα βρίσκονται σε διαφορετική ομάδα.

Έστω ότι έχουμε ένα σύνολο από N δεδομένα. Προσπαθούμε να βρούμε ένα σύνολο C με κ clusters (c1, c2 .. Ck), το οποίο διαχωρίζει τα N δεδομένα σε ομάδες N1,N2...Nk, έτσι ώστε κάθε μια από αυτές τις ομάδες Ni, να αποτελείται από τα δεδομένα που είναι πιο κοντά στο ci παρά από οποιοδήποτε άλλο κέντρο cj. Ο στόχος λοιπόν είναι η επιλογή του συνόλου C, ελαχιστοποιώντας το άθροισμα της τετραγωνικής απόστασης:

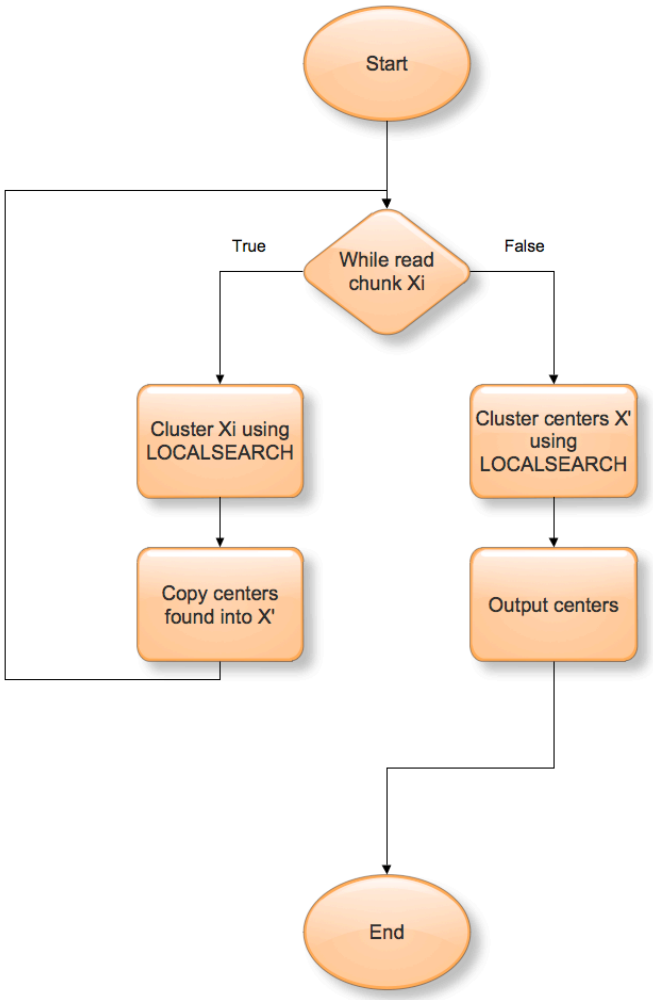
$$SSQ(N,C) = \sum_{i=1}^{k} \sum_{x \in N_i} d(x, c_i).$$

Στην εφαρμογή, αντί να περιορίζεται ο αριθμός των cluster να είναι κ, χρησιμοποιείται μια άλλη προσέγγιση για την επίλυση του προβλήματος. Κάθε κέντρο (center ή facility) χαρακτηρίζεται από ένα κόστος (facility cost). Το κόστος αυτό περιορίζει την λύση από το να αποτελείται από πολλά clusters, αλλά δεν θέτει περιορισμό στο πόσα από αυτά θα δημιουργηθούν. Επομένως, το πρόβλημα πλέον προσπαθεί να ελαχιστοποιήσει την εξίσωση του facility cost:

$$FC(N,C) = z|C| + \sum_{i=1}^{k} \sum_{x \in N_i} \Delta(x, c_i).$$

Το κόστος ενός facility ci, αποτελείται από το άθροισμα της τετραγωνικής απόστασης όλων των σημείων που έχουν ανατεθεί σε αυτό.

Λόγω της ανάγκης του αλγορίθμου να εφαρμόζεται σε stream από δεδομένα, η λειτουργία αυτή επιτυγχάνεται με την διάσπαση των δεδομένων σε κομμάτια (chunks), για τα οποία θα τεθεί να επεξεργαστεί. Στην συνέχεια, αφού εξευρεθούν τα facilities όλων των κομματιών, τότε ο αλγόριθμος επαναλαμβάνεται με δεδομένα εισόδου αυτά τα facilities έτσι ώστε το clustering να υλοποιηθεί σε όλο το data set. Το σχήμα 6.9 δείχνει διαγραμματικά την ροή του αλγορίθμου.



Σχήμα 6.9 Διάγραμμα ροής της εφαρμογής stream cluster

Όπως φαίνεται, οι υπολογισμοί της εφαρμογής υλοποιούνται από την συνάρτηση LOCALSEARCH, όπως αυτή περιγράφεται στο [7]. Ο αλγόριθμος της συνάρτησης, μαζί με όλες τις βοηθητικές συναρτήσεις που τον αποτελούν, επεξηγείται πιο κάτω.

Η συνάρτηση LOCALSEARCH, βασίζεται στις λειτουργίες της συνάρτησης pkmedian. Αφού αρχικοποιήσει τις παραμέτρους της εφαρμογής, καλεί την συνάρτηση αυτή έτσι ώστε να γίνει το clustering των δεδομένων.

Πρώτο στάδιο του αλγορίθμου είναι να υπολογίσει τη μέγιστη τιμή κόστους που μπορεί να έχει μια λύση από το παρόν chunk. Αυτό επιτυγχάνεται αθροίζοντας την απόσταση όλων των στοιχείων του chunk από το πρώτο. Η λειτουργία αυτή

παραλληλοποιείται διαμοιράζοντας τα δεδομένα αυτά στα νήματα εκτέλεσης, όπου και υπολογίζεται η απόσταση του κάθε σημείου με το πρώτο (κόστος). Παράλληλα, το κάθε νήμα υπολογίζει το τοπικό άθροισμα του κόστους από όλα τα σημεία που του έχουν ανατεθεί. Στην συνέχεια, το νήμα εκτέλεσης με ταυτότητα 0, αθροίζει τα κόστοι των νημάτων έτσι ώστε να βρει το μέγιστο κόστος

Η τιμή αυτή θα χρησιμοποιηθεί σαν το μέγιστο δυνατό κόστος που μπορεί να έχει μια λύση (zmax), ενώ η ελάχιστη τιμή κόστους αρχικοποιείται με 0 (zmin). Με βάση αυτά τα όρια υπολογίζεται το κόστος z με τον τύπο (zmax+zmin)/2, τιμή η οποία θα χρησιμοποιηθεί στον αλγόριθμο για την εξεύρεση μιας αρχικής λύσης του προβλήματος. Για την εξεύρεση της αρχικής λύσης στο πρόβλημα, όλα τα σημεία του chunk ανακατεύονται από το νήμα εκτέλεσης με ταυτότητα 0, και στην συνέχεια καλείται η συνάρτηση pspeedy.

Η συνάρτηση αυτή διαμοιράζει τα στοιχεία του chunk στα νήματα εκτέλεσης. Στην συνέχεια, θεωρεί το πρώτο στοιχείο του ανακατεμένου chunk σαν κέντρο, και επομένως τα νήματα εκτέλεσης υπολογίζουν το κόστος κάθε σημείου τους με βάση την απόσταση τους από το πρώτο, και τα αναθέτουν το κάθε σημείο στο κέντρο αυτό.

Στην συνέχεια, το νήμα εκτέλεσης με ταυτότητα 0, αποφασίζει ποια από τα σημεία του chunk θα αποτελούν τα κέντρα της αρχικής λύσης. Αυτό υπολογίζεται συγκρίνοντας το πηλίκο κόστος/z με μια τυχαία τιμή. (Όπου κόστος είναι η απόσταση του σημείου από το κέντρο στο οποίο έχει ανατεθεί). Αν το πηλίκο είναι μικρότερο της τυχαίας τιμής, τότε το σημείο αυτό γίνεται κέντρο. Για κάθε νέο κέντρο που δημιουργείται, τα νήματα εκτέλεσης υπολογίζουν το κόστος των σημείων που τους έχουν ανατεθεί με το νέο κέντρο, και αν το κόστος του νέου κέντρου είναι μικρότερο από το υπάρχον κέντρο που βρίσκονται τότε μεταφέρονται σε αυτό.

Αφού τελειώσει η ποιο πάνω λειτουργία, υπολογίζεται το συνολικό κόστος της παρούσας λύσης. Δηλαδή, κάθε νήμα εκτέλεσης υπολογίζει το άθροισμα του κόστους των σημείων του, και στην συνέχεια το νήμα εκτέλεσης με ταυτότητα 0 υπολογίζει το συνολικό κόστος. Η διαδικασία αυτή αποτελεί και το τέλος της εύρεσης μιας αρχικής λύσης, και κατ’ επέκταση επισημάνει το τέλος της συνάρτησης pspeedy.

Με την εύρεση της αρχικής λύσης, ελέγχεται κατά πόσο η λύση αυτή αποτελείται από τουλάχιστον kmin κέντρα, παράμετρος η οποία καθορίζεται από τον χρήστη. Αν δεν πληρεί την ικανοποίηση αυτό, τότε επαναλαμβάνεται η λειτουργία της συνάρτησης pspeedy για SP φορές, δίνοντάς της έτσι την δυνατότητα να δημιουργήσει νέα λύση που πληρεί τον περιορισμό αυτό.

Αν ακόμη η λύση δεν αποτελείται από τουλάχιστον kmin κέντρα, τότε το κόστος z που έχουμε επιλέξει αρχικά είναι πολύ μεγάλο. Επομένως, το κόστος zmax παίρνει την τιμή z, και το νέο κόστος z υπολογίζεται πάλι με τον τύπο (zmin+zmax)/2. Στην συνέχεια, ανακατεύονται τα σημεία από το νήμα εκτέλεσης με ταυτότητα 0, και καλείται η λειτουργία της συνάρτησης pspeedy για ακόμη μία φορά.

Με βάση αυτή την αρχική λύση, η εφαρμογή καλείται να εξεύρει μια καλύτερη λύση, η οποία είναι κοντά στην βέλτιστη. Η λειτουργία αυτή υλοποιείται με την συνάρτηση pFl. Η συγκεκριμένη συνάρτηση, καλείται μέχρι να βρούμε λύση η οποία αποτελείται από κ κέντρα, τέτοια ώστε kmin <= k <= kmax. Μετά από το τέλος μιας κλήσης της συνάρτησης, ελέγχεται κατά πόσο ο αριθμός των κέντρων που έχει η νέα λύση είναι εντός των πιο πάνω ορίων. Αν ο αριθμός αυτός είναι πολύ μικρός (κ < kmin) τότε το κόστος z που έχουμε επιλέξει είναι πολύ μεγάλο, και επομένως πρέπει να μειωθεί. Στην περίπτωση όπου ο αριθμός των κέντρων της νέας λύσης είναι μεγαλύτερος από το kmax, τότε το κόστος μεγαλώνει.

Η λειτουργία της συνάρτησης pFL, είναι η εξής. Αφού ανακατευθούν τα στοιχεία του chunk, επιλέγεται ένα τυχαίο σημείο από αυτά. Στην συνέχεια καλείται η συνάρτηση pgain, η οποία δημιουργεί το σημείο σαν κέντρο, και κάνει όλες τις απαραίτητες μετακινήσεις έτσι ώστε τα σημεία που βρίσκονται κοντινότερα στο νέο κέντρο να ανατεθούν σε αυτό. Η συνάρτηση αυτή, κλίνει τυχόν κέντρα τα οποία δεν χρειάζονται. Επίσης, υπολογίζει και το κέρδος του κόστους με αυτή την αναδιοργάνωση. Αυτή η λειτουργία εκτελείται μέχρι να παραχθεί μια λύση, για την οποία η διαφορά του κόστους που παράγει σε σχέση με την προηγούμενη είναι πολύ μικρή.

Στην συνάρτηση pgain, αποφασίζεται κατά πόσο θα δημιουργηθεί νέο κέντρο, και υλοποιούνται όλες η απαραίτητες πράξεις έτσι ώστε να μεταφερθούν τα σημεία στο νέο κέντρο. Κάθε νήμα εκτέλεσης, ελέγχει τα σημεία που του έχουν ανατεθεί κατά πόσο το κόστος που θα έχουν αν μεταφερθούν από το παρόν τους κέντρο στο νέο σημείο θα μειωθεί. Στην συνέχεια, κάθε νήμα εκτέλεσης υπολογίζει το συνολικό κόστος των σημείων που του έχουν ανατεθεί αν αυτά μεταφερθούν στο νέο κέντρο. Ακολούθως το νήμα εκτέλεσης με ταυτότητα 0 υπολογίζει το συνολικό κόστος. Αν το κόστος αυτό μειώνει το κόστος της παρούσας λύσης, τότε κάθε νήμα εκτέλεσης κλείνει τα κέντρα που πρέπει να κλείσουν μετά την ανάθεση των σημείων στο νέο κέντρο, και μετακινεί τα σημεία στο νέο κέντρο.

6.5.1 Πολυπλοκότητα Αλγορίθμου

Σύμφωνα με την περιγραφή του αλγορίθμου όπως γίνεται πιο πάνω, στο σημείο αυτό φαίνεται η πολυπλοκότητα των συναρτήσεων που αποτελούν την συνάρτηση LOCALSEARCH (Πίνακας 6.5). Στον πίνακα 6.6 παρουσιάζεται η επεξήγηση των συμβόλων που χρησιμοποιούνται.

Όνομα Συνάρτησης	Πολυπλοκότητα
Pspeedy	n + (k+1)*n/t + t
Pgain	2n/t + t +k'*n/t
pFL	n*(pgain)
Localsearch	(N/t+t) + (SP+1)*pspeedy + p*pFL

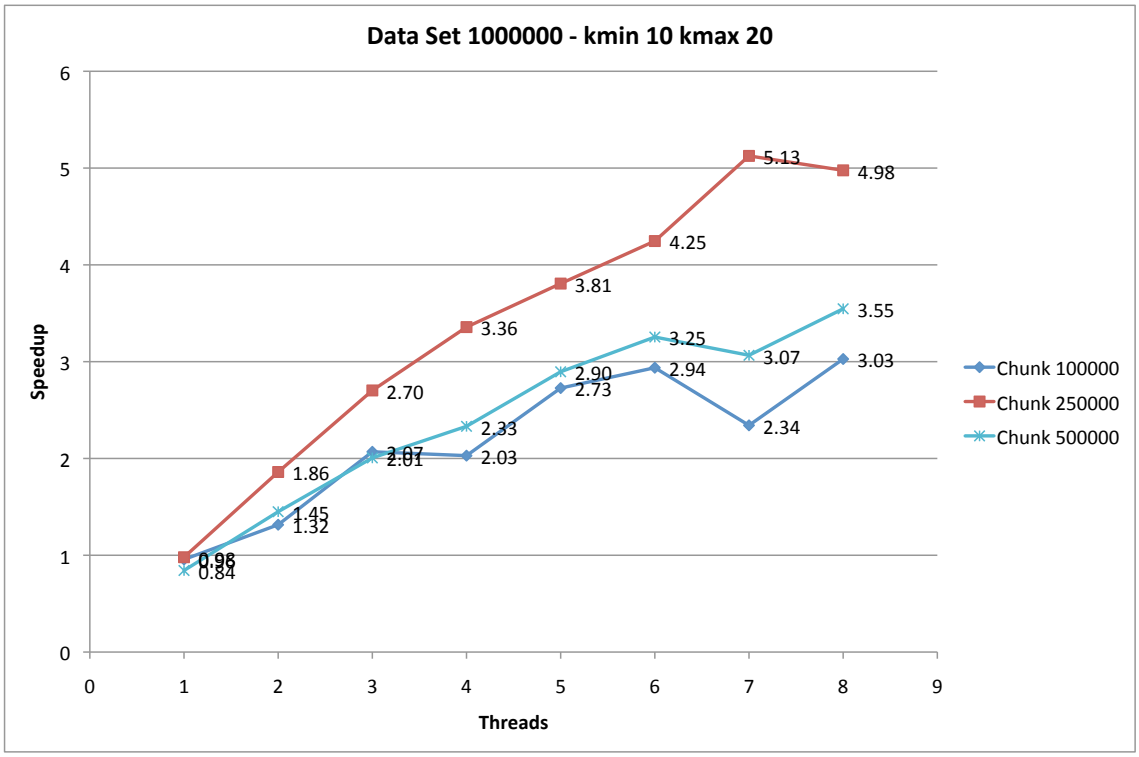
Πίνακας 6.5 Πολυπλοκότητα συναρτήσεων εφαρμογής streamcluster

Σύμβολο	Επεξήγηση
n	Αριθμός σημείων που αποτελείται το chunk
t	Αριθμός νημάτων εκτέλεσης
κ	Αριθμός κέντρων λύσης
κ'	Αριθμός νέων κέντρων που δημιουργούνται από την pspeedy
SP	Αριθμός φορών που εκτελείται η συνάρτηση pspeedy
p	Αριθμός φορών που θα εκτελεστεί η pFL μέχρι να εξευρεθεί λύση εντός των ορίων kmin, kmax

Πίνακας 6.6 Επεξήγηση συμβόλων πολυπλοκότητας συναρτήσεων

6.5.2 Αποτελέσματα πειραμάτων

Η γραφική παράσταση που ακολουθεί παρουσιάζει τα αποτελέσματα των πειραμάτων όπου η λύση πρέπει να υπακούει στους περιορισμούς όπου ο αριθμός των κεντρών πρέπει να βρίσκεται ανάμεσα στα όρια 10 και 20.



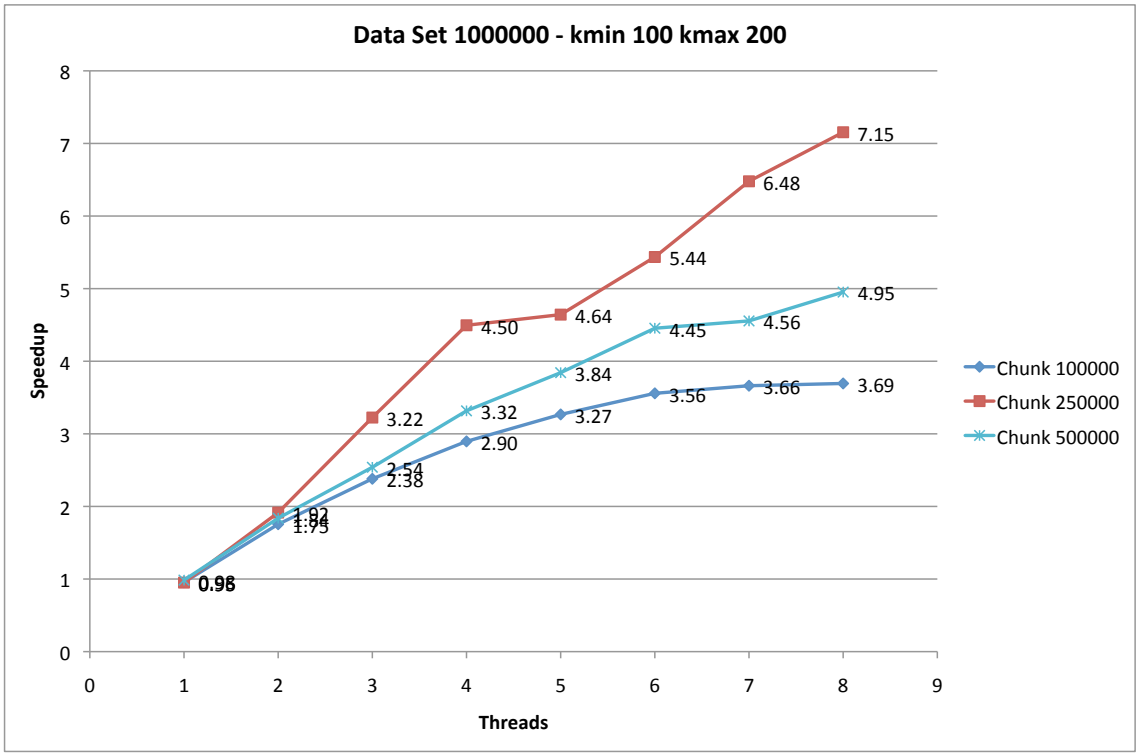
Σχήμα 6.9 Αποτελέσματα πειραμάτων εφαρμογής streamcluster με περιορισμό κέντρων μεταξύ 10 και 20

Στην περίπτωση όπου η ομαδοποίηση εφαρμόζεται ανά 100000 σημεία, τότε βλέπουμε το speedup που επιτυγχάνεται να είναι πολύ μικρό. Αυτό οφείλεται στο γεγονός ότι τα δεδομένα είναι μικρού μεγέθους, επομένως ο φόρτος εργασίας είναι επίσης μικρός, με αποτέλεσμα τα overheads της εφαρμογής να περιορίζουν την επίδοση της. Με την ομαδοποίηση να εφαρμόζεται σε 250000 δεδομένα διακρίνουμε ότι το speedup ανεβαίνει, αφού αυξάνονται και οι υπολογισμοί που γίνονται στην εφαρμογή. Παρόλα αυτά, η ομαδοποίηση σε δεδομένα με αριθμό 500000 είναι η χαμηλότερη από όλες, με το σειριακό κομμάτι της εφαρμογής να γίνεται ο κυρίαρχος παράγοντας του χρόνου εκτέλεσης.

Το γεγονός αυτό επηρεάζει και το ανακάτεμα των δεδομένων από το νήμα εκτέλεσης σε διάφορα στάδια του αλγόριθμου, όπου με αυτό τον τρόπο δημιουργείται έμμεσα και επικοινωνία μεταξύ των νημάτων εκτέλεσης. Σε κάθε κλήση της συνάρτησης pshuffle, όπου υλοποιείται σειριακά το ανακάτεμα των σημείων, σύμφωνα με το πρωτόκολλο συνοχής τα δεδομένα που βρίσκονται στις κρυφές μνήμες των διαφόρων πυρήνων πρέπει να γίνουν άκυρα (invalid), έτσι ώστε να τροποποιηθούν από το νήμα εκτέλεσης με ταυτότητα 0. Στην συνέχεια, ακολουθώντας παράλληλο κομμάτι στην εφαρμογή, τα νήματα εκτέλεσης ζητούν τα δεδομένα αυτά από το νήμα εκτέλεσης με ταυτότητα 0. Όπως έχουμε δεί από το πείραμα στο κεφάλαιο 5, το overhead του πρωτοκόλλου συνοχής είναι πολύ μεγάλο με αποτέλεσμα πλέον να επηρεάζει την επίδοση της εφαρμογής όταν ο αριθμός των δεδομένων είναι πολύ μεγάλος. Επίσης, η σειριακή εξυπηρέτηση των αιτήσεων πάνω στον δίαυλο επικοινωνίας, όπου μετά από το ανακάτεμα όλα τα νήματα εκτέλεσης ζητούν ταυτόχρονα δεδομένα από το νήμα εκτέλεσης με ταυτότητα 0, περιορίζει ακόμη περισσότερο το speedup της εφαρμογής.

Παρόλα αυτά, ακόμη και με 250000 σημεία, το speedup της εφαρμογής είναι περιορισμένο, κάτι που υποδηλώνει ότι υπάρχει και άλλος παράγοντας που περιορίζει την επίδοση των εφαρμογών. Συγκεκριμένα, ο περιορισμός των 10 με 20 κέντρων της λύσης είναι πολύ χαλαρός, αφού μια τέτοια λύση μπορεί να εξευρεθεί σε πολύ μικρό χρονικό διάστημα. Σαν αποτέλεσμα, οι υπολογισμοί που πρέπει να γίνουν είναι λίγοι, με το overhead της εφαρμογής να γίνεται ο κυρίαρχος παράγοντας του χρόνου.

Η συμπεριφορά αυτή μπορεί να αλλαχθεί βάζοντας νέους περιορισμούς, όπου η εφαρμογή καλείται να εξεύρει λύση με κέντρα μεταξύ 100 και 200. Η γραφική παράσταση (Σχήμα 6.10) δείχνει τα αποτελέσματα της εφαρμογής όταν αυτή καλείται με τους πιο πάνω περιορισμούς.



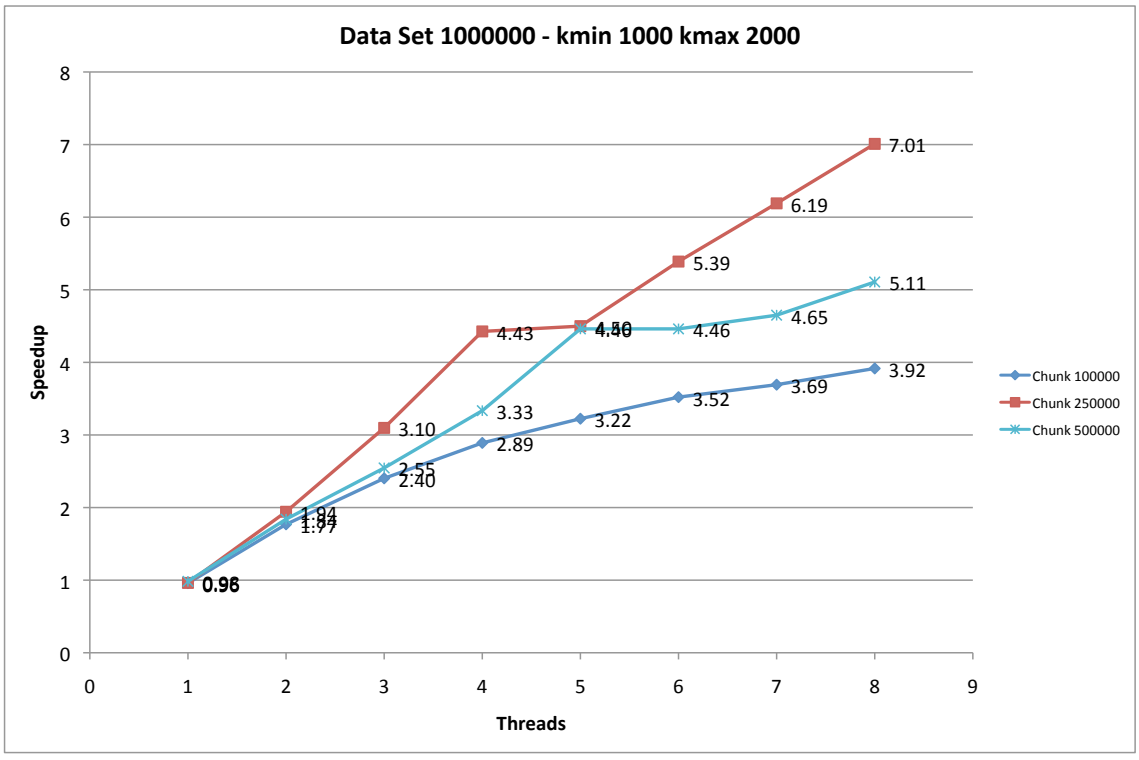
Σχήμα 6.10 Αποτελέσματα πειραμάτων εφαρμογής streamcluster με περιορισμό κέντρων μεταξύ 100 και 200

Παρατηρούμε από την γραφική παράσταση ότι αυξάνεται η επίδοση της εφαρμογής με τους πιο πάνω περιορισμούς, αφού δημιουργείται περισσότερος φόρτος εργασίας έτσι ώστε να φανεί η συνεισφορά του παραλληλισμού σε αυτήν. Όπως και στο προηγούμενο παράδειγμα, η ομαδοποίηση των δεδομένων στα 250000 σημεία κάθε φορά προσφέρει το μεγαλύτερο speedup, όπου με 8 νήματα εκτέλεσης έχουμε speedup 7. Σημαντική παρατήρηση είναι στις περιπτώσεις με 3 και 4 νήματα εκτέλεσης, όπου το speedup που επιτυγχάνεται είναι μεγαλύτερο από το θεωρητικό μέγιστο, γεγονός που υποδηλώνει ότι έχουν υλοποιηθεί λιγότεροι υπολογισμοί από αυτούς του σειριακού κώδικα.

Η συμπεριφορά αυτή οφείλεται στην φύση του αλγορίθμου, να τερματίζει όταν δύο διαδοχικές λύσεις δεν έχουν μεγάλη διαφορά, γι' αυτό και τερματίζει. Με την χρήση υπολογισμών σε πραγματικούς αριθμούς, η χρονοδρομολόγηση των νημάτων εκτέλεσης γίνεται με διαφορετικό ρυθμό. Λόγω του περιορισμού που έχουν οι μηχανές στην ακρίβεια αναπαράστασης πραγματικών αριθμών, και το γεγονός ότι οι πράξεις των λύσεων γίνονται με διαφορετική σειρά σε κάθε περίπτωση,

δημιουργούνται συνθήκες όπου ο αλγόριθμος τερματίζει πιο γρήγορα σε σχέση με αυτό του σειριακού κώδικα.

Το πιο πάνω παράδειγμα υποδηλώνει και το μέγιστο speedup που μπορεί να επιτευχθεί από την εφαρμογή, αφού όπως φαίνεται και από το σχήμα 6.11, αυξάνοντας ακόμη περισσότερο τον φόρτο εργασίας με νέους περιορισμούς, τότε το speedup παραμένει στα ίδια επίπεδα.



Σχήμα 6.11 Αποτελέσματα πειραμάτων εφαρμογής streamcluster με περιορισμό κέντρων μεταξύ 100 και 200

6.6 Σύνοψη - Παράγοντες που επηρεάζουν την επίδοση

Στο μέρος αυτός παρουσιάζεται σύνοψη των παραγόντων που επηρεάζουν την επίδοση, και κατ΄επέκταση το speedup που επιτυγχάνεται από τις πιο πάνω εφαρμογές.

Στην εφαρμογή fluidanimate έχουμε δει ότι ο κύριος παράγοντας που επηρεάζει την επίδοση της εφαρμογής είναι αυτός της επικοινωνίας μεταξύ των νημάτων εκτέλεσης.

Αναδιοργανώνοντας τα μόρια έτσι ώστε να βρίσκονται στο κέντρο κάθε περιοχής που ανατίθεται σε κάθε νήμα εκτέλεσης και μειώνοντας τον ρυθμό επιτάχυνσης των μορίων στο χώρο, μειώνεται η επικοινωνία μεταξύ των νημάτων εκτέλεσης. Επίσης, έχουμε δει ότι όταν τα δεδομένα έχουν πολύ μικρό μέγεθος, τότε το overhead της εφαρμογής γίνεται ο επικρατέστερος παράγοντας που επηρεάζει τον χρόνο εκτέλεσης, περιορίζοντας έτσι το speedup της εφαρμογής.

Στις εφαρμογές blackscholes και swaptions, οι υπολογισμοί είναι πλήρες ανεξάρτητοι μεταξύ τους, και επομένως το speedup που επιτυγχάνεται είναι στα επίπεδα του θεωρητικού μεγίστου. Στην εφαρμογή swaptions, βλέπουμε ότι η χρήση γεννήτριας ψευδοτυχαίων αριθμών μπορεί να επηρεάσει τον φόρτο εργασίας για πειράματα με διαφορετικό αριθμό νημάτων εκτέλεσης, με αποτέλεσμα να έχουμε μια μικρή διακύμανση στο speedup της εφαρμογής.

Τέλος, στην εφαρμογή streamcluster βλέπουμε το σειριακό κομμάτι της εφαρμογής να περιορίζει το μέγιστο επίπεδο speedup που μπορεί να επιτευχθεί. Επίσης, η επικοινωνία που δημιουργείται με το ανακάτεμα των δεδομένων λόγω του προτοκόλλου συνοχής μνήμης επηρεάζει σημαντικά την επίδοση της εφαρμογής, όταν τα δεδομένα γίνουν αρκετά μεγάλα. Παρόμοια με την εφαρμογή fluidanimate, τόσο το μέγεθος των δεδομένων όσο και οι περιορισμοί στην λύση της εφαρμογής περιορίζουν το speedup όταν είναι πολύ μικρά, με αποτέλεσμα το overhead της εφαρμογής να καθορίζει τον συνολικό χρόνο εκτέλεσης.

Κεφάλαιο 7

Συμπεράσματα

Η παρούσα εργασία είχε σαν στόχο την μελέτη της οργάνωσης ενός σύγχρονου παράλληλου υπολογιστικού συστήματος, και τις διάφορες τεχνολογίες που το απαρτίζουν. Βλέπουμε ότι μέσα στην πάροδο του χρόνου έχουν γίνει πολλές προσπάθειες στην αρχιτεκτονική συστημάτων έτσι ώστε να επιτυγχάνεται όσο το δυνατό περισσότερος παραλληλισμός στις εφαρμογές, αφού η αρχιτεκτονική του συστήματος επηρεάζει σημαντικά την απόδοση τους.

Η νέες τάσεις της τεχνολογίας, δείχνουν να απομακρύνονται από την χρήση ενός κοινού διαύλου επικοινωνίας μεταξύ των μονάδων του συστήματος. Το γεγονός ότι οι δίαυλοι αυτοί εξυπηρετούν τις αιτήσεις που εκδίδονται σε αυτούς σειριακά, προκαλεί συμφόρηση στο σύστημα με αποτέλεσμα να μειώνεται σημαντικά η απόδοση των υπολογιστικών μονάδων που το απαρτίζουν. Η αύξηση των μονάδων αυτών σε κάθε νέα γενιά συστημάτων, επιδεινώνουν το πρόβλημα αυτό, αφού αυξάνεται και η ανάγκη επικοινωνίας μεταξύ τους. Επομένως, η μεταφορά σε point-to-point συνδέσεις μεταξύ τους, καθώς επίσης και η ύπαρξη κατανεμημένης κύριας μνήμης τίθενται να δώσουν λύσεις στο πρόβλημα.

Η παρουσία του πρωτοκόλλου συνοχής, παρά το γεγονός ότι είναι απαραίτητη στα συστήματα πολυεπεξεργαστών/πολυπυρήνων, εντούτοις μειώνει σημαντικά τον χρόνο εκτέλεσης των εφαρμογών, και επομένως περιορίζει το επίπεδο παραλληλισμού που μπορεί να επιτευχθεί. Με το πείραμα του παραγωγού - καταναλωτή, έχουμε δει ότι η μη ύπαρξη μηνυμάτων συνοχής μειώνει σημαντικά τον χρόνο εκτέλεσης της εφαρμογής. Για τον λόγο αυτό, οι εταιρίες προσπαθούν μέσα από διάφορες αρχιτεκτονικές δομές (snoop filter), καθώς επίσης και επεκτάσεις του πρωτοκόλλου που χρησιμοποιείται στα συστήματα, να μειώσουν το overhead που προκαλείται από αυτό.

Με βάση τα παράλληλα προγράμματα που έχουν εξεταστεί, παρατηρούμε ότι για να αξιοποιηθεί ο παραλληλισμός που προσφέρει το σύστημα, πρέπει αυτά να αποφεύγουν την επικοινωνία και τον συγχρονισμό αναμεταξύ τους, με το κάθε νήμα εκτέλεσης τα επεξεργάζεται μόνο τα δεδομένα που του έχουν ανατεθεί. Το χαρακτηριστικό αυτό κατέχουν οι εφαρμογές swaptions και blackscholes, όπου επιτυγχάνεται speedup μέχρι και 7.89, επίπεδο που είναι πολύ κοντά στο μέγιστο θεωρητικό που μπορεί να επιτευχθεί.

Σε αντίθεση με αυτές, στην εφαρμογή fluidanimate ο παράγοντας της επικοινωνίας μεταξύ των νημάτων εκτέλεσης έχει σημαντικό αντίκτυπο στο επίπεδο παραλληλισμού που μπορεί να επιτευχθεί. Στην περίπτωση όπου υπάρχουν μόρια σε κελιά όπου υλοποιείται επικοινωνία, τότε το speedup είναι πολύ μειωμένο και φτάνει μέχρι το 5. Παρόλα αυτά, σε συνθήκες όπου γίνεται λιγότερο ποσοστό επικοινωνίας, τότε επιτυγχάνεται speedup μέχρι και 7. Βλέπουμε λοιπόν ότι η φύση της εφαρμογής, δίνει ιδιότητες στα δεδομένα, τα οποία επηρεάζουν την συμπεριφορά της.

Μέσα από την εφαρμογή streamcluster διακρίνουμε τον παράγοντα του σειριακού κώδικα μέσα σε παράλληλες εφαρμογές να περιορίζει την επίδοση της. Στην συγκεκριμένη περίπτωση, χρειάζεται η εξεύρεση ενός καλού αριθμού δεδομένων έτσι ώστε να αποτραπεί από τον σειριακό υπολογισμό να είναι ο κυρίαρχος παράγοντας στον καθορισμό του speedup της εφαρμογής. Επίσης, η μετατροπή από σειριακό σε παράλληλο κώδικα επηρεάζει άμεσα την επίδοση της εφαρμογής, αφού στην περίπτωση όπου γίνεται επεξεργασία στα ίδια δεδομένα κατά της δύο φάσεις τότε τίθεται σε λειτουργία το πρωτόκολλο συνοχής. Σε αυτή την περίπτωση περιμένουμε να δημιουργείται και επικοινωνία μεταξύ των νημάτων εκτέλεσης, με αποτέλεσμα να μειώνεται το speedup.

Βιβλιογραφία

[1] C. Bienia, S. Kumar, J.P. Singh and K. Li, “The PARSEC: Benchmark Suit: Characterization and Architectural Implications”

[2] J.L. Hennesy and D.A. Patterson, “Computer Architecture - A Quantitative Approach”, 2006

[3] Intel, “Intel 6A and IA-32 Architectures Software Developer’s Manual”, 2009

[4] Intel, “An Introduction to the Intel QuickPath Interconnect”, 2009

[5] Intel, Forward State for use in cache coherency in a multiprocessing system, 2004

[6] C. Lin and L. Snyder, “Principles of Parallel Programming”, 2009

[7] L. O’Callaghan, N. Mishra, A. Meyerson, S. Guha, R. Motwani, “Streaming-Data Algorithms For High Quality Clustering”, 2001

[8] S. Vadlamani and S. Jenks, “C2CBench: A Cache-to-Cache Benchmark User’s Manual”, 2007

Παράρτημα Α

Οι εφαρμογές που έχουν χρησιμοποιηθεί σε αυτή την εργασία βασίζονται στην δεύτερη έκδοση του Parsec Benchmark Suit. Τα διάφορα προβλήματα που παρουσιάζονται στις εφαρμογές μπορούν να επιλυθούν με την χρήση επιπλέον κώδικα (patches) τα οποία μπορούν να εξευρεθούν από την wiki ιστοσελίδα του Parsec.

Από τις εφαρμογές που έχουμε χρησιμοποιήσει, το fluidanimate και το streamcluster έχουν βρεθεί να παρουσιάζουν κάποια προβλήματα. Το πρώτο, παρουσιάζει λάθος στο σημείο όπου διαβάζονται τα δεδομένα εισόδου του σειριακού κώδικα, ενώ το δεύτερο παρουσιάζει προβλήματα deadlock κατά την παράλληλη εκτέλεση.

Για το fluidanimate, ο διορθωτικός κώδικας έχει δημιουργηθεί από εμένα και συμπεριλαμβάνεται στην wiki ιστοσελίδα του Parsec, καθώς επίσης ακολουθεί και πιο κάτω. Για το streamcluster, ο διορθωτικός κώδικας έχει παρθεί από την ιστοσελίδα. Οι κώδικες αυτοί μπορούν να εφαρμοστούν πάνω στις διάφορες εφαρμογές, εκτελώντας την εντολή patch < correct_code.patch βρισκόμενοι στον κατάλογο που είναι τα αρχεία της εφαρμογής που θα διορθωθούν.

```
--- parsec-2.0/pkgs/apps/fluidanimate/src/serial.cpp          2008-11-10
21:59:57.000000000 +0200
+++ parsec-2.0-new/pkgs/apps/fluidanimate/src/serial.cpp          2009-03-03
18:44:19.000000000 +0200
@@ -167,7 +167,6 @@
                    file.read((char *)&vx, 4);
                    file.read((char *)&vy, 4);
                    file.read((char *)&vz, 4);
-                   file.read((char *)&vz, 4);
            if(!isLittleEndian()) {
              px  = bswap_float(px);
              py  = bswap_float(py);
```

Παράρτημα Β

Για την μεταγλώττιση και εκτέλεση των εφαρμογών του, το Parsec παρέχει ένα βοηθητικό εργαλείο το οποίο ευκολύνει την λειτουργία των προγραμμάτων. Το πρόγραμμα αυτό είναι γραμμένο στην γλώσσα του κελύφους bash και έχει την ονομασία parsecmgmt.

Οι διάφοροι παράμετροι του script παρουσιάζονται όταν αυτό καλείται μόνο με το όνομα του. Ο πίνακας που ακολουθεί παρουσιάζει συνοπτικά τις πιο σημαντικές από αυτές, οι οποίες χρησιμοποιήθηκαν κατά την εκτέλεση των πειραμάτων.

Παράμετρος	Επεξήγηση	Επιλογές Παραμέτρου
-a	Καθορίζει την ενέργεια που θα υλοποιηθεί	build, run, clean, info
-p	Το όνομα της εφαρμογής στην οποία θα υλοποιηθεί η ενέργεια	parsecmgmt -a info για λίστα με εφαρμογές
-c	Η διαμόρφωση (configuration) της εφαρμογής	gcc-serial, gcc-pthreads
-i	δεδομένα εισόδου που θα χρησιμοποιηθούν	test, simsmall, simmedium, simlarge, native
-n	Αριθμός των νημάτων εκτέλεσης που θα δημιουργηθούν	Κατα την εκτέλεση της σειριακής εφαρμογής, ο αριθμός αυτός πρέπει να είναι 1

Για παράδειγμα, η μεταγλώττιση και η εκτέλεση του σειριακού κώδικα της εφαρμογής fluidanimate, με δεδομένα εισόδου native επιτυγχάνεται με τις ακόλουθες εντολές.

parsecmgmt -a build -p fluidanimate -c gcc-serial
parsecmgmt -a run -p fluidanimate -c gcc-serial -i native -n 1

Στον κατάλογο parsec-2.0/config βρίσκονται τα αρχεία που είναι υπεύθυνα για την ρύθμιση των γενικών παραμέτρων των προγραμμάτων. Εδώ καθορίζονται τα μονοπάτια για τις διάφορες εφαρμογές που χρησιμοποιεί το script. Επιπλέον, εδώ καθορίζονται και τα ονόματα των δεδομένων εισόδου που είναι επιτρεπτά με την παράμετρο -i. Ακολούθως, κάθε εφαρμογή έχει δικό της υποκατάλογο στο μονοπάτι parsec-2.0/pkgs όπου βρίσκονται οι ρυθμίσεις που αφορούν την ίδια την εφαρμογή.

Ο κώδικας των εφαρμογών βρίσκεται στον υποκατάλογο src, με βάση το μονοπάτι της εφαρμογής. Ανάλογα με την εφαρμογή, ο κώδικας μπορεί να βρίσκεται σε διάφορα αρχεία, ή σε ένα. Με την χρήση του script parsecmgmt η επιλογή του ορθού αρχείου για μεταγλώττιση, και του ορθού εκτελέσιμου προγράμματος γίνονται αυτόματα, όπου καθορίζονται και οι ανάλογες παράμετροι του preprocessor.

Υπάρχουν εφαρμογές για τις οποίες παράμετροι του preprocessor δεν μπορούν να αλλάξουν από το script, γιαυτό και χρειάζεται επέμβαση μέσα στον κώδικα. Στα πειράματα που έχουν υλοποιηθεί, έχει τροποποιηθεί ο κώδικας της εφαρμογής streamcluster όπου ενεργοποιούνται οι παράμετροι PROFILE και CACHE_LINE μέσα στον κώδικα. Η πρώτη παρουσιάζει μετρήσεις του χρόνου της κάθε συνάρτησης, ενώ η δεύτερη καθορίζει το μέγεθος του cache line έτσι ώστε να αποφευχθεί το φαινόμενο του false sharing κατά την εκτέλεση της παράλληλης εφαρμογής.