ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Αποδοτική Αξιοποίηση Σύγχρονων Δικτυακών Τεχνολογιών στην Παράλληλη Εκτέλεση Υπολογισμών σε Συστοιχίες Υπολογιστών Υψηλών Επιδόσεων

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Αριστείδης Β. Σωτηρόπουλος

Αθήνα, Φεβρουάριος 2004

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

Αποδοτική Αξιοποίηση Σύγχρονων Δικτυακών Τεχνολογιών στην Παράλληλη Εκτέλεση Υπολογισμών σε Συστοιχίες Υπολογιστών Υψηλών Επιδόσεων

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

Αριστείδη Β. Σωτηρόπουλου

Συμβουλευτική Επιτροπή: Νεκτάριος Γ. Κοζύρης
Γεώργιος Κ. Παπακωνσταντίνου
Παναγιώτης Δ. Τσανάκας

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 9η Φεβρουαρίου 2004.

Ν. Κοζύρης Επ. Καθηγητής Ε.Μ.Π	Γ. Παπακωνσταντίνου Καθηγητής Ε.Μ.Π	Π. Τσανάκας Καθηγητής Ε.Μ.Π
Α. Σταφυλοπάτης Καθηγητής Ε.Μ.Π	Τ. Σελλής Καθηγητής Ε.Μ.Π	Γ. Στασινόπουλος Καθηγητής Ε.Μ.Π
	Ε. Παπαδρακάκης Καθηγητής Ε.Μ.Π	

Αθήνα, Φεβρουάριος 2004.

(Αριστείδης Β. Σωτηρόπουλος)
Διδάκτωρ του Εθνικού Μετσόβιου Πολυτεχνείου

στους γονείς μου

Βασίλη και Άννα

Περίληψη

Το μοντέλο της συστοιχίας υπολογιστών αποτελεί την πλέον διαδεδομένη αρχιτεκτονική συστημάτων παράλληλης επεξεργασίας, η οποία υιοθετείται από την πλειοψηφία των ισχυρότερων υπολογιστικών συστημάτων. Σύμφωνα με αυτό, οι σταθμοί εργασίας, που απαρτίζουν μια συστοιχία υπολογιστών, διασυνδέονται μέσω ενός δικτύου υπολογιστών, δημιουργώντας ένα παράλληλο σύστημα κατανεμημένης μνήμης.

Αρκετές συστοιχίες υπολογιστών χρησιμοποιούν συμβατικά δίκτυα διασύνδεσης με μεγάλη δυνατότητα παροχής δεδομένων (bandwidth) και μικρή καθυστέρηση (latency). Όμως, οι τεχνολογικές αυτές δυνατότητες δεν αξιοποιούνται αποδοτικά από τις εφαρμογές, κυρίως λόγω της καθυστέρησης που εισάγουν τα στρώματα του λογισμικού επικοινωνίας. Η ύπαρξη αυτών οφείλεται στην υφιστάμενη αρχιτεκτονική κάθε κόμβου και του λειτουργικού συστήματος που εκτελείται σε αυτόν. Τα τελευταία χρόνια, έχουν εμφανιστεί νέες δικτυακές τεχνολογίες, οι οποίες διαθέτουν προηγμένα χαρακτηριστικά, η αξιοποίηση των οποίων μπορεί να μειώσει σημαντικά την καθυστέρηση της επικοινωνίας, ενώ σε μερικές περιπτώσεις η καθυστέρηση λόγω λογισμικού εξαλείφεται εντελώς.

Η μεγάλη διάρκεια εκτέλεσης των επιστημονικών υπολογιστικών προγραμμάτων οφείλεται κυρίως στις επαναληπτικές δομές που περιέχουν, οι οποίες εμφανίζονται στον κώδικα με τη μορφή φωλιασμένων βρόχων. Χρησιμοποιώντας το μετασχηματισμό υπερκόμβου, έναν από τους σημαντικότερους μετασχηματισμούς που εφαρμόζεται σε φωλιασμένους βρόχους, δημιουργούμε σύνολα επαναλήψεων (υπερκόμβους) που εκτελούνται ατομικά. Οι υπερκόμβοι αντιστοιχίζονται στους επεξεργαστικούς κόμβους μιας συστοιχίας υπολογιστών, μέσω μιας κατάλληλης μεθόδου απεικόνισης, όπου μπορούν να εκτελεστούν παράλληλα.

Χρονοδρομολόγηση, ως γνωστόν, είναι η απεικόνιση ομάδων (επαναλήψεων) υπολογισμών σε χρονικές στιγμές εκτέλεσης. Στη βιβλιογραφία εμφανίζεται η συμβατική μέθοδος χρονοδρομολόγησης, η οποία χρησιμοποιείται πάνω από κοινά δίκτυα διασύνδεσης εμφανίζοντας χαμηλή απόδοση λόγω των περιορισμών της αρχιτεκτονικής και κυρίως της μη αποδοτικής αξιοποίησης της ΚΜΕ σε συνδυασμό με το δικτυακό υλικό.

Δεν υπάρχει μέθοδος χρονοδρομολόγησης υπερκόμβων που να επιτρέπει τη χρονική επικάλυψη των υπολογισμών με την επικοινωνία. Η παρούσα διατριβή προτείνει μία νέα μέθοδο χρονοδρομολόγησης που αξιοποιεί πολλά από τα προηγμένα χαρακτηριστικά των νέων δικτυακών τεχνολογιών, επιτρέποντας την επικάλυψη χρόνου υπολογισμών με χρόνο επικοινωνίας. Μέσω της προτεινόμενης επικαλυπτόμενης χρονοδρομολόγησης (χρονοδρομολόγηση σωλήνωσης -

pipelined schedule), η εκτέλεση παραλληλοποιημένων φωλιασμένων βρόχων μπορεί θεωρητικά να περατωθεί στο ήμισυ του χρόνου της συμβατικής εκτέλεσης. Επίσης, αναπτύχθηκε θεωρητικό μοντέλο του συνολικού χρόνου εκτέλεσης των παράλληλων εφαρμογών, στο οποίο για πρώτη φορά λαμβάνονται υπόψη παράμετροι των νέων δικτυακών τεχνολογιών, όπως η δυνατότητα ταυτόχρονης χρήσης της ΚΜΕ και του δικτυακού προσαρμογέα.

Σε πειραματικό επίπεδο, διενεργήθηκε σύγκριση των δύο μεθόδων χρονοδρομολόγησης (δηλ. της συμβατικής και της προτεινόμενης) σε παραλληλοποιημένα τμήματα κώδικα από επιστημονικά υπολογιστικά προβλήματα. Τα αποτελέσματα αποδεικνύουν ότι η προτεινόμενη μέθοδος μειώνει σημαντικά το συνολικό χρόνο εκτέλεσης των εφαρμογών σε σχέση με το συμβατικό τρόπο δρομολόγησης. Επίσης, οι συνολικοί χρόνοι εκτέλεσης που δίνει το θεωρητικό μοντέλο, προσεγγίζουν σημαντικά τους πραγματικούς χρόνους εκτέλεσης, επιβεβαιώνοντας την ορθότητα του μοντέλου τόσο ως προς τις παραμέτρους που λήφθηκαν υπόψη, όσο και προς την ακρίβεια των τιμών που χρησιμοποιήθηκαν.

Λέξεις Κλειδιά: Συστοιχίες Υπολογιστών, Προηγμένα Δίκτυα Διασύνδεσης, Τέλεια Φωλιασμένοι Βρόχοι, Επικαλυπτόμενη Χρονοδρομολόγηση, Χρονοδρομολόγηση Σωλήνωσης, Μετασχηματισμός Υπερκόμβου, Άμεση Προσπέλαση Μνήμης, Προγραμματιζόμενη Είσοδος/Εξοδος, Scalable Coherent Interface, Δικτυακά Πρωτόκολλα, Στρώματα Λογισμικού, MPI, Επικοινωνία Επιπέδου Χρήστη, Καθυστέρηση Δικτύου, Δυνατότητα Παροχής Δεδομένων, Αρχιτεκτονική Εικονικού Προσαρμογέα.

Abstract

Computer cluster model comprises the most popular parallel processing system architecture, and is adopted by the majority of the most powerful computing systems. According to this model, the workstations, that constitute a computer cluster, are interconnected through a computer network, creating a distributed memory parallel system.

Several computer clusters use conventional interconnection networks with high bandwidth and low latency capabilities. However, these technological capabilities are not efficiently utilized by applications, mainly due to the latency introduced by the communication software layers. Their existence is enforced by computer node architecture and by the operating system that is executed on it. Recently, new networking technologies that incorporate advanced networking features, have been introduced. Their exploitation can significantly reduce communication latency, while in several cases, software latency is eliminated completely.

The large duration of scientific computational programs mainly depends on the included iteration structures, that appear with the form of nested loops. Utilizing the supernode transformation, one of the most important transformations applied to nested loops, we create iteration sets (supernodes) that are atomically executed. Supernodes are mapped to the computational nodes of a computer cluster through an appropriate mapping method, where they can be executed in parallel.

Scheduling is defined as the mapping of (iterations) computation sets to execution time steps. In literature, a conventional scheduling method appears, that is utilized over common interconnection networks, providing low performance due to architectural limitations and non-efficient CPU utilization in combination with the networking hardware.

There is no scheduling method that permits the timing overlapping of computations with communication. In this thesis we propose a new time scheduling method, that exploits many advanced characteristics of the new networking technologies, thus allowing the overlapping of computation time with communication time. Through the proposed overlapping schedule (pipelined schedule), parallelized nested loops execution can be theoretically completed in half the duration of conventional execution. In addition, a new theoretical model of total execution time has been developed, in which new networking technologies' parameters, such as the concurrent utilization of CPU and network adapter, have been considered for the first time.

In experimental level, comparison of the two scheduling methods (the conventional and the proposed one) was carried out on parallelized code segments of scientific computational problems. Results prove that the proposed method, compared to the conventional one, significantly

reduces total execution time of applications. Furthermore, total execution times produced by the theoretical model, approximates real execution times, confirming the validity of our model, in terms of the chosen parameters and the accuracy of their respected values.

Keywords: Computer Clusters, Advanced Interconnection Networks, Perfectly Nested Loops, Overlapping Schedule, Pipelined Schedule, Supernode Transformation, Direct Memory Access, Programmable Input/Output, Scalable Coherent Interface, Network Protocols, Software Layers, MPI, User-Level Communication, Latency, Bandwidth, Virtual Interface Architecture.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Αντί Προλόγου

Η παρούσα διδακτορική διατριβή εκπονήθηκε στον *Τομέα Τεχνολογίας Πληροφορικής & Υπολογιστών* της *Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών* του *Εθνικού Μετσόβιου Πολυτεχνείου*. Περιλαμβάνει την έρευνα και τα αποτελέσματα που προέκυψαν κατά τη διάρκεια των μεταπτυχιακών μου σπουδών στο *Εργαστήριο Υπολογιστικών Συστημάτων* της εν λόγω σχολής.

Το αντικείμενο, με το οποίο ασχολήθηκα κατά τη διάρκεια των σπουδών μου στο ΕΜΠ, είναι η Παράλληλη Επεξεργασία σε περιβάλλοντα συστοιχιών υπολογιστών, που χρησιμοποιούν μοντέρνες δικτυακές τεχνολογίες για τη διασύνδεση των κόμβων τους. Αυτές ενσωματώνουν προηγμένα χαρακτηριστικά επικοινωνίας που μπορούν να υιοθετηθούν από τη θεωρία παραλληλοποίησης αλγορίθμων και να οδηγήσουν σε βελτίωση της απόδοσης των εκτελούμενων εφαρμογών. Δεδομένου ότι η παράλληλη αρχιτεκτονική που ακολουθεί το μοντέλο της συστοιχίας υπολογιστών έχει γίνει το de facto πρότυπο περιβάλλον εκτέλεσης παράλληλων εφαρμογών, η μελέτη της αρχιτεκτονικής και του τρόπου λειτουργίας των δικτυακών τεχνολογιών, οι οποίες αποτελούν ένα ζωτικής σημασίας κομμάτι του μοντέλου συστοιχίας υπολογιστών, είναι πολύ σημαντική.

Μέσα από τις γραμμές αυτές, θα ήθελα να εκφράσω τις ευχαριστίες μου σε ένα πλήθος ανθρώπων, που όχι μόνο με βοήθησαν, αλλά συνέβαλαν ουσιαστικά στην πραγματοποίηση της παρούσας διατριβής. Αρχικά, θα ήθελα να ευχαριστήσω τον Επίκουρο Καθηγητή ΕΜΠ, κ. Νεκτάριο Κοζύρη, επιβλέποντα καθηγητή της παρούσας διατριβής, για την καθοδήγησή του κατά τη διάρκεια της ερευνητικής μου εργασίας. Η συμβολή του ήταν πλέον καθοριστική και χωρίς τη βοήθειά του η παρούσα διατριβή δεν θα είχε ολοκληρωθεί ποτέ. Τον ευχαριστώ, επίσης, για το κουράγιο που μου έδωσε σε στιγμές απογοήτευσης και για τις πολύτιμες συμβουλές του, οι οποίες με έκαναν καλύτερο άνθρωπο. Του είμαι πραγματικά ευγνώμων.

Θα ήθελα να ευχαριστήσω τον καθηγητή ΕΜΠ, κ. Γεώργιο Παπακωνσταντίνου, ο οποίος με βοήθησε σημαντικά κατά τα πρώτα στάδια των μεταπτυχιακών μου σπουδών και συνέβαλε στην επιστημονική κατεύθυνση που ακολούθησα. Επίσης, όντας επικεφαλής του Εργαστηρίου Υπολογιστικών Συστημάτων, μου έδωσε την ευκαιρία να βρίσκομαι σε ένα πλήρως οργανωμένο και επιστημονικό περιβάλλον, η περιρρέουσα γνώση του οποίου έπαιξε σημαντικό στην επιστημονική μου εξέλιξη.

Ευχαριστώ θερμά το δεύτερο καθηγητή του Εργαστηρίου μας, κ. Παναγιώτη Τσανάκα, για τη σημαντική βοήθεια που μου παρείχε και για τη μετάδοση της εμπειρίας του σε κρίσιμα θέματα της ενδο-επιστημονικής και εξω-επιστημονικής μας ζωής.

Ευχαριστώ ιδιαιτέρως τον προπτυχιακό φοιτητή Γεώργιο Τσουκαλά, για τις ατέλειωτες ώρες συζητήσεων επί ερευνητικών και μη θεμάτων, οι οποίες με έκαναν να προχωρήσω τόσο ως άνθρωπος όσο και ως επιστήμων. Αισθάνομαι ιδιαίτερα τυχερός που είχα την ευκαιρία να συνεργαστώ μαζί του κατά τη διάρκεια των σπουδών μου και τον ευχαριστώ πολύ για το χρόνο που μου αφιέρωσε.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους συναδέλφους στο Εργαστήριο Υπολογιστικών Συστημάτων και ιδιαίτερα τους Άγγελο Κοκορόγιαννη, Παναγιώτη Θεοδωρόπουλο, Γιάννη Δροσίτη, Γιώργο Γκούμα, Μαρία Αθανασάκη, Στέργιο Στεργίου, Βαγγέλη Κούκη, Νίκο Δροσινό, Βάλια Αθανασάκη, Αντώνη Χαζάπη, Κορνήλιο Κούρτη και Αντώνη Ζήσιμο για την υπομονή και την αντοχή τους να βρίσκονται κοντά μου όλα αυτά τα χρόνια.

Η εργασία αυτή αφιερώνεται στους γονείς μου, οι οποίοι με στήριξαν υλικά και ηθικά, ώστε να μπορέσω απερίσπαστος να ολοκληρώσω το ερευνητικό μου έργο.

Αθήνα, Φεβρουάριος 2004

Αριστείδης Β. Σωτηρόπουλος

Κεφάλαιο 1

Εισαγωγή

«Όταν ρωτάμε αν οι υπολογιστές μπορούν να σκεφτούν,
είναι σαν να ρωτάμε αν τα υποβρύχια μπορούν να κολυμπήσουν.»
– Edsger Dijkstra

Η περιοχή των Συστημάτων Παράλληλης Επεξεργασίας αποτελεί αντικείμενο μελέτης της παγκόσμιας επιστημονικής κοινότητας, που ασχολείται με τον τομέα των εφαρμογών υψηλών επιδόσεων, από τη δεκαετία του 1970. Η ανάγκη για επίτευξη ολοένα και μικρότερων χρόνων εκτέλεσης, έδινε πάντα ώθηση στον τομέα της παραλληλοποίησης προγραμμάτων σε διαρκώς αυξανόμενο αριθμό επεξεργαστικών στοιχείων.

Αρχικά, η παράλληλη επεξεργασία πραγματοποιούνταν σε μηχανές που ονομάζονταν υπερυπολογιστές (supercomputers). Το πολύ υψηλό τους κόστος, έκανε την απόκτησή τους απαγορευτική, ενώ οι αγοραστές περιορίζονταν μόνο σε μεγάλα κρατικά ερευνητικά ιδρύματα. Τα είκοσι τελευταία χρόνια, ο τομέας υπόκειται σε σημαντικές αλλαγές που έχουν ως αποτέλεσμα τη δημιουργία ενός νέου τοπίου στην παράλληλη επεξεργασία, η οποία πραγματοποιείται πλέον σε ομάδες διασυνδεδεμένων προσωπικών υπολογιστών (PC) ή σταθμών εργασίας (workstations), που ονομάζονται συστοιχίες υπολογιστών (computer clusters).

Όσον αφορά στις εφαρμογές, οι πλέον χρονοβόρες πηγές καθυστέρησης σε ένα μεγάλο πλήθος προγραμμάτων που υπολογίζουν τη λύση πολλών προβλημάτων, είναι οι επαναληπτικές δομές εκτέλεσης που περιέχουν. Οι συγκεκριμένες δομές είναι στη συντριπτική τους πλειοψηφία πολλαπλά φωλιασμένοι βρόχοι (multiple nested loops).

Χρησιμοποιώντας ειδικούς μετασχηματισμούς, έχουμε τη δυνατότητα να χωρίσουμε τους φωλιασμένους βρόχους σε τμήματα που εκτελούνται ατομικά. Τα τμήματα αυτά ανατίθενται στα επεξεργαστικά στοιχεία ενός παράλληλου συστήματος όπου εκτελούνται παράλληλα ακολουθώντας μια κατάλληλη μέθοδο χρονοδρομολόγησης.

Εκτός από το χρόνο εκτέλεσης που απαιτείται για να υπολογίσουν τα επεξεργαστικά στοιχεία ένα μέρος του συνολικού προβλήματος, μεγάλο παράγοντα της συνολικής καθυστέρησης αποτελεί και ο συνολικός χρόνος επικοινωνίας για ανταλλαγή δεδομένων μεταξύ των επεξεργαστικών στοιχείων, ιδιαίτερα σε ένα περιβάλλον «χαλαρά» συνδεδεμένων υπολογιστών (loosely-coupled computers).

Η επιδίωξη μικρών χρόνων εκτέλεσης εφαρμογών που επιλύουν δύσκολα υπολογιστικά προβλήματα, επιβάλει την ενασχόληση με τους εξής χώρους: Πρώτον, το χώρο των τεχνολογικών επιτευγμάτων στον τομέα των δικτυακών αρχιτεκτονικών που επηρεάζουντον τρόπο επικοινωνίας και το χρονικό κόστος που αυτή επιβάλλει. Δεύτερον, το χώρο των αλγόριθμων και ειδικότερα των μεθόδων παραλληλοποίησης αυτών, καθώς και δρομολόγησης και εκτέλεσης των επιμέρους δημιουργούμενων κομματιών του αρχικού προγράμματος σε πολλαπλά επεξεργαστικά στοιχεία, με στόχο την πλήρη αξιοποίηση των μοντέρνων δικτυακών αρχιτεκτονικών, σε περιβάλλοντα συστοιχιών υπολογιστών υψηλών επιδόσεων.

1.1 Αντικείμενο της Διατριβής

Η παρούσα διατριβή κινείται στον ευρύτερο χώρο της παράλληλης επεξεργασίας. Ασχολείται με την αποδοτική δρομολόγηση αλγορίθμων χρησιμοποιώντας τη θεωρία των υπερκόμβων που στοχεύουν στην ταχύτερη εκτέλεση τμημάτων κώδικα που αποτελούνται από φωλιασμένους βρόχους. Η χρησιμοποιούμενη αρχιτεκτονική παράλληλης επεξεργασίας είναι κατανεμημένης μνήμης με «χαλαρά» συνδεδεμένους επεξεργαστικούς κόμβους. Λόγω της σημαντικότητας της επικοινωνίας, προτείνονται αλλαγές στον τρόπο χρονοδρομολόγησης και απεικόνισης των παραπάνω κομματιών των παράλληλων προγραμμάτων στους επεξεργαστές, καθώς και αλλαγές στο συνολικό περιβάλλον λογισμικού των παράλληλων εφαρμογών, ώστε να μειωθεί η καθυστέρηση που επιβάλλεται. Αυτό επιτυγχάνεται με την αξιοποίηση προηγμένων τεχνολογικών επιτευγμάτων που προσφέρονται από τα μοντέρνα δίκτυα υπολογιστών. Αποτέλεσμα της συγκεκριμένης διαδικασίας είναι η συνολική επιτάχυνση των παράλληλων προγραμμάτων.

Αναλυτικότερα, τα τελευταία χρόνια, λόγω της εκρηκτικής αύξησης της ισχύος των επεξεργαστών και του μειωμένου τους κόστους, οι «κοινοί» σταθμοί εργασίας έχουν μετατραπεί σε ισχυρά υπολογιστικά συστήματα. Επιπλέον, η παρουσία νέων δικτύων διασύνδεσης υπολογιστών με ταχύτητες επικοινωνίας που αυξάνονται με μεγάλους ρυθμούς, έχει οδηγήσει στη δημιουργία ενός νέου μοντέλου υπολογιστικού περιβάλλοντος, τη συστοιχία υπολογιστών. Σύμφωνα με αυτό, πολλοί ισχυροί σταθμοί εργασίας διασυνδέονται μεταξύ τους χρησιμοποιώντας μοντέρνα δίκτυα διασύνδεσης. Στα πλαίσια της παρούσας εργασίας, γίνεται εκτενής μελέτη της συστοιχίας υπολογιστών, ενός μοντέλου παράλληλης αρχιτεκτονικής κατανεμημένης μνήμης, που χρησιμοποιείται ευρέως και έχει αντικαταστήσει τα παλαιότερα μοντέλα.

Χρησιμοποιώντας ως βάση τη θεωρία των υπερκόμβων, η οποία βρίσκει εφαρμογή στην περίπτωση της παραλληλοποίησης τμημάτων κώδικα που αποτελούνται από φωλιασμένους βρόχους, προτείνονται αλλαγές στη διαδικασία εκτέλεσης των παράλληλων προγραμμάτων, ούτως ώστε να πραγματοποιείται αποδοτική εκμετάλλευση των προηγμένων χαρακτηριστικών που προσφέρονται από τα μοντέρνα δίκτυα υπολογιστών. Οι συγκεκριμένες αλλαγές αφορούν αφενός στον τρόπο χρονοδρομολόγησης και απεικόνισης των υπερκόμβων στις επιμέρους μονάδες επεξεργασίας μιας συστοιχίας υπολογιστών και αφετέρου, στο προγραμματιστικό περιβάλλον των παράλληλων εφαρμογών.

Η προτεινόμενη διαδικασία εκμεταλλεύεται τη δυνατότητα των σύγχρονων δικτύων υπολογιστών να πραγματοποιούν επικοινωνία, χωρίς να απαιτούν τη συνεχή παρέμβαση της κεντρικής μονάδας επεξεργασίας. Με τον τρόπο αυτό πραγματοποιούνται ταυτόχρονα υπολογισμοί από την πλευρά της κεντρικής μονάδας επεξεργασίας και ανταλλαγές δεδομένων από την πλευρά του συστήματος διασύνδεσης. Αποδεικνύεται θεωρητικά ότι με την αξιοποίηση της παράλληλης λειτουργίας των επιμέρους υποσυστημάτων στο εσωτερικό ενός υπολογιστικού κόμβου και συγκεκριμένα του υλικού του επεξεργαστή και του δικτύου, επιτυγχάνεται αποδοτικότερη εκτέλεση των εφαρμογών.

Επίσης, γίνεται εκτενής ανάλυση της δυνατότητας αξιοποίησης των παραπάνω χαρακτηριστικών μέσω υφιστάμενου λογισμικού παράλληλης επεξεργασίας. Μελετώνται, από πλευράς συστήματος και εφαρμογών, οι υποδομές λογισμικού με τη χρήση των οποίων είναι εφικτή η υλοποίηση των προαναφερθέντων βελτιστοποιήσεων. Επιπλέον, προτείνεται η χρήση νέας στοίβας λεπτών στρωμάτων δικτυακού λογισμικού, που υλοποιούν νέες τεχνικές ενδοκομβικής επικοινωνίας (επικοινωνία σε επίπεδο χρήστη και χωρίς αντιγραφές) μεταξύ των βασικών μονάδων ενός υπολογιστικού κόμβου (επεξεργαστής–μνήμη–δίκτυο), για την πραγματοποίηση συνολικά ταχύτερης επικοινωνίας μεταξύ των κόμβων των συστοιχιών υπολογιστών.

Τέλος, παρατίθενται τα αποτελέσματα πειραματικών μετρήσεων εκτέλεσης βελτιστοποιημένων προγραμμάτων υπολογισμού φωλιασμένων βρόχων σε συστοιχίες υπολογιστών με διαφορετικές τεχνολογίες, τόσο στην υλική τους υποδομή όσο και την υποδομή λογισμικού. Από τα αποτελέσματα αποδεικνύεται ότι, μέσω της προτεινόμενης θεωρίας πραγματοποιείται αποδοτικότερη αξιοποίηση των δικτυακών τεχνολογιών, με επακόλουθο τη συνολική επιτάχυνση εκτέλεσης των παράλληλων προγραμμάτων.

1.2 Οργάνωση της Διατριβής

Η παρούσα διατριβή μελετά την παράλληλη εκτέλεση φωλιασμένων βρόχων σε συστοιχίες υπολογιστών υψηλών επιδόσεων, στοχεύοντας στη συνολική επιτάχυνση της εκτέλεσης προγραμμάτων, μέσω της αποδοτικής εκμετάλλευσης των προηγμένων χαρακτηριστικών που προσφέρουν

τα μοντέρνα δίκτυα υπολογιστών.

Στο παρόν Κεφάλαιο συνοψίζεται το αντικείμενο της διατριβής, ο τρόπος με τον οποίο οργανώνεται, καθώς και οι δημοσιεύσεις που προέκυψαν κατά τη διάρκεια εκπόνησής της.

Στο Κεφάλαιο 2 γίνεται μια εισαγωγή στη θεωρία των φωλιασμένων βρόχων. Παρουσιάζεται το μοντέλο των προβλημάτων που μελετάμε, καθώς και η σημειολογία που ακολουθούμε. Επίσης, πραγματοποιείται παρουσίαση του μετασχηματισμού υπερκόμβου, ενός από τους σημαντικότερους μετασχηματισμούς που εφαρμόζεται σε φωλιασμένους βρόχους.

Στο Κεφάλαιο 3 παρουσιάζεται το μοντέλο της συστοιχίας υπολογιστών, ενός περιβάλλοντος παράλληλης επεξεργασίας το οποίο χρησιμοποιείται εκτενώς τα τελευταία χρόνια. Αναφέρονται οι λόγοι επικράτησης του συγκεκριμένου μοντέλου έναντι των παραδοσιακών συστημάτων παράλληλης επεξεργασίας. Αναλύεται ο τρόπος λειτουργίας των συμβατικών δικτύων διασύνδεσης σε συνδυασμό με τα πρωτόκολλα επικοινωνίας που χρησιμοποιούν. Εντοπίζονται τα σημεία που προκαλούν αύξηση της καθυστέρησης επικοινωνίας, η οποία επηρεάζει την εκτέλεση των παράλληλων εφαρμογών. Παρουσιάζονται νέα αρχιτεκτονικά χαρακτηριστικά επικοινωνίας που παρακάμπτουν το λειτουργικό σύστημα, εξαλείφουν τις μη-αποδοτικές αντιγραφές δεδομένων και παρέχουν τη δυνατότητα για ταυτόχρονη χρήση της ΚΜΕ και του δικτυακού υποσυστήματος ενός υπολογιστικού κόμβου.

Στο Κεφάλαιο 4 αναλύεται η θεωρία που αναφέρεται στο συμβατικό τρόπο χρονοδρομολόγησης υπερκόμβων σε περιβάλλον συστοιχιών υπολογιστών. Στη συνέχεια προτείνεται ένας νέος τρόπος χρονοδρομολόγησης που λαμβάνει υπόψη προηγμένα χαρακτηριστικά των δικτυακών τεχνολογιών. Πραγματοποιείται θεωρητική σύγκριση των δύο μεθόδων χρονοδρομολόγησης για τα προβλήματα που μας ενδιαφέρουν.

Στο Κεφάλαιο 5 γίνεται η πειραματική σύγκριση του προτεινόμενου τρόπου επικαλυπτόμενης χρονοδρομολόγησης με τον αντίστοιχο τρόπο συμβατικής χρονοδρομολόγησης. Αναλύεται ο τρόπος που οι εφαρμογές υλοποιούν τους διαφορετικούς τρόπους δρομολόγησης τόσο πάνω από συμβατικές δικτυακές τεχνολογίες, όσο και από μοντέρνες. Δίνεται ιδιαίτερη έμφαση στην εκμετάλλευση των προηγμένων χαρακτηριστικών των δικτύων προκειμένου να υλοποιηθεί η προτεινόμενη δρομολόγηση. Τέλος, γίνεται αξιολόγηση και σχολιασμός των πειραματικών αποτελεσμάτων.

Τα συμπεράσματα της διατριβής αναλύονται και σχολιάζονται στο Κεφάλαιο 6. Στο ίδιο Κεφάλαιο παρατίθενται αρκετές σκέψεις για τη συνέχιση της παρούσας έρευνας στο χώρο της παράλληλης επεξεργασίας, ιδιαίτερα στο κομμάτι της επικοινωνίας των υπολογιστικών συστημάτων μέσω δικτύων υψηλών επιδόσεων.

1.3 Συμβολή της Διατριβής

Βασικές καινοτομίες της διατριβής:

- Αναδεικνύεται η σημασία της επικοινωνίας στην παράλληλη επεξεργασία, με έμφαση στις ιδιαιτερότητες που χαρακτηρίζουν τα νέα περιβάλλοντα παράλληλων αρχιτεκτονικών που βασίζονται σε κατανεμημένα επεξεργαστικά στοιχεία, όπως είναι οι συστοιχίες υπολογιστών.

- Αναδεικνύεται ο τρόπος λειτουργίας των δικτύων διασύνδεσης υπολογιστικών κόμβων και αναλύονται σε επίπεδο λειτουργιών συστήματος οι κυριότερες διαδικασίες που πραγματοποιούνται για την υλοποίηση της επικοινωνίας. Συγκεκριμένα, παρουσιάζονται οι περιπτώσεις της αποστολής και της λήψης δεδομένων τόσο σε ευρέως χρησιμοποιούμενες, όσο και σε μοντέρνες δικτυακές τεχνολογίες.

- Σε θεωρητικό επίπεδο, προτείνεται ένας νέος τρόπος χρονοδρομολόγησης και απεικόνισης υπερκόμβων στα επεξεργαστικά στοιχεία μιας συστοιχίας υπολογιστών, η επικαλυπτόμενη χρονοδρομολόγηση ή χρονοδρομολόγηση σωλήνωσης (pipelined schedule). Ο τρόπος αυτός λαμβάνει υπόψη τα προηγμένα αρχιτεκτονικά χαρακτηριστικά των σύγχρονων δικτύων διασύνδεσης υπολογιστών, με στόχο την επικάλυψη χρόνου επεξεργασίας με χρόνο επικοινωνίας. Αποτέλεσμα είναι η συνολική επιτάχυνση όλων των εκτελούμενων παράλληλων εφαρμογών σε περιβάλλοντα συστοιχιών με προηγμένα δίκτυα διασύνδεσης.

- Σε επίπεδο υλοποίησης, παρουσιάζεται ένας νέος τρόπος προγραμματισμού παράλληλων εφαρμογών που δίνει τη δυνατότητα να αξιοποιηθεί η παραπάνω δυνατότητα επικάλυψης χρόνων επεξεργασίας και επικοινωνίας. Από τη μελέτη προκύπτει ότι οι συμβατικοί τρόποι προγραμματισμού δεν λαμβάνουν καθόλου υπόψη, ή σε πολύ μικρό βαθμό, τις νέες δυνατότητες των δικτύων υπολογιστών, με αποτέλεσμα την υποχρησιμοποίηση της υφιστάμενης δικτυακής υποδομής.

- Αναλύονται οι βασικές ιδέες που επιτρέπουν τη γενίκευση των παραπάνω συμπερασμάτων στον ευρύτερο τομέα της παράλληλης επεξεργασίας και ιδιαίτερα στο χώρο των επικοινωνιών υπολογιστών υψηλών απαιτήσεων. Τα συμπεράσματα της ειδικής μελέτης των δικτυακών τεχνολογιών, των στρωμάτων δικτυακού λογισμικού, αλλά και της συνολικής αρχιτεκτονικής των συστοιχιών υπολογιστών που προέκυψαν κατά τη διάρκεια της παρούσας ερευνητικής εργασίας, μπορούν να γενικευτούν ώστε να χρησιμοποιηθούν σε περιπτώσεις εφαρμογών που εκτελούνται σε πλατφόρμες συστοιχιών με προηγμένα δίκτυα διασύνδεσης υψηλών επιδόσεων.

1.4 Δημοσιεύσεις

Στη συνέχεια, αναφέρονται συγκεντρωτικά οι δημοσιεύσεις που προέκυψαν από την παρούσα διατριβή, οι οποίες και περιλαμβάνουν αθροιστικά τη συνολική ερευνητική μελέτη καθώς και τα συμπεράσματα αυτής.

ΔΙΕΘΝΗ ΠΕΡΙΟΔΙΚΑ

- N. Koziris, A. Sotiropoulos, G. Goumas, "**A Pipelined Schedule to Minimize Completion Time for Loop Tiling with Computation and Communication Overlapping**", *Journal of Parallel and Distributed Computing*, Volume 63, Issue 11, pp. 1138–1151, November 2003

ΔΙΕΘΝΗ ΣΥΝΕΔΡΙΑ

- G. Goumas, A. Sotiropoulos, N. Koziris, "**Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping**", In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS2001), IEEE Press*, San Francisco, California, April 2001 (**BEST PAPER AWARD**)

- A. Sotiropoulos, G. Tsoukalas, N. Koziris, "**Efficient Utilization of Memory Mapped NICs onto Clusters using Pipelined Schedules**", In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002)*, Berlin, Germany, May 2002

- A. Sotiropoulos, G. Tsoukalas, N. Koziris, "**Enhancing the Performance of Tiled Loop Execution onto Clusters using Memory Mapped Interfaces and Pipelined Schedules**", In *Proceedings of the 2002 Workshop on Communication Architecture for Clusters (CAC2002) held in conjunction with the 2002 International Parallel and Distributed Processing Symposium (IPDPS2002)*, Fort Lauderdale, Florida, April 2002

- M. Athanasaki, A. Sotiropoulos, G. Tsoukalas, N. Koziris, "**Pipelined Scheduling of Tiled Nested Loops onto Clusters of SMPs with Computation and Communication Overlapping**", In *Proceedings of the ACM/IEEE Supercomputing 2002: High Performance Networking and Computing Conference (SC2002)*, Baltimore, Maryland, November 2002

- A. Sotiropoulos, G. Tsoukalas, N. Koziris, "**A Pipelined Execution of Tiled Nested Loops onto a Cluster of PCs using PCI-SCI NICs**", In *Proceedings of the 2001 SCI-Europe Conference*, Dublin, Ireland, October 2001

- M. Athanasaki, A. Sotiropoulos, G. Tsoukalas, N. Koziris, "**A Pipelined Execution of Tiled Nested Loops on SMPs with Computation and Communication Overlapping**",

In *Proceedings of the Workshop on Compile/Runtime Techniques for Parallel Computing, held in conjunction with the 2002 International Conference on Parallel Processing (ICPP2002)*, Vancouver, Canada, August 2002

- A. Sotiropoulos, N. Koziris, "**A Pipelined Schedule for Loop Tiling to Minimize Overall Completion Time**", In *Proceedings of the 8th Panhellenic Conference on Informatics*, Nicosia, Cyprus, November 2001

Κεφάλαιο 2

Εφαρμογές – Προβλήματα – Υπολογισμοί

«Πρέπει να μας απασχολεί το καθετί,
γιατί μπορούμε να ερμηνεύσουμε το καθετί.»
– Hermann Hesse

Στο Κεφάλαιο αυτό αναφερόμαστε στη βασική θεωρία των φωλιασμένων βρόχων. Παρουσιάζουμε το μοντέλο των προβλημάτων που μας απασχολούν και τη σημειολογία που ακολουθούμε. Έχει πραγματοποιηθεί αρκετή έρευνα στον τομέα της επιτάχυνσης τμημάτων κώδικα που περιέχουν φωλιασμένους βρόχους, επειδή αποτελούν το σημαντικότερο παράγοντα της συνολικής καθυστέρησης της περάτωσης εκτέλεσης των προγραμμάτων. Η επιτάχυνση της εκτέλεσης φωλιασμένων βρόχων επιτυγχάνεται με τη χρήση κατάλληλων μετασχηματισμών, οι οποίοι αλλάζουν τη σειρά εκτέλεσης των στιγμιότυπων ενός φωλιασμένου βρόχου με έγκυρο τρόπο, οδηγώντας στην καλύτερη εκμετάλλευση των αρχιτεκτονικών χαρακτηριστικών των υπολογιστικών συστημάτων. Επικεντρωνόμαστε στο μετασχηματισμό υπερκόμβου, μια από τις χαρακτηριστικές ιδιότητες του οποίου είναι η δυνατότητα παράλληλης εκτέλεσης ομάδων επαναλήψεων του φωλιασμένου βρόχου, στον οποίο εφαρμόζεται, από διαφορετικούς επεξεργαστές.

2.1 Καθυστέρηση Περάτωσης Εκτέλεσης Κώδικα

Είναι γνωστό πως ο βασικότερος παράγοντας καθυστέρησης της εκτέλεσης των προγραμμάτων είναι οι επαναληπτικές δομές που περιέχουν. Αυτές εμφανίζονται συνήθως με τη μορφή φωλιασμένων βρόχων. Αν υποτεθεί ότι δεν υπάρχουν επαναληπτικές δομές εντός των προγραμμάτων,

τότε ο χρόνος περάτωσης της εκτέλεσης ενός προγράμματος θα εξαρτόταν μόνο από το μέγεθος του εκτελέσιμου κώδικα και την ισχύ του χρησιμοποιούμενου επεξεργαστή.

Αν θεωρήσουμε ότι ένα αρχείο εκτελέσιμου κώδικα έχει μέγεθος 100MB (που είναι σχετικά μεγάλο μέγεθος εκτελέσιμου αρχείου), το οποίο απασχολεί έντονα κυρίως την ΚΜΕ του συστήματος (CPU bound), τότε σε ένα υπολογιστικό σύστημα με επεξεργαστή με εσωτερική συχνότητα ρολογιού 1 Ghz που έχει δυνατότητα εκτέλεσης μιας εντολής μεγέθους 4 bytes ανά κύκλο ρολογιού, ο συνολικός χρόνος εκτέλεσης του αρχείου (κάνοντας την απλοποίηση $2^{10} \simeq 10^3$) είναι περίπου 25 msec.

Για παράδειγμα, σε συγκεκριμένη μοντέρνα μηχανή με επεξεργαστή της οικογένειας x86, με εσωτερική συχνότητα ρολογιού 2083 MHz, ο χρόνος περάτωσης ενός εκτελέσιμου προγράμματος μεγέθους 118407281 bytes, το οποίο δεν περιέχει εντολές διακλάδωσης, διαρκεί 0.1582 sec. Ο παραπάνω χρόνος είναι ο μέσος χρόνος 100 εκτελέσεων του κώδικα.

Παρόλα αυτά, από την εμπειρία μας γνωρίζουμε ότι η εκτέλεση αρκετών προγραμμάτων, ενδεχομένως με πολύ μικρότερο μέγεθος από αυτό του παραδείγματος, μπορεί να διαρκέσει πολύ περισσότερο από 0.1582 sec. Η εκτέλεση γνωστών υπολογιστικών προγραμμάτων μπορεί να διαρκέσει ώρες, μέρες ή ακόμα και εβδομάδες. Συνεπώς, αν θέλουμε να επιταχύνουμε την εκτέλεση αλγορίθμων που περιέχουν επαναληπτικές δομές, θα πρέπει να επικεντρωθούμε στην ελαχιστοποίηση του χρόνου εκτέλεσης των φωλιασμένων βρόχων. Στην παρούσα διατριβή αναλύεται η διαδικασία μέσω της οποίας μπορεί να επιταχυνθεί η παράλληλη εκτέλεση τέτοιων επαναληπτικών δομών, αξιοποιώντας πολλά χαρακτηριστικά των σύγχρονων παράλληλων αρχιτεκτονικών.

Στο παρόν κεφάλαιο παρουσιάζονται οι βασικές έννοιες των υπό εξέταση προβλημάτων και δίνονται οι γενικοί ορισμοί της σημειολογίας που ακολουθείται. Στη συνέχεια παρατίθεται ο μετασχηματισμός υπερκόμβου, που επιτρέπει την παραλληλοποίηση των συγκεκριμένων αλγορίθμων. Τέλος, εξετάζονται οι παράμετροι που επηρεάζουν την απόδοση των παραλληλοποιημένων αλγορίθμων σε περιβάλλον συστοιχίας υπολογιστών.

2.2 Μοντέλο Προβλημάτων

Η παρούσα διατριβή ασχολείται με αλγόριθμους που περιέχουν φωλιασμένους βρόχους με ομοιόμορφες εξαρτήσεις. Αναλυτικότερα, οι αλγόριθμοι είναι της μορφής:

```
for (i1 = l1; i1 <= u1; i1++){
    for (i2 = l2; i2 <= u2; i2++){
        ...
            for (in = ln; in <= un; in++){
                AS1(i)
                AS2(i)
```

```
                ...
                AS_k(i)
            } /* i_n */
        ...
    } /* i_2 */
} /* i_1 */
```

όπου:

1. l_i και u_i είναι σταθεροί ακέραιοι, που σημαίνει ότι το σύνολο επαναλήψεων είναι ένα παραλληλεπίπεδο,
2. $i = (i_1, \ldots, i_n)$ και
3. $AS_1, AS_2, \ldots, AS_k$ είναι δηλώσεις ανάθεσης της μορφής $V_0 = E(V_1, V_2, \ldots, V_l)$, όπου το V_0 είναι μεταβλητή εξόδου με δείκτη το i και παράγεται από την έκφραση E όταν εφαρμοστεί στις μεταβλητές εισόδου $V_1, V_2, \ldots, V_l$, με δείκτη το i.

2.2.1 Σημειολογία

Η σημειολογία που ακολουθείται στην παρούσα διατριβή είναι η ακόλουθη: $\mathbb{N}$ είναι το σύνολο των φυσικών αριθμών, $\mathbb{Z}$ είναι το σύνολο των ακεραίων, n είναι ο αριθμός των φωλιασμένων βρόχων του αλγόριθμου και m ο αριθμός των διανυσμάτων εξάρτησης του αλγόριθμου. $\mathbb{J}^n \subset \mathbb{Z}^n$ είναι το σύνολο των σημείων: $\mathbb{J}^n = \{j(j_1, \ldots, j_n) \mid j_i \in \mathbb{Z} \wedge l_i \leq j_i \leq u_i, 1 \leq i \leq n\}$. Κάθε σημείο σε αυτόν τον n-διάστατο χώρο ακεραίων είναι ένα ξεχωριστό στιγμιότυπο του σώματος του βρόχου. Ένα διάνυσμα εξαρτήσεων συμβολίζεται ως $d_i = (d_{i1}, \ldots, d_{in}), 1 \leq i \leq m$. Το σύνολο εξαρτήσεων ενός αλγόριθμου Α είναι το σύνολο όλων των διανυσμάτων εξάρτησης του αλγόριθμου: $D = \{d_1, d_2, \ldots, d_m\}$. Τέλος, όλα τα διανύσματα εξαρτήσεων θεωρούνται ομοιόμορφα, δηλ. οι τιμές τους είναι ανεξάρτητες των δεικτών υπολογισμού, και σταθερά. Σε έναν πίνακα A, με a_{ij} απεικονίζουμε το στοιχείο του πίνακα που βρίσκεται στην i-στή γραμμή και στην j-στή στήλη.

Στη συνέχεια κάνουμε μια εισαγωγή στο μετασχηματισμό υπερκόμβου, έναν από τους σημαντικότερους μετασχηματισμούς που εφαρμόζεται σε φωλιασμένους βρόχους. Ξεκινάμε παρουσιάζοντας τις δύο κυριότερες εφαρμογές του και στη συνέχεια προχωράμε σε μια μαθηματική προσέγγισή του.

2.3 Μετασχηματισμός Υπερκόμβου για Τοπικότητα Αναφορών

Η ταχύτητα του υποσυστήματος μνήμης επηρεάζει σημαντικά την απόδοση ενός υπολογιστικού συστήματος. Η ταχύτητα εκτέλεσης πράξεων των επεξεργαστών έχει αυξηθεί δραματικά σε

σχέση με την ταχύτητα της κεντρικής μνήμης, με αποτέλεσμα η καθυστέρηση της μνήμης να γίνεται όλο και πιο αισθητή ως παράγοντας που επιβραδύνει την εκτέλεση των προγραμμάτων. Για να αντιμετωπιστεί η καθυστέρηση, οι σχεδιαστές συστημάτων έχουν τοποθετήσει τις μικρές και γρήγορες κρυφές μνήμες (caches) κοντά στον επεξεργαστή του συστήματος, και τις μεγάλες και πιο αργές μνήμες μακρύτερα. Για να αξιοποιηθεί η συγκεκριμένη *ιεραρχία μνήμης*, είναι αναγκαίο τα δεδομένα που επαναχρησιμοποιούνται να μένουν στις γρήγορες και κοντινές στον επεξεργαστή μνήμες.

Ενώ οι κρυφές μνήμες είναι αναγκαίες για τη συγκράτηση της καθυστέρησης σε ανεκτά επίπεδα, οι αριθμητικοί κώδικες που επιλύουν διάφορα προβλήματα, δεν τις αξιοποιούν σε μεγάλο βαθμό. Ο λόγος για τον οποίο οι συγκεκριμένοι κώδικες δεν εκμεταλλεύονται πλήρως τις κρυφές μνήμες είναι ότι εργάζονται σε μεγάλα σύνολα δεδομένων. Όμως μια κρυφή μνήμη μπορεί να αποθηκεύσει μόνο ένα μικρό τμήμα αυτών (π.χ. ένα μικρό τμήμα ενός μεγάλου πίνακα). Στην περίπτωση αυτή, ακόμα και αν τα δεδομένα επαναχρησιμοποιούνται από τον κώδικα του προγράμματος, μπορεί να έχουν απομακρυνθεί από την κρυφή μνήμη μέχρι τη στιγμή που θα επαναχρησιμοποιηθούν. Υπάρχουν αρκετές οι αριθμητικές εφαρμογές που έχουν μεγάλο ποσοστό αστοχιών κρυφής μνήμης (cache misses) και μικρή τοπικότητα αναφορών.

Για να επιτευχθεί αποδεκτή απόδοση σε προγράμματα επιστημονικών υπολογισμών, θα πρέπει να γίνει σωστή εκμετάλλευση της ιεραρχίας μνήμης. Ο μετασχηματισμός υπερκόμβου είναι μια τεχνική που βελτιώνει την τοπικότητα των αναφορών σε επιστημονικούς κώδικες. Ο συγκεκριμένος μετασχηματισμός εφαρμόζεται για να βελτιώσει τη χρησιμοποίηση της κρυφής μνήμης, των καταχωρητών, της εικονικής μνήμης, του Translation Lookaside Buffer (TLB), κλπ.

Για να περιγράψουμε το μετασχηματισμό υπερκόμβου και τη σημασία του στην περίπτωση της επαναχρησιμοποίησης δεδομένων, ας θεωρήσουμε την περίπτωση του πολλαπλασιασμού πινάκων:

```
for (i1 = 1; i1 <= N; i1++){
    for (i2 = 1; i2 <= N; i2++){
        for (i3 = 1; i3 <= N; i3++){
            Z[i1,i3] += X[i1,i2] * Y[i2,i3];
        } /* i3 */
    } /* i2 */
} /* i1 */
```

Σε αυτόν τον κώδικα, η ίδια γραμμή του `Z` χρησιμοποιείται διαρκώς από τις επαναλήψεις του μεσαίου (i_2) βρόχου. Αν το N είναι μεγάλο σε σχέση με το μέγεθος της κρυφής μνήμης, έτσι ώστε μια ολόκληρη σειρά του πίνακα να μην χωράει στην κρυφή μνήμη, τότε στοιχεία της γραμμής μπορεί να μην βρίσκονται στην κρυφή μνήμη μεταξύ των επαναχρησιμοποιήσεων τους. Στην περίπτωση του πίνακα `Y`, η επαναχρησιμοποίηση στοιχείων του πίνακα συμβαίνει

στον εξωτερικό κόμβο (i_1). Ανάμεσα σε επαναχρησιμοποιήσεις του πίνακα `Y`, ολόκληρος ο πίνακας έχει μεταφερθεί στην κρυφή μνήμη, πράγμα που σημαίνει ότι οι αναφορές στον `Y` δεν θα βρεθούν στην κρυφή μνήμη.

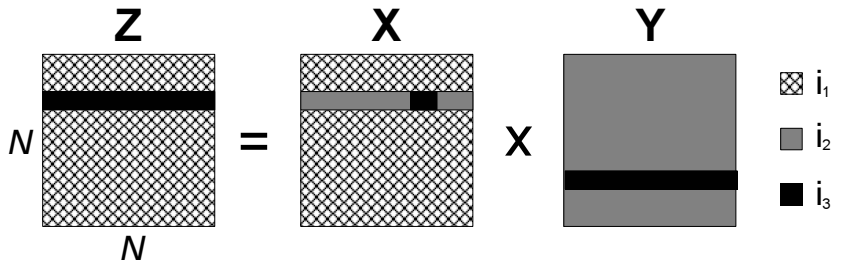
Ο μετασχηματισμός υπερκόμβου τροποποιεί τη σειρά με την οποία εκτελούνται οι επιμέρους επαναλήψεις, έτσι ώστε επαναλήψεις των εξωτερικών βρόχων να εκτελούνται πριν ολοκληρωθούν όλες οι επαναλήψεις των εσωτερικών βρόχων. Ο μετασχηματισμένος κώδικας του πολλαπλασιασμού πινάκων είναι:

```
for (ii2 = 1; ii2 <= N; ii2+=B){
    for (ii3 = 1; ii3 <= N; ii3+=B){
        for (i1 = 1; i1 <= N; i1++){
            for (i2 = ii2; i2 <= min(ii2 + B - 1, N); i2++){
                for (i3 = ii3; i3 <= min(ii3 + B - 1, N); i3++){
                    Z[i1,i3] += X[i1,i2] * Y[i2,i3];
                } /* i3 */
            } /* i3 */
        } /* i1 */
    } /* ii3 */
} /* ii2 */
```

Ο μετασχηματισμός έχει ως αποτέλεσμα τη μείωση του πλήθους των ενδιάμεσων επαναλήψεων ανάμεσα στις επαναχρησιμοποιήσεις δεδομένων. Συνεπώς, στην κρυφή μνήμη αποθηκεύονται λιγότερα δεδομένα, τα οποία μπορούν να επαναχρησιμοποιηθούν κατά τη διάρκεια των επαναλήψεων. Η παράμετρος B, που εκφράζει το μέγεθος του υπερκόμβου, μπορεί να επιλεγεί έτσι ώστε να επιτρέπει τη μέγιστη επαναχρησιμοποίηση για το συγκεκριμένο μέγεθος κρυφής μνήμης. Αν η τιμή του B είναι πολύ μεγάλη, τότε ανάμεσα σε 2 επαναχρησιμοποιήσεις του ίδιου στοιχείου x παρεμβάλονται πολλές χρήσεις άλλων στοιχείων, αυξάνοντας την πιθανότητα το στοιχείο x να έχει απομακρυνθεί από την κρυφή μνήμη.

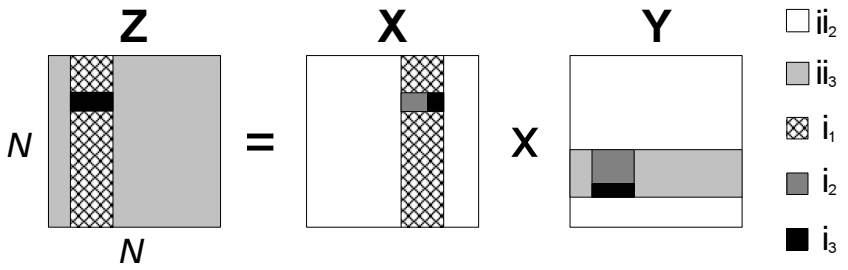
Στο Σχήμα 2.1, φαίνεται πώς προσπελαύνονται τα δεδομένα στον κλασσικό πολλαπλασιασμό πινάκων. Το ίδιο στοιχείο `X[`i_1`,`i_2`]` χρησιμοποιείται από όλες τις επαναλήψεις του εσωτερικού βρόχου. Υποθέτοντας ότι οι πίνακες είναι τοποθετημένοι στη μνήμη κατά σειρά (row major order), ο εσωτερικός βρόχος αυτού του κώδικα προσπελαύνει συνεχόμενα δεδομένα των πινάκων `Y` και `Z`, αξιοποιώντας πλήρως τους μηχανισμούς προφόρτωσης (prefetch) που διαθέτουν οι κρυφές μνήμες των επεξεργαστών. Η ίδια σειρά του πίνακα `Z`, που προσπελαύνεται από τον εσωτερικό βρόχο, επαναχρησιμοποιείται στην επόμενη επανάληψη του μεσαίου βρόχου και η ίδια σειρά του πίνακα `Y` επαναχρησιμοποιείται στον εξωτερικό βρόχο. Εάν ή όχι τα δεδομένα βρίσκονται στην κρυφή μνήμη τη στιγμή της επαναχρησιμοποίησης, εξαρτάται από το μέγεθος της κρυφής μνήμης. Αν η κρυφή μνήμη δεν είναι αρκετά μεγάλη ώστε να αποθηκεύσει τουλάχιστον έναν πίνακα $N \times N$, τα δεδομένα του πίνακα `Y` θα έχουν απομακρυνθεί μέχρι

την επαναχρησιμοποίησή τους. Αν η κρυφή μνήμη δεν μπορεί να αποθηκεύσει ούτε μία σειρά δεδομένων, τότε τα δεδομένα του πίνακα $\mathbb{Z}$ δεν μπορούν να επαναχρησιμοποιηθούν. Το μεγάλο ποσοστό προσπελάσεων της κεντρικής μνήμης προς αριθμητικές πράξεις μειώνει σημαντικά την απόδοση του συστήματος, αφού η κεντρική μνήμη έχει μεγάλη καθυστέρηση.



Σχήμα 2.1: Ο πολλαπλασιασμός πινάκων.

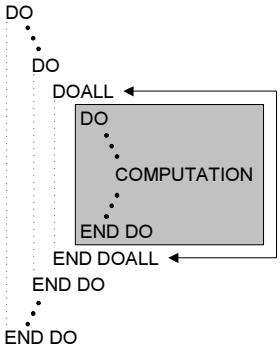
Στο Σχήμα 2.2, φαίνεται πώς τα δεδομένα προσπελαύνονται στην περίπτωση του πολλαπλασιασμού πινάκων στον οποίο έχει εφαρμοστεί ο μετασχηματισμός υπερκόμβου. Προφανώς, τα σχήματα προσπέλασης είναι τα ίδια, αλλά οι πράξεις εκτελούνται σε υποπίνακες του $\mathbb{Z}$ μεγέθους $B \times B$. Αν γίνει σωστή επιλογή του B, ο υποπίνακας χωράει πλήρως στην κρυφή μνήμη και μπορεί να επαναχρησιμοποιηθεί πολλές φορές. Αυτό επιτρέπει και στους τρεις πίνακες να έχουν πολύ καλό ποσοστό επαναχρησιμοποίησης.



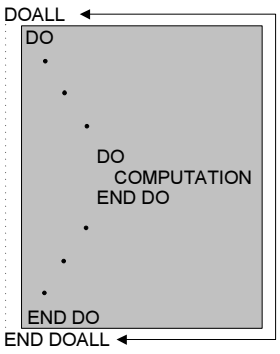
Σχήμα 2.2: Ο πολλαπλασιασμός πινάκων στον οποίο έχει εφαρμοστεί ο μετασχηματισμός υπερκόμβου.

2.4 Μετασχηματισμός Υπερκόμβου για Παραλληλία

Ο κώδικας των αριθμητικών προβλημάτων που μας ενδιαφέρουν περιέχει φωλιασμένους βρόχους. Αρκετά από αυτά τα προβλήματα είναι τέτοιας μορφής, ώστε να μπορούν να μετασχηματιστούν σε ισοδύναμο κώδικα που να περιέχει τουλάχιστον μία εντολή DOALL. Το γεγονός αυτό απλοποιεί την εργασία της παραλληλοποίησης, μιας και ό,τι βρίσκεται εσωτερικά της DOALL, μπορεί να εκτελεστεί παράλληλα (βλ. Σχήμα 2.3). Χρησιμοποιώντας κατάλληλους



Σχήμα 2.3: Κώδικας με φωλιασμένους βρόχους που περιέχει ένα DOALL. Φαίνεται σκιασμένο το τμήμα του κώδικα που μπορεί να εκτελεστεί παράλληλα.



Σχήμα 2.4: Μετασχηματισμένος κώδικας στον οποίο η DOALL είναι η εξωτερικός βρόχος.

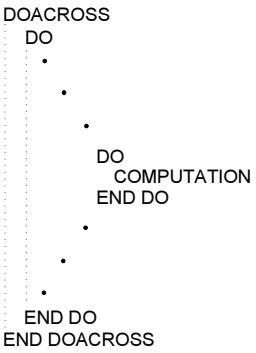
μετασχηματισμούς, η μορφή του κώδικα του Σχήματος 2.3 μπορεί να μετατραπεί σε αυτή του Σχήματος 2.4, όπου η DOALL είναι ο εξωτερικός βρόχος. Στην περίπτωση αυτή, τα επεξεργαστικά στοιχεία εκτελούν ανεξάρτητα το καθένα το εσωτερικό του DOALL και δεν χρειάζεται να επικοινωνήσουν καθόλου μεταξύ τους κατά τη διάρκεια της εκτέλεσης (υπάρχει μία μόνο επικοινωνία στην αρχή της εκτέλεσης, ώστε να καθοριστούν τα όρια μέσα στα οποία θα κινηθεί το κάθε επεξεργαστικό στοιχείο.)

Και στις δύο παραπάνω περιπτώσεις, ο κώδικας εσωτερικά του DOALL είναι αρκετά σημαντικός ώστε να κρατήσει τα επεξεργαστικά στοιχεία αρκετά απασχολημένα ώστε να εκτελέσουν ένα σημαντικό κομμάτι της συνολικής δουλειάς πριν αναγκαστούν να επικοινωνήσουν μεταξύ τους. Ιδίως σε περιβάλλοντα συστοιχιών υπολογιστών, όπου η κάθε επικοινωνία έχει σημαντικό κόστος, είναι επιθυμητό οι ανταλλαγές δεδομένων μεταξύ των στοιχείων επεξεργασίας να είναι όσο το δυνατόν λιγότερες. Πρέπει να σημειωθεί ότι δεν είναι πάντα δυνατόν να φτάσουμε σε πλήρως παραλληλοποιήσιμη μορφή κώδικα, αλλά είναι βολικό να υπάρχει έστω και μία εντολή DOALL στο φωλιασμένο βρόχο.

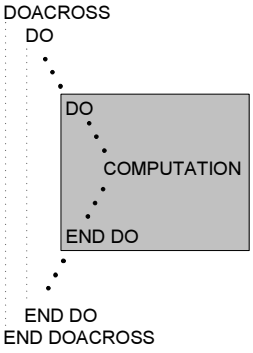
Όμως, δεν είναι τέτοιας μορφής όλες οι περιπτώσεις φωλιασμένων βρόχων που συναντώνται στα αριθμητικά προβλήματα. Υπάρχουν περιπτώσεις στις οποίες όλες οι εντολές ενός φωλιασμένου βρόχου είναι DOACROSS. Αυτό σημαίνει ότι οι εξαρτήσεις μεταξύ των επαναλήψεων είναι τέτοιες ώστε κάθε επανάληψη να εξαρτάται από κάποια προηγούμενή της έτσι ώστε απαιτείται συγχρονισμός.

Δοκιμάζοντας να παραλληλοποιήσουμε έναν κώδικα αυτής της μορφής, συμβαίνει το εξής: Αναλαμβάνοντας τα επεξεργαστικά στοιχεία να εκτελέσουν ένα στοιχείο του χώρου επαναλήψεων, πρέπει να επικοινωνήσουν κατάλληλα μεταξύ τους, ώστε να μπορέσουν να αποκτήσουν την απαιτούμενη πληροφορία για την ικανοποίηση των εξαρτήσεων. Όμως, η συγκεκριμένη

διαδικασία συμβαίνει σε κάθε βήμα επανάληψης, καταλήγοντας σε παράλληλη εκτέλεση fine grain. Η συγκεκριμένου τύπου εκτέλεση είναι εντελώς ακατάλληλη για εκτέλεση σε αρχιτεκτονικές χαλαρά συνδεδεμένων στοιχείων, μιας και κυρίαρχος παράγοντας του χρόνου εκτέλεσης γίνεται η επικοινωνία αντί των υπολογισμών.



Σχήμα 2.5: Κώδικας με φωλιασμένους βρόχους που περιέχει μόνο DOACROSS. Δεν υπάρχει κανένα τμήμα του κώδικα το οποίο να μπορεί να εκτελεστεί αποδοτικά σε παράλληλο σύστημα loosely coupled.



Σχήμα 2.6: Μετασχηματισμένος κώδικας με φωλιασμένους βρόχους που περιέχει μόνο DOACROSS. Έχουν δημιουργηθεί υπερκόμβοι των οποίων η παράλληλη εκτέλεση έχει νόημα σε σύστημα loosely coupled.

Αυτή η συνέπεια της εκτέλεσης fine grain οφείλεται στο ότι η επικοινωνία που επιβάλλεται ανάμεσα σε δύο στάδια υπολογισμών διαρκεί πολύ περισσότερο από αυτούς. Είναι πρακτικά ασύμφορο να πραγματοποιείται επικοινωνία για κάθε εκτέλεση ενός μικρού κομματιού του συνολικού προβλήματος.

Ο μετασχηματισμός υπερκόμβου επιτρέπει την παραλληλοποίηση αλγορίθμων DOACROSS, δημιουργώντας σύνολα επαναλήψεων τα οποία εκτελούνται ατομικά από κάθε επεξεργαστικό στοιχείο. Χρησιμοποιώντας μια ειδική μέθοδο απεικόνισης, οι υπερκόμβοι αντιστοιχίζονται στους υπολογιστικούς κόμβους μιας συστοιχίας υπολογιστών. Λαμβάνοντας υπόψη την επικοινωνία που ακολουθεί κάθε βήμα υπολογισμού, αντί κάθε επεξεργαστικό στοιχείο να αναλαμβάνει ένα μόνο σημείο του αρχικού χώρου επαναλήψεων, αναλαμβάνει ένα σύνολο σημείων. Αν το μέγεθος του συνόλου αυτού επιλεγεί κατάλληλα, αποκτάει νόημα η εκτέλεση των συγκεκριμένων συνόλων με παράλληλο τρόπο. Ο συγκεκριμένος μετασχηματισμός μετατρέπει την παράλληλη εκτέλεση fine grain σε coarse grain, που είναι κατάλληλη για περιβάλλοντα συστοιχιών υπολογιστών.

Με το μετασχηματισμό υπερκόμβου δημιουργείται ένας νέος χώρος επαναλήψεων στον οποίο η εκτέλεση ενός σημείου είναι συγκρίσιμη με την επικοινωνία που ακολουθεί κάθε εκτέλεση. Μεταξύ των σημείων του αρχικού χώρου, που ανήκουν σε έναν υπερκόμβο, συνεχίζουν να ισχύουν οι εξαρτήσεις του αρχικού χώρου, αλλά πλέον τα δεδομένα βρίσκονται στον ίδιο υπολογιστή και δεν απαιτείται επικοινωνία μέσω δικτύου για την ικανοποίηση των εξαρτήσεων.

Η επικοινωνία σε αυτή την περίπτωση πραγματοποιείται με τη μορφή εγγραφών και αναγνώσεων από την ιεραρχία μνήμης του υπολογιστή.

Στην περίπτωση αυτή, η επικοινωνία μεταξύ των κόμβων της παράλληλης αρχιτεκτονικής περιλαμβάνει τη μεταφορά περισσότερων δεδομένων ανά μεταφορά, αλλά συνολικά λιγότερες μεταφορές. Το συνολικό πλήθος των χρήσιμων δεδομένων που μεταφέρονται είναι το ίδιο. Όμως, η παράλληλη εκτέλεση επιταχύνεται, δεδομένου ότι πλέον υφίστανται λιγότερες φάσεις επικοινωνίας και συνεπώς λιγότερες αρχικοποιήσεις επικοινωνίας, οι οποίες επιβάλλουν σημαντική καθυστέρηση ανεξαρτήτως του μεγέθους των δεδομένων που μεταφέρονται.

2.5 Μαθηματική Προσέγγιση του Μετασχηματισμού Υπερκόμβου

Σύμφωνα με το μετασχηματισμό υπερκόμβου (supernode transformation – tiling), ο χώρος δεικτών $\mathbb{J}^n$ διαιρείται σε πανομοιότυπες n-διάστατες παραλληλεπίπεδες περιοχές, οι οποίες ονομάζονται υπερκόμβοι ή tiles και σχηματίζονται από n ανεξάρτητες οικογένειες παράλληλων υπερεπιπέδων. Ο μετασχηματισμός υπερκόμβου ορίζεται από τον n-διάστατο τετραγωνικό πίνακα H. Κάθε διάνυσμα γραμμής του H είναι κάθετο σε μια οικογένεια υπερεπιπέδων που σχηματίζουν τους υπερκόμβους.

Ισοδύναμα, ο μετασχηματισμός υπερκόμβου μπορεί να οριστεί με n γραμμικώς ανεξάρτητα διανύσματα, τα οποία είναι πλευρές του υπερκόμβου. Ομοίως με τον πίνακα H, ο πίνακας P περιέχει τα διανύσματα–πλευρές του υπερκόμβου σαν διανύσματα στήλες. Η σχέση που συνδέει τους δύο πίνακες είναι $P = H^{-1}$. Ο μετασχηματισμός υπερκόμβου ορίζεται πιο αυστηρά ως εξής:

$$r : \mathbb{Z}^n \longrightarrow \mathbb{Z}^{2n}, r(j) = \begin{bmatrix} \lfloor Hj \rfloor \\ j - H^{-1}\lfloor Hj \rfloor \end{bmatrix},$$

όπου $\lfloor Hj \rfloor$ είναι οι συντεταγμένες του υπερκόμβου που απεικονίζεται το σημείο $j(j_1, j_2, \ldots, j_n)$. Η έκφραση $j - H^{-1}\lfloor Hj \rfloor$ δίνει τις συντεταγμένες του σημείου j μέσα στον υπερκόμβο, ως προς την αρχή του υπερκόμβου. Συνεπώς, ο αρχικός n-διάστατος χώρος δεικτών μετασχηματίζεται σε ένα $2n$-διάστατο χώρο, το χώρο των υπερκόμβων και το χώρο των δεικτών μέσα στους υπερκόμβους. Οι δείκτες εσωτερικά των υπερκόμβων πρέπει να εκτελούνται ακολουθιακά, ενώ οι υπερκόμβοι μπορούν να ανατεθούν σε επεξεργαστές και να εκτελεστούν παράλληλα, σύμφωνα με μια έγκυρη χρονοδρομολόγηση.

Οι χώροι που ακολουθούν, παράγονται από έναν μετασχηματισμό υπερκόμβου H, όταν αυτός εφαρμοστεί σε ένα χώρο επαναλήψεων, στον οποίο ισχύουν οι εξαρτήσεις του συνόλου D.

1. Ο Χώρος Επαναλήψεων Υπερκόμβου (Tile Iteration Space)

$TIS(H) = \{j \in \mathbb{Z}^n \mid 0 \leq \lfloor Hj \rfloor \leq 1\}$ περιέχει όλα τα σημεία που ανήκουν στον υπερκόμβο που ξεκινάει από την αρχή των αξόνων.

2. Ο Χώρος Υπερκόμβων $\mathbb{J}^S(\mathbb{J}^n, H) = \left\{j^S \mid j^S = \lfloor Hj \rfloor, j \in \mathbb{J}^n\right\}$ περιέχει τις απεικονίσεις όλων των σημείων $j \in \mathbb{J}^n$ σύμφωνα με το μετασχηματισμό υπερκόμβου.

3. Χώρος Αρχής Υπερκόμβων (Tile Origin Space)
$TOS(\mathbb{J}^S, H^{-1}) = \left\{j \in \mathbb{Z}^n \mid j = H^{-1}j^S, j^S \in \mathbb{J}^S\right\}$ περιέχει όλες τις αρχές των υπερκόμβων στον αρχικό χώρο.

4. Ο Μετασχηματισμένος Πίνακας Εξαρτήσεων (Transformed Dependence Matrix)
$D^S = \left\{d^S \mid d^S = \lfloor H(j + d) \rfloor, d \in D, j \in TIS\right\}$ είναι ο πίνακας εξαρτήσεων του μετασχηματισμένου χώρου.

Σύμφωνα με τα παραπάνω ισχύουν τα εξής: $\mathbb{J}^n \xrightarrow{H} \mathbb{J}^S$ και $\mathbb{J}^S \xrightarrow{P} TOS$. Πρέπει να επισημανθεί ότι όλα τα σημεία του $\mathbb{J}^n$ που ανήκουν στον ίδιο υπερκόμβο, απεικονίζονται στο ίδιο σημείο στον $\mathbb{J}^S$, αφού ο μετασχηματισμός υπερκόμβου δεν είναι «1–1». Αυτό σημαίνει ότι κάθε σημείο j^S σε αυτόν τον n-διάστατο χώρο ακεραίων $\mathbb{J}^S$ είναι ένας ξεχωριστός υπερκόμβος με συντεταγμένες $(j_1^S, j_2^S, \ldots, j_n^S)$.

Δοθέντος ενός αλγόριθμου με πίνακα εξαρτήσεων D, για να είναι έγκυρος ένας μετασχηματισμός υπερκόμβου, πρέπει να ισχύει $HD \geq 0$. Με τον τρόπο αυτό εξασφαλίζεται η ατομικότητα των υπερκόμβων και η διατήρηση της αρχικής σειράς εκτέλεσης [IT88][RS92][SF91]. Σε αντίθετη περίπτωση, οποιαδήποτε σειρά εκτέλεσης των υπερκόμβων οδηγεί σε αδιέξοδο.

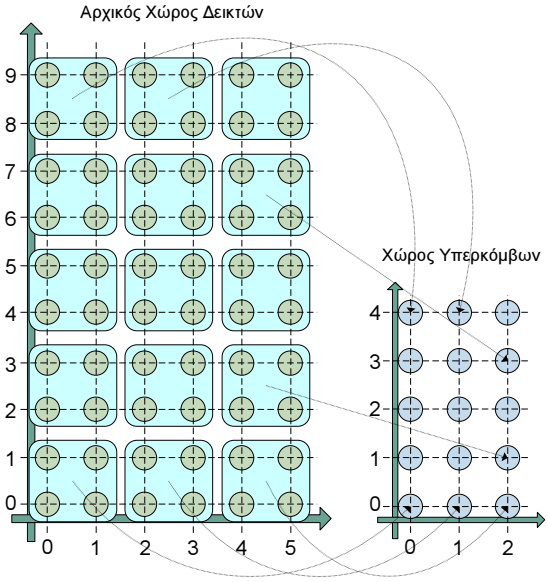
Στην παρούσα διατριβή, όλα τα διανύσματα εξαρτήσεων θεωρούνται μικρότερα από το μέγεθος του υπερκόμβου, ούτως ώστε να περιέχονται πλήρως στην περιοχή κάθε υπερκόμβου [Xue97b]. Αυτό σημαίνει ότι $|HD| < 1$, ή αλλιώς ότι ο πίνακας εξαρτήσεων των υπερκόμβων περιέχει μόνο τα στοιχεία «0» και «1». Η συγκεκριμένη παραδοχή είναι εύλογη, αφού σε συστήματα με ισχυρούς επεξεργαστές, όπως είναι τα συστήματα που μελετάμε, τα διανύσματα εξάρτησης των προβλημάτων είναι σχετικά μικρά, ενώ τα μεγέθη των υπερκόμβων είναι τάξεις μεγέθους μεγαλύτερα. Σε αυτή την περίπτωση, κάθε υπερκόμβος χρειάζεται να ανταλλάξει δεδομένα μόνο με τους πλησιέστερους γειτονικούς υπερκόμβους, ένα σε κάθε διάσταση του χώρου $\mathbb{J}^n$.

2.5.1 Παράδειγμα Μετασχηματισμού Υπερκόμβου

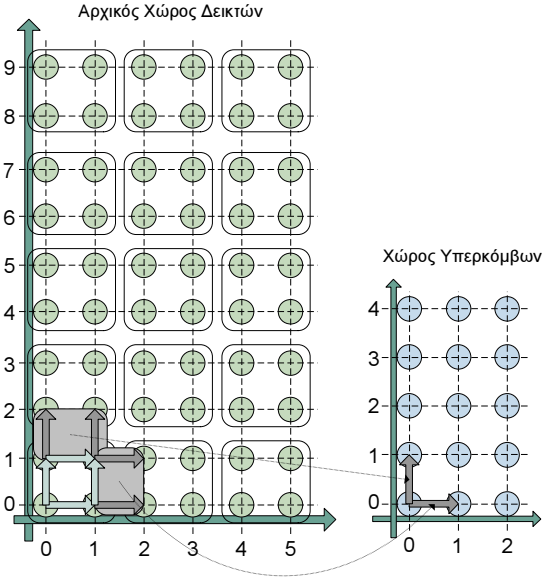
Ας θεωρήσουμε τον πίνακα A μεγέθους 6×10. Το επόμενο κομμάτι πηγαίου κώδικα σε γλώσσα C, υπολογίζει τα στοιχεία του πίνακα. Κάθε στοιχείο του πίνακα ισούται με το άθροισμα δύο άλλων στοιχείων του πίνακα.

```
for (i = 0; i <= 5; i++){
    for (j = 0; j <= 9; j++){
        A[i][j] = A[i-1][j] + A[i][j-1];
    }
}
```

Όπως φαίνεται από τον κώδικα, ο Αρχικός Χώρος Δεικτών περιέχει τα στοιχεία $\mathbb{J} = \{j \in (i,j) \mid 0 \leq i \leq 5,\ 0 \leq j \leq 9\}$ και ο πίνακας εξαρτήσεων είναι $D = \{(1,0),(0,1)\}$. Θεωρώντας τον τετραγωνικό πίνακα μετασχηματισμό $H = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$, ισχύει $P = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$. Στο Σχήμα 2.7 φαίνεται ο αρχικός χώρος δεικτών, καθώς και ο μετασχηματισμένος χώρος αυτών. Τα διανύσματα στήλες του πίνακα P, που είναι πλευρές του υπερκόμβου, είναι τα $(2,0)$ και $(0,2)$. Συνεπώς, κάθε υπερκόμβος περιέχει 4 στοιχεία, όπως φαίνεται στο Σχήμα 2.7. Όπως φαίνεται στο ίδιο σχήμα, ο Χώρος Υπερκόμβων περιέχει τα στοιχεία



Σχήμα 2.7: Η ομαδοποίηση των επιμέρους επαναλήψεων στο μετασχηματισμό υπερκόμβου.



Σχήμα 2.8: Η ομαδοποίηση των εξαρτήσεων στο μετασχηματισμό υπερκόμβου.

$\mathbb{J}^S = \left\{j^S \in (i,j) \mid 0 \leq i \leq 2,\ 0 \leq j \leq 4\right\}$. Στο Σχήμα 2.8 φαίνονται οι ίδιοι χώροι μαζί με κάποιες χαρακτηριστικές εξαρτήσεις, οι οποίες ομαδοποιούνται. Ο Μετασχηματισμένος Πίνακας Εξαρτήσεων περιέχει τα διανύσματα $D^S = \{(1,0),(0,1)\}$.

2.5.2 Κόστος Υπολογισμού Υπερκόμβου

Ο συνολικός αριθμός των σημείων επανάληψης μέσα σε έναν υπερκόμβο αναπαριστά το συνολικό υπολογιστικό κόστος του συγκεκριμένου υπερκόμβου και υπολογίζεται από την ορίζουσα του πίνακα P ($\det(P)$). Έτσι, ισχύει ότι το κόστος υπολογισμού $V_{comp} = |\det(P)|$. Στο παράδειγμα της προηγούμενης παραγράφου, ισχύει $|\det(P)| = |\det\left(\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\right)| = 4$.

2.6 Κόστος Επικοινωνίας μεταξύ Υπερκόμβων

Το κόστος επικοινωνίας ενός υπερκόμβου είναι ανάλογο του πλήθους των σημείων επανάληψης που χρειάζεται να στείλουν δεδομένα σε γειτονικούς υπερκόμβους. Με άλλα λόγια, είναι ανάλογο του αθροίσματος των διανυσμάτων εξάρτησης που τέμνουν τις πλευρές του υπερκόμβου. Τα συγκεκριμένα σημεία ονομάζονται σημεία επικοινωνίας. Ο αριθμός των σημείων επικοινωνίας υπολογίζεται από την έκφραση ([HS98])

$$V_{comm}(H) = \frac{1}{|det(H)|} \sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{j=1}^{m} h_{i,k} d_{k,j}.$$

Πρακτικά, ο τύπος αυτός υπολογίζει το κόστος επικοινωνίας V_{comm} σαν το άθροισμα όλων των πιθανών $h_i d_i$, το οποίο εκφράζει τη συνεισφορά κάθε διανύσματος εξάρτησης στην επικοινωνία, για κάθε οριακή επιφάνεια του υπερκόμβου.

Αν οι υπερκόμβοι κατά μήκος της ίδιας διάστασης ανατεθούν για εκτέλεση στον ίδιο επεξεργαστή, τα διανύσματα εξάρτησης που τέμνουν την οριακή επιφάνεια στην αντίστοιχη διάσταση δεν επιβάλλουν επιπλέον διεπεξεργαστική επικοινωνία. Στην περίπτωση αυτή, το κόστος επικοινωνίας υπολογίζεται από την έκφραση

$$V_{comm}(H) = \frac{1}{|det(H)|} \sum_{\substack{i \in \{1,\ldots,x-1,x+1,\ldots,n\} \\ j \in \{1,\ldots,m\}}} (H_{-x} D)_{i,j'},$$

όπου το H_{-x} είναι ο πίνακας H χωρίς το διάνυσμα στήλη που είναι κάθετο στην οριακή επιφάνεια στη διάσταση απεικόνισης στον ίδιο επεξεργαστή.

2.7 Παράγοντες Απόδοσης

Χρησιμοποιώντας το μετασχηματισμό υπερκόμβου, μας παρέχεται η δυνατότητα να ομαδοποιούμε πολλά στοιχεία υπολογισμού σε έναν υπερκόμβο, ο οποίος εκτελείται ατομικά. Με όμοιο τρόπο ομαδοποιούμε και τις εξαρτήσεις μεταξύ των υπερκόμβων σε μηνύματα, τα οποία

περιέχουν τη συνολική πληροφορία που απαιτείται για την εκτέλεση των επόμενων βημάτων του υπολογισμού.

Θεωρώντας ότι κάθε υπερκόμβος απεικονίζεται σε ένα μόνο επεξεργαστικό στοιχείο προκειμένου να εκτελεστεί, και ότι οι μεταξύ τους εξαρτήσεις αντιστοιχούν σε επικοινωνία ανάμεσα στα επεξεργαστικά στοιχεία, γίνεται εμφανές ότι οι δύο παράγοντες που καθορίζουν την απόδοση της παράλληλης εκτέλεσης του μετασχηματισμένου βρόχου είναι η ταχύτητα εκτέλεσης των υπερκόμβων και η ταχύτητα ανταλλαγής δεδομένων μεταξύ των υπερκόμβων. Τα δύο υποσυστήματα που είναι υπεύθυνα για τους παράγοντες αυτούς είναι το υποσύστημα της κεντρικής μονάδας επεξεργασίας και το υποσύστημα του δικτύου.

Στα επόμενα κεφάλαια θα προταθεί μία μέθοδος αξιοποίησης της δυνατότητας συνδυασμένης λειτουργίας της ΚΜΕ και του υποσυστήματος δικτύου, σε ένα υπολογιστικό σύστημα, για υπολογιστικά προβλήματα που έχουν μετασχηματιστεί με τη μέθοδο των υπερκόμβων. Η παρούσα εργασία είναι η πρώτη η οποία συσχετίζει τα προηγμένα χαρακτηριστικά των μοντέρνων δικτυακών τεχνολογιών με τη θεωρία των υπερκόμβων. Η προτεινόμενη μέθοδος, λαμβάνει υπόψη τις επιμέρους αρχιτεκτονικές παραμέτρους, οδηγώντας σε χρονοδρομολόγηση που αξιοποιεί πλήρως τα προηγμένα δίκτυα διασύνδεσης κατά την παράλληλη εκτέλεση αλγορίθμων σε συστοιχίες υπολογιστών. Στο Κεφάλαιο 3 παρουσιάζεται ο τρόπος με τον οποίο λειτουργούν οι συμβατικές δικτυακές τεχνολογίες προκειμένου να παράσχουν την υπηρεσία της επικοινωνίας. Στη συνέχεια αναφέρονται τα προηγμένα χαρακτηριστικά που προσφέρουν τα νέα δίκτυα διασύνδεσης και αναλύεται ο τρόπος λειτουργίας τους. Στο Κεφάλαιο 4 παρουσιάζεται η προτεινόμενη μέθοδος επικαλυπτόμενης χρονοδρομολόγησης, η οποία λαμβάνει υπόψη της τα νέα χαρακτηριστικά των δικτύων διασύνδεσης, σε αντιδιαστολή με το συμβατικό τρόπο χρονοδρομολόγησης των παράλληλων εφαρμογών.

Κεφάλαιο 3

Συστοιχίες Υπολογιστών στην Παράλληλη Επεξεργασία

«Είμαστε δυο, είμαστε τρεις,
είμαστε χίλιοι δεκατρείς...»
– Μίκης Θεοδωράκης

Στο παρόν κεφάλαιο, παρουσιάζεται το μοντέλο της συστοιχίας υπολογιστών, ενός περιβάλλοντος παράλληλης επεξεργασίας το οποίο χρησιμοποιείται εκτενώς τα τελευταία χρόνια. Αναφέρονται οι λόγοι επικράτησης του συγκεκριμένου μοντέλου έναντι των παραδοσιακών συστημάτων παράλληλης επεξεργασίας. Αναλύεται ο τρόπος λειτουργίας των συμβατικών δικτύων διασύνδεσης σε συνδυασμό με τα πρωτόκολλα επικοινωνίας που χρησιμοποιούν. Εντοπίζονται τα σημεία που προκαλούν αύξηση της καθυστέρησης επικοινωνίας, η οποία επηρεάζει την εκτέλεση των παράλληλων εφαρμογών. Παρουσιάζονται νέα αρχιτεκτονικά χαρακτηριστικά επικοινωνίας που παρακάμπτουν το Λ.Σ., εξαλείφουν τις μη-αποδοτικές αντιγραφές δεδομένων και παρέχουν τη δυνατότητα για ταυτόχρονη χρήση της ΚΜΕ και του δικτυακού υποσυστήματος ενός υπολογιστικού κόμβου. Τα προηγμένα χαρακτηριστικά του υλικού και του λογισμικού επικοινωνίας που παρουσιάζονται στο παρόν κεφάλαιο, θα χρησιμοποιηθούν στα επόμενα κεφάλαια για την υλοποίηση εφαρμογής που επιτυγχάνει επικάλυψη χρόνου υπολογισμού με χρόνο επικοινωνίας.

3.1 Αρχιτεκτονικές Παράλληλης Επεξεργασίας

Η περιοχή των υπολογισμών υψηλών επιδόσεων υποστηρίζεται από αρκετά υπολογιστικά συστήματα. Τα συστήματα αυτά ταξινομούνται στις ακόλουθες κατηγορίες, ανάλογα με τον τύπο

των επεξεργαστών τους, την κεντρική μνήμη τους και το δίκτυο διασύνδεσής τους:

- Μαζικά Παράλληλοι Επεξεργαστές (Massively Parallel Processors – MPP)
- Συμμετρικοί Πολυεπεξεργαστές (Symmetric Multiprocessors – SMP)
- Υπολογιστές με Ανομοιόμορφη Πρόσβαση στη Μνήμη με Συνάφεια Κρυφής Μνήμης (Cache-Coherent Nonuniform Memory Access – CC-NUMA)
- Κατανεμημένα Συστήματα (Distributed Systems)
- Συστοιχίες Υπολογιστών (Clusters)

Οι μαζικά παράλληλοι υπολογιστές είναι μεγάλα συστήματα παράλληλης επεξεργασίας, των οποίων η αρχιτεκτονική ακολουθεί το μοντέλο «shared-nothing». Συνήθως, αποτελούνται από εκατοντάδες επεξεργαστικά στοιχεία (κόμβους), τα οποία είναι διασυνδεδεμένα μέσω ενός δικτύου–μεταγωγέα υψηλής ταχύτητας. Κάθε κόμβος μπορεί να αποτελείται από ποικίλα στοιχεία υλικού, αλλά στη γενική περίπτωση αποτελείται από μία κεντρική μνήμη και έναν ή περισσότερους επεξεργαστές. Επιπρόσθετα, ειδικοί κόμβοι μπορούν να επικοινωνούν με περιφερειακά, όπως δίσκους ή συστήματα αντιγράφων ασφαλείας, κλπ. Τέλος, ένα ξεχωριστό αντίγραφο του λειτουργικού συστήματος (operating system image) εκτελείται σε κάθε κόμβο.

Οι συμμετρικοί πολυεπεξεργαστές περιέχουν από δύο μέχρι λίγες εκατοντάδες κόμβους και μπορούν να θεωρηθούν ότι ακολουθούν την αρχιτεκτονική «shared-everything». Στα συστήματα αυτά, οι επεξεργαστές μοιράζονται όλους τους γενικούς διαθέσιμους πόρους (διαύλους, μνήμη, υποσυστήματα Ε/Ε, κλπ.) Όλοι οι κόμβοι εκτελούν το ίδιο αντίγραφο του λειτουργικού συστήματος. Αυτή η κατηγορία συστημάτων παράλληλης επεξεργασίας οφείλει την ονομασία της στο γεγονός ότι η προσπέλαση στην κοινή μνήμη κοστίζει το ίδιο, ανεξάρτητα από τον επεξεργαστή που πραγματοποιεί πρόσβαση στα δεδομένα. Ένα τυπικό σύστημα συμμετρικής πολυεπεξεργασίας περιέχει συνήθως δύο ή τέσσερις επεξεργαστές.

Οι υπολογιστές CC-NUMA είναι επεκτάσιμα πολυεπεξεργαστικά συστήματα που ακολουθούν την αρχιτεκτονική Μη-Ομοιόμορφης Προσπέλασης Μνήμης με Συνάφεια Κρυφής Μνήμης. Όπως και στην περίπτωση των συμμετρικών πολυεπεξεργαστών, κάθε επεξεργαστής σε ένα σύστημα CC-NUMA έχει κεντρική πρόσβαση σε ολόκληρη τη μνήμη. Η ονομασία τους οφείλεται στα ανομοιόμορφα χρονικά διαστήματα που απαιτούνται για την προσπέλαση των κοντινότερων τμημάτων της φυσικής μνήμης σε σχέση με τα πιο απομακρυσμένα.

Τα κατανεμημένα συστήματα μπορούν να θεωρηθούν σαν συμβατικά δίκτυα ανεξάρτητων υπολογιστών. Κάθε κόμβος εκτελεί διαφορετικό αντίγραφο του λειτουργικού συστήματος και οι επιμέρους μηχανές μπορεί να είναι, για παράδειγμα, συνδυασμός από μαζικά παράλληλους υπολογιστές, συμμετρικούς πολυεπεξεργαστές, συστοιχίες ή ανεξάρτητους υπολογιστές. Στην κατηγορία αυτή ανήκουν τα κατανεμημένα συστήματα Grids.

Χαρακτηριστικό	Μαζικά Παράλληλοι Υπολογιστές	Συμμετρικοί Πολυεπεξεργαστές / CC-NUMA	Κατανεμημένα Συστήματα	Συστοιχίες
Πλήθος Κόμβων	100–1000	10–100	10–10000	10–1000
Χώρος/οι Διευθύνσεων	Πολλαπλοί	Ενιαίος	Πολλαπλοί	Πολλαπλοί
Διακομβική επικοινωνία	Πέρασμα Μηνυμάτων	Μοιραζόμενη Μνήμη	Μοιραζόμενα Αρχεία, RPC, Πέρασμα Μηνυμάτων	Πέρασμα Μηνυμάτων

Πίνακας 3.1: Χαρακτηριστικά Παράλληλων Υπολογιστών

Τέλος, οι συστοιχίες υπολογιστών είναι συλλογές από σταθμούς εργασίας ή προσωπικούς υπολογιστές που είναι διασυνδεδεμένοι με μια δικτυακή τεχνολογία. Η συστοιχία υπολογιστών, στην οποία εκτελούνται παράλληλοι υπολογισμοί, αποτελείται συνήθως από όμοιους σταθμούς εργασίας υψηλής επίδοσης που διασυνδέονται μέσω ενός δικτύου υψηλής ταχύτητας. Αν και κάθε κόμβος της συστοιχίας εκτελεί ένα διαφορετικό αντίγραφο του λειτουργικού συστήματος, υπάρχει συνήθως ένα στρώμα ενδιάμεσου λογισμικού (middleware), ο ρόλος του οποίου είναι να ενοποιεί το πλήθος των κόμβων της συστοιχίας, ώστε να φαίνεται στον εξωτερικό κόσμο σαν ένας ενοποιημένος πόρος. Στον Πίνακα 3.1, φαίνονται τα βασικότερα χαρακτηριστικά των παραλλήλων υπολογιστών και πώς αυτά διαφοροποιούνται στις διάφορες κατηγορίες.

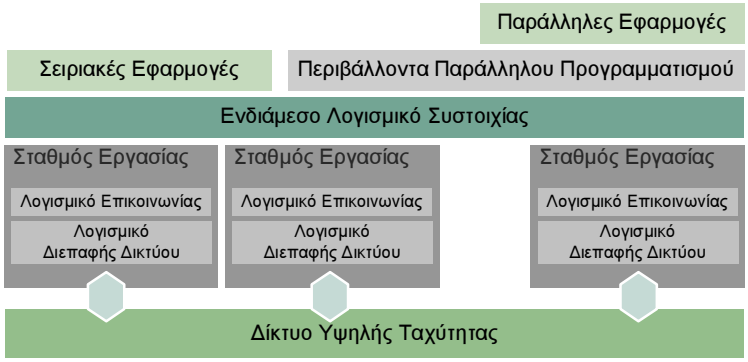
3.2 Περί Συστοιχιών Υπολογιστών

Τα τελευταία δέκα χρόνια, ο τομέας των αρχιτεκτονικών παράλληλης επεξεργασίας υπόκειται σε σημαντικές αλλαγές λόγω δύο βασικών εξελίξεων· την αύξηση της ισχύος των μικροεπεξεργαστών και την αύξηση της ταχύτητας των δικτύων δεδομένων. Οι δύο αυτές εξελίξεις, σε συνδυασμό με το χαμηλό κόστος απόκτησης σταθμών εργασίας και δικτυακών υποδομών, έχουν δημιουργήσει ένα νέο τοπίο στην παράλληλη επεξεργασία, η οποία πραγματοποιείται πλέον σε ομάδες διασυνδεδεμένων προσωπικών υπολογιστών (PC) ή σταθμών εργασίας (workstations), που ονομάζονται συστοιχίες υπολογιστών (computer clusters) [Pfi98, Buy99].

Η συστοιχία υπολογιστών, ως αρχιτεκτονική παράλληλης επεξεργασίας, είναι ένας τύπος κατανεμημένου επεξεργαστικού συστήματος, που αποτελείται από μια συλλογή ανεξάρτητων υπολογιστών, οι οποίοι εργάζονται μαζί σαν ένας μοναδικός, ενοποιημένος πόρος.

Ένας υπολογιστικός κόμβος μπορεί να είναι μονοεπεξεργαστικός ή πολυεπεξεργαστικός (προσωπικός υπολογιστής, σταθμός εργασίας, συμμετρικός πολυεπεξεργαστής SMP) με μνήμη, μονάδες εισόδου/εξόδου και λειτουργικό σύστημα. Οι κόμβοι που απαρτίζουν μια συστοιχία μπορεί να είναι είτε μέσα σε κάποιο ικρίωμα (rack), είτε φυσικώς διαχωρισμένοι. Σε κάθε περίπτωση, είναι διασυνδεδεμένοι μέσω ενός γρήγορου τοπικού δικτύου. Η τυπική αρχιτεκτονική

μιας συστοιχίας υπολογιστών φαίνεται στο Σχήμα 3.1.



Σχήμα 3.1: Αρχιτεκτονική Συστοιχίας Υπολογιστών

3.2.1 Γιατί Συστοιχίες Υπολογιστών;

Τη δεκαετία του 1980, ήταν γενική πεποίθηση ότι ο καλύτερος τρόπος για βελτίωση της υπολογιστικής επίδοσης ήταν μέσω της δημιουργίας γρηγορότερων και πιο αποδοτικών επεξεργαστών. Από τις αρχές του 1990 υπήρχε μία αυξανόμενη τάση για απομάκρυνση από τους ακριβούς και κλειστούς παράλληλους υπερυπολογιστές προς δίκτυα σταθμών εργασίας. Ανάμεσα στις δυνάμεις που κατέστησαν εφικτή τη συγκεκριμένη μετάβαση, είναι η ταχεία βελτίωση της διαθεσιμότητας κοινών «εξαρτημάτων» (components) υψηλής επίδοσης για τους σταθμούς εργασίας και τα δίκτυα. Οι τεχνολογίες αυτές, καθιστούν τις συστοιχίες υπολογιστών ένα δελεαστικό μέσο για την πραγματοποίηση παράλληλης επεξεργασίας, οδηγώντας προς την εποχή της υπερυπολογιστικής ισχύος χαμηλού κόστους.

Η αξιοποίηση της παράλληλης επεξεργασίας ως το μέσο που θα παρείχε υπολογιστικές μονάδες υψηλής επίδοσης για εφαρμογές μεγάλης κλίμακας, έχει εξερευνηθεί διεξοδικά. Μέχρι πρόσφατα, όμως, τα πλεονεκτήματα αυτής της έρευνας περιορίζονταν μόνο σε αυτούς που είχαν πρόσβαση σε τέτοια συστήματα. Πλέον, η τάση στην παράλληλη επεξεργασία είναι η απομάκρυνση από τις εξειδικευμένες υπερυπολογιστικές πλατφόρμες, π.χ. όπως ο υπολογιστής CRAY/SGI T3E, σε φθηνότερα, γενικού σκοπού συστήματα που αποτελούνται από χαλαρά-συνδεδεμένα στοιχεία· μονοεπεξεργαστικούς ή πολυεπεξεργαστικούς σταθμούς εργασίας. Αυτή η προσέγγιση έχει ένα πλήθος πλεονεκτημάτων, συμπεριλαμβάνοντας τη δυνατότητα δημιουργίας–συναρμολόγησης πλατφόρμας για δεδομένο προϋπολογισμό, γεγονός πολύ βολικό για ένα μεγάλο πλήθος διαφορετικών εφαρμογών και φόρτων εργασίας.

Η χρήση συστοιχιών υπολογιστών για την προτυποποίηση, αποσφαλμάτωση και την εκτέλεση παράλληλων εφαρμογών, γίνεται μια αυξανόμενα δημοφιλής εναλλακτική της χρήσης εξειδικευμένων, συνήθως πανάκριβων, υπολογιστικών πλατφόρμων. Ένας σημαντικός παράγο-

ντας που έχει κάνει τη χρήση συστοιχιών μια πρακτική πρόταση είναι η προτυποποίηση αρκετών λογισμικών εργαλείων που χρησιμοποιούνται από τις παράλληλες εφαρμογές. Παραδείγματα αυτών των προτύπων είναι οι βιβλιοθήκες ανταλλαγής μηνυμάτων MPI [MF] και PVM [PVM]. Αυτή η προτυποποίηση επιτρέπει την ανάπτυξη, τον έλεγχο και την εκτέλεση εφαρμογών σε συστοιχίες και σε ένα επόμενο στάδιο τη μεταφορά τους, ίσως με λίγες τροποποιήσεις, σε εξειδικευμένες παράλληλες πλατφόρμες όπου ο υπολογιστικός χρόνος χρεώνεται. Το κομμάτι της μεταφοράς σε εξειδικευμένες παράλληλες πλατφόρμες ακολουθείται όλο και λιγότερο, ενώ οι συστοιχίες χρησιμοποιούνται και για την επιχειρησιακή εκτέλεση των αναπτυγμένων εφαρμογών.

Η ακόλουθη λίστα αναφέρει μερικούς από τους λόγους για τους οποίους οι συστοιχίες υπολογιστών προτιμώνται έναντι των εξειδικευμένων παράλληλων υπολογιστών [ACP95, BFY96]:

- Οι επιμέρους σταθμοί εργασίας γίνονται ολοένα και πιο ισχυροί. Η επίδοση των μικροεπεξεργαστών έχει αυξηθεί δραματικά τα τελευταία χρόνια και διπλασιάζεται κάθε 18 μήνες [PH94], ακολουθώντας το νόμο του Moore [Moo65]. Η τάση αυτή είναι πιθανόν να συνεχιστεί και τα επόμενα χρόνια, με την έλευση στην αγορά γρηγορότερων επεξεργαστών και πιο αποδοτικών πολυεπεξεργαστών.

- Η δυνατότητα παροχής (bandwidth) των δικτύων που διασυνδέουν σταθμούς εργασίας αυξάνεται, ενώ η καθυστέρησή τους μειώνεται, όσο νέες τεχνολογίες και πρωτόκολλα υλοποιούνται στα περιβάλλοντα τοπικών δικτύων.

- Οι συστοιχίες υπολογιστών είναι ευκολότερο να ενσωματωθούν στα υπάρχοντα δίκτυα, σε αντίθεση με τους παράλληλους υπολογιστές.

- Η ανάπτυξη λογισμικών εργαλείων για τους σταθμούς εργασίας είναι πιο ώριμη σε σύγκριση με τις κλειστές λύσεις των παράλληλων υπολογιστών, γεγονός που οφείλεται κυρίως στη μη-προτυποποιημένη φύση των παράλληλων συστημάτων.

- Οι συστοιχίες υπολογιστών είναι μια φθηνή και έτοιμη προς χρήση εναλλακτική των εξειδικευμένων υπολογιστικών πλατφόρμων υψηλών επιδόσεων.

- Οι συστοιχίες υπολογιστών μπορούν εύκολα να επεκταθούν. Επίσης, οι δυνατότητες των επιμέρους κόμβων μπορούν να βελτιωθούν εύκολα, προσθέτοντας επιπλέον μνήμη και επεξεργαστές.

- Η πολυπλοκότητα των εξειδικευμένων παράλληλων υπολογιστών είναι πολύ μεγαλύτερη σε σχέση με αυτή των κόμβων που χρησιμοποιούνται ως συστατικό των συστοιχιών υπολογιστών. Έτσι, τα τεχνολογικά επιτεύγματα στους τομείς των επεξεργαστών και των δικτύων ενσωματώνονται σχεδόν άμεσα στην αρχιτεκτονική των συστοιχιών, σε αντίθεση

με τα κλειστά παράλληλα συστήματα που περιέχουν μονάδες των οποίων η τεχνολογία είναι παλαιότερη και συνήθως παρωχημένη.

Παραδοσιακά, στον επιστημονικό τομέα και στον τομέα των επιχειρήσεων, ο όρος «σταθμός εργασίας» αναφερόταν σε μία πλατφόρμα που εκτελούσε λειτουργικό σύστημα UNIX, ενώ οι επικρατούσες λειτουργίες ενός προσωπικού υπολογιστή ήταν οι διαχειριστικές εργασίες και η επεξεργασία κειμένου. Όμως υπήρξε μία ταχεία σύγκλιση στις επιδόσεις των επεξεργαστών και στη λειτουργικότητα του πυρήνα των λειτουργικών συστημάτων των σταθμών εργασίας UNIX και των προσωπικών υπολογιστών (κυρίως λόγω της εισόδου των επεξεργαστών της οικογένειας Pentium Pro της Intel και του λειτουργικού συστήματος Linux). Αυτή η σύγκλιση έχει οδηγήσει σε μία αυξανόμενη χρήση των συστημάτων προσωπικών υπολογιστών σαν έναν αποτελεσματικό υπολογιστικό πόρο για εκτέλεση παράλληλων υπολογισμών. Ο συγκεκριμένος παράγοντας, σε συνδυασμό με το συγκριτικά χαμηλό κόστος των προσωπικών υπολογιστών και την ευρεία διαθεσιμότητά τους στον ακαδημαϊκό χώρο και στο χώρο των επιχειρήσεων, έχει προωθήσει την ανάπτυξη αρκετών έργων κατασκευής λογισμικού, που στόχο έχουν την εκμετάλλευση αυτών των πόρων με έναν τρόπο που να ευνοεί τη συνεργασία και τη διαλειτουργικότητα.

Τα τελευταία χρόνια οι συστοιχίες υπολογιστών έχουν εισβάλλει σε όλους τους τομείς που απαιτούν υψηλή επεξεργαστική ισχύ. Το 1998 πρωτοεμφανίστηκαν στη λίστα με τα 500 ισχυρότερα υπολογιστικά συστήματα του κόσμου [TOP]. Από τότε και μετά, ανεβαίνουν συνεχώς στην κλίμακα των επιδόσεων, ενώ πολλαπλασιάζονται τα συστήματα συστοιχιών που περιλαμβάνονται σε αυτή. Την παρούσα χρονική περίοδο, υπάρχουν στον κατάλογο 208 συστοιχίες υπολογιστών, ενώ έξι μήνες πριν υπήρχαν 149. Από τα 10 κορυφαία συστήματα, τα επτά είναι συστοιχίες υπολογιστών. Η ισχυρότερη συστοιχία καταλαμβάνει την 3η θέση στην κατάταξη, με διαφορά στην επίδοση στα καθορισμένα μετροπρογράμματα που χρησιμοποιούνται (Linpack) [Don03], μόλις 26% από το δεύτερο σύστημα.

Από τα παραπάνω στοιχεία γίνεται φανερό ότι οι συστοιχίες υπολογιστών αποτελούν πλέον το βασικότερο περιβάλλον εκτέλεσης παράλληλων εφαρμογών και δίνεται ιδιαίτερη προσοχή στις επιδόσεις τους, οι οποίες καθορίζονται κυρίως από τις ταχύτητες των επεξεργαστών και των δικτύων διασύνδεσης, όσον αφορά το υλικό. Η παρούσα διατριβή, αγγίζει τη συγκεκριμένη περιοχή, καταπιανόμενη με το ζήτημα της βελτιστοποίησης εκτέλεσης παράλληλων εφαρμογών, συνδυάζοντας ιδιαίτερα χαρακτηριστικά των δικτυακών τεχνολογιών και των μοντέρνων πρωτοκόλλων επικοινωνίας.

3.3 Καθυστέρηση Δικτύου, Δυνατότητα Παροχής, Ρυθμός Παροχής

Η αξιολόγηση των επιδόσεων των συστημάτων επικοινωνίας πραγματοποιείται συνήθως μέσω δύο βασικών χαρακτηριστικών τους: της *Καθυστέρησης (Latency)* και της *Δυνατότητας Παροχής (Bandwidth)*. Αν και οι συγκεκριμένοι όροι δεν ορίζονται αυστηρά στη βιβλιογραφία, η Καθυστέρηση είναι το μέτρο που χαρακτηρίζει την ταχύτητα με την οποία συγχρονίζονται δύο διεργασίες μέσω ανταλλαγής μηνυμάτων, ενώ η Δυνατότητα Παροχής χαρακτηρίζει την ταχύτητα με την οποία ολοκληρώνεται μια μεταφορά δεδομένων. Πιο συγκεκριμένα, η Καθυστέρηση L αποτελεί τον αναγκαίο χρόνο που απαιτείται για να φτάσει η μικρότερη δυνατή ποσότητα δεδομένων από έναν κόμβο (αποστολέας) σε ένα δεύτερο κόμβο (παραλήπτης). Η Δυνατότητα Παροχής B αποτελεί το ρυθμό με τον οποίο μεταφέρονται τα δεδομένα από τον ένα κόμβο στον άλλο, αν θεωρήσουμε ότι μεταδίδεται μία πολύ μεγάλη ποσότητα δεδομένων. Για να απομονώσουμε το χρόνο μεταφοράς δεδομένων από τις επιβαρύνσεις που αντιστοιχούν σε αιτίες συγχρονισμού, η Δυνατότητα Παροχής μετριέται με τη μεταφορά πολύ μεγάλης (ιδανικά άπειρης) ποσότητας δεδομένων. Συνεπώς, ισχύουν τα εξής:

$$B = \lim_{D \to \infty} \frac{D}{t},$$

όπου D είναι η ποσότητα δεδομένων που μεταφέρεται σε χρόνο t.

Όμως πέρα από το θεωρητικό μέγιστο της ταχύτητας μεταφοράς δεδομένων κάποιας δικτυακής τεχνολογίας, μας ενδιαφέρει και η πραγματική ταχύτητα μεταφοράς δεδομένων, η οποία επηρεάζεται και από την Καθυστέρηση. Το μέγεθος που μας δίνει τη συγκεκριμένη πληροφορία είναι ο *Ρυθμός Παροχής (Throughput)*. Για το ρυθμό Παροχής T ισχύει:

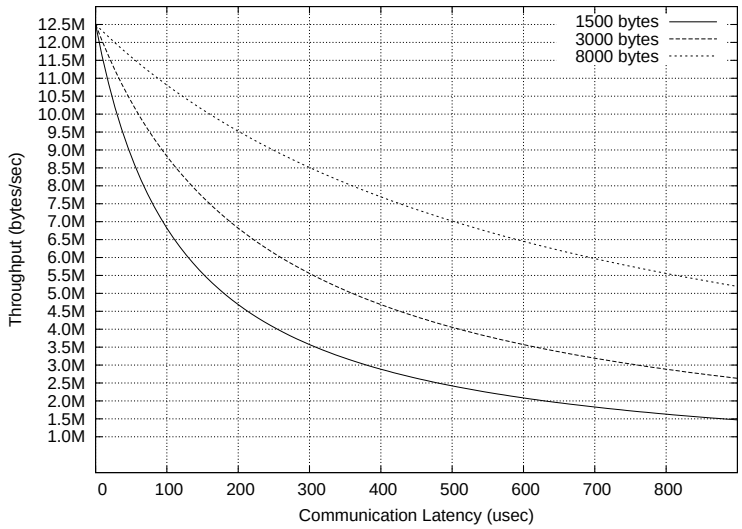
$$T = \frac{D}{t}, \tag{3.1}$$

όπου D είναι μια ποσότητα δεδομένων που μεταφέρεται από τον αποστολέα κόμβο στον παραλήπτη σε χρόνο t. Λόγω της ύπαρξης καθυστέρησης, για το χρόνο ισχύει $t = L + \frac{D}{B}$. Οπότε λόγω της (3.1), ισχύει:

$$T = \frac{1}{\frac{L}{D} + \frac{1}{B}} \tag{3.2}$$

Αν στη σχέση (3.2) θεωρήσουμε ότι $D \to \infty$, τότε ισχύει $T = B$, δηλ. ο *Ρυθμός Παροχής* γίνεται ίσος με την *Δυνατότητα Παροχής*. Στο Σχήμα 3.2 παρουσιάζεται ο Ρυθμός Παροχής δικτύου δεδομένων με Δυνατότητα Παροχής 100 Mbps (12.5 MB/sec), ίση με αυτή του δικτύου FastEthernet, συναρτήσει της Καθυστέρησης του δικτύου για τη μεταφορά διαφορετικών ποσοτήτων

δεδομένων. Όπως φαίνεται, ο Ρυθμός Παροχής ελαττώνεται υπερβολικά καθώς αυξάνεται η Καθυστέρηση του δικτύου. Συγκεκριμένα, για μηδενική (θεωρητική) Καθυστέρηση, ο Ρυθμός Παροχής του δικτύου ισούται με τη δυνατότητα Παροχής του, 12.5 MB/sec. Στην περίπτωση τη μεταφοράς 1500 bytes, για καθυστέρηση ίση με 100 μsec, ο Ρυθμός Παροχής πέφτει στα 6.82 MB/sec, ενώ για Καθυστέρηση 300 μsec πέφτει στα 3.57 MB/sec.



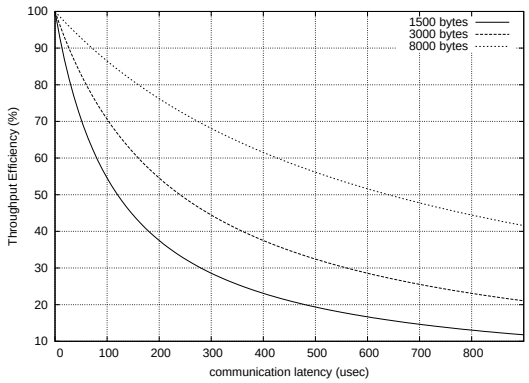
Σχήμα 3.2: Ρυθμός Παροχής δικτύου δεδομένων με Δυνατότητα Παροχής 100 Mbps (12.5 MB/sec) συναρτήσει της Καθυστέρησης του δικτύου.

Η απόδοση του Ρυθμού Παροχής ως προς τη δυνατότητα Παροχής, που ορίζεται ως $e(T) = \frac{T}{B}$, δίνεται από τον τύπο:

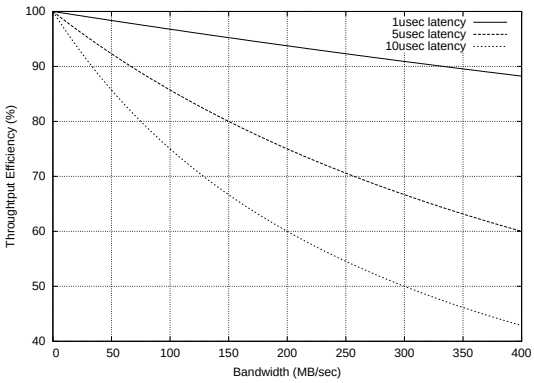
$$e(T) = \frac{1}{1 + \frac{BL}{D}}$$

Η γραφική παράσταση της συγκεκριμένης σχέσης σε συνάρτηση με την Καθυστέρηση L, για 3 διαφορετικές ποσότητες δεδομένων, φαίνεται στο Σχήμα 3.3. Για παράδειγμα, στην περίπτωση της μετάδοσης 3000 bytes, αν η Καθυστέρηση του δικτύου είναι 200 μsec, τότε η απόδοση του Ρυθμού Παροχής ως προς τη Δυνατότητα Παροχής είναι 54.5%. Στο Σχήμα 3.4 απεικονίζεται η γραφική παράσταση της απόδοσης του Ρυθμού Παροχής ως προς τη Δυνατότητα Παροχής σε συνάρτηση της δυνατότητας παροχής, για τη μεταφορά 3000 bytes. Όπως φαίνεται, ακόμη και για Καθυστερήσεις δικτύου της τάξης των μονάδων μsec, όσο αυξάνεται η Δυνατότητα Παροχής, η απόδοσης του Ρυθμού Παροχής ως προς τη Δυνατότητα Παροχής πέφτει σημαντικά. Για παράδειγμα, για Δυνατότητα Παροχής 150 MB/sec, η απόδοση του Ρυθμού Παροχής πέφτει στο 80%.

Από τα παραπάνω προκύπτει ότι, όσο πιο γρήγορη (μεγαλύτερη Δυνατότητα Παροχής δεδομένων) γίνεται μια δικτυακή τεχνολογία, τόσο περισσότερο επηρεάζεται από την Καθυστέρηση



Σχήμα 3.3: Απόδοση Ρυθμού Παροχής ως προς Δυνατότητα Παροχής δικτύου δεδομένων με Δυνατότητα Παροχής 100 Mbps (12.5 MB/sec) συναρτήσει της Καθυστέρησης του δικτύου.



Σχήμα 3.4: Απόδοση Ρυθμού Παροχής ως προς Δυνατότητα Παροχής δικτύου δεδομένων συναρτήσει της Δυνατότητα Παροχής του δικτύου.

που προστίθεται στην επικοινωνία από διάφορους παράγοντες. Τα στοιχεία που προσθέτουν Καθυστέρηση στην επικοινωνία είναι τα εξής (βλ. Σχήμα 3.5):

Υλικό κόμβου Το υλικό του κόμβου που παρεμβάλλεται μεταξύ μνήμης και κάρτας δικτύου. Σε αυτό συμπεριλαμβάνονται η ίδια η μνήμη καθώς και όλοι οι δίαυλοι δεδομένων (buses).

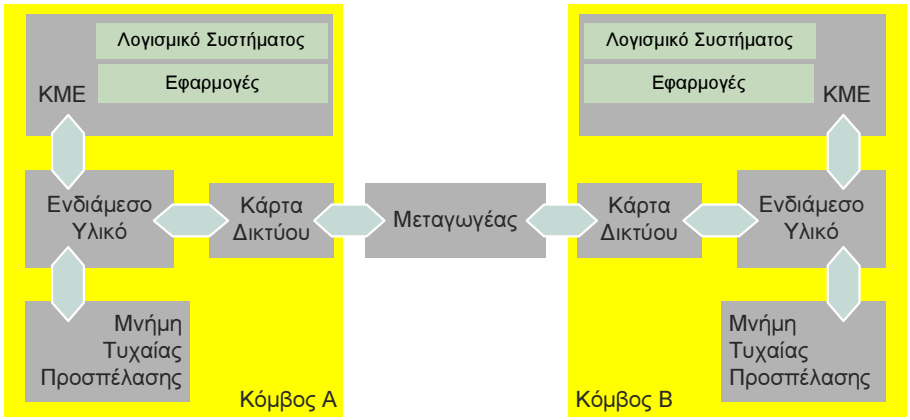
Υλικό δικτύου Το ίδιο το υλικό του δικτύου που αποτελείται από τις κάρτες δικτύου που περιέχονται στους κόμβους και τα υπόλοιπα παρεμβαλλόμενα στοιχεία του δικτύου, όπως μεταγωγείς, δρομολογητές, κλπ.

Λογισμικό συστήματος Το λογισμικό συστήματος (λειτουργικό σύστημα και οδηγοί συσκευών) έχει τη δική του συνεισφορά στη συνολική Καθυστέρηση.

Λογισμικό εφαρμογών Το λογισμικό εφαρμογών είναι το στοιχείο που προσθέτει το μεγαλύτερο ποσοστό στη συνολική Καθυστέρηση. Οι πιο διάσημες εφαρμογές και περιβάλλοντα (λογισμικού) παράλληλης επεξεργασίας είναι ανεπτυγμένα με στόχο τη μεταφερσιμότητα και όχι την υψηλή απόδοση, με αποτέλεσμα να εισάγουν μεγάλο χρόνο καθυστέρησης σε κάθε δικτυακή λειτουργία.

3.4 Λειτουργία ενός Συμβατικού Προσαρμογέα Δικτύου

Το υλικό ενός Προσαρμογέα Δικτύου (Network Interface) χωρίζεται σε 2 κύρια μέρη· το μέρος που συνδέει τον προσαρμογέα στο δίαυλο Ε/Ε του υπολογιστή (στις περιπτώσεις των



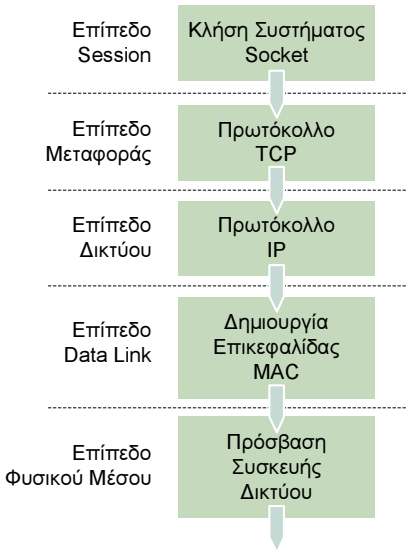
Σχήμα 3.5: Σχεδιάγραμμα 2 διασυνδεδεμένων κόμβων όπου φαίνονται οι διάφοροι παράγοντες που αυξάνουν τη συνολική Καθυστέρηση του δικτύου.

Προσωπικών Υπολογιστών πρόκειται για το δίαυλο Peripheral Component Interconnect , ή αλλιώς PCI) και το μέρος που υλοποιεί τις λειτουργίες επικοινωνίας με άλλους κόμβους. Για να λειτουργήσει ο προσαρμογέας δικτύου, απαιτείται η συνδρομή δικτυακού λογισμικού, το οποίο εκτελείται στον επεξεργαστή του κόμβου και αναλαμβάνει τις αιτήσεις των εφαρμογών και τη διαχείριση του υλικού.

Για λόγους κατηγοριοποίησης και προτυποποίησης του συνόλου των δικτυακών λειτουργιών, ο Διεθνής Οργανισμός Προτύπων (International Standards Organization – ISO) έχει δημιουργήσει το πρότυπο Ανοικτή Διασύνδεση Συστημάτων (Open Systems Interconnection – OSI). Σύμφωνα με το συγκεκριμένο πρότυπο, υπάρχει μια στοίβα στρωμάτων δικτυακών λειτουργιών, κάθε ένα από τα οποία επιτελεί μια ιδιαίτερη λειτουργία προκειμένου να επιτευχθεί η επικοινωνία. Χαρακτηριστικό στοιχείο της στρωμάτωσης είναι ότι κάθε επίπεδο βασίζεται σε μια υπηρεσία που παρέχει το αμέσως πιο χαμηλό επίπεδο. Στους κοινούς προσαρμογείς δικτύου, οι λειτουργίες που επιτελούνται στα επίπεδα του OSI, εκτός από αυτές των επιπέδων 1 και 2, υλοποιούνται από λογισμικό που εκτελείται στον επεξεργαστή του κόμβου.

Τη σημερινή εποχή, οι περισσότεροι σταθμοί εργασίας χρησιμοποιούν δικτυακό κώδικα που βασίζεται στην έκδοση του Λ.Σ. Unix του Πανεπιστημίου Berkeley. Οι εφαρμογές επικοινωνούν πάνω από το δίκτυο χρησιμοποιώντας τη διεπαφή των Sockets (socket interface) και τα Διαδικτυακά πρωτόκολλα TCP/IP ή UDP/IP . Στην Εικόνα 3.6 φαίνονται τα μέρη του λογισμικού οργανωμένα σύμφωνα με το μοντέλο OSI του ISO.

Στη συνέχεια αναλύουμε τις διαδικασίες που λαμβάνουν χώρα κατά την αποστολή και λήψη ενός πακέτου δεδομένων σε ένα συμβατικό περιβάλλον επικοινωνίας. Θεωρούμε ότι πραγματοποιείται επικοινωνία δύο κόμβων σε δίκτυο TCP/IP πάνω απο δικτυακή τεχνολογία Ethernet, χρησιμοποιώντας την διεπαφή των Berkeley sockets.



Σχήμα 3.6: Τα επίπεδα δικτύου σύμφωνα με το μοντέλο Open Systems Interconnection. Δεν φαίνονται τα επίπεδα 6 (Παρουσίαση - Presentation) και 7 (Εφαρμογή - Application).

3.4.1 Τυπική Αποστολή και Λήψη πακέτου

Σε μία αποστολή δεδομένων, το στρώμα των sockets αντιγράφει τα δεδομένα από το χώρο του χρήστη σε προσωρινό χώρο συστήματος (system buffer), όπου λειτουργούν τα πρωτόκολλα μεταφοράς (transport) και δικτύου (network). Στην περίπτωση αξιόπιστης επικοινωνίας χρησιμοποιείται ο συνδυασμός των πρωτοκόλλων μετάδοσης και δικτύου TCP/IP, τα οποία περιλαμβάνουν τη δημιουργία πακέτων, τις διαδικασίες για έλεγχο λαθών, έλεγχο ροής από άκρη-σε-άκρη, δρομολόγηση, και έλεγχο συμφόρησης. Ο συνδυασμός UDP/IP είναι απλούστερος και υλοποιεί μόνο ένα μέρος των προηγούμενων διαδικασιών. Όταν ολοκληρωθούν οι προηγούμενες διαδικασίες, το πακέτο δίνεται στο παρακάτω επίπεδο για να δημιουργηθεί η επικεφαλίδα επιπέδου Media Access Control (MAC) και να μεταδοθεί στη συνέχεια στο δίκτυο.

Μία παρόμοια λειτουργία επιτελείται και στην πλευρά του παραλήπτη, αλλά με την αντίστροφη σειρά. Ειδικότερα, το φυσικό επίπεδο τοποθετεί τα εισερχόμενα πακέτα σε προσωρινούς χώρους του συστήματος που ονομάζονται Προσωρινές Μνήμες Λήψης (receive buffers) και μετά τη διεργασία των πρωτοκόλλων, τα δεδομένα αντιγράφονται στο χώρο του χρήστη από το επίπεδο των sockets, σαν κομμάτι της κλήσης για λήψη της εφαρμογής.

Η διαδικασία για έλεγχο λαθών, που υλοποιείται από τα αξιόπιστα πρωτόκολλα, έχει σημαντικό αντίκτυπο στο πώς η δικτυακή στοίβα χειρίζεται τα δεδομένα. Τα πρωτόκολλα συνήθως χρησιμοποιούν έλεγχο αθροίσματος από άκρη σε άκρη (end-to-end checksum) για να επιβεβαιώσουν την ακεραιότητα των δεδομένων και χρονομετρητών (timeouts) για να ανιχνεύσουν τα χαμένα πακέτα. Η διαδικασία checksumming απαιτεί από το πρωτόκολλο TCP τόσο στην

πλευρά του αποστολέα όσο και στην πλευρά του παραλήπτη να διαβάσουν τα δεδομένα και να υπολογίζουν το checksum. Αν το checksum στην πλευρά του παραλήπτη διαφέρει από αυτό που εισήγαγε ο αποστολέας μέσα στο πακέτο, τότε ο παραλήπτης αγνοεί το πακέτο.

Παρατηρούμε ότι κατά τη διάρκεια της αποστολής του πακέτου από την εφαρμογή του αποστολέα μέχρι τη λήψη του από την αντίστοιχη εφαρμογή του παραλήπτη, λαμβάνουν χώρα ένα σύνολο αντιγραφών και επεξεργασιών των δεδομένων, που αυξάνουν την συνολική καθυστέρηση της επικοινωνίας. Ο χρόνος που διαρκούν συνολικά αυτές οι διαδικασίες, για όλες τις περιπτώσεις επικοινωνίας μιας παράλληλης εφαρμογής, αντιστοιχεί σε ένα σημαντικό τμήμα του συνολικού χρόνου εκτέλεσης. Η κατάσταση γίνεται ακόμη χειρότερη αν αναλογιστεί κανείς ότι όλες τις παραπάνω διαδικασίες πραγματοποιεί η ΚΜΕ. Δηλαδή, αντί της πραγματοποίησης υπολογισμών για την ολοκλήρωση της εκτέλεσης της παράλληλης εφαρμογής, η ΚΜΕ είναι απασχολημένη ένα μεγάλο χρονικό διάστημα με την εκτέλεση των πρωτοκόλλων επικοινωνίας και τις αντιγραφές που αυτά επιβάλλουν. Στη συνέχεια αναλύουμε τις μεταφορές δεδομένων που συμβαίνουν κατά τη διάρκεια της παραπάνω επικοινωνίας.

3.4.2 Μεταφορές Δεδομένων μέσω Διαύλου Συστήματος

Στο Σχήμα 3.7 απεικονίζονται όλες οι μεταφορές δεδομένων που έλαβαν χώρα μέσω του διαύλου επεξεργαστή-μνήμης, για να εκτελεστεί μια αποστολή μηνύματος. Αρχικά, η εφαρμογή δεσμεύει κάποιο κομμάτι μνήμης, στο οποίο γράφει τα δεδομένα προς αποστολή (1). Η μνήμη δεσμεύεται στο χώρο διευθύνσεων της εκτελούμενης διεργασίας. Αυτό μπορεί να γίνει με τον ακόλουθο πηγαίο κώδικα σε γλώσσα C:

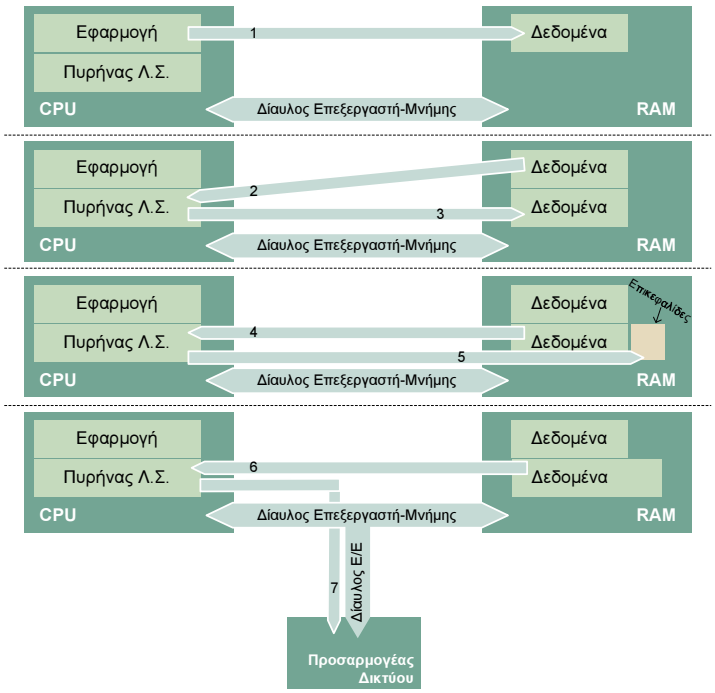
```
#define MSG "This is a message"
char *buffer;
buffer = malloc(sizeof(MSG));
sprintf(buffer, "%s", MSG);
```

Στη συνέχεια, χρησιμοποιώντας την κλήση συστήματος `write`, τα δεδομένα γράφονται σε ένα socket, το οποίο έχει προηγουμένως συνδεθεί με μια διεργασία που εκτελείται σε κάποιο απομακρυσμένο στο δίκτυο κόμβο.

```
write(sd, buffer, strlen(buffer));
```

Αρχικά, ο κώδικας των sockets αντιγράφει τα δεδομένα σε μια προσωρινή μνήμη συστήματος (2, 3). Η αντιγραφή πραγματοποιείται για δύο λόγους: Πρώτον, για να έχει ο πυρήνας του Λ.Σ. τη δυνατότητα να χρησιμοποιήσει τα δεδομένα για τις επόμενες λειτουργίες, χωρίς να δεσμεύει την προσωρινή μνήμη στο χώρο του χρήστη. Δεύτερον, για προστασία του Λ.Σ., αφού ο χρήστης δεν πρέπει να μπορεί να μεταβάλει δεδομένα τα οποία χειρίζεται ο πυρήνας. Αρκεί να φανταστεί κανείς ότι ένας κακόβουλος χρήστης μπορεί να μεταβάλει τις επικεφαλίδες

του δημιουργούμενου πακέτου, στην περίπτωση που αυτό βρίσκεται στο χώρο διευθύνσεων του χρήστη.

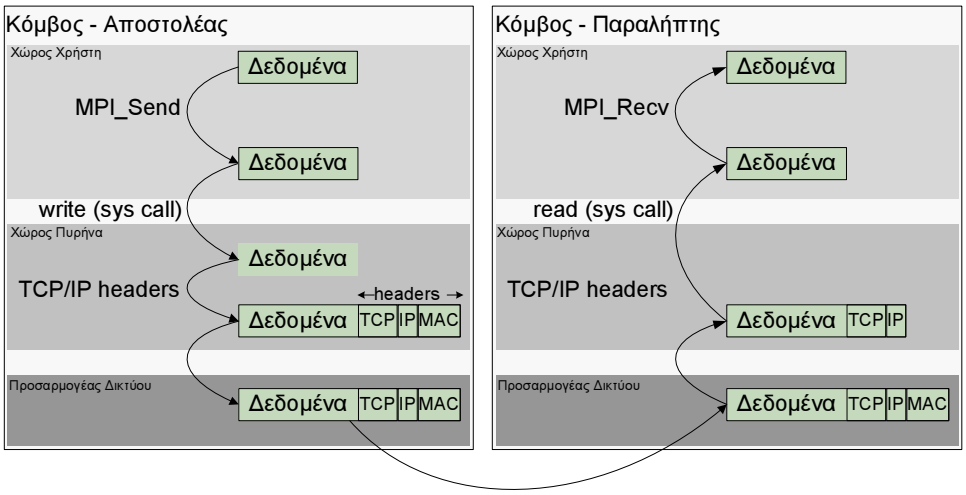


Σχήμα 3.7: Το σύνολο των μεταφορών δεδομένων που λαμβάνουν χώρα στο δίαυλο συστήματος για την αποστολή ενός μηνύματος.

Στη συνέχεια, ο κώδικας που υλοποιεί το συνδυασμό των πρωτοκόλλων TCP/IP, διαβάζει ένα μέρος από τα δεδομένα που βρίσκονται (4) στον buffer συστήματος, τα αντιγράφει σε νέα θέση, ενώ υπολογίζει το TCP checksum και το τοποθετεί μαζί με τις επικεφαλίδες του TCP και του IP (5), στη μνήμη που έπεται των δεδομένων. Το μέρος των δεδομένων που αντιγράφεται, εξαρτάται από το μέγιστο μέγεθος πακέτου (Maximum Transfer Unit – MTU) που υποστηρίζεται από την υφιστάμενη δικτυακή τεχνολογία. Η συγκεκριμένη διαδικασία επαναλαμβάνεται, έως ότου υπολογιστούν οι επικεφαλίδες TCP/IP, για το σύνολο των δεδομένων.

Τέλος, ο επεξεργαστής εκτελεί των κώδικα που υλοποιεί τη λειτουργικότητα του επιπέδου Data Link, ο οποίος διαβάζει τα δεδομένα από την τελική τους θέση στη μνήμη και τα γράφει στη μνήμη του δικτυακού προσαρμογέα, για να μεταδοθούν στο δικτυακό μέσο. Κατά τη διάρκεια αυτής της τυπικής αποστολής πακέτου, τα δεδομένα πέρασαν (διαβάστηκαν ή γράφτηκαν) συνολικά επτά φορές από το δίαυλο του συστήματος, συνυπολογίζοντας και τη δημιουργία των δεδομένων από την εφαρμογή. Ο χρόνος που απαιτείται για την επικοινωνία μπορεί να είναι στην πραγματικότητα μικρότερος εξαιτίας της επίδρασης της ιεραρχίας μνήμης. Πιο συγκεκριμένα, ορισμένες προσβάσεις στην κεντρική μνήμη μπορούν να αποφευχθούν υποθέτοντας ότι ένα

τμήμα των δεδομένων βρίσκεται στην κρυφή μνήμη του επεξεργαστή. Σε κάθε περίπτωση όμως, οι συνεχείς μεταφορές δεδομένων αποτελούν σημαντικό παράγοντα καθυστέρησης της συνολικής επικοινωνίας. Στο Σχήμα 3.8, φαίνονται οι αντιγραφές που λαμβάνουν χώρα σε μια τυπική αποστολή και λήψη μηνύματος στο προαναφερθέν σύστημα, χρησιμοποιώντας πάνω από αυτό τη βιβλιοθήκη ανταλλαγής μηνυμάτων MPI.



Σχήμα 3.8: Το σύνολο των αντιγραφών που λαμβάνουν χώρα κατά τη διάρκεια της αποστολής και λήψης ενός μηνύματος πάνω από MPI. Η αντιγραφή των δεδομένων από το χώρο Πυρήνα στον Προσαρμογέα Δικτύου, γίνεται είτε μέσω της ΚΜΕ, είτε μέσω μηχανής Άμεσης Προσπέλασης Μνήμης που βρίσκεται στον προσαρμογέα δικτύου.

Στο τέλος της παραγράφου 3.3, αναφέρθηκαν οι παράγοντες που επιβαρύνουν την καθυστέρηση επικοινωνίας σε μια συστοιχία υπολογιστών. Στο προηγούμενο παράδειγμα της αποστολής μηνύματος έχουμε την ακόλουθη κατάταξη:

Υλικό δικτύου Σε αυτό ανήκει ο προσαρμογέας δικτύου.

Υλικό κόμβου Εδώ ανήκουν η ΚΜΕ, το υποσύστημα μνήμης, οι δίαυλοι συστήματος και Ε/Ε.

Λογισμικό συστήματος Εδώ ανήκει όλο το λογισμικό του πυρήνα που υλοποιεί τις κλήσεις συστήματος, το υποσύστημα των sockets, καθώς και η υλοποίηση των πρωτοκόλλων TCP και IP.

Λογισμικό εφαρμογών Εδώ ανήκουν το λογισμικό της εφαρμογής, η βιβλιοθήκη που υλοποιεί το API για τον προγραμματισμό των sockets και η βιβλιοθήκη της γλώσσας C που περιέχει το σύνολο των συναρτήσεων της γλώσσας.

Είναι προφανές ότι όσο περισσότερες λειτουργίες πραγματοποιούνται σε κάθε αποστολή δεδομένων, τόσο μεγαλώνει η καθυστέρηση της επικοινωνία με τους υπόλοιπους κόμβους σε ένα

περιβάλλον συστοιχίας υπολογιστών. Στη συνέχεια, παρουσιάζονται τρόποι μέσω των οποίων είναι δυνατόν να μειωθεί η συνολική καθυστέρηση επικοινωνίας.

3.4.3 Αύξηση Καθυστέρησης λόγω Αύξησης Δυνατότητας Παροχής

Ενδιαφέρον παρουσιάζει η περίπτωση της δικτυακής τεχνολογίας Gigabit Ethernet (GbE), η οποία αποτελεί αναβάθμιση της τεχνολογίας Fast Ethernet. Η δυνατότητα παροχής του GbE δεκαπλασιάστηκε από αυτή του Fast Ethernet φτάνοντας το 1 Gb/sec. Το μέγεθος των δικτυακών πακέτων στην περίπτωση του GbE παρέμεινε το ίδιο (1500 bytes). Αυτό πρακτικά σημαίνει ότι, λόγω της μεγαλύτερης δυνατότητας παροχής, από έναν κόμβο λαμβάνονται περισσότερα πακέτα και κατά συνέπεια πραγματοποιούνται περισσότερες διακοπές (interrupts) στη μονάδα του χρόνου. Θέλοντας να διατηρηθεί ο ρυθμός παροχής σε υψηλά επίπεδα, χωρίς την συνεχή ενόχληση της ΚΜΕ, οι διακοπές μαζεύονται και εξυπηρετούνται ανά ομάδες (interrupt coalescing). Ο τρόπος αυτός έχει ως αποτέλεσμα την σημαντική αύξηση της καθυστέρηση επικοινωνίας, αφού για να φτάσει ένα μήνυμα που βρίσκεται στον προσαρμογέα δικτύου μέχρι την εφαρμογή, πρέπει να περιμένει να μαζευτεί ένα ικανό πλήθος μηνυμάτων (ή να περάσει κάποιο χρονικό διάστημα timeout). Η αύξηση της καθυστέρησης επηρεάζει πολύ το χρόνο εκτέλεσης των παραλλήλων εφαρμογών. Γίνεται σαφές ότι, παρότι η τεχνολογία επιτρέπει την επίτευξη μικρότερων χρόνων επικοινωνίας, για να γίνουν οι εφαρμογές άμεσοι αποδέκτες των βελτιώσεων, θα πρέπει να χρησιμοποιηθούν νέα πρωτόκολλα που να μειώνουν το πλήθος και τη διάρκεια των ενδιάμεσων διαδικασιών.

3.5 Πρωτόκολλα Επικοινωνίας Επιπέδου Χρήστη για Αποδοτική Επικοινωνία

Τα μοντέρνα τοπικά δίκτυα υψηλών ταχυτήτων προσφέρουν αρκετές δυνατότητες στις εφαρμογές που τα χρησιμοποιούν, αλλά η απόδοσή τους περιορίζεται από τη χρήση των παραδοσιακών πρωτοκόλλων επικοινωνίας, όπως το TCP/IP. Ο στόχος της αποδοτικής επικοινωνίας είναι, κάποια δεδομένα που βρίσκονται στο χώρο διευθύνσεων μιας διεργασίας να μεταφερθούν, όσο το δυνατόν ταχύτερα, στο χώρο διευθύνσεων μιας άλλης διεργασίας σε ένα απομακρυσμένο κόμβο. Στις περισσότερες περιπτώσεις, τα συμβατικά πρωτόκολλα απαιτούν όλες οι προσβάσεις στους δικτυακούς προσαρμογείς να γίνονται μέσα από τον πυρήνα του λειτουργικού συστήματος, πραγματοποιώντας χρονοβόρες αντιγραφές των δεδομένων. Οι διαδικασίες αυτές προσθέτουν σημαντική καθυστέρηση, τόσο κατά την αποστολή, όσο και κατά τη λήψη ενός μηνύματος.

Για την αντιμετώπιση της καθυστέρησης αυτής, έχουν αναπτυχθεί αρχιτεκτονικές επικοινωνίας επιπέδου χρήστη, οι οποίες αφαιρούν το κομμάτι του λειτουργικού συστήματος από το

κρίσιμο μονοπάτι της επικοινωνίας, καθώς και τις περισσότερες αντιγραφές. Όμως, το Λ.Σ. βρίσκεται στο κρίσιμο μονοπάτι της επικοινωνίας προκειμένου να παρέχει κάποια λειτουργικότητα μέσω ειδικών μηχανισμών, που είναι οι εξής:

- Μηχανισμός Μεταφοράς Δεδομένων
- Μετάφραση Διευθύνσεων
- Προστασία
- Μηχανισμός Μεταφοράς Ελέγχου

Στη συνέχεια αναλύουμε τους συγκεκριμένους μηχανισμούς για να κατανοήσουμε την λειτουργικότητα που παρέχει το λειτουργικό σύστημα στην διαδικασία της επικοινωνίας.

3.5.1 Μηχανισμός Μεταφοράς Δεδομένων

Για να επιτευχθεί η επικοινωνία, θα πρέπει δεδομένα που βρίσκονται στην ιεραρχία μνήμης του υπολογιστή να μεταφερθούν στο δικτυακό προσαρμογέα και στη συνέχεια στο δικτυακό παραλήπτη. Η μεταφορά είναι δυνατόν να πραγματοποιηθεί με δύο διαφορετικούς τρόπους· με Προγραμματιζόμενη Ε/Ε και με Άμεση Προσπέλαση Μνήμης (Direct Memory Access - DMA).

Σύμφωνα με την Προγραμματιζόμενη Ε/Ε , ο επεξεργαστής είναι υπεύθυνος για τη συνολική μεταφορά των δεδομένων, διαβάζοντας από την κεντρική μνήμη τα προς αποστολή δεδομένα και γράφοντάς τα στο δικτυακό προσαρμογέα. Η ανάγνωση πραγματοποιείται με εντολές load, ενώ η εγγραφή πραγματοποιείται είτε με εντολές out, είτε με εντολές store. Στη δεύτερη περίπτωση, ο προσαρμογέας θα πρέπει να έχει απεικονιστεί σε χώρο διευθύνσεων (memory-mapped I/O).

Σε κάθε περίπτωση, η επικοινωνία με Προγραμματιζόμενη Ε/Ε δεν αποτελεί ελκυστική προσέγγιση, μιας και κατά τη διάρκεια της αντιγραφής των δεδομένων από την ιεραρχία μνήμης στον προσαρμογέα, ο επεξεργαστής είναι συνεχώς απασχολημένος. Την κατάσταση επιβαρύνει το γεγονός ότι κάθε εγγραφή στο δικτυακό προσαρμογέα απαιτεί την προσπέλαση του διαύλου Ε/Ε, ο οποίος είναι σημαντικά πιο αργός από το δίαυλο επεξεργαστή-μνήμης. Επίσης, η διαδικασία της αντιγραφής πραγματοποιείται ανά μία λέξη, για το σύνολο των λέξεων του αποστελλόμενου μηνύματος. Στις μηχανές που μας απασχολούν, κάθε λέξη έχει μέγεθος 32 ή 64 bytes και κάθε μήνυμα μπορεί να έχει μέγεθος από μερικές δεκάδες μέχρι εκατοντάδες χιλιάδες bytes. Αν ληφθεί υπόψη το πλήθος των μηνυμάτων που αποστέλλονται μέσω δικτύου, συμπεραίνεται ότι στην περίπτωση της Προγραμματιζόμενης Ε/Ε, η ΚΜΕ καταναλώνει ένα σημαντικό κομμάτι του χρόνου της αντιγράφοντας δεδομένα.

Ανάλογα με την αρχιτεκτονική της ΚΜΕ και του συνολικού συστήματος, έχουν προταθεί και υλοποιηθεί τεχνικές οι οποίες βελτιώνουν της απόδοση της μεθόδου της Προγραμματιζόμενης Ε/Ε. Η πιο γνωστή από αυτές ονομάζεται Συνδυασμός Εγγραφών (Write Combining) [Int98]. Σύμφωνα με αυτή, αντί η ΚΜΕ να αντιγράφει ανά μία τις λέξεις από την ιεραρχία της μνήμης στον προσαρμογέα, τις συγκεντρώνει σε μια ειδική μνήμη μέσα στο κύκλωμα του επεξεργαστή, που ονομάζεται Προσωρινή Μνήμη Συνδυασμού Εγγραφών (Write Combining Buffer). Όταν συγκεντρωθεί μία ικανή ποσότητα δεδομένων (π.χ. 64 bytes) ή περάσει κάποιο χρονικό διάστημα (timeout), οι λέξεις αποστέλλονται μαζικά στον προσαρμογέα χρησιμοποιώντας ειδικές εντολές burst, οι οποίες είναι βελτιστοποιημένες για μαζικές μεταφορές δεδομένων και υποστηρίζονται από το δίαυλο Ε/Ε. Η ίδια ιδέα χρησιμοποιείται και στην περίπτωση των εγγραφών της ΚΜΕ προς την κεντρική μνήμη του συστήματος, κατά τη διάρκεια της οποίας οι εγγραφές συλλέγονται στην Προσωρινή Μνήμη Εγγραφών (Write Buffer) και στη συνέχεια αποστέλλονται μαζικά.

Η συγκεκριμένη διαδικασία είναι αντίστοιχη της αύξησης του Ρυθμού Μετάδοσης με την αύξηση του μεγέθους του μηνύματος. Αν και ο ρυθμός μετάδοσης αυξάνεται σημαντικά, το γεγονός της απαίτησης της συνδρομής της ΚΜΕ για τη μεταφορά ολόκληρης της ποσότητας των δεδομένων, για κάθε μήνυμα, καθώς και η αδυναμία χρήσης της μεθόδου κατά τη λήψη δεδομένων, καθιστά την Προγραμματιζόμενη Ε/Ε κατάλληλη μόνο για μικρού μεγέθους και συχνότητας μεταφορές.

Αντιθέτως, σύμφωνα με τη μέθοδο της Άμεσης Προσπέλασης Μνήμης (ΑΠΜ), η ΚΜΕ απλώς προγραμματίζει την ειδική μηχανή ΑΠΜ που βρίσκεται ενσωματωμένη στο δικτυακό προσαρμογέα, η οποία στη συνέχεια αναλαμβάνει τη μεταφορά των δεδομένων. Συνεπώς, όσον αφορά την ΚΜΕ, η μεταφορά μέσω ΑΠΜ κοστίζει όσο ο προγραμματισμός της μηχανής ΑΠΜ. Κατά τη διάρκεια όλης της μεταφοράς, η ΚΜΕ είναι διαθέσιμη για εκτέλεση άλλων λειτουργιών.

3.5.2 Μετάφραση Διευθύνσεων

Η μεταφορά δεδομένων με ΑΠΜ, που αναφέρθηκε προηγουμένως, έχει κάποια ιδιαιτερότητα όταν δεν πραγματοποιείται μέσω του Λ.Σ. Συγκεκριμένα, ο χώρος διευθύνσεων που γνωρίζει η εφαρμογή είναι ο γραμμικός χώρος των εικονικών διευθύνσεων. Για παράδειγμα, σύμφωνα με το μοντέλο της εικονικής μνήμης που υιοθετούν υα περισσότερα σύγχρονα Λ.Σ., δύο εφαρμογές είναι δυνατό να έχουν πρόσβαση στην ίδια εικονική διεύθυνση. Όμως, τα δεδομένα βρίσκονται φυσικώς σε διαφορετική διεύθυνση της κεντρικής μνήμης του υπολογιστή. Τη συγκεκριμένη διαδικασία μετάφρασης εικονικών διευθύνσεων σε φυσικές αναλαμβάνει η μονάδα διαχείρισης μνήμης της ΚΜΕ, με τη βοήθεια του Λ.Σ.

Για τη μεταφορά δεδομένων, οι μηχανές ΑΠΜ των δικτυακών προσαρμογέων χρειάζονται τη φυσική διεύθυνση των δεδομένων στη μνήμη του υπολογιστή. Όμως τη μετάφραση από εικονική σε φυσική τη γνωρίζει το Λ.Σ. και όχι η εφαρμογή. Υπάρχουν τρόποι για τη μετάφραση

των διευθύνσεων στο χώρο του χρήστη, ή στην αυτόματη μετάφραση μέσω ειδικού υλικού στο δικτυακό προσαρμογέα (NIC MMU).

3.5.3 Προστασία

Το νόημα της επικοινωνίας επιπέδου χρήστη είναι να μπορούν οι εφαρμογές να κάνουν άμεση χρήση των υπηρεσιών που προσφέρουν οι δικτυακοί προσαρμογείς, χωρίς να απαιτείται η μεσολάβηση του Λ.Σ. Ωστόσο, η παροχή ανεξέλεγκτης πρόσβασης στο δικτυακό προσαρμογέα επιτρέπει την ανάγνωση, ή ακόμη χειρότερα τη μεταβολή των δεδομένων που βρίσκονται σε αυτόν, από όλες τις διεργασίες του κόμβου. Τον έλεγχο πρόσβασης πραγματοποιούσε ο πυρήνας του Λ.Σ. Θεωρώντας ότι το Λ.Σ. δεν παρεμβάλλεται στη διαδικασία πρόσβασης, η προστασία των δεδομένων των διεργασιών μπορεί να επιτευχθεί με εναλλακτικούς τρόπους.

Ο πιο εύκολος, αλλά καθόλου πρακτικός τρόπος, είναι να επιτρέπεται η επικοινωνία μόνο με μία διεργασία κάθε φορά. Αυτό μπορεί να επιτευχθεί με διάφορες μεθόδους, αλλά το αποτέλεσμα δεν είναι, ούτως ή άλλως, επιθυμητό σε ένα πολυδιεργασιακό περιβάλλον παράλληλης επεξεργασίας.

3.5.4 Μηχανισμός Μεταφοράς Ελέγχου

Με τον όρο «μηχανισμός Μεταφοράς Ελέγχου» εννοούμε τον τρόπο με τον οποίο ειδοποιείται η εκάστοτε διεργασία ότι έχει φτάσει κάποιο μήνυμα για αυτήν. Δύο είναι οι κύριοι τρόποι που υπάρχουν για το συγκεκριμένο σκοπό. Η διαδικασία της δειγματοληψίας (polling) και οι διακοπές (interrupts). Οι διακοπές, που χρησιμοποιούνται από την πλειοψηφία των σημερινών πρωτοκόλλων επικοινωνίας, πραγματοποιούνται μόνο μέσω του Λ.Σ.

Σύμφωνα με τη δειγματοληψία, η διεργασία που επιθυμεί να ενημερωθεί αν υπάρχει κάποιο μήνυμα για αυτή, πρέπει να διαβάζει συνεχώς κάποια συγκεκριμένη θέση μνήμης στο δικτυακό προσαρμογέα. Ανάλογα με το περιεχόμενο της μνήμης, σημαίνει ότι ήρθε ή όχι κάποιο μήνυμα. Οι πολύ συχνές αναγνώσεις της συγκεκριμένης μνήμης, μειώνουν την καθυστέρηση της επικοινωνίας, αλλά αυξάνουν τον «χαμένο» χρόνο, κατά τη διάρκεια του οποίου η διεργασία απλώς περιμένει να έρθει κάποιο μήνυμα. Από την άλλη, οι πολύ χρονικά αραιές αναγνώσεις αυξάνουν τον «χρήσιμο» χρόνο της διεργασίας, ενώ έχουν τον κίνδυνο της απώλειας μηνυμάτων, αφού δύο μηνύματα μπορεί να φτάσουν πολύ κοντά το ένα με το άλλο, με αποτέλεσμα το δεύτερο να διαγράψει το πρώτο.

3.6 Συγκεκριμένες Υλοποιήσεις

Υπάρχει ένα μεγάλο κομμάτι βιβλιογραφίας που ασχολείται με το πώς και πού πραγματοποιείται η μετάφραση διευθύνσεων σε ένα υποσύστημα επικοινωνίας. Τα τέσσερα σημαντικότερα

ζητήματα είναι τα εξής: Πρώτον, πώς πραγματοποιούνται (initiated) οι αιτήσεις για μεταφορά δεδομένων από την εφαρμογή. Δεύτερον, πώς διατηρούνται συνεπείς οι μεταφράσεις στον κυρίως επεξεργαστή με αυτές του δικτυακού προσαρμογέα. Τρίτον, πώς αντικαθίστανται οι εγγραφές μετάφρασης στο δικτυακό προσαρμογέα. Και τέταρτον, πώς αντιμετωπίζεται το ζήτημα της προστασίας σε ένα πολυδιεργασιακό περιβάλλον.

3.6.1 Αγκιστρωμένοι (Pinned-down) Buffers

Αρχικές υλοποιήσεις υποσυστημάτων επικοινωνίας χρησιμοποιούσαν ειδικούς, αγκιστρωμένους, buffers αποστολής και λήψης, μοναδικούς για ολόκληρο το σύστημα. Κάθε τέτοιος buffer είναι συνεχής στη φυσική μνήμη του υπολογιστή έτσι ώστε ο δικτυακός προσαρμογέας να χρειάζεται να ξέρει μόνο την αρχική φυσική του διεύθυνση, καθώς και το μέγεθός του σε bytes για μια μεταφορά δεδομένων. Για την αποστολή ενός μηνύματος η εφαρμογή χρησιμοποιεί το λειτουργικό σύστημα για να αρχίσει την αποστολή. Το λειτουργικό σύστημα αντιγράφει τα δεδομένα στον buffer αποστολής του συστήματος. Ο δικτυακός προσαρμογέας αποστέλλει το μήνυμα από το συγκεκριμένο buffer. Στην άφιξη του μηνύματος, ο δικτυακός προσαρμογέας χρησιμοποιώντας απευθείας προσπέλαση μνήμης (DMA) τοποθετεί τα δεδομένα στον buffer συστήματος. Από εκεί το λειτουργικό σύστημα αντιγράφει τα δεδομένα στον buffer της εφαρμογής. Η μετάφραση διευθύνσεων πραγματοποιείται σε δύο σημεία: όταν αντιγράφονται τα δεδομένα από και προς τον buffer της εφαρμογής, και όταν ο προσαρμογέας δικτύου προσπελαύνει τον buffer του πυρήνα.

3.6.2 Λίστες Scatter/Gather

Μια μέθοδος για να αρθεί ο περιορισμός της χρήσης ενός μεγάλου και συνεχόμενου τμήματος φυσικής μνήμης ως buffer συστήματος είναι η χρήση ενός πίνακα ή μιας αλυσίδας από περιγραφητές (descriptors). Κάθε περιγραφητής περιέχει τη μετάφραση διεύθυνσης για ένα μικρό συνεχόμενο τμήμα του buffer συστήματος. Ο πίνακας περιγραφητών είναι αποθηκευμένος σαν μια συνδεδεμένη λίστα στον προσαρμογέα δικτύου. Η προσέγγιση αυτή ονομάζεται αλλιώς και λίστα scatter/gather. Αυτή επιτρέπει στο λειτουργικό σύστημα να ξεκινήσει μεταφορές δεδομένων από οποιοδήποτε σημείο της φυσικής μνήμης. Επιπλέον, για να αποφευχθούν οι αντιγραφές από και προς τον buffer της εφαρμογής, το λειτουργικό σύστημα μπορεί να αγκιστρώσει τον buffer και να αποθηκεύσει τις μεταφρασμένες διευθύνσεις του στη λίστα scatter/gather. Για να δημιουργηθούν οι περιγραφητές εντός του χώρου του λειτουργικού συστήματος απαιτούνται κλήσεις συστήματος. Το Autonet [SBB+91], για παράδειγμα, χρησιμοποιεί μια συνδεδεμένη λίστα περιγραφητών και το VAXClusters [KLS86] χρησιμοποιεί πίνακα περιγραφητών.

3.6.3 Επαναπεικόνιση Σελίδων

Η επαναπεικόνιση σελίδων (page remmaping) είναι μια μέθοδος για να αποφεύγονται οι αντιγραφές [LLD+83, DP93, JZ93, BS96]. Όταν οι μεταφορές είναι στοιχισμένες (aligned) και έχουν το «σωστό» μέγεθος (π.χ. πολλαπλάσιο του μεγέθους σελίδας), το λειτουργικό σύστημα αντιμεταθέτει τις απεικονίσεις εικονική-προς-φυσική μεταξύ των σελίδων του buffer πυρήνα και του buffer της εφαρμογής. Με τη συγκεκριμένη μέθοδο επιτυγχάνεται και zero-copy. Όμως, η επαναπεικόνιση σελίδων προκαλεί σημαντική επιβάρυνση λόγω της εμπλοκής του λειτουργικού συστήματος, της επεξεργασίας των διακοπών, και του context switch.

3.6.4 Επεξεργαστές Επικοινωνίας

Μερικά συστήματα χρησιμοποιούν έναν επεξεργαστή επικοινωνίας που τρέχει σαν κομμάτι του λειτουργικού συστήματος για να προσπελάσει ή να αποθηκεύσει τον πίνακα σελίδων του λειτουργικού συστήματος ώστε να μεταφράσει τις εικονικές διευθύνσεις των buffer της εφαρμογής και να ξεκινήσει απευθείας προσπελάσεις της μνήμης στον προσαρμογέα δικτύου. Το Intel Paragon [PR94] αφιερώνει έναν επεξεργαστή που βρίσκεται στο δίαυλο μνήμης με συνάφεια κρυφής μνήμης για αυτόν το σκοπό. Ο επεξεργαστής αυτός έχει τη δυνατότητα να ελέγχει την αγκίστρωση και την ανταλλαγή, που εξαλείφει την ανάγκη για κλήσεις συστήματος και διακοπές, με κόστος την απομάκρυνση του συγκεκριμένου επεξεργαστή από χρήσιμη εκτέλεση πράξεων.

3.6.5 Επεξεργαστές Πρωτοκόλλου

Μια άλλη προσέγγιση είναι η χρήση επεξεργαστών πρωτοκόλλου για την αντιμετώπιση της μετάφρασης των διευθύνσεων και της μεταφοράς δεδομένων. Το Meiko CS-2 [HM93] και το Typhoon [RLW94] χρησιμοποιούν επεξεργαστή πρωτοκόλλου. Ο πολυεπεξεργαστής Stanford FLASH [KOH+94] χρησιμοποιεί έναν προγραμματιζόμενο επεξεργαστή για την ολοκλήρωση ενός ελεγκτή μνήμης, ενός ελεγκτή Ε/Ε, ενός δικτυακού προσαρμογέα, και ενός επεξεργαστή πρωτοκόλλου.

3.6.6 Εικονικός Προσαρμογέας Δικτύου

Μερικά υποσυστήματα επικοινωνίας μεταφέρουν δεδομένα κατευθείαν ανάμεσα στους buffers της εφαρμογής και του προσαρμογέα δικτύου [DWB+93, ST93, PLC95]. Με αυτή την προσέγγιση, η εφαρμογή είναι υπεύθυνη για τη μετάφραση διευθύνσεων ή για την πραγματοποίηση λειτουργιών προγραμματισμένης Ε/Ε για να προσπελάσει το δίκτυο. Όμως αυτή η προσέγγιση δεν είναι σχεδιασμένη για πολυδιεργασιακά περιβάλλοντα.

Μια βελτίωση αυτής της προσέγγισης είναι ο εικονικός προσαρμογέας δικτύου. Η βασική μέθοδος είναι να παρέχει μια έννοια (abstraction) εικονικής πόρτας επικοινωνίας μέσω της οποίας μια διεργασία μπορεί να εκδίδει (issue) αιτήσεις στο δικτυακό προσαρμογέα, παρακάμπτοντας το Λ.Σ. Παραδείγματα τέτοιων συστημάτων περιλαμβάνουν τα Application Device Channels (ADC) [DPD94], Hamlyn [BJM+96], U-Net [EBBV95], και την Virtual Interface Architecture (VIA) [CIM97]. Όλα απαιτούν από την εφαρμογή να αγκιστρώσει ρητώς τους buffers και να εγκαταστήσει τους περιγραφητές στον προσαρμογέα δικτύου. Οι περιγραφητές περιέχουν μεταφράσεις και για τις πλευρές των αποστολών και για των λήψεων. Η προστασία αντιμετωπίζεται με τη χρήση ενός κλειδιού.

3.6.7 Επικοινωνία μέσω Απεικόνισης Μνήμης

Η επικοινωνία με απεικόνιση μνήμης (memory-mapped) πραγματοποιεί μια απευθείας προσέγγιση. Το PRAM [LS88], το SHRIMP [BLA+94] και το Memory Channel [GCP96] υλοποιούν δομοστοιχεία (modules) επικοινωνίας memory-mapped [Spe82] που επιτρέπουν στις εφαρμογές να στείλουν μηνύματα σε απομακρυσμένη μνήμη. Το PRAM υλοποιεί ένα μοντέλο απεικόνισης στη φυσική μνήμη που επιτρέπει σε μια εφαρμογή να απεικονίσει τη δυναμική μνήμη του δικτυακού προσαρμογέα στο δικό της χώρο διευθύνσεων. Οι εγγραφές σε αυτή τη μνήμη μεταδίδονται στη μνήμη του απομακρυσμένου προσαρμογέα δικτύου. Είναι ευθύνη της εφαρμογής η μεταφορά των δεδομένων ανάμεσα στη μνήμη του προσαρμογέα δικτύου και των δικών της buffers. Η προσέγγιση του SHRIMP [BLA+94, BDFL96, BAC+98] υλοποιεί προστατευόμενη επικοινωνία σε επίπεδο χρήστη χρησιμοποιώντας το μοντέλο Virtual Memory-Mapped Communication (VMMC). Με τον τρόπο αυτό, μια εφαρμογή μπορεί να στείλει δεδομένα από την εικονική της μνήμη, απευθείας στην εικονική μνήμη μιας απομακρυσμένης διεργασίας, σε ένα πολυδιεργασιακό περιβάλλον. Σε αυτήν την προσέγγιση, οι παραλήπτες πρέπει να αγκιστρώνουν τους buffers λήψης πριν ξεκινήσει η μεταφορά. Το λειτουργικό σύστημα μεταφράζει τις εικονικές διευθύνσεις και αποθηκεύει τις φυσικές στον προσαρμογέα δικτύου. Η υλοποίηση του SHRIMP χρησιμοποιεί ένα τροποποιημένο λειτουργικό σύστημα για την αυτόματη αγκίστρωση των buffers αποστολής των εφαρμογών. Ένας μηχανισμός Απευθείας Προσπέλασης Μνήμης σε Επίπεδο Χρήστη χρησιμοποιείται για να επιτρέψει στον προσαρμογέα δικτύου να αποκτήσει τις μεταφράσεις από εικονική-προς-φυσική χωρίς την παρέμβαση του Λ.Σ. Το SHRIMP παρέχει επίσης και αυτόματες ενημερώσεις με τις οποίες οι αλλαγές στους buffers αποστολής διαδίδονται στους απομακρυσμένους buffers εικονικής μνήμης. Το Memory Channel [GCP96] της Digital συνδυάζει μια προσέγγιση όμοια με του PRAM στην πλευρά του αποστολέα με την αυτόματη ενημέρωση του SHRIMP στην πλευρά του παραλήπτη.

Μια άλλη άμεση προσέγγιση επιτρέπει στις εφαρμογές να συντάσσουν και να δέχονται μηνύματα χρησιμοποιώντας καταχωρητές προσαρμογέα δικτύου [HJ92, Sco96]. Ο Cray T3E

[Sco96] υποστηρίζει προσπέλαση απομακρυσμένης μνήμης με μια προσέγγιση όμοια με την Απευθείας Προσπέλαση Μνήμης σε Επίπεδο Χρήστη (UDMA). Χρησιμοποιεί πλήρεις πίνακες σελίδων για να περιγράψει καθολικά τμήματα (μνήμης) επικοινωνίας και όλες οι σελίδες επικοινωνίας είναι αγκιστρωμένες στη μνήμη.

Μια άλλη άμεση προσέγγιση μπορεί να κρατήσει μόνο περιορισμένο αριθμό καταχωρήσεων μετάφρασης, που θέτει το ερώτημα πώς θα διατηρηθεί ο συνέπεια μεταξύ των μεταφράσεων του κυρίως συστήματος και του συστήματος επικοινωνίας και πώς θα αντιμετωπιστούν οι περιπτώσεις που ο προσαρμογέας δικτύου δεν έχει την πληροφορία για να μεταφράσει κάποια διεύθυνση. Μια προσέγγιση είναι να επιτρέψουμε στον προσαρμογέα να διακόψει τον επεξεργαστή στην περίπτωση αδυναμίας μετάφρασης, οπότε επεξεργαστής θα τοποθετήσει τη ζητούμενη καταχώρηση στον προσαρμογέα. Το VMMC [DBLP97] για μια συστοιχία προσωπικών υπολογιστών διασυνδεδεμένη με Myrinet εφαρμόζει αυτή την προσέγγιση. Χρησιμοποιεί ένα πίνακα μετάφρασης ανά διεργασία στον προσαρμογέα δικτύου.

Το UNet-MM [WBE97], μια επέκταση του U-Net [EBBV95], αποθηκεύει μεταφράσεις διευθύνσεων σε μια cache μεταφράσεων στο δικτυακό προσαρμογέα. Αποτυχίες στη μετάφραση αντιμετωπίζονται από το Λ.Σ. το οποίο αγκιστρώνει τις ζητούμενες σελίδες και εγκαθιστά τις μεταφράσεις του στον προσαρμογέα δικτύου.

Τα νέα δίκτυα υψηλών ταχυτήτων παρέχουν αυξημένες επιδόσεις που δεν μπορούν να αξιοποιηθούν από την αργή εξέλιξη του λογισμικού και οι παλιές στοίβες πρωτοκόλλων δεν είναι αρκετές γι' αυτού του επιπέδου τις ταχύτητες. Καθώς αυξάνεται το εύρος επικοινωνίας, η καθυστέρηση θα πρέπει να μειώνεται ώστε το σύστημα να κρατείται σε ισορροπία. Με την υπάρχουσα δικτυακή τεχνολογία, η στενωπός (bottleneck) είναι συνήθως το λογισμικό που μεσολαβεί [TL93, KC94] ανάμεσα στο υλικό και το χρήστη. Για το λόγο αυτό αναπτύχθηκαν νέα πρωτόκολλα μετάδοσης που στοχεύουν στην παράλληλη επεξεργασία και έχουν ως βασικό στόχο να φτάσει το μέγιστο της απόδοσης του υλικού στο χρήστη [BRB98, ABD+98].

3.6.8 Active Messages

Στον παραπάνω σκοπό στόχευσε το έργο Active Messages (AM) [MC95, Eic93] του Πανεπιστημίου Berkeley της California. Τα AM παρείχαν στο χρήση ένα μοντέλο όπου ο έλεγχος και οι μεταφορές δεδομένων ήταν ενοποιημένες διαδικασίες. Κάθε μήνυμα καθορίζει ένα απομακρυσμένο handler ο οποίος εκτελείται κατά την παραλαβή του μηνύματος.

3.6.9 Fast Messages

Παρόμοια λειτουργικότητα με τα AM παρείχαν τα Fast Messages [LPC98, PKC97, PLC95] του πανεπιστημίου Urbana-Champaign του Illinois. Το συγκεκριμένο στρώμα επικοινωνίας

Σύστημα	Μεταφορά Δεδομένων (host-NI)	Μετάφραση	Προστασία	Μεταφορά Ελέγχου	Υποστήριξη multicast
AM-II	Προγραμ. Ε/Ε & ΑΠΜ	Περιοχές ΑΠΜ	Ναι	Δειγματ. & Διακοπές	Όχι
FM	Προγραματιζόμενη Ε/Ε	Περιοχές ΑΠΜ	Όχι	Δειγματοληψία	Όχι
FM/MC	Προγραματιζόμενη Ε/Ε	Περιοχές ΑΠΜ	Όχι	Δειγματ. & Διακοπές	Ναι
PM	ΑΠΜ	Λογισμικό TLB στον προσαρμογέα δικτύου	Ναι	Δειγματοληψία	Ναι
VMMC	ΑΠΜ	Λογισμικό TLB στον προσαρμογέα δικτύου	Ναι	Δειγματ. & Διακοπές	Όχι
VMMC-2	ΑΠΜ	UTLB στον πυρήνα	Ναι	Δειγματ. & Διακοπές	Όχι
LFC	Προγραματιζόμενη Ε/Ε	Μεταφράσεις χρήστη	Όχι	Δειγματ. & Διακοπές	Ναι
Hamlyn	Προγραμ. Ε/Ε & ΑΠΜ	Περιοχές ΑΠΜ	Ναι	Δειγματ. & Διακοπές	Όχι
Trapeze	ΑΠΜ	ΑΠΜ σε πλαίσια σελίδων	Όχι	Δειγματ. & Διακοπές	Όχι
BIP	Προγραμ. Ε/Ε & ΑΠΜ	Μεταφράσεις χρήστη	Όχι	Δειγματοληψία	Όχι
U-Net	ΑΠΜ	TLB στον προσαρμογέα δικτύου (U-Net/MM)	Ναι	Δειγματ. & Διακοπές	Όχι

Πίνακας 3.2: 11 συστήματα επικοινωνίας επιπέδου χρήστη.

ήταν φορητό και μπορούσε να επιτύχει υψηλούς ρυθμούς δικτυακών δεδομένων, ακόμα και για μικρά μηνύματα. Μπορούσε να επιτύχει τη χαμηλή καθυστέρηση που απαιτούνταν για το συγχρονισμό και τη μεταφορά δεδομένων σε παράλληλα συστήματα μεγάλης κλίμακας.

3.6.10 BIP

BIP [PT98] είναι η συντομογραφία του Basic Inteface for Parallelism. Η βασική ιδέα ήταν να δημιουργηθεί μια βιβλιοθήκη προσβάσιμη από τις εφαρμογές, που θα υλοποιούσε ένα πρωτόκολλο υψηλής ταχύτητας με τις λιγότερες δυνατές προσβάσεις στον πυρήνα.

Το BIP προσέφερε διαχείριση του δικτύου από το επίπεδο χρήστη και zero copy στις επικοινωνίες μέσω της κράτησης αγκιστρωμένων (pinned-down) τμημάτων μνήμης και της απεικόνισής τους στο χώρο χρήστη. Για να το επιτύχει αυτό, το BIP είχε ειδικές ρουτίνες που έμπαιναν σε χώρο πυρήνα για τη μετάφραση εικονικών διευθύνσεων σε φυσικές, ήταν ανάγκη να τρέχει μόνο μία εφαρμογή κάθε φορά αφού δεν υπήρχε προστασία μεταξύ των διεργασιών και το δίκτυο διασύνδεσης έπρεπε να ήταν προγραμματιζόμενο, ώστε να τοποθετηθούν σε αυτό ορισμένες από τις λειτουργίες του BIP. Για το λόγο αυτό επελέγη το Myrinet.

3.6.11 Virtual Interface Architecture

Η δυνατότητα παροχής των δικτύων αυξάνεται και οι καθυστερήσεις μέσω αυτών των δικτύων μειώνονται. Δυστυχώς, οι εφαρμογές δεν μπορούν να αξιοποιήσουν πλήρως αυτές τις βελτιώσεις επιδόσεων, εξαιτίας των πολλών επιπέδων λογισμικού χρήστη και πυρήνα που απαιτούνται για να μεταφερθεί η πληροφορία από και προς το δίκτυο.

Για να αντιμετωπίσουν το πρόβλημα, οι εταιρείες Intel, Compaq (πλέον Hewlett Packard) και Microsoft συνέταξαν τις προδιαγραφές [CIM97] για την Αρχιτεκτονική Εικονικού Προσαρμογέα (Virtual Interface Architecture – VIA). Παρακάτω αναλύονται τα βασικότερα χαρακτηριστικά της VIA, ενώ περισσότερες πληροφορίες υπάρχουν στα [DRM+98, EV98, SASB99, KKJ01, DS99, BAP00, Buo99, KKJ02].

Η VIA μειώνει σημαντικά την επιβάρυνση του λογισμικού ανάμεσα σε ένα υποσύστημα επεξεργαστή-μνήμης και σε ένα δίκτυο υψηλών επιδόσεων. Ορίζει ένα σύνολο από συναρτήσεις, δομές δεδομένων και την αντίστοιχη σημασιολογία (semantics) για τη μεταφορά δεδομένων προς και από τη μνήμη μιας διεργασίας. Επιτυγχάνει επικοινωνία και ανταλλαγές δεδομένων χαμηλής καθυστέρησης και υψηλού ρυθμού παροχής μεταξύ διεργασιών που τρέχουν σε δύο κόμβους μιας συστοιχίας υπολογιστών, με πολύ χαμηλό ποσοστό χρησιμοποίησης της ΚΜΕ. Η VIA δίνει σε μια διεργασία χρήστη απευθείας πρόσβαση στο δικτυακό προσαρμογέα, αποφεύγοντας τις ενδιάμεσες αντιγραφές δεδομένων και παρακάμπτοντας το λειτουργικό σύστημα με ένα απολύτως ασφαλή τρόπο. Ελαχιστοποιεί τη χρήση της ΚΜΕ αποφεύγοντας τις διακοπές

και τις μεταγωγές περιεχομένου (context switches) όποτε αυτό είναι εφικτό.

3.6.11.1 Διαδιεργασιακή Επικοινωνία

Η VIA αντιμετωπίζει το πρόβλημα της σχετικά χαμηλής απόδοσης της διαδιεργασιακής επικοινωνίας μέσα σε μια συστοιχία υπολογιστών. Η απόδοση της διαδιεργασιακής επικοινωνίας εξαρτάται από την επιβάρυνση του λογισμικού στην αποστολή και τη λήψη μηνυμάτων και το χρόνο που απαιτείται για να μεταφερθεί το μήνυμα μέσω του δικτύου. Ο αριθμός των επιπέδων που προσπελαύνονται και ο αριθμός των διακοπών, των context switches, και των αντιγραφών δεδομένων στα όρια μεταξύ των επιπέδων, επιβαρύνουν σημαντικά τη διαδικασία της επικοινωνίας [KC94].

Βασικά, οι ταχύτεροι επεξεργαστές οι οποίοι εκτελούν τα δικτυακά πρωτόκολλα σε λιγότερο χρόνο, μειώνουν την επιβάρυνση της επικοινωνίας λόγω πολλών ενδιάμεσων επιπέδων λογισμικού. Για να αξιοποιηθεί η αύξηση στη συχνότητα του ρολογιού, οι σχεδιαστές έχουν χρησιμοποιήσει βαθύτερα pipelines, έξυπνους αλγορίθμους για πρόβλεψη διακλαδώσεων στο υλικό, περισσότερους καταχωρητές, μεγαλύτερες και γρηγορότερες κρυφές μνήμες cache. Αυτές οι καινοτομίες οδηγούν στην ταχύτερη εκτέλεση κομματιών κώδικα. Όμως, οδηγούν και σε μεγαλύτερες ποινές (μετριούνται συνήθως σε κύκλους ρολογιού) στις περιπτώσεις που διακόπτεται η εκτέλεση κώδικα, στη μετάβαση στον κώδικα άλλης διεργασίας με το περιβάλλον της, και στο άδειασμα ενός τμήματος της κρυφής μνήμης εντολών. Δηλαδή, η αύξηση της συχνότητας ρολογιού δεν οδηγεί αναγκαία σε ανάλογη μείωση στην επιβάρυνση λογισμικού για κάθε μήνυμα.

Αυτή τη στιγμή το πρότυπο de facto στις δικτυακές ταχύτητες είναι τα 100 Mbps που προσφέρει το FastEthernet, ενώ υπάρχουν αρκετές εγκαταστάσεις που έχουν υιοθετήσει το 1 Gbps, μέσω του Gigabit Ethernet. Τα 10 Gbps είναι ακόμα αρκετά ακριβά, αλλά δεν θα αργήσουν να εμφανιστούν σε εγκαταστάσεις συστοιχιών υπολογιστών. Αυτές οι εντυπωσιακές αυξήσεις στη δυνατότητα παροχής έχουν μειώσει το χρόνο ολοκλήρωσης μεγάλων μεταφορών. Όμως, όπως αναφέρθηκε στην παράγραφο 3.3, δεν μπορούμε να εκτιμήσουμε το ρυθμό παροχής άμεσα από τη δυνατότητα παροχής. Πρέπει να γνωρίζουμε την καθυστέρηση που εισάγουν τα στοιχεία λογισμικού και υλικού για να καταλήξουμε σε αξιόπιστα συμπεράσματα.

3.6.11.2 Επικοινωνιακή Κίνηση

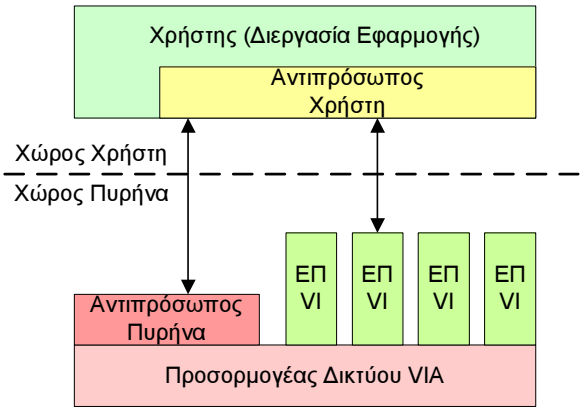
Χωρίς τη γνώση της κίνησης ενός συστήματος, δεν μπορούμε να εκτιμήσουμε τη σχετική σημασία της μείωσης της επιβάρυνσης του λογισμικού σε σχέση με την αύξηση της δυνατότητας παροχής. Αν η πλειοψηφία των μηνυμάτων είναι μεγάλα μηνύματα, τότε η μεγάλη δυνατότητα παροχής είναι σημαντική (βλ. Παράγραφο 3.3). Όμως, αν η πλειοψηφία των μηνυμάτων είναι

μικρά μηνύματα, το μέσο κόστος ανά byte είναι σχετικά υψηλό, και η καθυστέρηση λογισμικού γίνεται ο βασικότερος παράγοντας στην καθυστέρηση του μηνύματος.

Μελέτες στην επικοινωνία σε τοπικά δίκτυα δείχνουν ότι το τυπικό σχήμα επικοινωνίας είναι διπολικό με πάνω από το 80% των μηνυμάτων να είναι μικρότερα ή ίσα με 200 bytes και περίπου 8% των μηνυμάτων με μέγεθος μεγαλύτερο από 8192 bytes. Το πιο σημαντικό πρόβλημα της απόδοσης επικοινωνίας σε συστοιχίες υπολογιστών είναι η επιβάρυνση λογισμικού στα λειτουργικά συστήματα με εικονική μνήμη (όλα τα μοντέρνα λειτουργικά συστήματα υποστηρίζουν εικονική μνήμη) κατά την αποστολή και τη λήψη μηνυμάτων.

3.6.11.3 Περιγραφή της VIA

Χρησιμοποιούμε τους εξής δύο όρους όταν περιγράφουμε την αρχιτεκτονική VI (βλ. Σχήμα 3.9): αντιπρόσωπος χρήστη (user agent) και αντιπρόσωπος πυρήνα (kernel agent). Ο αντιπρόσωπος χρήστη είναι το στρώμα λογισμικού που χρησιμοποιεί την αρχιτεκτονική. Μπορεί να είναι είτε μια εφαρμογή είτε ένα στρώμα υπηρεσίας επικοινωνίας. Ο αντιπρόσωπος πυρήνα είναι ένα οδηγός που τρέχει σε κατάσταση πυρήνα (kernel mode). Αυτός χρειάζεται για να αρχικοποιήσει τους αναγκαίους πίνακες και δομές που επιτρέπουν την επικοινωνία μεταξύ συνεργαζόμενων διεργασιών. Σημαντικό στοιχεία είναι ότι δεν βρίσκεται στο κρίσιμο μονοπάτι των μεταφορών δεδομένων. Παρακάτω περιγράφουμε τις λειτουργίες του.



Σχήμα 3.9: Η Αρχιτεκτονική Εικονικού Προσαρμογέα (VIA).

Η VIA ορίζει ένα απλό σύνολο λειτουργιών που μεταφέρουν δεδομένα μεταξύ συνδεσιοστρεφών άκρων με πολύ χαμηλή καθυστέρηση. Επιτυγχάνει χαμηλή καθυστέρηση σε ένα περιβάλλον ανταλλαγής μηνυμάτων ακολουθώντας τους εξής κανόνες:

- Αποφεύγει οποιαδήποτε ενδιάμεση αντιγραφή δεδομένων.

- Αποφεύγει διακοπές στο λειτουργικό σύστημα όποτε είναι δυνατόν ώστε να αποφύγει τα context switches στην ΚΜΕ όπως επίσης και τη μόλυνση της κρυφής μνήμης.

- Εξαλείφει την ανάγκη για οδηγό συσκευής σε κατάσταση πυρήνα για την πολυπλεξία ενός υλικού πόρου (του προσαρμογέα δικτύου) μεταξύ των πολλαπλών ταυτόχρονων διεργασιών.

- Ελαχιστοποιεί τον αριθμό των εντολών που πρέπει να εκτελέσει μια διεργασία προκειμένου να ξεκινήσει μεταφορά δεδομένων.

- Εξαλείφει τον περιορισμό της απαίτησης διακοπής κατά την αρχικοποίηση ή/και ολοκλήρωση μιας λειτουργίας Ε/Ε.

- Ορίζει ένα απλό σύνολο λειτουργιών για την αποστολή και τη λήψη δεδομένων.

- Κρατάει την αρχιτεκτονική αρκετά απλή ώστε να μπορεί να υλοποιηθεί τόσο σε λογισμικό (emulation) όσο και σε υλικό.

3.6.11.4 VI instances

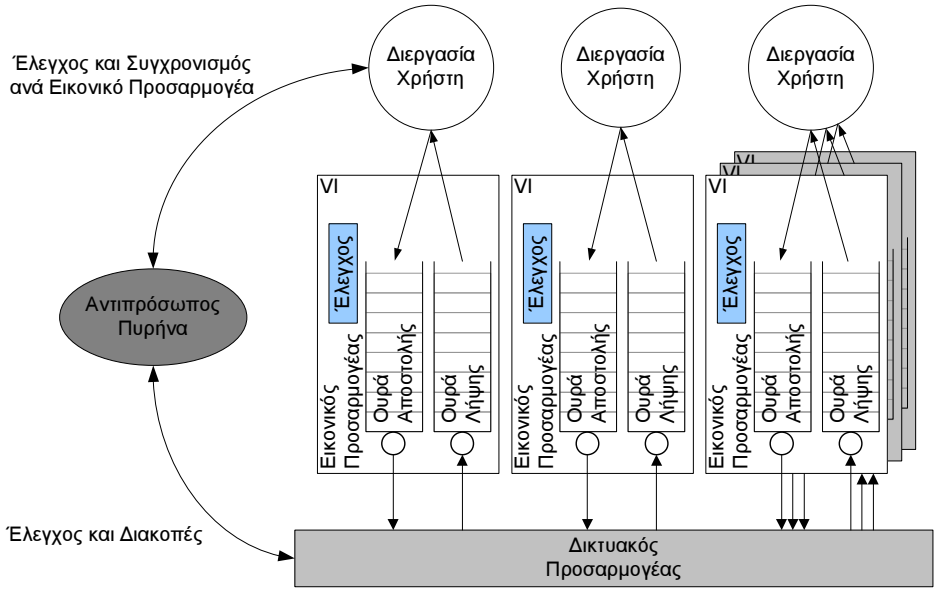
Μέσω της VIA, κάθε διεργασία έχει την ψευδαίσθηση ότι ο δικτυακός προσαρμογέας της ανήκει. Αυτό πραγματοποιείται με την έννοια του στιγμιότυπου VI (VI instance). Κάθε Εικονικός Προσαρμογέας αποτελείται από μια ουρά αποστολής και μια ουρά λήψης. Ο εικονικός προασρμογέας ανήκει σε μια μόνο διεργασία.

Μια διεργασία μπορεί να έχει πολλούς εικονικούς προσαρμογείς, . Ο πυρήνας μπορεί επίσης να έχει εικονικούς προσαρμογείς (βλ. Σχήμα 3.10).

Κάθε ουρά στον εικονικό προσαρμογέα σχηματίζεται από μια συνδεδεμένη λίστα από περιγραφητές. Για να προστεθεί ένας περιγραφητής στην ουρά, ο χρήστης δημιουργεί τον περιγραφητή και τον τοποθετεί στο τέλος της κατάλληλης ουράς εργασίας. Ο ίδιος χρήστης τραβάει περιγραφητές από την κεφαλή της ουράς ολοκλήρωσης όπου τοποθετήθηκαν.

Η προσθήκη ενός περιγραφητή περιλαμβάνει 1) τη σύνδεση του περιγραφητή που προστίθεται με τον τελευταίο στην επιθυμητή ουρά και 2) την ειδοποίηση του δικτυακού προσαρμογέα για την προσθήκη του περιγραφητή. Η ειδοποίηση πραγματοποιείται γράφοντας σε έναν απεικονισμένο στη μνήμη καταχωρητή που ονομάζεται doorbell.

Η διεργασία που κατέχει έναν εικονικό προσαρμογέα μπορεί να τοποθετήσει 4 ειδών περιγραφητές. Περιγραφητές αποστολής, απομακρυσμένης-ΑΠΜ/εγγραφής και απομακρυσμένης-ΑΠΜ/ανάγνωσης τοποθετούνται στην ουρά αποστολής ενός εικονικού προσαρμογέα. Περιγραφητές λήψης τοποθετούνται στην ουρά λήψης ενός εικονικού προσαρμογέα.



Σχήμα 3.10: Οι ουρές εργασίας στην Αρχιτεκτονική Εικονικού Προσαρμογέα (VIA).

3.6.11.5 Συγχρονισμός

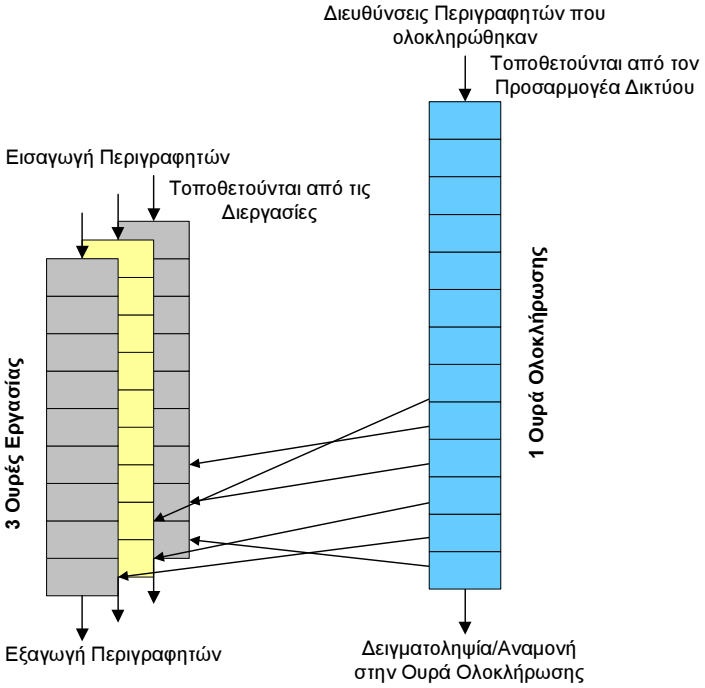
Η VIA παρέχει δύο τρόπους για το συγχρονισμό των διεργασιών με τις λειτουργίες που έχουν ολοκληρωθεί: δειγματοληψία (polling) και διακοπή (interrupt). Όταν η επεξεργασία ενός περιγραφητή ολοκληρωθεί, ο προσαρμογέας δικτύου γράφει ενός bit ολοκλήρωσης και συμπεριλαμβάνει οποιαδήποτε bit λαθών που συσχετίζονται με τον περιγραφητή σε καθορισμένα πεδία μέσα σε αυτόν.

Η κεφαλή κάθε ουράς κάθε εικονικού προσαρμογέα μπορεί να γίνει αντικείμενο δειγματοληψίας. Η δειγματοληψία ελέγχει αν ο περιγραφητής στην κεφαλή της ουράς εργασίας έχει σημειωθεί σαν ολοκληρωμένος. Αν έχει ολοκληρωθεί, η κλήση της συνάρτησης διαγράφει τον περιγραφητή από την κεφαλή και επιστρέφει τη διεύθυνσή του στην καλούσα διεργασία. Διαφορετικά, επιστρέφει την κατάσταση και η κεφαλή της ουράς δεν αλλάζει.

Ο χρήστης μπορεί, επίσης, να χρησιμοποιήσει μια κλήση blocking για να ελέγξει την κεφαλή της ουράς για ολοκληρωμένο περιγραφητή. Αν είναι ολοκληρωμένος, η συνάρτηση κάνει τα ίδια με την προηγούμενη περίπτωση. Ειδάλλως, η συνάρτηση ζητάει από το λειτουργικό σύστημα να αφαιρέσει την καλούσα διεργασία από τη λίστα των προς εκτέλεση διεργασιών, μέχρι ο περιγραφητής στην κεφαλή της ουράς ολοκληρωθεί. Τότε θα δημιουργηθεί μια διακοπή. Η VIA υποστηρίζει τόσο την περίπτωση των διακοπών όσο και τις κλήσεις συναρτήσεων callback ως μέσο αφύπνισης διεργασιών.

3.6.11.6 Ουρές Ολοκλήρωσης

Αυτές οι ουρές (βλ. Σχήμα 3.11) είναι μία προσθήκη που επιτρέπει την ομαδοποίηση ειδοποιήσεων ολοκλήρωσης από πολλαπλές εργασίες σε μία ουρά. Οι δύο ουρές εργασίας μπορούν να συσχετιστούν με ουρές ολοκλήρωσης, ανεξάρτητα η μια από την άλλη. Οι ουρές ολοκλήρωσης του ίδιου εικονικού προσαρμογέα μπορούν να συσχετιστούν με διαφορετικές ουρές ολοκλήρωσης· είναι δυνατόν να συσχετίσουμε μόνο μία ουρά εργασίας ενός εικονικού προσαρμογέα με μια ουρά ολοκλήρωσης.



Σχήμα 3.11: Παράδειγμα με μία ουρά ολοκλήρωσης η οποία περιέχει τις εργασίες που ολοκληρώθηκαν από τρεις ουρές εργασίας.

Όταν ολοκληρωθεί ο περιγραφητής που αντιστοιχεί σε μια ουρά ολοκλήρωσης, ο δικτυακός προσαρμογέας σημειώνει τον περιγραφητή και τοποθετεί ένα δείκτη σε αυτόν στο τέλος της αντίστοιχης ουράς ολοκλήρωσης. Αν μια ουρά εργασίας εικονικού προσαρμογέα είναι συσχετισμένη με μια ουρά ολοκλήρωσης, ο συγχρονισμός με τους ολοκληρωμένους περιγραφητές μπορεί να επιτευχθεί είτε με δειγματοληψία είτε περιμένοντας στην ουρά ολοκλήρωσης, αλλά όχι στην ίδια την ουρά εργασίας.

3.6.11.7 Περιγραφητές

Αυτές οι δομές περιγράφουν τις εργασίες που πρέπει να εκτελεστούν από το δικτυακό προσαρμογέα. Περιγραφητές αποστολής και λήψης περιλαμβάνουν ένα τμήμα ελέγχου και ένα μεταβλητό πλήθος από τμήματα δεδομένων. Περιγραφητές για εγγραφές και αναγνώσεις απομακρυσμένης-ΑΠΜ περιέχουν ένα επιπλέον τμήμα διεύθυνσης.

Το τμήμα ελέγχου περιέχει τον τύπο του περιγραφητή, τα άμεσα δεδομένα, αν υπάρχουν, το πλήθος των τμημάτων που ακολουθούν το τμήμα ελέγχου, την εικονική διεύθυνση του επόμενου περιγραφητή στην ουρά και το αντίστοιχο χειριστήριο μνήμης, την κατάσταση του περιγραφητή (αρχικά κενό όταν τοποθετείται και συμπληρώνεται από τον προσαρμογέα δικτύου με την ολοκλήρωση) και το συνολικό μήκος των δεδομένων που πρέπει να μεταφερθούν από τον περιγραφητή.

Το τμήμα διεύθυνσης περιέχει την εικονική διεύθυνση του απομακρυσμένου τμήματος μνήμης και το αντίστοιχο χειριστήριο μνήμης.

Κάθε τμήμα δεδομένων περιγράφει ένα τμήμα μνήμης και περιέχει την εικονική διεύθυνση του τοπικού τμήματος μνήμης, το αντίστοιχο χειριστήριο μνήμης (memory handle) και το μήκος των δεδομένων που αντιστοιχούν σε αυτό το τμήμα μνήμης.

Στους περιγραφητές εγγραφής απομακρυσμένης-ΑΠΜ, το ζεύγος διεύθυνση/χειριστήριο μνήμης στο μέρος της διεύθυνσης είναι η αρχική τοποθεσία του τμήματος όπου τοποθετούνται τα δεδομένα. Αν και υποστηρίζεται μόνο διεύθυνση τμήματος απομακρυσμένης μνήμης ανά περιγραφητή, ο χρήστης μπορεί να καθορίσει πολλαπλά τοπικά τμήματα μνήμης σε κάθε περιγραφητή εγγραφής απομακρυσμένης ΑΠΜ. Συνεπώς, είναι δυνατόν να επιτευχθεί η λειτουργία gather των δεδομένων, αλλά δεν γίνεται να επιτευχθεί η λειτουργία scatter με ένα μόνο περιγραφητή εγγραφής απομακρυσμένης-ΑΠΜ.

Στους περιγραφητές ανάγνωσης απομακρυσμένης-ΑΠΜ, το ζεύγος διεύθυνση/χειριστήριο μνήμης στο μέρος της διεύθυνσης είναι η αρχική τοποθεσία του τμήματος όπου τοποθετούνται τα δεδομένα. Όμοια, υποστηρίζονται πολλαπλά τοπικά τμήματα μνήμης αλλά ένα απομακρυσμένο. Δηλαδή, μπορεί να επιτευχθεί η λειτουργία scatter αλλά όχι η gather.

3.6.11.8 Άμεσα Δεδομένα

Η VIA επιτρέπει να καθοριστούν 32 bits άμεσων δεδομένων σε κάθε περιγραφητή. Όταν υπάρχουν άμεσα δεδομένα σε έναν περιγραφητή αποστολής, ο προσαρμογέας δικτύου μεταφέρει το πεδίο κατευθείαν στον περιγραφητή λήψης. Η παρουσία άμεσων δεδομένων σε έναν περιγραφητή εγγραφής απομακρυσμένης-ΑΠΜ προκαλεί την κατανάλωση ενός περιγραφητή λήψης στον απομακρυσμένο εικονικό προσαρμογέα με το πεδίο άμεσων δεδομένων να μεταφέρεται στον περιγραφητή λήψης που καταναλώθηκε. Η μέριμνα για τα άμεσα δεδομένα έχει σκοπό

να επιταχύνει τη διαδικασία αποστολής μικρής ποσότητα δεδομένων (μέχρι 32 bits), η οποία πρέπει να μεταφερθεί στον παραλήπτη με την ελάχιστη δυνατή καθυστέρηση.

3.6.11.9 Σειρά στις Ουρές Εργασίας

Η VIA διατηρεί κανόνες για τη διατήρηση της σειράς και της συνέπειας των δεδομένων μέσα σε έναν εικονικό προσαρμογέα, αλλά όχι ανάμεσα σε διαφορετικούς εικονικούς προσαρμογείς. Οι περιγραφητές που τοποθετούνται σε έναν εικονικό προσαρμογέα, επεξεργάζονται με σειρά FIFO. Η VIA δεν καθορίζει τη σειρά ολοκλήρωσης των περιγραφητών που τοποθετήθηκαν σε διαφορετικούς εικονικούς προσαρμογείς· η σειρά εξαρτάται από τον αλγόριθμο δρομολόγησης που υλοποιείται.

Ο κανόνας διατηρείται εύκολα στην περίπτωση των αποστολών και των εγγραφών απομακρυσμένης-ΑΠΜ, διότι οι αποστολές συμπεριφέρονται σαν απομακρυσμένες εγγραφές χωρίς το τμήμα απομακρυσμένης μνήμης. Ο περιγραφητής λήψης στην κεφαλή της ουράς καθορίζει την απομακρυσμένη διεύθυνση (τις απομακρυσμένες διευθύνσεις).

Στην περίπτωση των αναγνώσεων απομακρυσμένης-ΑΠΜ, είναι δυσκολότερο να διατηρηθεί η συνέπεια των δεδομένων με τα γεμάτα pipelines. Οι αναγνώσεις απομακρυσμένης-ΑΠΜ είναι συναλλαγές round-trip και δεν είναι ολοκληρωμένες αν τα δεδομένα δεν επιστρέψουν από τον απομακρυσμένο κόμβο στον αρχικό.

3.6.11.10 Δρομολόγηση μεταξύ των Ουρών Εργασίας

Δεν υπάρχει συγκεκριμένη σειρά ανάμεσα στην εκτέλεση των περιγραφητών που βρίσκονται σε διαφορετικούς εικονικούς προσαρμογείς. Η δρομολόγηση της υπηρεσίας για κάθε εικονικό προσαρμογέα εξαρτάται από τον αλγόριθμο που χρησιμοποιείται στο δικτυακό προσαρμογέα, τα μεγέθη των μηνυμάτων που αντιστοιχίζονται στους ενεργούς περιγραφητές και στον υφιστάμενο τρόπο μεταφοράς που χρησιμοποιείται.

3.6.11.11 Προστασία Μνήμης

Η VIA παρέχει προστασία μνήμης για όλες τις λειτουργίες με τους εικονικούς προσαρμογείς, για να εξασφαλίσει ότι καμία διεργασία δεν μπορεί να στείλει δεδομένα σε (ή να λάβει δεδομένα από) μνήμη που δεν της ανήκει. Ο μηχανισμός *ετικέτας προστασίας*, παρέχει την υπηρεσίας προστασίας μνήμης. Αυτός προγραμματίζεται από τον αντιπρόσωπο πυρήνα στον πίνακα μετάφρασης και προστασίας.

Οι ετικέτες προστασίας είναι μοναδικές ταυτότητες που αντιστοιχούν σε εικονικούς προσαρμογείς και τμήματα μνήμης. Ένας χρήστης, πριν δημιουργήσει ένα νέο εικονικό προσαρμογέα ή δηλώσει ένα τμήμα μνήμης, πρέπει να προμηθευτεί μια ετικέτα προστασίας. Όταν ο χρήστης

ζητήσει τη δημιουργία ενός εικονικού προσαρμογέα ή τη δήλωση τμήματος μνήμης, περνάει ως παράμετρο στην κλήση την ετικέτα προστασίας. Ο αντιπρόσωπος πυρήνα ελέγχει για να επιβεβαιώσει ότι ο χρήστης που κάνει την αίτηση όντως κατέχει την ετικέτα προστασίας.

Όμοια, όταν ο χρήστης δηλώνει ένα τμήμα μνήμης, πρέπει να καθορίσει την ετικέτα μνήμης που αντιστοιχεί στο συγκεκριμένο τμήμα, αν είναι ενεργοποιημένες οι απομακρυσμένες εγγραφές και αναγνώσεις. Ο αντιπρόσωπος πυρήνα ελέγχει για να επιβεβαιώσει ότι η διεργασία που δηλώνει τα τμήματα μνήμης κατέχει το τμήμα μνήμης και την ετικέτα προστασίας. Στη συνέχεια, η δήλωση της μνήμης επιτυγχάνεται και ο αντιπρόσωπος πυρήνα προγραμματίζει τις κατάλληλες θέσεις και αντίστοιχα bits στον πίνακα μετάφρασης και προστασίας και επιστρέφει το χειριστήριο μνήμης για το συγκεκριμένο τμήμα.

Μια διεργασία μπορεί να έχει δημιουργήσει πολλούς εικονικούς προσαρμογείς, με την ίδια ή με διαφορετικές ετικέτες προστασίας. Κάθε ετικέτα προστασίας είναι διαφορετική· διαφορετικές διεργασίες δεν μπορούν να μοιράζονται ετικέτες προστασίας. Οι εικονικοί προσαρμογείς μπορούν να προσπελαύνουν περιοχές μνήμης μόνο με την ίδια ετικέτα προστασίας. Συνεπώς, δεν μπορούν όλοι οι εικονικοί προσαρμογείς που ανήκουν σε μια διεργασία, να προσπελάσουν όλα τα τμήματα μνήμης που ανήκουν στη συγκεκριμένη διεργασία.

3.6.11.12 Μετάφραση Εικονικής Μνήμης

Μία σημαντική εργασία που πραγματοποιείται από τον αντιπρόσωπο πυρήνα λαμβάνει χώρα όταν δηλώνεται ένα τμήμα μνήμης, δίνοντας στο δικτυακό προσαρμογέα τη δυνατότητα μετάφρασης εικονικών διευθύνσεων σε φυσικές.

Όταν ο χρήστης ζητάει τη δήλωση μιας περιοχής μνήμης, ο αντιπρόσωπος πυρήνα πραγματοποιεί τον έλεγχο ιδιοκτησίας, αγκιστρώνει τις σελίδες στη φυσική μνήμη και πραγματοποιεί τις μεταφράσεις εικονικής μνήμης σε φυσική. Ο αντιπρόσωπος πυρήνα τοποθετεί τις φυσικές διευθύνσεις στον πίνακα μετάφρασης και προστασίας και επιστρέφει ένα χειριστήριο μνήμης. Ο χρήστης μπορεί να αναφέρεται στην περιοχή μνήμης, χρησιμοποιώντας το χειριστήριο, χωρίς να ανησυχεί για το αν θα περάσει τα όρια σελίδων.

Η χρήση ζεύγους χειριστηρίου μνήμης-εικονικής διεύθυνσης είναι μια καινοτομία της VIA. Σε αντίθεση με τις αρχιτεκτονικές Hamlyn [BJM+96] και U-Net [EBBV95], στις οποίες οι διευθύνσεις καθορίζονται ως ετικέτα και απόσταση, η VIA επιτρέπει την απευθείας χρήση εικονικών διευθύνσεων. Αυτό σημαίνει ότι η εφαρμογή δεν χρειάζεται να κρατάει αντιστοιχίες μεταξύ εικονικών διευθύνσεων και ετικετών.

Ο δικτυακός προσαρμογέας, που λαμβάνει ένα ζεύγος χειριστηρίου μνήμης/ εικονικής διεύθυνσης, είναι υπεύθυνος για τον καθορισμό της φυσικής διεύθυνσης και την επιβεβαίωση ότι ο χρήστης έχει τα κατάλληλα δικαιώματα για τη ζητούμενη λειτουργία. Ο σχεδιαστής του προσαρμογέα είναι ελεύθερος να επιλέξει τη μορφή του χειριστηρίου μνήμης, καθώς και του

τρόπου που η πληροφορία μετάφρασης και προστασίας αποθηκεύεται και ανακτάται.

Υπάρχουν δύο επιπλέον συνέπειες της δήλωση περιοχών μνήμης σε αντίθεση με την πιο κοινή δήλωση σελίδων. Ο χρήστης δεν ασχολείται με το πέρασμα των ορίων των σελίδων. Συνεπώς, ο δικτυακός προσαρμογέας πρέπει να γνωρίζει το μέγεθος των σελίδων, έτσι ώστε, όταν μια εικονική διεύθυνση περάσει σε άλλη σελίδα, ο ελεγκτής λαμβάνει νέα μετάφραση (οι φυσικές διευθύνσεις δεν χρειάζεται να είναι συνεχόμενες). Αν και ο χρήστης μπορεί να ζητήσει να δηλωθεί μόνο ένα μέρος μιας σελίδας, στην πραγματικότητα θα δηλωθεί ολόκληρη η σελίδα και ολόκληρη η σελίδα υπόκειται στις ιδιότητες προστασίας για το τμήμα που δηλώθηκε.

3.7 Scalable Coherent Interface

Το Scalable Coherent Interface (SCI) είναι μία καινοτόμος δικτυακή τεχνολογία για την περιοχή των συστημάτων και δικτύων υψηλών επιδόσεων, που περιγράφεται στο πρότυπο (standard) 1596-1992 του διεθνούς οργανισμού IEEE [Ins92]. Περισσότερες λεπτομέρειες για το πρότυπο και τον τρόπο λειτουργίας του μπορούν να βρεθούν στα [Gus92, GL94, TR99, Tra98, WSB01, Hel99, HR99, Cla96, Aam98, TR01, Sei99a, Han01, ZHS99, Oma95, OP97, SWR01, Tvi92, TRBS00, TRS99, TSR00, JLGS90, HEH98, ISSW97, Sol99, WGRB02, Sei99b, WB99, SH98, GOB01, CKR$^+$00, EHK$^+$98, ISSA98, Sch97, HT94, SRH99, MBB$^+$]. Στη συνέχεια θα γίνει μια συνοπτική περιγραφή των κυριότερων χαρακτηριστικών που αξιοποιούνται στη διασύνδεση των υπολογιστικών κόμβων μιας συστοιχίας μέσω SCI.

3.7.1 Ιστορικά στοιχεία

Το SCI προέρχεται από την προσπάθεια ειδικών σε τεχνολογίες διαύλων, στα τέλη της δεκαετίας του 1980, να δημιουργήσουν ένα δίαυλο υπολογιστών που θα υποστήριζε πολυεπεξεργασία μεγάλης κλίμακας (δηλ. πολλών επεξεργαστικών στοιχείων). Όμως, η ομάδα κατάλαβε νωρίς ότι ένας «δίαυλος», με την έννοια του backplane, δεν θα μπορούσε να ικανοποιήσει τις απαιτήσεις, για τους εξής λόγους:

- Ένας δίαυλος είναι κεντρικοποιημένος πόρος και η στενωπός της σειριοποίησης θα ήταν σημαντικός παράγοντας πτώσης της απόδοσης, ιδιαίτερα με την αύξηση της ταχύτητας των επεξεργαστών.
- Η σηματοδοσία των διαύλων πλησιάζει τα θεμελιώδη όριά της (ταχύτητα του φωτός), καταλήγοντας σε ηλεκτρικά σύνθετες και ακριβές λύσεις, καθώς και σε διαύλους μικρού μήκους.

Αποτέλεσμα ήταν να εγκαταλειφθεί η κατασκευή ενός διαύλου, ενώ για να ξεπεραστούν τα προβλήματα του μοιραζόμενου πόρου και της σηματοδοσίας αναπτύχθηκαν καινοτόμες,

κατανεμημένες λύσεις. Αυτές διατηρούσαν το συνολικό στόχο του ορισμού μιας τεχνολογίας διασύνδεσης, η οποία προσφέρει τις βολικές υπηρεσίες που προσφέρουν οι δίαυλοι.

Οι προδιαγραφές που προέκυψαν, το πρότυπο SCI, έγιναν δεκτές το 1992. Περιγράφουν το υλικό και τα πρωτόκολλα που παρέχουν στους επεξεργαστές την ενοποιημένη εικόνα της μνήμης. Επίσης, το SCI παρέχει λειτουργίες για ανάγνωση, εγγραφή και κλείδωμα θέσεων μνήμης χωρίς την παρέμβαση λογισμικού, καθώς και τη μετάδοση μηνυμάτων και διακοπών. Πρωτόκολλα υλικού που θα παρείχαν συνάφεια μνήμης ορίζονταν προαιρετικά για την κάθε υλοποίηση. Σε αντίθεση με τις προηγούμενες λύσεις, το δίκτυο SCI, το υποσύστημα μνήμης και τα αντίστοιχα πρωτόκολλα είναι πλήρως κατανεμημένα και επεκτάσιμα. Ένα δίκτυο SCI βασίζεται μόνο σε συνδέσμους σημείου-προς-σημείο και υλοποιεί Κατανεμημένη Μοιραζόμενη Μνήμη (Distributed Shared Memory - DSM) στο επίπεδο υλικού.

3.7.2 Στόχοι

Μερικοί φιλόδοξοι στόχοι που προσδιόρισαν τη διαδικασία προδιαγραφών του SCI και καθόρισαν μερικώς το όνομά του ήταν οι εξής:

Υψηλή Απόδοση: Ο βασικός αντικειμενικός στόχος του SCI, όπως και κάθε δικτύου διασύνδεσης και των αντίστοιχων πρωτοκόλλων, είναι η παροχή υψηλής απόδοσης επικοινωνίας στις παράλληλες και κατανεμημένες εφαρμογές. Οι τρεις παράγοντες που συμβάλλουν στο στόχο αυτό είναι:

- Υψηλό διατηρούμενο Ρυθμό Παροχής δεδομένων
- Χαμηλή Καθυστέρηση
- Χαμηλή επιβάρυνση της ΚΜΕ στις λειτουργίες επικοινωνίας

Οι στόχοι απόδοσης που τέθηκαν ήταν της τάξης των GByte/δευτερόλεπτο για την ταχύτητα συνδέσμων και καθυστερήσεις των μονάδων μικροδευτερολέπτων σε χαλαρά συνδεδεμένα συστήματα.

Επεκτασιμότητα: Το SCI επινοήθηκε για να αντιμετωπίσει το θέμα της επεκτασιμότητας σε πολλές πλευρές, ανάμεσα στις οποίες οι εξής:

- Επεκτασιμότητα της απόδοσης (συνολική Δυνατότητα Παροχής δεδομένων) καθώς ο αριθμός των συνδεδεμένων κόμβων αυξάνεται.
- Επεκτασιμότητα στην απόσταση δικτύωσης, από εκατοστά σε εκατοντάδες μέτρα, ανάλογα με την υλοποίηση του φυσικού μέσου.

- Επεκτασιμότητα του υποσυστήματος μνήμης και συγκεκριμένα των πρωτοκόλλων συνάφειας μνήμης, τα οποία δεν πρέπει να περιέχουν όριο στον αριθμό των επεξεργαστών ή των δομοστοιχείων μνήμης που μπορούν να χειριστούν.
- Υψηλή Δυνατότητα Διευθυνσιοδότησης: Ένα αρκετά μεγάλο σχήμα διευθυνσιοδότησης για KMM, ώστε να υποστηρίζεται ένα μεγάλο πλήθος κόμβων, καθώς και ένα μεγάλο μέγεθος μνήμης σε κάθε κόμβο.

Υποσύστημα μνήμης με Συνάφεια Κρυφής Μνήμης: Οι κρυφές μνήμες που υπάρχουν στους επεξεργαστές, γίνονται όλο και πιο σημαντικές για τη μείωση του μέσου χρόνου προσπέλασης δεδομένων. Αυτό ισχύει και για τα συστήματα Κατανεμημένης Μοιραζόμενης Μνήμης (KMM) με χαρακτηριστικά NUMA, όπου οι απομακρυσμένες προσπελάσεις είναι περίπου μία τάξη μεγέθους πιο ακριβές από τις τοπικές. Για την υποστήριξη ενός βολικού προγραμματιστικού μοντέλου, π.χ. όπως συμβαίνει στους συμμετρικούς πολυεπεξεργαστές, η συνάφεια κρυφής μνήμης πρέπει να υλοποιείται στο υλικό.

Χακαρτηριστικά διεπαφής: Οι προδιαγραφές του SCI περιγράφουν μία προτυποποιημένη διεπαφή του δικτύου διασύνδεσης, η οποία επιτρέπει σε πολλές συσκευές από πολλούς κατασκευαστές να προσαρτηθούν σε αυτό και να διαλειτουργήσουν. Με άλλα λόγια, το SCI είναι ένας ανοικτός κατανεμημένος δίαυλος που συνδέει διάφορα εξαρτήματα όπως μικροεπεξεργαστές, μνήμες, έξυπνες συσκευές E/E, σε ένα σύστημα υψηλής ταχύτητας.

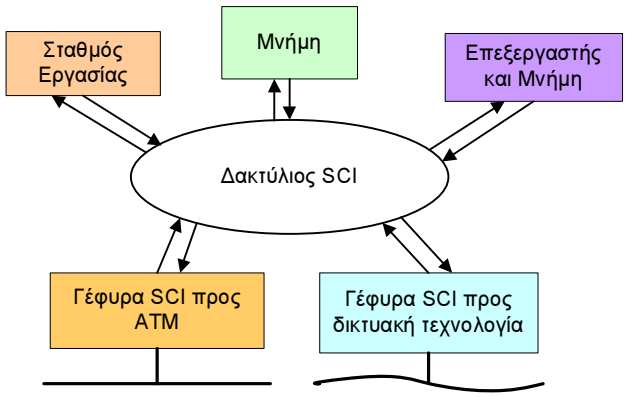
3.7.3 Έννοιες

Πολλοί από τους στόχους που είχαν αρχικά τεθεί, ενσωματώθηκαν με επιτυχία στο πρότυπο SCI. Παρακάτω περιγράφονται μερικές από τις σημαντικότερες έννοιες και πλεονεκτήματα του SCI σε μια προσπάθεια να αποτιμήσουμε τα επιτεύγματα και τη σημερινή του κατάσταση.

Σύνδεσμοι σημείου-προς-σημείο: Ένα δίκτυο SCI ορίζεται μόνο από κατευθυνόμενους συνδέσμους σημείου-προς-σημείο ανάμεσα στους συμμετέχοντας κόμβους. Οι σύνδεσμοι αυτοί μπορούν να χρησιμοποιηθούν για ταυτόχρονες μεταφορές δεδομένων, σε αντίθεση με ό,τι συμβαίνει στους διαύλους υπολογιστών. Ο αριθμός των συνδέσμων μεγαλώνει καθώς νέοι κόμβοι προστίθενται στο σύστημα, με αποτέλεσμα την αύξηση της συνολικής (aggregate) δυνατότητας παροχής δεδομένων του δικτύου.

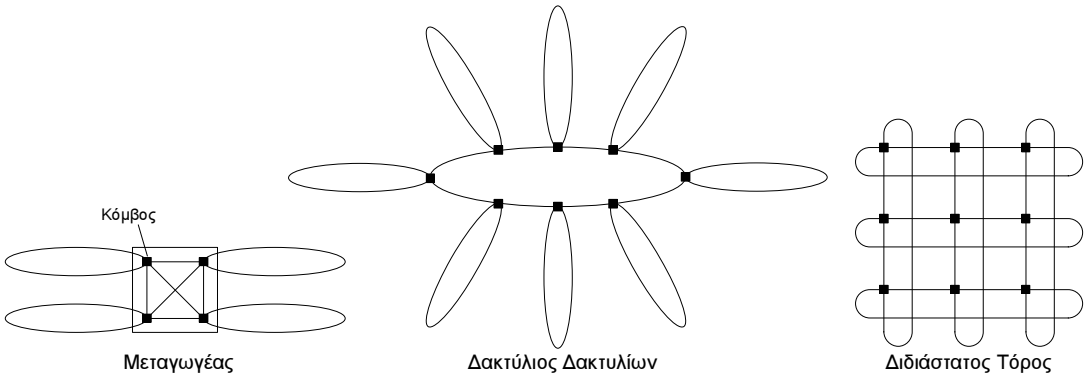
Κόμβοι: Το SCI έχει σχεδιαστεί για να διασυνδέει μεγάλο πλήθος κόμβων (μέχρι 64K). Ο κόμβος μπορεί να είναι ένας πλήρης σταθμός εργασίας, ένας επεξεργαστής με κρυφή μνήμη, μία μνήμη, ελεγκτές και συσκευές E/E, ή γέφυρες προς άλλους διαύλους ή δίκτυα, όπως φαίνεται στο Σχήμα 3.12. Κάθε κόμβος οφείλει να έχει έναν ειδικό προσαρμογέα για να

προσαρτηθεί στο δίκτυο SCI. Στα περισσότερα συστήματα που έχουν υλοποιηθεί μέχρι στιγμής, οι κόμβοι είναι πλήρεις σταθμοί εργασίας (μονοεπεξεργαστικοί ή πολυεπεξεργαστικοί).



Σχήμα 3.12: Παράδειγμα διαφορετικών κόμβων που υποστηρίζονται από το SCI.

Ανεξαρτησία Τοπολογίας: Το SCI επιτρέπει τη δημιουργία πολύπλοκων τοπολογιών (βλ. Σχήμα 3.13). Για μικρά συστήματα, η προτεινόμενη τοπολογία είναι ο μικρός δακτύλιος (ringlet). Για μεγαλύτερα συστήματα, είναι δυνατές οι τοπολογίες πολλαπλών δακτυλίων μέσω μεταγωγέα, δακτύλιοι δακτυλίων και πολυδιάστατοι τόροι. Τα περισσότερα συστήματα μέχρι σήμερα χρησιμοποιούν δακτυλίους, μεταγωγείς, πολλαπλούς δακτύλιους, ή 2-διάστατους τόρους.



Σχήμα 3.13: Απλές δικτυακές τοπολογίες SCI.

Μορφή Διευθύνσεων: Το SCI χρησιμοποιεί ένα 64-bit σχήμα διευθύνσεων που ορίζεται από την αρχιτεκτονική Control and Status (IEEE Std 1212-1991). Η 64-bit διεύθυνση χωρίζεται σε δύο μέρη σταθερού μεγέθους: Τα 16 πιο σημαντικά bits καθορίζουν την ταυτότητα

του κόμβου (διεύθυνση κόμβου), έτσι ώστε ένα δίκτυο SCI μπορεί να επεκτεθεί έως $2^{16} = 64K$ κόμβους. Τα υπόλοιπα 48 bits χρησιμοποιούνται για τη διευθυνσιοδότηση μέσα στους κόμβους.

Κατανεμημένη Μοιραζόμενη Μνήμη σε Υλικό: Το σχήμα διευθυνσιοδότησης του SCI ορίζει έναν 64-bit χώρο διευθύνσεων. Με άλλα λόγια, έναν κατανεμημένο σύστημα μοιραζόμενης μνήμης. Η κατανομή του χώρου είναι διαφανής στο λογισμικό και τους επεξεργαστές, π.χ. η μνήμη είναι λογικά μοιρασμένη όπως σε ένα σύστημα συμμετρικής πολυεπεξεργασίας. Μία πρόσβαση μνήμης κατευθύνεται στο αντίστοιχο δομοστοιχείο μνήμης από το υλικό του SCI.

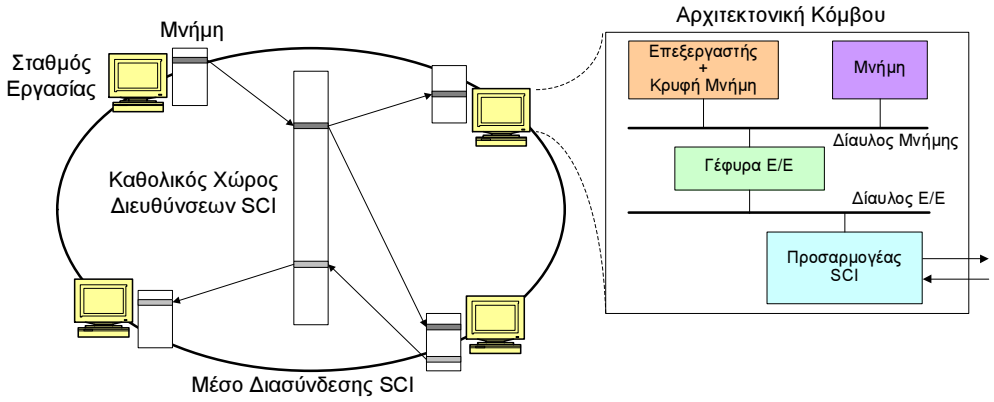
Το μεγάλο πλεονέκτημα αυτού του γνωρίσματος είναι ότι η διακομβική επικοινωνία πραγματοποιείται με εντολές φόρτωσης (load) και αποθήκευσης (store) του επεξεργαστή, χωρίς μεσολάβηση στοίβας λογισμικού δικτυακών πρωτοκόλλων. Εντολές που προσπελαύνουν απομακρυσμένη μνήμη μπορούν να εκτελεστούν στο επίπεδο χρήστη· το λειτουργικό σύστημα δεν χρειάζεται να παρεμβληθεί στη διαδικασία. Το αποτέλεσμα είναι χαμηλή καθυστέρηση της τάξης των μsec για την επικοινωνία.

3.7.4 Το SCI ως Δίκτο Διασύνδεσης για Συστοιχίες Υπολογιστών

Το SCI αρχικά σχεδιάστηκε ως ένα δίκτυο μοιραζόμενης μνήμης και η πρώτη υλοποίηση ενός τέτοιου δικτύου αναδείχθηκε το 1994 στον πολυεπεξεργαστή HP/Convex Exemplar SPP. Όμως, η ευελιξία και οι προοπτικές απόδοσης του SCI για άλλες εφαρμογές, π.χ. σαν τοπικό δίκτυο, γρήγορα αξιοποιήθηκαν από τους κατασκευαστές. Παρακάτω γίνεται μια εισαγωγή σε αυτού του είδους τις κλασσικές εφαρμογές του SCI.

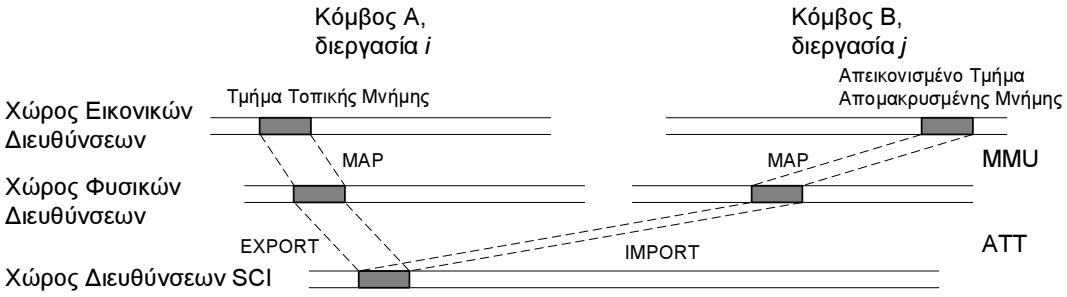
Οι συστοιχίες υπολογιστών γίνονται όλο και πιο σημαντικές σαν φτηνά μέσα παράλληλης και κατανεμημένης επεξεργασίας. Ένα τοπικό δίκτυο SCI παρέχει δυνατότητες επικοινωνίας υψηλών επιδόσεων σε μια τέτοια συστοιχία. Σε αυτή την εφαρμογή, το υλικό διασύνδεσης προσαρτίζεται στο δίαυλο E/E των κόμβων (π.χ. PCI) μέσω μίας κάρτας-προσαρμογέα (βλ. Σχήμα 3.14). Σε αντίθεση με τα υπόλοιπα τοπικά δίκτυα, λόγω του κοινού χώρου διευθύνσεων SCI και των αντίστοιχων συναλλαγών, παρέχεται Κατανεμημένη Μοιραζόμενη Μνήμη βασισμένη στο υλικό. Στο Σχήμα 3.14, φαίνεται η ΚΜΜ σε υψηλό επίπεδο. Μία συστοιχία SCI είναι στενότερα συνδεδεμένη από ένα κοινό τοπικό δίκτυο, εμφανίζοντας χαρακτηριστικά ενός παράλληλου συστήματος NUMA.

Οι προσαρμογείς SCI, μαζί με το λογισμικό οδηγών του SCI, δημιουργούν περιβάλλον ΚΜΜ, όπως φαίνεται στο Σχήμα 3.15. Ένας κόμβος, που επιθυμεί να μοιραστεί μνήμη με άλλους κόμβους, δημιουργεί στη φυσική του μνήμη τμήματα μοιραζόμενης μνήμης και τα εξάγει στο χώρο διευθύνσεων SCI. Οι άλλοι κόμβοι εισάγουν αυτά τα τμήματα ΚΜΜ στο χώ-



Σχήμα 3.14: Το μοντέλο της συστοιχίας υπολογιστών SCI.

ρο διευθύνσεων Ε/Ε. Χρησιμοποιώντας πίνακες μετάφρασης διευθύνσεων στον προσαρμογέα, διατηρούνται απεικονίσεις ανάμεσα στις τοπικές διευθύνσεις Ε/Ε και στις γενικές διευθύνσεις SCI. Οι διεργασίες στους κόμβους μπορούν στη συνέχεια να απεικονίσουν τα τμήματα ΚΜΜ στον εικονικό χώρο διευθύνσεών τους. Οι τελευταίες απεικονίσεις διατηρούνται συμβατικά από τις δομές του Λ.Σ. και τη Μονάδα Διαχείρισης Μνήμης (Memory Management Unit–MMU) του επεξεργαστή.



Σχήμα 3.15: Χώροι διευθύνσεων και μεταφράσεις διευθύνσεων σε συστοιχίες SCI.

Όταν δημιουργηθούν οι απεικονίσεις, μπορεί να πραγματοποιηθεί διακομβική επικοινωνία μεταξύ των διεργασιών σε επίπεδο χρήστη, με απλές λειτουργίες load και store στα τμήματα ΚΜΜ που αντιστοιχούν σε απομακρυσμένη μνήμη. Οι προσαρμογείς SCI μεταφράζουν τις συναλλαγές διαύλου Ε/Ε που προκύπτουν από τέτοιες προσπελάσεις μνήμης σε συναλλαγές SCI, και αντίστροφα, και τις εκτελούν εκ μέρους του επεξεργαστή. Συνεπώς, οι προσπελάσεις απομακρυσμένης μνήμης είναι διαφανείς στις διεργασίες και δεν απαιτούν τη μεσολάβηση του Λ.Σ. Με άλλα λόγια, στις προσπελάσεις μνήμης δεν περιλαμβάνεται η εκτέλεση στοίβας πρωτοκόλλων λογισμικού, καταλήγοντας σε χαμηλές καθυστερήσεις.

3.7.5 Παράδειγμα Επικοινωνίας με SCI

Το βασικό χαρακτηριστικό και δυνατό σημείο του SCI είναι η δυνατότητα προσπέλασης της μνήμης ενός απομακρυσμένου κόμβου. Αν το τμήμα απομακρυσμένης μνήμης απεικονίζεται στο χώρο διευθύνσεων μιας τοπικής διεργασίας, φαίνεται σαν να είναι τοπικό και η μεταφορά δεδομένων είναι τόσο απλή όσο ένα `memcpy()`. Σε αυτή την περίπτωση η ΚΜΕ είναι που διαβάζει ή γράφει στην απομακρυσμένη μνήμη, διαδικασία γνωστή ως Προγραμματιζόμενη Ε/Ε (Programmable I/O - PIO). Εναλλακτικά, η τοπική διεργασία μπορεί να επιλέξει την προσέγγιση με ΑΠΜ, όπου η ΚΜΕ δίνει απλώς τις εντολές στον προσαρμογέα SCI για τη μεταφορά (διεύθυνση πηγής, διεύθυνση προορισμού και μέγεθος) και στη συνέχεια είναι ελεύθερη να ασχοληθεί με άλλες εργασίες.

Και στις δύο περιπτώσεις, αυτό που χρειάζεται είναι ένας τρόπος διαχείρισης τμημάτων τοπικής μνήμης στη μία πλευρά και έναν τρόπο προσάρτισης τους σε τμήματα απομακρυσμένης μνήμης.

Πριν χρησιμοποιηθεί ένα τμήμα της μνήμης, πρέπει να δεσμευθεί. Η δέσμευση ενός τμήματος πραγματοποιείται μέσω μιας ειδικής συνάρτησης και όχι μέσω της γνωστής συνάρτησης βιβλιοθήκης `malloc()`. Ο λόγος για τον οποίο συμβαίνει αυτό είναι ότι ο οδηγός συσκευής πρέπει να γνωρίζει για το δημιουργούμενο τμήμα μνήμης. Το Λ.Σ. απαιτεί το συγκεκριμένο τμήμα μνήμης να είναι αγκιστρωμένο και συνεχόμενο στη φυσική μνήμη.

Μόλις το τμήμα μνήμης δεσμευθεί, πρέπει να γίνει ορατό στους υπόλοιπους κόμβους που είναι συνδεδεμένοι στο δίκτυο SCI. Η συγκεκριμένη λειτουργία, όπως και αυτή της δέσμευσης μνήμης, πραγματοποιείται μέσω ειδικής κλήσης συστήματος που επικοινωνεί με τον οδηγό συσκευής. Πρόκειται για λειτουργίες οι οποίες δεν μπορούν να πραγματοποιηθούν σε επίπεδο χρήστη.

Σε αυτό το σημείο η κατάσταση έχει ως εξής: μία διεργασία στον κόμβο παραλήπτη έχει δεσμεύσει ένα τμήμα μνήμης το οποίο είναι ορατό στους κόμβους του δικτύου. Το πρώτο πράγμα που κάνει ο κόμβος αποστολέας είναι να συνδεθεί στο τμήμα απομακρυσμένης μνήμης που έχει διαθέσει ο παραλήπτης. Πρακτικά, η σύνδεση αποτελείται από την εύρεση της διεύθυνσης και του μεγέθους του απομακρυσμένου τμήματος στο χώρο διευθύνσεων SCI. Η λειτουργία πραγματοποιείται μέσω ειδικής κλήσης συστήματος.

Όταν υπάρχει διαθέσιμο τμήμα μνήμης, αυτό μπορεί να προσπελαστεί με 2 τρόπους: Μπορεί είτε να απεικονιστεί στο χώρο διευθύνσεων μιας διεργασίας και στη συνέχεια να προσπελαστεί σαν κανονική μνήμη, είτε να χρησιμοποιηθεί μέσω της προσέγγισης ΑΠΜ, η οποία αναθέτει τη μεταφορά στον προσαρμογέα SCI.

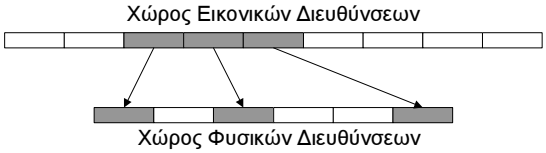
Για την πρώτη περίπτωση, η απεικόνιση ενός τμήματος τοπικής είτε απομακρυσμένης μνήμης πραγματοποιείται μέσω μιας κλήσης συστήματος. Στη συνέχεια η προσπέλαση στο συγκεκριμένο τμήμα μνήμης πραγματοποιείται όπως σε κάθε τμήμα μνήμης που έχει δεσμευθεί

με `malloc()` ή παρόμοιες συναρτήσεις, δηλ. μέσω δεικτών. Η διεργασία μπορεί να διαβάσει από αυτό, να γράψει σε αυτό, να κάνει `memcpy()`, κλπ.

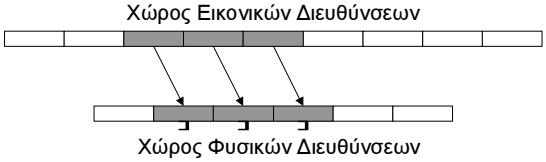
Χρησιμοποιώντας ΑΠΜ, δεν είναι η ΚΜΕ που αντιγράφει δεδομένα από τον ένα κόμβο στον άλλο. Αντ'αυτής, η μεταφορά δεδομένων ανατίθεται στη μηχανή ΑΠΜ που υπάρχει στον προσαρμογέα SCI. Αυτή η προσέγγιση επιτρέπει στην ΚΜΕ να κάνει χρήσιμες εργασίες κατά τη διάρκεια της μεταφοράς. Όμως, επειδή η αρχικοποίηση της μηχανής ΑΠΜ δεν πραγματοποιείται σε επίπεδο χρήστη, αλλά σε επίπεδο πυρήνα μέσω κλήσης συστήματος, ο χρόνος που απαιτείται για την αρχικοποίηση της μηχανής ΑΠΜ είναι μεγαλύτερος από μία πρόσβαση μέσω Προγραμματιζόμενης Ε/Ε.

3.7.5.1 Δέσμευση μνήμης

Η δέσμευση μνήμης δεν πραγματοποιείται μέσω των «κλασσικών» εντολών (π.χ. `malloc()`) ώστε το Λ.Σ., μέσω του οδηγού συσκευής, να μεριμνήσει ειδικά για τις συγκεκριμένες περιπτώσεις. Μέσω των απλών συναρτήσεων δέσμευσης μνήμης, η μνήμη που δεσμεύεται είναι συνεχόμενη στο χώρο των εικονικών (γραμμικών) διευθύνσεων, αλλά ενδέχεται να μην είναι συνεχόμενη στο χώρο των φυσικών διευθύνσεων (βλ. Σχήμα 3.16). Χρησιμοποιώντας ειδικές



Σχήμα 3.16: Δέσμευση 12KB μνήμης τα οποία δεν είναι συνεχόμενα στη φυσική μνήμη. Κάθε σελίδα έχει μέγεθος 4KB.



Σχήμα 3.17: Δέσμευση 12KB μνήμης τα οποία είναι συνεχόμενα στη φυσική μνήμη. Οι αντίστοιχες σελίδες είναι αγκιστρωμένες, με αποτέλεσμα να μην μπορούν να γίνουν swap out στο δίσκο.

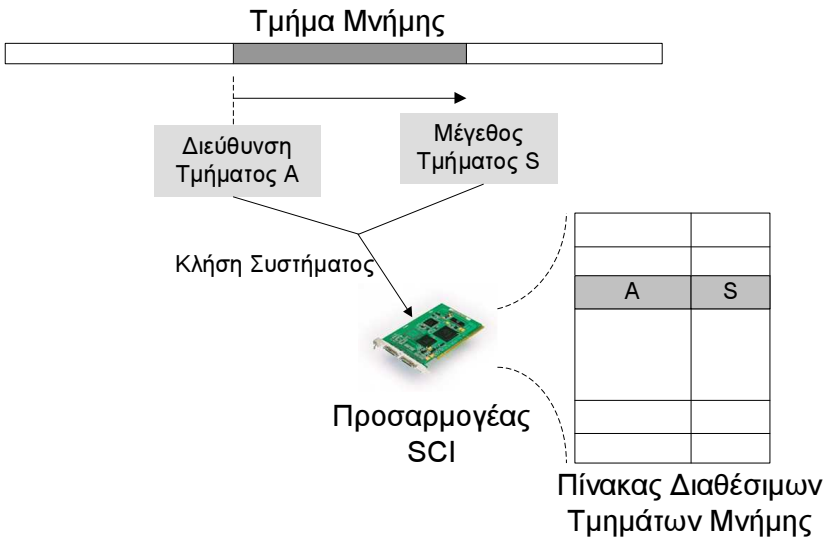
κλήσεις δέσμευσης μνήμης του Λ.Σ., ο δεσμευόμενος χώρος είναι συνεχόμενος στη φυσική μνήμη (βλ. Σχήμα 3.17). Με τη συγκεκριμένη μέθοδο, το τμήμα δεσμευμένης μνήμης μπορεί να χαρακτηριστεί με ένα ζευγάρι τιμών (διεύθυνση τμήματος, μέγεθος τμήματος), κάνοντας πιο εύκολη τη διαχείριση. Ένα ζευγάρι (εικονική διεύθυνση τμήματος, μέγεθος τμήματος) μετατρέπεται εύκολα στο αντίστοιχο ζευγάρι (φυσική διεύθυνση τμήματος, μέγεθος τμήματος). Διαφορετικά, για το σχήμα του Σχήματος 3.16, το ζευγάρι (εικονική διεύθυνση τμήματος, μέγεθος τμήματος) θα αντιστοιχεί σε 3 διαφορετικά ζευγάρια (φυσική διεύθυνση τμήματος #1/#2/#3, μέγεθος σελίδας).

Η μνήμη που δεσμεύεται με την εντολή `malloc()` μπορεί να γίνει swap out στο σκληρό δίσκο, σε περίπτωση που η φυσική μνήμη δεν είναι αρκετή για να ικανοποιήσει όλες τις απαιτήσεις όλων των χρηστών του συστήματος. Μέσω της χρήσης ειδικών συναρτήσεων, οι

σελίδες που δεσμεύονται, αγκιστρώνονται (γίνονται pin-down), μέσω της αφαίρεσής τους από το σύνολο των σελίδων που είναι διαθέσιμες για swap.

3.7.5.2 Δημοσίευση στο χώρο SCI

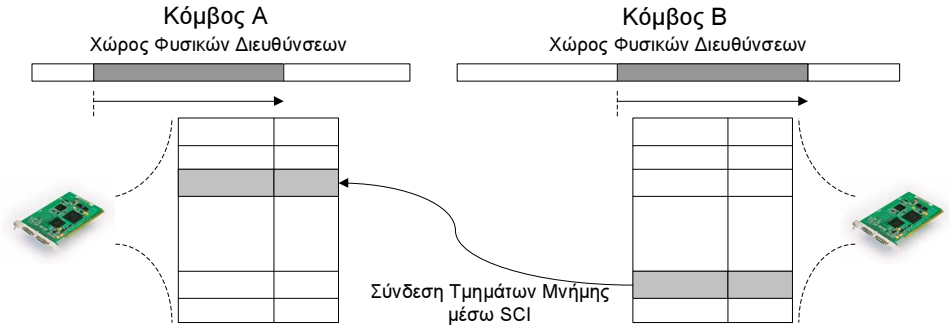
Έχοντας απλώς δεσμεύσει ένα τμήμα συνεχόμενης φυσικής μνήμης, το τμήμα αυτό δεν είναι ορατό από τους υπόλοιπους κόμβους SCI. Πρέπει να γίνει ένας διαχωρισμός των τμημάτων μνήμης που είναι διαθέσιμα για επικοινωνία, από αυτά τα οποία δεν είναι και απλώς υπάρχουν ως απλή, δεσμευμένη από τις διεργασίες μνήμη. Για να γίνει ορατό ένα τμήμα μνήμης, θα πρέπει ο προσαρμογέας SCI να αποκτήσει την πληροφορία το συγκεκριμένο τμήμα. Αυτό πραγματοποιείται μέσω ειδικής κλήσης συστήματος η οποία μεταφέρει στον προσαρμογέα τη διεύθυνση και το μέγεθος του τμήματος μνήμης που είναι διαθέσιμο στο χώρο SCI (βλ. Σχήμα 3.18).



Σχήμα 3.18: Μεταφορά της κατάλληλης πληροφορίας στον προσαρμογέα SCI, ώστε ένα τμήμα μνήμης να γίνει διαθέσιμο στο χώρο διευθύνσεων SCI.

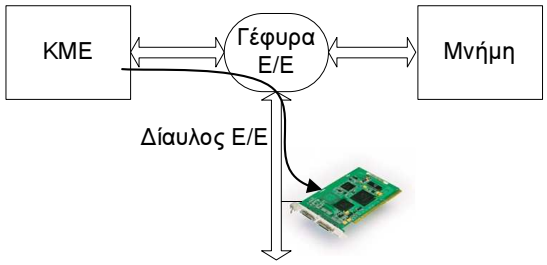
3.7.5.3 Σύνδεση Τμημάτων Μνήμης μέσω SCI

Με την προηγούμενη διαδικασία, ένας κόμβος έκανε διαθέσιμο ένα τμήμα της μνήμης του. Ένας άλλος κόμβος που επιθυμεί να συνδεθεί σε αυτό, χρησιμοποιεί μία κλήση συστήματος μέσω της οποίας επιτυγχάνεται η συγκεκριμένη σύνδεση (βλ. Σχήμα 3.19). Αν στον κόμβο B γίνει μία προσπέλαση στο τμήμα μνήμης που φαίνεται στο Σχήμα 3.19, τότε η προσπέλαση αντί να πραγματοποιηθεί στη φυσική μνήμη του κόμβου B, πραγματοποιείται στον προσαρμογέα SCI του κόμβου B. Αυτή η απεικόνιση είναι δυνατή μέσω της διαδικασίας memory mapped I/O,



Σχήμα 3.19: Σύνδεση δύο τμημάτων μνήμης, που ανήκουν σε διαφορετικούς κόμβους, μέσω του δικτύου SCI.

κατά την οποία προγραμματίζονται κατάλληλα ο προσαρμογέας SCI στο δίαυλο Ε/Ε, καθώς και η γέφυρα μεταξύ του διαύλου συστήματος και του διαύλου Ε/Ε.



Σχήμα 3.20: Η προσπέλαση σε συσκευή η οποία έχει γίνει memory mapped γίνεται μέσω κλασσικών εντολών προσπέλασης μνήμης. Για τη διαδικασία memory mapped I/O προγραμματίζονται ειδικά η συσκευή καθώς και η γέφυρα Ε/Ε.

3.7.5.4 Προσπέλαση Τμημάτων Μνήμης

Οι δύο κόμβοι που λαμβάνουν μέρος στην επικοινωνία έχουν δεσμεύσει δύο τμήματα στους δικούς του χώρους διευθύνσεων. Ο ένας κόμβος κάνει διαθέσιμο ένα τμήμα της μνήμης του προς τους υπόλοιπους κόμβους SCI, ενώ ο άλλος κόμβος συνδέει ένα δικό του τμήμα του χώρου διευθύνσεων με το τμήμα του πρώτου.

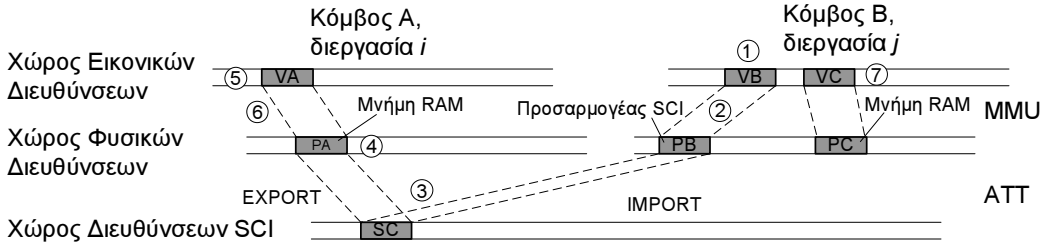
Για να πραγματοποιηθεί η επικοινωνία δύο διεργασιών που βρίσκονται στους δύο κόμβους αντίστοιχα, πρέπει οι διεργασίες να προσπελάσουν τα συγκεκριμένα τμήματα. Όμως, οι διεργασίες δεν μπορούν να προσπελάσουν άμεσα το χώρο φυσικών διευθύνσεων. Η προσπέλαση γίνεται μέσω των εικονικών διευθύνσεων κάθε διεργασίας, οι οποίες μέσω της συνεργασίας του Λ.Σ. και του Συστήματος Διαχείρισης Μνήμης (MMU) του επεξεργαστή, μεταφράζονται σε φυσικές. Συνεπώς, ο κόμβος Β ζητάει από το Λ.Σ. να αντιστοιχίσει ένα τμήμα του εικονικού χώρου διευθύνσεων με το ζητούμενο τμήμα χώρου φυσικών διευθύνσεων. Το Λ.Σ. ενημερώνει

τους πίνακες σελίδων κατάλληλα, έτσι ώστε όταν ο επεξεργαστής λάβει μία εικονική διεύθυνση από την εκτελούμενη διεργασία, να τη μεταφράσει στην αντίστοιχη φυσική, από όπου προκύπτει η προσπέλαση της συσκευής και κατ'επέκταση η προσπέλαση του απομακρυσμένου τμήματος μνήμης. Την ίδια διαδικασία ακολουθεί και η διεργασία στον κόμβο Α, ώστε να έχει τη δυνατότητα να προσπελαύνει το τμήμα της δική της μνήμης. Αυτή είναι η προσπέλαση με Προγραμματιζόμενη Ε/Ε.

Έστω ότι η διεργασία τον κόμβο Β πραγματοποιήσει μία εγγραφή στην απεικονισμένη εικονική της μνήμη. Μέσω του MMU της ΚΜΕ, η προσπέλαση γίνεται σε μία φυσική διεύθυνση. Καθώς η διεύθυνση βρίσκεται στο δίαυλο ΚΜΕ-μνήμης, η γέφυρα Ε/Ε αναγνωρίζει ότι η διεύθυνση εξυπηρετείται από τον προσαρμογέα SCI και την προωθεί σε αυτόν. Ο προσαρμογέας, μέσω του πίνακα αντιστοίχισης Address Translation Table (ATT), γνωρίζει ότι η προσπέλαση αυτή πρέπει να προωθηθεί στον κόμβο Α για να εξυπηρετηθεί. Φτάνοντας η αίτηση στον κόμβο Α, εξυπηρετείται από τον αντίστοιχο προσαρμογέα. Ο προσαρμογέας χρησιμοποιώντας ΑΠΜ (χωρίς να διακόψει την ΚΜΕ του κόμβου Α) πραγματοποιεί τη ζητούμενη προσπέλαση. Αν η προσπέλαση είναι ανάγνωση, τότε τα δεδομένα που έλαβε ο προσαρμογέας Α, αποστέλλονται στον προσαρμογέα Β μέσω του δικτύου και δίνονται στην εφαρμογή. Οι διαφορετικοί χώροι διευθύνσεων, καθώς και οι απεικονίσεις μεταξύ τους για την πραγματοποίηση επικοινωνίας, φαίνονται στο Σχήμα 3.15.

Αν η διεργασία στον κόμβο Β χρησιμοποιεί αποστολή ή λήψη με ΑΠΜ, τότε η διαδικασία στον κόμβο Α γίνεται όπως και στην προηγούμενη περίπτωση. Όμως στον κόμβο Β δεν είναι αναγκαία η απεικόνιση στον εικονικό χώρο διευθύνσεων, αφού πλέον δεν θα χρησιμοποιηθούν εντολές load και store για την προσπέλαση του τμήματος μνήμης. Αντ'αυτών θα χρησιμοποιηθεί ειδική κλήση συστήματος, στην οποία θα δοθεί ως παράμετρος ένα χειριστήριο του συγκεκριμένου τμήματος μνήμης. Η μηχανή ΑΠΜ του προσαρμογέα αναλαμβάνει να μεταφέρει τα δεδομένα από (ή προς) το συνδεδεμένο τμήμα μνήμης του κόμβου Β.

Έστω ότι η διεργασία j στον κόμβο Β (βλ. Σχήμα 3.21) πραγματοποιεί μία εγγραφή στον τμήμα VB του χώρου εικονικών διευθύνσεων (1). Η εικονική διεύθυνση στην οποία γίνεται η εγγραφή, μεταφράζεται σε φυσική διεύθυνση (2) μέσω της μονάδας διαχείρισης διευθύνσεων του επεξεργαστή (MMU) και στη συνέχεια εμφανίζεται στο δίαυλο επεξεργαστή-μνήμης του συστήματος. Η γέφυρα Ε/Ε, που συνδέει το δίαυλο επεξεργαστή-μνήμης με το δίαυλο Ε/Ε, αναγνωρίζει τη συγκεκριμένη φυσική διεύθυνση και την περνάει στο δίαυλο Ε/Ε, όπου χρησιμοποιείται από τον προσαρμογέα SCI. Σύμφωνα με τον πίνακα μετάφρασης διευθύνσεων (ATT) που υπάρχει στον προσαρμογέα, η διεύθυνση, στην οποία έγινε η εγγραφή, ανήκει στον κόμβο Α και συγκεκριμένα στο τμήμα διευθύνσεων PA. Ο προσαρμογέας SCI δημιουργεί ένα πακέτο δικτύου, στο οποίο ενσωματώνει την εντολή προσπέλασης, και το στέλνει στον κόμβο Α (3). Όταν ο προσαρμογέας του κόμβου Α λάβει το πακέτο, εκτελεί την εντολή που περιλαμβάνεται



Σχήμα 3.21: Παράδειγμα επικοινωνία Προγραμματιζόμενης Ε/Ε και Άμεσης Προσπέλασης Μνήμης πάνω από δίκτυο SCI. Οι αριθμοί σε κύκλο ορίζουν τη σειρά εκτέλεσης των λειτουργιών και επεξηγούνται στο κείμενο.

σε αυτό. Συγκεκριμένα, χρησιμοποιώντας τη μηχανή ΑΠΜ που διαθέτει, αποθηκεύει την τιμή της εγγραφής στη διεύθυνση του χώρου PA (4). Επειδή ο χώρος φυσικών διευθύνσεων PA αντιστοιχεί σε μνήμη RAM, η εγγραφή περνάει από το δίαυλο Ε/Ε, μέσω της γέφυρας Ε/Ε, στο δίαυλο επεξεργαστή-μνήμης και καταλήγει στη μνήμη RAM.

Αν θελήσει η διεργασία i να διαβάσει την τιμή που γράφτηκε στη μνήμη RAM, δίνει μία εντολή ανάγνωσης στη αντίστοιχη θέση του χώρου εικονικών διευθύνσεων της VA (5). Η διεύθυνση της προσπέλασης μεταφράζεται από τη μονάδα διαχείρισης διευθύνσεων και περνάει στο δίαυλο επεξεργαστή-μνήμης (6). Στη συνέχεια, σαν κλασσική εντολή ανάγνωσης από τη μνήμη, το αποτέλεσμα της ανάγνωσης επιστρέφει στην εντολή.

Μέσα από αυτή τη διαδικασία, η τιμή η οποία γράφτηκε από τη διεργασία j στον κόμβο Β, γίνεται γνωστή στη διεργασία i στον κόμβο Α, επιτυγχάνοντας την επικοινωνία. Πρέπει να σημειωθεί ότι κατά τη διάρκεια όλης της επικοινωνίας, δεν χρειάστηκε η παρέμβαση του Λ.Σ. σε κανέναν από τους κόμβους. Οι εγγραφές και αναγνώσεις στους δύο κόμβους έγιναν σε επίπεδο χρήστη, με τη μεσολάβηση μόνο του υλικού. Ο συγκεκριμένος τρόπος επικοινωνίας παρουσιάζει μικρή καθυστέρηση.

Στη συνέχεια εξετάζουμε τι συμβαίνει στην περίπτωση που ο κόμβος Β στέλνει πολλά δεδομένα στον κόμβο Α. Χρησιμοποιώντας την Προγραμματιζόμενη Ε/Ε, η διεργασία στον κόμβο Β θα πρέπει να αποστείλει ανά ένα όλα τα δεδομένα στον κόμβο Α.

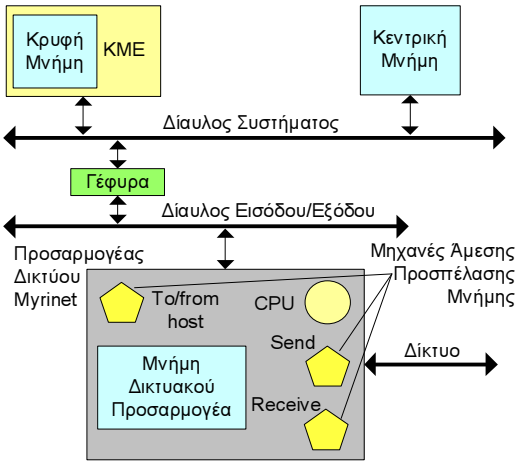
Αν θεωρήσουμε ότι τα δεδομένα βρίσκονται αποθηκευμένα στο τμήμα μνήμης PB του κόμβου Β, τότε μπορεί να χρησιμοποιηθεί η μέθοδος Απομακρυσμένης Άμεσης Προσπέλασης Μνήμης (RDMA) για την αποστολή των δεδομένων. Σύμφωνα με αυτόν, η διεργασία j προγραμματίζει τη μηχανή ΑΠΜ που βρίσκεται στον προσαρμογέα SCI, περνώντας του ως παραμέτρους δύο χειριστήρια (7), που αντιστοιχούν στην πηγή (PC), στον προορισμό (PB) καθώς και το μέγεθος των δεδομένων. Στη συνέχεια, ο προσαρμογέας SCI αναλαμβάνει με ΑΠΜ, να διαβάσει τα δεδομένα από τη μνήμη PC και να τα μεταφέρει στον προορισμό (PA), ακολουθώντας τη διαδικασία που περιγράφηκε προηγουμένως. Κατά τη διάρκεια της αντιγραφής

και μεταφοράς πάνω από το δίκτυο, οι διεργασίες i και j δεν ασχολούνται με τη μεταφορά, αλλά μπορούν να εκτελούν άλλες εργασίες. Σε αυτή την περίπτωση, ο προγραμματισμός της μηχανής ΑΠΜ του κόμβου Β από τη διεργασία j χρειάστηκε να εκτελεστεί σε επίπεδο πυρήνα, αυξάνοντας την καθυστέρηση της επικοινωνίας. Όμως, μετά τον προγραμματισμό, η διεργασία j μπορούσε να εκτελέσει άλλες εργασίες, έχοντας αφήσει το υλικό δικτύου να αναλάβει όλη την επικοινωνία.

3.8 Myrinet

Το Myrinet [BCF+95, CMC98] είναι ένα δίκτυο μεταγώγισης (switched), τεχνολογίας Gigabit ανά δευτερόλεπτο που χρησιμοποιεί πακέτα μεταβλητού μεγέθους. Τα πακέτα δρομολογούνται μέσα από ένα δίκτυο αξιόπιστων συνδέσμων και μεταγωγέων crossbar. Η δρομολόγηση τύπου «σκουληκότρυπας» (wormhole routing) που χρησιμοποιεί το Myrinet απαντάται συνήθως σε υπερυπολογιστές, γεγονός που φανερώνει τα προηγμένα χαρακτηριστικά του.

Στο Σχήμα 3.22, φαίνεται η αρχιτεκτονική ενός κόμβου σε μια συστοιχία Myrinet. Κάθε κόμβος περιέχει ένα δικτυακό προσαρμογέα που περιέχει έναν επεξεργαστή και κάποια ποσότητα μνήμης, η οποία χρησιμοποιείται για την αποθήκευση του προγράμματος ελέγχου και των δεδομένων του προσαρμογέα. Ο προσαρμογέας δικτύου συνδέεται στο δίαυλο Εισόδου/Εξόδου—μια τυπική διάταξη για κοινό υλικό.



Σχήμα 3.22: Η αρχιτεκτονική του κόμβου και του προσαρμογέα δικτύου Myrinet.

Το Myrinet απαιτεί όλα τα πακέτα να περνάνε από τη μνήμη του προσαρμογέα δικτύου και κατά την αποστολή και κατά τη λήψη. Χρησιμοποιεί γρήγορες (και ακριβές) μνήμες στατικής RAM, οι οποίες περιορίζουν το μέγεθος της συνολικής μνήμης. Για να προσπελάσουν δεδομένα στη μνήμη του άλλου, ο κόμβος και ο προσαρμογέας χρησιμοποιούν ΑΠΜ. Ο κόμβος έχει τη

δυνατότητα να προσπελάσει τη μνήμη του προσαρμογέα μέσω προγραμματιζόμενης Ε/Ε.

Το Myrinet είναι η πιο διαδεδομένη δικτυακή τεχνολογία που περιέχει προγραμματιζόμενο επεξεργαστή στον προσαρμογέα. Αν και δίνεται μεγαλύτερη ευελιξία στους σχεδιαστές πρωτοκόλλων, λόγω της χαμηλής ταχύτητας της ΚΜΕ του προσαρμογέα σε σχέση με την κεντρική ΚΜΕ του συστήματος, μπορεί να πραγματοποιεί μόνο απλές λειτουργίες. Προσθέτοντας μερικές εντολές μόνο στο κρίσιμο μονοπάτι του προγράμματος ελέγχου του Myrinet, αυξάνεται σημαντικά η καθυστέρηση από άκρο-σε-άκρο (end-to-end latency).

Μερικά τυπικά χαρακτηριστικά ενός προσαρμογέα Myrinet είναι τα εξής:

- Κεντρική μονάδα επεξεργασίας με σύνολο εντολών μειωμένου ρεπερτορίου (RISC) με συχνότητα λειτουργίας 133MHz.
- Μνήμη τεχνολογίας SRAM μεγέθους 2MB
- Σύνδεσμοι οπτικών ινών
- Ταχύτητα συνδέσμων 2×1.28Gbps
- Σύνδεση με την ΚΜΕ μέσω διαύλου PCI 64 bit/66 MHz

Ο βασικός τρόπος επικοινωνίας γίνεται με τη χρήση της βιβλιοθήκης GM και ενός στρώματος λογισμικού στον πυρήνα του λειτουργικού. Πάνω από το GM έχουν υλοποιηθεί διάφορες γνωστές βιβλιοθήκες και στρώματα επικοινωνίας όπως TCP/IP, Active Messages, MPI, κλπ.

3.9 Λογισμικό

3.9.1 Software Infrastructure for SCI

Για την αποδοτική εκμετάλλευση της επικοινωνίας μεταξύ κόμβων διασυνδεδεμένων με SCI, ήταν αναγκαία η ανάπτυξη μιας σταθερής και πλούσιας υποδομής λογισμικού. Οι επιμέρους κόμβοι έπρεπε να παρουσιάζονται στο χρήση ως ένα μοναδικό μηχάνημα, λαμβάνοντας υπόψη τις αδυναμίες που οφείλονται στα λειτουργικά συστήματα. Τα λειτουργικά συστήματα αντιμετωπίζουν κάθε κόμβο μιας συστοιχίας ως μοναδική οντότητα και αγνοούν την διαλειτουργικότητα με τους υπόλοιπους κόμβους της συστοιχίας.

Το έργο Standard Software Infrastructure for SCI-based Parallel Systems (SISCI) [EHSH97], που χρηματοδοτήθηκε από την Ευρωπαϊκή Ένωση, αντιμετώπισε αυτό ακριβώς το πρόβλημα. Στα πλαίσια του έργου δημιουργήθηκε ένα λεπτό στρώμα λογισμικού, το οποίο μπορούσε να χρησιμοποιηθεί είτε απευθείας για δικτυακό προγραμματισμό, μέσω της χρήσης μιας βιβλιοθήκης, είτε για να υλοποιηθούν πάνω από αυτό άλλα πιο βολικά και εύχρηστα στρώματα επικοινωνίας, όπως βιβλιοθήκες BSD sockets, PVM, MPI, Active Messages, Fast Messages, κ.α.

Το SCI είναι μία μοντέρνα δικτυακή τεχνολογία με δυνατότητα επικοινωνίας στο επίπεδο χρήστη. Προκειμένου να επικοινωνήσουν δύο εφαρμογές μέσω SCI, πραγματοποιούν μια συγκεκριμένη διαδικασία αρχικοποίησης, που παρουσιάστηκε στην παράγραφο 3.7.5. Η διαδικασία περιλαμβάνει την χρήση υπηρεσιών του δικτύου SCI, που παρέχονται από τον οδηγό συσκευής που έχει ενσωματωθεί στον πυρήνα του Λ.Σ. Για να μπορέσουν οι εφαρμογές να χρησιμοποιήσουν τις συγκεκριμένες υπηρεσίες, απαιτείται ένα στρώμα λογισμικού επιπέδου χρήστη που να καλεί τις αντίστοιχες ρουτίνες του πυρήνα. Αυτό το στρώμα λογισμικού είναι το SISCI. Πραγματοποιεί μια σχεδόν «1-1»μετάφραση των εντολών που δίνονται σε αυτό, σε εντολές για τον οδηγό συσκευής του SCI. Παραδείγματα επικοινωνίας SCI Προγραμματιζόμενης Ε/Ε και ΑΠΜ, χρησιμοποιώντας το λογισμικό SISCI, παρουσιάζονται στο Παράρτημα Γ.

3.9.2 Message Passing Interface – MPI

Το MPI έχει γίνει το de facto πρότυπο εργαλείο για τον προγραμματισμό παράλληλων αρχιτεκτονικών. Η προσπάθεια προτυποποίησης του MPI ξεκίνησε από τις κυβερνητικές υπηρεσίες των ΗΠΑ και στη συνέχεια υιοθετήθηκε από τους χρήστες και τους κατασκευαστές συστημάτων υψηλών επιδόσεων, για τον παράλληλο προγραμματισμό επιστημονικών υπολογισμών. Το MPI δημιουργήθηκε βασιζόμενο στην εμπειρία άλλων συστημάτων ανταλλαγής μηνυμάτων, όπως το Parallel Virtual Machine (PVM). Ανάμεσα στους στόχους του MPI ήταν η δημιουργία μιας μεταφέρσιμης (portable) διεπιφάνειας για παράλληλο προγραμματισμό, που να προσφέρει τη μέγιστη απόδοση. Το MPI προσφέρει ένα μοναδικό συνδυασμό αφαίρεσης (abstraction), απόδοσης και μεταφερσιμότητας. Όλα αυτά τα χαρακτηριστικά συνεισέφεραν στην επιτυχία του MPI σαν το πρότυπο περιβάλλον παράλληλου προγραμματισμού. Τα τελευταία χρόνια, οι χρήστες του MPI έχουν επεκταθεί ώστε να συμπεριλάβουν και τους χρήστες της ταχύτατα αναπτυσσόμενης περιοχής των συστοιχιών υπολογιστών.

Το MPI παρέχει τη δυνατότητα στους προγραμματιστές παράλληλων εφαρμογών να δημιουργούν αλγόριθμους που μπορούν εύκολα να μεταφερθούν σε διαφορετικές πλατφόρμες, διατηρώντας την υψηλή απόδοση. Η συγκεκριμένη δυνατότητα βασίζεται στη μεταφέρσιμη διεπαφή, η οποία μπορεί να βελτιστοποιηθεί σε διαφορετικά αρχιτεκτονικά συστήματα, αξιοποιώντας τα δυνατά σημεία που παρέχουν. Αν αυτές οι βελτιστοποιήσεις μεταφερθούν στο λογισμικό της εφαρμογής, τότε ο κώδικας δεν θα είναι πλέον μεταφέρσιμος. Το γεγονός αυτό είναι λιγότερο δημοφιλές από τη μεταφερσιμότητα, αλλά βοηθάει στο να γίνεται προσπάθεια βελτιστοποιήσεων που είναι στενά συνδεδεμένες σε μία μόνο αρχιτεκτονική συστήματος.

Η επιτυχία του MPI οδήγησε στην επέκταση των προδιαγραφών του αρχικού προτύπου και στη δημιουργία του MPI-2. Το MPI-2 παρέχει μεγαλύτερη λειτουργικότητα από το MPI, περιλαμβάνοντας παράλληλη Ε/Ε από και προς αρχεία, περισσότερες συλλογικές λειτουργίες, μονόπλευρη επικοινωνία (one-sided communication) και δυναμική διαχείριση διεργασιών. Η

δυναμική διαχείριση διεργασιών αντιμετωπίζει τον περιορισμό του MPI που σχετίζεται με το στατικό μοντέλο για δημιουργία διεργασιών. Το MPI-2 επιτρέπει στις εργασίες MPI να αυξάνονται, δίνοντας τη δυνατότητα για δυναμική δημιουργία νέων διεργασιών. Η συγκεκριμένη δυνατότητα είναι χρήσιμη σε πολλές εφαρμογές, όπως π.χ. σε περιπτώσεις αλλαγής ρυθμού των δεδομένων εισόδου.

Επειδή υπάρχουν μόνο εμπορικές υλοποιήσεις του προτύπου MPI-2, στη συνέχεια αναφερόμαστε μόνο στο MPI-1.2, το οποίο έχει υλοποίηση με λογισμικό ανοιχτού κώδικα (MPICH) για πάρα πολλές πλατφόρμες. Επίσης, η συντριπτική πλειοψηφία των προγραμμάτων MPI χρησιμοποιεί το πρότυπο MPI-1.2.

3.9.2.1 Δυνατότητες του MPI

Ένα από τα δυνατά σημεία του MPI είναι ότι υποστηρίζει 125 διαφορετικές συναρτήσεις, οι οποίες προσφέρουν στους χρήστες του ένα μεγάλο πλήθος δυνατοτήτων. Ταυτόχρονα, όμως, πολλά προγράμματα στο MPI μπορούν να γραφτούν με τη χρήση μόνο 6 βασικών συναρτήσεων. Οι πλειοψηφία των χρηστών του MPI δημιουργούν προγράμματα βασιζόμενοι σε ένα μικρό υποσύνολο των συναρτήσεων που υποστηρίζονται από το πρότυπο. Για το λόγο αυτό, υιοθετείται γρήγορα και από επιστήμονες που το αντικείμενό τους δεν είναι η επιστήμη των υπολογιστών, όπως φυσικοί, χημικοί, βιολόγοι, κλπ.

Το πρότυπο του MPI παρέχει τις εξής δυνατότητες:

- Επικοινωνία σημείο-προς-σημείο
- Λειτουργίες συλλογικής επικοινωνίας
- Ομαδοποίηση διεργασιών
- Επικοινωνία domain
- Τοπολογίες διεργασιών
- Διεπαφή για profiling
- Δυνατότητα για χρήση γλωσσών Fortran 77 και C

Για να γίνει καλύτερη αξιολόγηση των δυνατοτήτων που προσφέρει το πρότυπο του MPI, αναφέρουμε επιπλέον τι δεν ορίζεται από το πρότυπο:

- Ρητή υποστήριξη μοιραζόμενης μνήμης
- Εργαλεία για την κατασκευή προγραμμάτων
- Εργαλεία για αποσφαλμάτωση
- Υποστήριξη για νήματα εκτέλεσης (threads)

Κατάσταση Επικοινωνίας	Συναρτήσεις
Standard	`MPI_Send` και `MPI_Recv`
Buffered	`MPI_Bsend`
Synchronous	`MPI_Ssend`
Ready	`MPI_Rsend`

Πίνακας 3.3: Οι καταστάσεις επικοινωνίας που υποστηρίζονται από το πρότυπο του MPI.

- Υποστήριξη για διαχείριση εργασιών (συμπεριλήφθηκε στο MPI-2)
- Συναρτήσεις Ε/Ε (συμπεριλήφθηκε στο MPI-2)

Επικοινωνία Σημείο-προς-σημείο Επικεντρωνόμαστε στη σημαντικότερη δυνατότητα που προσφέρει το MPI που είναι η επικοινωνία. Ενώ οι προγραμματιστές παράλληλων εφαρμογών μπορούν να χρησιμοποιήσουν άλλου είδους πλατφόρμες για την επίτευξη της επικοινωνίας (π.χ. διεπαφή sockets), η συγγραφή τέτοιων προγραμμάτων είναι επίπονη και απαιτεί σημαντική εμπειρία. Το MPI προσφέρει τις συναρτήσεις `MPI_Send` και `MPI_Recv` μέσω των οποίων δύο διεργασίες μπορούν να επικοινωνήσουν, αποφεύγοντας τις λεπτομέρειες της επικοινωνίας.

Για πιο εξειδικευμένες περιπτώσεις, το MPI προσφέρει και άλλες καταστάσεις επικοινωνίας, εκτός της προκαθορισμένης (standard), οι οποίες φαίνονται στον Πίνακα 3.3, των οποίων η σημασία αναλύεται παρακάτω. Όσον αφορά τις λήψεις δεδομένων, υποστηρίζεται μόνο μία συνάρτηση (`MPI_Recv`), όποια και αν είναι η συνάρτηση με την οποία έγινε η αποστολή. Παραδείγματα προγραμματισμού παράλληλων εφαρμογών σε MPI παρουσιάζονται στο Παράρτημα Α.

3.9.2.2 Καταστάσεις Επικοινωνίας

Οι κλήσεις επικοινωνίας που περιγράφηκαν προηγουμένως χρησιμοποιούν την κατάσταση επικοινωνίας standard. Σε αυτή την κατάσταση, η υλοποίηση του MPI αποφασίζει αν τα προς αποστολή μηνύματα θα αντιγραφούν προσωρινά (buffered) ή όχι. Στην περίπτωση που το MPI επιλέξει την προσωρινή αντιγραφή, η κλήση αποστολής μπορεί να ολοκληρωθεί, πριν κληθεί η αντίστοιχη συνάρτηση λήψης. Όμως, υπάρχει και η περίπτωση στην οποία δεν υπάρχει διαθέσιμος χώρος για προσωρινή αποθήκευση, ή το MPI μπορεί να επιλέξει να μην αποθηκεύσει προσωρινά τα μηνύματα, για λόγους απόδοσης. Σε αυτή την περίπτωση, η κλήση αποστολής δεν θα τερματιστεί έως ότου μια αντίστοιχη κλήση λήψης έχει κληθεί από τον παραλήπτη και τα δεδομένα έχουν μεταφερθεί σε αυτόν. Συνεπώς, μια αποστολή σε κατάσταση standard μπορεί να ξεκινήσει, ανεξαρτήτως αν στον παραλήπτη έχει κληθεί συνάρτηση λήψης.

Όμοια, μία λειτουργία αποστολής κατάστασης buffered μπορεί να ξεκινήσει, ανεξαρτήτως του αν στον παραλήπτη έχει κληθεί αντίστοιχη συνάρτηση λήψης. Ενδεχομένως να ολοκληρωθεί

πριν στον παραλήπτη κληθεί συνάρτηση λήψης. Η διαφορά της με την κατάσταση standard είναι ότι το αν ολοκληρωθεί ή όχι η συνάρτηση, εξαρτάται μόνο από τον τοπικό κόμβο (local completion semantics). Επίσης, στη συγκεκριμένη περίπτωση, η προσωρινή μνήμη πρέπει να δοθεί από την εφαρμογή. Η προσωρινή μνήμη, που καταλαμβάνει το μήνυμα, ελευθερώνεται όταν το μήνυμα φτάσει στον προορισμό του.

Και στην περίπτωση της κατάστασης synchronous στην αποστολή μηνύματος, η λειτουργία μπορεί να ξεκινήσει ανεξαρτήτως από κλήση αντίστοιχης συνάρτησης λήψης στον παραλήπτη. Όμως, η αποστολή θα ολοκληρωθεί επιτυχώς μόνο αν κληθεί συνάρτηση λήψης και η λειτουργία της λήψης έχει αρχίσει να λαμβάνει το μήνυμα που στάλθηκε από τη σύγχρονη αποστολή. Αυτό σημαίνει ότι με την ολοκλήρωση μιας σύγχρονης αποστολής, η προσωρινή μνήμη μπορεί να επαναχρησιμοποιηθεί, αλλά και ότι ο παραλήπτης φτάνει σε κάποιο συγκεκριμένο σημείο της εκτέλεσής του, δηλ. έχει ξεκινήσει την εκτέλεση της λήψης. Η κατάσταση synchronous, προσφέρει δυνατότητα σύγχρονης επικοινωνίας: η επικοινωνία δεν περατούται σε κανένα από τα δύο άκρα, πριν οι δύο διεργασίες συνεννοηθούν (rendezvous). Είναι προφανές ότι η ολοκλήρωση της κλήσης δεν εξαρτάται μόνο από τον τοπικό κόμβο.

Μία αποστολή κατάστασης ready μπορεί να ξεκινήσει, μόνο αν μια αντίστοιχη συνάρτηση έχει ήδη κληθεί. Διαφορετικά, η λειτουργία επιστρέφει σφάλμα. Σε μερικά συστήματα, η κατάσταση αυτή επιτρέπει την παράκαμψη της συνεννόησης με αποτέλεσμα τη βελτίωση της απόδοσης.

Οι συναρτήσεις που περιγράφηκαν, ονομάζονται στο πρότυπο του MPI ως συναρτήσεις blocking. Αυτό σημαίνει ότι οι συγκεκριμένες κλήσεις πραγματοποιούν ολόκληρη τη διαδικασία επικοινωνίας πριν επιτραπεί η εκτέλεση των επόμενων εντολών του προγράμματος. Επιπλέον, το MPI ορίζει και την κατάσταση immediate για τις παραπάνω συναρτήσεις. Το νόημα της κατάστασης αυτής είναι να προγραμματιστούν οι εργασίες που πρέπει να γίνουν από τη συνάρτηση, ενώ η εκτέλεση του κώδικα συνεχίζεται αμέσως, χωρίς να δίνονται εγγυήσεις για το σημείο στο οποίο φτάνει η εκτέλεση της συνάρτησης. Ως συμπλήρωμα στην κατάσταση immediate, παρέχονται συναρτήσεις `MPI_Test` και `MPI_Wait` για τον έλεγχο ολοκλήρωσης των συναρτήσεων επικοινωνίας immediate. Οι κλήσεις αποστολής και λήψης δεδομένων immediate για την κατάσταση standard είναι οι `MPI_Isend` και `MPI_IRecv`.

3.9.3 Προδιαγραφές Πρωτοκόλλου Απομακρυσμένης Άμεσης Προσπέλασης Μνήμης

Η επικοινωνία που πραγματοποιείται πάνω από συμβατικές τεχνολογίες, επιβάλει τη μεσολάβηση της ΚΜΕ του κόμβου προορισμού, απασχολώντας την από άλλες εργασίες που ενδεχομένως εκτελεί. Συγκεκριμένα, η αποστολή ενός μηνύματος σε έναν κόμβο Κ, περιλαμβάνει τη μεσολάβηση της ΚΜΕ του Κ, σε κάποιο κομμάτι της συνολικής διαδικασίας αποθήκευσης του

μηνύματος στη μνήμη. Όμως, οι νέες δικτυακές τεχνολογίες (π.χ. SCI) καθιστούν δυνατή την επικοινωνία ενός άκρου (one-sided communication), κατά την οποία η ΚΜΕ του παραλήπτη κόμβου δεν χρειάζεται να παρεμβληθεί.

Η αναγνώριση της αξίας της επικοινωνίας ενός άκρου και η ύπαρξη αρκετών ιδιωτικών πρωτοκόλλων και υλοποιήσεων κινητοποίησε τις εταιρείες IBM και Hewlett-Packard στον ορισμό των προδιαγραφών του πρωτοκόλλου Απομακρυσμένης Άμεσης Προσπέλασης Μνήμης (Remote DMA Protocol). Το κείμενο περιγράφει ένα πρωτόκολλο ΑΑΠΜ το οποίο παρέχει υπηρεσίες εγγραφής και ανάγνωσης άμεσα στις εφαρμογές και καθιστά δυνατή τη μεταφορά δεδομένων κατευθείαν σε προσωρινή μνήμη των διεργασιών επιπέδου χρήστη, χωρίς ενδιάμεσες αντιγραφές δεδομένων. Αυτό σημαίνει ότι αποφεύγεται η μεσολάβηση του πυρήνα του Λ.Σ.

Σήμερα, οι επικοινωνίες πάνω από το TCP/IP απαιτούν λειτουργίες αντιγραφής, οι οποίες προσθέτουν καθυστέρηση και καταναλώνουν σημαντικούς πόρους της ΚΜΕ και της μνήμης. Το πρωτόκολλο ΑΑΠΜ επιτρέπει την αποφυγή των αντιγραφών και μειώνει τις καθυστερήσεις επιτρέποντας σε μία εφαρμογή να διαβάσει και να γράψει δεδομένα σε μία απομακρυσμένη μνήμη, με ελάχιστες απαιτήσεις σε μνήμη και επεξεργασία, διατηρώντας την προστασία της μνήμης.

3.9.3.1 Αρχιτεκτονικοί Στόχοι

Το πρωτόκολλο ΑΑΠΜ σχεδιάστηκε για τους ακόλουθους βασικούς αρχιτεκτονικούς στόχους υψηλού επιπέδου:

- Να παρέχει τη λειτουργία μεταφοράς δεδομένων που επιτρέπει στο τοπικό άκρο να μεταφέρει μέχρι $2^{32}-1$ bytes κατευθείαν σε μία απομακρυσμένη μνήμη, χωρίς ενδιάμεσες αντιγραφές. Αυτή η λειτουργία αναφέρεται ως Εγγραφή ΑΑΠΜ.

- Να παρέχει τη λειτουργία μεταφοράς δεδομένων που επιτρέπει στο τοπικό άκρο να μεταφέρει μέχρι $2^{32}-1$ bytes κατευθείαν από μία απομακρυσμένη μνήμη, χωρίς ενδιάμεσες αντιγραφές. Αυτή η λειτουργία αναφέρεται ως Ανάγνωση ΑΑΠΜ.

3.10 Σύνοψη

Στο παρόν κεφάλαιο παρουσιάστηκε το μοντέλο της συστοιχίας υπολογιστών ως αρχιτεκτονική παράλληλης επεξεργασίας. Αναλύθηκαν οι λόγοι για τους οποίους οι συστοιχίες υπολογιστών αποτελούν τη δημοφιλέστερη πλατφόρμα εκτέλεσης παράλληλων εφαρμογών.

Στη συνέχεια, έγινε ανάλυση των παραμέτρων απόδοσης των δικτυακών τεχνολογιών (Καθυστέρηση Δικτύου, Δυνατότητα Παροχής και Ρυθμός Παροχής) και δόθηκε η μεταξύ τους σχέση.

Έπειτα αναλύθηκαν οι λειτουργίες που λαμβάνουν χώρα κατά την αποστολή και λήψη ενός πακέτου πάνω από ένα συμβατικό δίκτυο διασύνδεσης. Παρουσιάστηκαν οι μεταφορές και αντιγραφές δεδομένων που πραγματοποιούνται στο δίαυλο συστήματος κατά τη διάρκεια των παραπάνω λειτουργιών, καθώς και οι παρεμβάσεις του Λ.Σ. οι οποίες αυξάνουν την καθυστέρηση της επικοινωνίας.

Έγινε εκτενής αναφορά σε πρωτόκολλα επικοινωνίας επιπέδου χρήστη τα οποία μειώνουν την καθυστέρηση της επικοινωνίας. Αναφέρθηκαν διάφορες υλοποιήσεις που έχουν προταθεί για παράκαμψη του Λ.Σ., διατηρώντας τη λειτουργικότητα που είναι αναγκαία για την επίτευξη της επικοινωνίας.

Αναλύθηκε η Αρχιτεκτονική Εικονικού Προσαρμογέα, ενός συνόλου προδιαγραφών που στόχο έχουν την αποδοτική επικοινωνία σε περιβάλλοντα συστοιχιών υπολογιστών, καθώς και το πρωτόκολλο ΑΑΠΜ για τη μονόπλευρη επικοινωνία. Παρουσιάστηκε ν δικτυακή τεχνολογία SCI, η οποία υιοθετεί πολλά από τα παραπάνω προηγμένα χαρακτηριστικά και θα χρησιμοποιηθεί στις πειραματικές μετρήσεις του κεφαλαίου 5.

Όσον αφορά το προγραμματιστικό μοντέλο, έγινε παρουσίαση του MPI που είναι το πλέον χρησιμοποιούμενο περιβάλλον ανάπτυξης παράλληλων εφαρμογών.

Μελετήθηκαν προηγμένα δίκτυα διασύνδεσης και αρχιτεκτονικές επικοινωνίας που παρέχουν σημαντική μείωση της καθυστέρησης επικοινωνίας, μέσω της παράκαμψης του Λ.Σ. και της μείωσης των αντιγραφών δεδομένων. Επίσης, τα δίκτυα αυτά προσφέρουν τη δυνατότητα ταυτόχρονης χρησιμοποίησης της ΚΜΕ και του δικτυακού συστήματος ενός υπολογιστικού κόμβου, μαζί με τη δυνατότητα μονόπλευρης επικοινωνίας. Στο επόμενο κεφάλαιο θα παρουσιάσουμε μία νέα μέθοδο χρονοδρομολόγησης υπερκόμβων η οποία λαμβάνει υπόψη της τα παραπάνω προηγμένα χαρακτηριστικά επικοινωνίας και βελτιώνει την απόδοση των παράλληλων εφαρμογών που τη χρησιμοποιούν.

Κεφάλαιο 4

Αποδοτική Χρονοδρομολόγηση

«Ο απαισιόδοξος βλέπει τη δυσκολία με κάθε ευκαιρία·
ο αισιόδοξος βλέπει την ευκαιρία σε κάθε δυσκολία.»
– *Winston Churchill*

Στο παρόν κεφάλαιο, αναλύουμε το συμβατικό τρόπο εκτέλεσης παράλληλων εφαρμογών, επικεντρώνοντας το ενδιαφέρον μας στις εφαρμογές που περιέχουν φωλιασμένους βρόχους με ομοιόμορφες εξαρτήσεις. Από την ανάλυση αυτή φαίνεται ότι η εφαρμογή των υφιστάμενων στη βιβλιογραφία τρόπων χρονοδρομολόγησης περιορίζεται σε περιβάλλοντα που χρησιμοποιούν συμβατικές δικτυακές τεχνολογίες, με συνέπεια να μην μπορούν να εκμεταλλευτούν τις δυνατότητες που προσφέρουν τα μοντέρνα δίκτυα διασύνδεσης, ιδιαίτερα σε περιβάλλοντα παράλληλης επεξεργασίας που ακολουθούν την αρχιτεκτονική συστοιχίας υπολογιστών. Στη συνέχεια, προτείνεται ένα *νέος* τρόπος χρονοδρομολόγησης, ο οποίος λαμβάνει υπόψη τις προηγμένες αυτές δυνατότητες των δικτύων διασύνδεσης, βελτιώνοντας την απόδοση των παράλληλων εφαρμογών.

4.1 Βιβλιογραφική Έρευνα

Λόγω της καθυστέρησης που επιβάλει η επικοινωνία στην εκτέλεση παράλληλων εφαρμογών, έχουν προταθεί μέθοδοι διαίρεσης του αρχικού χώρου επαναλήψεων σε σύνολα από επαναλήψεις, τα οποία ανατίθενται σε διαφορετικούς επεξεργαστές [SF92][Hol92]. Η καθυστέρηση αυτή είναι ιδιαίτερα μεγάλη σε περιβάλλοντα συστοιχιών υπολογιστών, όπου τα δίκτυα διασύνδεσης είναι κοινής τεχνολογίας, χωρίς προηγμένα χαρακτηριστικά επικοινωνίας.

Όσον αφορά στον παραλληλισμό με αδρό βήμα επανάληψης (coarse grain), έχουν γίνει προσπάθειες να μετριαστεί η επιβάρυνση που εισάγει η επικοινωνία εφαρμόζοντας το μετασχη-

ματισμό υπερκόμβου. Το συγκεκριμένο μοντέλο παραλληλοποίησης φωλιασμένων βρόχων και η σημειολογία του παρουσιάστηκαν στο Κεφάλαιο 2. Σύμφωνα με το συγκεκριμένο μετασχηματισμό, τα γειτονικά σημεία επανάληψης ομαδοποιούνται, ώστε να σχηματίσουν ένα μεγαλύτερο κόμβο υπολογισμού που εκτελείται ατομικά, χωρίς καμία διακοπή. Επίσης, όπως αναφέρθηκε και προηγουμένως, ομαδοποιούνται και οι ανταλλαγές δεδομένων σε ένα μήνυμα, οι οποίες διεξάγονται για κάθε γειτονικό επεξεργαστή στο τέλος κάθε ατομικής εκτέλεσης υπερκόμβου. Η διαμέριση του χώρου επανάληψης σε υπερκόμβους προτάθηκε από τους Irigoin και Triolet στην εργασία [IT88].

Είναι επίσης γνωστό [RS92] ότι υπάρχει ισοδυναμία μεταξύ του προβλήματος εύρεσης ενός συνόλου ακραίων διανυσμάτων δεδομένου ενός συνόλου διανυσμάτων εξάρτησης και του προβλήματος εύρεσης ενός μετασχηματισμού υπερκόμβου H που παράγει «νόμιμους–έγκυρους» υπερκόμβους, οι οποίοι δεν προκαλούν αδιέξοδα. Η χρήση μιας συνάρτησης επικοινωνίας, που πρέπει να ελαχιστοποιηθεί μέσω της προσέγγισης του γραμμικού προγραμματισμού, χρησιμοποιήθηκε από τους Boulet et al. στην εργασία [BDRR94]. Αυτοί υπολόγισαν το συνολικό κόστος επικοινωνίας που παράγεται από έναν υπερκόμβο, σαν μια συνάρτηση των πλευρών και του σχήματος του και απέδειξαν ότι η ελαχιστοποίησή του μπορεί να γίνει ανεξαρτήτως του όγκου του υπερκόμβου.

Παρόλα αυτά, όλες οι παραπάνω προσεγγίσεις αγνοούν τα πραγματικά όρια του χώρου επαναλήψεων. Αν και το σχήμα του υπερκόμβου είναι μεγάλης σημασίας για τη μείωση της επικοινωνίας, ο αντικειμενικός στόχος θα πρέπει να είναι ο συνολικός χρόνος ολοκλήρωσης του χώρου υπερκόμβων. Οι Hodzic και Shang [HS98] πρότειναν μια μέθοδο για το συσχετισμό του βέλτιστου μεγέθους και σχήματος υπερκόμβου, βασιζόμενοι στο συνολικό χρόνο ολοκλήρωσης. Θεωρούν μετασχηματισμούς υπερκόμβων όπου οι ανταλλαγές δεδομένων γίνονται μεταξύ γειτονικών και συνεχόμενων υπερκόμβων. Ο χώρος υπερκόμβων αποτελεί ένα νέο χώρο επαναλήψεων με μοναδιαίες εξαρτήσεις. Εφάρμοσαν το μετασχηματισμό υπερεπιπέδου (hyperplane transformation) στους βρόχους και παρήγαγαν μια δρομολόγηση, όπου ο αντικειμενικός στόχος ήταν η μείωση του συνολικού χρόνου εκτέλεσης, ρυθμίζοντας κατάλληλα το μέγεθος και το σχήμα του υπερκόμβου. Η παρούσα εργασία συνεχίζει την ερευνητική δουλειά που παρουσιάστηκε στη διδακτορική διατριβή του E. Hodzic [Hod99] και στην εργασία [HS98], εφαρμόζοντας μια *νέα* δρομολόγηση υπερκόμβων. Μέσω αυτής, θεωρώντας ιδανική δικτυακή τεχνολογία με μηδαμινό κόστος αρχικοποίησης της επικοινωνίας, επιτυγχάνουμε θεωρητική επιτάχυνση που προσεγγίζει το 50% του χρόνου που διαρκεί η εκτέλεση της παράλληλης εφαρμογής, η οποία ακολουθεί το συμβατικό τρόπο δρομολόγησης των Hodzic και Shang.

4.2 Απλή–Συμβατική Χρονοδρομολόγηση

Ας θεωρήσουμε ότι έχουμε να εκτελέσουμε παράλληλα το κομμάτι μιας εφαρμογής, το οποίο προσπελαύνει και μεταβάλει τα στοιχεία του πίνακα A μεγέθους $X \times Y$ με τον ακόλουθο τρόπο. Κάθε στοιχείο του πίνακα ισούται με το αποτέλεσμα της συνάρτησης f(element0, element1), η οποία δέχεται ως ορίσματα τις τιμές των προηγούμενων σε κάθε διάσταση στοιχείων. Ο κώδικας αυτός έχει τη μορφή:

```
for (i = 0; i <= X − 1; i++){
    for (j = 0; j <= Y − 1; j++){
        A[i, j] = f(A[i − 1, j] , A[i, j − 1]);
    }
}
```

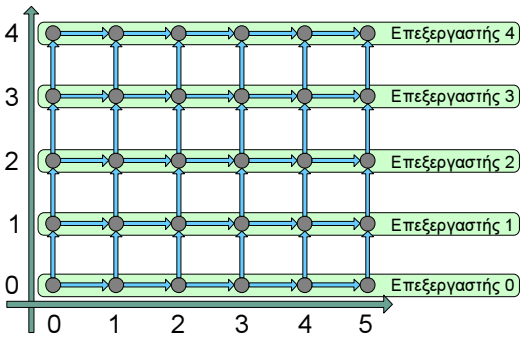
Έστω ότι ο παραπάνω δισδιάστατος κώδικας, χρησιμοποιώντας το μετασχηματισμού υπερκόμβου με κατάλληλο πίνακα μετασχηματισμού H, μετατρέπεται στον εξής κώδικα:

```
for (ii = 0; ii <= 5; ii+=tile_x){
    for (jj = 0; jj <= 4; jj+=tile_y){
        for (i = ii; i <= min(ii + tile_x − 1, X); i++){
            for (j = jj; j <= min(jj + tile_y − 1, Y); j++){
                A[ii, jj] = f(A[ii − 1, jj] , A[ii, jj − 1]);
            }
        }
    }
}
```

Κάθε υπερκόμβος έχει μέγεθος $tile_x \times tile_y$. Ο χώρος υπερκόμβων είναι ο $\mathbb{J}^S = \{j^S \in (ii, jj) \mid 0 \leq ii \leq 5,\ 0 \leq jj \leq 4\}$ και ο μετασχηματισμένος πίνακας εξαρτήσεων είναι ο $D^S = \{(1, 0), (0, 1)\}$. Κάθε υπερκόμβος απεικονίζεται για εκτέλεση σε έναν επεξεργαστή μιας συστοιχίας με πέντε μονοεπεξεργαστικούς κόμβους. Οι υπερκόμβοι κατά μήκος της μίας διάστασης απεικονίζονται στον ίδιο επεξεργαστή, όπως φαίνεται στο Σχήμα 4.1.

Στο ίδιο σχήμα, οι εξαρτήσεις μεταξύ των υπερκόμβων που έχουν απεικονιστεί στον ίδιο επεξεργαστή, δεν περιλαμβάνουν δικτυακή επικοινωνία, αφού τα δεδομένα, που απαιτούνται για την εκτέλεση του «επόμενου» υπερκόμβου, βρίσκονται στην ίδια φυσική μνήμη. Αντιθέτως, οι εξαρτήσεις, μεταξύ των υπερκόμβων που έχουν απεικονιστεί σε διαφορετικούς επεξεργαστές, περιλαμβάνουν δικτυακή επικοινωνία.

Στο παράδειγμα του Σχήματος 4.1, η μία διάσταση έχει μήκος 6, ενώ η άλλη 5, εννοώντας το πλήθος των υπερκόμβων που απεικονίζονται κατά μήκος της διάστασης. Όταν το πλήθος των υπερκόμβων στις διαστάσεις είναι διαφορετικό μεταξύ τους (και συνήθως αυτό συμβαίνει), τότε στον ίδιο επεξεργαστή απεικονίζονται οι υπερκόμβοι της «μεγαλύτερης» διάστασης



Σχήμα 4.1: Απεικόνιση των υπερκόμβων στους επεξεργαστές μιας συστοιχίας.

[HS98]. Η επιλογή αυτή γίνεται προκειμένου να ελαχιστοποιηθούν οι δικτυακές επικοινωνίες που επιβάλλονται εξαιτίας των εξαρτήσεων μεταξύ διαφορετικών επεξεργαστών.

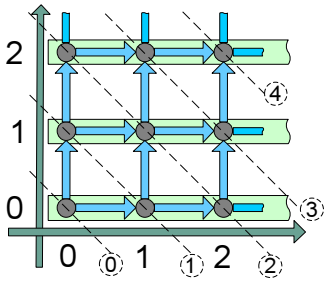
Η χρονική στιγμή στην οποία εκτελούνται οι υπερκόμβοι καθορίζεται από τη χρονοδρομολόγηση που ακολουθείται. Στην παρούσα εργασία θεωρούμε γραμμικές χρονοδρομολογήσεις [SF91], μια ειδική κατηγορία δρομολογήσεων που περιγράφονται από μια γραμμική απεικόνιση των δεικτών υπολογισμού στο χρόνο. Στην περίπτωση των προβλημάτων που εξετάζουμε, όπου ισχύει $HD > 0$ (βλ. Παράγραφο 2.5), οι υπερκόμβοι εκτελούνται ατομικά και διατηρούν την αρχική τους σειρά εκτέλεσης. Συνεπώς, τα στοιχεία του μετασχηματισμένου χώρου J^S (υπερκόμβοι), μπορούν να δρομολογηθούν χρησιμοποιώντας τεχνικές δρομολόγησης όμοιες με αυτές του αρχικού χώρου J^n. Ένα σημείο $j \in J^S$, που δρομολογείται σύμφωνα με ένα γραμμικό διάνυσμα δρομολόγησης Π, εκτελείται το χρονικό βήμα

$$t_{j^S} = \lfloor \frac{\Pi j^S + t_0}{disp\Pi} \rfloor, \tag{4.1}$$

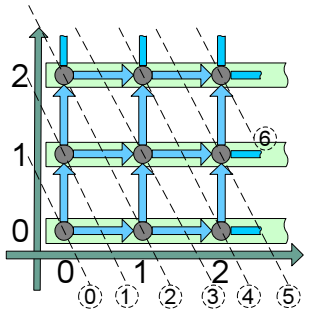
όπου $t_0 = -\min\{\Pi i : i \in J^S\}$ και $disp\Pi = \min\{\Pi d_i^S : d_i^S \in D^S\}$.

Αφού ο μετασχηματισμένος χώρος που μελετάμε (Σχήμα 4.1) έχει μόνο μοναδιαία διανύσματα εξάρτησης, αποδεικνύεται ότι ο βέλτιστος τρόπος δρομολόγησης είναι: $\Pi = [\underbrace{1, 1, \ \ldots \ , 1}_{n}]$, δηλ. στη συγκεκριμένη περίπτωση όπου έχουμε 2 διαστάσεις, $\Pi = [1, 1]$. Οι τιμές των t_0 και $disp\Pi$ είναι 0 και 1, αντίστοιχα. Για παράδειγμα, χρησιμοποιώντας την (4.1), το χρονικό βήμα που εκτελείται ο υπερκόμβος $(2, 2)$ είναι $t_{(2,2)} = [1, 1] \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 4$.

Τα υπερεπίπεδα (παρουσιάζονται ως διακεκομμένες γραμμές στα Σχήματα 4.2 και 4.3) που περιγράφονται από τη συνάρτηση $\Pi j = c$, για διάφορες τιμές του c, βοηθάνε στην απεικόνιση της δρομολόγησης, γιατί όλα τα σημεία που περιέχονται σε αυτά εκτελούνται την ίδια χρονική στιγμή. Ανάλογα με το διάνυσμα χρονοδρομολόγησης Π, δημιουργούνται διαφορετικές οικογέ-



Σχήμα 4.2: Παρουσιάζεται η βέλτιστη χρονοδρομολόγηση $\Pi = [1, 1]$ για ένα τμήμα του χώρου στο Σχήματος 4.1. Φαίνονται τα υπερεπίπεδα για τα οποία ισχύει $c = 0, \ldots, 4$.



Σχήμα 4.3: Παρουσιάζεται μη βέλτιστη χρονοδρομολόγηση $\Pi = [2, 1]$ για ένα τμήμα του χώρου στο Σχήματος 4.1. Φαίνονται τα υπερεπίπεδα για τα οποία ισχύει $c = 0, \ldots, 6$.

νειες υπερεπιπέδων, οι οποίες υποδεικνύουν τη σειρά με την οποία εκτελούνται οι υπερκόμβοι στους επεξεργαστές. Κάθε μια οικογένεια επιτυγχάνει την εκτέλεση του κώδικα σε συγκεκριμένο αριθμό βημάτων, ίσο με

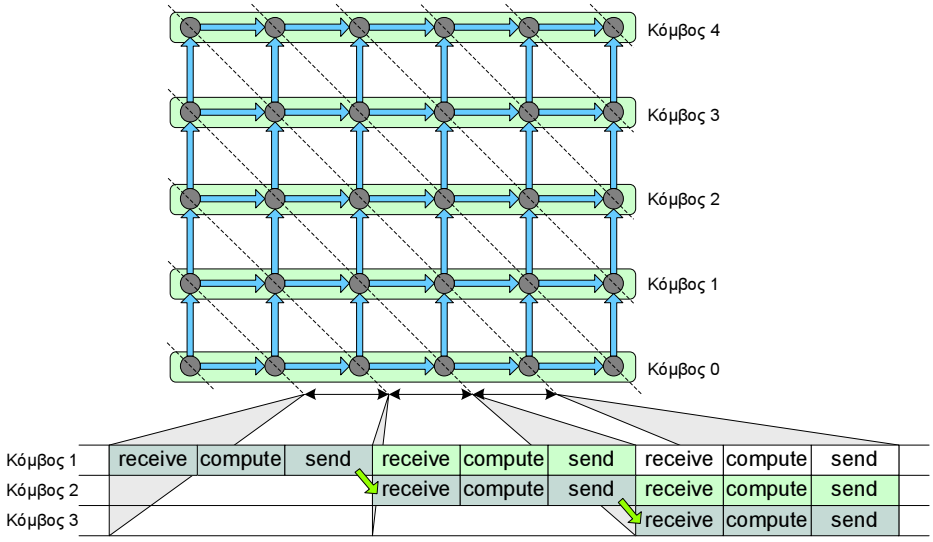
$$\left\lfloor \frac{\max\{\Pi j_1^S - \Pi j_2^S : j_1^S, j_2^S \in J^S\}}{\min\{\Pi d_i^S : d_i^S \in D^S\}} \right\rfloor + 1$$

Αν εφαρμόσουμε τις δρομολογήσεις που παρουσιάζονται στα Σχήματα 4.2 ($\Pi = [1, 1]$) και 4.3 ($\Pi = [1, 2]$) στο χώρο του Σχήματος 4.1, που έχει ακραία σημεία τα $(0, 0)$ και $(5, 4)$, τότε το συνολικό πλήθος βημάτων εκτέλεσης του αλγόριθμου για κάθε δρομολόγηση είναι $\left([1\ 1] \begin{bmatrix} 5 \\ 4 \end{bmatrix} - [1\ 1] \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) + 1 = 10$ και $\left([1\ 2] \begin{bmatrix} 5 \\ 4 \end{bmatrix} - [1\ 2] \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) + 1 = 14$, αντίστοιχα.

Παρόλα αυτά, ο χρόνος, που απαιτείται για την περάτωση της εκτέλεση ενός παράλληλα εκτελούμενου κώδικα, εξαρτάται και από άλλους παράγοντες εκτός του πλήθους των βημάτων. Έτσι, στο προηγούμενο παράδειγμα, ενώ με τη βέλτιστη χρονοδρομολόγηση απαιτούνται 10 βήματα εκτέλεσης και με τη μη-βέλτιστη 14, ενδεχομένως το χρονικό διάστημα που διαρκεί κάθε βήμα επανάληψης να είναι πολύ μικρότερο στην 2η χρονοδρομολόγηση, με αποτέλεσμα ο συνολικός χρόνος εκτέλεσης της εφαρμογής να είναι μικρότερος στην 2η περίπτωση.

Κατά την εκτέλεση του παράλληλου κώδικα, λαμβάνουν χώρα οι ακόλουθες διαδικασίες: Αρχικά, κάθε κόμβος, για την ικανοποίηση των εξαρτήσεων, οφείλει να λάβει δεδομένα από τους γειτονικούς δικτυακούς κόμβους. Στη συνέχεια, χρησιμοποιώντας τα δεδομένα αυτά, υπολογίζει τις τιμές των στοιχείων του υπερκόμβου για τον οποίο είναι υπεύθυνος. Τελειώνοντας τη φάση του υπολογισμού, πρέπει να αποστείλει στον «επόμενο» κόμβο δεδομένα για την ικανοποίηση των εξαρτήσεών του.

Στο Σχήμα 4.4 παρουσιάζεται η χρονοδρομολόγηση των Hodzic και Shang, μαζί τα προαναφερθέντα χαρακτηριστικά, σε ένα σύστημα 5 μονοεπεξεργαστικών κόμβων. Τη συγκεκριμένη χρονοδρομολόγηση την ονομάζουμε *μη-επικαλυπτόμενη*. Όπως φαίνεται στο σχήμα, σε κάθε χρονικό βήμα μεταξύ δύο διαδοχικών υπερεπιπέδων, κάθε κόμβος εκτελεί ατομικά μια τριπλέτα φάσεων «λήψης – υπολογισμού – αποστολής», οι οποίες δεν επικαλύπτονται μεταξύ τους. Αν ενοποιήσουμε τις φάσεις αποστολών και λήψεων σε μία φάση επικοινωνίας, τότε παρουσιάζεται μια εκ περιτροπής εναλλαγή των φάσεων υπολογισμού και επικοινωνίας.



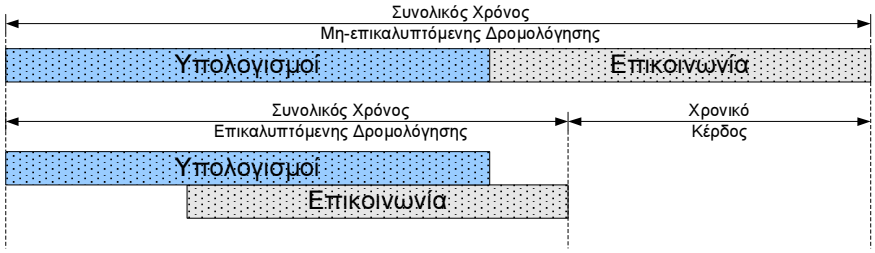
Σχήμα 4.4: Μη-επικαλυπτόμενη χρονική δρομολόγηση, όπου αναλύονται οι φάσεις λήψης, υπολογισμού και αποστολής, για τους κόμβους 1, 2 και 3.

Αυτό το ξεκάθαρο μοντέλο επικοινωνίας επιτυγχάνει καλούς χρόνους επικοινωνίας, εκμεταλλευόμενο τον παραλληλισμό στις εκτελέσεις υπερκόμβων στους διαφορετικούς επεξεργαστές. Όμως, εξετάζοντας την αξιοποίηση που επιτυγχάνει σε κάθε επεξεργαστή και κάθε δικτυακό υποσύστημα, παρατηρούμε τα εξής: Κάθε επεξεργαστής πραγματοποιεί μια φάση υπολογισμού και στη συνέχεια περνάει μια φάση αναμονής για την πραγματοποίηση της δικτυακής επικοινωνίας, τόσο για να αποσταλούν δεδομένα στον «επόμενο» κόμβο, όσο και για να ληφθούν δεδομένα από τον «προηγούμενο». Αντίστοιχα, κάθε δικτυακό υποσύστημα αξιοποιείται στην πραγματοποίηση μιας φάσης επικοινωνίας (αποστολή και λήψη δεδομένων) και στη συνέχεια μένει αναξιοποίητο σε μια φάση αναμονής, κατά τη διάρκεια της οποίας ο επεξεργαστής εκτελεί υπολογισμούς. Συνεπώς, με το συγκεκριμένο τρόπο δρομολόγησης, δεν επιτυγχάνεται αποδοτική εκμετάλλευση, ούτε του επεξεργαστή, ούτε του δικτυακού υποσυστήματος.

4.3 Επικαλυπτόμενη Χρονοδρομολόγηση

Η γραμμική, μη-επικαλυπτόμενη δρομολόγηση, που παρουσιάστηκε στην παράγραφο 4.2, επιτυγχάνει μια μέτρια αξιοποίηση του επεξεργαστή και του δικτυακού υποσυστήματος. Κάθε χρονική στιγμή, οι κόμβοι είτε πραγματοποιούν όλοι υπολογισμούς, είτε πραγματοποιούν όλοι επικοινωνία. Όμως, οι νέες δικτυακές τεχνολογίες, σε συνδυασμό με προηγμένα πρωτόκολλα επικοινωνίας και τις αλλαγές στη λειτουργία και νοοτροπία των λειτουργικών συστημάτων, παρέχουν δυνατότητες για ταυτόχρονη πραγματοποίηση των λειτουργιών του επεξεργαστή (υπολογισμοί) και του δικτυακού υποσυστήματος (επικοινωνία). Συγκεκριμένα, οι μηχανές άμεσης προσπέλασης μνήμης, που πλέον συμπεριλαμβάνονται σε όλους τους σύγχρονους δικτυακούς προσαρμογείς, επιτρέπουν την ταυτόχρονη μετάδοση δεδομένων από την πλευρά του δικτύου και την πραγματοποίηση υπολογισμών από την πλευρά της ΚΜΕ.

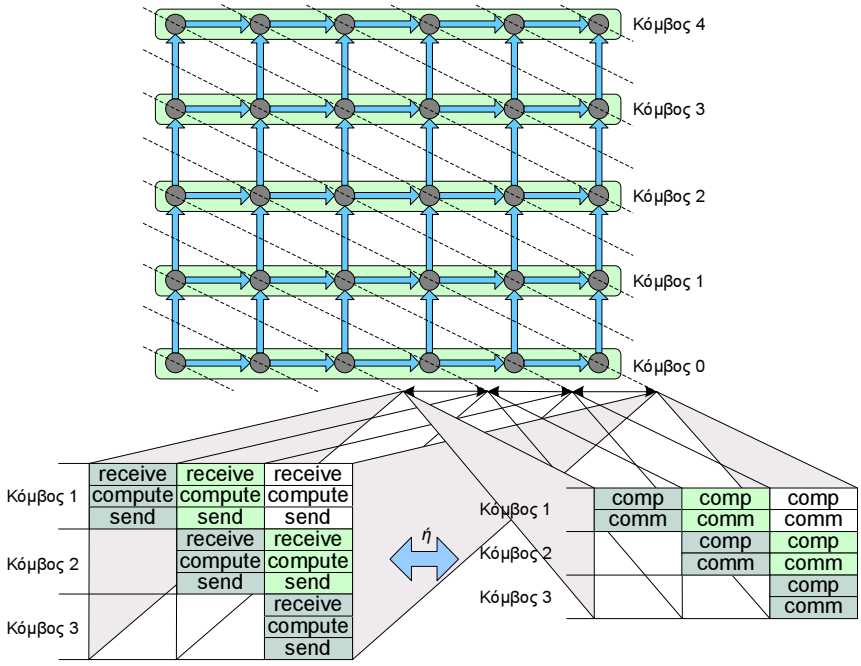
Θεωρούμε έναν κόμβο που συμμετέχει στην παράλληλη εκτέλεση ενός προβλήματος. Επίσης, θεωρούμε ότι ο συνολικός χρόνος συμμετοχής του στην εκτέλεση χωρίζεται σε δύο τμήματα· στο τμήμα των υπολογισμών και στο τμήμα της επικοινωνίας. Τότε, είναι δυνατόν να μειώσουμε το συνολικό χρόνο εκτέλεσης επικαλύπτοντας ένα τμήμα της επικοινωνίας με αυτό των υπολογισμών (Σχήμα 4.5). Ιδανικά, ολόκληρο το χρονικό διάστημα της επικοινωνίας θα μπορούσε να επικαλυφθεί με το χρόνο υπολογισμών, έτσι ώστε να επιτύχουμε γραμμική αύξηση της απόδοσης του παράλληλου συστήματος, συναρτήσει των κόμβων που συμμετέχουν σε αυτό.



Σχήμα 4.5: Το χρονικό κέρδος που επιφέρει η επικάλυψη ενός μέρους του χρόνου υπολογισμών με χρόνο επικοινωνίας.

Όμως, ο πραγματικός λόγος που επιβάλει τη μη αποδοτική αξιοποίηση της ΚΜΕ, οδηγώντας σε μεγαλύτερους χρόνους εκτέλεσης, είναι η ροή των δεδομένων ανάμεσα στα διαδοχικά χρονικά βήματα. Ειδικότερα, φαίνεται ότι οι φάσεις υπολογισμού και επικοινωνίας πρέπει να είναι σειριοποιημένες για να διατηρείται η σωστή σειρά εκτέλεσης. Σε κάθε βήμα εκτέλεσης του προγράμματος, κάθε επεξεργαστής πρέπει πρώτα να λάβει δεδομένα, στη συνέχεια να κάνει υπολογισμούς χρησιμοποιώντας αυτά τα δεδομένα και τέλος, να αποστείλει τα παραγόμενα δεδομένα, ώστε να χρησιμοποιηθούν από το γειτονικό κόμβο (βλ. Σχήμα 4.4).

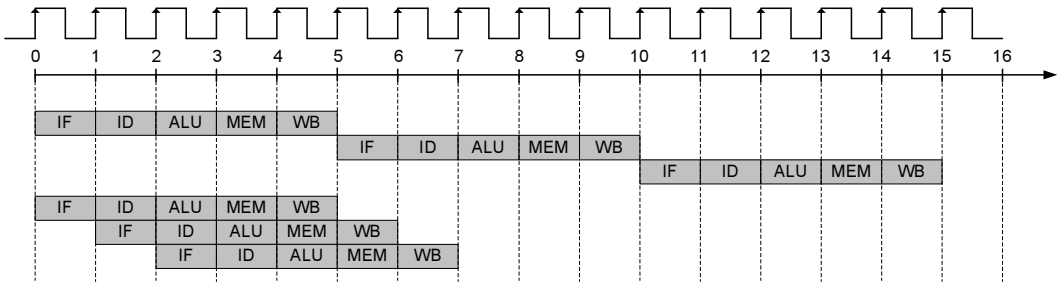
Η παρούσα εργασία προτείνει ένα νέο γραμμικό τρόπο δρομολόγησης, που επιτυγχάνει επικάλυψη του χρόνου επικοινωνίας με χρόνο υπολογισμών, μειώνοντας το χρονικό διάστημα που διαρκεί κάθε βήμα του υπολογισμού. Αυτό σημαίνει ότι, σε κάθε βήμα, ο κόμβος λαμβάνει και στέλνει δεδομένα που δεν εξαρτώνται άμεσα με τα υπολογιζόμενα δεδομένα του συγκεκριμένου χρονικού βήματος. Το προτεινόμενο έγκυρο σχήμα εκτέλεσης είναι το εξής: *Έστω ότι το παρόν βήμα εκτέλεσης είναι το k-στό. Ο κόμβος εκτελεί υπολογισμούς, καθώς λαμβάνει δεδομένα από όλους τους —βάσει των εξαρτήσεων— γειτονικούς κόμβους, που θα τα χρησιμοποιήσει το χρονικό βήμα $k+1$, ενώ στέλνει δεδομένα που παρήχθησαν το χρονικό βήμα $k-1$* (βλ. Σχήμα 4.6). Σε αυτή την περίπτωση, κάθε κόμβος υπολογίζει τα περιεχόμενα ενός υπερκόμβου, ενώ, ταυτόχρονα, λαμβάνει δεδομένα που θα του χρησιμεύσουν για τον υπολογισμό των δεδομένων του επόμενου χρονικού βήματος, και στέλνει δεδομένα που παρήχθησαν το προηγούμενο βήμα. Την προτεινόμενη μέθοδο δρομολόγησης ονομάζουμε *επικαλυπτόμενη χρονοδρομολόγηση* ή *χρονοδρομολόγηση σωλήνωσης (pipelined schedule).*



Σχήμα 4.6: Η μέθοδος της Επικαλυπτόμενης Χρονοδρομολόγησης. Παρουσιάζεται η ανάλυση κάθε χρονικού βήματος στις επιμέρους λειτουργίες που εκτελούνται στους κόμβους 1,2 και 3. Επίσης, φαίνεται η ενσωμάτωση των λειτουργιών της λήψης και της αποστολής σε μία ενιαία λειτουργία επικοινωνίας.

Η συγκεκριμένη μέθοδος βελτιώνει τις επιδόσεις των παράλληλων προγραμμάτων που τη χρησιμοποιούν, όμοια με τη βελτίωση των επιδόσεων που επέφερε η έννοια της σωλήνωσης (pipeline) [PH94], αρχικά, στην αρχιτεκτονική της οικογένειας των επεξεργαστών μειωμένου

συνόλου εντολών RISC (Reduced Instruction Set Computers). Στο Σχήμα 4.7 παρουσιάζεται η μετάβαση από έναν επεξεργαστή με πολλούς κύκλους ρολογιού, σε ένα επεξεργαστή με σωλήνωση. Ενώ πρωτύτερα εκτελείτο μία εντολή κάθε πέντε κύκλους ρολογιού, με το pipeline εκτελείται μία εντολή κάθε έναν κύκλο ρολογιού.



Σχήμα 4.7: Η βελτίωση της επίδοσης των επεξεργαστών με τη μετάβαση από την αρχιτεκτονική με πολλούς κύκλους ρολογιού, στην αρχιτεκτονική pipeline. Η εκτέλεση 3 εντολών, οι οποίες διαιρούνται στις φάσεις Instruction Fetch, Instruction Decode, Arithmetic Logical Unit, Memory και Write-Back, διαρκεί 15 κύκλους στην πρώτη περίπτωση και 8 στη δεύτερη.

4.4 Θεωρητική Σύγκριση Μεθόδων Χρονοδρομολόγησης

Προηγούμενη δουλειά στον τομέα της δρομολόγησης UET-UCT [AKPT99], η οποία, εκτός από το χρονικό κόστος υπολογισμού, λάμβανε υπόψη και το χρονικό κόστος της επικοινωνίας, είχε δείξει ότι η δρομολόγηση είναι βέλτιστη όταν το πηλίκο χρόνου υπολογισμού προς χρόνο επικοινωνίας είναι ίσο με 1. Ομοίως, στην παρούσα εργασία γίνεται προσπάθεια να επικαλυφθεί το μεγαλύτερο μέρος της επικοινωνίας δύο γειτονικών κόμβων σε χρόνο υπολογισμών. Η βέλτιστη περίπτωση επιτυγχάνεται όταν το πηλίκο χρόνου υπολογισμού προς χρόνο επικοινωνίας είναι ίσο με 1.

Το μέγεθος, που μας ενδιαφέρει να ελαχιστοποιήσουμε, είναι ο συνολικός χρόνος εκτέλεσης του αλγόριθμου που επιτυγχάνεται χρησιμοποιώντας κάθε μέθοδο. Ο συνολικός χρόνος εκτέλεσης T ισούται με το γινόμενο του αριθμού των βημάτων P, που απαιτούνται για την ολοκλήρωση της εκτέλεσης, με τη χρονική διάρκεια κάθε βήματος T_{step}. Συνεπώς, έχουμε:

$$T(g) = P(g)T_{step}(g), \tag{4.2}$$

όπου $T(g)$ είναι ο συνολικός χρόνος εκτέλεσης, $P(g)$ είναι ο αριθμός των βημάτων και $T_{step}(g)$ είναι ο χρόνος που διαρκεί κάθε βήμα. Όλοι οι παραπάνω χρόνοι εξαρτώνται από το μέγεθος g του υπερκόμβου.

Για την περίπτωση της μη-επικαλυπτόμενης δρομολόγησης, ο συνολικός χρόνος εκτέλεσης του προγράμματος είναι:

$$T_n(g_n) = P_n(g_n)(T_{comp}(g_n) + T_{comm}(g_n)), \tag{4.3}$$

όπου $T_{comp}(g_n)$ είναι ο χρόνος υπολογισμού ενός υπερκόμβου και $T_{comm}(g_n)$ είναι ο απαιτούμενος χρόνος επικοινωνίας, για δεδομένο μέγεθος υπερκόμβου g_n. Όμοια, στην ιδανική περίπτωση της προτεινόμενης επικαλυπτόμενης δρομολόγησης, ο συνολικός χρόνος εκτέλεσης του προγράμματος είναι:

$$T_o(g_o) = P_o(g_o)(T_{comp}(g_o)), \tag{4.4}$$

όπου $T_{comp}(g_o)$ είναι ο χρόνος υπολογισμού ενός υπερκόμβου. Ο χρόνος επικοινωνίας δεν συμπεριλαμβάνεται στην (4.4), αφού επικαλύπτεται με το χρόνο υπολογισμού.

Έστω g, το μέγεθος υπερκόμβου με το οποίο επιτυγχάνεται ο ελάχιστος χρόνος στην περίπτωση της μη-επικαλυπτόμενης δρομολόγησης. Τότε ο συνολικός χρόνος εκτέλεσης είναι $T_{non}(g) = P_{non}(g)(T_{comp}(g) + T_{comm}(g))$. Επιλέγουμε το ίδιο μέγεθος υπερκόμβου και για την περίπτωση της επικαλυπτόμενης δρομολόγησης, με συνολικό χρόνο εκτέλεσης $T_{ov}(g) = P_{ov}(g)T_{comp}(g)$.

Το πλήθος των βημάτων εκτέλεσης για κάθε περίπτωση είναι το εξής: Στην περίπτωση της μη-επικαλυπτόμενης δρομολόγησης, όπου το διάνυσμα δρομολόγησης είναι $\Pi = [\underbrace{1, 1, \ldots, 1}_{n}]$, έχουμε $P_{non}(g) = \sum(x_i - 1) + 1$, όπου x_i είναι το πλήθος των κόμβων που απεικονίζονται στην i-οστή διάσταση. Στην περίπτωση της επικαλυπτόμενης δρομολόγησης, το διάνυσμα δρομολόγησης είναι $\Pi = [\underbrace{2, 2, \ldots, 2, \overbrace{1}^{j}, 2, 2, \ldots, 2}_{n}]$, όπου j είναι η διάσταση κατά μήκος της οποίας οι υπερκόμβοι απεικονίζονται στον ίδιο επεξεργαστή. Σε αυτή την περίπτωση το πλήθος των βημάτων εκτέλεσης είναι $P_{ov}(g) = 2\sum_{i \neq j}(x_i - 1) + x_j$. Για να δούμε σε ποιες περιπτώσεις ισχύει $T_{ov}(g) < T_{non}(g)$, τότε σύμφωνα με τις (4.3) και (4.4), θα πρέπει να ισχύουν τα εξής:

$$\begin{aligned}
\left[2\sum_{i \neq j}(x_i - 1) + x_j\right] T_{comp} &< \left[\sum(x_i - 1) + 1\right](T_{comp} + T_{comm}) \Leftrightarrow \\
\left[2\left(\sum_{i \neq j} x_i - (n-1)\right) + x_j\right] T_{comp} &< \left[\sum x_i - (n-1)\right](T_{comp} + T_{comm})
\end{aligned}$$

Αν θεωρήσουμε ότι ο χρόνος υπολογισμού επικαλύπτεται πλήρως με το χρόνο επικοινωνίας,

δηλ. ισχύει $T_{comp} = T_{comm}$, τότε η σχέση γίνεται:

$$\begin{aligned} \left[2\left(\sum_{i\neq j} x_i - (n-1)\right) + x_j\right] T_{comp} &< 2\left[\sum x_i - (n-1)\right] T_{comp} \Leftrightarrow \\ x_j &> 0 \end{aligned}$$

Η τελευταία ανισότητα ισχύει πάντα, αφού το πλήθος των υπερκόμβων κατά μήκος της μεγαλύτερης διάστασης είναι πάντα θετικό. Το συγκεκριμένο αποτέλεσμα είναι αναμενόμενο, αφού, θεωρώντας ότι όλοι οι παράγοντες επικοινωνίας επικαλύπτονται χρονικά με την εκτέλεση των υπερκόμβων στην ΚΜΕ, τότε η περίπτωση της επικαλυπτόμενης δρομολόγησης είναι πάντα καλύτερη από τη μη-επικαλυπτόμενη δρομολόγηση.

Στη συνέχεια, εξετάζουμε την περίπτωση που μόνο ένα ποσοστό της επικοινωνίας επικαλύπτεται από χρόνο υπολογισμού. Θεωρούμε D_{non} και D_{ov} το χρονικό διάστημα που διαρκεί η εκτέλεση ενός χρονικού βήματος στη μη-επικαλυπτόμενη και επικαλυπτόμενη περίπτωση, αντίστοιχα. Επίσης, θεωρούμε α το μέγεθος που ποσοτικοποιεί την επικάλυψη του χρόνου υπολογισμού με το χρόνο επικοινωνίας και ορίζεται ως $\alpha = \frac{D_{non}}{D_{ov}}$. Για το μέγεθος αυτό ισχύει $1 < \alpha < 2$, αφού στην καλύτερη περίπτωση, από πλευράς επικάλυψης επικοινωνίας, ο χρόνος της μη-επικαλυπτόμενης περίπτωσης ισούται με το διπλάσιο του χρόνου της επικαλυπτόμενης, ενώ στη χειρότερη, οι δύο χρόνοι είναι ίσοι. Ο χρόνος που διαρκεί η συνολική εκτέλεση του προγράμματος, για κάθε περίπτωση, είναι $T_{non} = \left[\sum(x_i - 1) + 1\right] D_{non}$ και $T_{ov} = \left[2\sum_{i\neq j}(x_i - 1) + x_j\right] D_{ov}$, αντίστοιχα. Για να ισχύει η σχέση $T_{non} > T_{ov}$, θα πρέπει να ισχύει:

$$\begin{aligned} \left[\sum(x_i - 1) + 1\right] D_{non} &> \left[2\sum_{i\neq j}(x_i - 1) + x_j\right] D_{ov} \Leftrightarrow \\ x_j &> \sum_{i\neq j}(x_i - 1)\frac{2-\alpha}{\alpha - 1} \end{aligned}$$

Σύμφωνα με την παραπάνω σχέση, για να έχει νόημα η χρήση της μεθόδου επικαλυπτόμενης δρομολόγησης, το μέγεθος της μεγαλύτερης σε μήκος διάστασης θα πρέπει να είναι μεγαλύτερο από μία τιμή, η οποία εξαρτάται από το πλήθος των διαστάσεων του προβλήματος, από το μήκος των διαστάσεων (πλην αυτής που έχει το μεγαλύτερο μήκος), καθώς από την τιμή α.

Στον Πίνακα 4.1 παρατηρούμε διάφορες τιμές της x_j με βάση χαρακτηριστικά των προβλημάτων που μας ενδιαφέρουν. Εξετάζοντας τις τιμές που εμφανίζονται, παρατηρούμε ότι ακόμη και για ένα πρόβλημα 2 διαστάσεων, σε μία συστοιχία υπολογιστών με 1024 κόμβους, και με ποσοστό επικάλυψης μόλις 1.1, αρκεί το πρόβλημα να είναι αρκετά μεγάλο, ώστε να

δημιουργηθούν περισσότεροι από 9207 υπερκόμβοι στη διάσταση x_j. Αυτό σημαίνει ότι, θεωρώντας μεγάλου μεγέθους προβλήματα, η επικαλυπτόμενη μέθοδος μειώνει πάντα το συνολικό χρόνο εκτέλεσης των προγραμμάτων, βελτιώνοντας την απόδοσή τους.

Διαστάσεις	**Κόμβοι**	α	x_j	**Διαστάσεις**	**Κόμβοι**	α	x_j
2	8	1.1	63.00	3	8	1.1	32.91
2	16	1.1	135.00	3	16	1.1	54.00
2	32	1.1	279.00	3	32	1.1	83.82
2	128	1.1	1143.00	3	128	1.1	185.65
2	1024	1.1	9207.00	3	1024	1.1	558.00
2	8	1.3	16.33	3	8	1.3	8.53
2	16	1.3	35.00	3	16	1.3	14.00
2	32	1.3	72.33	3	32	1.3	21.73
2	128	1.3	296.33	3	128	1.3	48.13
2	1024	1.3	2387.00	3	1024	1.3	144.67
2	8	1.5	7.00	3	8	1.5	3.66
2	16	1.5	15.00	3	16	1.5	6.00
2	32	1.5	31.00	3	32	1.5	9.31
2	128	1.5	127.00	3	128	1.5	20.63
2	1024	1.5	1023.00	3	1024	1.5	62.00
2	8	1.7	3.00	3	8	1.7	1.57
2	16	1.7	6.43	3	16	1.7	2.57
2	32	1.7	13.29	3	32	1.7	3.99
2	128	1.7	54.43	3	128	1.7	8.84
2	1024	1.7	438.43	3	1024	1.7	26.57
2	8	1.9	0.78	3	8	1.9	0.41
2	16	1.9	1.67	3	16	1.9	0.67
2	32	1.9	3.44	3	32	1.9	1.03
2	128	1.9	14.11	3	128	1.9	2.29
2	1024	1.9	113.67	3	1024	1.9	6.89

Πίνακας 4.1: Ελάχιστες τιμές του μήκους της διάστασης x_j, ώστε ο συνολικός χρόνος εκτέλεσης της επικαλυπτόμενης περίπτωσης να είναι μικρότερος από αυτόν της μη-επικαλυπτόμενης.

Από τα παραπάνω είναι προφανές, ότι αν θεωρήσουμε πραγματικά προβλήματα, τα οποία εκτελούνται σε συστοιχίες υπολογιστών με πεπερασμένο πλήθος επεξεργαστικών στοιχείων, ο συνολικός χρόνος εκτέλεσης, χρησιμοποιώντας την προτεινόμενη μέθοδο, είναι πάντα μικρότερος σε σχέση με το χρόνο που απαιτεί ο συμβατικός τρόπος εκτέλεσης των εφαρμογών. Οι περιπτώσεις για τις οποίες συμβαίνει το αντίθετο είναι δύο: α) Να μην υπάρχει καθόλου επικάλυψη επικοινωνίας με υπολογισμούς, γεγονός που δεν συμβαίνει στη δική μας περίπτωση, β) Το πρόβλημα να είναι πολύ μικρό σε σχέση με το πλήθος των επεξεργαστικών κόμβων της συστοιχίας. Τέτοιου είδους προβλήματα δεν απασχολούν τους χρήστες συστημάτων συστοιχιών υπολογισμών.

4.5 Πραγματικός–Ρεαλιστικός Χρόνος Εκτέλεσης

Στην παραπάνω θεωρητική μελέτη του μοντέλου εκτέλεσης παράλληλων εφαρμογών χρησιμοποιώντας την προτεινόμενη μέθοδο επικαλυπτόμενης δρομολόγησης και στη σύγκριση αυτής με τη συμβατική μέθοδο δρομολόγησης, θεωρήσαμε ένα ιδανικό περιβάλλον εκτέλεσης στο οποίο το σύνολο των λειτουργιών της επικοινωνίας ήταν δυνατό να επικαλυφθεί με υπολογισμούς που εκτελούνται στην ΚΜΕ του κόμβου. Στην πραγματικότητα όμως, ανάλογα με το συνολικό σύστημα επικοινωνίας που χρησιμοποιείται (συνδυασμός υλικού και λογισμικού) υπάρχει ένα χρονικό διάστημα το οποίο δεν μπορεί να επικαλυφθεί. Χαρακτηριστικό παράδειγμα είναι ο χρόνος που απαιτείται για την αρχικοποίηση της αποστολής ενός πακέτου στο δίκτυο. Συνεπώς, θεωρώντας την έκφραση (4.4), που δίνει το συνολικό χρόνο εκτέλεσης στην περίπτωση της επικαλυπτόμενης δρομολόγησης, η έκφραση που δίνει ένα ρεαλιστικότερο χρόνο εκτέλεσης είναι

$$T_{ov}(g) = P(g)(T_{comp}(g) + T_{startup}(g)), \tag{4.5}$$

όπου $T_{startup}$ είναι ο απαιτούμενος, ακόμη και για την επικαλυπτόμενη επικοινωνία, χρόνος προετοιμασίας αποστολής δεδομένων. Το συγκεκριμένο χρονικό διάστημα είναι πολύ μικρότερο από το συνολικό χρόνο επικοινωνίας που απαιτείται στην περίπτωση της μη-επικαλυπτόμενης δρομολόγησης.

Στο επόμενο Κεφάλαιο, πραγματοποιείται σύγκριση των διαφορετικών μεθόδων χρονοδρομολόγησης, με την παρουσίαση πειραματικών αποτελεσμάτων εκτέλεσης εφαρμογών, σε συστοιχία υπολογιστών χρησιμοποιώντας διαφορετικές δικτυακές τεχνολογίες.

Κεφάλαιο 5

Εφαρμογή θεωρίας σε συγκεκριμένες τεχνολογίες

«Στη θεωρία, δεν υπάρχει διαφορά ανάμεσα στη θεωρία και στην πράξη.
Στην πράξη, όμως, υπάρχει.»
– Jan van de Snepscheut

Στο Κεφάλαιο αυτό γίνεται σύγκριση της προτεινόμενης μεθόδου χρονοδρομολόγησης με τη συμβατική μέθοδο, χρησιμοποιώντας πειραματικές μετρήσεις σε κομμάτια (stencils) από πραγματικές εφαρμογές. Σε περιβάλλον συστοιχίας υπολογιστών με δύο διαφορετικές δικτυακές τεχνολογίες (FastEthernet και SCI) εκτελούνται παραλληλοποιημένα τμήματα κώδικα φωλιασμένων βρόχων. Σε κάθε περίπτωση, αναλύεται ο τρόπος εκτέλεσης της εφαρμογής, ενώ γίνεται αξιοποίηση των ιδιαίτερων χαρακτηριστικών της κάθε τεχνολογίας, όπως αυτά παρουσιάστηκαν στο Κεφάλαιο 3. Ο χρόνος εκτέλεσης των παραλληλοποιημένων τμημάτων κώδικα, χρησιμοποιώντας τον προτεινόμενο τρόπο επικαλυπτόμενης χρονοδρομολόγησης, είναι πάντα μικρότερος από τον αντίστοιχο χρόνο της συμβατικής χρονοδρομολόγησης. Σε αρκετές περιπτώσεις, η επιτάχυνση της προτεινόμενης μεθόδου προσεγγίζει το ήμισυ του χρόνου εκτέλεσης που ακολουθεί τη συμβατική χρονοδρομολόγηση. Τα αποτελέσματα αυτά καταδεικνύουν τη χρησιμότητα της συγκεκριμένης μεθόδου για την εκτέλεση παράλληλων εφαρμογών που περιέχουν πολλαπλά φωλιασμένους βρόχους.

5.1 Περιβάλλον Εκτέλεσης Πειραμάτων FastEthernet

Για την πειραματική αξιολόγηση της προτεινόμενης μεθόδου, αρχικά χρησιμοποιήθηκε μια συστοιχία υπολογιστών με 16 πανομοιότυπους κόμβους. Κάθε κόμβος περιείχε επεξεργαστή

Pentium III με εσωτερική συχνότητα ρολογιού 500 MHz. Το μέγεθος της μνήμης τυχαίας προσπέλασης κάθε κόμβου ήταν 128 MB. Η δικτυακή τεχνολογία που συνέδεε του κόμβους ήταν FastEthernet, ενώ το λειτουργικό σύστημα που εκτελούνταν στους κόμβους ήταν Linux με πυρήνα έκδοσης 2.2.14. Ο προγραμματισμός και η εκτέλεση της παραλληλοποιημένης έκδοσης της εφαρμογής έγινε χρησιμοποιώντας την υλοποίηση MPI Chameleon (MPICH) του πρότυπου MPI.

Η εφαρμογή που παραλληλοποιήθηκε είχε την ακόλουθη μορφή:

```
for(i = 1; i < I; i++){
   for(j = 1; j < J; j++){
      for(k = 1; k < K; k++){
        A[i][j][k] = sqrt(A[i-1][j][k] + A[i][j-1][k] + A[i][j][k-1]);
      }
   }
}
```

Τα στοιχεία του πίνακα `A` είναι αριθμοί κινητής υποδιαστολής (floats) και έχουν μέγεθος 4 bytes. Η συνάρτηση `sqrt(x)` υπολογίζει την τετραγωνική ρίζα της έκφρασης `x` και χρησιμοποιήθηκε, χωρίς βλάβη της γενικότητας, για να αυξηθεί ο χρόνος υπολογισμού, προσομοιώνοντας ρεαλιστικά σώματα βρόχων από τυπικές αριθμητικές εφαρμογές (βλ. Παράρτημα Β) που έχουν σημαντικό κόστος επεξεργασίας ανά επανάληψη (π.χ. πράξεις με αριθμούς κινητής υποδιαστολής).

Χρησιμοποιώντας τη μέθοδο παραλληλοποίησης με υπερκόμβους, το βέλτιστο σχήμα του υπερκόμβου είναι ορθογώνιο παραλληλεπίπεδο με πλευρές τις ij, ik και jk. Επιλέξαμε τη διάσταση k να είναι η μεγαλύτερη, έτσι ώστε όλοι οι υπερκόμβοι κατά μήκος της συγκεκριμένης διάστασης να απεικονίζονται στον ίδιο επεξεργαστή $P_i,\ i = (0, \ldots, 15)$.

Κατά τη διάρκεια κάθε χρονικού βήματος, κάθε επεξεργαστής στο επίπεδο ij με συντεταγμένες (i, j) λαμβάνει δεδομένα από τους γειτονικούς κόμβους $(i-1, j)$ και $(i, j-1)$, κάνει τον υπολογισμό των στοιχείων και αποστέλλει στους κόμβους $(i+1, j)$ και $(i, j+1)$ τα στοιχεία που χρειάζονται.

5.1.1 Υλοποίηση με MPI

Ακολουθώντας το προγραμματιστικό περιβάλλον του MPI, ο κώδικας που υλοποιεί τη μη-επικαλυπτόμενη μέθοδο δρομολόγησης είναι ο εξής:

```
for(i = 0; i < max_i_tile-1; i++){
   for(j = 0; j < max_j_tile-1; j++){
      ProcNON(i, j);
   }
}
```

όπου τα max_i_tile και max_j_tile συμβολίζουν το πλήθος των υπερκόμβων κατά μήκος των διαστάσεων i και j, αντίστοιχα. Ο παραπάνω κώδικας περνάει τις παραμέτρους (i, j) σε κάθε κόμβο, ώστε να γνωρίζει τη θέση του μέσα στο πλέγμα των επεξεργαστών. Η συνάρτηση ProcNON εκτελείται μια φορά σε κάθε κόμβο και υλοποιεί τις λειτουργίες υπολογισμών και επικοινωνίας για κάθε υπερκόμβο. Ο κώδικάς της είναι ο εξής:

```
for(k = 0; k < max_k_tile-1; k++){
 MPI_Recv(T(i-1, j), results(T(i-1, j), k));
 MPI_Recv(T(i, j-1), results(T(i, j-1), k));
 compute();
 MPI_Send(T(i+1, j), results(T(i, j), k));
 MPI_Send(T(i, j+1), results(T(i, j), k));
}
```

Στην περίπτωση της προτεινόμενης επικαλυπτόμενης χρονοδρομολόγησης, ο κώδικας που υλοποιεί την παραλληλοποιημένη μορφή της εφαρμογής είναι ο εξής:

```
for(i = 0; i < MAX_i_tile-1; i++){
   for(j = 0; j < MAX_j_tile-1; j++){
      ProcOV(i, j);
   }
}
```

Στην περίπτωση αυτή, ο κώδικας της συνάρτησης ProcOV είναι ο εξής:

```
for(k = 0; k < MAX_k_tile-1; k++){
 MPI_Isend(T(i+1, j), results(T(i, j), k-1), &s1);
 MPI_Isend(T(i, j+1), results(T(i, j), k-1), &s2);
 MPI_Irecv(T(i-1, j), results(T(i-1, j), k+1), &r1);
 MPI_Irecv(T(i, j-1), results(T(i, j-1), k+1), &r2);
 compute();
 MPI_Wait(s1);
 MPI_Wait(s2);
 MPI_Wait(r1);
 MPI_Wait(r2);
}
```

Σύμφωνα με το πρότυπο του MPI, οι συναρτήσεις MPI_Send και MPI_Recv υλοποιούν την αποστολής και λήψης μηνυμάτων ακολουθώντας την κατάσταση standard (βλ. Παράγραφο 3.9.2.2), σύμφωνα με την οποία, το σύστημα περιμένει την ολοκλήρωση της λειτουργία της αποστολής ή/και της λήψης που περιγράφεται από την καλούμενη συνάρτηση, για να συνεχίσει με την εκτέλεση των επόμενων εντολών. Αντιθέτως, οι αντίστοιχες εκφράσεις τους που υλοποιούν την κατάσταση immediate MPI_Isend και MPI_Irecv (βλ. Παράγραφο 3.9.2.2),

υλοποιούν τον άμεσο τρόπο αποστολής και λήψης, σύμφωνα με τον οποίο το σύστημα δεν περιμένει να ολοκληρωθεί η αποστολή ή/και η λήψη των μηνυμάτων, αλλά συνεχίζει με την εκτέλεση των εντολών που έπονται της καλούμενης συνάρτησης, μέχρι να συναντήσει εντολή `MPI_Wait`. Στον κώδικα του προγράμματος, οι εντολές που βρίσκονται ανάμεσα στις εντολές immediate και στις `MPI_Wait` εκτελούνται ταυτόχρονα με την αποστολή/λήψη των δεδομένων. Στην περίπτωση μας, η συνάρτηση που εκτελείται ταυτόχρονα με την επικοινωνία, είναι η `compute()`.

Μετά από την πραγματοποίηση μερικών σειρών μετρήσεων, παρατηρήθηκε ότι οι δύο παραπάνω κώδικες είχαν τον ίδιο συνολικό χρόνο εκτέλεσης, γεγονός που δεν συμφωνούσε με τη θεωρητική μελέτη που είχε προηγηθεί. Αναζητώντας την αιτία της συγκεκριμένης ανακολουθίας, ανακαλύψαμε ότι, παρά τις υποδείξεις του προτύπου MPI, στις διάφορες υλοποίησεις του MPI δεν υπάρχει διαφοροποίηση ανάμεσα στις συναρτήσεις `MPI_Isend` και `MPI_Irecv` από τις `MPI_Send` και `MPI_Recv`. Συνεπώς, στην περίπτωση της συναρτήσεων immediate δεν επιτυγχάνονταν καθόλου επικάλυψη χρόνου υπολογισμών με χρόνο επικοινωνίας.

Στη συνέχεια, παρατηρήσαμε ότι αυτός ο συγκεκριμένος τρόπος υλοποίησης έχει μια λογική εξήγηση. Το MPICH είναι μία βιβλιοθήκη, δηλ. ένα σύνολο συναρτήσεων που εκτελείται στο χώρο χρήστη. Μια διεργασία που εκτελεί κώδικα του MPI σε έναν κόμβο, χρησιμοποιεί τις παραπάνω συναρτήσεις (`MPI_Send` και `MPI_Isend`) προκειμένου να επικοινωνήσει με μια άλλη απομακρυσμένη διεργασία. Για να επικοινωνήσει σε ένα περιβάλλον TCP/IP χρησιμοποιεί τη διεπιφάνεια των sockets (βλ. Παράγραφο 3.4). Ο κώδικας των sockets με τη σειρά του χρησιμοποιεί τις υπηρεσίες κατώτερων στρωμάτων λογισμικού επικοινωνίας που είναι τμήμα του πυρήνα του λειτουργικού συστήματος. Η ζητούμενη λειτουργικότητα της χρονικής επικάλυψης υπολογισμών και επικοινωνίας μπορεί να επιτευχθεί μόνο με επέμβαση στα χαμηλότερα στρώματα του λογισμικού επικοινωνίας. Οποιαδήποτε δοκιμή αλλαγής της συγκεκριμένης συμπεριφοράς από τα ανώτερα στρώματα (π.χ. εφαρμογές, βιβλιοθήκες, κλπ.) δεν μπορεί να επιφέρει το επιθυμητό αποτέλεσμα.

Προκειμένου να ξεπεράσουμε το συγκεκριμένο εμπόδιο και να προχωρήσουμε με τη σύγκριση των δύο μεθόδων χρονοδρομολόγησης, πάνω από την ευρέως χρησιμοποιούμενη δικτυακή τεχνολογία FastEthernet, συνεχίσαμε λαμβάνοντας την εξής απόφαση: Για την περίπτωση της μη-επικαλυπτόμενης δρομολόγησης, κάναμε χρήση της συνάρτησης σύγχρονης επικοινωνίας MPI_Ssend σύμφωνα με την οποία, η κλήση της συνάρτησης περατούται όταν βεβαιωθεί ότι τα δεδομένα που στάλθηκαν παρελήφθησαν από τον κόμβο προορισμού. Για την περίπτωση της επικαλυπτόμενης δρομολόγησης, ο κώδικας παρέμεινε με τις συναρτήσεις immediate. Η νέα υλοποίηση των δύο συγκρινόμενων μεθόδων χρονοδρομολόγησης διαφέρει από την προηγούμενη έκδοση μόνο ως προς το κομμάτι της επικοινωνίας των κόμβων, για το οποίο πραγματοποιείται ενός είδους προσομοίωση. Το υπόλοιπο κομμάτι της εφαρμογής παραμένει

ακριβώς το ίδιο.

5.1.2 Χαρακτηριστικά Στοιχεία Υλοποίησης

Στον Πίνακα 5.1, φαίνονται οι λειτουργίες που πραγματοποιούνται σε έναν κόμβο κατά τη διάρκεια τριών συνεχόμενων χρονικών βημάτων $k-1$, k και $k+1$, στην περίπτωση της επικαλυπτόμενης δρομολόγησης. Επίσης, παρουσιάζονται οι εξαρτήσεις δεδομένων μεταξύ των γειτονικών κόμβων, ανάμεσα στα διαδοχικά χρονικά βήματα.

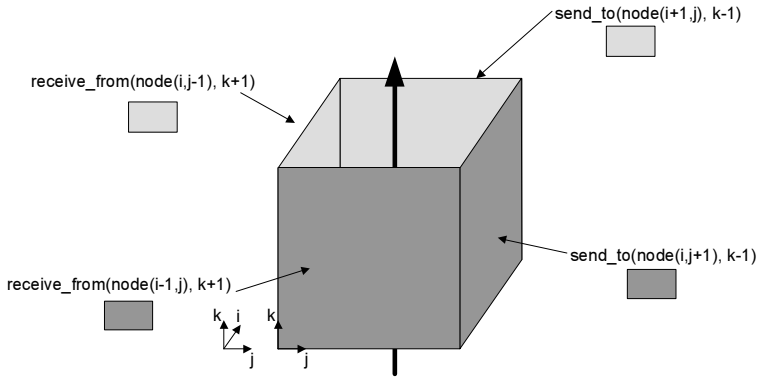
$k-1$	k	$k+1$
receive_from(node(i-1,j), k) receive_from(node(i,j-1), k)	receive_from(node(i-1,j), k+1) receive_from(node(i,j-1), k+1)	receive_from(node(i-1,j), k+2) receive_from(node(i,j-1), k+2)
compute(k-2, k)	compute(k-1, k+1)	compute(k, k+2)
send_to(node(i+1,j), k-2) send_to(node(i,j+1), k-2)	send_to(node(i+1,j), k-1) send_to(node(i,j+1), k-1)	send_to(node(i+1,j), k) send_to(node(i,j+1), k)

Πίνακας 5.1: Λειτουργίες που πραγματοποιούνται σε έναν υπερκόμβο και εξαρτήσεις δεδομένων μεταξύ των γειτονικών κόμβων στα χρονικά βήματα $k-1$, k και $k+1$, στην επικαλυπτόμενη δρομολόγηση. Η λειτουργία receive_from(node(i-1,j), k) σημαίνει ότι ο συγκεκριμένος κόμβος (i,j) λαμβάνει δεδομένα από το γειτονικό κόμβο $(i-1,j)$, δηλ. τον προηγούμενο στη διάσταση i κόμβο, και τα δεδομένα θα τα χρησιμοποιήσει για επεξεργασία το χρονικό βήμα k. Η λειτουργία compute(k-1, k+1) σημαίνει υπολογίζονται τα δεδομένα που ελήφθησαν το χρονικό βήμα $k-1$ και θα αποσταλούν το χρονικό βήμα $k+1$. Τέλος, η λειτουργία send_to(node(i+1,j), k-1) σημαίνει ότι αποστέλονται στο γειτονικό κόμβο $(i+1,j)$ τα δεδομένα που υπολογίστηκαν το χρονικό βήμα $k-1$.

Οι υλοποιήσεις των δύο συγκρινόμενων μεθόδων διαφέρουν και ως προς τις απαιτήσεις τους σε μνήμη. Συγκεκριμένα, ο προσωρινός αποθηκευτικός χώρος (buffer) στον οποίο βρίσκονται τα δεδομένα στην περίπτωση της λήψης και της αποστολής των μηνυμάτων είναι διαφορετικός σε κάθε περίπτωση.

Πιο «απλή» από πλευράς υλοποίησης και διαχείρισης μνήμης είναι η περίπτωση της μη-επικαλυπτόμενης δρομολόγησης. Στην περίπτωση αυτή, δεδομένου ότι, πρώτα λαμβάνονται δεδομένα, στη συνέχεια γίνονται υπολογισμοί που βασίζονται σε αυτά και έπειτα τα δεδομένα αυτά αποστέλλονται σε άλλο κόμβο, απαιτούνται μόνο $2 \times (n-1)$ προσωρινοί αποθηκευτικοί χώροι (buffers). Στην περίπτωση όπου $n=3$, οι συγκεκριμένοι αποθηκευτικοί χώροι φαίνονται στο Σχήμα 5.1. Ένας χώρος για τη λήψη και ένας χώρος για την αποστολή δεδομένων για κάθε μια απο τις διαστάσεις i και j. Για τη διάσταση k, δεν απαιτείται επιπλέον χώρος, μιας και τα δεδομένα δεν χρειάζεται να αποσταλούν σε άλλον κόμβο. Όλοι οι υπερκόμβοι κατά μήκος της διάστασης k απεικονίζονται στον ίδιο κόμβο.

Στην περίπτωση της υλοποίησης της επικαλυπτόμενης δρομολόγησης, χρησιμοποιείται η μέθοδος της *Διπλής Προσωρινής Αποθήκευσης* (double buffering). Δεδομένου ότι οι λειτουργίες της αποστολής, της λήψης και του υπολογισμού δεδομένων πραγματοποιούνται ταυτόχρονα,



Σχήμα 5.1: Προσωρινοί χώροι που απαιτούνται στην περίπτωση της μη-επικαλυπτόμενης δρομολόγησης. Οι υπερκόμβοι κατά μήκος της διάστασης k απεικονίζονται στον ίδιο κόμβο.

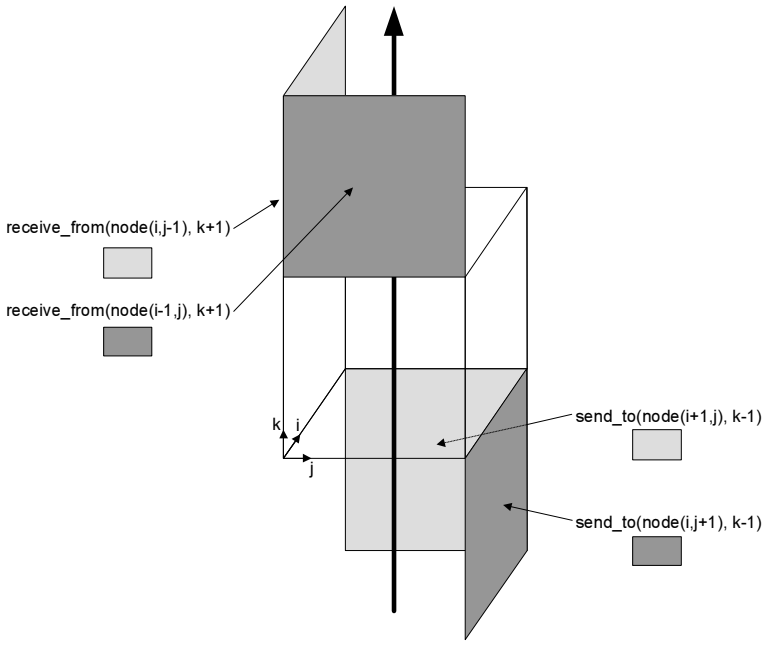
απαιτούνται διαφορετικοί χώροι αποθήκευσης δεδομένων ώστε να μην υπάρχει κίνδυνος να χαθούν δεδομένα (π.χ. ο κόμβος να λαμβάνει δεδομένα γρηγορότερα από ό,τι τα επεξεργάζεται).

Σύμφωνα με την προτεινόμενη μέθοδο, ο χώρος λήψης και αποστολής δεδομένων δεν είναι ο ίδιος σε κάθε χρονική στιγμή, ώστε να μην απαιτείται αντιγραφή δεδομένων από τον έναν χώρο στον άλλο. Για παράδειγμα, στην περίπτωση της λήψης δεδομένων από το γειτονικό κόμβο της διάστασης i, υπάρχουν δύο προσωρινοί αποθηκευτικοί χώροι. Σε κάθε χρονικό βήμα πραγματοποιείται εναλλαγή του χώρου στον οποίο αποθηκεύονται τα δεδομένα που λαμβάνονται. Προγραμματιστικά αυτό υλοποιείται με την εναλλαγή δύο δεικτών μνήμης (pointer swapping). Το συνολικό πλήθος των χώρων που απαιτούνται σε αυτή την περίπτωση είναι $2 \times 2 \times (n-1)$. Στο Σχήμα 5.2, φαίνονται οι επιπλέον χώροι που χρειάζονται στην περίπτωση της επικαλυπτόμενης δρομολόγησης για $n = 3$.

Από τα παραπάνω γίνεται σαφές ότι, προκειμένου να κερδίσουμε σε ταχύτητα εκτέλεσης, χρειαζόμαστε περισσότερη ποσότητα μνήμης. Όμως, είναι γεγονός ότι τα νέα υπολογιστικά συστήματα διαθέτουν και μπορούν να διαχειριστούν μεγάλη ποσότητα μνήμης RAM. Υπολογιστές με μέγεθος λέξης 64 bit, που μπορούν να διαχειριστούν εύκολα μέχρι και 2^{64} bytes ή 16 Hexabytes κεντρικής μνήμης, διατίθενται στην αγορά εδώ και μερικά χρόνια, ενώ η τιμή τους είναι πλέον αρκετά προσιτή, ώστε να χρησιμοποιούνται ως δομικά στοιχεία των νεώτερων συστοιχιών υπολογιστών. Θεωρούμε ότι η δέσμευση κάποιας επιπλέον ποσότητας μνήμης αποτελεί ένα λογικό tradeoff, προκειμένου να επιτύχουμε το επιθυμητό αποτέλεσμα.

5.2 Εφαρμογή - Βέλτιστος Υπερκόμβος

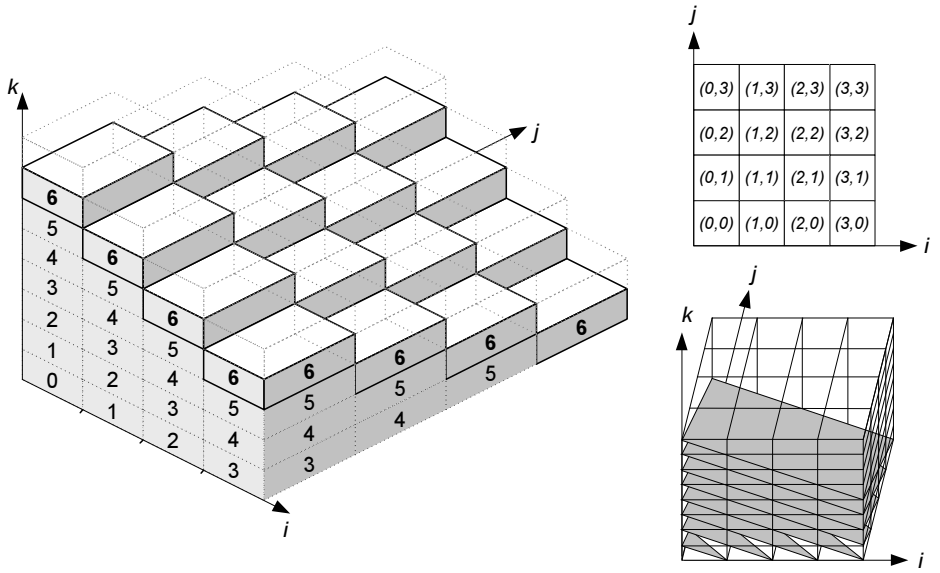
Ο κώδικας της εφαρμογής (βλ. Παράγραφο 5.1) που παραλληλοποιούμε είναι 3-διάστατος. Η πλατφόρμα εκτέλεσης των πειραμάτων διαθέτει 16 μονοεπεξεργαστικούς υπολογιστές, τους



Σχήμα 5.2: Προσωρινοί χώροι που απαιτούνται στην περίπτωση της επικαλυπτόμενης δρομολόγησης. Οι υπερκόμβοι κατά μήκος της διάστασης k απεικονίζονται στον ίδιο κόμβο.

οποίους χωρίζουμε σε ένα πλέγμα 4×4, στο επίπεδο ij. Είναι προφανές ότι με διάταξη του πλέγματος των επεξεργαστών διαφορετικής της τετραγωνικής, τα βήματα που θα απαιτούνταν για να αρχίσουν όλοι οι επεξεργαστές να υπολογίζουν στοιχεία του προβλήματος, θα ήταν περισσότερα. Στη χειρότερη περίπτωση, που οι επεξεργαστές δημιουργούν ένα πλέγμα 16×1, τότε απαιτούνται 16 χρονικά βήματα ώσπου να ξεκινήσουν όλοι τους υπολογισμούς.

Το βέλτιστο μέγεθος υπερκόμβου, για το οποίο ο συνολικός χρόνος εκτέλεσης της εφαρμογής είναι ο ελάχιστος, δεν είναι προφανής επιλογή. Αν θεωρήσουμε έναν υπερκόμβο μεγάλου μεγέθους (συγκρίσιμο με το μέγεθος του προβλήματος), τότε η εκτέλεση των πρώτων υπερκόμβων διαρκεί αρκετό χρόνο, με αποτέλεσμα να μην υπάρχει μεγάλο ποσοστό συνολικής παράλληλης εκτέλεσης (βλ. Σχήμα 5.4). Στην αντίθετη περίπτωση, που θεωρήσουμε πολύ μικρό μέγεθος υπερκόμβου, τότε αυξάνεται πολύ το ποσοστό επικοινωνίας στη συνολική εκτέλεση, με αποτέλεσμα η πραγματοποίηση πράξεων να αποτελεί ένα μικρό ποσοστό της επικοινωνίας. Στην περίπτωση αυτή, η περάτωση της συνολικής εκτέλεσης του προβλήματος, καθυστερεί επίσης. Το βέλτιστο μέγεθος υπερκόμβου είναι ανάμεσα στις δύο ακραίες περιπτώσεις, οι οποίες φαίνονται στο Σχήμα 5.4. Το χαρακτηριστικό της περίπτωσης του βέλτιστου μεγέθους υπερκόμβου είναι ότι επιτυγχάνεται υψηλός βαθμός παραλληλίας, χωρίς να υπάρχει περιττή επικοινωνία.

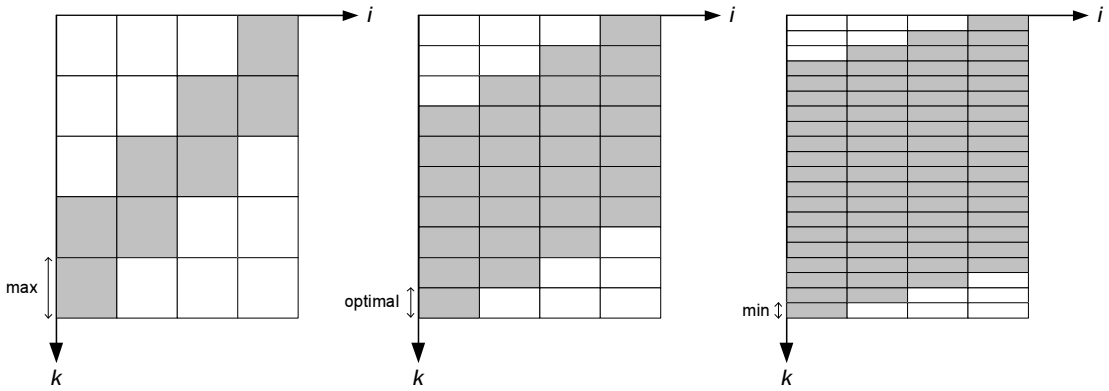


Σχήμα 5.3: Το πλέγμα των κόμβων και οι υπερκόμβοι που εκτελούνται κάθε χρονική στιγμή στην περίπτωση της μη επικαλυπτόμενης χρονοδρομολόγησης. Με έντονο περίγραμμα απεικονίζονται η υπερκόμβοι την πρώτη χρονική στιγμή που παρουσιάζεται ο μέγιστος βαθμός παραλληλισμού. Εκείνη τη χρονική στιγμή συμμετέχουν στο πρόβλημα όλοι οι κόμβοι· από τον (0,0) μέχρι τον (3,3).

5.3 Πειραματικές Μετρήσεις σε FastEthernet

Οι πειραματικές μετρήσεις εκτελούνται ως εξής: Ο κώδικας MPI που υλοποιεί τη μέθοδο της μη-επικαλυπτόμενης δρομολόγησης (βλ. Παράγραφο 5.1.1) εκτελείται για ένα πλήθος υπερκόμβων με διαφορετικό όγκο. Επειδή θεωρούμε υπερκόμβους με σταθερή βάση (διάσταση ij), η μεταβολή του όγκου πραγματοποιείται με μεταβολή μόνο του ύψους τους. Η εκτέλεση της εφαρμογής ξεκινάει με παράμετρο μία μικρή τιμή για το ύψος και καταλήγει σε ένα μία σχετικά μεγάλη τιμή. Στηριζόμενοι σε προηγούμενες εκτελέσεις, μεταβάλουμε το βήμα της εκτέλεσης κατά περίπτωση, ώστε η μέτρηση να γίνεται πιο λεπτομερής στα σημεία κοντά στο βέλτιστο ύψος υπερκόμβου, και λιγότερο λεπτομερής στα υπόλοιπα σημεία (κοντά στα άκρα).

Βέλτιστο ύψος (ή όγκος) υπερκόμβου είναι εκείνο για το οποίο ο συνολικός χρόνος εκτέλεσης ελαχιστοποιείται. Το βήμα αύξησης του ύψους είναι 2 για τα σημεία που παρουσιάζουν μεγαλύτερο ενδιαφέρον και μεγαλύτερο για τα υπόλοιπα. Αυτό σημαίνει ότι η εφαρμογή εκτελείται για ύψος k και στην επόμενη εκτέλεση για $k + 2$, κοκ. Πρέπει να αναφερθεί ότι για να βρεθεί ο χρόνος εκτέλεσης της εφαρμογής για όλες τις ενδιαφέρουσες τιμές του όγκου g του υπερκόμβου, κατά τη διάρκεια ενός και μόνο πειράματος η εφαρμογή εκτελείται πάνω από 1000 φορές. Κάθε πείραμα χαρακτηρίζεται από τον όγκο του συνολικού χώρου δεικτών υπερκόμβων. Στη συνέχεια παρατίθενται τα αποτελέσματα για τρεις διαφορετικούς χώρους



Σχήμα 5.4: Το ποσοστό παραλληλισμού που επιτυγχάνεται για διάφορα μεγέθη υπερκόμβων. Οι σκιασμένοι κόμβοι σημαίνουν ότι μπορούν να υπολογιστούν με το μεγαλύτερο βαθμό παραλληλίας.

υπερκόμβων: $16 \times 16 \times 16384$, $16 \times 16 \times 32768$ και $32 \times 32 \times 4096$.

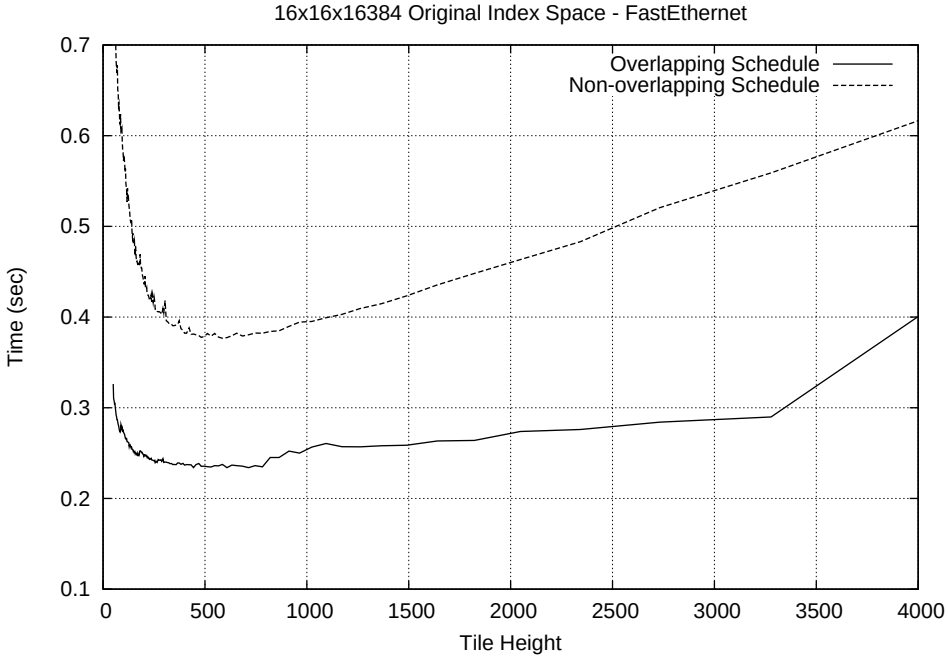
Ο χρόνος που διαρκεί η εκτέλεση της εφαρμογής λαμβάνεται με τη χρήση της κλήσης συστήματος `gettimeofday()`, η οποία επιστρέφει το χρόνο συστήματος με ακρίβεια εκατομμυριοστού του δευτερολέπτου. Ο πρώτος κόμβος που ξεκινάει την εκτέλεση (ο $(0,0)$ σε αυτές τις μετρήσεις), εκτελεί την `gettimeofday()` και λαμβάνει τη χρονική στιγμή που ξεκίνησε η εκτέλεση. Μετά τον υπολογισμό του συνολικού προβλήματος, ο τελικός κόμβος (ο $(3,3)$ σε αυτές τις μετρήσεις) ειδοποιεί τον $(0,0)$ ότι η εκτέλεση τελείωσε. Ο $(0,0)$ δεν γνωρίζει πότε ακριβώς τελειώνει η εκτέλεση, γιατί έχει σταματήσει να υπολογίζει μερικά βήματα νωρίτερα. Μετά την ειδοποίηση, λαμβάνει άλλη μία μέτρηση του χρόνου. Αφαιρώντας την αρχική από την τελική τιμή, βγαίνει ο συνολικός χρόνος εκτέλεσης. Η περίπτωση να καλέσει την `gettimeofday()` ο κόμβος $(3,3)$ και να στείλει την τιμή που έλαβε στον αρχικό, είναι ανακριβής διότι οι κόμβοι δεν έχουν όλοι την ίδια ακριβώς ώρα.

Για να ποσοτικοποιήσουμε τη βελτίωση που παρουσιάζει η προτεινόμενη μέθοδος επικαλυπτόμενης χρονοδρομολόγησης σε σχέση με τη συμβατική μέθοδο, ορίζουμε το μέγεθος της *επιτάχυνσης* ($speedup$) ως εξής:

$$speedup = \frac{T_{nonov} - T_{ov}}{T_{nonov}}, \tag{5.1}$$

όπου T_{nonov} είναι ο συνολικός χρόνος εκτέλεσης της εφαρμογής χρησιμοποιώντας τη μη-επικαλυπτόμενη χρονοδρομολόγηση και T_{ov} είναι ο συνολικός χρόνος εκτέλεσης της εφαρμογής χρησιμοποιώντας την προτεινόμενη επικαλυπτόμενη χρονοδρομολόγηση.

Στο Σχήμα 5.5, φαίνεται η σύγκριση των μεθόδων χρονοδρομολόγησης στην προαναφερθείσα συστοιχία υπολογιστών με δικτυακή τεχνολογία FastEthernet. Ο αρχικός χώρος δεικτών έχει

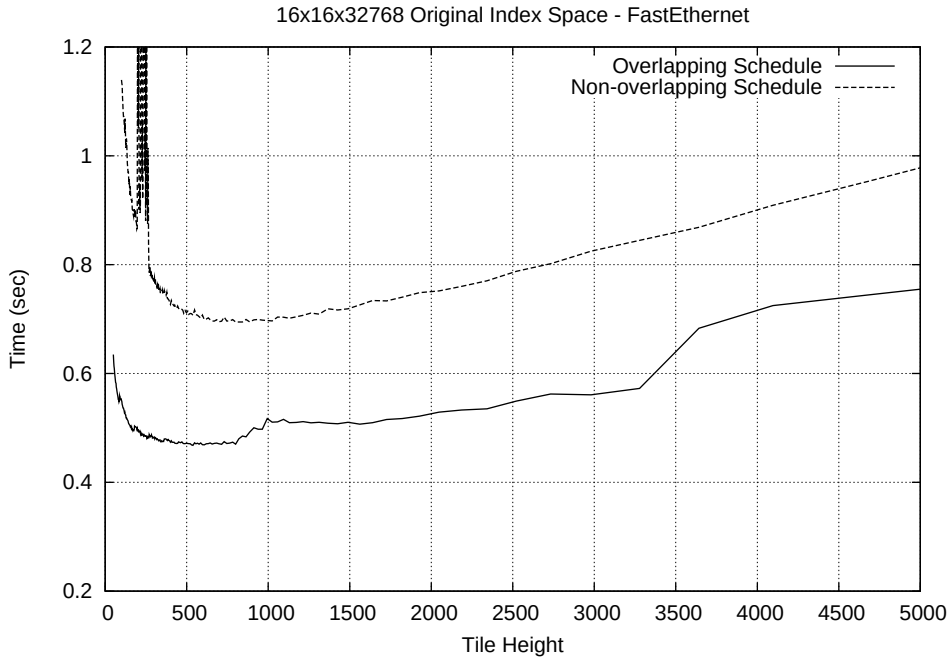


Σχήμα 5.5: Σύγκριση μεθόδων χρονοδρομολόγησης σε συστοιχία υπολογιστών διασυνδεδεμένη με FastEthernet για αρχικό χώρο δεικτών όγκου $16 \times 16 \times 16384$.

μέγεθος $16 \times 16 \times 16384$. Δεδομένου ότι η συστοιχία έχει 16 κόμβους, θεωρώντας τους κόμβους σαν ένα πλέγμα 4×4 κόμβων, κάθε υπερκόμβος έχει μέγεθος $4 \times 4 \times k$, όπου k είναι το ύψος του υπερκόμβου. Για διαφορετικά μεγέθη υπερκόμβων, η προτεινόμενη μέθοδος χρονοδρομολόγησης επιτυγχάνει καλύτερους συνολικούς χρόνους εκτέλεσης από αυτούς του συμβατικού τρόπου δρομολόγησης. Ο ελάχιστος χρόνος για την επικαλυπτόμενη περίπτωση 0.233923 sec επιτυγχάνεται για ύψος υπερκόμβου 444 (όγκος υπερκόμβου 7104). Ο ελάχιστος χρόνος για τη μη-επικαλυπτόμενη περίπτωση είναι 0.376637 sec και επιτυγχάνεται για ύψος υπερκόμβου ίσο με 586 (όγκος υπερκόμβου 9376). Η χρήση της προτεινόμενης μεθόδου επιτυγχάνει επιτάχυνση 37.9% στο συνολικό χρόνο εκτέλεσης σε σχέση με τη συμβατική μέθοδο.

Στο Σχήμα 5.6, φαίνεται η σύγκριση των μεθόδων χρονοδρομολόγησης για αρχικό χώρο δεικτών μεγέθους $16 \times 16 \times 32768$. Κάθε υπερκόμβος έχει μέγεθος $4 \times 4 \times k$, όπου k είναι το ύψος του υπερκόμβου. Ο ελάχιστος χρόνος για την επικαλυπτόμενη περίπτωση 0.467927 sec επιτυγχάνεται για ύψος υπερκόμβου 538 (όγκος υπερκόμβου 8608). Ο ελάχιστος χρόνος για τη μη-επικαλυπτόμενη περίπτωση είναι 0.694516 sec και επιτυγχάνεται για ύψος υπερκόμβου ίσο με 800 (όγκος υπερκόμβου 12800). Η χρήση της προτεινόμενης μεθόδου επιτυγχάνει επιτάχυνση 32.6% στο συνολικό χρόνο εκτέλεσης.

Στο Σχήμα 5.7, φαίνεται η σύγκριση των μεθόδων χρονοδρομολόγησης για αρχικό χώρο δεικτών μεγέθους $32 \times 32 \times 4096$. Κάθε υπερκόμβος έχει μέγεθος $8 \times 8 \times k$, όπου k είναι το



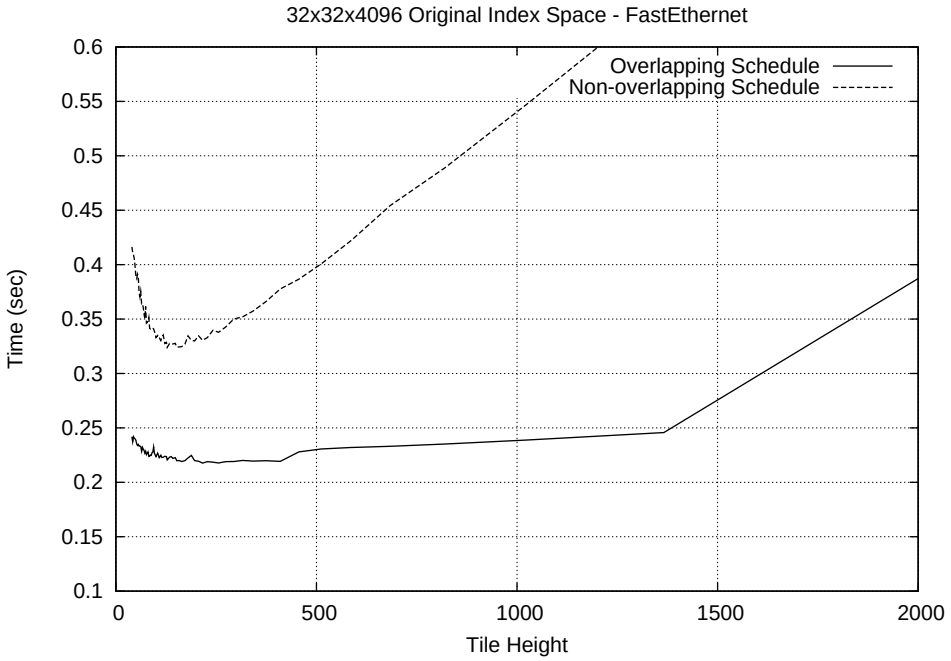
Σχήμα 5.6: Σύγκριση μεθόδων χρονοδρομολόγησης σε συστοιχία υπολογιστών διασυνδεδεμένη με FastEthernet για αρχικό χώρο δεικτών όγκου $16 \times 16 \times 32768$.

ύψος του υπερκόμβου. Ο ελάχιστος χρόνος για την επικαλυπτόμενη περίπτωση 0.219059 sec επιτυγχάνεται για ύψος υπερκόμβου 164 (όγκος υπερκόμβου 10496). Ο ελάχιστος χρόνος για τη μη-επικαλυπτόμενη περίπτωση είναι 0.324069 sec και επιτυγχάνεται για ύψος υπερκόμβου ίσο με 128 (όγκος υπερκόμβου 8192). Η χρήση της προτεινόμενης μεθόδου επιτυγχάνει επιτάχυνση 32.4% στο συνολικό χρόνο εκτέλεσης.

Τα αποτελέσματα για όλες τις περιπτώσεις που μετρήθηκαν για την περίπτωση της δικτυακής τεχνολογίας FastEthernet, φαίνονται στον Πίνακα 5.2.

Αρχικός Χώρος Δεικτών	$16 \times 16 \times 16384$	$16 \times 16 \times 32768$	$32 \times 32 \times 4096$
Βέλτιστο Ύψος Επικ.	444	538	164
Βέλτιστος Όγκος	7104	8608	10496
Βέλτιστος Χρόνος Μη-Επικ.	0.376637 sec	0.694516 sec	0.324069 sec
Βέλτιστος Χρόνος Επικ.	0.233923 sec	0.467929 sec	0.219059 sec
Επιτάχυνση	37.9%	32.6%	32.4%

Πίνακας 5.2: Σύνοψη αποτελεσμάτων για την περίπτωση της βιβλιοθήκης MPI πάνω από τη δικτυακή τεχνολογία FastEthernet.



Σχήμα 5.7: Σύγκριση μεθόδων χρονοδρομολόγησης σε συστοιχία υπολογιστών διασυνδεδεμένη με FastEthernet για αρχικό χώρο δεικτών όγκου $32 \times 32 \times 4096$.

5.4 Αξιολόγηση Αποτελεσμάτων FastEthernet

Τα πειραματικά αποτελέσματα χρησιμοποιώντας το MPI πάνω από δικτυακή τεχνολογία FastEthernet έδειξαν ότι η προτεινόμενη μέθοδος της επικαλυπτόμενη χρονοδρομολόγησης μπορεί να επιταχύνει σημαντικά την εκτέλεση τέτοιων εφαρμογών. Όμως, στο τμήμα της επικοινωνίας έγινε κάποιου είδους προσομοίωση, μιας και δεν μπορούσαμε να επικαλύψουμε πραγματικά χρόνο υπολογισμών με χρόνο επικοινωνίας.

Η υλοποίηση πάνω από τη δικτυακή τεχνολογία FastEthernet είχε τα εξής χαρακτηριστικό: Κώδικας MPI που υλοποιούσε την πραγματική εκτέλεσης των υπολογισμών συνδυάστηκε με προσωμοίωση της επικοινωνίας για την ζητούμενη μέθοδο χρονοδρομολόγησης. Το γεγονός ότι καταλήξαμε στην προσομοίωση του επιθυμητού τρόπου επικοινωνίας φανερώνει την αδυναμία επίτευξης επικάλυψης υπολογισμών και επικοινωνίας πάνω από συμβατικές δικτυακές τεχνολογίες.

Παρόλα αυτά, πιστεύουμε ότι ο παραπάνω τρόπος υλοποίησης είναι προτιμότερος από μια προσομοίωση της εκτέλεσης ολόκληρης της εφαρμογής. Στην περίπτωση αυτή, θα κατασκευάζαμε ένα ιδανικό σύστημα, το οποίο δεν θα λάμβανε υπόψη καμία από τις υπόλοιπες παραμέτρους του πραγματικού συστήματος (π.χ. ιεραρχία μνήμης, δίαυλος συστήματος, δίαυλος Ε/Ε, κλπ). Μπορεί το θεωρητικό μας μοντέλο να μην λαμβάνει υπόψη όλες τις δυνατές

παραμέτρους, αλλά με τη ρεαλιστική εκτέλεση του κώδικα όλες αυτές οι παράμετροι έχουν συμπεριληφθεί στον τελικό χρόνο που δίνουν τα πειραματικά αποτελέσματα.

Οι μετρήσεις πάνω από FastEthernet έδειξαν ότι δεν παρουσιάζεται κάποιο άλλο πρόβλημα στην εκτέλεση των εφαρμογών και ότι η πειραματική κατεύθυνση είναι σωστή. Προκειμένου να αποδείξουμε την ισχύ της θεωρία μας με πραγματική επικάλυψη υπολογισμών και επικοινωνίας, στραφήκαμε προς τη δικτυακή τεχνολογία SCI που δίνει στον προγραμματιστή μεγαλύτερη ευελιξία και δυνατότητες σε θέματα επικοινωνίας, συμπεριλαμβανομένης της δυνατότητας για επικάλυψη χρόνου υπολογισμών με χρόνο επικοινωνίας.

5.5 Περιβάλλον Εκτέλεσης Πειραμάτων SCI

Το υπολογιστικό περιβάλλον που χρησιμοποιήθηκε για την πειραματική σύγκριση των μεθόδων χρονοδρομολόγησης πάνω από τη δικτυακή τεχνολογίας SCI ήταν το εξής: Η συστοιχία υπολογιστών αποτελούνταν από 9 κόμβους με επεξεργαστές Pentium III στα 800 MHz. Η διασύνδεση SCI γινόταν με δικτυακούς προσαρμογείς D330 της εταιρείας Dolphin. Η τοπολογία του δικτύου ήταν δακτύλιος. Κάθε κόμβος περιείχε 128 MB μνήμης τυχαίας προσπέλασης. Το λειτουργικό σύστημα που εκτελούσε κάθε κόμβος ήταν το Linux με πυρήνα της σειρά 2.4.x.

Η εφαρμογή που παραλληλοποιήθηκε χρησιμοποιώντας το μετασχηματισμό υπερκόμβου ήταν η εξής:

```
for(i = 1; i < DIMX; i++){
   for(j = 1; j < DIMY; j++){
      for(k = 1; k < DIMZ; k++){
         A[i][j][k] = func(A[i-1][j][k],A[i][j-1][k],A[i][j][k-1]);
      }
   }
}
```

Οι 9 κόμβοι ήταν οργανωμένοι σε ένα πλέγμα επεξεργαστών 3×3 . Το βέλτιστο σχήμα υπερκόμβου είναι ορθογώνιο. Κάθε υπερκόμβος είναι ένα ορθογώνιο παραλληλεπίπεδο με πλευρές τις ij, ik και kj, όπως και στην περίπτωση του FasthEthernet. Χωρίς βλάβη της γενικότητας, επιλέγουμε τη διάσταση k να είναι η μεγαλύτερη, έτσι ώστε όλοι οι υπερκόμβοι κατά μήκος του άξονα k να απεικονίζονται στον ίδιο επεξεργαστή $P_i, i = (0, \ldots, 8)$. Κατά τη διάρκεια κάθε χρονικού βήματος, κάθε επεξεργαστής στο επίπεδο ij με συντεταγμένες (i, j) λαμβάνει δεδομένα από τους γειτονικούς επεξεργαστές $(i-1, j)$ και $(i, j-1)$, υπολογίζει και τα αποστέλλει στους επεξεργαστές $(i+1, j)$ και $(i, j+1)$.

Τα σημαντικότερα χαρακτηριστικά στοιχεία της δικτυακής τεχνολογίας SCI αναφέρθηκαν και αναλύθηκαν στην Παράγραφο 3.7. Όμως, έχει ενδιαφέρον να αναλύσουμε τον τρόπο με τον

οποίο προγραμματίστηκε η εφαρμογή που υλοποιεί τις δύο συγκρινόμενες μεθόδους χρονοδρομολόγησης, χρησιμοποιώντας το προγραμματιστικό περιβάλλον SISCI (βλ. Παράγραφο 3.9.1).

Στην περίπτωση του SCI, προτιμάμε οι ανταλλαγές δεδομένων να πραγματοποιούνται ως αποστολές δεδομένων. Η αποστολή δεδομένων, που πραγματοποιείται ως εναπόθεση δεδομένων στη μνήμη του παραλήπτη, πραγματοποιείται ταχύτερα. Αυτό συμβαίνει γιατί ο επεξεργαστής του αποστολέα απλώς αρχικοποιεί τη λειτουργία της αποστολής και στη συνέχεια ο προσαρμογέας δικτύου αναλαμβάνει να την ολοκληρώσει, μεταφέροντας τα δεδομένα στη μνήμη του παραλήπτη με Απομακρυσμένη Άμεση Προσπέλαση Μνήμης (βλ. Παράγραφο 3.9.3). Όμως στην περίπτωση της λήψης δεδομένων, ο ένας κόμβος πρέπει να ζητήσει τα δεδομένα από τον άλλο. Δηλαδή, λαμβάνει χώρα μια διαδικασία που περιλαμβάνει αφενός την ειδοποίηση του απομακρυσμένου προσαρμογέα και αφετέρου την αποστολή των δεδομένων από τον απομακρυσμένο κόμβο στον αρχικό.

Το SISCI παρέχει συναρτήσεις για επικοινωνία Προγραμματιζόμενης Ε/Ε και ΑΠΜ. Και οι δύο τρόποι υλοποιούν μονόπλευρη επικοινωνία (one-sided communication), όπου μόνο το ένα άκρο γνωρίζει ότι λαμβάνει χώρα η επικοινωνία. Για το λόγο αυτό, το SCI παρέχει ειδικές κλήσεις συχρονισμού με τις οποίες το ένα άκρο ειδοποιεί το άλλο για την κατάσταση που βρίσκεται η μεταξύ τους της επικοινωνίας. Οι κλήσεις αυτές ονομάζονται απομακρυσμένες διακοπές (remote interrupts).

Ας θεωρήσουμε ότι ο κόμβος Α περιμένει να λάβει δεδομένα από τον κόμβο Β. Όμως κατά τη διάρκεια της αποστολής δεδομένων, οι δύο κόμβοι εκτελούν υπολογισμούς, ακολουθώντας την επικαλυπτόμενη χρονοδρομολόγηση. Οι προσαρμογείς δικτύου SCI έχουν αναλάβει να διεκπεραιώσουν τη μεταφορά δεδομένων από τη μνήμη του Β στη μνήμη του Α, χωρίς να παρεμβάλλονται οι επεξεργαστές και τα λειτουργικά συστήματα των κόμβων. Όταν ο κόμβος Α ολοκληρώσει τους υπολογισμούς του, περνάει στο επόμενο χρονικό βήμα, κατά τη διάρκεια του οποίου χρησιμοποιεί τα δεδομένα που τοποθετήθηκαν στη μνήμη του. Πώς είναι σίγουρος ο κόμβος Α ότι η μεταφορά δεδομένων από τον κόμβο Β έχει ολοκληρωθεί; Την πληροφορία αυτή την έχει μόνο ο κόμβος Β, ο οποίος ενημερώνει τον Α με μία εντολή `SCITriggerInterrupt()`.

Συνεπώς, σε κάθε κόμβο και για κάθε χρονικό βήμα της εκτέλεσης της εφαρμογής, λαμβάνουν χώρα τα εξής: Ο κόμβος ειδοποιεί τους «προηγούμενους» κόμβους, $(i-1, j)$ και $(i, j-1)$, ότι είναι έτοιμος να δεχθεί τα δεδομένα στη μνήμη. Αυτό σημαίνει ότι ο χώρος στον οποίο θα εναποτεθούν τα δεδομένα δεν χρησιμοποιείται άμεσα. Στη συνέχεια περιμένει να ειδοποιηθεί από τους «επόμενους» κόμβους, $(i-1, j)$ και $(i, j-1)$, ότι αυτοί είναι έτοιμοι να δεχτούν τα δεδομένα που θα τους στείλει. Όταν συμβεί αυτό, ξεκινάει την αποστολή δεδομένων με ΑΠΜ. Όταν ο έλεγχος της επικοινωνίας περάσει στον προσαρμογέα δικτύου, τότε ο επεξεργαστής είναι ελεύθερος να πραγματοποιήσει τους υπολογισμούς της εφαρμογής. Τελειώνοντας τους

Σειρά Εκτέλεσης Συναρτήσεων	Αντιστοιχία Κλήσεων SISCI	Εξήγηση
`trigger_interrupt(n-1)`	`SCITriggerInterrupt()`	Πληροφόρησε τους προηγούμενους κόμβους ότι "είμαι έτοιμος να δεχτώ δεδομένα".
`wait_for_interrupt(n+1)`	`SCIWaitForInterrupt()`	Περίμενε μέχρι οι επόμενοι κόμβοι είναι έτοιμοι να δεχτούν δεδομένα.
`send_dma(n+1, data)`	`SCIPostDMAQueue()`	Πραγματοποίηση μεταφοράς ΑΠΜ στους γειτονικούς κόμβους.
`compute()`	`compute()`	Υπολογισμοί
`wait_for_dma()`	`SCIWaitForDMAQueue()`	Αναμονή να ολοκληρωθεί η μεταφορά ΑΠΜ.
`trigger_interrupt(n+1)`	`SCITriggerInterrupt()`	Πληροφόρησε τους επόμενους κόμβους ότι "τα δεδομένα σας έφτασαν".
`wait_for_interrupt(n-1)`	`SCIWaitForInterrupt()`	Περίμενε μέχρι οι προηγούμενοι κόμβοι να ολοκληρώσουν την μεταφορά τους.

Πίνακας 5.3: Ο εσωτερικός βρόχος του προγράμματος που υλοποιεί τη μέθοδο της επικαλυπτόμενης δρομολόγησης. Στην αριστερή στήλη παρουσιάζεται η σειρά των συναρτήσεων που εκτελεί το πρόγραμμα, στη μεσαία στήλη οι συναρτήσεις του λογισμικού SISCI που εκτελούν την πραγματική λειτουργία και στη δεξιά στήλη παρουσιάζεται η εξήγηση των λειτουργιών σε σχέση με την εκτελούμενη εφαρμογή.

υπολογισμούς, περιμένει να ειδοποιηθεί από τον τοπικό προσαρμογέα ότι η μεταφορά προς τους «επόμενους» κόμβους ολοκληρώθηκε. Έπειτα, ειδοποιεί τους «επόμενους» κόμβους ότι τα δεδομένα έχουν μεταφερθεί σε αυτούς. Οι υπολογισμοί πάνω στα δεδομένα που λάμβανε όλη αυτή την ώρα δεν μπορούν να ξεκινήσουν, παρά μόνο όταν ειδοποιηθεί από τους προηγούμενους κόμβους ότι η δική τους μεταφορά προς αυτόν έχει ολοκληρωθεί. Το εσωτερικό τμήμα του προγράμματος που υλοποιεί τη μέθοδο της επικαλυπτόμενης δρομολόγησης φαίνεται στον Πίνακα 5.3. Η μέθοδος της μη-επικαλυπτόμενης δρομολόγησης υλοποιείται με εναλλαγή των κλήσεων `compute()` και `send_dma(n+1, data)` στον κώδικα του προγράμματος.

Η προαναφερθείσα εφαρμογή εκτελέστηκε για ένα πλήθος αρχικών χώρων δεικτών `DIMX`×`DIMY`×`DIMZ`. Οι τυπικές τιμές για τις διαστάσεις `DIMX` και `DIMY` ήταν 12 και 24, ενώ για την `DIMZ` ήταν 256K, 512K, και 2048K. Μετρήσαμε τους χρόνους εκτέλεσης για τις ακόλουθες περιπτώσεις επικαλυπτόμενης και μη-επικαλυπτόμενης χρονοδρομολόγησης: $12 \times 12 \times 512K$, $24 \times 24 \times 256K$ και $24 \times 24 \times 2048K$.

Από τη σχέση (4.5) και χρησιμοποιώντας τα χαρακτηριστικά της επικαλυπτόμενης χρονοδρομολόγησης πάνω από δίκτυο SCI, έχουμε:

$$T_{ov}(z) = \left(2 \sum_{i \neq k} x_i + x_k + 1 \right) (t_{comp} + t_{start\_dma} + t_{sync}), \tag{5.2}$$

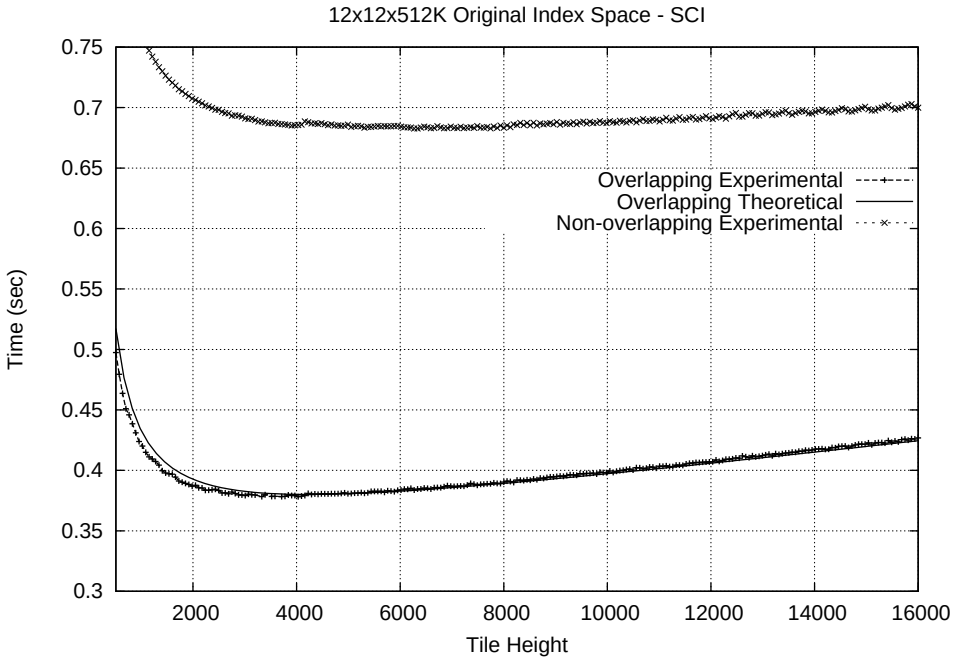
όπου για την περίπτωσή μας, επειδή υπάρχουν 3 επεξεργαστές σε κάθε διάσταση i και j, έχουμε $\sum_{i \neq k} x_i = 2 \times (3 - 1)$. Εφόσον το ύψος του αρχικού χώρου είναι $DIMZ$ και το ύψος

του υπερκόμβου z είναι η μεταβλητή του προβλήματος, υπάρχουν $\texttt{DIMZ}/z$ υπερκόμβοι στη διάσταση k. Συνεπώς, η τιμή του x_k είναι ίση με $\texttt{DIMZ}/z - 1$.

Η φάση επικοινωνίας ενός κόμβου με τους γειτονικούς του περιλαμβάνει την αποστολή ή τη λήψη $x_i \times z$ αριθμούς κινητής υποδιαστολής ή $4 \times x_i \times z$ bytes.

Για τις ανάγκες συγχρονισμού ανάμεσα στα χρονικά βήματα εκτέλεσης, οι κόμβοι πρέπει να ειδοποιηθούν μεταξύ τους χρησιμοποιώντας τις διακοπές SCI που επιβάλλουν σταθερή καθυστέρηση, $t_{sync} = 4 \times t_{sci\_interrupt}$. Εκτελέσαμε αρκετές δοκιμές επικοινωνίας ping-pong και υπολογίσαμε τις τιμές $t_{sci\_start\_dma} = 49.2\mu sec$ και $t_{sci\_interrupt} = 18.8\mu sec$.

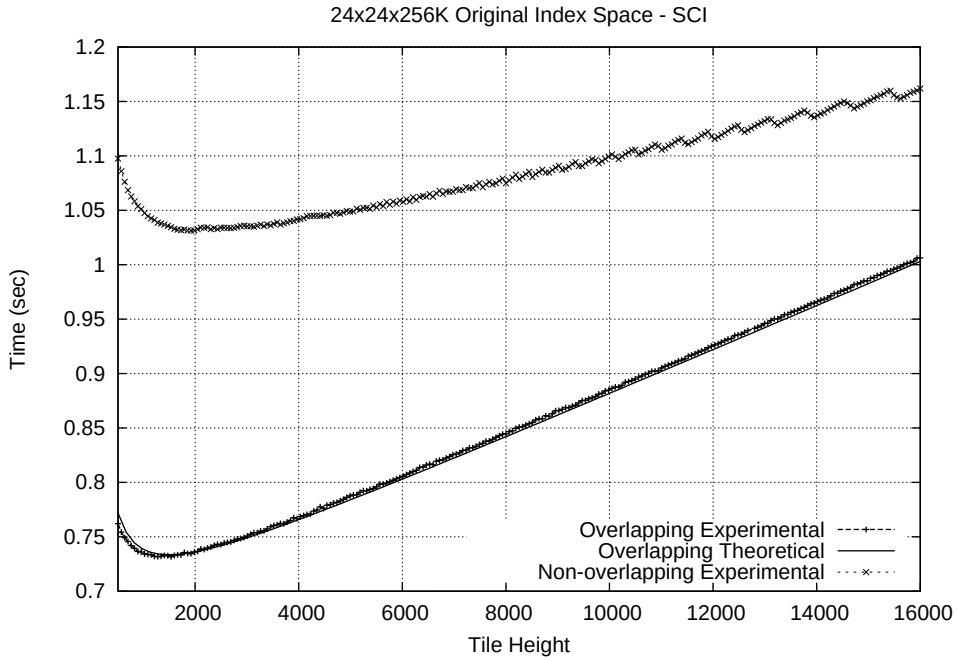
Οι συνολικοί χρόνοι εκτέλεσης για τον αρχικό χώρο δεικτών $12 \times 12 \times 512K$ φαίνονται στο Σχήμα 5.8. Είναι προφανές ότι οι πειραματικοί χρόνοι της προτεινόμενης χρονοδρομολόγησης είναι αρκετά μικρότεροι από τους αντίστοιχους της συμβατικής χρονοδρομολόγησης. Συγκεκριμένα, ο ελάχιστος χρόνος για την επικαλυπτόμενη περίπτωση 0.378205 sec επιτυγχάνεται για ύψος υπερκόμβου 4032 (όγκος υπερκόμβου 64512). Ο ελάχιστος χρόνος για τη μη-επικαλυπτόμενη περίπτωση είναι 0.682717 sec και επιτυγχάνεται για ύψος υπερκόμβου ίσο με 6336 (όγκος υπερκόμβου 101376). Η χρήση της προτεινόμενης μεθόδου, σύμφωνα με την σχέση 5.1, επιτυγχάνει επιτάχυνση 44.6% στο συνολικό χρόνο εκτέλεσης σε σχέση με τη συμβατική μέθοδο.



Σχήμα 5.8: Σύγκριση μεθόδων χρονοδρομολόγησης σε συστοιχία υπολογιστών διασυνδεδεμένη με SCI για αρχικό χώρο δεικτών όγκου $12 \times 12 \times 512K$.

Οι συνολικοί χρόνοι εκτέλεσης για τον αρχικό χώρο δεικτών $24 \times 24 \times 256K$ φαίνονται στο

Σχήμα 5.9. Ο ελάχιστος χρόνος για την επικαλυπτόμενη περίπτωση 0.731587 sec επιτυγχάνεται για ύψος υπερκόμβου 1536 (όγκος υπερκόμβου 98304). Ο ελάχιστος χρόνος για τη μη-επικαλυπτόμενη περίπτωση είναι 1.1031361 sec και επιτυγχάνεται για ύψος υπερκόμβου ίσο με 1920 (όγκος υπερκόμβου 122880). Η χρήση της προτεινόμενης μεθόδου επιτυγχάνει επιτάχυνση 29.06% στο συνολικό χρόνο εκτέλεσης σε σχέση με τη συμβατική μέθοδο.

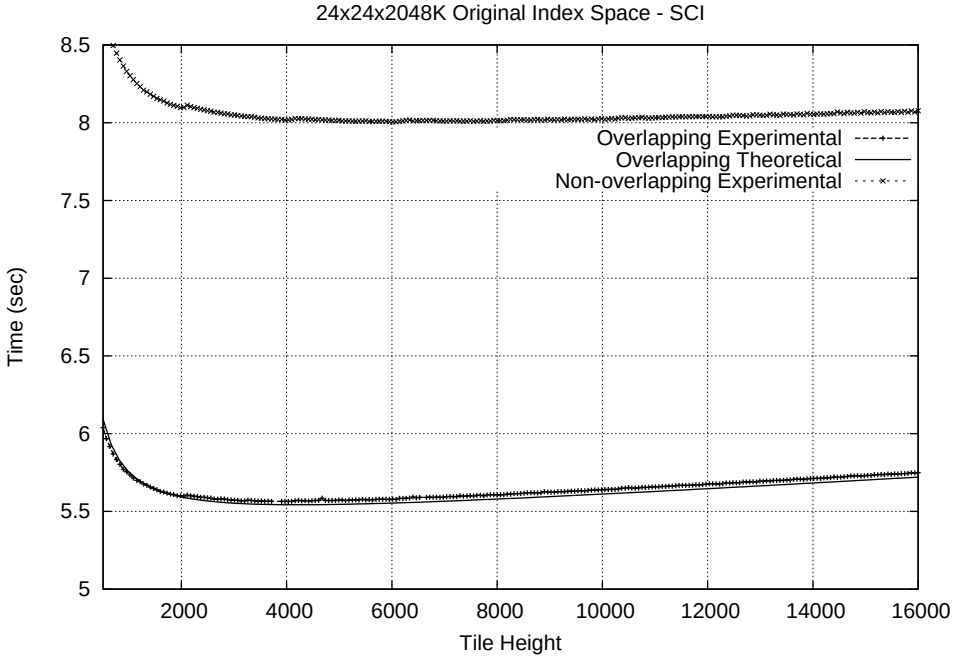


Σχήμα 5.9: Σύγκριση μεθόδων χρονοδρομολόγησης σε συστοιχία υπολογιστών διασυνδεδεμένη με SCI για αρχικό χώρο δεικτών όγκου $24 \times 24 \times 256K$.

Οι συνολικοί χρόνοι εκτέλεσης για τον αρχικό χώρο δεικτών $24 \times 24 \times 2048K$ φαίνονται στο Σχήμα 5.10. Ο ελάχιστος χρόνος για την επικαλυπτόμενη περίπτωση 5.563698 sec επιτυγχάνεται για ύψος υπερκόμβου 4032 (όγκος υπερκόμβου 258048). Ο ελάχιστος χρόνος για τη μη-επικαλυπτόμενη περίπτωση είναι 8.005366 sec και επιτυγχάνεται για ύψος υπερκόμβου ίσο με 6016 (όγκος υπερκόμβου 385024). Η χρήση της προτεινόμενης μεθόδου επιτυγχάνει επιτάχυνση 30.50% στο συνολικό χρόνο εκτέλεσης σε σχέση με τη συμβατική μέθοδο. Τα αποτελέσματα για όλες τις περιπτώσεις που μετρήθηκαν για την περίπτωση της δικτυακής τεχνολογίας SCI, φαίνονται στον Πίνακα 5.4.

Σε όλα τα παραπάνω Σχήματα εκτέλεσης των συγκρινόμενων μεθόδων χρονοδρομολόγησης φαίνεται και η γραφική παράσταση του χρόνου εκτέλεσης της επικαλυπτόμενης χρονοδρομολόγησης που βγαίνει από το θεωρητικό μας μοντέλο. Από τις γραφικές παραστάσεις συμπεραίνει κανείς ότι το θεωρητικό μας μοντέλο είναι αρκετά ακριβές.

Για να φανεί καλύτερα η ακρίβεια του, στην περίπτωση του αρχικού χώρου δεικτών $24 \times 24 \times$

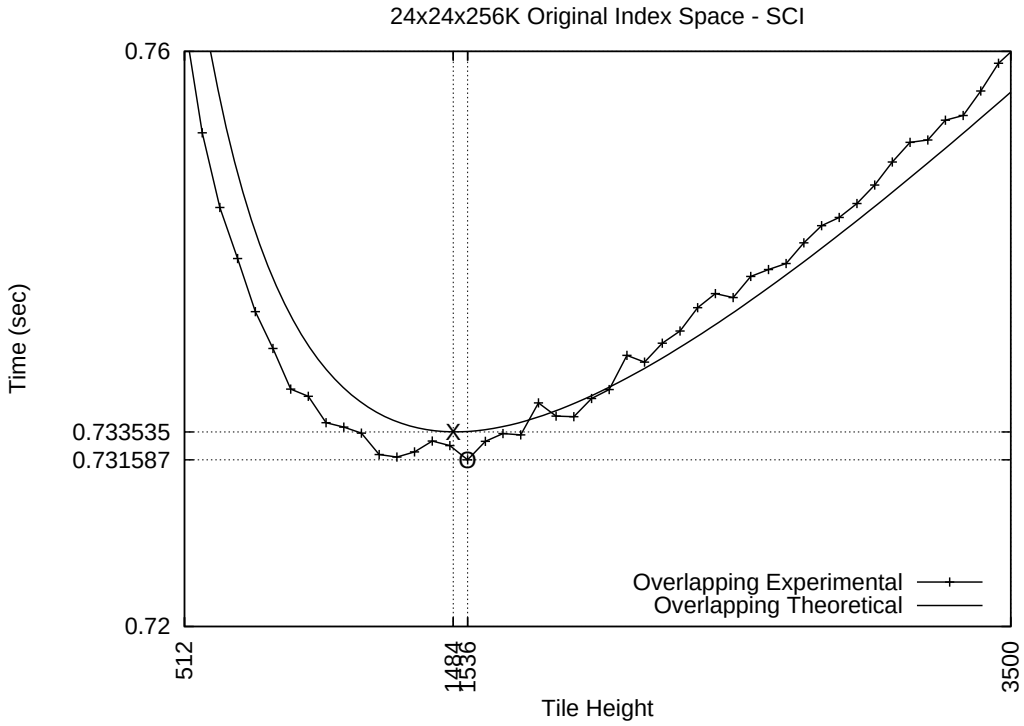


Σχήμα 5.10: Σύγκριση μεθόδων χρονοδρομολόγησης σε συστοιχία υπολογιστών διασυνδεδεμένη με SCI για αρχικό χώρο δεικτών όγκου $24 \times 24 \times 2048K$.

Αρχικός Χώρος Δεικτών	$12 \times 12 \times 512K$	$24 \times 24 \times 256K$	$24 \times 24 \times 2048K$
Βέλτιστο Ύψος Επικ.	4032	1536	4032
Βέλτιστος Όγκος	64512	98304	258048
Βέλτιστος Χρόνος Μη-Επικ.	0.682717 sec	1.1031361 sec	8.005366 sec
Βέλτιστος Χρόνος Επικ.	0.378205 sec	0.731587 sec	5.563698 sec
Επιτάχυνση	44.60%	29.06%	30.50%

Πίνακας 5.4: Σύνοψη αποτελεσμάτων για την περίπτωση της δικτυακής τεχνολογίας SCI.

$256K$ εστιάσαμε στο σημείο όπου βρίσκεται το βέλτιστο ύψος υπερκόμβου (βλ. Σχήμα 5.11). Δηλαδή, ο υπερκόμβος εκείνος για τον οποίο η εκτέλεση που ακολουθεί επικαλυπτόμενη δρομολόγηση διαρκεί τον ελάχιστο χρόνο. Όπως φαίνεται στο Σχήμα, ο πραγματικός ελάχιστος συνολικός χρόνος εκτέλεσης της εφαρμογής ήταν 0.731587 sec και συνέβη για ύψος υπερκόμβου ίσο με 1536. Ο θεωρητικός ελάχιστος συνολικός χρόνος εκτέλεσης του προβλήματος είναι 0.733535 sec και συμβαίνει για ύψος υπερκόμβου ίσο με 1484. Είναι προφανές ότι το θεωρητικό μοντέλο του συνολικού χρόνου της εκτέλεσης προσεγγίζει σημαντικά τον πραγματικό χρόνο.



Σχήμα 5.11: Μεγέθυνση του τμήματος που βρίσκεται το ελάχιστο της εκτέλεσης πάνω από SCI για αρχικό χώρο δεικτών όγκου $24 \times 24 \times 256K$.

5.6 Αξιολόγηση Αποτελεσμάτων SCI

Στα παραπάνω πειράματα χρησιμοποιήθηκε η δικτυακή τεχνολογία SCI που παρέχει τη δυνατότητα επικοινωνίας με ΑΠΜ. Επίσης, χρησιμοποιώντας Απομακρυσμένη ΑΠΜ γίνονται εναποθέσεις δεδομένων στη μνήμη ενός απομακρυσμένου κόμβου, χωρίς να γίνεται παρέμβαση της ΚΜΕ του. Συνεπώς, ο απομακρυσμένος κόμβος συνεχίζει να εκτελεί απερίσπαστος τους υπολογισμούς του. Χρησιμοποιώντας το λογισμικό SISCI, πετύχαμε πλήρη εκμετάλλευση των συγκεκριμένων δυνατοτήτων του SCI, οι οποίες μας επέτρεψαν να επικαλύψουμε χρόνο υπολογισμών με χρόνο επικοινωνίας.

Οι επιταχύνσεις που επιτύχαμε για τις διάφορες εκτελέσεις της εφαρμογής κυμαίνονται από ~30% μέχρι ~45%. Η επιτάχυνση δεν έφτασε ποτέ το θεωρητικό μέγιστο 50% για τους εξής λόγους:

1. Ο προγραμματισμός της μηχανής ΑΠΜ του προσαρμογέα SCI δεν γίνεται από το χώρο χρήστη, αλλά μέσω ειδικής κλήσης συστήματος, η οποία εισάγει καθυστέρηση.

2. Ο συγχρονισμός των διεργασιών που εκτελούν την εφαρμογή γίνεται με ειδικές διακοπές

του SCI, οι οποίες εισάγουν επίσης καθυστέρηση.

Πιστεύουμε ότι με τη χρήση άλλης δικτυακής τεχνολογίας (π.χ. τεχνολογία που ακολουθεί την Αρχιτεκτονική Εικονικού Προσαρμογέα), που θα προσέφερε τις παραπάνω λειτουργίες σε επίπεδο χρήστη, θα είχαμε φτάσει πολύ κοντά στο θεωρητικό μέγιστο της επιτάχυνσης.

Κεφάλαιο 6

Επίλογος

«Το μέλλον έχει ήδη φτάσει.
Απλά δεν είναι ακόμη ευρέως διαδεδομένο.»
– *William Gibson*

6.1 Περίληψη

Η παρούσα εργασία ασχολήθηκε με την αξιοποίηση των προηγμένων χαρακτηριστικών που προσφέρουν οι σύγχρονες δικτυακές τεχνολογίες, σε περιβάλλοντα συστοιχιών υπολογιστών, για την αποδοτικότερη παράλληλη εκτέλεση υπολογισμών.

Ξεκινώντας από τα προγράμματα που εκτελούνται στα υπολογιστικά συστήματα, δείξαμε ότι η μεγαλύτερη καθυστέρηση παρουσιάζεται στα κομμάτια του κώδικα που περιέχουν επαναλήψεις με τη μορφή φωλιασμένων βρόχων. Τα συγκεκριμένα σημεία ήταν ο στόχος της παραλληλοποίησης.

Η παραλληλοποίηση των τέλεια φωλιασμένων βρόχων, με τους οποίους ασχοληθήκαμε, δεν γίνεται πάντα με εύκολο τρόπο. Επικεντρωθήκαμε στις δύσκολες περιπτώσεις των τέλεια φωλιασμένων βρόχων DOACROSS. Η παραλληλοποίησή τους επιτεύχθηκε με τη χρησιμοποίηση του μετασχηματισμού υπερκόμβου, σύμφωνα με τον οποίο γειτονικά σημεία του αρχικού χώρου δεικτών του προβλήματος και οι αντίστοιχες εξαρτήσεις, ομαδοποιούνται σε υπερκόμβους, οι οποίοι εκτελούνται ατομικά.

Όσον αφορά το περιβάλλον εκτέλεσης παράλληλων εφαρμογών, επιλέξαμε τις συστοιχίες υπολογιστών, οι οποίες αποτελούν την επικρατούσα αρχιτεκτονική παράλληλων συστημάτων. Προσπαθήσαμε να βελτιώσουμε τις επιδόσεις των παραλλήλων προγραμμάτων σε τέτοιου είδους συστήματα, ακολουθώντας την τάση για δημιουργία συστημάτων παράλληλης επεξεργασίας συναρμολογημένα από κοινά και φτηνά εξαρτήματα.

Οι συμβατικές δικτυακές τεχνολογίες που χρησιμοποιούνται στο μεγαλύτερο πλήθος συστοιχιών υπολογιστών, παρουσιάζουν αρκετά μειονεκτήματα σε σχέση με τις νεώτερες τεχνολογίες που εμφανίζονται. Η εκτέλεση παράλληλων εφαρμογών πάνω από συμβατικές δικτυακές τεχνολογίες παρουσιάζει αρκετά προβλήματα απόδοσης, τα οποία οφείλονται κυρίως στην καθυστέρηση που εισάγει η στοίβα του λογισμικού επικοινωνίας επιπέδου χρήστη και επιπέδου πυρήνα.

Οι νέες δικτυακές τεχνολογίες, που έχουν προταθεί τα τελευταία χρόνια, προσπαθούν να παρακάμψουν τους κυριότερους παράγοντες που εισάγουν καθυστέρηση στην επικοινωνία, η οποία γίνεται ιδιαίτερα αισθητή σε περιβάλλοντα χαλαρά συνδεδεμένων επεξεργαστικών στοιχείων. Η μελέτη του τρόπου λειτουργίας των μοντέρνων δικτύων, μας ώθησε στην προσπάθεια εκμετάλλευσής τους από τη θεωρία της παραλληλοποίησης φωλιασμένων βρόχων.

Αν και οι σύγχρονες δικτυακές τεχνολογίες παρέχουν προηγμένα χαρακτηριστικά επικοινωνίας, μέσω των οποίων είναι δυνατή η αποδοτικότερη εκτέλεση παράλληλων εφαρμογών, αυτά δεν μπορούν να αξιοποιηθούν άμεσα λόγω των περιορισμών που τίθενται από το συμβατικό τρόπο δρομολόγησης. Η παρούσα εργασία πρότεινε ένα νέο τρόπο χρονοδρομολόγησης που αξιοποιεί πλήρως τα σύγχρονα χαρακτηριστικά των νέων τεχνολογιών. Εκμεταλλευόμενος τη δυνατότητα μεταφοράς των λειτουργιών της επικοινωνίας από την ΚΜΕ στο δικτυακό προσαρμογέα, ο προτεινόμενος τρόπος επιτυγχάνει την επικάλυψη χρόνου υπολογισμών με χρόνο επικοινωνίας, προκαλώντας επιτάχυνση των παράλληλων εφαρμογών που φτάνει θεωρητικά το 50% του χρόνου της συμβατικής εκτέλεσης.

Από τα αποτελέσματα των πειραματικών μετρήσεων συμπεραίνεται ότι η προτεινόμενη δρομολόγηση προσφέρει ικανοποιητική επιτάχυνση στις παράλληλες εφαρμογές τον χρησιμοποιούν. Επίσης, το θεωρητικό μοντέλο που χρησιμοποιήθηκε, λαμβάνει υπόψη του σε μεγάλη λεπτομέρεια τα χαρακτηριστικά των προβλημάτων και των δικτυακών τεχνολογιών, παρέχοντας αποτελέσματα πολύ κοντά στα πραγματικά.

Η παρούσα εργασία είναι η πρώτη στην επιστημονική βιβλιογραφία που λαμβάνει υπόψη προηγμένα χαρακτηριστικά των μοντέρνων δικτυακών τεχνολογιών στο μοντέλο της παράλληλης εκτέλεσης φωλιασμένων βρόχων μέσω του μετασχηματισμού υπερκόμβου.

6.2 Συμπεράσματα - Προτάσεις

6.2.1 Αυτόματη Παραλληλοποίηση

Η παρούσα διατριβή προσέθεσε ένα λίθο στο μεγάλο οικοδόμημα της αποδοτικής εκτέλεσης παράλληλων εφαρμογών. Η προτεινόμενη θεωρία, μαζί με άλλες προτάσεις που έχουν γίνει στον ευρύτερο χώρο, μπορούν να αξιοποιηθούν συνολικά για τη μείωση του χρόνου εκτέλεσης των φωλιασμένων βρόχων. Όμως, για να γίνουν οι χρήστες των παραλλήλων εφαρμογών αποδέκτες

των συγκεκριμένων βελτιώσεων, θα πρέπει να ασχοληθούν οι ίδιοι με τη βελτιστοποίηση των παράλληλων εφαρμογών τους, ενσωματώνοντας σε κάθε περίπτωση τις προτεινόμενες θεωρίες.

Αυτή τη στιγμή, είναι φανερή η απουσία εργαλείων τα οποία να παραλληλοποιούν αυτόματα σειριακό κώδικα. Τα υπάρχοντα εργαλεία παραλληλοποιούν είτε εξεζητημένους «τέλειους» κώδικες, είτε κώδικες που απαιτούν πολλές ενέργειες από την πλευρά του χρήστη ώστε να εκτελούνται αποδοτικά. Τις περισσότερες φορές, ο χρήστης αναγκάζεται να γίνει προγραμματιστής, αλλά σε αυτή του την πορεία δεν υιοθετεί τις προτάσεις για βελτίωση της απόδοσης.

Στόχος θα πρέπει να είναι η δημιουργία εργαλείων, τα οποία θα παραλληλοποιούν το σειριακό κώδικα και θα εξάγουν παραλληλοποιημένο κώδικα που θα μπορεί να εκτελεστεί αποδοτικά σε περιβάλλοντα συστοιχιών υπολογιστών.

6.2.2 Συστοιχίες Συμμετρικών Πολυεπεξεργαστών

Η παρούσα διατριβή πρότεινε ένα χρήσιμο τρόπο παραλληλοποίησης εφαρμογών, για την αποδοτική εκτέλεσή τους σε περιβάλλοντα συστοιχιών υπολογιστών, που αποτελούνται από μονοεπεξεργαστικούς κόμβους. Όμως, οι κόμβοι συμμετρικής πολυεπεξεργασίας (SMP) με μικρό πλήθος επεξεργαστών (μέχρι 4) αποτελούν πλέον ένα κοινό δομικό συστατικό των νέων συστοιχιών.

Προτείνεται η επέκταση της παρούσας εργασίας έτσι ώστε να συμπεριλάβει και την περίπτωση των μηχανών SMP. Μία κατεύθυνση προβληματισμού είναι, να αντιμετωπιστεί η επεξεργασία εντός του κόμβου SMP σαν ένα νέο παράλληλο σύστημα με τα δικά του ιδιαίτερα χαρακτηριστικά. Και σε αυτή την περίπτωση, θα πρέπει να αναζητηθεί ο συνολικός ελάχιστος χρόνος εκτέλεσης των παράλληλων εφαρμογών.

6.2.3 Άμεση Προσπέλαση Μνήμης σε Επίπεδο Χρήστη

Παρότι τα πειραματικά αποτελέσματα που παρήχθησαν κατά τη διάρκεια της διατριβής ήταν ικανοποιητικά, δεν έφτασαν ποτέ στη θεωρητική μέγιστη απόδοση. Ένα τμήμα της διαφοράς από το μέγιστο οφείλεται στη δικτυακή τεχνολογία που χρησιμοποιήθηκε, η οποία αν και δίνει αρκετές δυνατότητες στον προγραμματιστή, δεν παρείχε τη δυνατότητα επικοινωνίας με Άμεση Προσπέλαση Μνήμης σε Επίπεδο Χρήστη. Η αδυναμία αυτή πρόσθεσε αρκετή μη-επικαλυπτόμενη καθυστέρηση στην επικοινωνία και πιστεύουμε ότι κράτησε την απόδοση από το να φτάσει στο θεωρητικό μέγιστο.

Μελλοντικά, θα μπορούσε να διερευνηθεί εάν, με τη χρήση άλλης δικτυακής τεχνολογίας η οποία προσφέρει όλες τις δυνατότητες επικοινωνίας σε Επίπεδο Χρήστη (π.χ. δικτυακή τεχνολογία που να ακολουθεί την Αρχιτεκτονική Εικονικού Προσαρμογέα), τα πειραματικά αποτελέσματα φτάνουν πιο κοντά στο θεωρητικό μέγιστο της απόδοσης.

6.2.4 Λειτουργικά Συστήματα

Τα σύγχρονα λειτουργικά συστήματα που χρησιμοποιούνται από όλα τα υπολογιστικά συστήματα, δεν είναι βελτιστοποιημένα για περιβάλλοντα που ακολουθούν την αρχιτεκτονική της συστοιχίας υπολογιστών. Το λειτουργικό σύστημα, ένα αντίγραφο του οποίου εκτελείται σε κάθε κόμβο της συστοιχίας, αντιμετωπίζει τους υπόλοιπους κόμβους της συστοιχίας ως «ξένους». Αποτέλεσμα της συγκεκριμένης κατάστασης είναι ένα πολύπλοκο σύστημα διαδικασιών που παρεμβάλλονται για την επίτευξη της επικοινωνίας.

Αν και τα προηγμένα χαρακτηριστικά των νέων δικτύων παρέχουν δυνατότητες παράκαμψης αρκετών από τις χρονοβόρες λειτουργίες, δεν παύουν να είναι παρακάμψεις και όχι ο φυσιολογικός δρόμος για την αποδοτική επικοινωνία. Ίσως αυτός είναι ο λόγος που ο προγραμματισμός τέτοιου είδους συστημάτων παραμένει ακόμη δύσκολος, ενώ και η μερική ακόμα υιοθέτηση των δυνατοτήτων από πλατφόρμες γενικού σκοπού (π.χ. MPI) έρχεται με μεγάλη καθυστέρηση.

Γίνεται εμφανές ότι τα λειτουργικά συστήματα, ιδιαίτερα για τις συστοιχίες υπολογιστών, πρέπει να εκσυγχρονιστούν, έχοντας ως γνώμονα την αποδοτική και απλή επικοινωνία μεταξύ των κόμβων, καθώς και τη δυνατότητα εύκολης ολοκλήρωσής τους σε ενιαία συστήματα.

6.2.5 Αρχιτεκτονική Συστημάτων

Σημαντικό παράγοντα για την κατάσταση στην οποία βρίσκονται αυτή τη στιγμή τα λειτουργικά συστήματα, αποτελεί η υφιστάμενη αρχιτεκτονική των σταθμών εργασίας. Η επικοινωνία μεταξύ των διαφορετικών συστημάτων πραγματοποιείται μέσω συσκευών οι οποίες ονομάζονται περιφερειακές συσκευές ή συσκευές Εισόδου/Εξόδου. Από την ονομασία τους και μόνο μπορεί κανείς να αντιληφθεί ότι η επικοινωνία δεν είναι πρωτεύουσας σημασίας για το υπολογιστικό σύστημα, όπως π.χ. η σχέση του επεξεργαστή με τη μνήμη. Το γεγονός ότι σαν συσκευές Ε/Ε οι προσαρμογείς δικτύου βρίσκονται σε δευτερεύοντα δίαυλο μικρότερης συχνότητας και ταχύτητας από τον κεντρικό δίαυλο του συστήματος, αποδεικνύει τη μικρή σημασία που δίνεται στην αποδοτική επικοινωνία.

Μελλοντικά, πρέπει να γίνει συνολική αναθεώρηση της αρχιτεκτονικής των προσωπικών υπολογιστών και σταθμών εργασίας, βασισμένη στην εξίσωση της σημασίας επικοινωνίας του επεξεργαστή με όλα τα υπόλοιπα συστήματα (π.χ. άλλοι επεξεργαστές, μνήμη, περιφερειακά, κλπ.) Είναι κοινή πεποίθηση ότι η συγκεκριμένη εποχή δεν είναι πολύ μακριά, λαμβάνοντας υπόψη ότι νέες αρχιτεκτονικές και πρότυπα (όπως π.χ. το Infiniband) αρχίζουν να εμφανίζονται. Σύμφωνα με αυτά, δίνεται η ίδια βαρύτητα στην επικοινωνία μεταξύ όλων των επιμέρους συστημάτων, τα οποία αντιμετωπίζονται ως ισότιμα.

6.2.6 Επικάλυψη Εργασιών

Η ταυτόχρονη εκτέλεση υπολογισμών με επικοινωνία ήταν η βασική ιδέα που ώθησε την πραγματοποίηση της παρούσας εργασίας. Η διείσδυση των δικτύων σε πολλούς τομείς των ηλεκτρονικών μας αναγκών θεωρούμε ότι δίνει την δυνατότητα για περαιτέρω έρευνα στο πεδίο της εκμετάλλευσης της χρονικής επικάλυψης επικοινωνίας με άλλου είδους διαδικασίες.

Για παράδειγμα, η δικτυακή αποθήκευση δεδομένων λαμβάνει τρομακτικές διαστάσεις. Πολλές εταιρείες παρέχουν συστήματα για απομακρυσμένη αποθήκευση δεδομένων σε πολλαπλούς δίσκους, πάνω από ειδικές δικτυακές τεχνολογίες. Αν ληφθούν υπόψη οι ανάγκες των εφαρμογών που κάνουν χρήση των συγκεκριμένων συστημάτων, ενδεχομένως να είναι έχει νόημα η έρευνα στην κατεύθυνση της επικάλυψης χρόνου επικοινωνίας με χρόνο αποθήκευσης δεδομένων.

6.2.7 Σημασία Υποσυστήματος Μνήμης

Στην παρούσα εργασία, η δυνατότητα παροχής της μνήμης θεωρήθηκε άπειρη και συνεπώς μπορούσε να τροφοδοτήσει ταυτόχρονα και τον επεξεργαστή, που εκτελεί εργασίες υπολογισμού, αλλά και τον προσαρμογέα δικτύου, που εκτελεί λειτουργίες επικοινωνίας. Η συγκεκριμένη παραδοχή έγινε, διότι μετά από ειδικές μετρήσεις παρατηρήθηκε ότι, ο ρυθμός παροχής δεδομένων (throughput), που χρειάζεται μία υπολογιστικά πολύ δραστήρια διεργασία, αποτελεί ένα μικρό ποσοστό της δυνατότητας παροχής δεδομένων (bandwidth) της κεντρικής μνήμης.

Ενδεχομένως, με την αύξηση του πλήθους των επεξεργαστών σε έναν υπολογιστικό κόμβο ή/και του πλήθους των προσαρμογέων δικτύων που βρίσκονται σε αυτόν, να παρουσιαστούν προβλήματα στενωπού στη δυνατότητα παροχής της μνήμης. Στην περίπτωση αυτή, θα πρέπει να συμπεριληφθεί στο θεωρητικό μοντέλο και ο συγκεκριμένος παράγοντας, καθώς και πώς αυτός επηρεάζεται από τη λειτουργία των διαφόρων στοιχείων. Είναι λογικό, όταν παρουσιάζεται στενωπός στην εξυπηρέτηση από την κεντρική μνήμη, κάποια από τα παραπάνω στοιχεία να υπολειτουργούν, δίνοντας νέα διάσταση στο πρόβλημα της ελαχιστοποίησης του χρόνου εκτέλεσης.

Παράρτημα A

Παράδειγμα Προγραμματισμού MPI

Στη συνέχεια παραθέτουμε δύο παραδείγματα προγραμματισμού στην πλατφόρμα παράλληλου προγραμματισμού Message Passing Interface (MPI). Έστω ότι θέλουμε να παραλληλοποιήσουμε τον παρακάτω κώδικα, ο οποίος υπολογίζει την παράσταση $f(0) + f(1)$, όπου $f(x)$ μια συνάρτηση η οποία χρειάζεται πολύ χρόνο για να εκτελεστεί.

```
int main(int argc,char** argv){
   return (f(0)+f(1));
}
```

Ας υποθέσουμε ότι διαθέτουμε ένα παράλληλο σύστημα με 2 κόμβους, τους 0 και 1. Δίνουμε δύο διαφορετικά προγράμματα σε MPI υπολογίζουν τη ζητούμενη παράσταση.

A.1 Πρώτη Εκδοχή – Επικοινωνία Σημείο-προς-σημείο

Η πρώτη εκδοχή του κώδικα MPI, η οποία χρησιμοποιεί εντολές επικοινωνίας σημείο-προς-σημείο, είναι η εξής:

```
#include <mpi.h>

int main(int argc,char** argv){
   int v0,v1,sum,rank;

   MPI_Status stat;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);

   if(rank == 1)
      MPI_Send(&f(1),1,0,50,MPI_INT,MPI_COMM_WORLD);
```

```
   else
   if(rank == 0){
      v0 = f(0);
      MPI_Recv(&v1,1,1,50,MPI_INT,MPI_COMM_WORLD,&stat);
      sum = v0 + v1;
      return sum;
   }
   MPI_Finalize();
}
```

Α.2 Δεύτερη Εκδοχή – Συλλογική Επικοινωνία

Η δεύτερη εκδοχή του κώδικα MPI, η οποία χρησιμοποιεί εντολές συλλογικής επικοινωνίας, είναι η εξής:

```
#include <mpi.h>

int main(int argc,char** argv){
   int sum,rank;

   MPI_Status stat;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    MPI_Reduce(&f(rank),&sum,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
   return sum;
   MPI_Finalize();
}
```

Παράρτημα B

Κώδικας Τυπικών Εφαρμογών

B.1 Alternating Direction Implicit Integration – ADI

Η μέθοδος ADI χρησιμοποιείται για την επίλυση μερικών διαφορικών εξισώσεων. Πρόκειται για ένα πρόβλημα 3 διαστάσεων. Αποτελείται από 2 βήματα· στο πρώτο πραγματοποιούνται πράξεις κατά μήκος της διάστασης i και στο δεύτερο κατά μήκος της διάστασης j. Ενοποιώντας τα δύο βήματα, προκύπτει ο εξής φωλιασμένος βρόχος:

```
for (t = 1; t <= T; t++){
   for (i = 1; i <= I; i++){
      for (j = 1; j <= J; j++){
         X[t,i,j] = X[t-1,i,j] + X[t-1,i,j-1]*A[i,j]/B[t-1,i,j-1] -
                                 X[t-1,i-1,j]*A[i,j]/B[t-1,i-1,j];
         B[t,i,j] = B[t-1,i,j] - A[i,j]/B[t-1,i,j-1] -
                                 A[i,j]*A[i,j]/B[t-1,i-1,j];
      } /* j */
   } /* i */
} /* t */
```

B.2 Global Sequence Alignment: Αλγόριθμος Fickett

Ο παρακάτω κώδικας υλοποιεί τον αλγόριθμο Fickett και λύνει το πρόβλημα του Global Sequence Alignemt [AB03]. Το πρόβλημα προέρχεται από το χώρο της υπολογιστικής μοριακής βιολογίας και αυτή τη στιγμή αποτελεί ένα από τα πιο συχνά προβλήματα που επιλύονται στον συγκεκριμένο τομέα. Πρόκειται για ένα πρόβλημα 2 διαστάσεων του οποίου ο κώδικας είναι ο εξής:

```
for (i = 1; i <= I; i++){
    for (j = 1; j <= J; j++){
        l[i,j] = min(l[i-1,j-1] + f(a_i, b_j), l[i,j-1] + f(⊔, b_j),
                     l[i-1,j] + f(a_i, ⊔));
    } /* j */
} /* i */
```

Παράρτημα Γ

Παράδειγμα Προγραμματισμού SISCI

Η διεπαφή του SISCI, η πλέον γνωστή διεπαφή για προγραμματισμό σε δίκτυο SCI, απαιτεί περισσότερη προσπάθεια από τον προγραμματιστή. Προγραμματισμός σε SISCI αντιστοιχεί με τον προγραμματισμό σε sockets για τα δίκτυα TCP/IP. Παρακάτω παρουσιάζουμε 2 προγράμματα που υλοποιούν επικοινωνία Προγραμματιζόμενης Ε/Ε και "μεσης Προσπέλασης Μνήμης, αντίστοιχα, πάνω από SCI.

Γ.1 Παράδειγμα Επικοινωνίας με Προγραμματιζόμενη Ε/Ε

Πρόκειται για μια απλή εφαρμογή, στην οποία υλοποιούνται 2 άκρα επικοινωνίας. Το ένα άκρο (αποστολέας) στέλνει, πάνω από Κατανεμημένη Μοιραζόμενη, μία εντολή στο άλλο (παραλήπτης). Ο παραλήπτης, με τη λήψη της εντολής, τυπώνει ένα μήνυμα.

Στα παρακάτω προγράμματα, έχουν αφαιρεθεί οι εντολές ελέγχου σφαλμάτων, για να γίνει πιο ευανάγνωστος ο κώδικας.

Γ.1.1 Κώδικας Παραλήπτη

```
/* Κώδικας Παραλήπτη */
#include "sisci_api.h"
#include <stdio.h>
#define RECEIVER_MEM_ID 4
#define RECEIVER_MEM_SIZE 4096
#define ADAPTER_NO 0
#define NO_CALLBACK 0
#define NO_ARG 0
#define NO_FLAGS 0
#define PRINT_COMMAND 1
```

```
int main(int argc, char* argv[])
{
   sci_desc_t v_dev;
   sci_error_t error;
   sci_local_segment_t local_segment;
   sci_map_t local_map;
   int* local_address;

   /* Αρχικοποίηση του περιβάλλοντος SCI */
   SCIInitialize(NO_FLAGS, &error);

   /* Δημιουργία εικονικής συσκευής */
   SCIOpen(&v_dev, NO_FLAGS, &error);

   /* Δέσμευση τοπικού τμήματος μνήμης */
   SCICreateSegment(v_dev, &local_segment,
                    RECEIVER_MEM_ID, RECEIVER_MEM_SIZE,
                    NO_CALLBACK, NO_ARG, N

   /* Απεικόνιση του τοπικού τμήματος μνήμης στο χώρο εικονικών
      διευθύνσεων της διεργασίας */
   local_address = (int*)SCIMapLocalSegment(local_segment, &local_map,
                                    0 /* offset */, RECEIVER_MEM_SIZE,
                                    0 /* address hint */, NO_FLAGS, &error);

   /* Αρχικοποίηση των περιεχομένων της πρώτης λέξης του μοιραζόμενου τμήματος */
   *local_address = ~PRINT_COMMAND;

   /* Εξαγωγή τοπικού τμήματος μνήμης */
   SCIPrepareSegment(local_segment, ADAPTER_NO, NO_FLAGS, &error);
   SCISetSegmentAvailable(local_segment, ADAPTER_NO, NO_FLAGS, &error);

   /* Ενεργός αναμονή μέχρι να σταλεί η εντολή PRINT */
   while (*l_addr != PRINT_COMMAND) ;
   printf("Hello, World!");

   /* Καθαρισμός */
   SCISetSegmentUnavailable(segment, ADAPTER_NO, NO_FLAGS, &error);
   SCIRemoveSegment(local_segment, NO_FLAGS, error);
   SCIClose(v_dev, NO_FLAGS, &error);
```

```
   SCITerminate();
   return 0;
}
```

Γ.1.2 Κώδικας Αποστολέα

```
/* Κώδικας του αποστολέα */
#include "sisci_api.h"
#define RECEIVER_NODE_ID 4
#define RECEIVER_MEM_ID 4
#define ADAPTER_NO 0
#define NO_CALLBACK 0
#define NO_ARG 0
#define NO_FLAGS 0
#define PRINT_COMMAND 1

int main(int argc, char* argv[])
{
   sci_desc_t v_dev;
   sci_error_t error;
   sci_remote_segment_t remote_segment;
   unsigned int remote_segment_size;
   sci_map_t remote_map;
   volatile int* remote_address;

   /* Αρχικοποίηση του περιβάλλοντος SCI */
   SCIInitialize(NO_FLAGS, &error);

   /* Δημιουργία εικονικής συσκευής */
   SCIOpen(&v_dev, NO_FLAGS, &error);

   /* Σύνδεση με το απομακρυσμένο τμήμα μνήμης */
  SCIConnectSegment(v_dev, &remote_segment, RECEIVER_NODE_ID, RECEIVER_MEM_ID,
                    ADAPTER_NO, NO_CALLBACK, NO_ARG,
                    SCI_INFINITE_TIMEOUT, NO_FLAGS, &error);
   remote_segment_size = SCIGetRemoteSegmentSize(remote_segment);

  /* Απεικόνιση του απομεκρυσμένου τμήματος μνήμης στο χώρο εικονικών
     διευθύνσεων της διεργασίας */
   remote_address = (volatile int*)
                   SCIMapRemoteSegment(remote_segment, &remote_map,
```

```
                                    0 /* offset */, remote_segment_size,
                                    0 /* address hint */, NO_FLAGS, &error);

   /* Αποστολή της εντολής PRINT */
   *remote_address = PRINT_COMMAND;

   /* Καθαρισμός */
   SCIUnmapSegment(remote_map, NO_FLAGS, &error);
   SCIDisconnectSegment(remote_segment, NO_FLAGS, &error);
   SCIClose(v_dev, NO_FLAGS, &error);
   SCITerminate();
   return 0;
}
```

Γ.2 Παράδειγμα Επικοινωνίας με Άμεση Προσπέλαση Μνήμης

Γ.2.1 Κώδικας Αποστολέα

```
/* Κώδικας του αποστολέα */
#include "sisci_api.h"
#define RECEIVER_NODE_ID 4
#define RECEIVER_MEM_ID 4
#define SENDER_MEM_ID 4
#define SENDER_MEM_SIZE 4
#define ADAPTER_NO 0
#define NO_CALLBACK 0
#define NO_ARG 0
#define NO_FLAGS 0
#define PRINT_COMMAND 1

int main(int argc, char* argv[])
{
   sci_desc_t v_dev;
   sci_error_t error;
   sci_remote_segment_t remote_segment;
   unsigned int remote_segment_size;
   sci_local_segment_t local_segment;
   sci_map_t local_map;
   int* local_address;
   sci_dma_queue_t dma_queue;
   sci_dma_queue_state_t dma_queue_state;
```

```
/* Αρχικοποίηση του περιβάλλοντος SCI */
SCIInitialize(NO_FLAGS, &error);

/* Δημιουργία εικονικής συσκευής */
SCIOpen(&v_dev, NO_FLAGS, &error);

/* Δέσμευση τοπικού τμήματος μνήμης */
SCICreateSegment(v_dev, &local_segment, SENDER_MEM_ID, SENDER_MEM_SIZE,
                 NO_CALLBACK, NO_ARG, NO_FLAGS, &error);

/* Απεικόνιση του τοπικού τμήματος μνήμης στο χώρο εικονικών
   διευθύνσεων της διεργασίας */
local_address = (int*)
               SCIMapLocalSegment(local_segment, &local_map,
                                  0 /* offset */, SENDER_MEM_SIZE,
                                0 /* address hint */, NO_FLAGS, &error);

/* Δημιουργία ουράς ΑΠΜ */
SCICreateDMAQueue(v_dev, &dma_queue, ADAPTER_NO,
                  1 /* max entries */, NO_FLAGS, &error);

/* Σύνδεση στο απομακρυσμένο τμήμα μνήμης */
SCIConnectSegment(v_dev, &remote_segment,
                  RECEIVER_NODE_ID, RECEIVER_MEM_ID,
                  ADAPTER_NO, NO_CALLBACK, 0,
                  SCI_INFINITE_TIMEOUT, NO_FLAGS, &error);

/* Αρχικοποίηση του τοπικού τμήματος μνήμης, ώστε να περιέχει
   την εντολή PRINT */
*local_address = PRINT_COMMAND;

/* Προσθήκη της μεταφοράς δεδομένων στην ουρά ΑΠΜ */
SCIEnqueueDMATransfer(dma_queue, local_segment, remote_segment,
                      0 /* local offset */, 0 /* remote offset */,
                       sizeof(PRINT_COMMAND) /* transfer size */,
                       NO_FLAGS, &error);

/* Πέρασμα της ουράς ΑΠΜ στον προσαρμογέα SCI */
/* Αυτό πραγματοποιεί τη μεταφορά */
SCIPostDMAQueue(dma_queue, NO_CALLBACK, NO_ARG, NO_FLAGS, &error);
```

```
   /* Αναμονή για ολοκλήρωση επεξεργασίας ουράς ΑΠΜ */
   SCIWaitForDMAQueue(dma_queue, INFINITE_TIMEOUT, NO_FLAGS, &error);

   /* Έλεγχος για επιτυχή μεταφορά */
   dma_queue_state = SCIDMAQueueState(dma_queue);

   /* Καθαρισμός */
   SCIRemoveDMAQueue(dma_queue, NO_FLAGS, &error);
   SCIUnmapSegment(local_map, NO_FLAGS, &error);
   SCIRemoveSegment(local_segment, NO_FLAGS, &error);
   SCIDisconnectSegment(remote_segment, NO_FLAGS, &error);
   SCIClose(v_dev, NO_FLAGS, &error);
   SCITerminate();
   return
}
```

Γ.2.2 Κώδικας Παραλήπτη

```
/* Κώδικας του παραλήπτη */
#include "sisci_api.h"
#include <stdio.h>
#define RECEIVER_MEM_ID 4
#define RECEIVER_MEM_SIZE 4096
#define ADAPTER_NO 0
#define NO_CALLBACK 0
#define NO_ARG 0
#define NO_FLAGS 0
#define PRINT_COMMAND 1

int main(int argc, char* argv[])
{
   sci_desc_t v_dev;
   sci_error_t error;
   sci_local_segment_t local_segment;
   sci_map_t local_map;
   int* local_address;

   /* Αρχικοποίηση του περιβάλλοντος SCI */
   SCIInitialize(NO_FLAGS, &error);
```

```
   /* Δημιουργία εικονικής συσκευής */
   SCIOpen(&v_dev, NO_FLAGS, &error);

   /* Δέσμευση τοπικού τμήματος μνήμης */
   SCICreateSegment(v_dev, &local_segment,
                    RECEIVER_MEM_ID, RECEIVER_MEM_SIZE,
                    NO_CALLBACK, NO_ARG, NO_FLAGS, &error);

   /* Απεικόνιση του τοπικού τμήματος μνήμης στο χώρο εικονικών
      διευθύνσεων της διεργασίας */
   local_address = (int*)
                   SCIMapLocalSegment(local_segment, &local_map,
                                      0 /* offset */,
                                      RECEIVER_MEM_SIZE,
                                      0 /* address hint */,
                                      NO_FLAGS, &error);

  /* Αρχικοποίηση των περιεχομένων της πρώτης λέξης του μοιραζόμενου τμήματος */
   *local_address = ~PRINT_COMMAND;

   /* Εξαγωγή τοπικού τμήματος μνήμης */
   SCIPrepareSegment(local_segment, ADAPTER_NO, NO_FLAGS, &error);
  SCISetSegmentAvailable(local_segment, ADAPTER_NO, NO_FLAGS, &error);

   /* Ενεργός αναμονή μέχρι να σταλεί η εντολή PRINT */
   while (*l_addr != PRINT_COMMAND) ;
   printf("Hello, World!");

   /* Καθαρισμός */
   SCISetSegmentUnavailable(segment, ADAPTER_NO, NO_FLAGS, &error);
   SCIRemoveSegment(local_segment, NO_FLAGS, error);
   SCIClose(v_dev, NO_FLAGS, &error);
   SCITerminate();
   return
}
```

Παράρτημα Δ

Λειτουργικό Σύστημα Linux

Στην παρούσα διατριβή χρησιμοποιήθηκαν δύο διαφορετικές συστοιχίες υπολογιστών με δίκτυα FastEthernet και SCI, αντίστοιχα. Κάθε κόμβος των συστοιχιών εκτελούσε Λ.Σ. Linux. Παρακάτω παραθέτουμε λίγα πράγματα για την ιστορία του Linux, το οποίο αποτέλεσε ένα σημαντικό εργαλείο για την πραγματοποίηση των πειραματικών μετρήσεων.

Δ.1 Ιστορία του Linux

Η ιστορία του Linux είναι στενα συνδεδεμένη με αυτή του GNU project, ένα project ελεύθερου λογισμικού. Το GNU project ξεκίνησε το 1983 για την ανάπτυξη ενός ολοκληρωμένου συστήματος UNIX που θα αποτελείτο από εντελώς ελεύθερο λογισμικό. Μέχρι το 1991 όπου γράφτηκε η πρώτη έκδοση του πυρήνα του Linux, το GNU project είχε δημιουργήσει σχεδόν όλα τα συστατικά αυτού του συστήματος, περιλαμβάνοντας ένα φλοιό, τη βιβλιοθήκη της γλώασσας C και έναν μεταγλωττιστή της C. Όμως, δεν υπήρχε ακόμα πυρήνας για το λειτουργικό σύστημα, γιατί η ανάπτυξη του πυρήνα που είχε επιλεγεί (Hurd) αποδείχτηκε πολύ δύσκολη.

Ο πυρήνας του Linux γράφτηκε αρχικά από ένα Φινλανδό φοιτητή, τον Linus Torvalds, που φοιτούσε στο Πανεπιστήμιο του Helsinki. Αρχικά ήταν ένας ελεύθερος πυρήνας που επιδεχόταν αλλαγές και έμοιαζε με το Minix. Το Minix ήταν ένα μικρό Λ.Σ. του Andrew Tannenbaum, αρκετά απλό ώστε να χρησιμεύει για τις ανάγκες διδασκαλίας, χωρίς καμία βλέψη για παραγωγική λειτουργία. Στη συνέχεια, χιλιάδες εθελοντές προγραμματιστές από ολόκληρο τον κόσμο συνείσφεραν στον πυρήνα του Linux. Ο Torvalds μαζί με τους υπόλοιπους αρχικούς προγραμματιστές υιοθέτησαν τα εργαλεία GNU για το λειτουργικό περιβάλλον του συστήματος, δημιουργώντας ένα πλήρες και αξιόπιστο λειτουργικό σύστημα που συμμορφώνεται στο πρότυπο POSIX.

Το Linux, σαν μια μοντέρνα έκδοση ενός Λ.Σ. Unix, έχει όλα τα χαρακτηριστικά ενός

μοντέρνου λειτουργικού συστήματος. Οι χρήστες του, αυτή τη στιγμή, υπολογίζονται σε 18 εκατομμύρια, παγκοσμίως. Τη στιγμή που γράφεται το παρόν κείμενο, η σταθερή έκδοση του πυρήνα του Linux είναι η 2.4.XX, ενώ η έκδοση που αναπτύσσεται είναι η 2.6.0-preX.

Δ.2 Χαρακτηριστικά του Linux

Ένα από τα βασικά χαρακτηριστικά του Linux είναι ότι ο πηγαίος του κώδικας είναι ελεύθερος. Οποιοσδήποτε επιθυμεί να γνωρίσει τον τρόπο λειτουργίας ενός παραγωγικού λειτουργικού συστήματος, δεν έχει παρά να κατεβάσει τον πηγαίο κώδικα από το Διαδίκτυο και να αρχίσει να τον μελετάει. Η μελέτη αυτή προϋποθέτει τη γνώση των βασικών στοιχείων της θεωρίας λειτουργικών συστημάτων.

Επίσης, το Linux έχει πολύ μεγάλη υποστήριξη από μια μεγάλη κοινότητα χρηστών του σε όλον τον κόσμο, κυρίως μέσω του Διαδικτύου. Κατά τη διάρκεια των πειραματικών μετρήσεων της παρούσας διατριβής χρειάστηκε να γίνει εμβάθυνση σε διάφορα ζητήματα που αφορούσαν τον τρόπο λειτουργίας του Linux και γενικότερα των λειτουργικών συστημάτων. Μέσω της ανάγνωσης του πηγαίου κώδικα και της βοήθειας της κοινότητας του Linux, πήραμε πολύτιμες πληροφορίες οι οποίες οδήγησαν στη ζητούμενη απάντηση.

Περισσότερες πληροφορίες για το Linux μπορούν να βρεθούν στην ιστοσελίδα `http://www.linux.org`.

Παράρτημα E

Στοιχειοθεσία Κειμένου

Το παρόν κείμενο δημιουργήθηκε με το πρόγραμμα LaTeX, στη διανομή teTeX για Linux. Τα πακέτα που χρησιμοποιήθηκαν είναι τα εξής:

- abstract με την παράμετρο addtotoc
- amsfonts
- caption2 με παραμέτρους small και centerlast
- color
- epigraph
- fancyhdr
- fncychap με την παράμετρο Lenny
- geometry
- graphicx
- layout
- listings
- makeidx
- setspace
- tabularx
- titlesec
- url

Για τη συγγραφή του κειμένου LaTeX χρησιμοποιήθηκε το περιβάλλον kile στο Linux.

Η γραμματοσειρά που χρησιμοποιήθηκε για το κείμενο είναι η Warnock-Pro της εταιρείας Adobe. Μετατράπηκε σε μορφή Postscript Type 1 χρησιμοποιώντας το εργαλείο pfaedit. Ως typewriter χρησιμοποιήθηκε η γραμματοσειρά Courier της εταιρείας Magenta.

Η πρώτη έκδοση των σχημάτων του κειμένου, δημιουργήθηκε με το πρόγραμμα xfig. Στη συνέχεια, τα σχήματα ξανασχεδιάστηκαν με το πρόγραμμα Visio Professional 2002 (έκδοση 10.0.525) της εταιρείας Microsoft. Μετατράπηκαν σε postscript μέσω εκτύπωσης στον εικονικό εκτυπωτή Generic PostScript Printer (AdobePSGenericPostScriptPrinter) της Adobe. Στη συνέχεια μετατράπηκαν σε μορφή .eps χρησιμοποιώντας το gsview32 έκδοση 4.5 για Windows. Οι γραφικές παραστάσεις δημιουργήθηκαν με το πρόγραμμα gnuplot.

Το ευρετήριο δημιουργήθηκε με το πρόγραμμα xindy. Ευχαριστώ τον Παναγιώτη Χείλαρη, υποψήφιο διδάκτορα ΕΜΠ, για τη βοήθειά του στην εγκατάσταση και χρήση του συγκεκριμένου προγράμματος.

Βιβλιογραφία

[Aam98] Tore Anders Aamodt. Design and implementation issues for an SCI cluster configuration system. In *Proc. of the SCI-Europe '98*, Bordeaux, France, Sep. 28-30 1998.

[AB03] R. Andonov and S. Balev. Optimal Semi-Oblique Tiling. *IEEE Transaction on Parallel and Distributed Systems*, 14(8):944–960, Sep 2003.

[ABD+98] S. Araki, A. Bilas, C. Dubnicki, J. Edler, K. Konishi, and J. Philbin. User-space Communication: A Quantitative Study. In *Proceedings of the Supercomputing '98*, Orlando, Florida, USA, Nov 1998.

[ACP95] T. Anderson, D. Culler, and D. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[AKPT99] T. Andronikos, N. Koziris, G. Papakonstantinou, and P. Tsanakas. Optimal Scheduling for UET/UET-UCT Generalized N-Dimensional Grid Task Graphs. *Journal of Parallel and Distributed Computing*, 57(2):140–165, May 1999.

[ASTK02a] M. Athanasaki, A. Sotiropoulos, G. Tsoukalas, and N. Koziris. A Pipelined Execution of Tiled Nested Loops on SMPs with Computation and Communication Overlapping. In *Proceedings of the Workshop on Compile/Runtime Techniques for Parallel Computing, held in conjuction with the 2002 International Conference on Parallel Processing (ICPP 2002)*, Vancouver, Canada, August 2002.

[ASTK02b] M. Athanasaki, A. Sotiropoulos, G. Tsoukalas, and N. Koziris. Pipelined Scheduling of Tiled Nested Loops onto Clusters of SMPs with Computation and Communication Overlapping. In *Proceedings of the ACM/IEEE Supercomputing 2002:*

High Performance Networking and Computing Conference, Baltimore, Maryland, November 2002.

[BAC+98] M. A. Blumrich, R. D. Albert, Y. Chen, D. W. Clark, S. N. Damianakis, C. Dubnicki, E. W. Felten, L. Iftode, K. Li, M. Martonosi, and R. A. Shillner. Design Choices in the SHRIMP System: An Empirical Study. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA'98)*, 1998.

[BAHPar] M. Banikazemi, B. Abali, L. Herger, and D. K. Panda. Design Alternatives for VIA and an Implementation on IBM Netfinity NT Cluster. *Special Issue of Journal of Parallel and Distributed Computing on Cluster and Network-Based Computing*, to appear.

[BAP00] M. Banikazemi, B. Abali, and D. K. Panda. Comparison and Evaluation of Design Choices for Implementing the Virtual Interface Architecture (VIA). In *Proceedings of the Fourth Int'l Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing (CANPC '00), held in conjunction with HPCA-6*, Toulouse, France, Jan 2000.

[BCF+95] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, Feb 1995.

[BDFL96] M. A. Blumrich, C. Dubnicki, E. W. Felten, and K. Li. Protected, User-level DMA for the SHRIMP Network Interface. In *Proceedings of the Second International Symposium on High-Performance Computer Architecture (HPCA'96)*, February 1996.

[BDRR94] P. Boulet, A. Darte, T. Risset, and Y. Robert. (Pen)-ultimate tiling? *INTEGRATION, The VLSI Jounal*, 17:33–51, 1994.

[BFY96] M. Baker, G. Fox, and H. Yau. A Review of Commercial and Research Cluster Management Software. NHSE Review, May 1996. `http://www.nhse.org/NHSEreview/CMS`.

[BJM+96] Gregory D. Buzzard, David Jacobson, Milon Mackey, Scott Marovich, and John Wilkes. An Implementation of the Hamlyn Sender-Managed Interface Architecture. In *Proceedings of the Operating Systems Design and Implementation (OSDI)*, pages 245–259, October 1996.

[BLA+94] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proceedings of the 21th Annual International Symposium on Computer Architecture (ISCA'94)*, pages 142–153, April 1994.

[Blu96] M. Blumrich. *Network Interface for Protected, User-Level Communication*. PhD thesis, Princeton University, Apr 1996.

[BRB98] R. A. F. Bhoedjang, T. Ruehl, and H. E. Bal. User-Level Network Interface Protocols. *IEEE Computer*, 31(11):53–60, Nov 1998.

[BS96] J. C. Brustoloni and P. Steenkiste. Effects of Buffering Semantics on I/O Performance. In *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 1996.

[Buo99] Philip Buonadonna. An Implementation and Analysis of the Virtual Interface Architecture. Master's thesis, Department of Electrical Engineering and Computer Science, Computer Science Division, University of California, Berkeley, May 1999.

[Buy99] Rajkumar Buyya. *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall PTR, June 1999.

[CIM97] Compaq, Intel, and Microsoft. Virtual Interface Architecture Specification Ver. 1.0, Dec 1997.

[CKR+00] S. H. Chung, H. C. Kwon, K.R. Ryu, H. K. Jang, J. H. Kim, and C. A. Choi. Parallel Information Retrieval on an SCI-Based PC-NOW. In *Proc. of Personal Computer based Networks Of Workstations (PC-NOW 2000)*, Cancun, Mexico, May 5 2000.

[Cla96] Roy Clark. SCI Interconnect Chipset and Adapter: Building Large Scale Enterprise Servers with Pentium Pro SHV Nodes, 1996.

[CMC98] B. N. Chun, A. M. Mainwaring, and D. E. Culler. Virtual Network Transport Protocols for Myrinet. *IEEE Micro*, 18(1):53–63, Jan/Feb 1998.

[CTHI98] F. O' Carroll, H. Tezuka, A. Hori, and Y. Ishikawa. The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network. In *Proceedings of the International Conference on Supercomputing*, pages 243–249, Melbourne, Australia, 1998.

[DBC+98] C. Dubnick, A. Bilas, Y. Chen, S. Damianakis, and K. Li. Project Update: Myrinet Communication. *IEEE Micro*, 18(1):50–52, January/February 1998.

[DBLP97] C. Dubnicki, A. Bilas, K. Li, and J. Philbin. Design and Implementation of Virtual Memory-Mapped Communication on Myrinet. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.

[DIFL96] C. Dubnicki, L. Iftode, E. W. Felten, and K. Li. Software Support for Virtual Memory-Mapped Communication. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS'96)*, pages 372–381, April 1996.

[Don03] Jack J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Technical Report CS-89-85, Computer Science Department, University of Tennessee, Dec 2003. `http://www.netlib/org/benchmark/performance.ps`.

[DP93] Peter Druschel and Larry L. Peterson. Fbufs: A High-Bandwidth Cross-Domain Transfer Facility. In *Proceefings of the 11th Symposium on Operating Systems Principles*, pages 189–202, December 1993.

[DPD94] Peter Druschel, Larry L. Peterson, and Bruce S. Davie. Experiences with a High-Speed Network Adaptor: A Software Perspective. In *Proceedings of SIGCOMM'94*, pages 2–13, September 1994.

[DRM+98] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, and C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, 18(2):66–76, Mar/Apr 1998.

[DS99] R. Dimitrov and A. Skjellum. Efficient MPI for Virtual Interface (VI) Architecture. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 6, pages 3094–3100, Las Vegas, Nevada, USA, June 1999.

[DWB+93] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J.Lumley. Afterburner. *IEEE Network*, 7(4):36–43, July 1993.

[EBBV95] T. Von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 40–53, Copper Mountain, Colorado, December 1995.

[EHHS97] M. Eberl, H. Hellwagner, B. Herland, and M. Schulz. SISCI - Implementing a Standard Software Infrastructure on an SCI Cluster. In *1st Workshop Cluster Computing*, TU-Chemnitz, Nov 1997.

[EHK+98] M. Eberl, H. Hellwagner, W. Karl, M. Leberecht, and J. Weidendorfer. Fast Communication Libraries on an SCI Cluster. In *Proc. of SCI Europe '98*, Bordeaux, France, Sep 1998.

[EHSH97] M. Eberl, H. Hellwagner, M. Shulz, and B. G. Herland. SISCI - Implementing a Standard Software Infrastructure on an SCI Cluster. In *Proc. of Workshop Cluster Computing*, TU-Chemnitz, Germany, Nov 1997.

[Eic93] Thorsten Von Eicken. *Active Messages: an Efficient Communication Architecture for Multiprocessors*. PhD thesis, Computer Science Division, University of California, Berkeley, California, USA, 1993.

[EV98] T. Von Eicken and W. Vogels. Evolution of the Virtual Interface Architecture. *IEEE Computer*, 31(11):61–68, Nov 1998.

[GAB+] F. Giacomini, T. Amundsen, A. Bogaerts, R. Hauser, B. Johnsen, H. Kohmann, R. Nordstrom, and P. Werner. Low Level SCI software functional specification-Software Infrastructure for SCI. ESPRIT Project 23174. `http://www.dolphinics.com/downloads/nt/pdf_zip/SISCI_API-2_1_1.p%df`.

[GCP96] R. Gillett, M. Collins, and D. Pimm. Overview of Network Memory Channel for PCI. In *Proceedings of the IEEE Spring COMPCON '96*, February 1996.

[GL94] D. Gustavson and Q. Li. Local-Area MultiProcessor: the Scalable Coherent Interface, 1994.

[GOB01] K. Ghouas, K. Omang, and H. Bugge. VIA over SCI - Consequences of a Zero Copy Implementation, and Comparison with VIA over Myrinet. In *Proc. of Communication Architectures for Clusters (CAC 2001) in conjunction with Int'l Parallel and Distributed Processing Symposium (IPDPS 2001)*, San Fransisco, California, Apr 2001.

[GSK01] G. Goumas, A. Sotiropoulos, and N. Koziris. Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping. In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium*. IEEE Press, April 2001. (best paper award).

[Gus92] D. B. Gustavson. The Scalable Coherent Interface and Related Standards Projects. *IEEE Micro*, 12(1):10–22, Jan/Feb 1992.

[Han01] Jorgen S. Hansen. I/O Buffer Management for Shared Storage Devices in SCI-based Clusters of Workstations. In *Proc. of the SCI-Europe 2001*, Dublin, Ireland, Oct 2001.

[HEH98] B. G. Herland, M. Eberl, and H. Hellwagner. A common messaging layer for MPI and PVM over SCI. In *HPCN Europe*, pages 576–587, 1998.

[Hel99] H. Hellwagner. The SCI Standard and Applications of SCI. In Hermann Hellwagner and Alexander Reinefeld, editors, *SCI: Scalable Coherent Interface, Architecture and Software for High-Performance Computer Clusters*, volume 1734 of *Lecture Notes in Computer Science*, chapter SCI and Competitive Interconnects for Cluster Computing, pages 3–37. Springer, 1999.

[HJ92] Dana S. Henry and Christopher F. Joerg. A Tightly-Coupled Processor-Network Interface. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, pages 111–121, 1992.

[HM93] M. Homewood and M. McLaren. Meiko CS-2 Interconnect Elan – Elite design. In *Proceedings of the IEEE Hot Interconnects '93 Symposium*, August 1993.

[Hod99] Edin Hodzic. *Time Optimal Tiling of Algorithms with Uniform Dependencies for Distributed Memory Parallel Computers*. Ph.d. thesis, Santa Clara University, School of Engineering, June 1999.

[Hol92] E. H. Hollander. Partitioning and Labeling Loops by Unimodular Transformations. *IEEE Trans. on Parallel and Distributed Systems*, 3(4):465–476, July 1992.

[HR99] H. Hellwagner and A. Reinefeld, editors. *SCI: Scalable Coherent Interface, Architecture and Software for High-Performance Compute Clusters*, volume 1734 of *Lecture Notes in Computer Science*. Springer, 1999.

[HS98] E. Hodzic and W. Shang. On Supernode Transformation with Minimized Total Running Time. *IEEE Trans. on Parallel and Distributed Systems*, 9(5):417–428, May 1998.

[HT94] R. A. Hexsel and N. P. Topham. The Performance of SCI Memory Hierarchies. In *Proc. of the Int'l Workshop on Support for Large Scale Shared Memory Architectures*, pages 1–17, Cancun, Mexico, Apr 1994.

[Ins92] Institute of Electrical and Electronic Engineering. IEEE-1596 Standard for Scalable Coherent Interface (SCI), 1992.

[Int98] Intel. Write Combining Memory Implementation Guidelines. Online PDF document, Nov 1998. http://www.intel.com/design/pentiumII/applnots/24442201.pdf.

[ISSA98] M. Ibel, M. Schmitt, K. E. Schauser, and A. Acharya. An Efficient Global Address Space Model with SCI. In *Proc. of SCI Europe '98*, Bordeaux, France, Sep 1998.

[ISSW97] M. Ibel, K. E. Schauser, C. J. Scheiman, and M. Weis. High Performace Cluster Computing Using SCI. In *Proc. of Hot Interconnects V*, Palo Alto, CA, Aug 1997.

[IT88] F. Irigoin and R. Triolet. Supernode Partitioning. In *Proc. 15th Ann. ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages*, pages 319–329, San Diego, California, Jan 1988.

[JLGS90] D. V. James, A. T. Laundrie, S. Gjessing, and G. S. Sohi. Distributed-Directory Scheme: Scalable Coherent Interface. *IEEE Computer*, 23(6):74–77, June 1990.

[JZ93] David B. Johnson and Willy Zwaenepoel. The Peregrine High-performance RPC System. *Software: Practice and Experience*, 23(2):201–221, February 1993.

[KC94] V. Karamcheti and A. Chien. Software Overhead in Messaging Layers: Where Does the Time Go? In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 51–60, Oct 1994.

[KKJ01] Jin-Soo Kim, Kangho Kim, and Sung-In Jung. Building a high-performance communication layer over virtual interface architecture on Linux clusters. In *Proceedings of the 15th International Conference on Supercomputing*, pages 335–347, Sorrento, Italy, 2001. ACM Press.

[KKJ02] Kangho Kim, Jin-Soo Kim, and Sung-In Jung. GNBD/VIA: A Network Block Device over Virtual Interface Architecture on Linux. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Fort Lauderdale, Florida, USA, Apr 2002.

[KLS86] Nancy P. Kronenberg, Henry M. Levy, and William D. Strecker. VAXClusters: A Closely-Coupled Distributed System. *ACM Transactions on Computer Systems*, 4(2):130–146, May 1986.

[KOH+94] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy.

The Stanford FLASH Multiprocessor. In *Proceedings of the 21st International Symposium on Computer Architecture*, April 1994.

[KSG03] N. Koziris, A. Sotiropoulos, and G. Goumas. A Pipelined Schedule to Minimize Completion Time for Loop Tiling with Computation and Communication Overlapping. *Journal of Parallel and Distributed Computing*, 63(11):1138–1151, November 2003.

[LLD+83] P. J. Leach, P. H. Levine, B. P. Douros, J. A. Hamilton, D. L. Nelson, and B. J. Stumpf. The Architecture of an Integrated Local Network. *IEEE Journal on Selected Areas in Communications*, SAC-1(5):842–857, 1983.

[LPC98] M. Lauria, S. Pakin, and A. A. Chien. Efficient layering for high speed communication: Fast messages 2.x. In *Proceeding of the seventh IEEE International Symposium on High Performance Distributed Computing*, Chicago, Illinois, USA, July 1998.

[LS88] R. Lipton and J. Sandberg. Pram: A scalable shared memory. Technical Report CS-TR-180-88, Princeton University, September 1988.

[MBB+] H. Muller, A. Bogaerts, J. Buytaert, R. Divia, A. Ivanov, R. Keyser, F. Lozano-Alemany, G. Mugnai, D. Samyn, and B. Skaali. First experience with the scalable coherent interface.

[MC95] A. Mainwaring and D. Culler. Active messages: Organization and applications programming interface. Technical Report, 1995.

[MF] MPI-Forum. `http://www.mpi-forum.org/docs/docs.html`.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Journal of Electronics*, 38(8):114–117, April 1965.

[Nag01] Shailabh Nagar. *Communication and Scheduling in Clusters: A User-Level Perspective*. PhD thesis, Department of Computer Science and Engineering, Graduate School, Pennsylvania State University, August 2001.

[Oma95] Knut Omang. Performance results from SALMON, a cluster of Workstations Connected by SCI. Research report 208, Department of Informatics, University of Oslo, Nov 1995.

[OP97] K. Omang and B. Parady. Performance of Low-Cost UltraSparc Multiprocessors connected by SCI. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation*, pages 109–115, Phoenix, Arizona, Jan 1997.

[Pfi98] Gregory F. Pfister. *In Search of Clusters*. Prentice Hall PTR, Jan 1998.

[PH94] D. Patterson and J. Hennessy. *Computer Organization & Design. The Hardware/-Software Interface*, pages 364–367. Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[PKC97] S. Pakin, V. Karamcheti, and A. Chien. Fast messages: Efficient, portable communication for workstation clusters and mpps. *IEEE Concurrency*, 5(2):60–73, Apr-June 1997.

[PLC95] S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois fast messages (fm) for myrinet. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, San Diego, California, United States, 1995. ACM Press.

[PR94] Paul Pierce and Greg Regnier. The Paragon Implementation of the NX Message Passing Interface. In *Proceedings of Scaleble High-Performance Computing Conference (SHPCC'94)*, 1994.

[PT98] L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance networking on Myrinet. In *Proceedings of the Workshop on Personal Computer based Networks Of Workstations '98, held in conjuction with the IPPS/SPDP '98*, Orlando, Florida, USA, Mar/Apr 1998.

[PVM] PVM Home Page. `http://www.csm.ornl.gov/pvm/pvm_home.html`.

[RLW94] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the 21th Annual International Symposium on Computer Architecture (ISCA'94)*, pages 325–337, April 1994.

[RS92] J. Ramanujam and P. Sadayappan. Tiling Multidimensional Iteration Spaces for Multicomputers. *Journal of Parallel and Distributed Computing*, 16:108–120, 1992.

[SASB99] E. Speight, H. Abdel-Shafi, and J. K. Bennett. Realizing the Performance Potential of the Virtual Interface Architecture. In *Proceedings of the 13th International Conference on Supercomputing*, pages 184–192, Rhodes, Greece, June 1999. ACM Press.

[SBB+91] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, and Thomas L. Rodeheffer. Autonet: A High-speed, Self-Configuring Local Area Network Using Point-to-Point Links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–1335, October 1991.

[Sch97] M. Schulz. Low-level SCI-API requirements for Pthreads. Working paper for the SISCI project, June 1997.

[Sch99] M. Schulz. SISCI Pthreads Implementation report. ESPRIT Project 23174 - SISCI, Deliverable 4.1.4, Technische Universitat Munchen, Nov 1999.

[Sco96] Steven L. Scott. Synchronization and Communication in the T3E Multiprocessor. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 26–36, October 1996.

[Sei99a] F. Seifert. Design and Implementation of System Software for Transparent Mode Communication over SCI, Feb 1999. supervisor: Prof. W. Rehm.

[Sei99b] F. Seifert. Development of system software to integrate the Virtual Interface Architecture (VIA) into the Linux operating system kernel for optimized message passing. Diploma thesis, University of Technology Chemnitz, Sep 1999.

[SF91] W. Shang and J. A. B. Fortes. Time Optimal Linear Schedules for Algorithms with Uniform Dependencies. *IEEE Trans. on Computers*, 40(6):723–742, June 1991.

[SF92] W. Shang and J. A. B. Fortes. Independent Partitioning of Algorithms with Uniform Dependencies. *IEEE Trans. on Computers*, 41(2):190–206, Feb. 1992.

[SH98] M. Schulz and H. Hellwagner. Global Virtual Memory based on SCI-DSM. In *Proc of SCI-Europe '98*, Bordeaux, France, Sep 1998.

[SK01] A. Sotiropoulos and N. Koziris. A Pipelined Schedule for Loop Tiling to Minimize Overall Completion Time. In *Proceedings of the 8th Panhellenic Conference on Informatics*, Nicosia, Cyprus, November 2001.

[Sol99] Dolphin Interconnect Solutions. *PCI-SCI Adapter Card D320/321 Functional Overview Ver. 1.01*, Nov. 30 1999. Part no.: D1950-10299.

[Spe82] A. Z. Spector. Performing Remote Operations Efficiently on a Local Computer Network. *Communications of the ACM*, 25(4):246–260, April 1982.

[SRH99] J. Simon, A. Reinefeld, and O. Heinz. Large-Scale SCI Clusters in Practice: Architecture and Performance. In Hermann Hellwagner and Alexander Reinefeld, editors, *SCI: Scalable Coherent Interface, Architecture and Software for High-Performance Compute Clusters*, volume 1734 of *Lecture Notes in Computer Science*, chapter Benchmark Results and Application Experiences, pages 367–379. Springer, 1999. ISBN 3-540-66696-6.

[ST93] J. M. Smith and C. B. S. Traw. Giving Applications Access to Gb/s Networking. *IEEE Network*, 7(4):44–52, July 1993.

[STK01] A. Sotiropoulos, G. Tsoukalas, and N. Koziris. A Pipelined Execution of Tiled Nested Loops onto a Cluster of PCs using PCI-SCI NICs. In *Proceedings of the 2001 SCI-Europe Conference*, October 2001.

[STK02a] A. Sotiropoulos, G. Tsoukalas, and N. Koziris. Efficient Utilization of Memory Mapped NICs onto Clusters using Pipelined Schedules. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin, Germany, May 2002.

[STK02b] A. Sotiropoulos, G. Tsoukalas, and N. Koziris. Enhancing the Performance of Tiled Loop Execution onto Clusters using Memory Mapped Iinterfaces and Pipelined Schedules. In *Proceedings of the Workshop on Communication Architecture for Clusters (CAC), held in Conjunction with Int'l Parallel and Distributed Processing Symposium (IPDPS2002)*, Fort Lauderdale, Florida, April 2002.

[SWR01] Friedrich Seifert, Joachim Worringen, and Wolfgang Rehm. Using Arbitrary Memory Regions for SCI Communication. In *Proc. of SCI Europe 2001 Conference*, Dublin, Ireland, Oct 2001.

[TL93] C. A. Thekkath and H. M. Levy. Limits to Low-Latency Communication on High-Speed Networks. *ACM Transactions on Computer Systems*, 11(2):179–203, May 1993.

[TOP] TOP500 Supercomputer Sites. `http://www.top500.org`.

[TR99] M. Trams and W. Rehm. A new generic and reconfigurable PCI-SCI bridge. In *Proceedings of the SCI-Europe '99, held as a Conference Stream of Euro-PAR '99*, Toulouse, France, Sep 1999.

[TR01] M. Trams and W. Rehm. SCI Transaction Management in out FPGA-based PCI-SCI Bridge. In *Proc. of SCI Europe 2001*, Trinity College, Dublin, Ireland, Oct. 1-3 2001.

[Tra98] M. Trams. Diplomarbeit: Design of a System Friendly PCI-SCI Bridge with an optimized User-Interface, Sep 1998. TU Chemnitz, Fakultat fur Informatik.

[TRBS00] M. Trams, W. Rehm, D. Balkanski, and S. Simeonov. Memory Management in a Combined VIA/SCI Hardware. In *Proceedings of Intl. Workshop on Personal Computer based Networks of Workstations (PC-NOW 2000), help with IPDPS 2000*, pages 4–15, Cancun, Mexico, May 2000.

[TRS99] M. Trams, W. Rehm, and F. Seifert. An Advanced PCI-SCI bridge with VIA support. In *Proc. of 2nd Cluster-Computing Workshop*, University of Karlsruhe, Germany, Mar. 25-26 1999.

[TSR00] M. Trams, R. Schlosser, and W. Rehm. Design Choices and first Results of our VIA-capable PCI-SCI Bridge. In *Proc. of IEEE International Conference on Cluster Computing (CLUSTER2000)*, pages 349–350, Chemnitz, Germany, Nov 28 - Dec 1 2000. ISBN 0-7695-0896-0.

[Tvi92] Ivan Tving. Multiprocessor interconnection using SCI. Master's thesis, Technical University of Denmark, 1992.

[VIA] The Virtual Interface Specification Version 1.0. `http://www.viarch.org`.

[WB99] J. Worringen and T. Bemmerl. MPICH for SCI-connected Clusters. In *Proc. of SAVE Autumn Meeting*, Aachen, Germany, Oct 7-8 1999.

[WBE96] M. Welsh, A. Basu, and T. Von Eicken. Low-Latency Communication over Fast Ethernet. In *Proceedings of the Euro-Par '96*, Lyon, France, Aug 1996.

[WBE97] Matt Welsh, Anindya Basu, and Thorsten Von Eicken. Incorporating Memory Management into User-Level Network Interfaces. Technical Report TR97-1620, Computer Science Department, Cornell University, 13, 1997.

[WGRB02] J. Worringen, A. Gaer, F. Reker, and T. Bemmerl. Exploiting Transparent Remote Memory Access for Non-Contiguous- and One-Sided-Communication. In *Proc. of International Parallel and Distributed Processing Symposium 2002*, Fort Lauderdale, Florida, Apr. 15-19 2002.

[Wol92] Michael Edward Wolf. *Improving Locality and Parallelism in Nested Loops*. Ph.d. thesis, Stanford University, Palo Alto, California, August 1992.

[WSB01] J. Worringen, F. Seifert, and T. Bemmerl. Efficient Asynchronous Message Passing via SCI with Zero-Copying. In *Proc. of SCI-Europe '01*, Trinity College Dublin, Dublin, Ireland, Oct 1-3 2001.

[Xue97a] J. Xue. Communication-Minimal Tiling of Uniform Dependence Loops. *Journal of Parallel and Distributed Computing*, 42(1):42–59, 1997.

[Xue97b] J. Xue. On Tiling as a Loop Transformation. *Parallel Processing Letters*, 7(4):409–424, 1997.

[ZHS99] I. Zoraja, H. Hellwagner, and V. Sunderam. SCIPVM: Parallel Distributed Computing on SCI Workstation Clusters. *Concurrency: Practice and Experience*, 11(13), Mar 1999.

Ευρετήριο