**Τεχνολογικό Εκπαιδευτικό Ίδρυμα Δυτικής Ελλάδας**

# ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

**του φοιτητή**

**Νικόλαου Αντωνάτου του Παναγιώτη (Α.Μ.13605)**

| **Ακαδημαϊκό έτος: 2014** | **Εξάμηνο: ΠΤΥΧΙΟ** |
|---|---|
| **ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΟΛΛΑΠΛΩΝ ΑΡΧΙΤΕΚΤΟΝΙΚΩΝ** | |
| **Επιβλέπων Καθηγητής:**<br>**Καθ. Δρόσος Λάμπρος** | e-mail: antonatos.n@gmail.com<br>τηλέφωνα: 6970815146 - 2105980200 |

# ΠΕΡΙΕΧΟΜΕΝΑ

# ΕΙΣΑΓΩΓΗ

Κύριος σκοπός του παράλληλου προγραμματισμού είναι η αποτελεσματική χρήση υπολογιστών παράλληλης επεξεργασίας, με μερικά διαφορετικά χαρακτηριστικά και ιδιαιτερότητες από το συμβατικό (ακολουθιακό) προγραμματισμό, αφού οι λειτουργίες (εντολές) δεν εκτελούνται με αυστηρώς προκαθορισμένη σειρά.

Σήμερα υπάρχει πληθώρα παράλληλων υπολογιστών στο εμπόριο και στα ερευνητικά κέντρα, ωστόσο δε φαίνεται να ξεχωρίζει κάποια μηχανή ως επικρατέστερη, αφού κάθε μια από αυτές προτιμάται για ορισμένο πεδίο εφαρμογών. Αν και γενικά υπάρχουν ορισμένες τυποποιημένες τεχνικές που μπορούν να χρησιμοποιηθούν, εντούτοις ο παράλληλος προγραμματισμός (όπως και ο ακολουθιακός) θεωρείται ως ένα είδος τέχνης, δεδομένου ότι κάθε προγραμματιστής χρησιμοποιεί το δικό του ιδιαίτερο στυλ όταν δημιουργεί παράλληλο κώδικα. Για να δημιουργηθεί ένας παράλληλος αλγόριθμος δεν είναι απαραίτητο να ξεκινήσει κάποιος από τον ακολουθιακό αλγόριθμο, αφού ο καλύτερος ακολουθιακός αλγόριθμος δεν είναι σίγουρο ότι μπορεί να μετατραπεί στον καλύτερο παράλληλο. Για όσο δυνατόν καλύτερα αποτελέσματα πρέπει να επανεξετάζονται οι προδιαγραφές κάθε προβλήματος.

Ο παραλληλισμός επηρεάζει το σύνολο ενός υπολογιστικού συστήματος. Στον παραλληλισμό υπάρχουν: παράλληλες αρχιτεκτονικές, λειτουργικά συστήματα τα οποία υποστηρίζουν τον παραλληλισμό, παράλληλες γλώσσες, βιβλιοθήκες προγραμματισμού, παράλληλοι αλγόριθμοι αλλά και παράλληλες μεθοδολογίες ανάπτυξης λογισμικού ή εφαρμογών. Οι παράλληλες εκδοχές σε σχέση με τις ακολουθιακές επιτρέπουν ελευθερία σε μεγάλο βαθμό και συνακόλουθα εναλλακτικές προσεγγίσεις. Για να υιοθετηθεί μια προσέγγιση είναι συνδυασμός ορθότητας και τεχνολογικής συγκυρίας.

Ο προγραμματιστής που θέλει να αναπτύξει μια εφαρμογή παράλληλου προγραμματισμού πρέπει να εξοικειωθεί με τις αναπόφευκτες αλλαγές σε όλα τα επίπεδα του προγραμματισμού, οι οποίες προκύπτουν από τη μετάβαση από το ακολουθιακό στο παράλληλο υπολογισμό. Τέτοιες αλλαγές υφίστανται, σε διαφορετικό βαθμό, σε όλα τα επίπεδα αφαίρεσης πάνω από την αρχιτεκτονική των παράλληλων υπολογιστών: Στο λειτουργικό σύστημα και το λογισμικό συστήματος. στις γλώσσες προγραμματισμού και στα συνοδευτικά εργαλεία, στα μοντέλα προγραμματισμού και τη σχεδίαση αλγορίθμων.

# ΓΕΝΙΚΟ ΜΕΡΟΣ

## 1. Εξέλιξη παράλληλης επεξεργασίας

Η έννοια της παραλληλίας, υπό μορφή αξιώματος ενσωματώθηκε από τον Ευκλείδη στην ομώνυμη γεωμετρία του. Τα γεγονότα, όπου συνυπάρχουν αντικείμενα και δραστηριότητες, συμβαίνουν στο χώρο και στο χρόνο. Τα γεγονότα συμβαίνουν στον ίδιο χώρο, κατ' ακολουθία, αλλά και σε διαφορετικούς χώρους στον ίδιο χρόνο, δηλαδή παράλληλα. Οι υπολογιστικές μηχανές με δυνατότητες παράλληλης επεξεργασίας δεδομένων χαρακτηρίζονται ως παράλληλοι υπολογιστές.

Οι πρώτοι παράλληλοι υπολογιστές έκαναν την εμφάνισή τους στις αρχές της δεκαετίας του 1960 και σχεδιάστηκαν από τον Daniel Slotnick στο Πανεπιστήμιο του Illinois: ο Solomon, κατασκευάστηκε από την Westinghouse Electric Company, ενώ ο ILLIAC IV, στις αρχές της δεκαετίας του 1970 συναρμολογήθηκε από την Burroughs Corporation.

Στα μέσα της δεκαετία του 1970, δύο καλά τεκμηριωμένοι παράλληλοι υπολογιστές, ο C.mmp και ο Cm*, κατασκευάστηκαν στο Πανεπιστήμιο Carnegie-Mellon. Στις αρχές της δεκαετίας του 1980, επιστήμονες-ερευνητές από τις εταιρείες Ametek, Intel και nCUBE στο Caltech κατασκεύασαν τον Cosmic Cube, (πρόγονο πολύ-υπολογιστών [multicomputers]. Στα μέσα της δεκαετίας του 1980 εμφανίστηκαν και άλλοι εμπορικοί παράλληλοι υπολογιστές, κατασκευασμένοι με μικροεπεξεργαστές.

Η απόδοση των μικροεπεξεργαστών αυξήθηκε ταχύτερα από την απόδοση άλλων ειδών επεξεργαστών. Μέχρι τα μέσα της δεκαετίας του 1970, οι βασικές αρχιτεκτονικές πρόοδοι, όπως είναι η bit-παράλληλη μνήμη, η bit- παράλληλη αριθμητική, η κρυφή μνήμη (cache memory), οι δίαυλοι, η δια-φυλλωμένη μνήμη (interleaved memory), η σωλήνωση εντολών, οι πολλαπλές λειτουργικές μονάδες, οι σωληνωμένες λειτουργικές μονάδες και η σωλήνωση δεδομένων, είχαν ήδη συμπεριληφθεί στους σχεδιασμούς υπερυπολογιστών. Έκτοτε, αύξηση της απόδοσης επεξεργαστή σήμαινε μείωση χρόνου του κύκλου εντολής. Αυτό έγινε ιδιαίτερα δύσκολο, εφόσον η ταχύτητα των ηλεκτρονικοί κυκλωμάτων περιορίζεται από την ταχύτητα του φωτός.

Η σύγκλιση στη σχετική απόδοση μεταξύ μικροϋπολογιστών και παραδοσιακών υπέρ-υπολογιστών κατέληξε στην ανάπτυξη εμπορικά βιώσιμων παράλληλων υπολογιστών, οι οποίοι αποτελούνται από δεκάδες, εκατοντάδες ή και χιλιάδες μικροεπεξεργαστές και λέγονται μαζικά παράλληλοι. Σε πλήρη αποδοτικότητα, μαζικά παράλληλοι υπολογιστές, όπως ο Paragon XP/S της Intel και ο CM-5 της Thinking Machines, υπερακοντίζουν την ταχύτητα των παραδοσιακών υπερυπολογιστών με έναν επεξεργαστή, όπως του Cray Y/MP και του SX-3 της NEC.

Τον Δεκέμβριο του 1996 πραγματοποιήθηκε η ιστορική διάσπαση του υπολογιστικού φράγματος του 1 tf από τον υπερυπολογιστή Intel ASCI Teraflops. Το υπολογιστικό αυτό σύστημα χρησιμοποιήθηκε ήδη για την εκτίμηση του αποτελέσματος της πρόσκρουσης ενός κομήτη πλάτους ενός χιλιομέτρου στον Ατλαντικό Ωκεανό. Αυτός ο μαζικά παράλληλος υπολογιστής έχει εγκατασταθεί στο New Mexico (Sandia National Laboratory των Η.Π.Α.). Διαθέτει τα εξής χαρακτηριστικά: 9200 επεξεργαστές Pentium Pro 200MHz, 573 gb μνήμη συστήματος και 2.25 tb αποθήκευση δίσκου. Ζυγίζει 44 τόνους, καταναλώνει 850 Kilowatts κατά μέγιστο, απαιτεί 300 τόνους ψυκτικού εξοπλισμού, περιλαμβάνει 86 μικρούς θαλάμους, οι οποίοι καταλαμβάνουν έκταση 1728 τετραγωνικών ποδών και η διασύνδεση μεταξύ αυτών των θαλάμων και των κόμβων του συστήματος επιτυγχάνεται με καλώδια μήκους δύο μιλίων. Συμπληρώνει 40 δισεκατομμύρια υπολογισμούς σε ένα πεντηκοστό του δευτερολέπτου, και μια χρήση του είναι η επιβεβαίωση της ασφάλειας, της αξιοπιστίας και της

αποτελεσματικότητας της πυρηνικής αποθήκης των Η.Π.Α. μέσω αριθμητικής προσομοίωσης αντί πυρηνικών δοκιμών.

Το 1997, η Intel Corporation ανήγγειλε ότι, με τη χρήση του βιομηχανικού προτύπου μεθόδου μέτρησης Unpack, ο υπερυπολογιστής της αυτός επέτυχε απόδοση 1.34 tf.

Τον Οκτώβριο του 1998, η IBM παρέδωσε στην κυβέρνηση των Η.Π.Α. έναν υπερυπολογιστή της, ο οποίος είχε δυνατότητα εκτέλεσης 3.9 tf, δηλαδή 15.000 φορές ταχύτερος από έναν κοινό προσωπικό υπολογιστή. Ο υπολογιστής αυτός ο οποίος ονομάστηκε "Blue Pacific", έγινε με τη συνεργασία της IBM και του Εθνικού Εργαστηρίου Lawrence Livermore του Υπουργείου Ενέργειας των Η.Π.Α. στην Καλιφόρνια. Κύρια χαρακτηριστικά του: 2.6t b μνήμη, 80.000 φορές ταχύτερη μνήμη από έναν κοινό προσωπικό υπολογιστή, με δυνατότητα αποθήκευσης όλων των βιβλίων της Βιβλιοθήκης του Κογκρέσου. Ο Blue Pacific θα πραγματοποιούσε σε ένα δευτερόλεπτο όσους υπολογισμούς θα πραγματοποιούσε ένας άνθρωπος με μια αριθμομηχανή σε 63.000 χρόνια.

Τον Ιούνιο του 2000, ανακοινώθηκε από την IBM ο υπερυπολογιστής της Accelerated Strategic Computing Initiative White, ή ASCI White και είναι από τους ταχύτερους υπερυπολογιστές μέχρι σήμερα στον κόσμο. Είναι μια μαζικά παράλληλη μηχανή που αποτελείται από 512 υπηρέτες (servers) τύπου RS/6000 SP της ίδιας εταιρείας. Χαρακτηριστικά του είναι τα εξής: ζυγίζει 106 τόνους, διαθέτει 8.192 επεξεργαστές, μεγίστη απόδοση 12.3 tf με προοπτική ανάπτυξης 100 tf μέχρι το 2004, εκτείνεται σε 3023.616 m2, κόστος κατασκευής U.S.$ 110*106, οι μισές εφαρμογές του είναι επιστημονικές και οι υπόλοιπες εμπορικές. Βασικό μειονέκτημα είναι το λογισμικό, βασίζεται στο Λ.Σ. Unix. Ωστόσο, είναι ταχύτερος από πυρηνική αντίδραση, η απαιτουμένη ενέργεια λειτουργίας του είναι 1.2 MWatts, ενώ πολλές άλλες πληροφορίες, οι οποίες αφορούν αυτόν τον υπερυπολογιστή, διατηρούνται απόρρητες.

## 1.1. Παράλληλα υπολογιστικά συστήματα

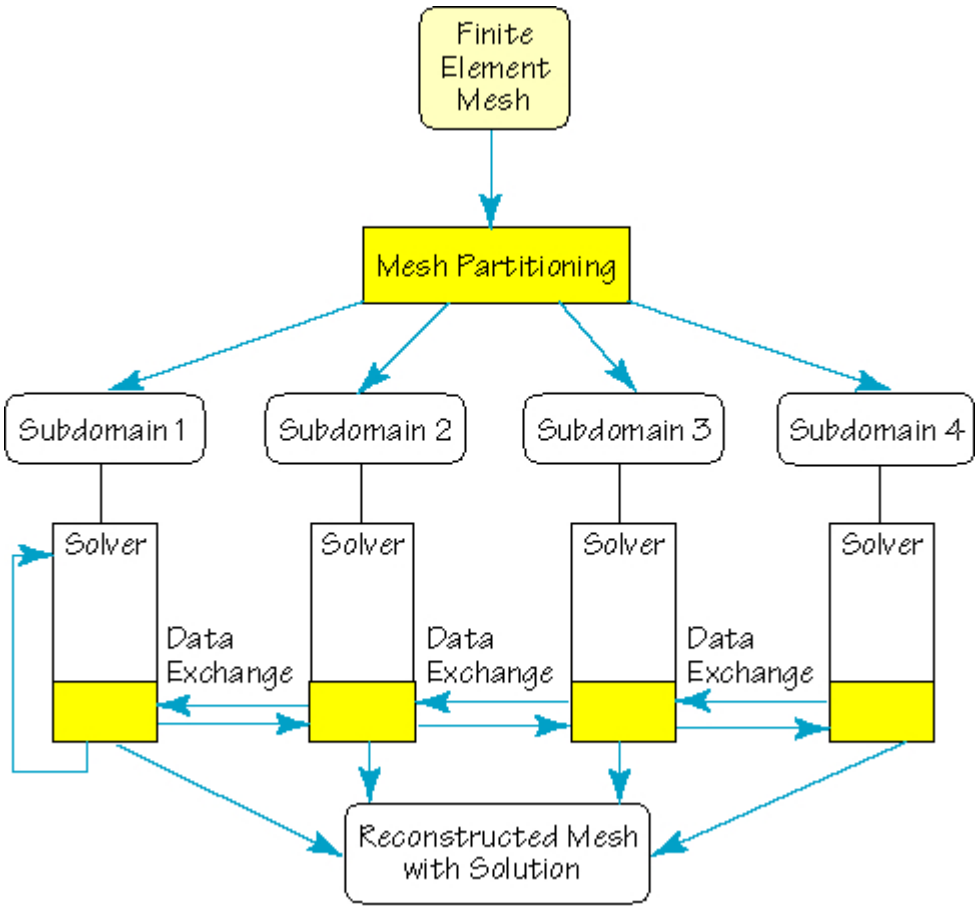
Στην παράλληλη επεξεργασία πραγματοποιείται ταυτόχρονος χειρισμός δεδομένων, τα οποία ανήκουν σε μία ή περισσότερες διεργασίες. Οι διεργασίες λύνουν το ίδιο πρόβλημα και επιτυγχάνεται συνδρομικότητα (concurrency) σε έναν υπολογισμό, καθώς και αύξηση του αριθμού λειτουργιών που εκτελούνται σε κάθε χρονική στιγμή. Τόσο η σωλήνωση όσο και ο παραλληλισμός δεδομένων επιτυγχάνουν την αύξηση της συνδρομικότητας σε έναν υπολογισμό.

## 1.2. Παράλληλος Προγραμματισμός

Τα παράλληλα υπολογιστικά συστήματα με μεγάλο αριθμό επεξεργαστών δημιουργούν νέες απαιτήσεις στο λογισμικό. Σ' ένα πρόγραμμα που βασίζεται σε κοινό σειριακό σύστημα ο επεξεργαστής πρέπει να ακολουθήσει- "διαγράψει" μια σειρά από λειτουργίες. Αντίστοιχα ένα πρόγραμμα βασιζόμενο σε παράλληλο σύστημα κάθε επεξεργαστής πρέπει να ακολουθήσει-"διαγράψει" μια σειρά από λειτουργίες παράλληλα, περιλαμβάνοντας λειτουργίες που συντονίζουν τους χωριστούς επεξεργαστές στην εκτέλεση μιας ενιαίας διεργασίας. Η δημιουργία και ο συντονισμός πολλών παράλληλων διεργασιών προσδίδει μια καινούργια διάσταση στον προγραμματισμό. Αλγόριθμοι για συγκεκριμένα προβλήματα πρέπει να σχεδιάζονται με τρόπο ώστε να παράγεται και να δημιουργείται ένας μεγάλος αριθμός παράλληλων λειτουργιών οι οποίες να εκτελούνται από διαφορετικούς επεξεργαστές.

Η πλήρη κατανόηση των παράλληλων γλωσσών προγραμματισμού και ο σχεδιασμός παράλληλων αλγορίθμων είναι αναγκαίο να πραγματοποιείται, παρόλο που οι αρχιτεκτονικές παράλληλων συστημάτων διαμοιραζόμενης και κατανεμημένης μνήμης εξασφαλίζουν τεράστια αύξησης της υπολογιστικής δύναμης με λογικό κόστος. Για τη δημιουργία προγραμμάτων σε παράλληλα συστήματα, χρήσιμο εννοιολογικό εργαλείο, αποτελεί η κατανόηση της έννοιας της διεργασίας. Διεργασία είναι διαδοχή (σειρά) των λειτουργιών οι οποίες εκτελούνται από έναν απλό επεξεργαστή. Η διεργασία χρησιμοποιείται ως βασική δομή για την ολοκλήρωση παράλληλων προγραμμάτων: κάθε επεξεργαστής εκτελεί μια συγκεκριμένη διεργασία σε οποιαδήποτε στιγμή. Ειδικότερα, μια διεργασία νοείται ως διαδικασία ή υπορουτίνα διαδικασία, η οποία εκτελείται από ένα συγκεκριμένο φυσικό επεξεργαστή. Η διαθεσιμότητα σε έναν υπολογιστή αρκετών επεξεργαστών, σημαίνει ότι πολλές διεργασίες λογισμικού μπορεί να εκτελούνται παράλληλα από το υλικό του υπολογιστή. Η γλώσσα προγραμματισμού πρέπει να διαθέτει μηχανισμούς για το συγχρονισμό των διεργασιών, από τη στιγμή που οι διεργασίες εκτελούνται με μεταβαλλόμενες ταχύτητες σε διαφορετικούς φυσικούς επεξεργαστές.

Τα παράλληλα συστήματα κατανεμημένης και διαμοιραζόμενης μνήμης έγιναν γνωστά στη δεκαετία του 1980 και η έννοια της διεργασίας προστέθηκε σε έναν αριθμό εφαρμοσμένων γλωσσών προγραμματισμού ευρείας χρήσης όπως η Fortran, C, Pascal και Lisp. Παρ' όλα αυτά, εξαιτίας της απουσίας καθιερωμένων παράλληλων γλωσσών προγραμματισμού, ο κάθε κατασκευαστής συστήματος δημιουργούσε τη δική του ποικιλία.



## 1.3. Παράλληλες αρχιτεκτονικές ειδικού σκοπού

Η τάση για συστήματα παράλληλης επεξεργασίας δεν σταμάτησε στη σχεδίαση προγραμματιζόμενων μηχανών γενικού σκοπού, αλλά επεκτάθηκε και στη σχεδίαση αρχιτεκτονικών ειδικού σκοπού, για την εκτέλεση συγκεκριμένων εφαρμογών. Έτσι, παρουσιάσθηκαν μηχανές ειδικά σχεδιασμένες για επεξεργασία σήματος, επεξεργασία εικόνας, βάσεις δεδομένων κ.α. Οι μηχανές αυτές είναι συνήθως βοηθητικές (back-end) κάποιου άλλου κεντρικού φιλοξενούντος υπολογιστή (host), ο οποίος ενεργοποιεί τις μηχανές ειδικού σκοπού, τους παρέχει δεδομένα εισόδου και λαμβάνει αποτελέσματα. Με τις μηχανές ειδικού σκοπού διευκολύνεται στο μέγιστο δυνατό βαθμό η παράλληλη εκτέλεση προκαθορισμένων εφαρμογών.

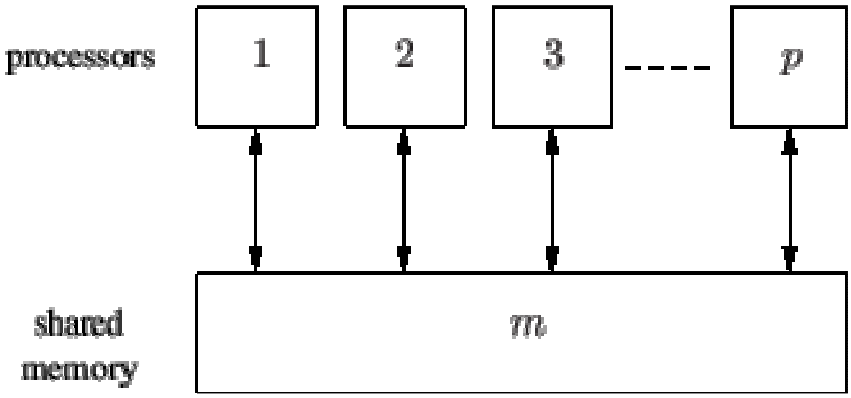
Αρχιτεκτονικές σχεδιασμένες αποκλειστικά για την εκτέλεση ενός αλγόριθμου είναι για παράδειγμα ο διακριτός μετασχηματισμός Fourier ή ο πολλαπλασιασμός δισδιάστατων πινάκων). Κύρια πλεονεκτήματα αυτών των μηχανών είναι ο υψηλός βαθμός παραλληλισμού, η απλότητα δικτύου επικοινωνίας και η απλότητα των χρησιμοποιούμενων μονάδων επεξεργασίας. Κύρια μειονεκτήματα αυτών των μηχανών, είναι ότι στις αρχιτεκτονικές αυτές υλοποιούνται συγκεκριμένοι αλγόριθμοι που συχνά θέτουν όρια στις διαστάσεις των δεδομένων εισόδου. Επίσης, σημαντικό μειονέκτημα των συστημάτων αυτών είναι το υψηλό κόστος σχεδίασής τους, αφού για κάθε αλγόριθμο αναζητείται και η βέλτιστη αρχιτεκτονική τόσο σε σχέση με το βαθμό παραλληλισμού. Ωστόσο, για να επιλυθεί αυτό το πρόβλημα πραγματοποιούνται προσπάθειες για την όσο το δυνατόν μεγαλύτερη αυτοματοποίηση της διαδικασίας σχεδίασης αρχιτεκτονικών ειδικού σκοπού.

Χαρακτηριστική κατηγορία ειδικευμένων αρχιτεκτονικών είναι τα συστολικά συστήματα. Τα συστήματα αυτά αποτελούνται από ένα σύνολο διασυνδεδεμένων κυττάρων (μονάδων επεξεργασίας), όπου κάθε ένα από αυτά εκτελεί πολύ απλές λειτουργίες. Αναλόγως με τη μορφή δικτύου επικοινωνίας μεταξύ των κυττάρων, διακρίνονται τα συστολικά μητρώα (systolic arrays) και τα συστολικά δένδρα (systolic trees).

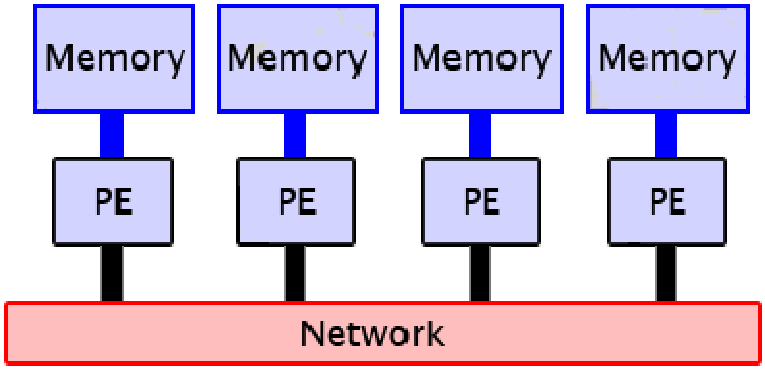
## 1.4. Πρότυπα (Models) παράλληλου υπολογισμού

Τα πρότυπα παράλληλου υπολογισμού χωρίζονται συνήθως στις εξής δύο κατηγορίες:
(1) τα πρότυπα πολυεπεξεργασίας (multiprocessors models)
(2) τα πρότυπα έργου βάθους (work depth models)
Πρότυπα πολυεπεξεργαστών
Υπάρχουν τρία κύρια πρότυπα πολυεπεξεργαστών:

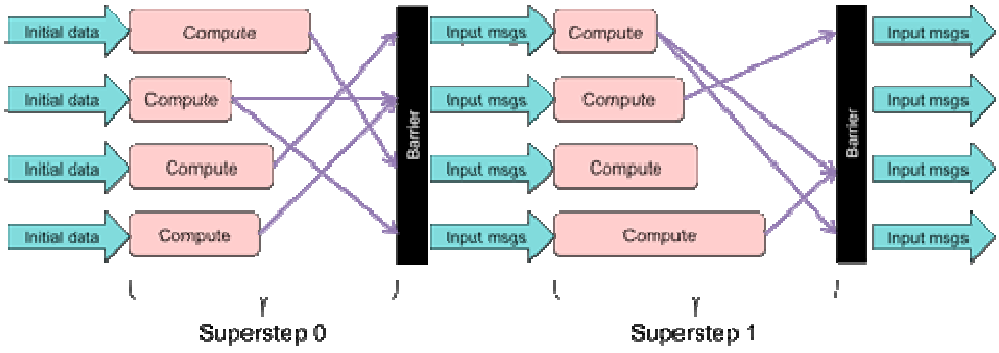
- το PRAM (Parallel Random Access Machine), το οποίο χαρακτηρίζεται ως πρότυπο διαμοιραζόμενης ή κοινής μνήμης (shared memory model)




• το πρότυπο δικτύου (network model), το οποίο χαρακτηρίζεται ως πρότυπο κατανεμημένης μνήμης (distributed memory model)




• το BSP (Bulk Synchronous Parallel Machine), το οποίο χαρακτηρίζεται ως πρότυπο που γεφυρώνει το κενό, το οποίο υπάρχει στον παράλληλο υπολογισμό, μεταξύ υλικού και λογισμικού.




Στην κατηγορία των αρχιτεκτονικών κατανεμημένης μνήμης (distributed memory architectures) περιλαμβάνονται όλοι οι παράλληλοι υπολογιστές που αποτελούνται από ένα σύνολο διασυνδεδεμένων επεξεργαστών, κάθε ένας από τους οποίους διαθέτει προ-σπέλαση σε ανεξάρτητη τοπική μνήμη. Οι επεξεργαστές είναι συνήθως τυποποιημένοι μικροεπεξεργαστές μεσαίας κλίμακας και χαμηλού κόστους.

Τα συστήματα αυτά, αν και παρουσιάζουν ορισμένες δυσκολίες στην αξιοποίησή τους (προγραμματισμός περάσματος μηνυμάτων και συγχρονισμός), διαθέτουν το πλεονέκτημα της επεκτασιμότητας, για την ενσωμάτωση μεγάλου πλήθους επεξεργαστών.

## 1.5. Πρότυπα έργου-βάθους

Στα πρότυπα έργου-βάθους δίνεται έμφαση στους παράλληλους αλγορίθμους και όχι στις παράλληλες μηχανές. Τα πρότυπα αυτά είναι περισσότερο "αφηρημένα" (abstact) σε σχέση με τα πρότυπα πολυεπεξεργαστών και δεν ενσωματώνουν χαρακτηριστικά των αρχιτεκτονικών των παράλληλων μηχανών. Σε ένα πρότυπο έργου-βάθους, το κόστος του αλγορίθμου καθορίζεται από τις αλληλεξαρτήσεις που υπάρχουν μεταξύ των λειτουργιών αυτών, καθώς και από τον αριθμό των λειτουργιών, τις οποίες ο αλγόριθμος εκτελεί.

### 1.5.1. Συμβατικοί παράλληλοι επεξεργαστές

Κύριο χαρακτηριστικό των συμβατικών παράλληλων υπολογιστών είναι ότι αποτελούνται από ένα σύνολο μονάδων επεξεργασίας. Κάθε μία από τις μονάδες επεξεργασίας λειτουργεί με βάση το κλασικό μοντέλο του Von Neumann (μοντέλο αποθηκευμένου προγράμματος). Στο μοντέλο Von Neumann, οι εντολές εκτελούνται ακολουθιακά, με τη βοήθεια μετρητή προγράμματος, όπου κάθε μονάδα επεξεργασίας εκτελεί μία ακολουθία εντολών πάνω σε μία ακολουθία δεδομένων. Σχετικά με το τρόπο παροχής εντολών και δεδομένων στις μονάδες επεξεργασίας, διακρίνονται τέσσερις κατηγορίες μηχανών, από τις οποίες η μία είναι ακολουθιακή και οι υπόλοιπες έχουν δυνατότητες παράλληλης επεξεργασίας. Αυτές οι κατηγορίες αυτές που προτάθηκαν από τον Flynn είναι:

α. Οι Μηχανές Μοναδικής Εντολής, Μοναδικών Δεδομένων: ΜΕΜΔ (Single Instruction Single Data, SISD). Σ' αυτές τις μηχανές μια μονάδα επεξεργασίας εκτελεί ακολουθιακά τις εντολές (μία προς μία) πάνω σε μία σειρά δεδομένων. (Κλασικό μοντέλο Von Neumann).

β. Οι Μηχανές Μοναδικής Εντολής, Πολλαπλών Δεδομένων: ΜΕΠΔ (Single Instruction Multiple Data, SIMD). Εδώ οι μονάδες επεξεργασίας εκτελούν (συγχρονισμένα) μια κοινή ακολουθία εντολών πάνω σε διαφορετικά δεδομένα. Οι μηχανές αυτής της κατηγορίας ονομάζονται επίσης επεξεργαστές μητρώου (array processors).

γ. Οι Μηχανές Πολλαπλών Εντολών, Μοναδικών Δεδομένων: ΠΕΜΔ (Multiple Instruction Single Data, MISD). Εδώ οι μονάδες επεξεργασίας διατάσσονται σε μια αλυσιδωτή μορφή, όπου εκτελούν ανεξάρτητες λειτουργίες (εντολές), με τα δεδομένα μιας μονάδας επεξεργασίας να είναι το αποτέλεσμα της προηγούμενης. Οι μηχανές αυτού του τύπου ονομάζονται και μηχανές αγωγού (pipeline machines).

δ. Οι Μηχανές Πολλαπλών Εντολών, Πολλαπλών Δεδομένων: ΠΕΠΔ (Multiple Instruction Multiple Data, MIMD). Εδώ, οι μονάδες επεξεργασίας εκτελούν ελεύθερα (ανεξάρτητα) οποιαδήποτε εντολή με οποιαδήποτε δεδομένα. Ονομάζονται και μηχανές πολυεπεξεργασίας (multiprocessor machines).

## 1.6. Μηχανές πολυεπεξεργασίας (multi¬processor machines)

Στις μηχανές πολυεπεξεργασίας κύριο χαρακτηριστικό είναι ότι οι χρησιμοποιούμενες μονάδες επεξεργασίας (ΜΕ) είναι αυτόνομοι επεξεργαστές που εκτελούν ανεξάρτητα προγράμματα, χρησιμοποιώντας ανεξάρτητα δεδομένα. Οι μηχανές πολυεπεξεργασίας, ονομάζονται και ως μηχανές πολλαπλών Εντολών - Πολλαπλών Δεδομένων, ΠΕΠΔ (Multiple Instruction Multiple Data, MIMD), εξαιτίας ότι εκτελούν οποιαδήποτε εντολή με οποιαδήποτε δεδομένα.

Τα δεδομένα και οι εντολές των μονάδων επεξεργασίας μπορεί να λαμβάνονται από την κεντρική μνήμη της μηχανής, που συνήθως αποτελείται από αρκετές ανεξάρτητες μονάδες μνήμης (ΜΜ).

Σ' ένα σύστημα πολυεπεξεργασίας, οι μονάδες επεξεργασίας είναι ισοδύναμες (συχνά είναι όμοιοι επεξεργαστές), λειτουργούν παράλληλα, με σκοπό τη διεκπεραίωση ενός γενικότερου υπολογιστικού προβλήματος. Κάθε σύστημα πολυεπεξεργασίας ελέγχεται κατά κανόνα από ένα λειτουργικό σύστημα, μέρος του οποίου εκτελείται σε κάθε μονάδα επεξεργασίας. Συνήθως σ' ένα σύστημα πολυεπεξεργασίας όλες οι μονάδες επεξεργασίας βρίσκονται σε σχετικά μικρές αποστάσεις, ώστε να ανταλλάσσονται ταχέα τα δεδομένα μεταξύ μονάδων επεξεργασίας. Παρ' όλα αυτά, σε κάποιες ειδικές εφαρμογές οι μονάδες επεξεργασίας είναι τοποθετημένες σε απομακρυσμένα σημεία, επικοινωνώντας μεταξύ τους σειριακά (μέσω γραμμών επικοινωνίας). Τα συστήματα αυτά αναφέρονται συχνά και ως κατανεμημένα συστήματα (distributed systems).

## 1.7. Αρχιτεκτονική υπολογιστικού συστήματος

Η χρήση πολλών επεξεργαστών, οι οποίοι συνυπάρχουν στο ίδιο υπολογιστικό σύστημα, εισάγει σ' αυτό επιπλέον χαρακτηριστικά και πολυπλοκότητα συγκριτικά προς ένα σύστημα μ' έναν επεξεργαστή (ακολουθιακός υπολογιστής). Αυτό είναι αναμενόμενο εφόσον, προκειμένου οι επεξεργαστές να είναι σε θέση να συνεργασθούν για την επίλυση του ίδιου υπολογιστικού' προβλήματος, θα πρέπει να μπορούν να επικοινωνούν μεταξύ τους και να έχουν πρόσβαση ή, με άλλα λόγια, να μοιράζονται τα ίδια δεδομένα.

Από πλευράς αρχιτεκτονικής του παράλληλου υπολογιστικού συστήματος, υπάρχουν δύο προσεγγίσεις ικανοποίησης της προηγούμενης απαίτησης:
(1) Η χρήση κοινής ή διαμοιραζόμενης μνήμης (shared memory).
(2) Η επικοινωνία των επεξεργαστών μέσω περάσματος μηνύματος (message passing).

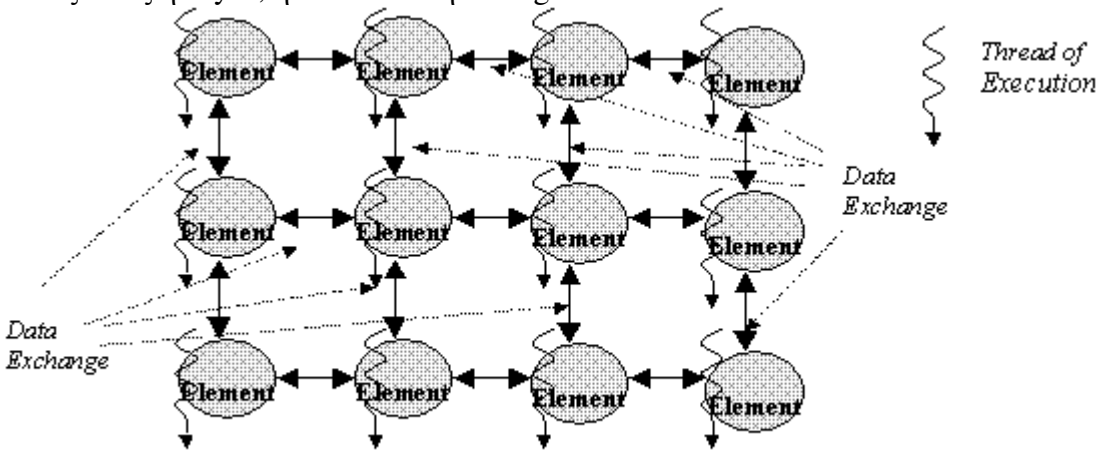
## 1.8. Κοινή μνήμη

Σε ένα κοινό μοντέλο μνήμης, οι παράλληλες εργασίες μοιράζονται ένα σφαιρικό διάστημα διευθύνσεων στο οποίο διαβάζουν και γράφουν ασύγχρονα. Αυτό απαιτεί μηχανισμούς προστασίας όπως κλειδώματα, σηματοφορείς και οθόνες για έλεγχο ταυτόχρονης πρόσβασης. Η κοινή μνήμη μπορεί να εξομοιωθεί και σε συστήματα κατανεμημένης μνήμης.

Οι παράλληλοι υπολογιστές, των οποίων οι αρχιτεκτονικές χαρακτηρίζονται από τη χρήση κοινής μνήμης, καλούνται συνήθως πολυεπεξεργαστές (multiprocessors). Οι πολυεπεξεργαστές αποτελούνται από έναν αριθμό πλήρως προγραμματιζόμενων (fully programmable) επεξεργαστών κάθε ένας από τους οποίους μπορεί να εκτελεί το δικό του πρόγραμμα. Όλοι οι επεξεργαστές έχουν πρόσβαση στην ίδια κοινή μνήμη του που επιτρέπει την κοινή χρήση διαφόρων τιμών. Η κοινή μνήμη μπορεί να είναι ενιαία και η πρόσβαση σ' αυτήν να γίνεται μέσω ενός κοινού διαδρόμου (common bus) ή να είναι χωρισμένη σε πολλές διαφορετικές ενότητες ή μονάδες μνήμης (memory modules).

## 1.9. Μεταβίβαση μηνυμάτων

Σε ένα μήνυμα που περνά το μοντέλο, οι παράλληλες εργασίες ανταλλάσσουν δεδομένα από τη μία στην άλλη. Αυτές οι επικοινωνίες μπορούν να είτε ασύγχρονες είτε σύγχρονες. Η επεξεργασία διαδοχικών διαδικασιών επικοινωνίας (Communicating Sequential Processes-CSP) μεταφοράς μηνυμάτων των χρησιμοποιημένων καναλιών επικοινωνίας εφαρμόζονται για «να συνδέσουν» τις διαδικασίες και οδηγούν σε διάφορες σημαντικές γλώσσες όπως η Joyce, η occam και η Erlang.




Κατά τη μεταβίβαση μηνυμάτων, οι επεξεργαστές έχουν πρόσβαση στις μονάδες κοινής μνήμης μέσω ενός δικτύου διασύνδεσης επεξεργαστή-μνήμης. Οι παράλληλοι υπολογιστές των οποίων οι αρχιτεκτονικές χαρακτηρίζονται από την επικοινωνία των επεξεργαστών μέσω περάσματος μηνυμάτων, συνήθως καλούνται πολυυπολογιστές (multicomputers). Οι πολυυπολογιστές αποτελούνται από έναν αριθμό πλήρως προγραμματιζόμενων επεξεργαστών, κάθε ένας από τους οποίους μπορεί να εκτελεί το δικό του πρόγραμμα και έχει τη δική του τοπική μνήμη (local memory). Οι επεξεργαστές μοιράζονται δεδομένα ανταλλάσσοντας μηνύματα μέσω ενός δικτύου επικοινωνίας επεξεργαστή (processor communication network), το οποίο συνήθως καλείται δίκτυο διασύνδεσης (interconnection network).

## 2. Μοντέλα παράλληλου προγραμματισμού

Ένα μοντέλο παράλληλου προγραμματισμού είναι μια έννοια κατά την οποία επιτρέπονται παράλληλα προγράμματα να μπορούν να μεταγλωττιστούν και να εκτελεσθούν. Η αξία ενός μοντέλου προγραμματισμού κρίνεται συνήθως στη γενικότητά του • πόσο καλά μπορεί να εκφραστεί μια σειρά διαφορετικών προβλημάτων και πόσο καλά θα εκτελεστεί σε μια σειρά από διαφορετικές αρχιτεκτονικές. Η εφαρμογή ενός μοντέλου προγραμματισμού μπορεί να λάβει διάφορες μορφές όπως οι βιβλιοθήκες που καλούνται από τις παραδοσιακές γλώσσες, επεκτάσεις γλωσσών, ή πλήρη εκτέλεση νέων μοντέλων.

Η συναίνεση για ένα συγκεκριμένο μοντέλο προγραμματισμού είναι σημαντική, καθώς επιτρέπεται στο λογισμικό να "μεταφέρεται" μεταξύ διαφορετικών αρχιτεκτονικών. Το μοντέλο Neumann von έχει διευκολύνει αυτή τη διαδικασία με τις διαδοχικές αρχιτεκτονικές, δεδομένου ότι παρέχει μια αποδοτική γέφυρα μεταξύ υλικού (hardware) και λογισμικού (software), που σημαίνει ότι οι υψηλού επιπέδου γλώσσες μπορούν να μεταγλωττιστούν αποτελεσματικά και αυτό μπορεί να εφαρμοστεί αποτελεσματικά στο υλικό.

## 2.1. Κύριες ταξινομήσεις και παραδείγματα

Οι ταξινομήσεις των παράλληλων μοντέλων προγραμματισμού μπορούν να χωριστούν γενικά σε δύο τομείς: α) στην αλληλεπίδραση και β) στη διαδικασία αποσύνθεσης προβλήματος.

## 2.2. Αλληλεπίδραση διαδικασίας

Η αλληλεπίδραση διαδικασίας αφορά τους μηχανισμούς από τους οποίους οι παράλληλες διαδικασίες επικοινωνούν μεταξύ τους. Οι πιο κοινές μορφές αλληλεπίδρασης είναι η προσπέλαση κοινόχρηστων μεταβλητών στη κοινή μνήμη και η ανταλλαγή μηνυμάτων.

Σήμερα υπάρχει μεγάλη ποικιλία παράλληλων υπολογιστών, καθώς και λειτουργικών συστημάτων. Διαφορετικές παράλληλες μηχανές παρέχουν διαφορετικές ευκολίες, υποστηρίζοντας διαφορετικούς τρόπους λειτουργίας. Επίσης διάφορα παράλληλα λειτουργικά συστήματα προσφέρουν διαφορετικά εργαλεία προγραμματισμού στο χρήστη (άλλους τρόπους καθορισμού, κατανομής και επικοινωνίας των διεργασιών). Αποτέλεσμα αυτών των ασυμβατοτήτων είναι μη κοινή χρήση ενός καθιερωμένου μοντέλου παράλληλου προγραμματισμού που να χρησιμοποιείται σε όλους τους τύπους παράλληλων υπολογιστών. Στα μοντέλα παράλληλου προγραμματισμού παρέχεται ένα επίπεδο αφαίρεσης μεταξύ παράλληλων αλγορίθμων και παράλληλων αρχιτεκτονικών, με σκοπό η σχεδίαση και η συγγραφή του προγράμματος να έχει όσο το δυνατό πιο ευέλικτο και μεταφερτό κώδικα. Τα τελευταία χρόνια αναπτύχθηκαν αρκετά διαφορετικά μοντέλα παράλληλου προγραμματισμού τα σπουδαιότερα των οποίων είναι τα ακόλουθα:

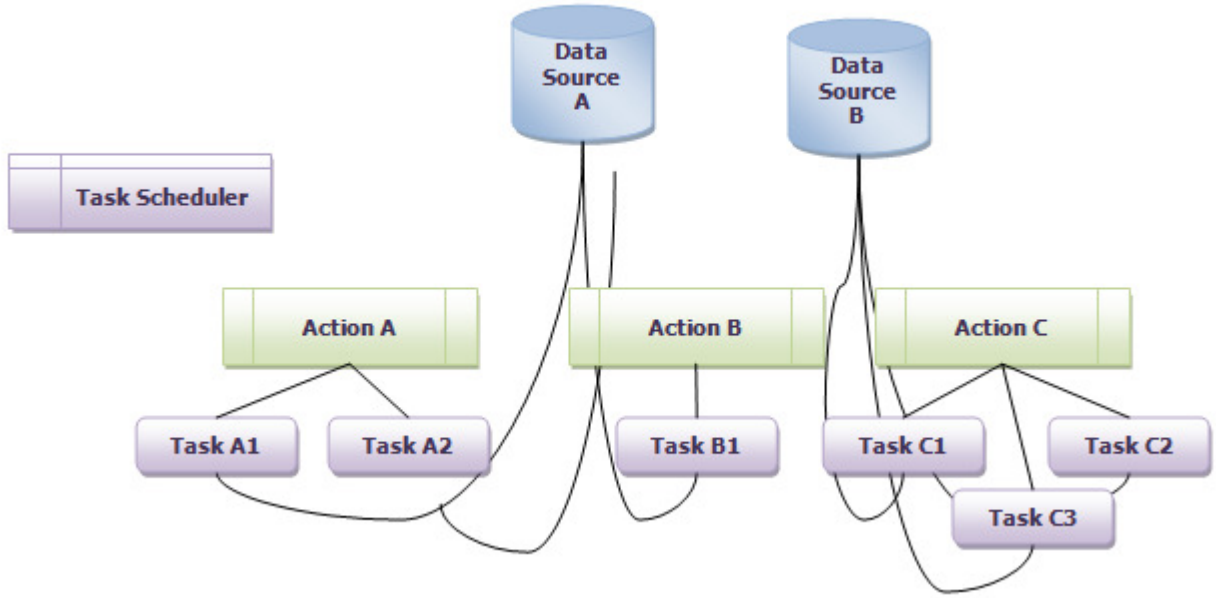
- Μοντέλο Διεργασιών.
- Μοντέλο Νημάτων.
- Μοντέλο Μεταβίβασης Μηνυμάτων.
- Μοντέλο Κοινής Μνήμης.
- Υβριδικό Μοντέλο

Στις επόμενες παραγράφους δίνονται κάποια βασικά στοιχεία αυτών των μοντέλων παράλληλου προγραμματισμού:

## 2.3. Μοντέλο Διεργασιών

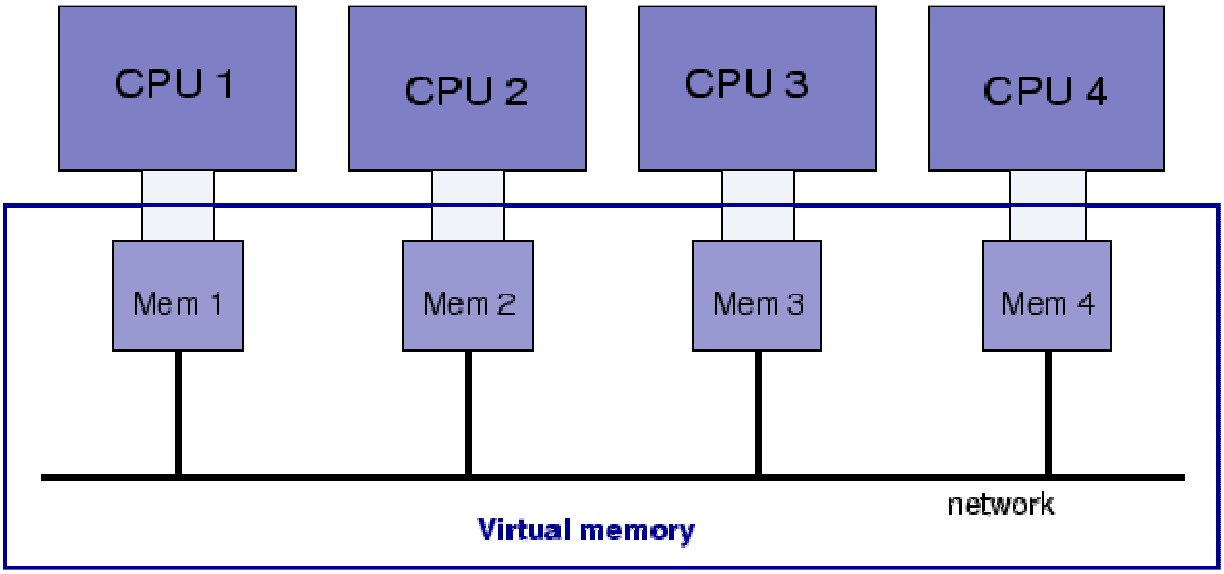
Το μοντέλο διεργασιών αποτελεί το βασικό μοντέλο ανάλυσης του Συντρέχοντος (Concurrent) προγραμματισμού, παραμένοντας κύριο εργαλείο ανάπτυξης-ανάλυσης λειτουργικών συστημάτων. Σ' αυτό το μοντέλο κάθε παράλληλη διεργασία έχει ιδιωτικό χώρο λογικών (εικονικών) διευθύνσεων, στον οποίο απεικονίζονται ένας ενιαίος χώρος φυσικών διευθύνσεων, χρησιμοποιώντας τεχνικές ιδεατής μνήμης. Ο συγχρονισμός και η επικοινωνία των διεργασιών πραγματοποιείται με μηχανισμούς διαδιεργασιακής επικοινωνίας π.χ. locks, pipes, semaphores, message queues, monitors, κοινόχρηστα αρχεία, ιδεατή μνήμη κ.α.

Βασικό πλεονέκτημα του μοντέλου διεργασιών είναι ο σαφής διαχωρισμός κώδικα εργασιών από τους μηχανισμούς συγχρονισμού και επικοινωνίας. Ενώ αντιθέτως κύριο μειονέκτημά του αποτελεί η μεγάλη επιβάρυνση δημιουργίας και διαχείρισης διεργασιών στα περισσότερα λειτουργικά συστήματα.



## 2.4. Μοντέλο Κοινής Μνήμης

Το μοντέλο κοινής μνήμης είναι αρκετά διαδεδομένο εξαιτίας της ομοιότητάς του με το σειριακό προγραμματισμό. Όλες οι διεργασίες, ενός παράλληλου προγράμματος, "βλέπουν" τον ίδιο χώρο διευθύνσεων. Οι διεργασίες επικοινωνούν μεταξύ τους γράφοντας και διαβάζοντας τιμές σε κοινές μεταβλητές (shared variables) που είναι αποθηκευμένες στη κοινή μνήμη, όπως και στο σειριακό προγραμματισμό. Το μοντέλο αυτό χρησιμοποιείται στους παράλληλους υπολογιστές κοινής μνήμης. Τα τελευταία χρόνια η ανάπτυξη συστημάτων κατανεμημένης κοινής μνήμης (Distributed Shared Memory-DSM), έχει επιτρέψει τη χρήση αυτού του προγραμματιστικού μοντέλου στους υπολογιστές κατανεμημένης μνήμης.




## 2.5. Μοντέλο Νημάτων

Οι εφαρμογές που στηρίζονται στο μοντέλο νημάτων, καθώς και οι υλοποιήσεις νημάτων σε επίπεδο επεξεργαστή, έχουν ξεκινήσει εδώ και αρκετά χρόνια. Εξαιτίας όμως ότι οι υλοποιήσεις ήταν αρκετά διαφορετικές μεταξύ τους με δεν υπήρχε προτυποποίηση και μεταφέρσιμος κώδικας.

Στο μοντέλο νημάτων, κάθε διεργασία έχει πολλά συντρέχοντα νήματα, όπως ροές εντολών οι οποίες εκτελούνται έχοντας κοινό χώρο λογικών διευθύνσεων, ιδιωτική στοίβα κλήσεων και κατάσταση επεξεργαστή. Συνήθως τα νήματα ονομάζονται ως "ελαφρές" διεργασίες.

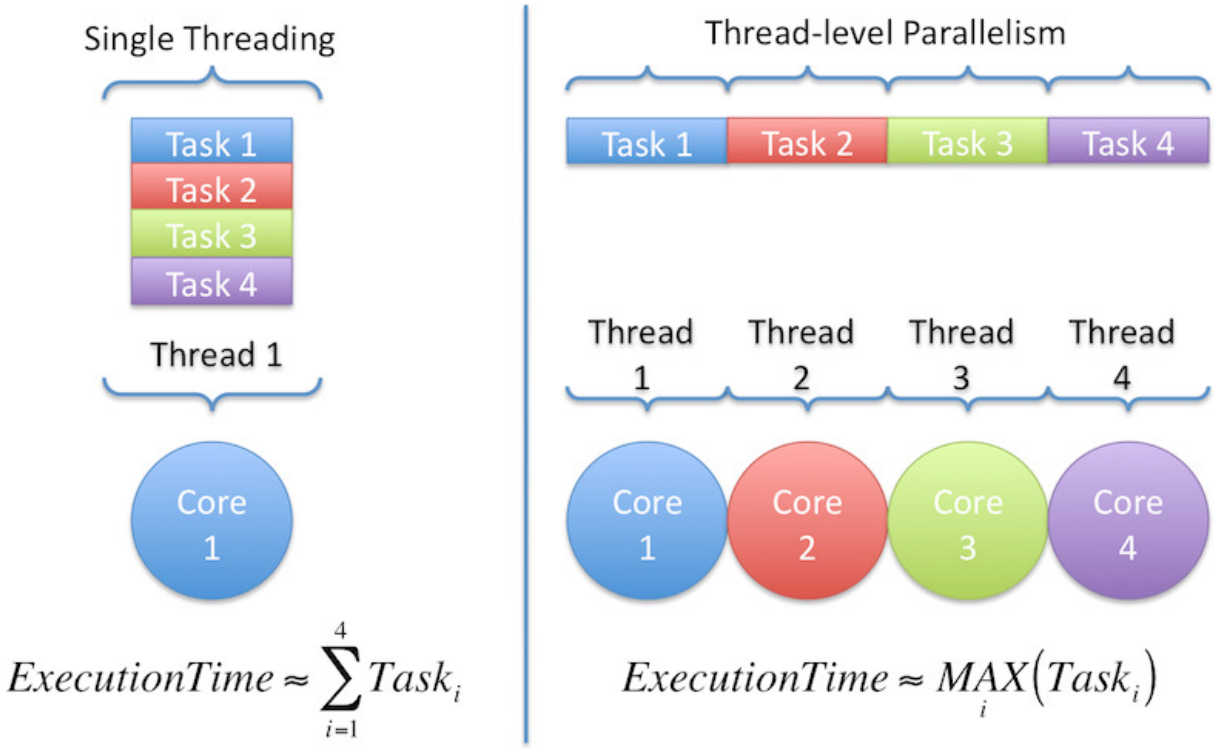
Εναλλακτικός τρόπος περιγραφής των νημάτων αποτελούν οι συντρέχουσες υπορουτίνες (co-routines) οι οποίες εκτελούνται σε έναν ή περισσότερους επεξεργαστές. Το μειονέκτημα του μοντέλου διεργασιών το οποίο αναφέρθηκε ανωτέρω λύνεται με το μοντέλο νημάτων. Ας δούμε πως υλοποιείται

Από άποψη προγραμματισμού, τα νήματα υλοποιούνται ως:

- σύνολο συναρτήσεων βιβλιοθήκης
- σύνολο οδηγιών προς το μεταγλωττιστή.

Ο προγραμματιστής και στις δύο περιπτώσεις είναι υπεύθυνος για τον καθορισμό του παραλληλισμού. Η διαχείριση γίνεται εξ' ολοκλήρου από το λειτουργικό σύστημα (Kernel Level Threads) ή εναλλακτικά το σύστημα που εκτελεί την εφαρμογή μπορεί να συμμετέχει

στη διαχείριση (User Level Threads). Όταν υπάρχει εκτέλεση των User Level Threads, γίνεται τυπική συντρέχουσα επεξεργασία σε επίπεδο εικονικής ή φυσικής μηχανής. Ενώ όταν υπάρχει εκτέλεση των Kernel Level Threads, η εφαρμογή εκτελείται παράλληλα ή συνδρομικά, ανάλογα με τη φυσική μηχανή που διαχειρίζεται ο πυρήνας.




## 2.6. Μοντέλο Μεταβίβασης Μηνυμάτων

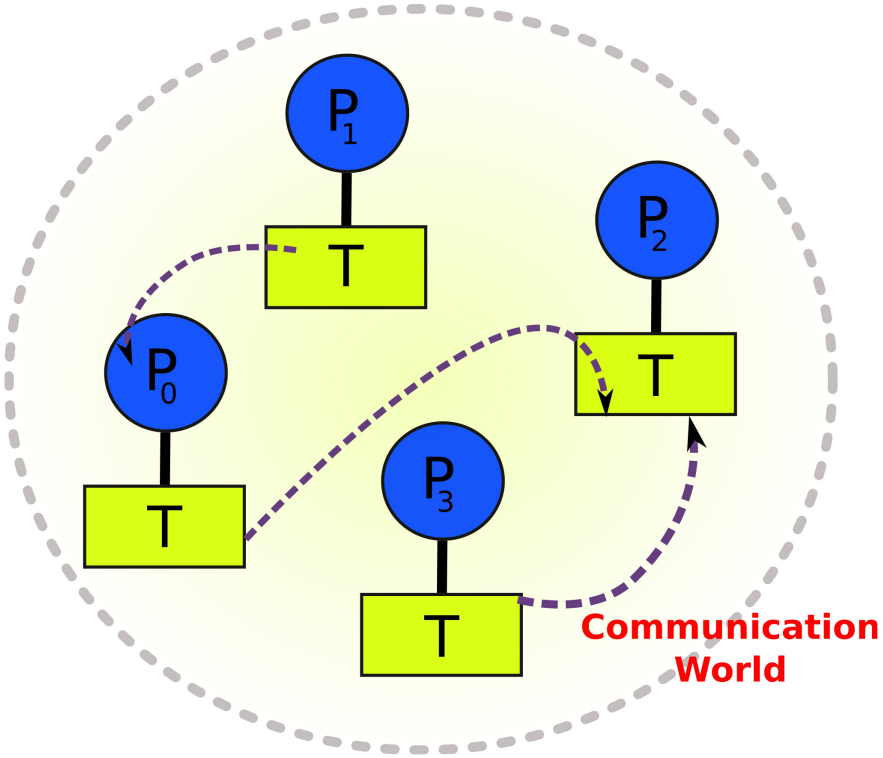
Το μοντέλο αυτό έχει τα εξής χαρακτηριστικά:

- Υπάρχει ένα σύνολο εργασιών, με τη κάθε μια να έχει ιδιωτικό χώρο λογικών διευθύνσεων. Οι χώροι διευθύνσεων αντιστοιχούν σε μνήμες που μοιράζονται κοινή φυσική μνήμη (τεχνικές ιδεατής μνήμης) ή τοπικές φυσικές μνήμες.
- Ο συγχρονισμός και η επικοινωνίας εργασιών γίνεται α) μέσω αποστολής-παραλαβής μηνυμάτων, β) μέσω δικτύου (φυσική σύνδεση), γ) μέσω διαμοιραζόμενης μνήμης (λογική σύνδεση, ουρές μηνυμάτων).
- Για την επικοινωνία και το συγχρονισμός απαιτούνται ζεύγη συνεργαζόμενων λειτουργιών τα οποία υλοποιούνται σε διαφορετικές εργασίες. π.χ. μια λειτουργία αποστολής (send) πρέπει να έχει και μια αντίστοιχη λειτουργία παραλαβής (receive).

Το μοντέλο μεταβίβασης μηνυμάτων από θέμα προγραμματισμού, υλοποιείται ως ένα σύνολο συναρτήσεων βιβλιοθήκης. Ο προγραμματιστής με κλήσεις των συναρτήσεων μέσα στο πρόγραμμα, καθορίζει τον παραλληλισμό.

Η υλοποίηση μεταβίβασης μηνυμάτων στη δεκαετία του 1980 πραγματοποιείται ευρέως, αλλά εξαιτίας ότι διέφερε αρκετά μεταξύ τους δεν επιτεύχθηκε η ανάπτυξη μεταφέρσιμων εφαρμογών. Το 1992, δημιουργήθηκε το MPI Forum, ώστε να δημιουργηθεί μια πρότυπη διεπαφή στις υλοποιήσεις μεταβίβασης μηνυμάτων. Η Διεπαφή Μεταβίβασης Μηνυμάτων (Message Passing Interface, MPI) δημοσιεύτηκε το 1994, ενώ μια βελτιωμένη έκδοση (MPI-2) δημοσιεύτηκε το 1996.

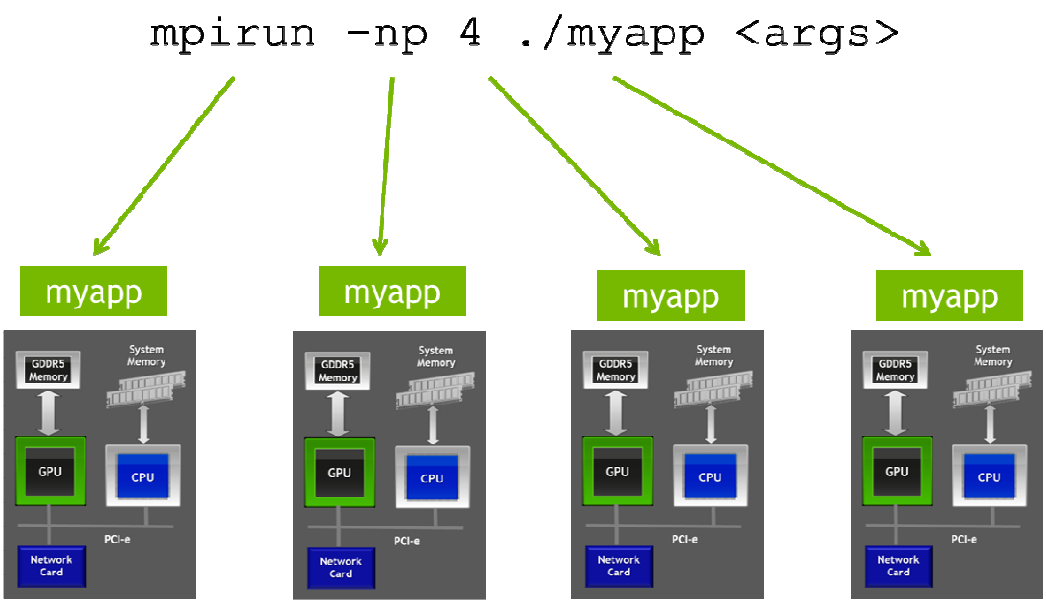
Στο μοντέλο μεταβίβασης μηνυμάτων, το MPI είναι πλέον καθιερωμένο, αντικαθιστώντας όλες τις άλλες υλοποιήσεις. Οι περισσότεροι κατασκευαστές παράλληλων συστημάτων παρέχουν υλοποιήσεις του MPI, αλλά και MPI-2. Γνωστές είναι οι OpenMPI και οι Mpich.




## 2.7. Υβριδικό Μοντέλο

Στο υβριδικό μοντέλο γίνεται συνδυασμός δύο ή περισσότερων μοντέλων. Αντιπροσωπευτικό είναι το παράδειγμα με συνδυασμό της μεταβίβασης μηνυμάτων (MPI) με το μοντέλο νημάτων (POSIX threads ή OpenMP). Το μοντέλο αυτό αποκτά ιδιαίτερη σημασία λόγω της επέκτασης των υβριδικών παράλληλων αρχιτεκτονικών, δηλαδή δικτυωμένων πολυπύρηνων συστημάτων.

Άλλο σχετικό παράδειγμα είναι ο συνδυασμός του OpenMP με CUDA ή MPI με CUDA. Επίσης υπάρχει και η συνύπαρξη και των τριών APIs, π.χ. MPI, OpenMP και CUDA. Γίνεται προσπάθεια να επιτρέπεται η κοινή ανάπτυξη κώδικα για CPUs και GPUs, μέσω του OpenCL ή άλλων APIs.




## 2.8. Μοντέλο Διαστήματος-Πλειάδων (Tuple-Space model)

Το διάστημα-πλειάδων είναι μια οντότητα που χρησιμοποιείται στην γλώσσα Linda για την επικοινωνία δύο παράλληλων δρώμενων. Ο μηχανισμός αυτός αναπτύχθηκε ως μια εναλλακτική λύση στα παραδοσιακά προγραμματιστικά μοντέλα της κοινής μνήμης και της ανταλλαγής μηνυμάτων. Πρόκειται για μια δομή που έχει μεγάλη συγγένεια με τις μεταβλητές κατανεμημένης κοινής μνήμης, αλλά και μια διαφορά, οι διευθύνσεις στο διάστημα πλειάδων είναι συσχετισμένες (associative address space). Το μοντέλο αυτό υποστηρίζεται στην ύπαρξη πλειάδων με διαφορετικό βαθμό συνοχής δεδομένων (different coherency semantics).

## 2.9. Μοντέλο Λογικού Προγραμματισμού (Logic Programming)

Οι γλώσσες λογικού προγραμματισμού είναι ενδογενώς παράλληλες. Αυτό σημαίνει ότι ο διαχωρισμός ενός προγράμματος σε παράλληλα δρώμενα γίνεται από την ίδια την γλώσσα. Στο μοντέλο λογικού προγραμματισμού, ο αλγόριθμος περιγράφεται με μια γλώσσα λογικού προγραμματισμού. Ο τύπος του παραλληλισμού και ο τρόπο με τον οποίο ο μεταφραστής της γλώσσας εξάγει τον παραλληλισμό διαφέρει από γλώσσα σε γλώσσα. Ο παραλληλισμός, OR, AND ή AND/OR, μπορεί να ορίζεται έμμεσα ή άμεσα με ειδική σημασιολογία. Οι πιο πολλές υλοποιήσεις γλωσσών παράλληλου λογικού προγραμματισμού έχουν γίνει σε μηχανές μοιραζόμενης μνήμης. Υπάρχουν πάντως πολλές υλοποιήσεις τέτοιων γλωσσών σε μηχανές κατανεμημένης μνήμης, αλλά και υλοποιήσεις που είναι κατάλληλες και για τους δύο τύπους αρχιτεκτονικών.

Παραδείγματα μοντέλων παράλληλου προγραμματισμού

Μερικά παραδείγματα μοντέλων παράλληλου προγραμματισμού είναι:

- Οι αλγοριθμικοί σκελετοί (Algorithmic Skeletons).
- Τα συστατικά (Components).
- Τα κατανεμημένα αντικείμενα (Distributed Objects).
- Η απομακρυσμένη μέθοδος επίκλησης (Remote Method Invocation).

- Η ροή εργασιών (Workflows).
- Η παράλληλη μηχανή τυχαίας προσπέλασης (Parallel Random Access Machine).
- Η επεξεργασία ρεύματος (Stream processing).
- Ο μαζικός σύγχρονος παραλληλισμός (Bulk synchronous parallelism).

Από: http://en.wikipedia.org/wiki/Parallel_programming_model

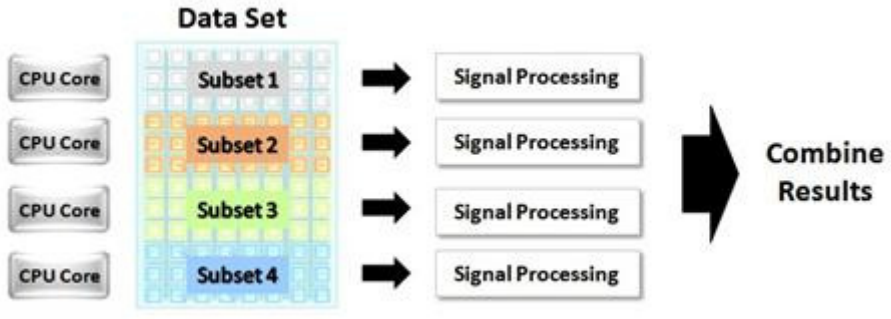
# 3. Παράλληλοι Αλγόριθμοι

Η ανάπτυξη συστημάτων παράλληλης επεξεργασίας προσφέρουν τεράστιες υπολογιστικές δυνατότητες και για την εκμετάλλευση αυτών είναι αναγκαία η επινόηση παράλληλων αλγόριθμων για τη διαχείριση ενός μεγάλου αριθμού επεξεργαστών που να εκτελούνται παράλληλα για την ολοκλήρωση ενός συνολικού υπολογισμού. Σε μερικές περιπτώσεις, απλοί σειριακοί αλγόριθμοι μπορούν εύκολα να προσαρμοστούν σε παράλληλα προβλήματα. Όμως, στις περισσότερες περιπτώσεις, το υπολογιστικό πρόβλημα πρέπει να αναλυθεί ξανά από την αρχή και να αναπτυχθούν νέοι παράλληλοι αλγόριθμοι. Τα τελευταία χρόνια, η σχεδίαση παράλληλων αλγορίθμων καλύπτει ένα ευρύ φάσμα πρακτικών προβλημάτων, π.χ. ταξινόμηση, επεξεργασία γραφημάτων, επίλυση γραμμικών και διαφορικών εξισώσεων, αλλά και προσομοίωση. Επίσης, αναπτύσσονται αποδοτικά παράλληλα προγράμματα σε συστήματα κατανεμημένης και διαμοιραζόμενης μνήμης.

Στη συνέχεια γίνεται παρουσίαση μερικών κατηγοριών παράλληλου προγραμματισμού και τεχνικών με μια σύντομη περιγραφή της κάθε κατηγορίας:

## 3.1. Παραλληλισμός Δεδομένων (Data Parallelism)

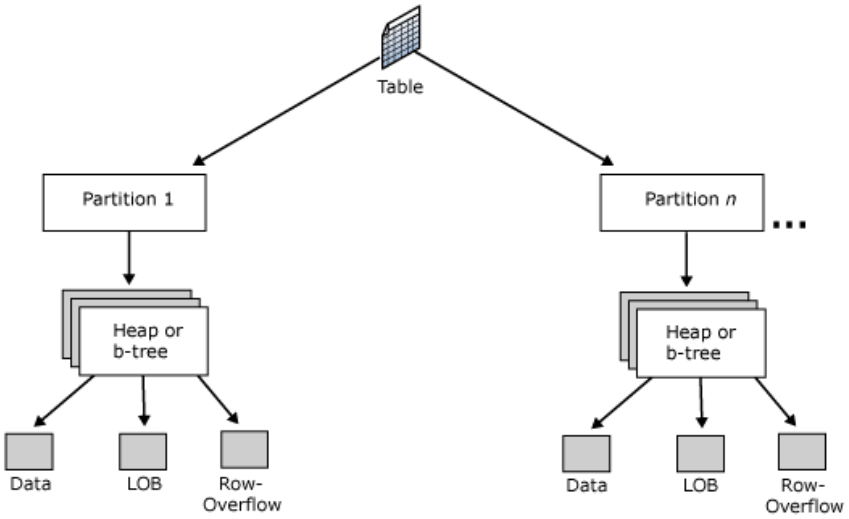
Στον παραλληλισμό δεδομένων ένας μεγάλος όγκος δεδομένων επεξεργάζεται παράλληλα. Χρησιμοποιείται σε αριθμητικούς αλγορίθμους που ασχολούνται με μεγάλους πίνακες και ανύσματα. Ο αλγόριθμος παράλληλης σειριακής ταξινόμησης χρησιμοποιεί παραλληλισμό δεδομένων, όπου κάθε στοιχείο από τη μη κατανεμημένη λίστα επεξεργάζεται με παρόμοιο τρόπο από διαφορετικό επεξεργαστή. Για την επίτευξη καλύτερης απόδοσης, τα περισσότερα παράλληλα προγράμματα χρησιμοποιούν κάποια μορφή παραλληλισμού δεδομένων.




## 3.2. Κατάτμηση Δεδομένων (Data Partitioning)

Αποτελεί ιδιαίτερο τύπο παραλληλισμού δεδομένων, όπου ο χώρος αποθήκευσης δεδομένων διαχωρίζεται φυσικά σε παρακείμενες περιοχές. Κάθε μία από τις παρακείμενες περιοχές υφίστανται παράλληλα επεξεργασία μέσω διαφορετικού επεξεργαστή. Χρησιμοποιείται στα παράλληλα συστήματα κατανεμημένης μνήμης, διότι η υπολογιστική

δραστηριότητα κάθε επεξεργαστή αφορά κυρίως την δική του περιοχή τοπικών δεδομένων. Σπάνια είναι η επικοινωνία αυτόνομων ενιαίων συστημάτων επεξεργαστών.




## 3.3. Ασύγχρονος Αλγόριθμος (Asychronous Algorithm)

Στο σύγχρονο αλγόριθμο, κάθε παράλληλη διεργασία λειτουργεί αυτόνομα, χωρίς να υπάρχει επικοινωνία ή συγχρονισμός μεταξύ των διεργασιών. Αποδίδεται καλύτερα στα συστήματα διαμοιραζόμενης μνήμης και κατανεμημένης μνήμης, έχοντας ως αποτέλεσμα πολύ μεγάλες επιταχύνσεις. Αξιοσημείωτο είναι ότι στον ασύγχρονο αλγόριθμο, παρ' όλο που οι επεξεργαστές προσπελαύνουν μερικά κοινά δεδομένα, κάθε επεξεργαστής υπολογίζει με απόλυτο αυτόνομο τρόπο, τις ενδιάμεσες τιμές δεδομένων, οι οποίες παράγονται από άλλους επεξεργαστές. Η απόλυτη αυτονομία κάθε επεξεργαστή αποτελεί γνώρισμα-κλειδί των ασύγχρονων αλγορίθμων, καθιστώντας εύκολο τον προγραμματισμό τους.

## 3.4. Σύγχρονη Επανάληψη (Synchronous Iteration)

Κατά τη διαδικασία της σύγχρονης επανάληψης, ο κάθε επεξεργαστής εκτελεί ίδια επαναληπτική λειτουργία σε διαφορετικό τμήμα δεδομένων. Οι επεξεργαστές πρέπει να συγχρονίζονται στο τέλος κάθε επανάληψης, εξασφαλίζοντας την αποτροπή του κάθε επεξεργαστή να αρχίσει την επόμενη επανάληψη, πριν ολοκληρώσουν και οι υπόλοιποι επεξεργαστές την τρέχουσα επανάληψη.

Κλασσικοί αριθμητικοί αλγόριθμοι που χρησιμοποιούνται στις φυσικές επιστήμες κατά τη μετατροπή τους σε παράλληλη μορφή καταλήγουν σε σύγχρονη επανάληψη. Η σύγχρονη επανάληψη αποδίδεται ικανοποιητικά στα παράλληλα συστήματα διαμοιραζόμενης μνήμης (αφού ο χρόνος που απαιτείται για το συγχρονισμό επεξεργαστών είναι σχετικά μικρός). Αυτό το μειονέκτημα του συγχρονισμού δε μειώνει αρκετά την απόδοση του προγράμματος. Παρ' όλα αυτά, το τίμημα του συγχρονισμού στα παράλληλα συστήματα κατανεμημένης μνήμης είναι αρκετά αυξημένο, λόγω της κατανεμημένης φύσης των επεξεργαστών. Έτσι η σύγχρονη επανάληψη πρέπει να χρησιμοποιείται με προσοχή στα παράλληλα συστήματα κατανεμημένης μνήμης ώστε να "γραφτούν" προγράμματα με αξιόπιστη απόδοση.

## 3.5. Υπολογιστική Διαδικασία Διασωλήνωσης (Pipeline Computation)

Αυτοί οι αλγόριθμοι εφαρμόζονται κυρίως στα παράλληλα συστήματα κατανεμημένης μνήμης, εξαιτίας του μεθοδικού τρόπου ροής των δεδομένων και της μη αναγκαιότητας για γενική προσπέλαση των διαμοιραζόμενων δεδομένων. Η διαδικασία έχει ως εξής: οι διεργασίες οργανώνονται σε μερικές κανονικές δομές, π.χ. δακτυλίου ή δισδιάστατου πλέγματος. Διαμέσου της κανονικής αυτής δομής διακινούνται τα δεδομένα, κάθε διεργασία εκτελεί συγκεκριμένο τμήμα της συνολικής υπολογιστικής διαδικασίας.

## 3.6. Προβλήματα που περιορίζουν την απόδοση παράλληλων προγραμμάτων

Οι κύριοι λόγοι που μειώνουν την απόδοση παράλληλων προγραμμάτων που εκτελούνται σε πραγματικά συστήματα είναι οι εξής:

1. Ανταγωνισμός μνήμης (Memory Contention). Ο ανταγωνισμός της μνήμης παρουσιάζεται μόνο όταν υπάρχει διαμοιραζόμενη μνήμη. Η εκτέλεση σε ένα επεξεργαστή καθυστερεί όταν αυτός περιμένει να αποκτήσει πρόσβαση σ' μια θέση μνήμης, η οποία χρησιμοποιείται αυτή τη χρονική στιγμή από κάποιον άλλο επεξεργαστή. Το πρόβλημα παρουσιάζεται επίσης όταν διαμοιράζονται δεδομένα σε ένα μεγάλο αριθμό παράλληλων επεξεργαστών.
2. Εκτεταμένος Σειριακός Κώδικας (Extensive Sequential Code). Σε οποιοδήποτε παράλληλο αλγόριθμο, υπάρχουν πάντα τμήματα τα οποία περιέχουν καθαρά σειριακό κώδικα και εκτελούνται συγκεκριμένα είδη κεντρικών λειτουργιών (αρχικοποίηση μεταβλητών). Επακόλουθο του σειριακού κώδικα είναι να στερεί από μερικούς αλγόριθμους αρκετό ποσοστό της μέγιστης συνολικής επιτάχυνση που μπορεί να επιτευχθεί.
3. Καθυστέρηση Επικοινωνίας (Communication Delay). Συμβαίνει στα παράλληλα συστήματα κατανεμημένης μνήμης, αφού οι επεξεργαστές επικοινωνούν με το μηχανισμό περάσματος μηνυμάτων. Σε μερικές περιπτώσεις, η επικοινωνία μεταξύ δύο επεξεργαστών μπορεί να χρειαστεί την προώθηση του μηνύματος σε ενδιάμεσους επεξεργαστές του δικτύου επικοινωνίας. Οι καθυστερήσεις επικοινωνίας που θα επακολουθήσουν, ίσως να μειώσουν αισθητά τη ταχύτητα εκτέλεσης ορισμένων παράλληλων αλγόριθμων.
4. Καθυστέρηση Συγχρονισμού (Synchronization Delay). Κατά το συγχρονισμό παράλληλων διεργασιών, μια διεργασία αναγκάζεται να περιμένει μια άλλη. Οι επακόλουθες καθυστερήσεις σε μερικά παράλληλα προγράμματα, άλλοτε προκαλούν "λειτουργική συμφόρηση", μειώνοντας τη συνολική επιτάχυνση.
5. Ανισορροπία Φορτίου (Load Imbalance). Οι υπολογιστικές εργασίες σε ορισμένα παράλληλα προγράμματα, παράγονται δυναμικά και με απρόβλεπτο τρόπο και πρέπει αμέσως μόλις παράγονται να μεταβιβάζονται στους επεξεργαστές. Έτσι, μερικοί επεξεργαστές παραμένουν αδρανείς, ενώ κάποιοι άλλοι έχουν περισσότερο υπολογιστικό φορτίο από αυτόν που μπορούν να χειριστούν.

# 4. Σύγχρονος και ασύγχρονος προγραμματισμός

Ο παράλληλος προγραμματισμός διακρίνεται σε δύο κύρια είδη, τον ασύγχρονο και το σύγχρονο, που βασίζονται σης αρχιτεκτονικές των αντίστοιχων παράλληλων μηχανών.

## 4.1. Σύγχρονος παράλληλος προγραμματισμός

Ο σύγχρονος παράλληλος προγραμματισμός χρησιμοποιείται σε επεξεργαστές μητρώου τύπου μηχανών μοναδικής εντολής, πολλαπλών δεδομένων, καθώς και σε διανυσματικούς επεξεργαστές. Στις μηχανές αυτές, οι εντολές είτε εκτελούνται παράλληλα σε πολλές μονάδες επεξεργασίας' ή αλληλοεπικαλύπτονται χρονικά πάνω σε διαφορετικά δεδομένα (μέθοδοι σωληνώσεως). Οι αλληλεπιδράσεις μεταξύ των στοιχείων επεξεργασίας στο σύγχρονο παράλληλο προγραμματισμό, λαμβάνουν ιδιαίτερα περιορισμένη μορφή και ως εκ τούτου ο προγραμματισμός των μηχανών αυτών είναι σχετικά απλός αλλά οι εφαρμογές τους είναι περιορισμένες.

Στο σύγχρονο παράλληλο προγραμματισμό απαιτείται συνήθως η επανάληψη κάποιας στοιχειώδους πράξης για όλα τα στοιχεία του διανύσματος ή των διανυσμάτων. Π.χ. η πρόσθεση δύο διανυσμάτων απαιτεί πρόσθεση των αντίστοιχων στοιχείων τους, ενώ η λήψη της ρίζας ενός διανύσματος υλοποιείται λαμβάνοντας τη ρίζα του κάθε στοιχείου του διανύσματος. Πολλές εφαρμογές επεξεργάζονται διανύσματα αποτελούμενα από πολλές χιλιάδες στοιχεία. Η ομοιομορφία και η ανεξαρτησία των στοιχειωδών αυτών πράξεων δημιουργούν κατάλληλες προϋποθέσεις για παράλληλη επεξεργασία.

Οι διανυσματικοί επεξεργαστές εκμεταλλεύονται τις παραπάνω προϋποθέσεις κάνοντας χρήση της τεχνικής της σωληνώσεως (pipelining). Η Αριθμητική Λογική Μονάδα τους αποτελείται από αριθμό υπομονάδων που ονομάζονται στάδια συνδεδεμένα μεταξύ τους με τη μορφή σωληνώσεως.

Ο σύγχρονος παράλληλος προγραμματισμός χρησιμοποιείται στους επεξεργαστές μητρώου μηχανών μοναδικής εντολής, πολλαπλών δεδομένων και σε διανυσματικούς επεξεργαστές. Οι μηχανές αυτές έχουν μία μόνο μονάδα ελέγχου και πολλά στοιχεία επεξεργασίας που είναι όλα συγχρονισμένα με το ρολόι του συστήματος. Ο σύγχρονος παράλληλος προγραμματισμός είναι σαφώς απλούστερος του ασύγχρονου παράλληλου προγραμματισμού αφού:

1. Δεν υπάρχουν πολλαπλές συνεργαζόμενες διεργασίες.
2. Η επικοινωνία γίνεται μόνο για την ανταλλαγή πληροφοριών, συγχρονισμός και αμοιβαίος αποκλεισμός δεν υπάρχουν, επίσης δεν συναντάται και το φαινόμενο του αδιέξοδου.

Ο συγχρονισμός διεργασιών μπορεί να δημιουργήσει σημεία συμφόρησης μέσα στο πρόγραμμα. Από τη φύση του ο συγχρονισμός είναι αντίθετος προς τον παραλληλισμό, αφού όλες οι διεργασίες πρέπει να βρίσκονται κάτω από κάποιον κεντρικό έλεγχο. Όμως αυτή η διαδικασία μπορεί να προκαλέσει ανταγωνισμό και καθυστέρηση στην εκτέλεση των διεργασιών.

## 4.2. Γλώσσες σύγχρονου παράλληλου προγραμματισμού

Στο σύγχρονο παράλληλο προγραμματισμό, δεν υπάρχουν πολλαπλές διεργασίες, μια διεργασία απλώς χειρίζεται παράλληλα πολλαπλά δεδομένα. Έτσι δεν απαιτούνται πολύπλοκες δομές επικοινωνίας. Ο παραλληλισμός μπορεί να εκφρασθεί μέσω της συνάρτησης δεδομένων με διαφορετικά στάδια σωληνώσεως ή με διαφορετικές μηχανές επεξεργασίας. Έτσι δεν απαιτείται χρήση πολύπλοκων δομών έκφρασης παραλληλισμού και εύκολα μπορεί να επεκταθεί μια κοινή σειριακή γλώσσα για την κάλυψη βασικών αναγκών του σύγχρονου παράλληλου προγραμματισμού. Αυτό έχει και τα πλεονεκτήματα της ταχύτερης μετατροπής υπαρχουσών εφαρμογών και της ευκολότερης παραδοχής από τους χρήστες.

Οι περισσότερες εφαρμογές που χρησιμοποιούν μεγάλα διανύσματα ή μεγάλα μητρώα ανήκουν στον επιστημονικό χώρο. Η Fortran κυριαρχεί σαν γλώσσα σύγχρονου παράλληλου προγραμματισμού. Όλοι σχεδόν οι διανυσματικοί επεξεργαστές καθώς και επεξεργαστές μητρώου που κατέκτησαν κάποια θέση στην αγορά, παρείχαν μεταγλωττι-στές της Fortran. Οι DAP Fortran, CDC Cyber Fortran και Cray-1 Fortran (CFT) είναι μερικά παραδείγματα χρήσεως Fortran στις αντίστοιχες μηχανές.

Οι διανυσματικοί επεξεργαστές συχνά συνοδεύονται από κοινές σειριακές γλώσσες και μεταγλωττιστές-παραλληλοποιητές (parallelizing compilers), που συνήθως παραλληλοποιούν εντολές LOOP πάνω σε διανύσματα. Στην περίπτωση των επεξεργαστών μητρώου τα δεδομένα πρέπει να είναι κατανεμημένα στις μνήμες των μηχανών επεξεργασίας. Η διασπορά των δεδομένων έχει υψηλό κόστος και το πρόγραμμα πρέπει να είναι γραμμένο κατά τέτοιο τρόπο, ώστε διαδοχικές λειτουργίες να κάνουν όσο το δυνατόν μεγαλύτερη χρήση των τοπικών δεδομένων των μηχανών επεξεργασίας δηλ. χωρίς ανακατάταξή τους. Ιδανική περίπτωση είναι όταν κάθε λειτουργία μπορεί να χρησιμοποιήσει τα δεδομένα όπως ακριβώς τα άφησε η προηγούμενή της χωρίς δηλ. την ανάγκη επικοινωνίας. Η υλοποίηση τέτοιων προγραμμάτων απαιτεί βαθιά γνώση των αντίστοιχων αλγορίθμων, γι' αυτό και η κατασκευή αυτών των αλγορίθμων είναι πολύπλοκη. Οι γλώσσες επεξεργαστών μητρώου συνήθως είναι είτε επεκτάσεις σειριακών γλωσσών που περιλαμβάνουν εντολές που εκφράζουν τον παραλληλισμό της μηχανής, είτε σειριακές γλώσσες μέσα στον κώδικα των οποίων μπορούν να παρεμβληθούν εντολές γλώσσας μηχανής. Έτσι η χρήση του παραλληλισμού αφήνεται αποκλειστικά στον προγραμματιστή.

## 4.3. Ασύγχρονος παράλληλος προγραμματισμός

Ο παράλληλος προγραμματισμός, (σε αντίθεση με τον ακολουθιακό) δεν έχει ακόμα φτάσει σε τέτοιο επίπεδο εξελίξεως ώστε ο προγραμματιστής εφαρμογής να μπορεί να προγραμματίζει ανεξάρτητα από το είδος της παράλληλης μηχανής που χρησιμοποιεί. Η μεγιστοποίηση της απόδοσης που αποτελεί τον κύριο σκοπό παράλληλης επεξεργασίας, δύσκολα επιτυγχάνεται χωρίς τη γνώση και εκμετάλλευση των ιδιαίτερων χαρακτηριστικών μιας παράλληλης μηχανής.

Ο ασύγχρονος παράλληλος προγραμματισμός χρησιμοποιείται σε συστήματα πολυεπεξεργαστών τύπου μηχανών πολλαπλών εντολών, πολλαπλών δεδομένων και βασίζεται σε παράλληλες ανεξάρτητες διεργασίες που αλληλεπιδρούν μόνον όταν η επικοινωνία τους είναι απαραίτητη. Ο ασύγχρονος παράλληλος προγραμματισμός ξεκίνησε στα λειτουργικά συστήματα της δεκαετίας του '70, τότε που οι παράλληλες διεργασίες μοιράζονταν κατά κανόνα τον ίδιο επεξεργαστή (πολυπρογραμματισμός).

Εργαλεία παράλληλου προγραμματισμού

Σε σχέση με το σειριακό προγραμματισμό, ο παράλληλος προγραμματισμός θεωρείται αρκετά δύσκολη δουλειά. Κύριες δυσκολίες του παράλληλου προγραμματισμού αποτελεί: α) η ύπαρξη δρώμενων που εκτελούν υπολογισμούς ασύγχρονα και επικοινωνούν μεταξύ τους ανταλλάσσοντας μηνύματα, β) η ανάγκη θεώρησης νέων παραγόντων π.χ. κόστος ανταλλαγής μηνυμάτων, γ) η εξισορρόπηση του υπολογιστικού φορτίου κάθε επεξεργαστή, δ) η έλλειψη κοινής μνήμης (στην περίπτωση των υπολογιστών κατανεμημένης μνήμης), ε) η μεταφερτότητα, αφού διαφορετικές μηχανές υποστηρίζουν διαφορετικές λειτουργίες, μοντέλα προγραμματισμού και έχουν διαφορετικό κόστος λειτουργίας και προγραμματισμού.

Ένα εργαλείο ανάπτυξης παραλλήλων προγραμμάτων πρέπει να είναι κατάλληλο στις περισσότερες μηχανές και η απαίτηση να αφορά στις μηχανές κοινής και κατανεμημένης

μνήμης. Ο προγραμματιστής πρέπει να έχει τη δυνατότητα να γράφει το πρόγραμμά του, χωρίς να γνωρίζει λεπτομέρειες και για την παράλληλη μηχανή που χρησιμοποιεί. Τα προγράμματα δεν θα πρέπει να αλλάζουν όταν "τρέχουν" σε περισσότερους επεξεργαστές ή σε μια μεγαλύτερη μηχανή του ίδιου τύπου ή όταν μεταφέρονται σε κάποια παράλληλη μηχανή διαφορετικού τύπου (εκτός αν χρειάζονται κάποια ρύθμιση για λόγους βελτίωσης της απόδοσης). Πρέπει να υπάρχει υποστήριξη στη δυναμική συμπεριφορά παραλλήλων προγραμμάτων. Υπολογισμοί που απαιτούν δυναμική συμπεριφορά του προγράμματος περιλαμβάνουν συμβολικό υπολογισμό, διακριτή βελτιστοποίηση και ειδικούς αριθμητικούς υπολογισμούς.

Τα τελευταία χρόνια έχουν αναπτυχθεί πλήθος εργαλείων παράλληλου προγραμματισμού, ώστε να διευκολυνθεί ο παράλληλος προγραμματισμός. Το μεγαλύτερο μέρος αυτών των εργαλείων αφορούν γλώσσες προγραμματισμού, συστήματα διαχείρισης κατανεμημένης μνήμης και λοιπά εργαλεία τα οποία διευκολύνουν την ανταλλαγή μηνυμάτων.

## 4.4. Ανάπτυξη παράλληλου προγράμματος

Για να αναπτυχθεί ένα παράλληλο πρόγραμμα ακολουθούνται τα εξής βήματα: α) διάσπαση συνόλου υπολογισμών σε μικρότερα σύνολα τα οποία εκτελούνται παράλληλα από κάποιο δρώμενο (παραλληλοποίηση), β) ανάθεση κάθε δρώμενου σε συγκεκριμένο επεξεργαστή (mapping), γ) απόφαση για συγχρονισμό και το πότε θα εκτελεστούν τα δρώμενα (scheduling), δ) έκφραση όλων των παραπάνω χρησιμοποιώντας τη σημασιολογία που προσφέρει η παράλληλη μηχανή. Η ανωτέρω ακολουθία βημάτων αποτελεί μια μετάβαση από το υψηλό επίπεδο (high-level) της παράλληλης μηχανής, στο χαμηλότερο επίπεδο (low-level) προγραμματισμού.

Διακρίνονται τέσσερις προσεγγίσεις σε όλα τα εργαλεία που αναπτύσσονται σχετικά με την υποστήριξη παράλληλου προγραμματισμού.

- Στην πρώτη προσέγγιση, ο προγραμματιστής γράφει την εφαρμογή του σε μια σειριακή γλώσσα (π.χ. C, FORTRAN). Έπειτα χρησιμοποιείται ένας μεταγλωττιστής παραλληλοποίησης (parallelizing compiler), ο οποίος παράγει παράλληλο πρόγραμμα, χρησιμοποιώντας τη σημασιολογία της παράλληλης μηχανής.
- Η γλώσσα PROLOG (έμμεση παράλληλη γλώσσα υψηλού επιπέδου για παράδειγμα, αποτελεί μια δεύτερη προσέγγιση. Ούτε εδώ ο προγραμματιστής ασχολείται με τη διαχείριση του παραλληλισμού. Ο μεταγλωττιστής σε συνδυασμό με το σύστημα εκτέλεσης, αποφασίζει με πιο τρόπο θα γίνει η εξαγωγή του παραλληλισμού. Αν και οι γλώσσες αυτού του τύπου είναι γενικά πιο αργές από γλώσσες π.χ. C, υπάρχουν περιοχές εφαρμογών όπου η χρήση τέτοιων γλωσσών μπορούν να μειώσουν το χρόνο ανάπτυξης. Η χρήση του παραλληλισμού μπορεί να αντισταθμίσει το μειονέκτημα της χαμηλής ταχύτητας. Με τη προσέγγιση αυτή χρησιμοποιείται ο παραλληλισμός ώστε να αυξηθεί η παραγωγικότητα των προγραμματιστών.
- Οι άμεσα παράλληλες γλώσσες υψηλού επιπέδου, μαζί με τα συστήματα χρόνου εκτέλεσής τους συνθέτουν τη τρίτη προσέγγιση. Οι γλώσσες αυτές αναλαμβάνουν τη διαχείριση αγαθών της παράλληλης μηχανής, επιτρέποντας στο χρήστη να ορίσει ποιές λειτουργίες θα εκτελεστούν παράλληλα χωρίς να είναι απαραίτητη η γνώση της παράλληλης αρχιτεκτονικής. Τέτοιες γλώσσες είναι η Concurrent Pascal, MultiLisp, Parlog.GHC) και έχουν προκύψει προσθέτοντας ειδική σημασιολογία έκφρασης του παραλληλισμού σε σειριακές γλώσσες. Άλλες γλώσσες προγραμματισμού που συνθέτουν τη τρίτη προσέγγιση

είναι η C-Linda, Occam, οι οποίες σχεδιασμένες από την αρχή ειδικά για παράλληλη επεξεργασία.

- Κάθε γλώσσα που ανήκει σε κάποια από τις προηγούμενες προσεγγίσεις έχει φτιαχτεί για μια συγκεκριμένη παράλληλη μηχανή, συνοδεύεται από ένα σύστημα χρόνου εκτέλεσης που διαχειρίζεται τα αγαθά της παράλληλης μηχανής. Ένα τέτοιο σύστημα παρέχει (δυναμική) ρύθμιση του φορτίου του κάθε επεξεργαστή, διαχείριση μνήμης (κατανεμημένης ή κοινής) και χρονοδρομολόγηση. Για να χρησιμοποιηθεί μια τέτοια γλώσσα σε έναν άλλο παράλληλο υπολογιστή, πρέπει να υλοποιηθεί απ' την αρχή το σύστημα χρόνου εκτέλεσης. Αυτό το κοινό μειονέκτημα των προηγούμενων προσεγγίσεων οδηγεί στη τέταρτη προσέγγιση εργαλείων παράλληλου προγραμματισμού. Στόχος αυτής της προσέγγισης είναι η δημιουργία ενός στρώματος λογισμικού μεταξύ παράλληλης μηχανής και γλωσσών προγραμματισμού ή στις εφαρμογές παράλληλου προγραμματισμού, το οποίο να είναι ευέλικτο, εύκολα μεταφερτό σε οποιαδήποτε παράλληλη αρχιτεκτονική, αρκετά δε ισχυρό στην υποστήριξη και ανάπτυξη μιας υψηλού επιπέδου γλώσσας παράλληλου προγραμματισμού. Αυτά τα εργαλεία (Εργαλεία ανάπτυξης παράλληλων μεταφέρσιμων εφαρμογών) είναι οι γλώσσες Chare, Orca, Kernel Language, καθώς και οι μεταφέρσιμες πλατφόρμες λογισμικού PVM, P4, Panda, Chare, Orchid. Η μεταφέρσιμη πλατφόρμα λογισμικού αποτελεί την ενδιάμεση γλώσσα για την παραλληλοποίηση οποιουδήποτε τύπου παράλληλης γλώσσας.

## 4.5. Παράγοντες απόδοσης αλγορίθμων

Η ιδανική τιμή της απόδοσης επιτυγχάνεται σε εξαιρετικά σπάνιες (και τεχνητές) περιπτώσεις. Αυτό οφείλεται κυρίως στον παράγοντα επικοινωνία.

Επικοινωνία. Η επιβάρυνση των παράλληλων αλγορίθμων με το κόστος της επικοινωνίας προκύπτει από την ανάγκη ανταλλαγής πληροφοριών μεταξύ συνεργαζόμενων επεξεργαστών, καθώς και από την ανάγκη συγχρονισμού μεταξύ τους. Η δεύτερη περίπτωση δεν συναντάται σε μηχανές μοναδικής εντολής, πολλαπλών δεδομένων, όπου υπάρχει μία μόνο μονάδα ελέγχου και συνεπώς δεν προκύπτει θέμα συγχρονισμού.

Η ανάγκη της επικοινωνίας είναι τόσο μικρότερη, όσο πιο ανεξάρτητες είναι οι διεργασίες (processes) που κατανέμονται σε διαφορετικούς επεξεργαστές. Οι πιο αποτελεσματικοί παράλληλοι αλγόριθμοι συνήθως περιορίζουν την εξάρτηση, μεταξύ των διεργασιών, πολλές φορές αυξάνοντας το φορτίο επεξεργασίας (συχνά ο υπολογισμός κάποιας τιμής είναι λιγότερο χρονοβόρος από τη μεταφορά της μεταξύ επεξεργαστών). Το κόστος επικοινωνίας εξαρτάται από το είδος της παράλληλης μηχανής. Για παράδειγμα, η επικοινωνία μέσω κοινής μνήμης είναι συνήθως ταχύτερη από την επικοινωνία με ανταλλαγή μηνυμάτων.

Τα συστήματα παράλληλης επεξεργασίας συχνά περιλαμβάνουν κοινά αγαθά των οποίων η χρήση δεν μπορεί να γίνεται από περισσότερους του ενός επεξεργαστές συγχρόνως. Παραδείγματα είναι η κοινή μνήμη (όπου μπορεί να φυλάσσεται κοινός κώδικας ή κοινά δεδομένα), ο εκτυπωτής ή η οθόνη. Σε τέτοιες περιπτώσεις οι επεξεργαστές πρέπει να συγχρονίζονται μεταξύ τους, έτσι ώστε να εξασφαλίζεται ότι μόνο ένας από αυτούς θα μπορεί να χρησιμοποιήσει το κοινό αγαθό ενώ οι υπόλοιποι πρέπει να περιμένουν. Αυτό ονομάζεται αμοιβαίος αποκλεισμός (mutual exclusion).

Οι τεχνικές επικοινωνίας βασίζονται στη χρήση κοινών μεταβλητών και έτσι είναι κατάλληλες για υλοποίηση σε παράλληλες αρχιτεκτονικές που διαθέτουν κοινή μνήμη. Αντίθετα το πέρασμα μηνυμάτων (message passing) δεν προϋποθέτει τη χρήση κοινών

μεταβλητών και είναι ιδιαίτερα χρήσιμο σε παράλληλες αρχιτεκτονικές με κατανεμημένη μνήμη.

Οι σηματοφορείς και οι δομές monitor (που είναι παραπέρα ανάπτυξη των σηματοφορέων) στηρίζονται στη γενική ιδέα ότι η επικοινωνία και ο συγχρονισμός μπορούν να γίνουν με τη βοήθεια κοινών μεταβλητών που μπορούν να προσπελαύνονται από όλες τις διεργασίες. Μια διαφορετική ιδέα είναι η επικοινωνία και ο συγχρονισμός των διεργασιών να γίνεται με το πέρασμα μηνυμάτων (message passing).

Η ανταλλαγή πληροφοριών επιτυγχάνεται με το πέρασμα τιμών από τη διεργασία που στέλνει το μήνυμα (αποστολέας), στη διεργασία που παίρνει το μήνυμα (λήπτης). Ο συγχρονισμός επιτυγχάνεται επειδή η λήψη του μηνύματος μπορεί να γίνει μόνο μετά την αποστολή του. Έτσι ο παραλήπτης πρέπει να περιμένει λήψη μηνύματος μέχρις ότου το στείλει ο αποστολέας.

## 5. Παραλληλοποίηση

Σήμερα, οι περισσότεροι αλγόριθμοι που χρησιμοποιούνται είναι ακολουθιακοί (κάθε αλγόριθμος αποτελείται από ένα σύνολο βημάτων, ενώ κάθε βήμα καθορίζει μία και μόνο λειτουργία). Ένας παράλληλος αλγόριθμος (σε αντιδιαστολή με τον ακολουθιακό) περιέχει βήματα, τα οποία αποτελούνται από πολλές λειτουργίες που εκτελούνται ταυτόχρονα.

Η παραλληλία μπορεί να βελτιώσει την απόδοση του αλγορίθμου σε διάφορα είδη μηχανών. Για παράδειγμα, σε έναν υπολογιστή με πολλούς επεξεργαστές κάποιες λειτουργίες που καθορίζει ο παράλληλος αλγόριθμος, μπορούν να εκτελεσθούν ταυτόχρονα από διαφορετικούς επεξεργαστές. Όμως η παραλληλία ενός αλγορίθμου μπορεί να αξιοποιηθεί ακόμα και σε υπολογιστές με έναν επεξεργαστή, χρησιμοποιώντας είτε σωληνωμένες λειτουργικές μονάδες, είτε πολλές λειτουργικές μονάδες ή σωληνωμένα συστήματα μνήμης. Επομένως είναι ιδιαίτερα σημαντικό να γίνει μια διάκριση μεταξύ παραλληλίας ενός αλγορίθμου και της δυνατότητας ενός υπολογιστή να εκτελεί πολλές λειτουργίες παράλληλα. Χαρακτηριστικά, ένας παράλληλος αλγόριθμος εκτελείται αποδοτικά σε έναν υπολογιστή, εάν ο αλγόριθμος περιέχει τουλάχιστον τόση παραλληλία, όση και ο υπολογιστής.

Προκειμένου ο σχεδιαστής παράλληλου αλγορίθμου να προβλέπει, με όσο το δυνατόν περισσότερη ακρίβεια, τη συμπεριφορά και το κόστος του αλγορίθμου είναι απαραίτητη η ύπαρξη ενός προτύπου παράλληλου υπολογιστή. Οι σχεδιαστές ακολουθιακών αλγορίθμων διατυπώνουν τους αλγόριθμους με βάση το πρότυπο ακολουθιακού υπολογισμού RAM (Random Access Machine). Σύμφωνα με το πρότυπο αυτό, η μηχανή αποτελείται από έναν επεξεργαστή, ο οποίος είναι συνδεδεμένος, με ένα σύστημα μνήμης, το οποίο αποτελείται από έναν αριθμήσιμο μη πεπερασμένο σύνολο θέσεων. Σκοπός του σχεδιαστή αλγορίθμου είναι να αναπτύξει αλγορίθμους με "μικρές" απαιτήσεις, τόσο σε ότι αφορά το χρόνο που αυτοί χρειάζονται για να εκτελεσθούν, όσο και σε ότι αφορά τη μνήμη που αυτοί χρησιμοποιούν. Η προτυποποίηση των παράλληλων υπολογισμών είναι πολύ πιο δύσκολη και περίπλοκη από αυτή των ακολουθιακών, για τον απλό λόγο ότι οι παράλληλοι υπολογιστές ποικίλουν περισσότερο σε ό,τι αφορά την αρχιτεκτονική τους, απ' ότι οι ακολουθιακοί.

Ένας αλγόριθμος συνήθως παραλληλοποιείται όταν διασπάται σε πολλαπλά τμήματα, στη συνέχεια αυτά ανατίθενται σε ξεχωριστές διεργασίες ή νήματα και εκτελούνται παράλληλα σε διαφορετικές επεξεργαστικές μονάδες. Παρ' όλα αυτά δεν είναι βέβαιο αν ένας αλγόριθμος που έχει υλοποιηθεί σε κάποιο πρόγραμμα, μπορεί να παραλληλοποιηθεί πάντα: για παράδειγμα μία υπορουτίνα που υπολογίζει μια ακολουθία Φιμπονάτσι μέσω ενός επαναληπτικού βρόχου δεν μπορεί να διασπαστεί σε περισσότερα και να γίνει διαμοιρασμό

των επαναλήψεων βρόχου σε άλλους μικρότερους υποβρόχους, αφού ο υπολογισμός που γίνεται σε κάθε επανάληψη εξαρτάται από τις δύο προηγούμενες επαναλήψεις – επομένως όλες οι επαναλήψεις πρέπει να εκτελεστούν σειριακά στο εσωτερικό μίας μόνο διεργασίας. Αυτές οι εξαρτήσεις αποτελούν το σημαντικότερο εμπόδιο για την παραλληλοποίηση.

Η παραλληλοποίηση ενός αλγορίθμου, διακρίνεται σε τέσσερα βήματα τα οποία εκτελούνται διαδοχικά:

- Διάσπαση ολικού υπολογισμού σε επιμέρους εργασίες. Εναπόκειται στον προγραμματιστή και εξαρτάται από το πρόβλημα.
- Ανάθεση εργασιών σε οντότητες εκτέλεσης (διακριτές διεργασίες, νήματα ή ίνες). Μπορεί να είναι είτε στατική (κάθε οντότητα αναλαμβάνει ένα προκαθορισμένο σύνολο εργασιών προς εκτέλεση) είτε δυναμική (οι εργασίες ανατίθενται μία-μία κατά τον χρόνος εκτέλεσης• όταν μία οντότητα ολοκληρώσει την τρέχουσα εργασία της ζητά να της δοθεί η επόμενη). Σε περίπτωση που οι επεξεργαστές έχουν διαφορετικό φόρτο (π.χ. αν κάποιοι επεξεργαστές εκτελούν και άλλα προγράμματα), η δυναμική λύση είναι προτιμότερη αλλά με μειονέκτημα κάποια χρονική επιβάρυνση.
- Ενορχήστρωση οντοτήτων. Καθορίζεται ο τρόπος επικοινωνίας και συντονισμού μεταξύ οντοτήτων εκτέλεσης π.χ. αποστολή ή λήψη μηνυμάτων, φράγματα εκτέλεσης κ.α.
- Αντιστοίχηση οντοτήτων σε επεξεργαστές. Άλλοτε είναι στατική (προκαθορίζεται από τον προγραμματιστή) άλλοτε δυναμική (ρυθμίζεται από το σύστημα κατά τον χρόνο εκτέλεσης).

Τα βήματα αυτά είναι κοινά για προγράμματα γραμμένα τόσο στο μοντέλο μεταβίβασης μηνυμάτων, όσο στο μοντέλο κοινού χώρου διευθύνσεων. Η διαφορά τους είναι στο τρόπο που γίνεται η επικοινωνία μεταξύ των οντοτήτων εκτέλεσης ώστε να συντονιστούν οι υπολογισμοί, να διαμοιραστεί ο υπολογιστικός φόρτος και να συλλεχθούν στο τέλος τα επιμέρους αποτελέσματα. Τα προβλήματα που από τη φύση τους απαιτούν μηδενικές ή ελάχιστες επικοινωνίες μεταξύ των εργασιών είναι αυτά που παραλληλοποιούνται πιο αποδοτικά και πιο εύκολα (π.χ. η πρόσθεση δύο ν-διάστατων διανυσμάτων).

Σε εργασίες σημαντική είναι η διάσπαση και μπορεί να γίνει με πολλούς τρόπους, ανάλογα με το εκάστοτε πρόβλημα, π.χ. λειτουργική διάσπαση, επαναληπτική διάσπαση ή αναδρομική διάσπαση (οι δύο πρώτες μέθοδοι προδιαγράφουν και το πώς γίνεται η ανάθεση σε οντότητες εκτέλεσης), γεωμετρική διάσπαση, διάσπαση πεδίου ορισμού. Υπάρχει συνήθως μία κύρια οντότητα εκτέλεσης, η οποία όταν χρειάζεται δημιουργεί τις υπόλοιπες και στο τέλος συλλέγει τα αποτελέσματα των υπολογισμών τους.

## 5.1. Τύποι παραλληλισμού

Ο παραλληλισμός έρχεται σε ποικίλα λεπτομερειών. Λέγοντας λεπτομέρεια εννοούμε τη κλίμακα του προβλήματος που παραλληλοποιείται. Ένα παράδειγμα απεικονίζεται με μια φασματική γραμμή επεξεργασίας. Στην περίπτωση αυτή, μεμονωμένα κανάλια είναι εντελώς ανεξάρτητα, έτσι ώστε, το καθένα να μπορεί να σταλεί σε ένα ξεχωριστό επεξεργαστή για υπολογισμό. Εξαιτίας ότι δεν υπάρχει επικοινωνία μεταξύ των επεξεργαστών, η επικοινωνία είναι αμελητέα και η επιτάχυνση είναι σχεδόν γραμμική με τον αριθμό των επεξεργαστών. Αυτό το είδος παραλληλοποίησης είναι εύκολο να εφαρμοστεί από τον προγραμματιστή, διότι η διεπαφή του Glish AIPS + + μπορεί να ελέγχει τις διεργασίες σε παράλληλο τρόπο. Δεν απαιτείται χαμηλό επίπεδο βελτιστοποίησης

## 5.2. Αποσύνθεση προβλήματος (problem decomposition)

Κάθε παράλληλο πρόγραμμα αποτελείται από ταυτόχρονη εκτέλεση διεργασιών • η αποσύνθεση προβλήματος σχετίζεται με τον τρόπο που διαμορφώνονται οι διαδικασίες αυτές. Η κατάταξη αυτή αναφέρεται και ως αλγοριθμικός σκελετός ή μοντέλο παράλληλου προγραμματισμού.

## 5.3. Παραλληλισμός εργασίας (task)

Ένα παράλληλο μοντέλο εργασιών εστιάζεται στις διαδικασίες ή τα νήματα της εκτέλεσης. Αυτές οι διαδικασίες είναι συχνά διακριτές και υπογραμμίζουν την ανάγκη για επικοινωνία. Ο παραλληλισμός εργασίας είναι ένας φυσικός τρόπος να εκφραστεί το "μήνυμα-πέρασμα" κατά την επικοινωνία. Συνήθως ταξινομείται ως MIMD/MPMD ή MISD.
Παραλληλισμός δεδομένων

Παραλληλισμός δεδομένων είναι η χρήση πολλών λειτουργικών μονάδων προκειμένου να εφαρμοσθεί η ίδια λειτουργία ταυτόχρονα στα στοιχεία ενός συνόλου δεδομένων. Η εφαρμογή παράλληλων υπολογισμών στα επιστημονικά προβλήματα πραγματοποιείται ευρέως και βασίζεται τόσο στη χρήση μεγάλων πολυδιάστατων πινάκων όσο και άλλων μεγάλων δομών δεδομένων. Για την παραγωγή κάποιου αριθμητικού αποτελέσματος, ο κεντρικός πυρήνας των περισσοτέρων παραδοσιακών αλγορίθμων αποτελείται από σειρές φωλιασμένων βρόχων, εκτελώντας πολύπλοκους χειρισμούς πινάκων. Ωστόσο, σήμερα τα περισσότερα παράλληλα προγράμματα δημιουργούνται με την αναδιοργάνωση αυτών των σειριακών αλγόριθμων με σκοπό οι φωλιασμένοι βρόχοι να εκτελούνται παράλληλα. Αυτό σημαίνει, ότι υπάρχει ο παραλληλισμός και στην παραγωγή των αποτελεσμάτων του προγράμματος, λόγω της παράλληλης εφαρμογής της ίδιας λειτουργίας σε διαφορετικά τμήματα του πίνακα δεδομένων.

Ένα παράλληλο μοντέλο δεδομένων εστιάζει στην εκτέλεση εργασιών σε ένα σύνολο δεδομένων, τα οποία είναι συνήθως δομημένα τακτικά σε μια σειρά. Μια σειρά από καθήκοντα που θα λειτουργούν σε αυτά τα δεδομένα, αλλά ανεξάρτητα σε διαφορετικές κατατμήσεις. Σε ένα σύστημα κοινής μνήμης, τα δεδομένα είναι προσβάσιμα σε όλους, αλλά σε ένα σύστημα κατανεμημένης μνήμης διαιρούνται μεταξύ των μνημών και "δουλεύουν" σε τοπικό επίπεδο.

Στον παραλληλισμό δεδομένων η δομή παραλληλισμού ανταποκρίνεται στη δομή δεδομένων. Σε μερικές περιπτώσεις, κάθε στοιχειώδης μονάδα δεδομένων υφίσταται παράλληλη επεξεργασία. Άλλοτε, η δομή δεδομένων κατηγοριοποιείται σε ομάδες ανεξάρτητων μονάδων δεδομένων, με τη κάθε ομάδα να υφίσταται παράλληλη επεξεργασία. Ο παραλληλισμός δεδομένων δεν εφαρμόζεται μόνο σε επιστημονικούς αλγόριθμους με πολλές αριθμητικές πράξεις αλλά και σε μη αριθμητικά προβλήματα π.χ. ταξινόμηση, αλγόριθμοι γραφημάτων, συνδυαστικής αναζήτησης κ.α. Οι εφαρμογές αυτές χρησιμοποιούν δομές που περιέχουν μεγάλους αριθμούς παρόμοιων μονάδων δεδομένων και αποθηκεύονται σε μεγάλους πίνακες (μερικές φορές χρησιμοποιούνται και δομές με δείκτες π.χ. συνδεδεμένες λίστες).

Στους παράλληλους τύπους δεδομένων γίνεται μεγάλη χρήση της εντολής FORALL (παράλληλος τύπος επαναληπτικού βρόχου), όπου όλες οι επαναλήψεις εκτελούνται παράλληλα. Παρ' όλα αυτά οι επαναληπτικοί βρόχοι δεν λειτουργούν αμέσως παράλληλα, αφού συχνά, ορισμένες προσωρινές τιμές συσσωρεύονται και μεταφέρονται από τη μία επανάληψη στην επόμενη, έχοντας ως αποτέλεσμα τη σειριακή εκτέλεση των επαναλήψεων.

Στις περιπτώσεις αυτές, η σειριακή έκδοση του αλγόριθμου πρέπει να τροποποιείται σημαντικά για να προκύψει η παράλληλη έκδοση, σε τέτοιο σημείο ώστε να πρόκειται ουσιαστικά για τη δημιουργία ενός νέου αλγορίθμου. Να σημειωθεί ότι με την παράλληλη εκτέλεση βρόχων προκύπτουν πολύπλοκα θέματα απόδοσης π.χ. χρόνος δημιουργίας διεργασιών, ανταγωνισμός πρόσβασης στη μνήμη, καθυστερήσεις συγχρονισμού, κ.α. τα οποία έχουν αναλυθεί.

## 5.4. Διαμέριση

Η διαμέριση (partitioning) είναι από τις σημαντικότερες παραμέτρους ενός παράλληλου υπολογιστικού συστήματος. Ένα παράλληλο υπολογιστικό σύστημα για να είναι διαμερίσιμο πρέπει:

- το παράλληλο σύστημα να μπορεί να διαμερισθεί σε ανεξάρτητα υποσυστήματα, με την έννοια ότι, κανένα υποσύστημα δεν θα πρέπει να μπορεί να επεμβαίνει στην απρόσκοπτη λειτουργία κάποιου άλλου υποσυστήματος και
- το δίκτυο διασύνδεσης του συστήματος να μπορεί να διαμερισθεί σε υποδίκτυα, τα οποία διατηρούν την πλήρη λειτουργικότητα του αρχικού δικτύου.
- Τα παράλληλα συστήματα για να διαμεριστούν πρέπει να διαθέτουν τα εξής χαρακτηριστικά, τα οποία είναι πολύ σημαντικά για τη λειτουργία και την απόδοσή τους:
- Ανοχή σφάλματος: σε περιπτώσεις που κάποιος επεξεργαστής δεν λειτουργεί σωστά, να μην επηρεάζεται η λειτουργία ολόκληρου του συστήματος, αλλά αυτές οι διαμερίσεις που περιλαμβάνουν το συγκεκριμένο επεξεργαστή.
- Πολλαπλοί ταυτόχρονοι χρήστες: εφόσον υπάρχουν πολλαπλές ανεξάρτητες διαμερίσεις της παράλληλης μηχανής, να υπάρχουν πολλαπλοί ταυτόχρονοι χρήστες, όπου ο καθένας να μπορεί να τρέχει διαφορετικό παράλληλο πρόγραμμα.
- Εντοπισμό σφάλματος: σε εφαρμογές που απαιτούν μεγάλη αξιοπιστία, να μπορεί να εκτελείται το ίδιο πρόγραμμα, με τα ίδια δεδομένα, σε διαφορετικές διαμερίσεις και να συγκρίνονται τα αντίστοιχα αποτελέσματα.
- Ευκολότερη ανάπτυξη προγραμμάτων: η αποσφαλμάτωση (debugging) ενός προγράμματος είναι ευκολότερη, όταν εκτελείται σε διαμερίσεις μικρού μεγέθους. Έτσι, αντί να επιτελείται η αποσφαλμάτωση παράλληλου προγράμματος σε όλους τους επεξεργαστές παράλληλης μηχανής (π.χ. για 572 επεξεργαστές), να δύναται σε μία διαμέριση μικρότερου μεγέθους (π.χ. 76 επεξεργαστών) και στη συνέχεια να επεκτείνεται στο σύνολο των επεξεργαστών.
- Αυξημένη χρησιμοποίηση του συστήματος (increased utilization): εάν μία διεργασία χρειάζεται π.χ. μόνον N/4 από τους N διαθέσιμους επεξεργαστές, οι υπόλοιποι 3N/4 μπορούν να αξιοποιηθούν από μία ή περισσότερες άλλες διεργασίες.
- Χρήση βέλτιστου αριθμού επεξεργαστών: σε αρκετές περιπτώσεις η χρήση περισσότερων από τους απαιτούμενους επεξεργαστές, αυξάνει το χρόνο εκτέλεσης ενός προγράμματος ή μιας διεργασίας. Με την επιλογή κατάλληλου μεγέθους υποσυστήματος που θα χρησιμοποιηθεί, ελαχιστοποιείται ο χρόνος εκτέλεσης του προγράμματος.

## 6. Πολυπύρηνα συστήματα

Σ' έναν πολυπύρηνο επεξεργαστή μπορεί να υπάρχουν δύο ή περισσότερες ανεξάρτητες κεντρικές μονάδες επεξεργασίες (καλούνται πυρήνες), πάνω στους οποίους μπορεί να "τρέχουν" πολλές διεργασίες, ή πολλά νήματα ταυτόχρονα. Αυτό πραγματοποιείται λόγω ότι οι διεργασίες τρέχουν σε ανεξάρτητους πυρήνες, μπορούν να επικοινωνούν μεταξύ τους όταν αυτό χρειάζεται. Ειδικότερα σε συστήματα κατανεμημένης μνήμης, κάθε επεξεργαστής έχει δική του μνήμη, η δε επικοινωνία μεταξύ διεργασιών και πυρήνων γίνεται με ανταλλαγή μηνυμάτων. Αντίθετα, σε συστήματα κοινής μνήμης, οι πυρήνες διασυνδέονται μεταξύ τους χρησιμοποιώντας κάποια κοινή μνήμη και με χρήση κοινών μεταβλητών επικοινωνούν οι διεργασίες για ανταλλαγή πληροφοριών.


Εικ. 1: Πολυπύρηνος επεξεργαστής Intel Core 2 Duo E6750 δύο πυρήνων

Τα σύγχρονα λειτουργικά συστήματα υποστηρίζουν πλήρως πολυπύρηνες αρχιτεκτονικές, στις οποίες δεν απαιτείται επιπλέον λογισμικό. Ένα πολυπύρηνο SMP είναι το CC-UMA (τυπικό παράλληλο σύστημα μοιραζόμενης μνήμης). Στα συστήματα αυτά, οι σύγχρονοι compilers (σε μερικές περιπτώσεις με τη βοήθεια ειδικών διαθέσιμων βιβλιοθηκών) μπορούν να παράγουν κατάλληλο κώδικα.

Οι περισσότερες γλώσσες προγραμματισμού λόγω της σταδιακής επικράτησης των πολυπύρηνων επεξεργαστών, ενισχύονται με βιβλιοθήκες και API πολυνηματικού προγραμματισμού. Ειδικότερα, η Intel έχει παρουσιάσει τα Thread Building Blocks (TBB), η Microsoft διαθέτει τη Task Parallel Library (TPL) για τη C# & Cilk++, η Java τη Java Threads κ.α. Επίσης υπάρχουν πλήθος δημόσιων διαθέσιμων βιβλιοθηκών, στις οποίες η λειτουργικότητα είναι παρόμοια αλλά με κάποιες διαφορετικές λεπτομέρειες. Σε συστήματα πολλαπλών πυρήνων ειδικού σκοπού κάθε πυρήνας ασχολείται με άλλη εργασία π.χ. media player, παιχνιδομηχανές κ.α.

# 7. Συμμετρική πολυεπεξεργασία

Στη συμμετρική πολυεπεξεργασία (symmetric multiprocessing-SMP) δύο ή περισσότεροι όμοιοι, ισότιμοι επεξεργαστές (υλοποιημένοι και ως διαφορετικοί «επεξεργαστικοί πυρήνες» στο ίδιο μικροτσίπ), συνδέονται με μια κοινή διαμοιραζόμενη κύρια μνήμη (μέσω ενός διαύλου), έχοντας πρόσβαση στις ίδιες συσκευές εισόδου/εξόδου. Σήμερα, τα περισσότερα πολυεπεξεργαστικά συστήματα χρησιμοποιούν συμμετρική αρχιτεκτονική. Στη συμμετρική πολυεπεξεργασία επιτρέπονται λειτουργίες συστήματος και εφαρμογές χρήστη να εκτελούνται σε οποιοδήποτε επεξεργαστή, προσδίδοντας μεγαλύτερη ευκινησία και καλύτερη απόδοση στο σύστημα. Με την υποστήριξη του λειτουργικού συστήματος ένα σύστημα συμμετρικής πολυεπεξεργασίας μετακινεί διεργασίες (μετανάστευση) μεταξύ επεξεργαστών προκειμένου να εξισορροπηθεί ο φόρτος εργασίας στον καθένα. Χρησιμοποιώντας υλικό κοινόχρηστης μνήμης σε μια συμμετρική πολυεπεξεργασία, μειώνεται σημαντικά η ποσότητα δεδομένων και ο αριθμός μηνυμάτων που μεταδίδονται. Στους προσωπικούς υπολογιστές οι μητρικές πλακέτες που κάνουν πολυεπεξεργασία χρησιμοποιούν τη συμμετρική.

Η συμμετρική πολυεπεξεργασία χρησιμοποιείται κυρίως στην επιστήμη, τη βιομηχανία και επιχειρήσεις, στις οποίες συχνά χρησιμοποιείται λογισμικό επεξεργασίας πολλαπλών νημάτων. Ωστόσο, τα περισσότερα καταναλωτικά προϊόντα, όπως επεξεργαστές κειμένου και ηλεκτρονικά παιχνίδια γράφονται με τέτοιο τρόπο ότι δεν μπορούν να κερδίσουν μεγάλα οφέλη από τα ταυτόχρονα συστήματα. Στα παιχνίδια, αυτό γίνεται συνήθως, επειδή η σύνταξη ενός προγράμματος για την αύξηση της απόδοσης σε συστήματα συμμετρικής πολυεπεξεργασίας, μπορεί να παράγει απώλεια επιδόσεων σε μονοεπεξεργαστικά συστήματα. Πρόσφατα, ωστόσο, τσιπ πολλαπλών πυρήνων ενσωματώνονται όλο και πιο συχνά σε νέους υπολογιστές και έτσι η ισορροπία μεταξύ εγκατεστημένου υπολογιστή ενός ή πολλών πυρήνων μπορεί να αλλάξει τα επόμενα έτη.

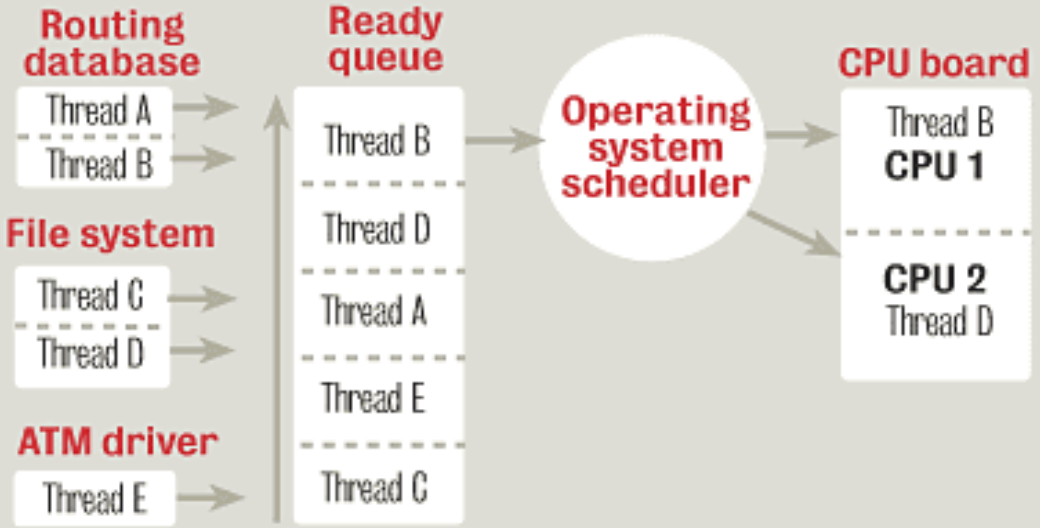
Τα συστήματα συμμετρικής πολυεπεξεργασίας απαιτούν διαφορετικές μεθόδους προγραμματισμού για να επιτευχθεί μέγιστη απόδοση. Προγράμματα που εκτελούνται σε συστήματα SMP, ενδέχεται να αντιμετωπίσουν αύξηση απόδοσης, ακόμα και όταν έχουν γραφεί για συστήματα υπολογιστών ενός επεξεργαστή. Σε ορισμένες εφαρμογές, ιδιαίτερα μεταγλωττιστές καθώς και κάποια projects κατανεμημένης επεξεργασίας, υπάρχει βελτίωση κατά ένα συντελεστή (σχεδόν) ο αριθμός των επιπλέον επεξεργαστών.

Ένα ομοιογενές σύστημα επεξεργαστή απαιτεί συνήθως επιπλέον καταχωρητές για "ειδικές οδηγίες", όπως SIMD (MMX, SSE, κλπ.), ενώ ένα ετερογενές σύστημα μπορεί να εφαρμόσει διαφορετικά είδη υλικού για διαφορετικές οδηγίες / χρήσεις.

## HOW IT WORKS

## Symmetric multiprocessing

**With SMP, the control plane of a network element can run many tasks, or threads, simultaneously, using a single CPU board. Performance is improved dramatically, as is system density.**



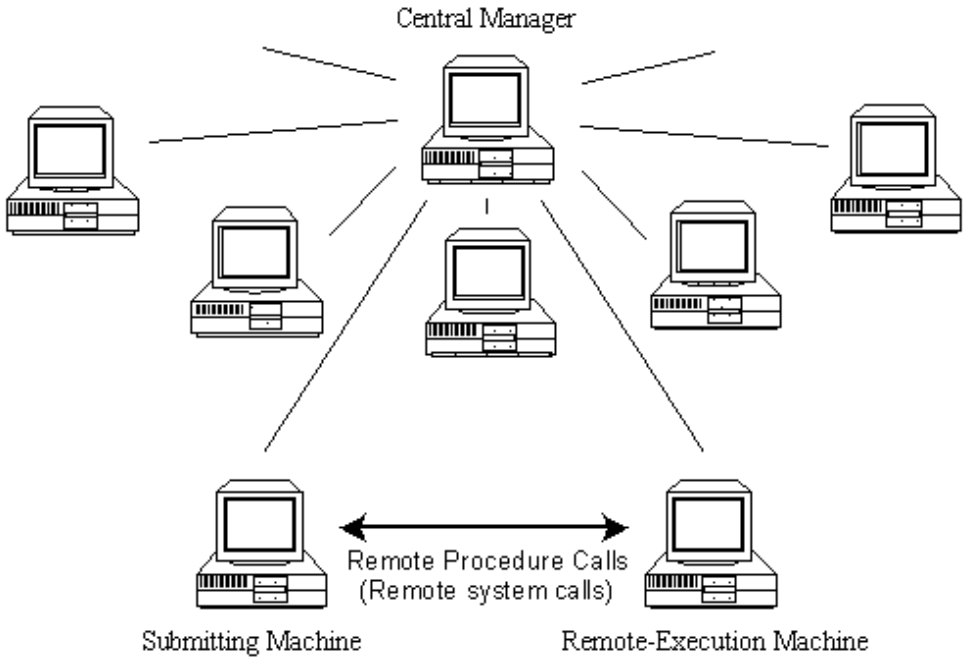

1. Several threads become ready to run.
2. The threads are queued according to their priority.
3. If there are more ready threads than CPUs, the operating system scheduler will use thread priority to decide which one runs first.
4. Multiple threads run simultaneously, one per CPU.

## 8. Κατανεμημένα υπολογιστικά συστήματα

Στα παράλληλα και κατανεμημένα υπολογιστικά συστήματα οι απαιτήσεις υπερβαίνουν τις δυνατότητες των πιο ισχυρών σχεδιαζόμενων ή υφιστάμενων σειριακών υπολογιστών. Οι εργασίες για κατανεμημένο υπολογισμό εκτείνονται ευρέως σε τομείς που αφορούν τη σχεδίαση παράλληλων μηχανών, προγραμματισμό παράλληλων γλωσσών, την ανάλυση ή ανάπτυξη παράλληλου αλγορίθμου, αλλά και θέματα που σχετίζονται με εφαρμογές.

Ο σχεδιασμός παράλληλων αλγορίθμων εφαρμόζεται σε πολλούς τομείς, δείχνοντας έτσι το βαθμό ποικιλομορφίας στην επιστημονική βιβλιογραφία σχετικά με το θέμα. Π.χ.

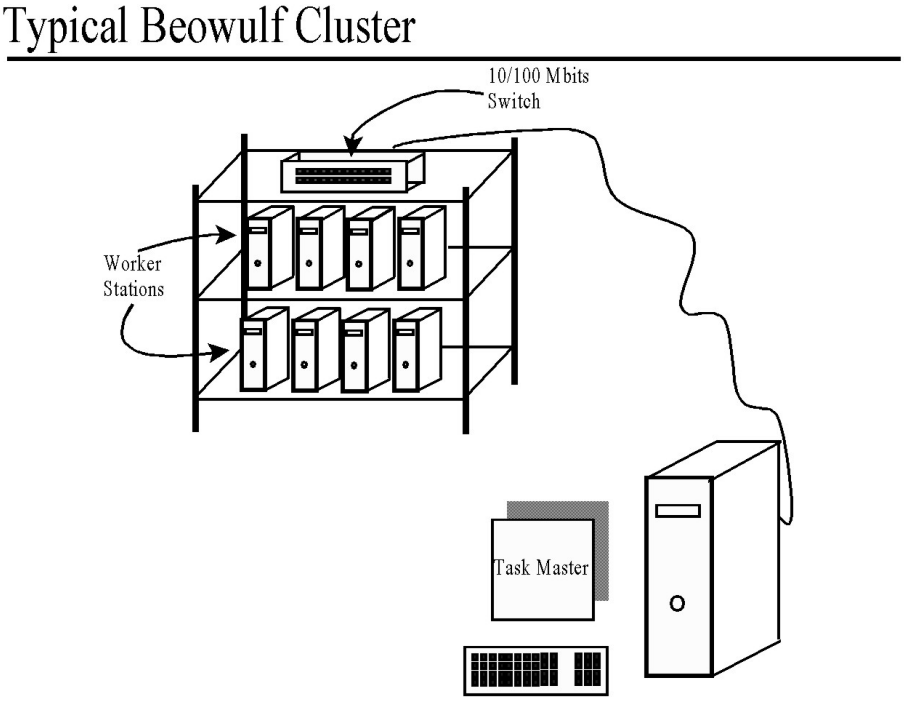
- Μια πρώτη προσέγγιση είναι η παραλληλοποίηση ενός υπάρχων σειριακού αλγορίθμου (ακόμα και μετά από τροποποιήσεις) ή η ανάπτυξη ενός νέους εύκολου αλγόριθμου παραλληλοποίησης, χωρίς να στην εξειδίκευση στην εφαρμογή συγκεκριμένων τύπων μηχανών. Τα θέματα εδώ είναι η σύγκλιση του αλγορίθμου, όπως και ο ρυθμός σύγκλισης (σύγχρονο, ασύγχρονο υπολογιστικό περιβάλλον), αλλά και η δυνατότητα επιτάχυνσης διεργασιών πάνω στο σειριακό αλγόριθμο.
- Μια δεύτερη προσέγγιση είναι η επικέντρωση μιας εφαρμογής στις λεπτομέρειες πάνω σ' ένα συγκεκριμένο τύπο μηχανής. Εδώ θα μπορούσε κάποιος να ασχοληθεί με την αλγοριθμική ορθότητα, το χρόνο και τη πολυπλοκότητα επικοινωνίας της εφαρμογής.
- Επίσης, άλλη προσέγγιση είναι η επιλογή αλγόριθμου και η παράλληλη μηχανή να αλληλεξαρτώνται σε σημείο όπου ο σχεδιασμός του ενός να έχει ισχυρή επίδραση στο σχεδιασμό του άλλου. Π.χ. όταν ένα τσιπ VLSI έχει σχεδιαστή για την αποτελεσματική εκτέλεση ενός ειδικού τύπου παράλληλου αλγόριθμου.



# 9. Cluster

Cluster είναι το τυπικό παράλληλο σύστημα κατανεμημένης μνήμης. Κάθε κόμβος του cluster έχει ένα πλήρες λειτουργικό σύστημα, στον οποίο διασφαλίζεται και η δικτυακή επικοινωνία μέσω TCP/IP. Επιπλέον, ειδικό ενδιάμεσο λογισμικό (middleware) εγκαθίσταται σε κάθε κόμβο, εξασφαλίζοντας:

- Δικτυακή (ενιαία) διαχείριση αρχείων σε όλους τους κόμβους (π.χ. NFS) για διαχείριση δεδομένων κ.α.
- γρήγορη επικοινωνία με ειδική διαχείριση του πρωτοκόλλου TCP/IP ή παράκαμψη.
- Βιβλιοθήκες, σύστημα εκτέλεσης (Run time system) παράλληλων προγραμμάτων, με επέκταση σε υπάρχουσες γλώσσες προγραμματισμού, μεταγλώττιση, μεταφορά κ.α.
- Ενημέρωση, παρακολούθηση λογισμικού συστήματος κόμβων, εργαλεία συντονισμένης διαχείρισης. Σχετικά με τα Clusters, υπάρχουν ειδικές διανομές λειτουργικών συστημάτων π.χ. τα Rocks, Oscar, Mosix κ.α. Όσον αφορά την εγκατάστασή τους γίνεται από μεμονωμένα πακέτα λογισμικού. Στις περισσότερες περιπτώσεις ένας από τους κόμβους του Cluster έχει πρωτεύοντα ρόλο (front-end) στον οποίο έχει εγκατασταθεί το σύνολο του παραπάνω λογισμικού και διεξάγεται η επικοινωνία με τον "έξω κόσμο", με μονάδες μαζικής αποθήκευσης δεδομένων. Αντίθετα στους υπόλοιπους κόμβους είναι εγκατεστημένο μόνο ένα απαραίτητο υποσύνολο.



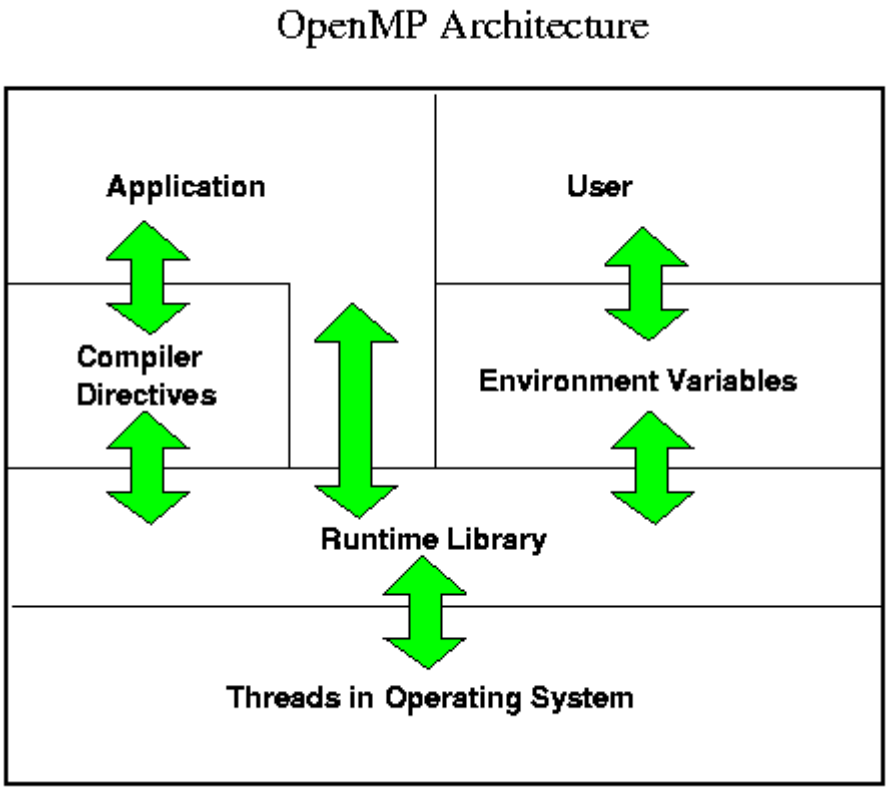
## 9.1. OPENMP

Η εφαρμογή OpenMP προγραμματισμού (API) είναι ένα πρότυπο παράλληλου προγραμματισμού σε πολυεπεξεργαστές κοινής μνήμης. Για την υποστήριξη του χρόνου εκτέλεσης, ορίζει ένα σύνολο οδηγιών του προγράμματος και μια βιβλιοθήκη στα πρότυπα C, C + + και Fortran. Η διαφορά με το MPI, είναι ότι το API μεταφοράς μηνυμάτων (OpenMP) επιτυγχάνεται καλύτερα στον παραλληλισμό διαδοχικών προγραμμάτων: Ο προγραμματιστής κάθε φορά μπορεί να προσθέτει μια οδηγία παραλληλισμού σε μια υπορουτίνα ή ένα βρόχο. Το OpenMP, σε αντίθεση με τα νήματα POSIX, προσαρμόζεται καλύτερα στις ανάγκες του επιστημονικού προγραμματισμού π.χ. Fortran και παραλληλισμό δεδομένων. Πλεονέκτημα έχει την ενοποίησης διαφορετικών μοντέλων σε μια ενιαία "σύνταξη" και "σημασιολογία", για παροχή φορητότητας σε μια-πηγή από παραλληλισμό κοινής μνήμης.

Στο API OpenMP, καθορίζεται ένα σύνολο οδηγιών προγράμματος, στο οποίο επιτρέπεται ο σχολιασμός ενός σειριακού-ακολουθιακού προγράμματος για την υπόδειξη πώς θα πρέπει να εκτελεστεί παράλληλα. Η θεμελιώδης οδηγία έκφρασης παραλληλισμού είναι η παράλληλη οδηγία. Σ' αυτήν ορίζεται μια παράλληλη περιοχή του προγράμματος, την οποία εκτελούν πολλαπλά νήματα. Τα νήματα που εκτελούν τον ίδιο υπολογισμό καθορίζονται στην παράλληλη περιοχή. Η "διαίρεση υπολογισμού" αποτελεί μία από τις οδηγίες κοινών εργασιών μεταξύ των νημάτων. Στην περίπτωση αυτή μια οδηγία διευκρινίζει ότι οι επαναλήψεις που βρόχου πρέπει να κατανέμονται μεταξύ των νημάτων, ώστε ένα μόνο νήμα να εκτελεί κάθε επανάληψη.

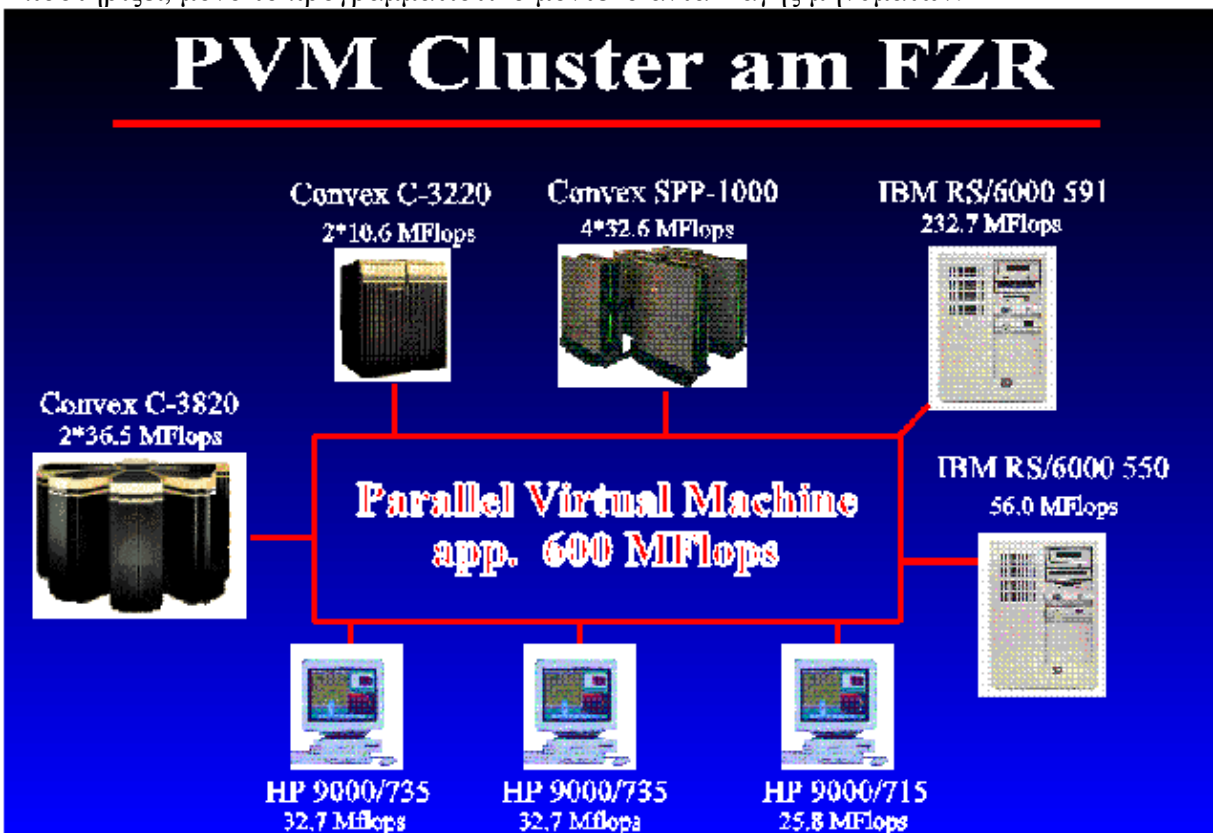
Εργαλεία ανάπτυξης παράλληλων μεταφέρσιμων εφαρμογών

Σήμερα υπάρχουν αρκετά εργαλεία παράλληλων μεταφέρσιμων εφαρμογών. Σ' αυτή την ενότητα παρουσιάζονται ενδεικτικά εργαλεία.



## 9.2. Σύστημα Parallel Virtual Machine - PVM

Το σύστημα παράλληλης εικονικής μηχανής (PVM) είναι η πιο διαδεδομένη και εξελιγμένη πλατφόρμα στην υλοποίηση εφαρμογών ανταλλαγής μηνυμάτων σε παράλληλα συστήματα UNIX. Αναπτύχθηκε μετά από συνεργασία του εργαστηρίου Oak Ridge με τα πανεπιστήμια του Tenesse, το πανεπιστήμιο Emory και το πανεπιστήμιο Carnegie Mellon. Το σύστημα PVM χρησιμοποιούν υπολογιστές μοιραζόμενης μνήμης, clusters, συστήματα κατανεμημένης μνήμης κ.α. Στο σύστημα PVM μπορεί να εκτελεσθεί χρονοδρομολόγηση εργασιών, δρομολόγηση μηνυμάτων στο δίκτυο, ισοστάθμιση φορτίου επεξεργαστών και τον μηχανισμό συγχρονισμού των barriers. Δεν διατίθεται μηχανισμός δυναμικής δημιουργίας διακλαδώσεων (threads), αλλά ούτε κάποιο σύστημα κοινής κατανεμημένης μνήμης. Υποστηρίζει, μόνο το προγραμματιστικό μοντέλο ανταλλαγής μηνυμάτων.



## 9.3. Σύστημα παράλληλου προγραμματισμού Chare Kernel

Η γλώσσα και το σύστημα παράλληλου προγραμματισμού Chare Kernel αναπτύχθηκαν στο πανεπιστήμιο Urbana του Ιλινόις. Προσφέρουν δυνατότητα ανάπτυξης παραλλήλων εφαρμογών τα οποία μπορούν να "τρέξουν" σε οποιαδήποτε παράλληλη μηχανή κατανεμημένης ή μοιραζόμενης. Στο σύστημα Chare Kernel είναι ενσωματωμένο ένα σύστημα εικονικής μοιραζόμενης μνήμης για υπολογιστές κατανεμημένης μνήμης. Η μετάβαση από/και στα συστήματα μοιραζόμενης ή κατανεμημένης μνήμης πραγματοποιείται διαφανώς για το χρήστη. Η γλώσσα Chare Kernel αποτελεί υπερσύνολο της γλώσσας C, χωρίς όμως τις καθολικές μεταβλητές της. Το chare (στοιχειώδης μονάδα λογισμικού) είναι οντότητα η οποία μοιάζει με τη διεργασία, το monitor ή ένα αντικείμενο παραλληλισμού, εκτελώντας κάποιους υπολογισμούς. Αυτή η οντότητα αποτελείται από δηλώσεις τοπικών μεταβλητών, δηλώσεις τύπων μηνυμάτων που ανταλλάζει, ένα block κώδικα C και τον κώδικα συναρτήσεων που διαχειρίζεται αυτά τα μηνύματα. Ένα πρόγραμμα που έχει υλοποιηθεί σε γλώσσα του Chare Kernel αποτελείται από δηλώσεις ενός συνόλου chares και δηλώσεις συνόλου συναρτήσεων (όπως και την C).

## 9.4. Σύστημα F-Code

Το σύστημα F-Code αποτελεί ενδιάμεση γλώσσα χαμηλού επιπέδου, η οποία δύναται για ανάπτυξη μεταγλωττιστών παραλληλοποίησης σε οποιαδήποτε γλώσσα υψηλού επιπέδου. Υποστηρίζει συναρτησιακό παραλληλισμό, υιοθετώντας σε μεγάλο ποσοστό τη σημασιολογία της γλώσσας Lisp. Παρ' όλα αυτά δεν είναι γλώσσα συναρτησιακού προγραμματισμού. Με το σύστημα F-Code μπορεί να πραγματοποιηθεί ο παραλληλισμός δεδομένων σε μη βαθμωτά αντικείμενα. Για παράδειγμα, η πράξη άθροισης δύο πινάκων δεν μεταφράζεται σε ένα σειριακό βρόγχο, αλλά εκτελείται συγχρόνως για όλα τα στοιχεία των πινάκων.

## 9.5. Πλατφόρμα Panda

Η Panda είναι μεταφέρσιμη πλατφόρμα, προσφέροντας ανάπτυξη συστημάτων χρόνου εκτέλεσης σε γλώσσες παράλληλου προγραμματισμού, threads, RPC κ.α.. Στο σύστημα Panda υπάρχουν πολλά επίπεδα, αλλά μόνο το πιο χαμηλό εξαρτάται άμεσα από την παράλληλη μηχανή και το λειτουργικό της σύστημα. Για τη μεταφορά του Panda σε μια νέα πλατφόρμα απαιτείται μόνο το χαμηλότερο επίπεδο.

## 9.6. Γλώσσα παράλληλου προγραμματισμού ORCA

Η υλοποίηση της γλώσσας παράλληλου προγραμματισμού ORCA έχει γίνει με τη βοήθεια μεταφέρσιμης πλατφόρμας παράλληλων εφαρμογών Panda. Η σημασιολογία και οι δομές της γλώσσας ORCA έχουν κοινά χαρακτηριστικά με τη γλώσσα C ή τη Modula 2, πλεονεκτεί δε στο ότι επιτρέπει την ύπαρξη παραλλήλων διεργασιών και μοιραζόμενων αντικειμένων (αντικείμενα δεδομένων τα οποία χρησιμοποιούνται από διεργασίες) που ενδεχομένως "τρέχουν" σε διαφορετικούς επεξεργαστές. Η ORCA διαχειρίζεται και τοποθετεί τα δεδομένα σε κάποια ή κάποιες τοπικές μνήμες. Για παράδειγμα στη γλώσσα ORCA υλοποιείται ένα σύστημα κατανεμημένης κοινόχρηστης μνήμης με δυνατότητες μετανάστευσης αντικειμένων, πολλαπλών αντιτύπων κ.α. Ο χρήστης μπορεί να ορίζει τα μοιραζόμενα δεδομένα που προσπελαύνονται από συναρτήσεις.

## 9.7. MPI

Το MPI (Message Passing Interface) είναι μία βιβλιοθήκη συναρτήσεων (σε C) ή υπορουτινών (σε Fortran) για τη συγγραφή παράλληλων προγραμμάτων με πέρασμα μηνυμάτων σε C ή FORTRAN σε μια μεγάλη ποικιλία συστημάτων. Περιλαμβάνει έναν μεγάλο αριθμό συναρτήσεων, οι οποίες διαχειρίζονται την επικοινωνία μεταξύ των επεξεργαστών/διεργασιών. Το MPI εξασφαλίζει μεταφερσιμότητα του πηγαίου κώδικα και επιτρέπει αποδοτική υλοποίηση σε διαφορετικές παράλληλες αρχιτεκτονικές. Είναι αποτέλεσμα συλλογικής προσπάθειας καθορισμού ενός πρότυπου πυρήνα μιας βιβλιοθήκης ανταλλαγής μηνυμάτων. Κύριο πλεονέκτημα στην επικράτηση ενός τέτοιου στάνταρτ MPI αποτελεί η μεταφερσιμότητα εφαρμογών ανταλλαγής μηνυμάτων. Οι κατασκευαστές παραλλήλων υπολογιστών με το MPI έχουν ένα εργαλείο με σαφώς ορισμένο βασικό σύνολο συναρτήσεων, το οποίο είναι αρκετό για την ανάπτυξη παράλληλων εφαρμογών το οποίο μπορεί εύκολα να υλοποιηθεί με λογισμικό ή hardware. Παρ' όλα αυτά, το MPI ορίζει μόνο ένα σύστημα επικοινωνιών, χωρίς όμως να περιέχει ένα πλήρες σύνολο ορισμών για συναρτήσεις υποστήριξης παράλληλων εφαρμογών.

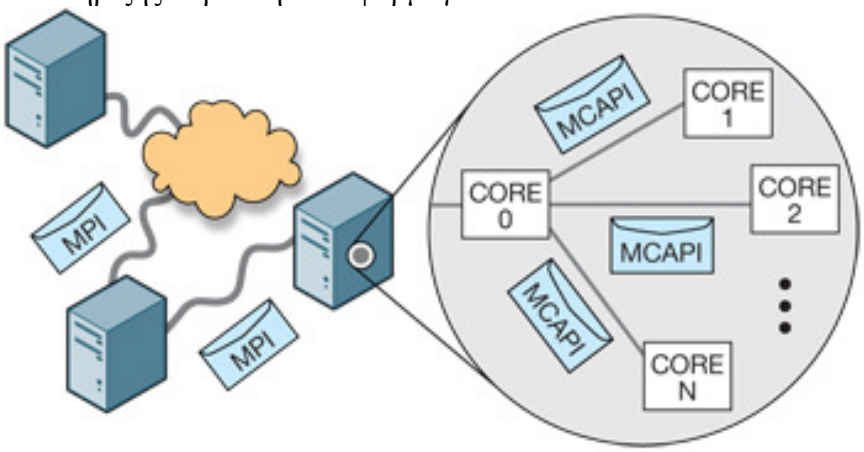



Figure 1 The accelerator cores run MCAPI instead of MPI, meaning that MPI messages run between the servers but MCAPI messages run between the cores in the server.

Ένα πρόγραμμα MPI αποτελείται από έναν αριθμό διεργασιών, οι οποίες εκτελούνται παράλληλα. Κάθε διεργασία χρησιμοποιεί το δικό της χώρο διευθύνσεων, χωρίς αυτό να σημαίνει ότι δεν υπάρχουν υλοποιήσεις του MPI για συστήματα με διαμοιραζόμενη μνήμη. Ο αριθμός των διεργασιών, οι οποίες δημιουργούνται σε ένα πρόγραμμα MPI, είναι σταθερός και καθορίζεται από τον προγραμματιστή, όταν εκτελεί την εφαρμογή. Κατά τη διάρκεια της εκτέλεσης δεν μπορεί να δημιουργήσει καινούργιες διεργασίες ή να καταστρέψει κάποια από τις υπάρχουσες. Δηλαδή, το πρόγραμμα πρέπει να εκτελεστεί από την αρχή, για να οριστεί ξανά ο καινούργιος αριθμός διεργασιών. Ωστόσο, νεώτερες υλοποιήσεις του MPI υποστηρίζουν τη δημιουργία και καταστροφή διεργασιών κατά το χρόνο εκτέλεσης (run-time process management).

Η μεταφορά δεδομένων από τις μεταβλητές μιας διεργασίας στις μεταβλητές μιας άλλης διεργασίας γίνεται με πέρασμα μηνυμάτων από τη μια διεργασία στην άλλη. Την αποστολή των μηνυμάτων την αναλαμβάνει το MPI. Ο προγραμματιστής καλεί τις συναρτήσεις αποστολής και παραλαβής μηνυμάτων μέσα από το πρόγραμμά του, αλλά δεν ασχολείται με τις λεπτομέρειες της επικοινωνίας μεταξύ των διεργασιών.

Το πρότυπο MPI-1 ορίσθηκε την άνοιξη του 1994. Αξίζει να σημειωθεί ότι:

- Το πρότυπο καθορίζει τα ονόματα, τις παραμέτρους κλήσεις και τις εξόδους συναρτήσεων και υπορουτινών προκειμένου να καλούνται από Fortran και C κώδικα αντίστοιχα. Όλες οι υλοποιήσεις του MPI πρέπει να υπακούουν σε αυτούς τους κανόνες, προκειμένου να εξασφαλίζεται η μεταφερσιμότητα (portability).
- Η λεπτομερής υλοποίηση της βιβλιοθήκης αφήνεται στους κατασκευαστές, οι οποίοι είναι έτσι ελεύθεροι να παράγουν βέλτιστες εκδόσεις (optimized versions) για τις μηχανές τους.
- Το πρότυπο έχει γίνει αποδεκτό από πολλούς φορείς και ερευνητές, τόσο στη βιομηχανία των υπολογιστών όσο και στα πανεπιστήμια. Υλοποιήσεις του υπάρχουν διαθέσιμες για μια μεγάλη ποικιλία από πλατφόρμες.

Οι ρουτίνες του MPI παρέχουν στον προγραμματιστή μια συνεπή προγραμματιστική διασύνδεση σε μια μεγάλη ποικιλία συστημάτων. Ο τρόπος που θα γίνει η ανάθεση των διεργασιών στους επεξεργαστές, δεν ορίζεται από το πρότυπο του MPI, διότι εξαρτάται από το συγκεκριμένο σύστημα, που χρησιμοποιείται. Κάθε υλοποίηση παρέχει τη δική της μέθοδο για την κατανομή των διεργασιών. Ωστόσο, σ' όλες τις υλοποιήσεις παρέχεται η συνάρτηση MPI_Init, η οποία ενεργοποιεί το MPI. Μέσω αυτής της συνάρτησης, η αρχικοποίηση του MPI καθώς και η αρχική κατανομή των διεργασιών στους επεξεργαστές, γίνονται από τη μεριά του προγραμματιστή, με τον ίδιο τρόπο

Το MPI εφαρμόζεται και σε αρχιτεκτονικές μοιραζόμενης μνήμης, όπου η μεταβίβαση μηνυμάτων υλοποιείται μέσω κοινής μνήμης, μια τεχνική που θυμίζει τα message queues του μοντέλου διεργασιών. Με την έννοια αυτή το μοντέλο μεταβίβασης μηνυμάτων είναι λογικά ισοδύναμο με το μοντέλο νημάτων, επιτρέποντας τη κοινή αντιμετώπιση προγραμματισμού συστημάτων Μοιραζόμενης και Κατανεμημένης Μνήμης. Παρ' όλα αυτά, το MPI διαχειρίζεται μόνο διεργασίες, ενώ σε περιπτώσεις που απαιτείται διαχείριση νημάτων η χρήση αντίστοιχου εργαλείου (όπως το OpenMP αποτελεί προϋπόθεση).

## 9.8. Επικοινωνία κόμβο με κόμβο

Ο βασικός μηχανισμός επικοινωνίας του MPI είναι η μετάδοση δεδομένων μηνυμάτων) μεταξύ δυο διεργασιών (η μία στέλνει και η άλλη λαμβάνει). Το MPI παρέχει ένα σύνολο συναρτήσεων για τη μετάδοση και λήψη ενός συγκεκριμένου τύπου δεδομένων. Σε μια επικοινωνία μπορούν να εμπλέκονται δυο ή και περισσότερες διεργασίες. Όταν εμπλέκονται μόνο δύο διεργασίες, λέμε ότι έχουμε μία επικοινωνία μεταξύ δύο σημείων ή επικοινωνία κόμβου με κόμβο (Point to Point Communication). Όταν εμπλέκονται περισσότερες από δύο διεργασίες, λέμε ότι έχουμε μια συλλογική επικοινωνία (collective communication). Το MPI παρέχει διαφορετικά είδη συναρτήσεων, για κάθε είδος επικοινωνίας.

### 9.8.1. Μηνύματα

Κάθε MPI-μήνυμα αποτελείται από δύο μέρη: από τον φάκελο (envelope) και από το κυρίως μήνυμα (message body). Ο φάκελος ενός MPI-μηνύματος περιλαμβάνει την εξής πληροφορία:

1. Το όνομα της διεργασίας-αφετηρίας, από την οποία στέλνεται το μήνυμα.
2. Το όνομα της διεργασίας-προορισμού, στην οποία κατευθύνεται το μήνυμα.
3. Τον προορισμό, στον οποίον ανήκουν οι διεργασίες αφετηρίας και προορισμού.
4. Μία ετικέτα, η οποία μπορεί να χρησιμοποιηθεί προκειμένου η διεργασία-παραλήπτης να ξεχωρίσει το είδος του μηνύματος. Έτσι, αν ληφθούν δύο μηνύματα από την ίδια διεργασία, με τη χρήση της ετικέτας, ο παραλήπτης μπορεί να τα διακρίνει.

Το κυρίως μήνυμα αποτελείται από την εξής πληροφορία:
1. Τη διεύθυνση αρχής της ενδιάμεσης μνήμης, όπου μπορούν να βρεθούν τα προς αποστολή δεδομένα στην περίπτωση μιας λειτουργίας αποστολής μηνύματος ή πρόκειται να αποθηκευθούν τα προς παραλαβή δεδομένα στην περίπτωση λειτουργίας παραλαβής μηνύματος.
2. Τον τύπο των δεδομένων του μηνύματος. Στις απλές περιπτώσεις, αυτός μπορεί να είναι ένας βασικός τύπος δεδομένων της C (int, float, κ.λπ.). Σε άλλες περιπτώσεις, μπορεί να είναι ένας τύπος δεδομένων ο οποίος ορίσθηκε από το χρήστη με τη χρήση των βασικών τύπων. Ένας τέτοιος τύπος δεδομένων στην περίπτωση, για παράδειγμα, της C, μπορεί να είναι ανάλογος με μία δομή της γλώσσας και να περιέχει δεδομένα, τα οποία δεν βρίσκονται αναγκαστικά σε διαδοχικές θέσεις μνήμης.
3. Τον αριθμό των αντικειμένων του τύπου δεδομένων, ο οποίος προσδιορίσθηκε προηγουμένως, που πρόκειται να αποσταλούν/ παραληφθούν.

### 9.8.2. Επικοινωνία κόμβου με κόμβο

Η επικοινωνία κόμβου με κόμβο είναι η απλούστερη μορφή επικοινωνίας διεργασιών, που διαθέτει το MPI. Όπως έχει ήδη αναφερθεί, στην επικοινωνία κόμβου με κόμβο εμπλέκονται μόνο δύο διεργασίες. Η διεργασία-αποστολέας στέλνει ένα μήνυμα στη διεργασία-παραλήπτη. Για να σταλεί ένα μήνυμα, η διεργασία-αποστολέας καλεί κάποια συνάρτηση αποστολής, παράμετροι της οποίας είναι η τάξη (rank) της διεργασίας-παραλήπτη (destination), καθώς και ο προορισμός (communicator) στον οποίο ανήκει η διεργασία-παραλήπτης.

Το MPI υποστηρίζει τέσσερις καταστάσεις επικοινωνίας, οι οποίες ορίζουν κριτήρια προκειμένου να προσδιορισθεί το πότε μια διαδικασία επικοινωνίας (δηλαδή, μια διαδικασία αποστολής ή μια διαδικασία παραλαβής) έχει ολοκληρωθεί. Οι καταστάσεις επικοινωνίας θα συζητηθούν στο επόμενο κεφάλαιο.

Οι υλοποιήσεις του MPI βασίζονται στους εξής κανόνες για τη σωστή διεξαγωγή της επικοινωνίας κόμβου με κόμβο.

- Διατήρηση της σειράς των μηνυμάτων.

Έστω, ότι έχουμε δύο MPI διεργασίες. Η διεργασία Α στέλνει δυο μηνύματα στη διεργασία Β, η οποία βρίσκεται στον ίδιο communicator με την Α. Τότε, τα δύο μηνύματα θα φτάσουν εγγυημένα, με τη σειρά με την οποία εστάλησαν.

Ο κανόνας ισχύει και όταν αποστέλλονται περισσότερα από δύο μηνύματα. Πάντα, θα λαμβάνονται με τη σειρά αποστολής τους.

- Αν έχουμε ένα send και ένα αντίστοιχο receive, τότε τουλάχιστον το ένα από τα δυο κάποτε ολοκληρώνεται.

Δηλαδή, αν έχουμε ένα send και ένα receive, δεν είναι δυνατόν η ολοκλήρωση και των δύο να αναβάλλεται συνεχώς, διότι τότε, η εφαρμογή οδηγείται σε αδιέξοδο (deadlock). Αν κάποια από τις δύο κλήσεις αναγκαστεί να περιμένει, συνήθως συμβαίνει ένα από τα παρακάτω.

- Το send λαμβάνεται από μια τρίτη διεργασία, η οποία έτυχε να εκτελεί ένα αντίστοιχο receive. Σ' αυτή την περίπτωση, το send ολοκληρώνεται, αλλά το receive της δεύτερης διεργασίας όχι.
- Μια τρίτη διεργασία στέλνει ένα μήνυμα, το οποίο λαμβάνεται από τη δεύτερη διεργασία. Σ' αυτή την περίπτωση, το receive ολοκληρώνεται, αλλά το send της πρώτης διεργασίας, όχι.

### 9.8.3. Παράδειγμα κώδικα με χρήση MPI

Το παρακάτω πρόγραμμα είναι γραμμένο σε MPI για το υπολογισμό των Πρώτων αριθμών.

```
# include <cstdlib>
# include <iostream>
# include <iomanip>
# include <cmath>
# include <ctime>

# include "mpi.h"

using namespace std;

int main ( int argc, char *argv[] );
int prime_number ( int n, int id, int p );
void timestamp ( );

//****************************************************************************

int main ( int argc, char *argv[] )

//****************************************************************************
/ /
/ /
/ /
/ / Το πρόγραμμα αυτό είναι μια έκδοση του PRIME_NUMBER (εύρεση πρώτων αριθμών)
που περιλαμβάνει
/ / MPI για παράλληλη επεξεργασία.
/ /
/ / Αδειοδότηση:
/ /
/ / Ο κωδικός αυτός διανέμεται υπό την άδεια χρήσης GNU LGPL.
/ /
/ /
/ / Συντάκτης:
/ /
/ / John Burkardt
/ /
{
  int i;
  int id;
  int n;
  int n_factor;
  int n_hi;
  int n_lo;
  int p;
  int primes;
```

```
  int primes_part;
  double wtime;

  n_lo = 1;
  n_hi = 262144;
  n_factor = 2;
//
//  Initialize MPI.
//
  MPI::Init ( argc, argv );
//
//  Η μεταβλητή p θα λάβει τον αριθμό των processes που μπορεί να χρησιμοποιήσει.
//
  p = MPI::COMM_WORLD.Get_size (  );
//
//  Determine this processes's rank.
//
  id = MPI::COMM_WORLD.Get_rank ( );

  if ( id == 0 )
  {
    timestamp ( );
    cout << "\n";
    cout << "PRIME_MPI\n";
    cout << "  C++/MPI version\n";
    cout << "\n";
    cout << "  An MPI example program to count the number of primes.\n";
    cout << "  The number of processes is " << p << "\n";
    cout << "\n";
    cout << "         N        Pi          Time\n";
    cout << "\n";
  }

  n = n_lo;

  while ( n <= n_hi )
  {
    if ( id == 0 )
    {
      wtime = MPI::Wtime ( );
    }
    MPI::COMM_WORLD.Bcast ( &n, 1, MPI::INT, 0 );

    primes_part = prime_number ( n, id, p );

    MPI::COMM_WORLD.Reduce ( &primes_part, &primes, 1, MPI::INT, MPI::SUM,
      0 );

    if ( id == 0 )
```

```
    {
      wtime = MPI::Wtime ( ) - wtime;

      cout << "  " << setw(8) << n
           << "  " << setw(8) << primes
           << "  " << setw(14) << wtime << "\n";
    }
    n = n * n_factor;
  }
//
//  Terminate MPI.
//
  MPI::Finalize ( );
//
//  Terminate.
//
  if ( id == 0 )
  {
    cout << "\n";
    cout << "PRIME_MPI - Master process:\n";
    cout << "  Normal end of execution.\n";
    cout << "\n";
    timestamp ( );
  }

  return 0;
}
//****************************************************************************
**80

int prime_number ( int n, int id, int p )

//****************************************************************************
**80
//
//  Purpose:
//
//PRIME_NUMBER επιστρέφει το πλήθος των πρώτων μεταξύ 1 και N.
//
//
//   Για να διαχωρίσουμε την εργασία ομοιομορφα στον συνολικό αριθμό επεξεργαστών το id
ξεκινά από 2+ID και προσπερνάτε ανά P
//
//    A native algorithm is used.
//
//    Mathematica can return the number of primes less than or equal to N
//    by the command PrimePi[N].
//
//                N  PRIME_NUMBER
```

```
//
//             1          0
//            10          4
//           100         25
//         1,000        168
//        10,000      1,229
//       100,000      9,592
//     1,000,000     78,498
//    10,000,000    664,579
//   100,000,000  5,761,455
//  1,000,000,000 50,847,534
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    21 May 2009
//
//  Author:
//
//    John Burkardt
//
//  Παράμετροι:
//
//    Input, int N, the maximum number to check.
//
//    Input, int ID, the ID of this process,
//    between 0 and P-1.
//
//    Input, int P, the number of processes.
//
//    Output, int PRIME_NUMBER, the number of prime numbers up to N.
//
{
  int i;
  int j;
  int prime;
  int total;

  total = 0;

  for ( i = 2 + id; i <= n; i = i + p )
  {
    prime = 1;
    for ( j = 2; j < i; j++ )
    {
      if ( ( i % j ) == 0 )
```

```
      {
        prime = 0;
        break;
      }
    }
    total = total + prime;
  }
  return total;
}
//****************************************************************************
**80

void timestamp ( )

//****************************************************************************
**80
//
//  Purpose:
//
//    TIMESTAMP prints the current YMDHMS date as a time stamp.
//
//  Example:
//
//    31 May 2001 09:45:54 AM
//
//  Licensing:
//
//    This code is distributed under the GNU LGPL license.
//
//  Modified:
//
//    24 September 2003
//
//  Author:
//
//    John Burkardt
//
//  Parameters:
//
//    None
//
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;
```

```
  now = time ( NULL );
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );

  cout << time_buffer << "\n";

  return;
# undef TIME_SIZE
}
```

## 9.9. Εγκατάσταση του LAM/MPI

Διαμόρφωση του LAM/MPI
Το πρώτο και πιο πολύπλοκο βήμα στην εγκατάσταση είναι η διαδικασία της διαμόρφωσης (configuration) . Αρκετές επιλογές του LAM/MPI μπορούν να τοποθετηθούν είτε κατά την διαμόρφωση είτε κατά την εκτέλεση του. Οι ρυθμίσεις κατά τη διαμόρφωση παρέχουν προκαθορισμένες τιμές (default values).
Η διανομή του LAM παρέχεται σε συμπιεσμένη μορφή είτε gzip είτε bzip2 και είναι διαθέσιμη από την κύρια ιστοσελίδα του LAM . Κατεβάζουμε το αρχείο lam-7.0.4.tar.gz ή lam-7.0.4.tar.bz2 και το αποσυμπιέζουμε για να εξαχθούν τα περιεχόμενα του με τις εντολές
tar zxf lam-7.0.4.tar.gz
ή
tar jxf lam-7.0.4.tar.bz2
Το LAM χρησιμοποιεί ένα GNU configure script για να φέρει σε πέρας τη διαδικασία της διαμόρφωσης . Αφού αποσυμπιέσουμε το αρχείο μεταφερόμαστε στον κατάλογο του LAM (lam-7.0.4) και εκτελούμε το configure script.
cd lam-7.0.4
./configure
Προεπιλεγμένα, το configure script τοποθετεί τον κατάλογο εγκατάστασης LAM στον γονικό από τον οποίο βρίσκεται η εντολή του LAM ,lamclean. Αυτό μπορεί μετατραπεί με την επιλογή - - prefix. Το configure script δημιουργεί αρκετά configuration files που χρησιμοποιούνται κατά τη διάρκεια του “build” .
Building LAM/MPI [10]
Αφού ολοκληρωθεί η διαμόρφωση ,τότε καθίσταται δυνατό το “χτίσιμο” του LAM .Αυτό γίνεται με την εντολή
make
στον κατάλογο που περιέχει το LAM .Η εντολή αυτή θα “κατασκευάσει” τα εκτελέσιμα αρχεία και τις βιβλιοθήκες του LAM , των οποίων η εγκατάσταση γίνεται με την εντολή
make install

# 10. MPI ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

## 10.1. Αρχικά Αρχικοποίηση MPI

Η πρώτη MPI ρουτίνα που καλείται σε οποιοδήποτε MPI πρόγραμμα πρέπει να είναι η ρουτίνα αρχικοποίησης MPI_INIT. Κάθε πρόγραμμα MPI πρέπει να καλέσει αυτήν την ρουτίνα μία φορά, πριν από οποιαδήποτε άλλη MPI ρουτίνα. Η εκτέλεση πολλαπλών κλήσεων της MPI_INIT είναι λανθασμένη.
Η ρουτίνα στη γλώσσα προγραμματισμού c δέχεται ως ορίσματα τα ορίσματα της main, argc και argv.
MPI::Init ( argc, argv );
Η MPI_INIT ορίζει τον MPI_COMM _WORLD για κάθε διεργασία που την καλεί. Το MPI_COMM_WORLD είναι ένας χώρος επικοινωνίας (communicator). Όλες οι κλήσεις επικοινωνίας MPI απαιτούν ένα όρισμα communicator και οι διεργασίες MPI μπορούν να επικοινωνήσουν μόνο εάν μοιράζονται έναν κοινό χώρο επικοινωνίας.
Κάθε χώρος επικοινωνίας περιέχει μία ομάδα (group) που είναι μία λίστα διεργασιών, υπολογιστών. Ο αριθμός της κάθε διεργασίας ονομάζεται τάξη (rank).
Η τάξη προσδιορίζει κάθε μία διεργασία στον χώρο επικοινωνίας. Η τάξη μπορεί να χρησιμοποιηθεί για να διευκρινιστεί η πηγή ή ο προορισμός ενός μηνύματος.

## 10.2. Τερματισμός MPI

Ένα MPI πρόγραμμα πρέπει να καλέσει την ρουτίνα MPI_FINALIZE όταν όλες οι επικοινωνίες έχουν ολοκληρωθεί. Αυτή η ρουτίνα καθαρίζει όλα τα δεδομένα του MPI . Είναι ευθύνη του προγραμματιστή να σιγουρευτείτε ότι όλες οι επικοινωνίες θα έχουν ολοκληρωθεί όταν κληθεί η ρουτίνα. Μόλις κληθεί αυτή η ρουτίνα, καμία άλλη κλήση δεν μπορεί να γίνει στις ρουτίνες MPI .
Όλα τα MPI προγράμματα πρέπει να περιλάβουν το αρχείο επικεφαλίδας (header file) που περιέχει τις απαιτούμενες συναρτήσεις για το mpi. Για προγράμματα στη C το header file είναι mpi.h . Λαμβάνοντας υπόψη τις
προηγούμενες δύο παραγράφους, βγαίνει το συμπέρασμα ότι κάθε MPI πρόγραμμα πρέπει να έχει περιληπτικά την ακόλουθη δομή.

```
#include <mpi.h>
/* περίληψη και των συνηθισμένων header files */
int main ( int argc, char *argv[] );
int main ( int argc, char *argv[] ){
MPI::Init ( argc, argv );   /* αρχικοποίηση MPI */
/* κύριο πρόγραμμα */
MPI::Finalize ( );   /* τερματισμός MPI */
return (0);
}
```

## 10.3. Πρόσβαση στις πληροφορίες του communicator

Μια διεργασία MPI μπορεί να ζητήσει πληροφορίες από τον χώρο επικοινωνίας για την ομάδα, με τις εντολές MPI_COMM_SIZE και
MPI_COMM_RANK.
MPI::COMM_WORLD.Get_rank ( )
Η παραπάνω εντολή επιστρέφει την τάξη της διεργασίας . MPI::COMM_WORLD.Get_size ( )
Η παραπάνω εντολή επιστρέφει τον αριθμό των διεργασιών της ομάδας .
5.3 Τα μηνύματα
Η επικοινωνία μεταξύ των διεργασιών ενός προγράμματος mpi επιτυγχάνεται με την ανταλλαγή μηνυμάτων μεταξύ των διεργασιών .Ένα MPI μήνυμα είναι μια σειρά στοιχείων ενός τύπου δεδομένων MPI (datatype).Όλα τα μηνύματα MPI δημιουργούνται με τη λογική ότι ο τύπος του περιεχομένου πρέπει να διευκρινιστεί κατά την αποστολή και τη λήψη . Οι βασικοί τύποι δεδομένων στο MPI αντιστοιχούν στους βασικούς τύπους δεδομένων της C όπως φαίνεται στον παρακάτω πίνακα.
Πίνακας : Τύποι δεδομένων στο MPI
MPI Datatype C datatype
MPI :: CHAR signed char
MPI :: SHORT signed short int
MPI :: INT signed int
MPI :: LONG signed long int
MPI :: UNSIGNED_CHAR unsigned char
MPI :: UNSIGNED_SHORT unsigned short int
MPI :: UNSIGNED unsigned int
MPI :: UNSIGNED_LONG unsigned long int
MPI :: FLOAT float
MPI :: DOUBLE double
MPI :: LONG_DOUBLE long double
MPI :: BYTE
MPI :: PACKED
Ο τύπος δεδομένων που καθορίζεται στη λήψη του μηνύματος πρέπει να ταιριάζει με τον τύπο δεδομένων που καθορίζεται στην αποστολή. Το μεγάλο πλεονέκτημα αυτού είναι ότι το MPI μπορεί να υποστηρίξει ετερογενείς παράλληλες αρχιτεκτονικές π.χ. παράλληλα συστήματα που χτίζονται από τους διαφορετικούς επεξεργαστές, επειδή η μετατροπή τύπων μπορεί να εκτελεσθεί όταν χρειάζεται. Κατά συνέπεια δύο επεξεργαστές μπορούν να εκφράζουν, για παράδειγμα, έναν ακέραιο αριθμό με διαφορετικούς τρόπους, αλλά οι MPI διεργασίες σε αυτούς τους επεξεργαστές μπορεί να χρησιμοποιήσουν το MPI για να στείλουν τα μηνύματα ακέραιων αριθμών χωρίς να είναι ενήμερα για την ετερογένεια. Πιο σύνθετοι τύποι δεδομένων μπορούν να κατασκευαστούν κατά την δημιουργία των προγραμμάτων.

## 10.4. Επικοινωνία

Οι τύποι με τους οποίους μπορούν να επικοινωνούν δύο διεργασίες είναι είτε άκρο με άκρο (point- to-point ) είτε συλλογικά . Μία point-to-point επικοινωνία προϋποθέτει πάντα ακριβώς δύο διεργασίες ,από τις οποίες η μια διεργασία στέλνει ένα μήνυμα στην άλλη διεργασία . Αυτό τη διαχωρίζει από τον άλλο τύπο επικοινωνίας του MPI, την συλλογική (collective)

επικοινωνία, κατά την εκτέλεση της οποίας η αποστολή και λήψη περιλαμβάνει μια ολόκληρη ομάδα διεργασιών συγχρόνως.
Για να στείλει ένα μήνυμα μία διεργασία κάνει MPI κλήση η οποία καθορίζει την διεργασία προορισμού από την τάξη στον κατάλληλο χώρο επικοινωνίας . Η διεργασία προορισμού επίσης πρέπει να κάνει MPI κλήση για να λάβει το μήνυμα.
Point-to-Point
Επικοινωνία με εμπόδια (blocking)
Κύρια αποστολή (standard send)
Η standard send έχει την ακόλουθη μορφή
MPI_SEND (buf, count, datatype, dest, tag, comm)
όπου

- buf είναι η διεύθυνση των δεδομένων προς αποστολή.
- count είναι ο πλήθος των δεδομένων προς αποστολή.
- datatype είναι ο τύπος δεδομένων MPI.
- dest είναι η διεργασία για την οποία προορίζεται το μήνυμα. Αυτό διευκρινίζεται από την τάξη της προοριζόμενης διεργασίας μέσα στην ομάδα του χώρου επικοινωνίας.
- tag είναι μία ετικέτα που χρησιμοποιείται από τον αποστολέα για τη διάκριση των διαφόρων τύπων μηνυμάτων.
- comm είναι ο χώρος επικοινωνίας που χρησιμοποιούν διεργασίες αποστολής και λήψης.

Ο απομονωτής (buffer) του μηνύματος περιγράφεται από τα ορίσματα buf , count και datatype . "Ολοκλήρωση" μιας αποστολής σημαίνει εξ ορισμού ότι τα δεδομένα έχουν παραδοθεί στο σύστημα και ο απομονωτής μπορεί να επαναχρησιμοποιηθεί ασφαλώς . Η κύρια αποστολή επιστρέφει αμέσως τον έλεγχο όταν ολοκληρωθεί ,το οποιο σημαίνει ότι το μήνυμα είτε μπορεί να έχει φθάσει στον προορισμό του είτε όχι .Σύγχρονη αποστολή(synchronous communication)
Εάν η αποστέλουσα διεργασία πρέπει να ξέρει ότι το μήνυμα έχει παραληφθεί από τη λαμβάνουσα διεργασία, τότε οι δύο διαδικασίες πρέπει να χρησιμοποιήσουν σύγχρονη επικοινωνία . Αυτό που συμβαίνει κατά τη διάρκεια της σύγχρονης επικοινωνίας είναι: η λαμβάνουσα διεργασία στέλνει πίσω μια αναγνώριση (μια διαδικασία γνωστή ως ‘χειραψία’ μεταξύ των διεργασιών) . Αυτή η αναγνώριση πρέπει να παραληφθεί από τον αποστολέα προτού η αποστολή θεωρείται πλήρης .
Η ρουτίνα MPI σύγχρονης αποστολής είναι παρόμοια σε μορφή με την κύρια αποστολή.
MPI_SSEND (buf, count, datatype, dest, tag, comm)
Εάν μια διεργασία που εκτελεί μία σύγχρονη , με εμπόδια αποστολή είναι "μπροστά" της διεργασίας που εκτελεί την ταιριαστή λήψη, τότε θα είναι αδρανής έως ότου προφθάσει η λαμβάνουσα διεργασία . Η σύγχρονη επικοινωνία είναι επομένως πιο αργή από την κύρια επικοινωνία . Η σύγχρονη επικοινωνία είναι εντούτοις ασφαλέστερη μέθοδος επικοινωνίας επειδή το δίκτυο επικοινωνίας δεν μπορεί ποτέ να υπερφορτωθεί με μη παραδομένα μηνύματα.
Αποστολή με χρήση απομονωτή (buffered send)
Η αποστολή με χρήση απομονωτή ολοκληρώνεται αμέσως , αντιγράφοντας το μήνυμα σε ένα απομονωτή του συστήματος για αργότερη μετάδοση εάν είναι απαραίτητο .Μειονέκτημα της αποστολής με χρήση απομονωτή είναι ότι ο προγραμματιστής δεν μπορεί να προϋποθέσει οποιοιδήποτε προ-διατιθέμενο χώρο απομονωτή (buffer space) και πρέπει ρητά να συνάπτει αρκετό χώρο για το πρόγραμμα με κλήσεις της MPI_BUFFER_ATTACH.
MPI_BUFFER_ATTACH (buffer, size)
Το Buffer space αποεπισυνάπτεται με τη ρουτίνα
MPI_BUFFER_DETACH (buffer, size)

Έτοιμη αποστολή (Ready Send)
Η έτοιμη αποστολή , όπως η αποστολή με χρήση απομονωτή , επιστρέφει αμέσως. Η επικοινωνία είναι εγγυημένη να πετύχει κανονικά εάν μία ταιριαστή λήψη έχει ήδη απαιτηθεί. Εντούτοις, αντίθετα από τις άλλες αποστολές, εάν καμία ταιριαστή λήψη δεν έχει απαιτηθεί, η έκβαση είναι απροσδιόριστη. Η αποστέλουσα διεργασία " ρίχνει" απλά το μήνυμα στο δίκτυο επικοινωνίας και ελπίζει ότι η λαμβάνουσα διεργασία περιμένει να το "πιάσει" .Εάν η λαμβάνουσα διεργασία είναι έτοιμη για το μήνυμα, θα παραληφθεί, αλλιώς το μήνυμα μπορεί σιωπηλά να "πέσει", ένα λάθος μπορεί να εμφανιστεί, κ.λ.π....
Το θετικό με την αποφυγή της ανάγκης για χειραψία ή για χρήση απομονωτή μεταξύ του αποστολέα και του δέκτη είναι η βελτίωση της απόδοσης. Η χρήση της έτοιμου τρόπου επικοινωνίας είναι ασφαλής μόνο εάν η λογική ροή του παράλληλου προγράμματος το επιτρέπει.
Η ready send έχει παρόμοια μορφή με τη standard send:
MPI_RSEND (buf, count, datatype, dest, tag, comm)
Λήψη με εμπόδια (blocking receive)
Η μορφή της λήψης με εμπόδια είναι:
MPI_RECV (buf, count, datatype, source, tag, comm, status)
Όπου

- buf είναι η διεύθυνση που τα δεδομένα πρέπει να τοποθετηθούν μόλις παραληφθούν
- count είναι το πλήθος των δεδομένων προς λήψη
- datatype είναι ο τύπος δεδομένων MPI. Αυτός πρέπει να ταιριάζει με τον τύπο δεδομένων που καθορίζεται στη ρουτίνα αποστολής .
- source είναι η τάξη της πηγής του μηνύματος στην ομάδα . Αντί του ορισμού της πηγής, τα μηνύματα μπορούν να παραληφθούν από οποιαδήποτε πηγή με τη χρησιμοποίηση ενός ορίσματος μπαλαντέρ,

MPI_ANY_SOURCE

- tag χρησιμοποιείται από τη λαμβάνουσα διαδικασία για να ορίσει ότι πρέπει να λάβει μήνυμα με με το συγκεκριμένο tag. Αντί του ορισμού της ετικέτας, ο μπαλαντέρ

MPI_ANY_TAG μπορεί να τοποθετηθεί στη θέση αυτού του ορίσματος.

- comm είναι ο χώρος επικοινωνίας που χρησιμοποιούν διεργασίες αποστολής και λήψης.

Η ολοκλήρωση μιας λήψης σημαίνει εξ ορισμού ότι ένα μήνυμα έφθασε δηλαδή τα δεδομένα έχουν ληφθεί.
Πληροφορίες της επικοινωνίας
Η επικοινωνία περιλαμβάνει πληροφορίες οι οποίες μπορεί να χρησιμοποιηθούν για τη διάκριση μεταξύ των μηνυμάτων. Αυτές οι πληροφορίες επιστρέφονται από την MPI_RECV σαν ιδιότητα (status).
Το status όρισμα μπορεί να ρωτηθεί άμεσα για να αποκαλύψει την πηγή ή την ετικέτα ενός μηνύματος το οποίο μόλις παραλήφθηκε. Αυτό είναι απαραίτητο μόνο εάν ένας μπαλαντέρ χρησιμοποιήθηκε σε ένα από τα ορίσματα στην κλήση λήψης . Η πηγαία διεργασία ενός μηνύματος που παραλαμβάνεται με τη χρήση του μπαλαντέρ MPI_ANY_SOURCE ως όρισμα μπορεί να βρεθεί για τη C με την εντολή
status.MPI_SOURCE
Παρόμοια, η ετικέτα από ένα μήνυμα που παραλαμβάνεται με τη χρήση του μπαλαντέρ MPI_ANY_TAG ως όρισμα μπορεί να βρεθεί στη C με:
status.MPI_TAG
Το μέγεθος του μηνύματος που παραλαμβάνεται από μια διεργασία μπορεί επίσης να βρεθεί. Για το σκοπό αυτό χρησιμοποιείται η ρουτίνα
MPI_GET_COUNT (status, datatype, count)

Επικοινωνία χωρίς εμπόδια
Οι επικοινωνίες που περιγράφθηκαν μέχρι τώρα είναι όλες επικοινωνίες με εμπόδια . Αυτό σημαίνει ότι δεν επιστρέφουν μέχρι η επικοινωνία να ολοκληρωθεί (υπό την έννοια ότι ο απομονωτής μπορεί να επαναχρησιμοποιηθεί).
Στην χωρίς εμπόδια επικοινωνία οι διεργασίες καλούν μία MPI ρουτίνα για να αρχίσει μία επικοινωνία (αποστολή ή λήψη) , αλλά η ρουτίνα επιστρέφει αμέσως . Η επικοινωνία μπορεί έπειτα να συνεχιστεί στο υπόβαθρο και η διεργασία μπορεί να συνεχίσει με άλλη εργασία .Η έτοιμη αποστολή καθώς και η αποστολή με απομονωτή δεν διαφέρουν είτε πρόκειται για επικοινωνία με εμπόδια είτε για επικοινωνία χωρίς εμπόδια .
Αποστολές χωρίς εμπόδια
Η διεργασία αποστολής αρχίζει την αποστολή με τη χρησιμοποίηση της ακόλουθης ρουτίνας (στη σύγχρονη μορφή):
MPI_ISSEND (buf, count, datatype, dest, tag, comm, request)
Οι χωρίς εμπόδια ρουτίνες έχουν τα ίδια ορίσματα με τις αντίστοιχες με εμπόδια εκτός από ένα πρόσθετο όρισμα. Αυτό το όρισμα , request, παρέχει ένα τρόπο που χρησιμοποιείται για να εξετάσει αν η επικοινωνία έχει ολοκληρωθεί. Αφού εκτελεστεί η ρουτίνα συνεχίζεται έπειτα η διεργασία με άλλους υπολογισμούς που δεν αλλάζουν τον απομονωτή αποστολή. Πριν η αποστέλουσα διεργασία ενημερώσει τον απομονωτή αποστολής πρέπει να ελέγξει αν η αποστολή έχει ολοκληρωθεί .
Ο έλεγχος ολοκλήρωσης γίνεται με δύο τρόπους:
• WAIT Αυτή η ρουτίνα εμποδίζει την εκτέλεση της διεργασίας έως ότου έχει ολοκληρωθεί η επικοινωνία. Αυτές είναι χρήσιμες όταν απαιτούνται τα δεδομένα από την επικοινωνία για υπολογισμούς ή όταν πρέπει ο απομονωτής επικοινωνίας να επαναχρησιμοποιηθεί . Επομένως μία επικοινωνία χωρίς εμπόδια αμέσως ακολουθούμενη από WAIT έλεγχο είναι ισοδύναμη με την αντιστοιχή επικοινωνία με εμπόδια .
Ο WAIT έλεγχος εκτελείται ως εξής:
MPI_WAIT (request, status)
και εμποδίζει την εκτέλεση της διεργασίας έως ότου η επικοινωνία που διευκρινίζεται από το request έχει ολοκληρωθεί.
• TEST Αυτές οι ρουτίνες επιστρέφουν TRUE ή FALSE τιμή ανάλογα με το εάν ή όχι έχει ολοκληρωθεί η επικοινωνία. Αυτές οι ρουτίνες δεν εμποδίζουν την εκτέλεση της διεργασίας και είναι χρήσιμες σε καταστάσεις που θέλουμε να ξέρουμε εάν η επικοινωνία έχει ολοκληρωθεί και δεν απαιτείται ακόμα το αποτέλεσμα ή η επαναχρησιμοποίηση του communication buffer
Ο έλεγχος test εκτελείται ως εξής
MPI_TEST (request, flag, status)
Λήψεις χωρίς εμπόδια
Η λαμβάνουσα διαδικασία ταχυδρομεί την ακόλουθη ρουτίνα λήψης για να αρχίσει η λήψη:
MPI_IRECV (buf, count, datatype, source, tag, comm, request)
Η λαμβάνουσα διεργασία μπορεί έπειτα να συνεχίσει με άλλους υπολογισμούς έως ότου χρειαστεί τα λαμβανόμενα δεδομένα. Ελέγχει έπειτα τον απομονωτή λήψης για να δει εάν η επικοινωνία έχει ολοκληρωθεί. Non-blocking λήψεις μπορεί να ταιριάξουν και με blocking αποστολές και αντίστροφα.
Συλλογική Επικοινωνία (Collective Communication)
Αυτό που διακρίνει την συλλογική επικοινωνία από την point-to-point επικοινωνία είναι ότι περιλαμβάνει όλες τις διεργασίες του καθορισμένου χώρου επικοινωνίας .
Τα χαρακτηριστικά της συλλογικής επικοινωνίας είναι:

- Η συλλογική επικοινωνία δεν μπορεί να παρέμβει σε point-to-point επικοινωνίες και αντίστροφα — συλλογική και point-to-point επικοινωνία είναι ανεξάρτητα το ένα με το άλλο. Δηλαδή, μία συλλογική αποστολή δεν μπορεί να επιλεχτεί από μία point-to-point λήψη .
- Όλες οι διεργασίες στον χώρο επικοινωνίας πρέπει να καλέσουν την συλλογική επικοινωνία.
- Ομοιότητες με την point-to-point communication είναι:
- Ένα μήνυμα είναι μια σειρά ενός τύπου δεδομένων
- Οι τύποι δεδομένων πρέπει να ταιριάζουν μεταξύ της λήψης και της αποστολής

MPI_BCAST

Μία broadcast έχει καθορισμένη μία κύρια διεργασία και κάθε διεργασία λαμβάνει ανά αντίγραφο από το μήνυμα της κύριας διεργασίας .

MPI_BCAST (buffer, count, datatype, root, comm)

Το root όρισμα είναι η τάξη της κύριας διεργασίας. Τα ορίσματα buffer, count και datatype αντιμετωπίζονται όπως στην point-to-point επικοινωνία .

MPI_SCATTER

Για τη ρουτίνα αυτή πρέπει επίσης να καθοριστεί μία κύρια διεργασία . Κατά την εκτέλεση της SCATTER κάθε διεργασία λαμβάνει ένα μέρος από το μήνυμα της κύριας διεργασίας .

MPI_SCATTER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)

Για την MPI_SCATTER, τα sendbuf, sendcount, sendtype ορίσματα αναφέρονται στην κύρια διεργασία και τα recvbuf, recvcount, recvtype στις υπόλοιπες διεργασίες .

MPI_GATHER

Για τη ρουτίνα αυτή πρέπει επίσης να καθοριστεί μία κύρια διεργασία . Κατά την εκτέλεση της GATHER κάθε διεργασία αποστέλλει τα δεδομένα της στη κύρια διεργασία.

MPI_GATHER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)

Για την MPI_GATHER, τα sendbuf, sendcount, sendtype ορίσματα αναφέρονται σε όλες τις διεργασίες και τα recvbuf, recvcount, recvtype στην κύρια διεργασία .

MPI_ALLGATHER

Η ρουτίνα αυτή λειτουργεί όπως η GATHER με τη διαφορά ότι τα δεδομένα αποστέλλονται σε όλες τις διεργασίες .

MPI_ALLGATHER (sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)

## 11. Υλοποίηση παράλληλου προγράμματος

Για να καταστεί φανερή η χρησιμότητα της συστοιχίας που δημιουργήθηκε στο εργαστήριο κρίθηκε αναγκαία η επίλυση ενός συγκεκριμένου προβλήματος με τη βοήθεια τόσο σειριακού όσο και παράλληλου προγραμματισμού. Το πρόβλημα το οποίο επιλέχθηκε να είναι η αναζήτηση της θέσης ενός στοιχείου πίνακα. Το πρόγραμμα εκτελεί την αναζήτηση και επιστρέφει τις θέσεις του πίνακα στις οποίες υπάρχει το αναζητούμενο στοιχείο αλλά και το συνολικό χρονικό διάστημα που απαιτείται για να ολοκληρωθεί η εργασία.

Στη συνέχεια ακολουθεί ο πηγαίος κώδικας του σειριακού προγράμματος σε γλώσσα προγραμματισμού C++.

```
#       include <cstdlib>
#       include <iostream.h>
#       include <time.h>
int main ( int argc, char *argv[] );
//*****************************************************
int main ( int argc, char *argv[] )
//*****************************************************
```

```
//
//        Σκοπός:
//
//        Εύρεση του χρόνου κατά την αναζήτηση στοιχείου σε
//        πίνακα
//
{ clock_t time,deltime; float secs;
time = clock();
int *a; int i; int n;
int target;
n = 3000000;  //Πλήθος στοιχείων
a = new int[n];
for ( i = 0; i < n; i++ )
{
a[i] = rand();
}
target = a[n/2];
for ( i = 0; i < n; i++ )
{
if ( a[i] == target )
{
cout << i << " " << a[i] << "\n";
}
}
delete [] ( a );
deltime = clock() - time;
secs = (float) deltime/CLOCKS_PER_SEC; cout << " #tics = " << deltime ;
cout << " time = " << secs << " secs ";
return 0;
}
```

Το παραπάνω πρόγραμμα αποτελεί μια απλή υλοποίηση αλγορίθμου αναζήτησης. Το μόνο λεπτό σημείο το οποίο θα σχολιάσουμε είναι η χρήση του χρονομετρητή (timer). Στο τμήμα δήλωσης μεταβλητών με την εντολή clock_t time,deltime; δημιουργούμε τα αντικείμενα time και deltime. Έπειτα δηλώνουμε τη μεταβλητή secs τύπου float. Τέλος με τη χρησιμοποίηση του ρολογιού του υπολογιστή καθορίζουμε τη μεταβλητή time με την εντολή time = clock(); . Μεταφερόμαστε μετά στο τέλος του προγράμματος όπου μετά την ολοκλήρωση της αναζήτησης η μεταβλητή deltime υπολογίζεται ως η αφαίρεση της τρέχουσας τιμής του ρολογιού του υπολογιστή πλην την αρχική, που έχει αποθηκευτεί στη μεταβλητή time.Ο χρόνος σε δευτερόλεπτα υπολογίζεται διαιρώντας την τιμή της μεταβλητής deltime με έναν αριθμό που δείχνει τα clocks του υπολογιστή ανά δευτερόλεπτο.
Στη συνέχεια ακολουθεί το παράλληλο πρόγραμμα .

```
#       include <cstdlib>
#       include <iostream>
#       include <time.h>
using namespace std;
# include "mpi.h"
int main ( int argc, char *argv[] );
//*********************************************************
*********
```

```
int main ( int argc, char *argv[] )
//*************************************************************
*********
//
//        Σκοπός
//
//        Εύρεση του χρόνου κατά την αναζήτηση στοιχείου σε πίνακα
//
//        Λειτουργία:
//
//
//        Η κύρια διεργασία παράγει τον πίνακα και την τιμή που πρέπει να
βρεθεί.
//        Στη συνέχεια διανέμει τα στοιχεία σε ένα πλήθος άλλων διεργασιών
//        και κάθε μία από αυτές στέλνει πίσω το δείκτη του στοιχείου
{ clock_t time,deltime; float secs;
time = clock();
int *a; int dest; int global; int i;
int ierr;
int master = 0; int my_id;
int n; int npart;
int num_procs; int source;
int start; MPI::Status status; int tag;
int tag_target = 1; int tag_size = 2; int tag_data = 3; int tag_found = 4; int tag_done = 5; int
target;
int workers_done; int x;
//        Αρχικοποίηση MPI. MPI::Init ( argc, argv );
//        Παίρνεται η τάξη αυτής της διεργασίας.
ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &my_id );
// Εύρεση του αριθμού των διεργασιών
ierr = MPI_Comm_size ( MPI_COMM_WORLD, &num_procs );
if ( my_id == master )
{
cout << " Ο αριθμός των διεργασιών είναι " << num_procs << "\n";
}
cout << "\n";
cout << "Διεργασία " << my_id << " ενεργή \n";
if ( my_id == master )
{
npart = 100 ;
n = npart * num_procs; // Πλήθος των στοιχείων του πίνακα
a = new int[n];
for ( i = 0; i < n; i++ )
{
a[i] = rand ( ) ;    // Αρχικοποιείται ο πίνακας
}
target = a[n/2];
//        Στέλνονται στις δευτερεύουσες διεργασίες η τιμή που αναζητείται ,το πλήθος των
```

```
//        δεδομένων που θα τις σταλούν καθώς τα δεδομένα του πίνακα για τα οποία θα κάνουν
την αναζήτηση.
for ( i = 1; i <= num_procs-1; i++ )
{
dest = i;
tag = tag_target;
MPI::COMM_WORLD.Send ( &target, 1, MPI::INTEGER, dest, tag );
tag = tag_size;
MPI::COMM_WORLD.Send ( &npart, 1, MPI::INTEGER, dest, tag );
start = ( i - 1 ) * npart; tag = tag_data;
MPI::COMM_WORLD.Send ( a+start, npart, MPI::INTEGER, dest,
tag );
}
// Ο κύριος υπολογιστής περιμένει τα αποτελέσματα από τους δευτερεύοντες υπολογιστές
workers_done = 0;
while ( workers_done < num_procs-1 )
{
MPI::COMM_WORLD.Recv ( &x, 1, MPI::INTEGER, MPI::ANY_SOURCE,
MPI::ANY_TAG,
status );
source = status.Get_source ( ); tag = status.Get_tag ( );
if ( tag == tag_done )
{
workers_done = workers_done + 1;
}
else     {
cout << "P" << source << " " << x << " " << a[x] << "\n";
}
}
delete [] a;
}
//        Οι δευτερεύουσες διεργασίες δέχονται την τιμή που αναζητείται ,το πλήθος των
//        δεδομένων καθώς τα δεδομένα του πίνακα για τα οποία θα κάνουν την αναζήτηση.
else
{
source = master; tag = tag_target;
MPI::COMM_WORLD.Recv ( &target, 1, MPI::INT, source, tag, status
);
source = master; tag = tag_size;
MPI::COMM_WORLD.Recv ( &npart, 1, MPI::INT, source, tag, status
);
a = new int[npart];
source = master; tag = tag_data;
MPI::COMM_WORLD.Recv ( a, npart, MPI::INT, source, tag, status );
for ( i = 0; i < npart; i++ )
{
if ( a[i] == target )
{
global = ( my_id - 1 ) * npart + i; dest = master;
```

```
tag = tag_found;
MPI::COMM_WORLD.Send ( &global, 1, MPI::INT, dest, tag );
}
}
dest = master; tag = tag_done;
MPI::COMM_WORLD.Send ( &target, 1, MPI::INT, dest, tag );
delete [] ( a );
}
MPI::Finalize ( );
if ( my_id == master )
{
deltime = clock() - time;
secs = (float) deltime/CLOCKS_PER_SEC; cout << " #tics = " << deltime ;
cout << " time = " << secs << " secs "; }
return 0;
}
```

Το πρόγραμμα εκτελείται ταυτόχρονα και στους έξι υπολογιστές (έναν κύριο και πέντε δευτερεύοντες). Το πιο κομμάτι του προγράμματος εκτελεί ο κάθε υπολογιστής, που καθορίζεται από τον αν είναι κύριος ή δευτερεύον, εξαρτάται από την τιμή που έχει ο καθένας στην μεταβλητή του my_id που μπορεί να προσδιοριστεί με την εντολή

```
ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &my_id );
```

Η εντολή αυτή εκτελείται σε όλους τους υπολογιστές και κάθε ένας από αυτούς γνωρίζει ποια είναι η τάξη του (rank). Έτσι αν η τάξη του είναι 0 είναι ο κύριος ενώ σε κάθε άλλη περίπτωση είναι δευτερεύον.

Το πιο κομμάτι του προγράμματος θα εκτελεστεί σε κάθε υπολογιστή καθορίζεται από την εντολή if-else . π.χ

```
if ( my_id == master )
```

Από εδώ μπορούμε να κατανοήσουμε ότι το κομμάτι που βρίσκεται μέσα στο if θα εκτελεστεί μόνο στον κύριο υπολογιστή ενώ στο κομμάτι που σε παραπέμπει το else θα εκτελεστεί σε όλους τους δευτερεύοντες. Τα κομμάτια στα οποία δεν καθορίζεται με κάποιο if ποιος υπολογιστής πρέπει να τα εκτελέσει, εκτελούνται σε όλους.

Έτσι ουσιαστικά στον κύριο υπολογιστή εκτελείται το παρακάτω προγράμμα.

```
{ clock_t time,deltime; float secs;
time = clock();
int *a; int dest; int global; int i;
int ierr;
int master = 0; int my_id;
int n; int npart;
int num_procs; int source;
int start; MPI::Status status; int tag;
int tag_target = 1; int tag_size = 2; int tag_data = 3; int tag_found = 4; int tag_done = 5; int target;
int workers_done; int x;
// Αρχικοποίηση MPI.
MPI::Init ( argc, argv );
// Παίρνεται η τάξη αυτού του υπολογιστή (0 διότι είναι ο κύριος) ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &my_id );
// Εύρεση του αριθμού των υπολογιστών που μετέχουν στη εκτέλεση του προγράμματος.
```

```
ierr = MPI_Comm_size ( MPI_COMM_WORLD, &num_procs );
cout << " Ο αριθμός των διεργασιών είναι " << num_procs << "\n";
cout << "\n";
cout << "Διεργασία " << my_id << " ενεργή \n";
npart = 100 ;
n = npart * num_procs; // Πλήθος των στοιχείων του πίνακα
a = new int[n];
for ( i = 0; i < n; i++ )
{
a[i] = rand ( ) ;    // Αρχικοποιείται ο πίνακας και παίρνει τυχαίες τιμές
}
target = a[n/2];
// Στέλνονται στους δευτερεύοντες υπολογιστές η τιμή που αναζητείται, το πλήθος των δεδομένων που θα τους σταλούν καθώς τα δεδομένα του πίνακα για τα οποία θα κάνουν την αναζήτηση.
for ( i = 1; i <= num_procs-1; i++ )
{
dest = i;
tag = tag_target;
MPI::COMM_WORLD.Send ( &target, 1, MPI::INTEGER, dest, tag );
tag = tag_size;
MPI::COMM_WORLD.Send ( &npart, 1, MPI::INTEGER, dest, tag );
start = ( i - 1 ) * npart; tag = tag_data;
MPI::COMM_WORLD.Send ( a+start, npart, MPI::INTEGER, dest,tag );
// Ο κύριος υπολογιστής περιμένει τα αποτελέσματα από τους δευτερεύοντες υπολογιστές
workers_done = 0;
while ( workers_done < num_procs-1 )
{
MPI::COMM_WORLD.Recv ( &x, 1, MPI::INTEGER, MPI::ANY_SOURCE, MPI::ANY_TAG,status );
source = status.Get_source ( ); tag = status.Get_tag ( );
if ( tag == tag_done )
{
workers_done = workers_done + 1;
}
else
{
cout << "P" << source << " " << x << " " << a[x] << "\n";
}
}
delete [] a;
}
deltime = clock() - time;
secs = (float) deltime/CLOCKS_PER_SEC;
cout << " #tics = " << deltime ; cout << " time = " << secs << " secs ";
return 0;
}
```

Ουσιαστικά στους δευτερεύοντες υπολογιστές εκτελείται το παρακάτω πρόγραμμα.

```
{ clock_t time,deltime; float secs;
```

```
time = clock();
int *a; int dest; int global; int i;
int ierr;
int master = 0; int my_id;
int n; int npart;
int num_procs; int source;
int start; MPI::Status status; int tag;
int tag_target = 1; int tag_size = 2; int tag_data = 3; int tag_found = 4; int tag_done = 5; int target;
int workers_done; int x;
//        Αρχικοποίηση MPI. MPI::Init ( argc, argv );
//        Παίρνεται η τάξη αυτού του υπολογιστή (1,2,3,4 ή 5 διότι είναι δευτερεύον)
ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &my_id );
// Εύρεση του αριθμού των υπολογιστών
ierr = MPI_Comm_size ( MPI_COMM_WORLD, &num_procs );
cout << "\n";
cout << "Διεργασία " << my_id << " ενεργή \n";
// Οι δευτερεύοντες υπολογιστές δέχονται την τιμή που αναζητείται ,το πλήθος των δεδομένων καθώς τα δεδομένα του πίνακα για τα οποία θα κάνουν την αναζήτηση. Στη συνέχεια κάνουν την αναζήτηση και στέλνουν στον κύριο υπολογιστή τα αποτελέσματα.
source = master; tag = tag_target;
MPI::COMM_WORLD.Recv ( &target, 1, MPI::INT, source, tag, status);
source = master; tag = tag_size;
MPI::COMM_WORLD.Recv ( &npart, 1, MPI::INT, source, tag, status);
a = new int[npart];
source = master; tag = tag_data;
MPI::COMM_WORLD.Recv ( a, npart, MPI::INT, source, tag, status );
for ( i = 0; i < npart; i++ )
{
if ( a[i] == target )
{
global = ( my_id - 1 ) * npart + i; dest = master;
tag = tag_found;
MPI::COMM_WORLD.Send ( &global, 1, MPI::INT, dest, tag );
}
}
dest = master; tag = tag_done;
MPI::COMM_WORLD.Send ( &target, 1, MPI::INT, dest, tag );
delete [] ( a );
return 0;
}
```

## 11.1. Αποτελέσματα της εκτέλεσης των προγραμμάτων

Έπειτα από την ανάλυση που έγινε παραπάνω τόσο για το σειριακό όσο και για το παράλληλο πρόγραμμα ακολουθεί εκτέλεση των δύο προγραμμάτων και καταγραφή των αποτελεσμάτων των χρονομετρητών.
Το παράλληλο πρόγραμμα εκτελέστηκε στη συστοιχία από έξι υπολογιστές που υπάρχει στο εργαστήριο. Η εκτέλεση επαναλήφθηκε έξι φορές αλλάζοντας κάθε φορά το πλήθος των αριθμών μέσα στους οποίους γίνεται η αναζήτηση. Ακριβώς η ίδια διαδικασία έγινε και για το σειριακό πρόγραμμα , το οποίο φυσικά έτρεξε σε έναν μόνο υπολογιστή ο οποίος αποτελεί μέλος της συστοιχίας έτσι ώστε τα αποτελέσματα παράλληλου-σειριακού προγράμματος να μπορούν να συγκριθούν μιας και προέκυψαν από ίδιας υπολογιστικής ισχύος μηχανήματα. Στη συνέχεια εκθέτουμε ένα πίνακα ο οποίος δείχνει το χρόνο στον οποίο ολοκληρώθηκε η αναζήτηση τόσο στο παράλληλο όσο και στο σειριακό πρόγραμμα για διάφορες τιμές του πλήθους των αριθμών. Ο χρόνος υπολογίστηκε με τη βοήθεια του χρονομετρητή του οποίου η λειτουργία αναλύθηκε παραπάνω.

| Πλήθος αριθμών μέσα στους οποίους γίνεται η αναζήτηση | Αποτελέσματα χρονομετρητη παράλληλου προγράμματος σε δευτερόλεπτα | Αποτελέσματα χρονομετρητή σειριακού προγράμματος σε δευτερόλεπτα |
|---|---|---|
| 30.000 | 0.02 | 0.01 |
| 180.000 | 0.15 | 0.05 |
| 600.000 | 0.24 | 0.17 |
| 1.800.000 | 0.58 | 0.5 |
| 6.000.000 | 1.7 | 1.67 |
| 18.000.000 | 5.64 | 5.61 |

Βλέποντας λοιπόν τα πειραματικά αποτελέσματα συμπεραίνουμε ότι για ένα μεγάλος εύρος πλήθους αριθμών, από 30.000 έως και 18.000.000 αριθμοί , η αναζήτηση στο σειριακό πρόγραμμα εκτελείται ταχύτερα. Κάτι τέτοιο ήταν αναμενόμενο για μικρές τιμές του πλήθους μιας ο χρόνος που απαιτείται για την επικοινωνία των υπολογιστών της συστοιχίας ακυρώνει το όποιο πλεονέκτημα αποκτούμε από τη χρήση πολλαπλάσιας υπολογιστικής ισχύος. Επίσης βλέπουμε ότι όσο μεγαλώνει το πλήθος των αριθμών μέσα στους οποίους γίνεται η αναζήτηση οι τιμές των χρονομετρητών των δύο προγραμμάτων συγκλίνουν. Αυτό δείχνει ότι το παράλληλο πρόγραμμα αποκτά χρησιμότητα για μεγάλες τιμές πλήθους αριθμών όπου ο χρόνος που «χάνεται» για επικοινωνία των υπολογιστών κερδίζεται από την αυξημένη υπολογιστική ισχύ , γεγονός που καθιστά φανερή τη χρησιμότητα της συστοιχίας . Εντούτοις τα αποτελέσματα δεν είναι αυτά ακριβώς που αναμέναμε , δηλαδή για πολύ μεγάλο πλήθος αριθμών, π.χ. 10.000.000, η συστοιχία να έχει πλεονέκτημα έναντι της σειριακής εκτέλεσης , διότι η σύνδεση των υπολογιστών μεταξύ τους, όπως αναφέρθηκε παραπάνω, γίνεται με κάρτες δικτύου Ethernet , γεγονός το οποίο καθιστά τη συστοιχία αργή σε σχέση με άλλες οι οποίες χρησιμοποιούν κάποιον άλλο διασυνδετή δικτύου, π.χ. Myrinet ή Quadrics. Από τα δύο προγράμματα , λοιπόν, τα οποία εκτελέσαμε προκύπτουν δύο βασικά συμπεράσματα : ότι η χρησιμότητα της συστοιχίας αναδεικνύεται για μεγάλο πλήθος αριθμών μέσα στους οποίους γίνεται η αναζήτηση και ότι η συστοιχία του εργαστηρίου έχει περιορισμούς στις δυνατότητές της που οφείλονται στον υπάρχον τρόπο σύνδεσης των υπολογιστών.

## 12. Αναστέλλουσα επικοινωνία

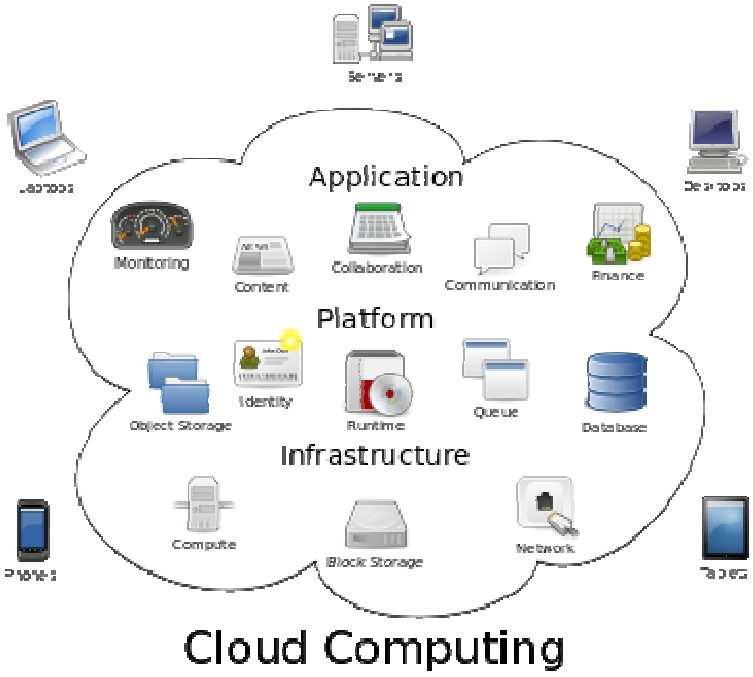
Στην αναστέλλουσα (blocking) επικοινωνία, η συνάρτηση αποστολής και η συνάρτηση παραλαβής μηνυμάτων αναστέλλουν την εκτέλεση της διεργασίας, που τις καλεί, έως ότου ολοκληρωθεί η διαδικασία αποστολής ή παραλαβής, αντίστοιχα. Υπάρχουν δύο βασικές συναρτήσεις, οι οποίες υλοποιούν την αναστέλλουσα επικοινωνία. Η MPI_Send και η MPI_Recv, οι οποίες χρησιμοποιούνται για αποστολή και λήψη μηνυμάτων αντίστοιχα. Και οι δυο συναρτήσεις αναστέλλουν τη διαδικασία που τις καλεί, αφού καμιά τους δεν επιστρέφει, έως ότου η αντίστοιχη διαδικασία επικοινωνίας (αποστολής ή λήψης) ολοκληρωθεί.

## 13. Μαζικά παράλληλοι επεξεργαστές

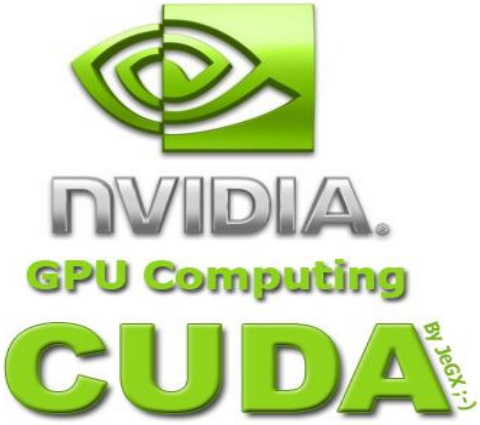
Οι μαζικά παράλληλοι επεξεργαστές (Massive Parallel Process - MPP) αναφέρονται σε ειδικά παράλληλα συστήματα με πολύ μεγάλο αριθμό επεξεργαστών. Οι μαζικά παράλληλοι επεξεργαστές έχουν πολλά κοινά χαρακτηριστικά με τα clusters, παρ' όλα αυτά έχουν εξειδικευμένα εσωτερικά διασυνδεδεμένα δίκτυα ενώ αντιθέτως τα clusters χρησιμοποιούν εμπορικό υλικό για δικτύωση. Συστήματα με μαζικά παράλληλους επεξεργαστές, τείνουν να είναι μεγαλύτεροι από τα clusters, αφού έχουν συνήθως πάνω από εκατό επεξεργαστές. Στο μαζικό παράλληλο επεξεργαστή κάθε μονάδα κεντρικής επεξεργασίας, περιέχει τη δική του μνήμη, αντίγραφο λειτουργικού συστήματος και λογισμικού εφαρμογής. Κάθε υποσύστημα επικοινωνεί με όλα τα άλλα μέσω διασύνδεσης υψηλής ταχύτητας Στους μαζικά παράλληλους επεξεργαστές, τα δίκτυα διασύνδεσης δεν είναι ούτε συμβατικοί δίαυλοι, αλλά ούτε και τοπικά δίκτυα διαχωρισμένα από συστοιχίες, clusters. Χρησιμοποιούνται συνήθως ιεραρχικά συνδεδεμένα συστήματα μεταγωγής ή ειδικά διασυνδετικά δίκτυα.

## 14. Special computing (GPU + Cloud)

Στο Cloud Computing παρέχονται υπηρεσίες πρόσβασης σε απομακρυσμένα δεδομένα ή αποθήκευσης, στις οποίες ο τελικός χρήστης δε γνωρίζει τη γεωγραφική θέση και τη διαμόρφωση του συστήματος που παρέχει τις υπηρεσίες που προσφέρονται σε αυτόν. Σκοπός είναι η μείωση της σπατάλης πόρων και ταυτόχρονα η παροχή υπολογιστικής ισχύος στου χρήστες Το Cloud Computing ολοκληρώνεται με τροφοδότηση δυναμικών, κλιμακωτών και συχνά εικονικών πόρων. Η πρόσβαση σε απομακρυσμένες τοποθεσίες υπολογιστών, μέσω διαδικτύου γίνεται με μορφή εφαρμογών ή εργαλείων, όπου οι χρήστες έχουν πρόσβαση μέσω ενός προγράμματος περιήγησης, κατά τέτοιο τρόπο ώστε να παρουσιάζεται «εικονικά» η εγκατάσταση προγραμμάτων στον υπολογιστή τους. Οι υπηρεσίες του Cloud Computing, είναι α) το Software-as-as-a-Service, β) το Platform-as-a-Service και το γ) Infrastructure-as-a-Service, με το καθένα να εξυπηρετεί διαφορετικές ανάγκες και υπηρεσίες. Μεγάλες παγκόσμιες εταιρείες στον τομέα της πληροφορικής, όπως η Google, η Amazon, η Microsoft, η Apple κ.α. έχουν αναπτύξει υπηρεσίες και προϊόντα, προωθώντας το cloud computing.




Το Cloud Computing, υποστηρίζει επεκτασιμότητα και εικονικότητα των πόρων. Μέσω του cloud γίνεται παροχή πόρων ως υπηρεσίες προς τους χρήστες κάνοντας χρήση του διαδικτύου. Οι υπηρεσίες που παρέχει ένα cloud είναι προσβάσιμες μέσω ενός διαφυλλιστή browser, ενώ τα δεδομένα και το λογισμικό είναι αποθηκευμένα στον εξυπηρετητή του συστήματος.

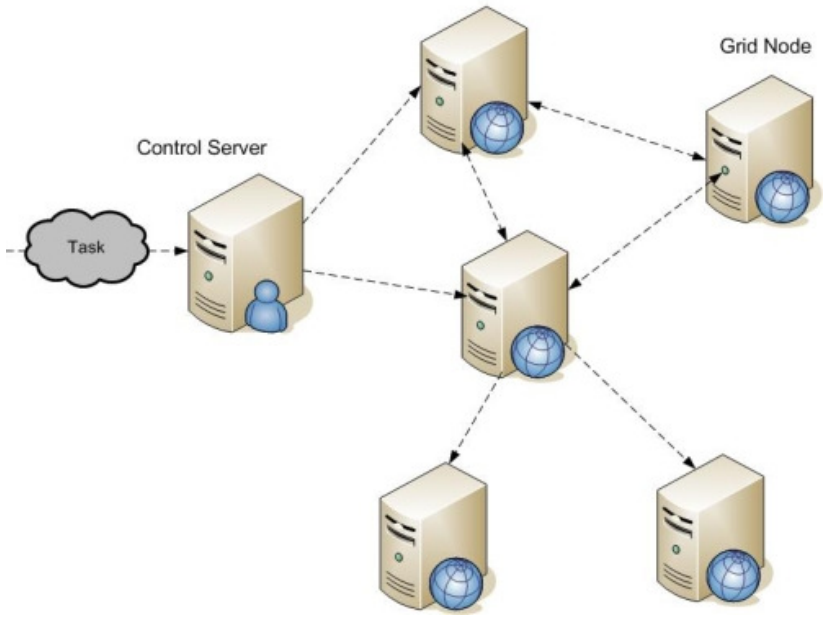



Οι GPUs (Graphics Processing Units) είναι συν-επεξεργαστές εξαιρετικά βελτιστοποιημένοι και χρησιμοποιούνται κυρίως για επεξεργασία γραφικών. Η επεξεργασία γραφικών αποτελεί πεδίο όπου κυριαρχούν λειτουργίες παραλληλισμού δεδομένων και ειδικότερα λειτουργίες γραμμικής άλγεβρας πινάκων. Αρκετό καιρό τα συστήματα GPUs χρησιμοποιούσαν συνηθισμένες εφαρμογές γραφικών (APIs), ώστε να εκτελέσουν ορισμένα προγράμματα. Τα τελευταία χρόνια αναπτύσσονται νέες γλώσσες προγραμματισμού και πλατφόρμες για υπολογισμούς και επίλυση προβλημάτων σε GPUs. Η NVIDIA, η AMD έχουν εκδώσει περιβάλλοντα προγραμματισμού όπως η CUDA και η CMT, η OpenCL κ.α. Γλώσσες προγραμματισμού GPUs είναι οι BrookGPU, PeakStream και RapidMind. Η NVIDIA έχει μάλιστα εκδώσει ειδικά πακέτα λογισμικού για εφαρμογή στις κάρτες Tesla.

GPU προγραμματίσιμων μονάδων ακολουθούν το προγραμματιστικό μοντέλο μοναδικού προγράμματος-πολλαπλών δεδομένων. Η GPU, επεξεργάζεται πολλά στοιχεία παράλληλα, χρησιμοποιώντας το ίδιο πρόγραμμα. Κάθε στοιχείο είναι ανεξάρτητο από το άλλο και στο βασικό μοντέλο προγραμματισμού τα στοιχεία δεν επικοινωνούν μεταξύ τους. Όλα τα προγράμματα GPU πρέπει να είναι δομημένα διαφορετικά: πολλά παράλληλα στοιχεία, το καθένα επεξεργαζόμενο παράλληλα από ένα μοναδικό πρόγραμμα. Κάθε στοιχείο λειτουργεί σε ένα 32-bit ακέραιο ή κινητής υποδιαστολής, δεδομένο με ένα ολοκληρωμένο σετ εντολών γενικού-σκοπού. Τα στοιχεία διαβάζουν δεδομένα από μια κοινή global μνήμη, ενώ με τις καινούριες GPUs γράφουν πίσω σε οποιεσδήποτε τοποθεσίες στη κοινή global μνήμη.

## 15. Grid computing

Grid είναι τα "Κατανεμημένα συστήματα ανωτέρου επιπέδου". Σε αντίθεση με τα απλά κατανεμημένα συστήματα είναι: α) ότι μπορούν να βρίσκονται σε πολύ μεγάλη απόσταση μεταξύ τους, β) η δημιουργία εικονικού ενοποιημένου συστήματος εκατοντάδων / χιλιάδων /... / συνδεδεμένων υπολογιστών. Αυτοί οι υπολογιστές επικοινωνούν είτε μέσω τοπικών δικτύων, είτε μέσω internet, είτε μέσω δικτύων ευρείας περιοχής (WAN) κ.α.




Εικ. 2: Υπολογιστικά συστήματα grid

Η αύξηση ταχύτητας δικτύων, καθώς και η τυποποίηση κοινόχρηστων πόρων μεταξύ συστημάτων οδηγούν και βοηθούν στην ανάπτυξη και εξάπλωση του grid. Το grid computing χρησιμοποιούνται στη βιοιατρική, σε επεξεργασία εικόνας, αλλά και σε πατροπαράδοτους επιστημονικούς κλάδους, π.χ. οικονομικά κ.α. Πολλοί ερευνητικοί οργανισμοί χρησιμοποιούν τις λειτουργίες grid, έχοντας μια εκθετική αύξηση απόδοσης του έργου τους.

Η εξάπλωση του grid computing προϋποθέτει την ύπαρξη διαφόρων ρόλων μέσα στο grid. Δημιουργούνται ιδεατοί (αρχεία) ή εικονικοί οργανισμοί που αποτελούνται από ομάδες χρηστών με κάποια κοινά χαρακτηριστικά (απαιτήσεις, εξοπλισμό κ.α.). Επίσης με το grid computing υπάρχει πρόσβαση σε επιπλέον πόρους, όπως software, άδειες χρήσης και άλλες υπηρεσίες. Το grid δίνει μια νέα ιδέα ασφάλειας και αξιοπιστίας, αφού οι υπολογιστές μπορούν να είναι διασκορπισμένοι σε οποιοδήποτε σημείο ανεξάρτητης απόστασης. Χρησιμοποιώντας πολυπρογραμματισμό σε περίπτωση που χαθεί ή που δε λειτουργεί σωστά κάποιο κομμάτι του προγράμματος μπορεί να ξαναεπεξεργαστεί και να αποσταλεί ξανά εκεί που ζητείται. Επίσης μια άλλη υλοποίηση είναι ότι το ίδιο πρόγραμμα εκτελείται ταυτόχρονα σε περισσότερους από έναν υπολογιστή.

Το σύστημα του grid ευθύνεται για την αποστολή και εκτέλεση μίας εργασίας σε ένα μηχάνημα. Σε απλά συστήματα grid, ο χρήστης επιλέγει από μία λίστα διαθέσιμους υπολογιστές και στη συνέχει εκτελώντας συγκεκριμένες εντολές του grid, στέλνει την εργασία στο μηχάνημα. Αντίστοιχα σε πολυπλοκότερα και πιο ανεπτυγμένα συστήματα grid, ο προγραμματιστής εργασιών είναι αυτός που αυτόματα βρίσκει τον κατάλληλο υπολογιστή για την εκάστοτε εργασία. Οι προγραμματιστές εργασιών αλληλεπιδρούν με τη διαθεσιμότητα των πόρων στο δίκτυο. Σε περιπτώσεις που ο υπολογιστής φορτωθεί με

τοπικές εργασίες ή όταν μια εργασία που έχει αιτηθεί να διεκπεραιωθεί μέσα από το grid, σταματάει μέχρι να τελειώσει η τοπική εργασία ή καθυστερεί, έχει ως αποτέλεσμα την απροσδόκητη καθυστέρηση εκτέλεσης, παρότι το μηχάνημα αυτό έχει μοιράσει τους πόρους του στο grid. Έτσι, για την αποφυγή τέτοιων περιστατικών σε αρκετές περιπτώσεις δεν επιτρέπεται σε υπολογιστές να έχουν τοπικές εργασίες και οι πόροι τους να δεσμεύονται στο grid.

Το Cloud computing με το Grid computing έχουν κάποια βασικά κοινά χαρακτηριστικά, όπως η υποστήριξη, συνάθροιση ετερογενών πόρων και η πρόσβαση χρηστών σε πόρους με διαφανή τρόπο. Οι διαφορές τους έγκειται στα εξής σημεία:

- Κοινή χρήση πόρων: Στο grid computing υποστηρίζεται η κοινή χρήση πόρων, ενώ στο cloud computing, δεν υποστηρίζεται εξαιτίας της απομόνωσης μέσου εικονοποίησης.
- Υψηλού επιπέδου υπηρεσίες: Στο grid computing υπάρχει πληθώρα τέτοιων υπηρεσιών, ενώ στο cloud computing έλλειψη, που ίσως οφείλεται στο χαμηλό επίπεδο ωριμότητας.
- Αρχιτεκτονική: Το grid computing προσανατολίζεται στις υπηρεσίες, ενώ το cloud computing σε αρχιτεκτονικές που εξαρτώνται από το χρήστη.
- Ροή εργασιών λογισμικού: Στο grid computing οι εφαρμογές απαιτούν προκαθορισμένη ροή εργασιών για συντονισμό υπηρεσιών, ενώ στο cloud computing η ροή εργασιών λογισμικού δεν παίζει σημαντικό ρόλο.
- Χρησιμότητα: Στο grid computing υπάρχει μικρός βαθμός ευχρηστίας, ενώ στο cloud computing μεγαλύτερος βαθμός ευχρηστίας μέσω απόκρυψης λειτουργιών.
- Προτυποποίηση: Στο grid computing υπάρχουν πρότυπα στα οποία επιτυγχάνεται η διαλειτουργικότητα, ενώ στο cloud computing υπάρχει ανάγκη προτύπων αποθήκευσης, καθορισμού διεπαφών και ποιότητας υπηρεσιών.
- Κόστος: Σχετικά με το grid computing η τιμολόγηση γίνεται βάση ενός σταθερού ποσού ανά υπηρεσία, ενώ στο cloud computing η τιμολόγηση γίνεται βάση της χρήσης της υπηρεσίας.

# 16. INFINIBAND

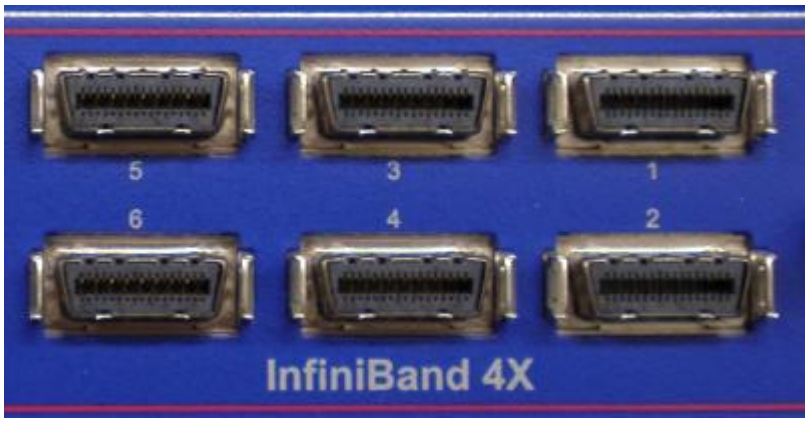
**Το InfiniBand** είναι ένα δίκτυο επικοινωνίας τύπου "δικτυοδομής μεταγωγής (switching fabrics)" που χρησιμοποιούνται σε υπολογιστές υψηλών επιδόσεων και σε datacenters

Τα χαρακτηριστικά του περιλαμβάνουν την υψηλή απόδοση, low latency, QoS (Quality of Service) και Failover και έχει σχεδιαστεί για να είναι επεκτάσιμο .

Η αρχιτεκτονική InfiniBand ορίζει μια σύνδεση μεταξύ των επεξεργαστών των κόμβων με υψηλής απόδοσης I/O κόμβους, όπως συσκευές αποθήκευσης (Storage).

Το Infiniband και τα switch (μεταγωγεις) έχουν κατασκευαστεί από την Mellanox και την Intel (η οποία απέκτησε και το QLogic 's infiniband τον Ιανουάριο του 2012 [1] ).

Το InfiniBand αποτελεί ένα υπερσύνολο της αρχιτεκτονικής Virtual Interface (VIA).

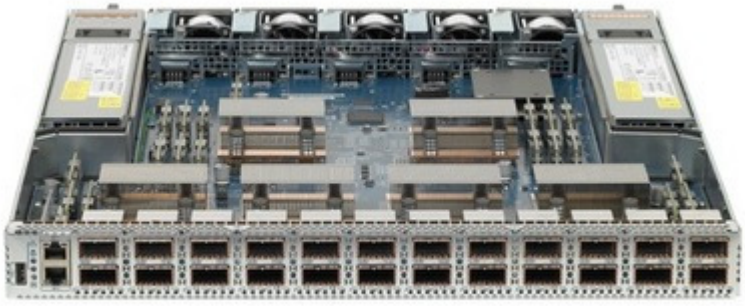



## 16.1. Περιγραφή

**Αποτελεσματική μονής κατεύθυνσης θεωρητική απόδοση**

| | SDR | DDR | QDR | FDR-10 | FDR | EDR |
|---|---|---|---|---|---|---|
| **1X** | 2 Gbit / s | 4 Gbit / s | 8 Gbit / s | 10.3125 Gbit / s | 13.64 Gbit / s | 25 Gbit / s |
| **4X** | 8 Gbit / s | 16 Gbit / s | 32 Gbit / s | 41.25 Gbit / s | 54.54 Gbit / s | 100 Gbit / s |
| **12X** | 24 Gbit / s | 48 Gbit / s | 96 Gbit / s | 123.75 Gbit / s | 163.64 Gbit / s | 300 Gbit / s |

Όπως και το Fibre Channel , PCI Express , Serial ATA , και πολλές άλλες σύγχρονες διασυνδέσεις, το InfiniBand προσφέρει σημείου-προς-σημείο αμφίδρομη σειριακή διασυνδέσεις που προορίζονται για τη σύνδεση των επεξεργαστών με υψηλής ταχύτητας περιφερειακά όπως δίσκους. Το InfiniBand προσφέρει επίσης και multicast λειτουργίες. Υποστηρίζει διάφορες τιμές σηματοδότησης και, όπως PCI Express , επίσης μπορούν να συνδέονται μεταξύ τους για πρόσθετη απόδοση (bonding).



## 16.2. Σηματοδοσία

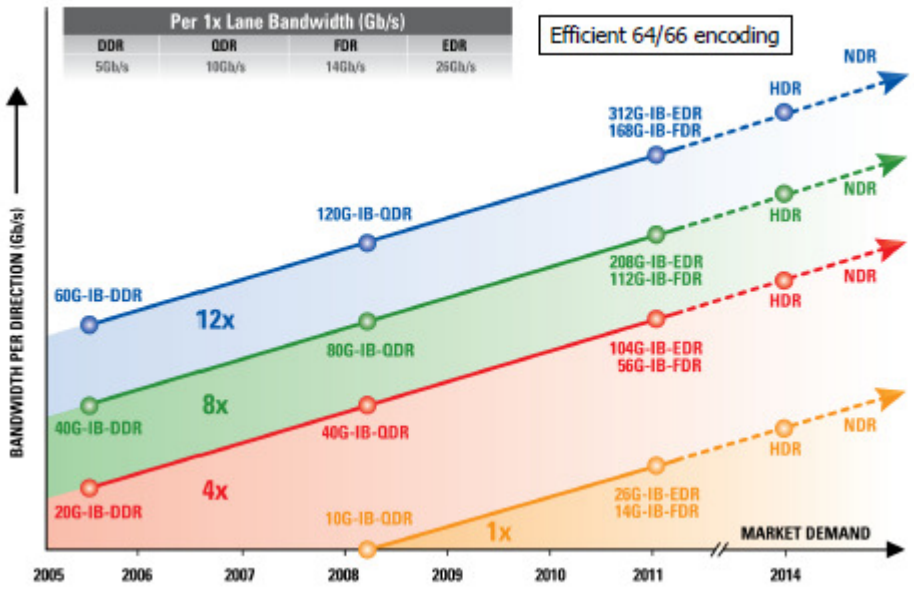
Μια σύνδεση Infiniband μπορεί να λειτουργεί σε σειριακή σύνδεση σε μία από τις πέντε ταχύτητες μεταφοράς δεδομένων single data rate (SDR), double data rate (DDR), quad data rate (QDR), fourteen data rate (FDR), and enhanced data rate (EDR).

Η ταχύτητα σήματος της σύνδεσης SDR είναι 2,5 gigabit ανά δευτερόλεπτο (Gbit / s) σε κάθε κατεύθυνση ανά σύνδεση. DDR είναι 5 Gbit / s και QDR είναι 10 Gbit / s. FDR είναι 14.0625 Gbit / s και EDR είναι 25,78125 Gbit / s ανά λωρίδα κυκλοφορίας.

Για SDR, DDR και QDR, συνδέσεις χρησιμοποιούν 8b/10b κωδικοποίηση - κάθε 10 bits αποστέλλονται μεταφέρουν 8bits των δεδομένων - επιτυγχάνοντας τη μετάδοση 4/5 της ταχύτητα RAW δεδομένα. Έτσι, για SDR, DDR και QDR ρυθμούς δεδομένων μεταφέρει 2, 4, ή 8 χρήσιμα στοιχεία Gbit / s, αντίστοιχα. Οι FDR-10, FDR και EDR, συνδέσεις χρησιμοποιούν 64b/66b κωδικοποίηση - κάθε 66 bits αποστέλλονται μεταφέρει 64 bits δεδομένων. (Σε κάθε από αυτούς τους υπολογισμούς λαμβάνουμε υπόψη τις πρόσθετες όπως το Physical Layer (Φυσικό επίπεδο και γενικά τα κοινά χαρακτηριστικά ή απαιτήσεις του πρωτοκόλλου, όπως StartOfFrame και EndOfFrame).

Οι κατασκευαστές μπορούν να ομαδοποιήσουν τις συνδέσεις σε μονάδες 4 ή 12, που ονομάζεται 4X ή 12X. Μια 12X σύνδεση QDR επομένως φέρνει 120 Gbit / s raw data, ή 96 Gbit / s userful data. Από το 2009 τα περισσότερα συστήματα χρησιμοποιούν ένα 4X συνολικά, πράγμα που συνεπάγεται 10 Gbit / s (SDR), 20 Gbit / s (DDR) ή 40 Gbit / s (QDR) σύνδεση. Τα μεγαλύτερα συστήματα με 12X συνδέσεις που χρησιμοποιούνται συνήθως για cluster και διασύνδεση υπερυπολογιστικών συστημάτων αλλά και διασύνδεση των ίδιων των μεταγωγών (Switches interconnect)

Το μέλλον του InfiniBand έχει επίσης «HDR» (High Data Rate), αναμένεται το 2014, και το "NDR" (Next Data Rate), "στο μέλλον" αλλά από τον Ιούνιο του 2010, τα ποσοστά αυτά δεν έχουν ακόμα θεσπιστεί




## 16.3. Latency

Το SDR switch chip έχει χρόνο Latency 200 νανοδευτερόλεπτα , DDR switch chip εχει χρόνο Latency 140 νανοδευτερόλεπτα και QDR switch chip εχει χρόνο Latency 100 νανοδευτερόλεπτα . Η end-to-end διασυνδέσεις έχουν εύρο που εκτείνεται από 1,07 μικροδευτερόλεπτα MPI latency ( Mellanox ConnectX QDR HCAs) σε 1,29 μικροδευτερόλεπτα MPI latency (Qlogic InfiniPath HCAs) σε 2,6 μικροδευτερόλεπτα (Mellanox InfiniHost DDR III HCAs). Από το 2009 διάφορες *host channel adapters* InfiniBand (HCA) υπάρχουν στην αγορά, το καθένα με διαφορετικά χαρακτηριστικά λανθάνουσα κατάσταση και το εύρος ζώνης. InfiniBand παρέχει επίσης RDMA δυνατότητες για τη χαμηλή χρήση της CPU. Ο λανθάνων χρόνος για RDMA εργασίες είναι μικρότερη από 1 μικροδευτερόλεπτο (Mellanox HCAs ConnectX).

## 16.4. Τοπολογία

InfiniBand χρησιμοποιεί ένα "δικτυοδομής μεταγωγής (switching fabrics)" ως τοπολογία, σε αντίθεση με ένα ιεραρχικό δίκτυο μεταγωγής όπως οι παραδοσιακά Ethernet αρχιτεκτονικές. Όλες οι μεταδόσεις αρχίζουν ή τελειώνουν σε "channel adapter" Κάθε επεξεργαστής περιέχει ένα *host channel adapter* (HCA) και κάθε περιφερειακή συσκευή διαθέτει έναν *target channel adapter* (TCA). Αυτοί οι προσαρμογείς μπορούν επίσης να ανταλλάσσουν πληροφορίες για την ασφάλεια ή την ποιότητα των υπηρεσιών (QoS).

## 16.5. Μηνύματα

Το InfiniBand μεταδίδει δεδομένα σε πακέτα μέχρι 4 KB που λαμβάνονται μαζί για να σχηματίσουν ένα *μήνυμα.* Ένα μήνυμα μπορεί να είναι:

- μια άμεση πρόσβαση στη μνήμη (direct memory access DMA) μπορεί να γράψει ή να αναγνώσει, ένα απομακρυσμένο κόμβο ( RDMA )
- ένα κανάλι για αποστολή και λήψη
- μια συναλλαγή με βάση τη λειτουργία (που μπορεί να αντιστραφεί)
- μια multicast μετάδοση.
- μια atomic operation

## 16.6. Εφαρμογές

Το InfiniBand έχει υιοθετηθεί στα datacenters των επιχειρήσεων, για παράδειγμα Oracle Exadata Database Machine, Oracle Exalogic Elastic Cloud και Oracle SPARC supercluster, χρηματοπιστωτικών τομέων, το cloud computing και πολλά άλλα. Το InfiniBand έχει ως επί το πλείστον χρησιμοποιείθει για την υψηλή απόδωση των επιδόσεων σε computer cluster εφαρμογές. Μεγάλος αριθμός των "TOP500" υπερυπολογιστών χρησιμοποιουν το InfiniBand συμπεριλαμβανομένης και της IBM Roadrunner "Πρωην ταχύτατος υπεροπολογιστή παγκοσμίος" .

SGI , LSI, DDN, Netapp, Oracle, Nimbus Data, Rorke Data, μεταξύ άλλων, έχουν κυκλοφορήσει storages που χρησιμοποιούν InfiniBand "target adapters". Τα προϊόντα αυτά ουσιαστικά ανταγωνίζονται αρχιτεκτονικές όπως Fibre channel, SCSI , και άλλες πιο παραδοσιακές μεθόδους συνδεσιμότητας. Το 2009, το Oak Ridge National-Lab Spider storage system χρησιμοποίησε αυτό το είδος της InfiniBand Attached Storage για να μεταδώσει πάνω από 240 gigabytes ανά δευτερόλεπτο.

## 16.7. Φυσική διασύνδεση

Το InfiniBand χρησιμοποιεί καλώδιο χαλκού ονομαζόμενο και ως CX4 καλώδιο για SDR και DDR τιμές - επίσης συνήθως χρησιμοποιείται για τη σύνδεση SAS ( Serial Attached SCSI ) HBAs σε εξωτερικές (SAS) συστοιχίες δίσκων. Στις SAS, αυτό είναι γνωστό ως SFF-8470 connector, και αναφέρεται ως "InfiniBand-style" Connector. Για συνδέσεις με QDR και FDR είναι QSFP (Quad SFP ) και μπορεί να είναι από χαλκό ή οπτικές ίνες, ανάλογα με το απαιτούμενο μήκος καλωδίου.

## 16.8. Προγραμματισμός

Το InfiniBand δεν έχει ακόμα κάποιο πρότυπο API για προγραμματισμού εντός των προδιαγραφών. Το πρότυπο απαριθμεί μόνο μια σειρά από «verbs» - Functions που υπάρχουν. Η σύνταξη αυτών των συναρτήσεων έχει αφεθεί στους κατασκευαστές. Το de facto πρότυπο σήμερα ήταν η σύνταξη που αναπτύχθηκε από τη OpenFabrics Alliance , η οποία εγκρίθηκε από τους περισσότερους από τους κατασκευαστές InfiniBand, για το GNU / Linux , FreeBSD , και MS τα Windows . Το λογισμικό InfiniBand αναπτύχθηκε από OpenFabrics Alliance και κυκλοφόρησε ως "Διανομή OpenFabrics Enterprise (OFED)", σύμφωνα με μια επιλογή από δύο άδειες GPL2 ή άδεια BSD για το Linux και FreeBSD, και ως «WinOF" για τα Windows.

## 16.9. Ιστορία

Το InfiniBand προήλθε από τη συγχώνευση δύο ανταγωνιστικών σχεδίων το 1999:

1. *Future I / O,* που αναπτύχθηκε από την Compaq , IBM και Hewlett-Packard
2. *Next Generation I / O (ngio),* που αναπτύχθηκε από την Intel , Microsoft και Sun

Το InfiniBand είχε αρχικά σχεδιαστεί ως ένα ολοκληρωμένο "system area network" που θα συνδέει επεξεργαστές και θα παρέχουν υψηλές ταχύτητές I/O για τις "back-office" εφαρμογές. Από το 2009 το InfiniBand έχει γίνει μια δημοφιλής διασύνδεσης για υπολογιστές υψηλών επιδόσεων, και η υιοθέτηση του όπως φαίνεται στο TOP500 supercomputers list και είναι ταχύτερη από ό, τι το Ethernet. [8] Τα τελευταία χρόνια InfiniBand έχει υιοθετείται όλο και περισσότερο στα datacenter των επιχειρήσεων.

Το 2008, Oracle Corporation κυκλοφόρησε το HP Oracle Database Machine για την κατασκευή RAC (Real Application clustering database) με την αποθήκευση παρέχονται σε έναν Exadata storage server που χρησιμοποιεί InfiniBand ως backend διασύνδεση για όλα τα IO Interconnects και την κυκλοφορία. Ενημερωμένες εκδόσεις του Exadata storage system, χρησιμοποιούν ακόμα Sun computing hardware, ώστε να συνεχίσουν να χρησιμοποιούν τις υποδομές InfiniBand.

Το 2009, η IBM ανακοίνωσε Δεκέμβριος 2009 Ημερομηνία απελευθέρωσης για τους DB2 pureScale προσφορά, ένα κοινό δίσκο συστήματος ομαδοποίησης (εμπνευσμένο από την παράλληλη sysplex DB2 για z / OS) που χρησιμοποιεί ένα σύμπλεγμα IBM System p servers ( POWER6 / 7 ) που επικοινωνεί με κάθε άλλο πέρα από μια διασύνδεση InfiniBand.

Τον Ιούνιο του 2011, FDR swtches και προσαρμογείς ανακοινώθηκαν στο Διεθνές Συνέδριο υπερυπολογιστών .

## 16.10. Εγκατάσταση Infiniband

Πακέτα που πρέπει να εγκατασταθούν είναι:

openib-1.4.1-6.el5.noarch

libibverbs-1.1.3-2.el5.x86_64

libnes-0.9.0-2.el5.x86_64

libibumad-1.3.3-1.el5.x86_64

opensm-libs-3.3.3-2.el5.x86_64

swig-1.3.29-2.el5.x86_64

ibutils-libs-1.2-11.1.el5.x86_64

ibutils-1.2-11.1.el5.x86_64

(provides ibdiagnet and others)

opensm-3.3.3-2.el5.x86_64

libibmad-1.3.3-1.el5.x86_64

infiniband-diags-1.5.3-1.el5.x86_64

(provides handy tools like ibstat and ibstatus)

libibverbs-utils-1.1.3-2.el5.x86_64

(provides handy tools ibv_devinfo and ibv_devices)

libibverbs-devel-1.1.3-2.el5.x86_64

Με την εντολή **yum -y install < Όνομα πακέτου >** εγκαθιστάμε στο Centos τα πακέτα που επιθυμούμε.

Ελέγχουμε την ορθή λειτουργία του hardware

$ lspci | grep fini

Βεβαιωθείτε ότι η κάρτα εμφανίζεται! Αν όχι, υπάρχει βασικό πρόβλημα στη κάρτα.

```
$ lspci | grep fini
```

Kernel driver

Αν εγκαταστήσαμε επιτυχώς το λογισμικό και την κάρτα, μετά από επανεκκίνηση ελέγχουμε αν το λειτουργικό μπορεί να την χρησιμοποιήσει,

```
$ dmesg | grep mth
ib_mthca: Mellanox InfiniBand HCA driver v1.0 (April 4, 2008)
ib_mthca: Initializing 0000:51:00.0
```

Χρειαζόμαστε την mthca

```
$ modprobe ib_mthca
```

Επιβεβαιώνουμε με :

$ **lsmod** | **grep** ib_mthca
ib_mthca 158053 0
ib_mad 70757 5 ib_mthca,ib_umad,ib_cm,ib_sa,mlx4_ib
ib_core 104901 17
ib_mthca,ib_iser,ib_srp,rds,ib_sdp,ib_ipoib,rdma_ucm,rdma_cm,ib_ucm,ib_uverbs,ib_umad,ib_cm,iw_cm,ib_sa,mlx4_ib,ib_mad,iw_cxgb3

Εφόσον εμφανιστεί πάμε στον φάκελο
/sys/class/infiniband:

```
$ ls /sys/class/infiniband
mthca0
```

Με την εντολή ifconfig βλέπουμε ότι αναγνωριστεί πλήρως, η κάρτα από το σύστημα ως δικτυακός προσαγωγέας.

```
$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:14:5E:F4:3A:A8
inet addr:172.20.102.2  Bcast:172.20.255.255  Mask:255.255.0.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:253460 errors:0 dropped:0 overruns:0 frame:0
TX packets:140500 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:351821939 (335.5 MiB)  TX bytes:12147168 (11.5 MiB)
Interrupt:185 Memory:e4000000-e4012800

ib0       Link encap:InfiniBand  HWaddr 80:00:04:04:FE:80:00:00:00:00:00:00:00:00:00:00:
inet addr:172.21.102.2  Bcast:172.21.255.255  Mask:255.255.0.0
UP BROADCAST RUNNING MULTICAST  MTU:2044  Metric:1
RX packets:25 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:1 overruns:0 carrier:0
collisions:0 txqueuelen:256
RX bytes:1400 (1.3 KiB)  TX bytes:0 (0.0 b)

lo        Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:16436  Metric:1
RX packets:1516 errors:0 dropped:0 overruns:0 frame:0
TX packets:1516 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:2394328 (2.2 MiB)  TX bytes:2394328 (2.2 MiB)
```

Παραμετροποιούμε κατάλληλα το κάθε Interface της κάρτας

```
/etc/sysconfig/network-scripts/ifcfg-ib0
```

και αποδίδουμε ipaddress

```
DEVICE=ib0
BOOTPROTO=none
ONBOOT=yes
IPADDR=172.21.102.2
NETMASK=255.255.0.0
```

Ελέγχουμε ότι υπάρχει επικοινωνία:

```
$ ping ivc2
PING ivc2 (172.21.102.2) 56(84) bytes of data.
64 bytes from ivc2 (172.21.102.2): icmp_seq=1 ttl=64 time=2.38 ms
```

## 16.11. Εγκατάσταση Cloud Computing Software

### 16.11.1. Εισαγωγή

Στο παρόν κείμενο θα αναφέρουμε την αρχιτεκτονική και μεθοδολογία υλοποίησης του OpenNebula σε 2 IBM BladeChassis H23, με storage device IBM Storewise V7000 και δικτύωση μέσω οπτικών ινών.
Το λογισμικό OpenNebula θα πρέπει να κάνει τουλάχιστον τις εξής λειτουργίες:

- Δημιουργία, Διαγραφή, Διαχείριση των εικονικών μηχανών
- Migrate εικονικών μηχανών με Loss of Service
- Live Migrate εικονικών μηχανών χωρίς Loss of Service.
- Δημιουργία εικονικών μηχανών από live cd προς εγκατάσταση σε μορφή ISO
- Μεταφορά και λειτουργία εικονικών μηχανών V2V από VMware σε OpenNebula
- Μεταφορά και λειτουργία εικονικών μηχανών P2V από μια φυσική μηχανή σε OpenNebula
- Ανεξαρτητοποίηση της σύνδεσης του κάθε HOST με το Storage IBM StoreWise V7000 μέσω των οπτικών ινών από το Switch Brocade. Εκμετάλλευση της δυνατότητας μεταφοράς δεδομένων έως 8 Gbps.
- Ανοχή σε βλάβες HOST (Fault Tolerance). Θα πρέπει το λογισμικό OpenNebula ή το λειτουργικό σύστημα σε εύλογο χρονικό διάστημα να αναγνωρίζει αν το Host blade server είναι εκτός υπηρεσίας και αυτομάτως και σε συνάρτηση με τη πολιτική που υπάρχει στο OpenNebula να γίνει μετάπτωση των εικονικών μηχανών σε άλλους Host blade servers (εφόσον υπάρχουν διαθέσιμοι φυσικοί πόροι να τις εικονικές μηχανές)
- Ανοχή σε βλάβες Networking(Fault Tolerance). Θα πρέπει το λογισμικό OpenNebula ή το λειτουργικό σύστημα σε εύλογο χρονικό διάστημα να αναγνωρίζει αν το Host blade server είναι εκτός σύνδεσης με το δίκτυο διαχείρισης (π.χ. βλάβη στο δικτυακό εξοπλισμό διαχείρισης) και αυτομάτως και σε συνάρτηση με τη πολιτική που υπάρχει στο OpenNebula να γίνει μετάπτωση των εικονικών μηχανών σε άλλους Host blade servers (εφόσον υπάρχουν διαθέσιμοι φυσικοί πόροι να τις εικονικές μηχανές)
- Ανοχή σε βλάβες Networking(Fault Tolerance). Θα πρέπει το λογισμικό OpenNebula ή το λειτουργικό σύστημα σε εύλογο χρονικό διάστημα να αναγνωρίζει αν το Host blade server είναι εκτός σύνδεσης με το δίκτυο διαχείρισης (π.χ. βλάβης στο δικτυακό εξοπλισμό ενός εκ των δυο δικτυακών Switch) και αυτομάτως να γίνεται επαναδρομολογηση της ροής δεδομένων από το λειτουργικό Switch.

## 16.12. Υποδομή

Η εγκατάσταση φέρει των εξής εξοπλισμό με τις ακόλουθες συνδεσμολογίες:

- 28 Blade server HOST X86_64
- 2 Blade Chassis για 14 host blade servers
- 2 Brocade Switch έως 8Gbps ανά πόρτα διασύνδεσης
- 8 BNT Gigabit Ethernet + fiber switch
- 1 IBM Storwize V7000 (Storage)
- 28 expansion modules για hosts 2 Gigabit Ethernet 2 FC έως 8Gbps

Η διασύνδεση έχει ως εξής.
Κάθε blade server φέρει από ένα expansion module. Όλοι οι blade servers συνδέονται με το blade chassis όπου από εκεί καταλήγουν αντίστοιχα στα Ethernet switch και FC switch
Κάθε blade server φέρει και 2 Gigabit Ethernet ports από τη motherboard.
Σύνολο κάθε blade server έχει 4 Gigabit Ethernet port και 2 FC ports.
Όλες οι Ethernet ports συνδέονται μέσω του midplane του Chassis με ένα από τα 4 switch υπάρχουν σε κάθε Blade Chassis.
Όλες οι FC ports συνδέονται με τα 2 brocade switch που βρίσκονται σε κάθε blade chassis.
Τα brocade switch συνδέονται με το IBM Storwize V7000.

## 16.13. Υλοποίηση

Η υλοποίηση του έργου έχει γίνει με πολιτική High Availability και Fault Tolerance. Σημαντικό είναι όλες οι εικονικές μηχανές να λειτουργούν σε υψηλή διαθεσιμότητα και σε περίπτωση βλάβης αυτόματα το σύστημα να τις ενεργοποιεί εντός εύλογου χρονικού διαστήματος χωρίς καμία ενέργεια από το χρήστη και μειώνοντας στο ελάχιστο το χρόνο Loss of Service.

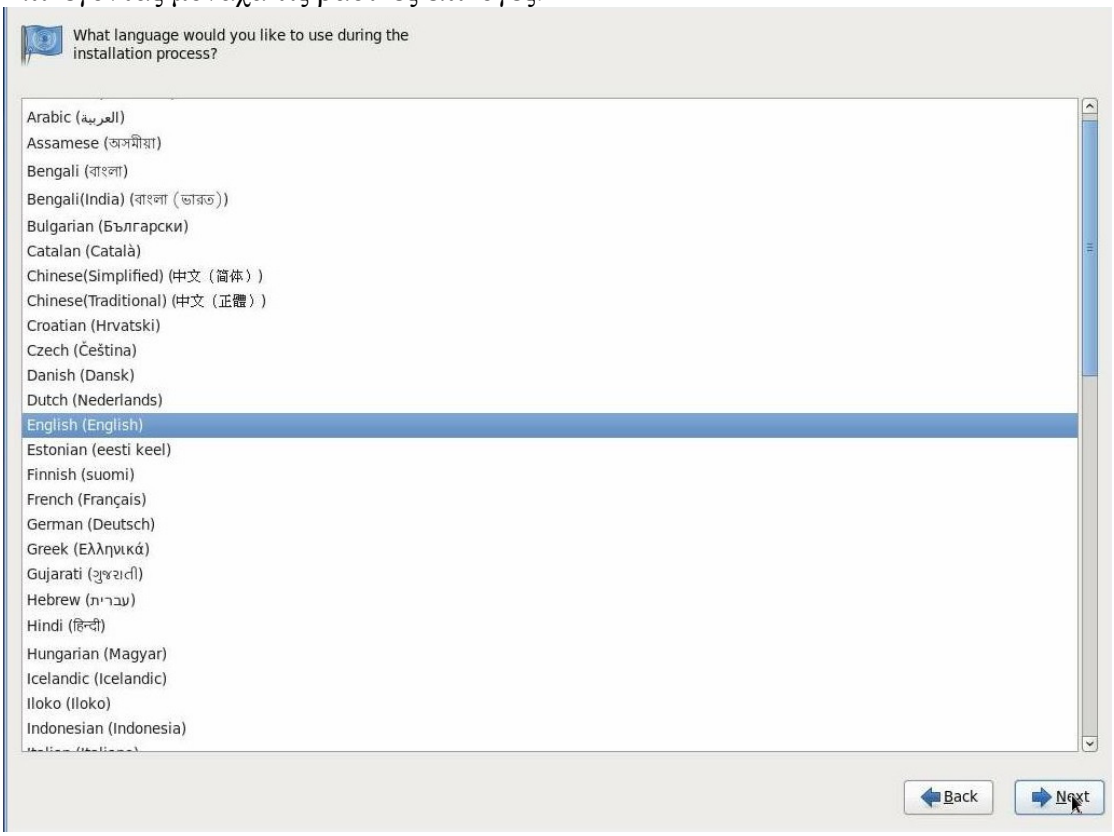
Οι εικονικές μηχανές θα είναι ανεξάρτητες από το blade chassis. Θα βρίσκονται και θα λειτουργούν στο Storage. Κάθε host θα μπορεί ταυτόχρονα να δει όλες τις εικονικές μηχανές και σε περίπτωση προβλήματος να αναλαμβάνει (μέσω εντολής από OpenNebula) την ενεργοποίηση της εικονικής μηχανής. Η επικοινωνία των host με το storage θα είναι έως 8 Gbps μέσω οπτικών ινών και brocade switch. Το λειτουργικό σύστημα θα είναι το Red Hat Enterprise Linux (64bit) σε όλα τα blade servers. Το λειτουργικό σύστημα θα πρέπει να επικοινωνεί με το storage σε τύπου Shared FS partition. Για την επικοινωνία με το storage οι 2 FC ports θα είναι σε multipath για High Availability. Θα γίνεται χρήση των 2 Ethernet που βρίσκονται στη μητρική μέσω των switch (Κάθετης τοποθέτησης) ως διαχείριση και γενικότερη επικοινωνία των blades. Αυτό θα γίνει σε διάταξη bonding (Active-Standby) για high availability Management connectivity. Αντίστοιχα οι άλλες 2 Ethernet ports πάλι σε διάταξη bonding (Active - Standby) θα χρησιμοποιούνται μόνο από τα VMs για σύνδεση με το δίκτυο.
Από τους 28 blade server θα χρησιμοποιηθούν 2 Server (1 server/chassis) για High Availability του OpenNebula.
Λόγω του ότι το OpenNebula κάνει χρήση SQL βάσης δεδομένων για αποθήκευση πληροφοριών VMs, User Accounts, Templates, Settings, Θα χρησιμοποιηθεί MySQL και στους 2 server σε replicate mode.
Οι δυο servers θα βρίσκονται σε διάταξη Active - Standby και με χρήση λογισμικών Heartbeat Και Pacemaker. Σε περίπτωση βλάβης του Primary blade 'Head' server ή απώλειας όλου του blade chassis θα αναλαμβάνει ο λειτουργικός server τη διαχείριση των VMs.
Σε περίπτωση Chassis Failure θα μπορεί να γίνει μεταφορά των εικονικών μηχανών από το ένα Chassis στο άλλο.

## 17. Εγκατάσταση OpenNebula

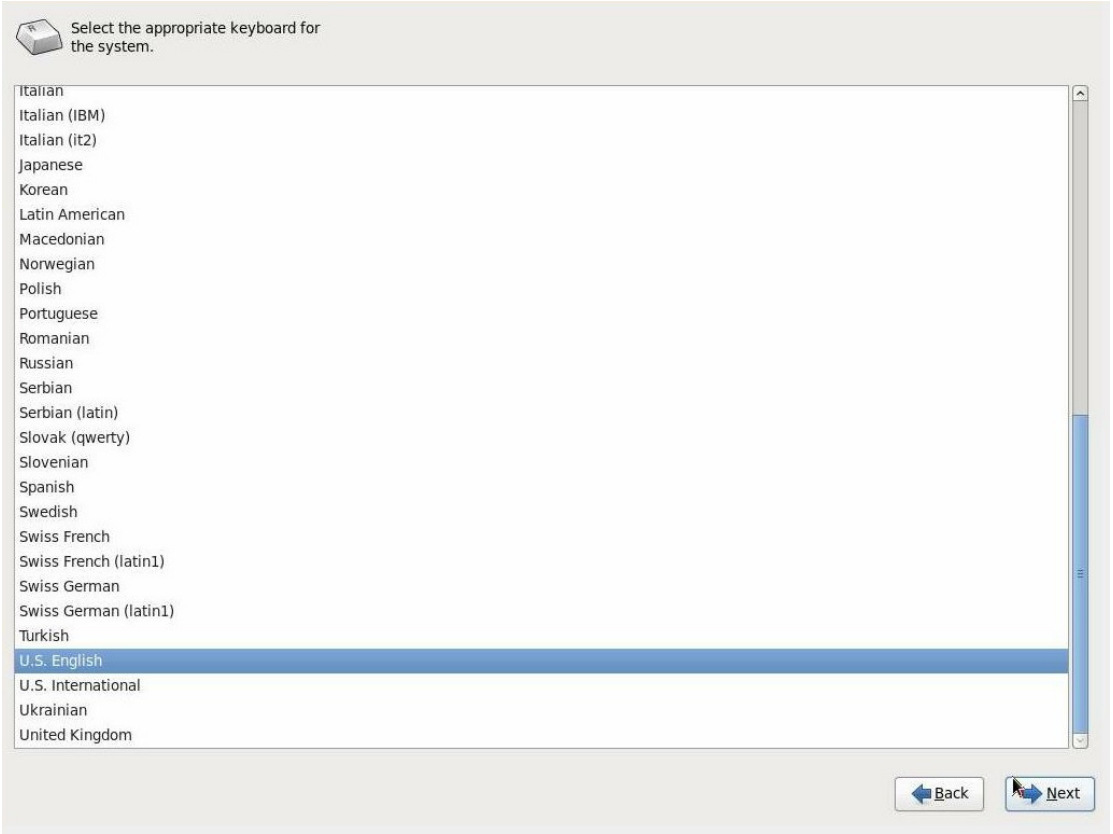
Γενικές Οδηγίες Εγκατάστασης

Εγκατάσταση Λειτουργικού Συστήματος RedHat Enterprise Linux 6.4 64bit

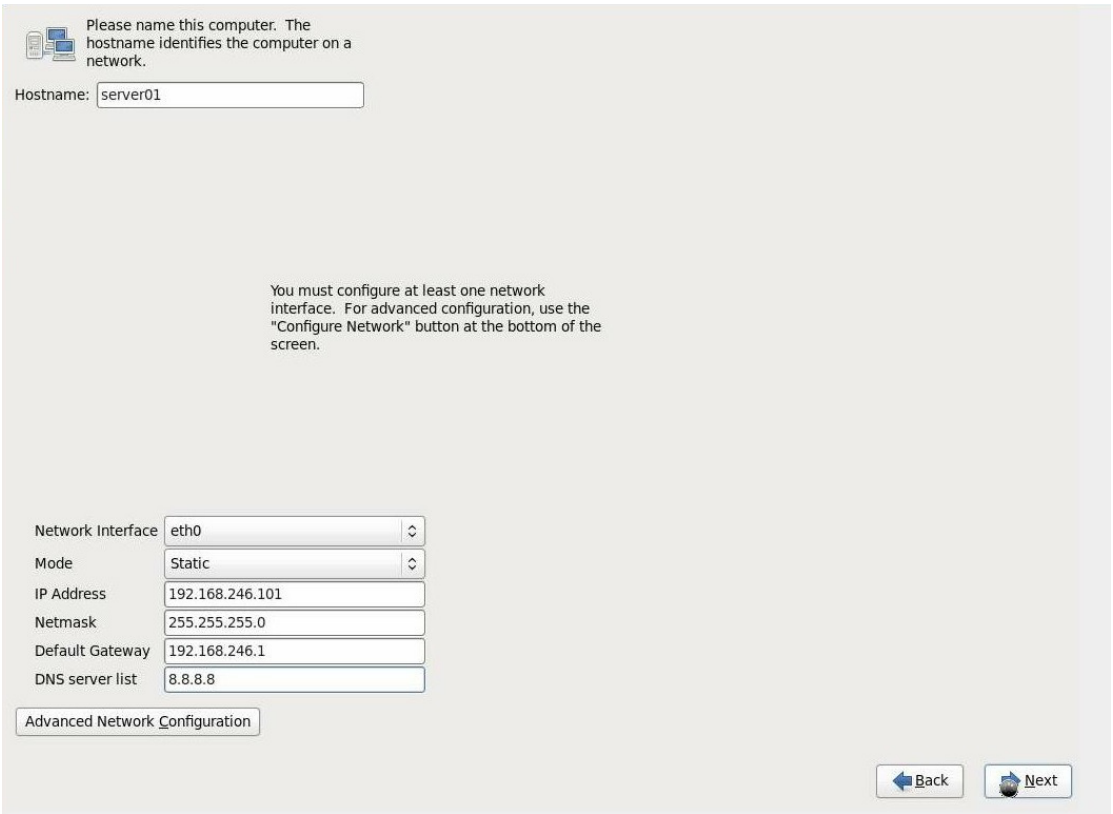

Πατάμε SKIP γιατί η μορφή είναι σε ISO και κατά συνέπεια δεν χρειάζεται έλεγχος. Επιλέγοντας μονάχα τις βασικές επιλογές.



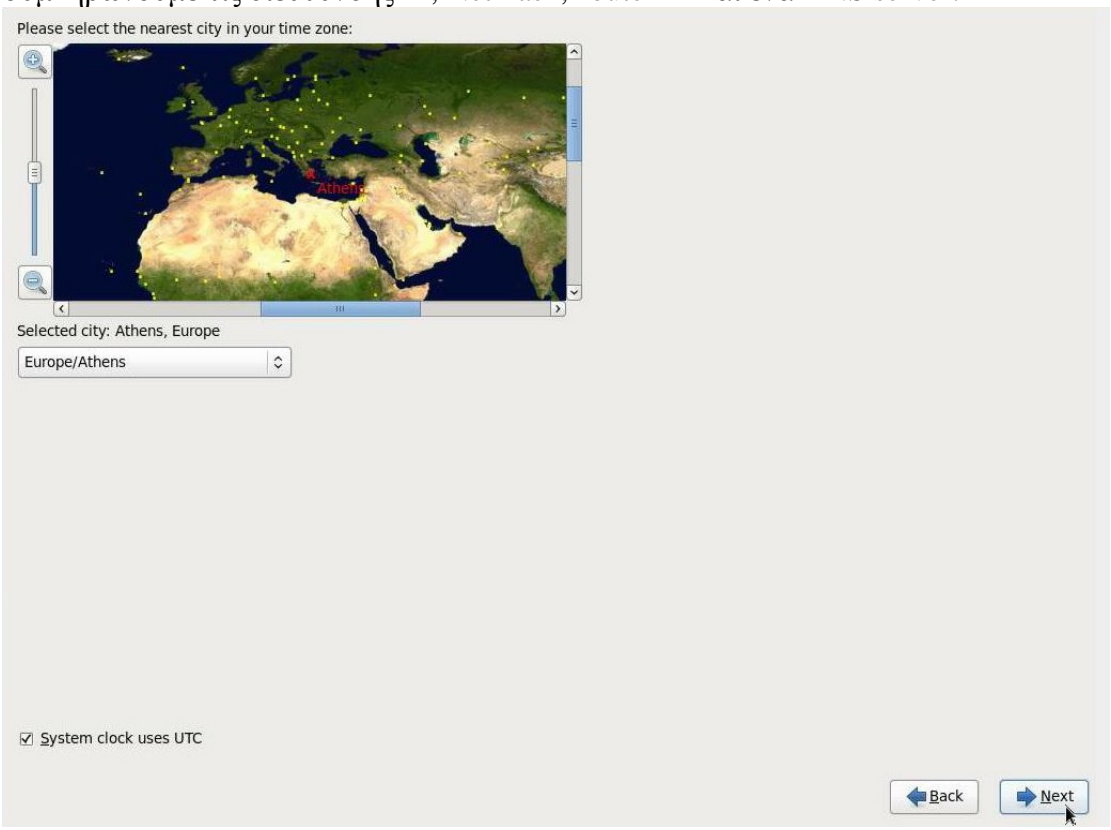

Επιλέγουμε Αγγλικά και πατάμε Next.



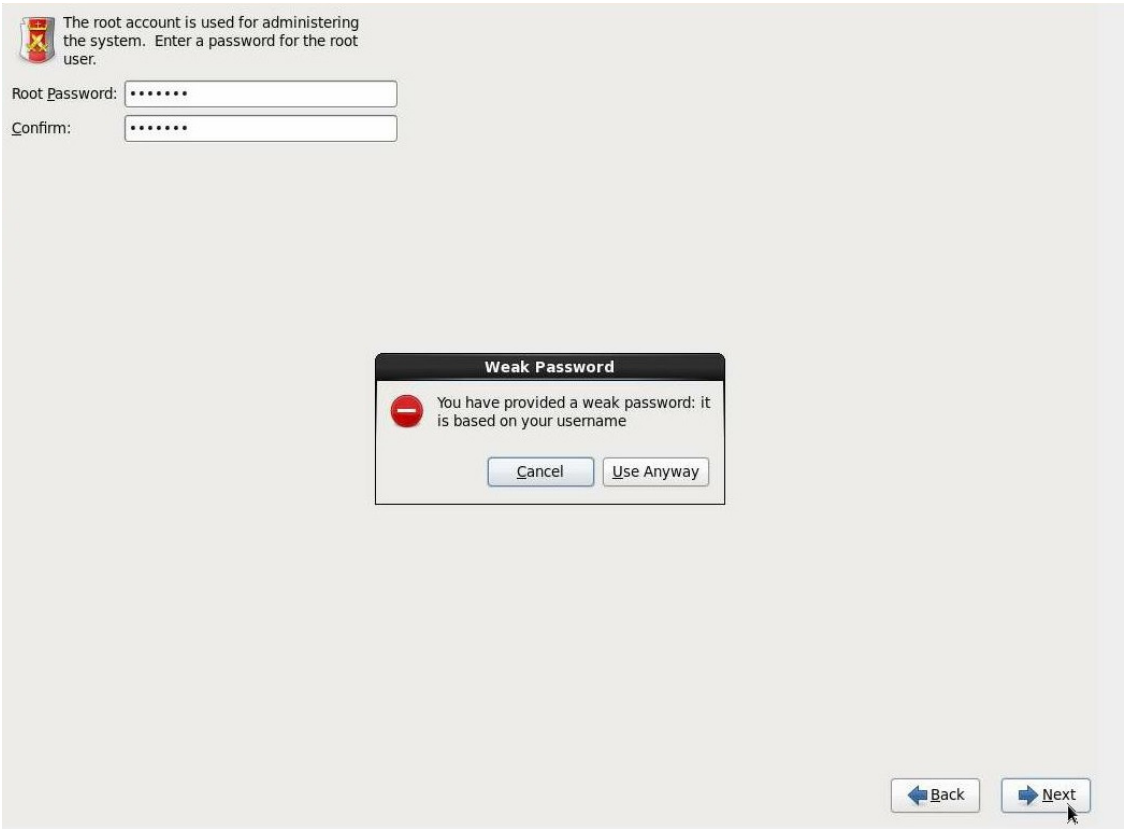

Το ίδιο και εδώ και πατάμε Next.



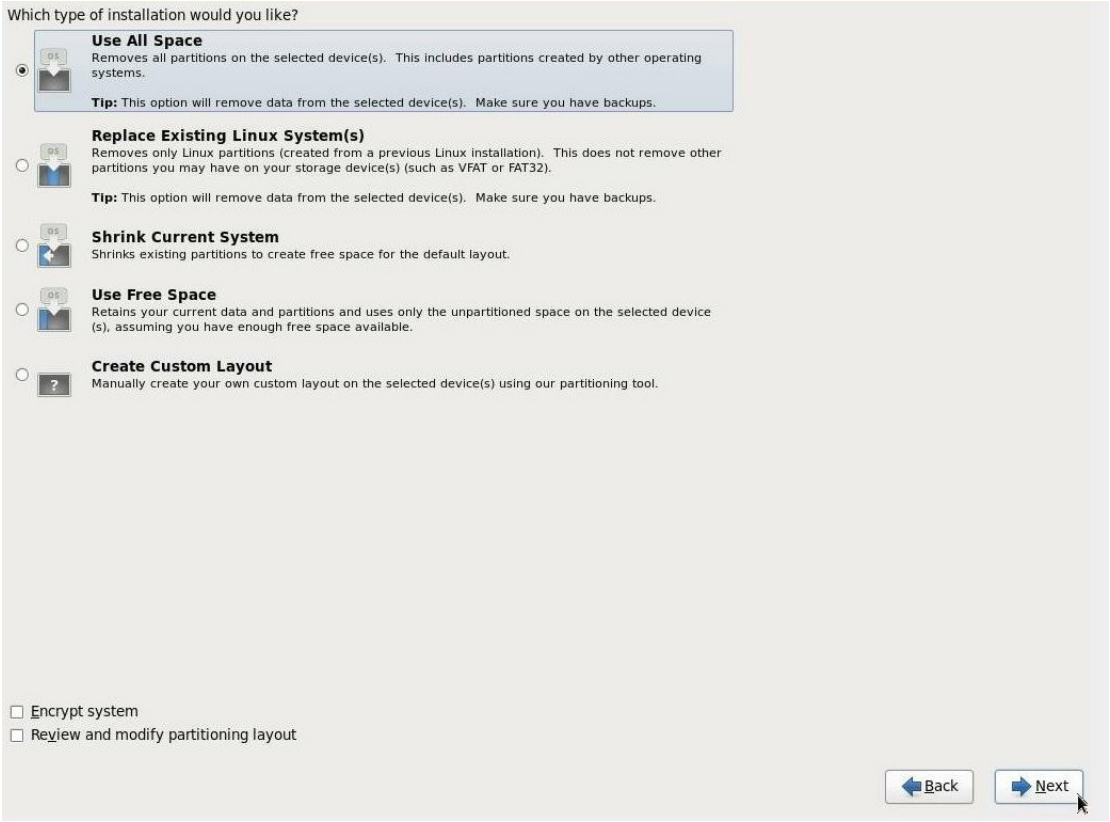
Εδώ χρειάζεται ιδιαίτερη προσοχή! Πρέπει να δώσουμε το HOSTNAME του μηχανήματος και να επιλέξουμε τη κάρτα δικτύου που λειτουργεί και έχει πρόσβαση στο διαδίκτυο. Τέλος συμληρώνουμε τις διευθυνσης IP, Netmask,Router IP και ενα DNS server.




Εδώ επιλέγουμε τη Ζώνη ώρας Europe/Athens




Δηλώνουμε το κωδικό για το λογαριασμό του Διαχειριστή (root)




Επιλέγουμε χρήση όλου του δίσκου.


Επιλέγουμε Reboot και Τέλος με την εγκατάσταση.

## 17.1. ΡΥΘΜΙΣΕΙΣ NICS (παράδειγμα blade01)

```
DEVICE=bond0
IPADDR=192.168.246.101
NETMASK=255.255.255.0
NETWORK=192.168.246.0
BROADCAST=192.168.246.25
5
GATEWAY=192.168.246.1
DNS1=8.8.8.8
DNS2=8.8.4.4
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
NM_CONTROLLED=no
BONDING_OPTS="mode=1
miimon=500"
BONDING_MASTER=yes
BONDING_SLAVE1=eth2
BONDING_SLAVE2=eth3
```

```
DEVICE=bond1
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
NM_CONTROLLED=no
BONDING_OPTS="mode=1
miimon=500"
BONDING_MASTER=yes
BONDING_SLAVE1=eth4
BONDING_SLAVE2=eth5
```

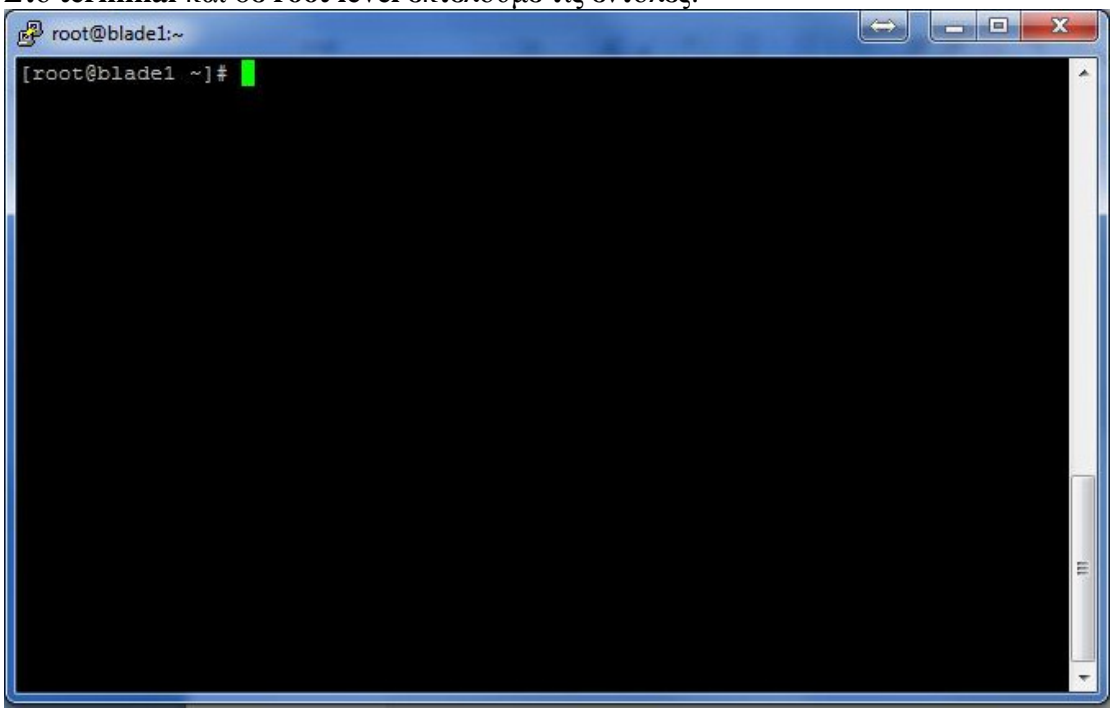
```
DEVICE=eth2
BOOTPROTO=none
ONBOOT=yes
TYPE=Ethernet
MASTER=bond0
SLAVE=yes
NM_CONTROLLED=no
```

```
DEVICE=eth3
BOOTPROTO=none
ONBOOT=yes
TYPE=Ethernet
MASTER=bond0
SLAVE=yes
NM_CONTROLLED=no
```

```
DEVICE=eth4
BOOTPROTO=none
ONBOOT=yes
TYPE=Ethernet
MASTER=bond1
SLAVE=yes
NM_CONTROLLED=no
```

```
DEVICE=eth5
BOOTPROTO=none
ONBOOT=yes
TYPE=Ethernet
MASTER=bond1
SLAVE=yes
NM_CONTROLLED=no
```

Εφόσον έχουμε ολοκληρώσει τις αρχικές ρυθμίσεις του λειτουργικού όπως Networking
Στο **terminal** και σε **root level** εκτελούμε τις εντολές:




**#subscription-manager register --proxy=ip-proxy:port**
**#subscription-manager attach -- auto --proxy=ip-proxy:port**

Set selinux off
**#setenforce 0**
**#vi /etc/selinux/config {disable}**
Set iptables off
**#service iptables stop**
**#chkconfig iptables off**
----------REBOOT----------
NTPD
**# chkconfig ntpd on**
**# cat "ip ntp server" >> /etc/ntpd.conf**
**# ntpdate –u "ip ntp server"**
**# hwclock –s**
**#vi /etc/hosts** (Εισάγουμε όλα τα hostname με τις ip που θα είναι στο Cluster)
**#vi /etc/yum.repo/redhat.repo**
Στο optional-rpms -> **enable=1**
**#wget** κατεβάζουμε το EPEL Repo
"epel (http://dl.fedoraproject.org/pub/epel/6/x86_64/repoview/)"
**#rpm –Uvh epel……..** Εγκατάσταση του EPEL Repo
Εγκαθιστάμε νέες αναβαθμίσεις στους κόμβους.
**#yum –y update**
----------REBOOT----------

**# yum install –y ntpd ntpdate ruby rubygem-json rubygem-nokogiri rubygem-rack rubygem-sequel rubygem-sinatra rubygem-sqlite3-ruby rubygem-thin rubygem-uuidtools rubygems genisoimage log4cpp xmlrpc3-server xmlrpc3-client xmlrpc-c-c++ xmlrpc-c-client++ qemu-img genisoimage log4cpp xmlrpc3-server xmlrpc3-client xmlrpc-c-c++ xmlrpc-c-client++**
Κατεβάζουμε το Gzip πακέτο με RPM για το distro μας.
Το αποσυμπιέζουμε και το μεταφέρουμε σε όλους τους κόμβους.

## Worker Nodes:

Εκτός από τα τις προηγούμενες ενέργειες για τους worker nodes κάνουμε τα εξής:

Εγκατάσταση των απαραίτητων λογισμικών
**#yum –y install libvirt qemu-kvm ruby**
Αλλαγή ρυθμίσεων:
**# vi /etc/libvirt/libvirtd.conf**
**listen_tls=0**
**listen_tcp=1**

**# vi /etc/sysconfig/libvirtd**
**LIBVIRTD_ARGS="--listen"**
Ορισμός του libvirtd να ξεκινάει κατά την εκκίνηση
**#chkconfig libvirtd on**
Ενεργοποίηση του libvirt daemon
**#service libvirtd restart**
Παραμετροποιουμε το /etc/fstab ώστε να κάνει mount το κοινόχρηστο φάκελο κατά την εκκίνηση:
**# vi /etc/fstab**
**/dev/mapper/mpatha /var/lib/one gfs2 defaults,noatime,nodiratime 0 0**

Εγκατάσταση του OpenNebula:
**# rpm –ivh opennebula-common***
**# rpm –ivh opennebula-node***

Διορθώνουμε το path του brctl
**#vi /var/lib/one/remotes/vnm/OpenNebulaNetwork.rb**
**:brctl => "sudo /usr/sbin/brctl",**

Καταχωρούμε στους **sudoers** το χρήστη **oneadmin**
Βάζουμε σχόλιο στο
**#Defaults requiretty**
Εισάγουμε
**oneadmin ALL=(ALL) NOPASSWD: ALL**

## Head Nodes:

Εκτός από τα τις προηγούμενες ενέργειες για τους Head nodes κάνουμε τα εξής:
**#rpm –ivh opennebula***

Ενεργοποιούμε τα services κατά την εκκίνηση
**# chkconfig opennebula on**
**# chkconfig opennebula-sunstone on**

**#vi /etc/one/oned.conf**

```
#*******************************************************************************
#                    OpenNebula Configuration file
#*******************************************************************************

#*******************************************************************************
# Daemon configuration attributes
#-------------------------------------------------------------------------------
#  MANAGER_TIMER: Time in seconds the core uses to evaluate periodical functions.
#  MONITORING_INTERVAL cannot have a smaller value than MANAGER_TIMER.
#
#  MONITORING_INTERVAL: Time in seconds between host and VM monitorization.
#
#  HOST_PER_INTERVAL: Number of hosts monitored in each interval.
#  HOST_MONITORING_EXPIRATION_TIME: Time, in seconds, to expire monitoring
#  information. Use 0 to disable HOST monitoring recording.
#
#  VM_PER_INTERVAL: Number of VMs monitored in each interval.
#  VM_MONITORING_EXPIRATION_TIME: Time, in seconds, to expire monitoring
#  information. Use 0 to disable VM monitoring recording.
#
#  SCRIPTS_REMOTE_DIR: Remote path to store the monitoring and VM management
#  scripts.
#
#  PORT: Port where oned will listen for xmlrpc calls.
#
#  DB: Configuration attributes for the database backend
#   backend : can be sqlite or mysql (default is sqlite)
#   server  : (mysql) host name or an IP address for the MySQL server
#   port    : (mysql) port for the connection to the server.
#                 If set to 0, the default port is used.
#   user    : (mysql) user's MySQL login ID
#   passwd  : (mysql) the password for user
#   db_name : (mysql) the database name
#
#  VNC_BASE_PORT: VNC ports for VMs can be automatically set to VNC_BASE_PORT +
#  VMID
#
#  LOG: Configuration for the logging system
#   system: defines the logging system:
#      file      to log in the oned.log file
#      syslog    to use the syslog facilities
#   debug_level: 0 = ERROR, 1 = WARNING, 2 = INFO, 3 = DEBUG
#
```

```
#  VM_SUBMIT_ON_HOLD: Forces VMs to be created on hold state instead of pending.
#  Values: YES or NO.
#*******************************************************************************

LOG = [
  system      = "file",
  debug_level = 3
]

MANAGER_TIMER = 5

MONITORING_INTERVAL             = 5

#HOST_PER_INTERVAL              = 15
HOST_MONITORING_EXPIRATION_TIME = 10

#VM_PER_INTERVAL              = 5
#VM_MONITORING_EXPIRATION_TIME = 10

SCRIPTS_REMOTE_DIR=/var/lib/one/tmp

PORT = 2633

#DB = [ backend = "sqlite" ]

# Sample configuration for MySQL
 DB = [ backend = "mysql",
        server  = "localhost",
        port    = 0,
        user    = "oneadmin",
        passwd  = "oneadmin",
        db_name = "opennebula" ]

VNC_BASE_PORT = 5900

#VM_SUBMIT_ON_HOLD = "NO"

#*******************************************************************************
# Physical Networks configuration
#*******************************************************************************
#  NETWORK_SIZE: Here you can define the default size for the virtual networks
#
#  MAC_PREFIX: Default MAC prefix to be used to create the auto-generated MAC
#  addresses is defined here (this can be overrided by the Virtual Network
#  template)
#*******************************************************************************

NETWORK_SIZE = 254

MAC_PREFIX   = "02:00"

#*******************************************************************************
# DataStore Configuration
#*******************************************************************************
```

```
#  DATASTORE_LOCATION: *Default* Path for Datastores in the hosts. It IS the
#  same for all the hosts in the cluster. DATASTORE_LOCATION IS ONLY FOR THE
#  HOSTS AND *NOT* THE FRONT-END. It defaults to /var/lib/one/datastores (or
#  $ONE_LOCATION/var/datastores in self-contained mode)
#
#  You can define a different DATASTORE_LOCATION in each cluster by updating
#  its properties with onecluster update.
#
#  DATASTORE_CAPACITY_CHECK: Checks that there is enough capacity before
#  creating a new imag. Defaults to Yes
#
#  DEFAULT_IMAGE_TYPE: This can take values
#       OS        Image file holding an operating system
#       CDROM     Image file holding a CDROM
#       DATABLOCK Image file holding a datablock,
#                 always created as an empty block
#  DEFAULT_DEVICE_PREFIX: This can be set to
#       hd        IDE prefix
#       sd        SCSI
#       xvd       XEN Virtual Disk
#       vd        KVM virtual disk
#*******************************************************************************

#DATASTORE_LOCATION = /var/lib/one/datastores

DATASTORE_CAPACITY_CHECK = "yes"

DEFAULT_IMAGE_TYPE    = "OS"
DEFAULT_DEVICE_PREFIX = "hd"


#*******************************************************************************
# Information Driver Configuration
#*******************************************************************************
# You can add more information managers with different configurations but make
# sure it has different names.
#
#   name      : name for this information manager
#
#   executable: path of the information driver executable, can be an
#               absolute path or relative to $ONE_LOCATION/lib/mads (or
#               /usr/lib/one/mads/ if OpenNebula was installed in /)
#
#   arguments : for the driver executable, usually a probe configuration file,
#               can be an absolute path or relative to $ONE_LOCATION/etc (or
#               /etc/one/ if OpenNebula was installed in /)
#*******************************************************************************

#-------------------------------------------------------------------------------
#  KVM Information Driver Manager Configuration
#    -r number of retries when monitoring a host
#    -t number of threads, i.e. number of hosts monitored at the same time
#-------------------------------------------------------------------------------
IM_MAD = [
```

```
    name       = "kvm",
    executable = "one_im_ssh",
    arguments  = "-r 0 -t 15 kvm" ]
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#  XEN Information Driver Manager Configuration
#    -r number of retries when monitoring a host
#    -t number of threads, i.e. number of hosts monitored at the same time
#-------------------------------------------------------------------------------

# Driver for Xen 3.x
#IM_MAD = [
#    name       = "xen",
#    executable = "one_im_ssh",
#    arguments  = "xen3" ]

# Driver for Xen 4.x
#IM_MAD = [
#    name       = "xen",
#    executable = "one_im_ssh",
#    arguments  = "xen4" ]

#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#  VMware Information Driver Manager Configuration
#    -r number of retries when monitoring a host
#    -t number of threads, i.e. number of hosts monitored at the same time
#-------------------------------------------------------------------------------
#IM_MAD = [
#      name       = "vmware",
#      executable = "one_im_sh",
#      arguments  = "-c -t 15 -r 0 vmware" ]
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#  EC2 Information Driver Manager Configuration
#-------------------------------------------------------------------------------
#IM_MAD = [
#      name       = "ec2",
#      executable = "one_im_ec2",
#      arguments  = "im_ec2/im_ec2.conf" ]
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#  Ganglia Information Driver Manager Configuration
#-------------------------------------------------------------------------------
#IM_MAD = [
#      name       = "ganglia",
#      executable = "one_im_sh",
#      arguments  = "ganglia" ]
#-------------------------------------------------------------------------------
```

```
#-------------------------------------------------------------------------------
#  Dummy Information Driver Manager Configuration
#-------------------------------------------------------------------------------
#IM_MAD = [ name="dummy", executable="one_im_dummy"]
#-------------------------------------------------------------------------------

#*******************************************************************************
# Virtualization Driver Configuration
#*******************************************************************************
# You can add more virtualization managers with different configurations but
# make sure it has different names.
#
#   name      : name of the virtual machine manager driver
#
#   executable: path of the virtualization driver executable, can be an
#               absolute path or relative to $ONE_LOCATION/lib/mads (or
#               /usr/lib/one/mads/ if OpenNebula was installed in /)
#
#   arguments : for the driver executable
#
#   default   : default values and configuration parameters for the driver, can
#               be an absolute path or relative to $ONE_LOCATION/etc (or
#               /etc/one/ if OpenNebula was installed in /)
#
#   type      : driver type, supported drivers: xen, kvm, xml
#*******************************************************************************

#-------------------------------------------------------------------------------
#  KVM Virtualization Driver Manager Configuration
#    -r number of retries when monitoring a host
#    -t number of threads, i.e. number of hosts monitored at the same time
#    -l <actions[=command_name]> actions executed locally, command can be
#        overridden for each action.
#        Valid actions: deploy, shutdown, cancel, save, restore, migrate, poll
#        An example: "-l migrate,poll=poll_ganglia,save"
#
#  Note: You can use type = "qemu" to use qemu emulated guests, e.g. if your
#  CPU does not have virtualization extensions or use nested Qemu-KVM hosts
#-------------------------------------------------------------------------------
VM_MAD = [
    name       = "kvm",
    executable = "one_vmm_exec",
    arguments  = "-t 15 -r 0 kvm",
    default    = "vmm_exec/vmm_exec_kvm.conf",
    type       = "kvm" ]
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#  XEN Virtualization Driver Manager Configuration
#    -r number of retries when monitoring a host
#    -t number of threads, i.e. number of hosts monitored at the same time
#    -l <actions[=command_name]> actions executed locally, command can be
#        overridden for each action.
#        Valid actions: deploy, shutdown, cancel, save, restore, migrate, poll
```

```
#       An example: "-l migrate,poll=poll_ganglia,save"
#-------------------------------------------------------------------------------

# Driver for Xen 3.x
#VM_MAD = [
#    name       = "xen",
#    executable = "one_vmm_exec",
#    arguments  = "-t 15 -r 0 xen3",
#    default    = "vmm_exec/vmm_exec_xen3.conf",
#    type       = "xen" ]

# Driver for Xen 4.x
#VM_MAD = [
#    name       = "xen",
#    executable = "one_vmm_exec",
#    arguments  = "-t 15 -r 0 xen4",
#    default    = "vmm_exec/vmm_exec_xen4.conf",
#    type       = "xen" ]

#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#  VMware Virtualization Driver Manager Configuration
#    -r number of retries when monitoring a host
#    -t number of threads, i.e. number of hosts monitored at the same time
#-------------------------------------------------------------------------------
#VM_MAD = [
#    name       = "vmware",
#    executable = "one_vmm_sh",
#    arguments  = "-t 15 -r 0 vmware -s sh",
#    default    = "vmm_exec/vmm_exec_vmware.conf",
#    type       = "vmware" ]
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#  EC2 Virtualization Driver Manager Configuration
#    arguments: default values for the EC2 driver, can be an absolute path or
#               relative to $ONE_LOCATION/etc (or /etc/one/ if OpenNebula was
#               installed in /).
#-------------------------------------------------------------------------------
#VM_MAD = [
#    name       = "ec2",
#    executable = "one_vmm_ec2",
#    arguments  = "vmm_ec2/vmm_ec2.conf",
#    type       = "xml" ]
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#  Dummy Virtualization Driver Configuration
#-------------------------------------------------------------------------------
#VM_MAD = [ name="dummy", executable="one_vmm_dummy", type="xml" ]
#-------------------------------------------------------------------------------

#*******************************************************************************
```

```
# Transfer Manager Driver Configuration
#*******************************************************************************
# You can add more transfer managers with different configurations but make
# sure it has different names.
#   name      : name for this transfer driver
#
#   executable: path of the transfer driver executable, can be an
#               absolute path or relative to $ONE_LOCATION/lib/mads (or
#               /usr/lib/one/mads/ if OpenNebula was installed in /)
#   arguments :
#       -t: number of threads, i.e. number of transfers made at the same time
#       -d: list of transfer drivers separated by commas, if not defined all the
#           drivers available will be enabled
#*******************************************************************************

TM_MAD = [
    executable = "one_tm",
    arguments  = "-t 15 -d dummy,lvm,shared,qcow2,ssh,vmfs,iscsi,ceph" ]

#*******************************************************************************
# Datastore Driver Configuration
#*******************************************************************************
# Drivers to manage the datastores, specialized for the storage backend
#   executable: path of the transfer driver executable, can be an
#               absolute path or relative to $ONE_LOCATION/lib/mads (or
#               /usr/lib/one/mads/ if OpenNebula was installed in /)
#
#   arguments : for the driver executable
#       -t number of threads, i.e. number of repo operations at the same time
#       -d datastore mads separated by commas
#*******************************************************************************

DATASTORE_MAD = [
    executable = "one_datastore",
    arguments  = "-t 15 -d dummy,fs,vmfs,iscsi,lvm,ceph"
]

#*******************************************************************************
# Hook Manager Configuration
#*******************************************************************************
# The Driver (HM_MAD)
# -------------------------------------------
#
# Used to execute the Hooks:
#   executable: path of the hook driver executable, can be an
#               absolute path or relative to $ONE_LOCATION/lib/mads (or
#               /usr/lib/one/mads/ if OpenNebula was installed in /)
#
#   arguments : for the driver executable, can be an absolute path or relative
#               to $ONE_LOCATION/etc (or /etc/one/ if OpenNebula was installed
#               in /)
#
# Virtual Machine Hooks (VM_HOOK)
# -------------------------------
```

```
#
# Defined by:
#   name      : for the hook, useful to track the hook (OPTIONAL)
#   on        : when the hook should be executed,
#               - CREATE, when the VM is created (onevm create)
#               - PROLOG, when the VM is in the prolog state
#               - RUNNING, after the VM is successfully booted
#               - UNKNOWN, when the VM is in the unknown state
#               - SHUTDOWN, after the VM is shutdown
#               - STOP, after the VM is stopped (including VM image transfers)
#               - DONE, after the VM is deleted or shutdown
#               - FAILED, when the VM enters the failed state
#               - CUSTOM, user defined specific STATE and LCM_STATE combination
#                 of states to trigger the hook.
#   command   : path is relative to $ONE_LOCATION/var/remotes/hook
#               (self-contained) or to /var/lib/one/remotes/hook (system-wide).
#               That directory will be copied on the hosts under
#               SCRIPTS_REMOTE_DIR. It can be an absolute path that must exist
#               on the target host
#   arguments : for the hook. You can access to VM information with $
#               - $ID, the ID of the virtual machine
#               - $TEMPLATE, the VM template in xml and base64 encoded
#               - $PREV_STATE, the previous STATE of the Virtual Machine
#               - $PREV_LCM_STATE, the previous LCM STATE of the Virtual Machine
#   remote    : values,
#               - YES, The hook is executed in the host where the VM was
#                 allocated
#               - NO, The hook is executed in the OpenNebula server (default)
#
# Example Virtual Machine Hook
# ----------------------------
#
# VM_HOOK = [
#   name      = "advanced_hook",
#   on        = "CUSTOM",
#   state     = "ACTIVE",
#   lcm_state = "BOOT_UNKNOWN",
#   command   = "log.rb",
#   arguments = "$ID $PREV_STATE $PREV_LCM_STATE" ]
#
# Host Hooks (HOST_HOOK)
# ----------------------------
#
# Defined by:
#   name      : for the hook, useful to track the hook (OPTIONAL)
#   on        : when the hook should be executed,
#               - CREATE, when the Host is created (onehost create)
#               - ERROR, when the Host enters the error state
#               - DISABLE, when the Host is disabled
#   command   : path is relative to $ONE_LOCATION/var/remotes/hook
#               (self-contained) or to /var/lib/one/remotes/hook (system-wide).
#               That directory will be copied on the hosts under
#               SCRIPTS_REMOTE_DIR. It can be an absolute path that must exist
#               on the target host.
```

```
#   arguments : for the hook. You can use the following Host information:
#               - $ID, the ID of the host
#               - $TEMPLATE, the Host template in xml and base64 encoded
#   remote    : values,
#               - YES, The hook is executed in the host
#               - NO, The hook is executed in the OpenNebula server (default)
#
# Virtual Network (VNET_HOOK)
# User (USER_HOOK)
# Group (GROUP_HOOK)
# Image (IMAGE_HOOK)
# ------------------------------
#
# These hooks are executed when one of the referring entities are created or
# removed. Each hook is defined by:
#   name      : for the hook, useful to track the hook (OPTIONAL)
#   on        : when the hook should be executed,
#               - CREATE, when the vnet is created
#               - REMOVE, when the vnet is removed
#   command   : path is relative to $ONE_LOCATION/var/remotes/hook
#               (self-contained) or to /var/lib/one/remotes/hook (system-wide).
#               That directory will be copied on the hosts under
#               SCRIPTS_REMOTE_DIR. It can be an absolute path that must exist
#               on the target host.
#   arguments : for the hook. You can use the following Host information:
#               - $ID, the ID of the host
#               - $TEMPLATE, the vnet template in xml and base64 encoded
#-------------------------------------------------------------------------------
HM_MAD = [
    executable = "one_hm" ]

#*******************************************************************************
# Fault Tolerance Hooks
#*******************************************************************************
# This hook is used to perform recovery actions when a host fails.
# Script to implement host failure tolerance
#   It can be set to
#           -r recreate VMs running in the host
#           -d delete VMs running in the host
#   Additional flags
#           -f force resubmission of suspended VMs
#           -p <n> avoid resubmission if host comes
#                  back after n monitoring cycles
#*******************************************************************************
#
#HOST_HOOK = [
#    name      = "error",
#    on        = "ERROR",
#    command   = "ft/host_error.rb",
#    arguments = "$ID -r",
#    remote    = "no" ]
#-------------------------------------------------------------------------------
# These two hooks can be used to automatically delete or resubmit VMs that reach
# the "failed" state. This way, the administrator doesn't have to interact
```

```
# manually to release its resources or retry the deployment.
#
#
# Only one of them should be uncommented.
#-------------------------------------------------------------------------------
#
#VM_HOOK = [
#   name      = "on_failure_delete",
#   on        = "FAILED",
#   command   = "/usr/bin/env onevm delete",
#   arguments = "$ID" ]
#
VM_HOOK = [
  name      = "on_failure_recreate",
  on        = "FAILED",
  command   = "/usr/bin/env onevm delete --recreate",
  arguments = "$ID" ]
VM_HOOK = [
  name      = "on_failure_recreate",
  on        = "UNKNOWN",
  command   = "/usr/bin/env onevm delete --recreate",
  arguments = "$ID" ]
#-------------------------------------------------------------------------------

#*******************************************************************************
# Auth Manager Configuration
#*******************************************************************************
# AUTH_MAD: The Driver that will be used to authenticate (authn) and
# authorize (authz) OpenNebula requests. If defined OpenNebula will use the
# built-in auth policies.
#
#   executable: path of the auth driver executable, can be an
#               absolute path or relative to $ONE_LOCATION/lib/mads (or
#               /usr/lib/one/mads/ if OpenNebula was installed in /)
#
#   authn     : list of authentication modules separated by commas, if not
#               defined all the modules available will be enabled
#   authz     : list of authentication modules separated by commas
#
# SESSION_EXPIRATION_TIME: Time in seconds to keep an authenticated token as
# valid. During this time, the driver is not used. Use 0 to disable session
# caching
#
# ENABLE_OTHER_PERMISSIONS: Whether or not users can set the permissions for
# 'other', so publishing or sharing resources with others. Users in the oneadmin
# group will still be able to change these permissions. Values: YES or NO.
#
# DEFAULT_UMASK: Similar to Unix umask, sets the default resources permissions.
# Its format must be 3 octal digits. For example a umask of 137 will set
# the new object's permissions to 640 "um- u-- ---"
#*******************************************************************************

AUTH_MAD = [
    executable = "one_auth_mad",
```

```
    authn = "ssh,x509,ldap,server_cipher,server_x509"]

SESSION_EXPIRATION_TIME = 900

#ENABLE_OTHER_PERMISSIONS = "YES"

DEFAULT_UMASK = 177

#*******************************************************************************
# Restricted Attributes Configuration
#*******************************************************************************
# The following attributes are restricted to users outside the oneadmin group
#*******************************************************************************

VM_RESTRICTED_ATTR = "CONTEXT/FILES"
VM_RESTRICTED_ATTR = "NIC/MAC"
VM_RESTRICTED_ATTR = "NIC/VLAN_ID"

#VM_RESTRICTED_ATTR = "RANK"
#VM_RESTRICTED_ATTR = "SCHED_RANK"
#VM_RESTRICTED_ATTR = "REQUIREMENTS"
#VM_RESTRICTED_ATTR = "SCHED_REQUIREMENTS"

IMAGE_RESTRICTED_ATTR = "SOURCE"

#*******************************************************************************
# OneGate
#*******************************************************************************

#ONEGATE_ENDPOINT = "http://frontend:5030"
```

Ενεργοποιούμε τις υπηρεσίες

**#service opennebula start**
**#service opennebula-sunstone start**

## 14.2 ΕΓΚΑΤΑΣΤΑΣΗ CLUSTER

Σε όλους τους κόμβους εγκαθιστούμε τα παρακάτω πακέτα (πλην του luci που για καθαρά διαχειριστικούς λόγους το εγκαθιστάμε μόνο σε Head Nodes)
#yum -y install corosync gfs2-utils cman rgmanager luci ricci modclusterd clvmd

Εδώ είναι το γενικό config του Cluster το οποίο πρέπει να υπάρχει σε όλους τους κόμβους.
#vi /etc/cluster/cluster.conf

```
########################################################################
<?xml version="1.0"?>
<cluster config_version="48" name="Idika_Cloud">
    <clusternodes>
        <clusternode name="blade1" nodeid="1" weight="1">
            <fence>
                <method name="Method">
                    <device name="Blade1" port="1"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="blade2" nodeid="2">
            <fence>
                <method name="Method">
                    <device name="Blade1" port="2"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="blade3" nodeid="3">
            <fence>
                <method name="Method">
                    <device name="Blade1" port="3"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="blade4" nodeid="4">
            <fence>
                <method name="Method">
                    <device name="Blade1" port="4"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="blade5" nodeid="5">
            <fence>
                <method name="Method">
                    <device name="Blade1" port="5"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="blade15" nodeid="6">
```

```
                <fence>
                    <method name="Blade2">
                        <device name="Blade2" port="1"/>
                    </method>
                </fence>
            </clusternode>
            <clusternode name="blade16" nodeid="7">
                <fence>
                    <method name="Blade2">
                        <device name="Blade2" port="2"/>
                    </method>
                </fence>
            </clusternode>
            <clusternode name="blade17" nodeid="8">
                <fence>
                    <method name="Blade2">
                        <device name="Blade2" port="3"/>
                    </method>
                </fence>
            </clusternode>
            <clusternode name="blade18" nodeid="9">
                <fence>
                    <method name="Blade2">
                        <device name="Blade2" port="4"/>
                    </method>
                </fence>
            </clusternode>
            <clusternode name="blade19" nodeid="10">
                <fence>
                    <method name="Blade2">
                        <device name="Blade2" port="5"/>
                    </method>
                </fence>
            </clusternode>
        </clusternodes>
        <cman expected_votes="19" transport="udpu"/>
        <quorumd label="qdisk"/>
        <fencedevices>
            <fencedevice agent="fence_bladecenter" ipaddr="192.168.246.20"
login="USERID" name="Blade1" passwd="PASSW0RD" power_wait="12"/>
            <fencedevice agent="fence_bladecenter" ipaddr="192.168.246.30"
login="USERID" name="Blade2" passwd="PASSW0RD" power_wait="12"/>
        </fencedevices>
        <rm>
            <resources>
                <clusterfs device="550290f4-27f3-1aad-ee0f-eec9be7dc661" fsid="14800"
fstype="gfs2" mountpoint="/mnt" name="Cloud" options="defaults,noatime,nodiratime 0
0"/>
            </resources>
```

```
        </rm>
        <logging>
                <logging_daemon debug="on" name="fenced"/>
        </logging>
</cluster>
```

##########################################################################

Ενεργοποίηση των services κατά την εκκίνηση

**#chkconfig gfs2 on**
**#chkconfig clvm on**
**#chkconfig cman on**
**#chkconfig mysqld on**
**#chkconfig rgmanager on**
**#chkconfig luci on**
**#chkconfig ricci on**

Αλλαγή password του χρήστη ricci σύμφωνα με το cluster.conf

**#passwd ricci**

Εκκίνηση της υπηρεσίας Ricci

**#service ricci restart**

Συγχρονισμός – Μεταφορά του Cluster config σε όλους τους Nodes

**#ccs_sync**

Εκκίνηση των services του cluster

**#service cman restart**
**#service rgmanager restart**
**#service clvm restart**
**#service gfs2 restart**

Από το λογισμικό luci ελέγχουμε ότι το cluster είναι online και επίσης μας ενημερώνει ποιοι κόμβοι συμμετέχουν στο cluster.

**MySQL Database + Replication**

Ακολουθούμε τα βήματα του παρακάτω συνδέσμου.

http://www.tecmint.com/how-to-setup-mysql-master-slave-replication-in-rhel-centos-fedora/

## ΣΥΜΠΕΡΑΣΜΑΤΑ ΓΕΝΙΚΟΥ ΜΕΡΟΥΣ

Οι παράλληλες εφαρμογές σε σχέση με τις ακολουθιακές είναι περισσότερο σύνθετες, τόσο με τις πολλαπλές ροές εντολών που εκτελούνται ταυτόχρονα, όσο και με τις πολλαπλές ομάδες δεδομένων που υφίστανται ταυτόχρονη επεξεργασία. Το κόστος αυτής της πολυπλοκότητας σε σχεδόν όλες τις φάσεις ανάπτυξης της εφαρμογής, αφορά το σχεδιασμό, συγγραφή, εκσφαλμάτωση, βελτιστοποίηση και συντήρηση.

Ιδιαίτερα σημαντική για την ανάπτυξη ενός λογισμικού είναι η τήρηση καλών πρακτικών ανάπτυξης αφού θα χρησιμοποιηθεί και από άλλους χρήστες.

Η φορητότητα παράλληλων εφαρμογών είναι αρκετά μεγάλη, λόγω της προτυποποίησης αρκετών APIs, π.χ. MPI, OpenMP κ.α. Ωστόσο επειδή υπάρχουν διαφορές στην υλοποίησή τους απαιτείται προσαρμογή του κώδικα. Σε κάποιες περιπτώσεις παραμένουν τα συνήθη προβλήματα συμβατότητας, αφού διάφοροι κατασκευαστές προσφέρουν "βελτιωμένες" βιβλιοθήκες με ειδικές συναρτήσεις, μειώνοντας έτσι τη χρήση φορητότητας. Τόσο τα συστήματα εκτέλεσης, όσο και τα λειτουργικά συστήματα επηρεάζουν σε μεγάλο βαθμό τη φορητότητα. Επίσης η αρχιτεκτονική αποτελεί βασικό περιορισμό στη φορητότητα.

# ΒΙΒΛΙΟΓΡΑΦΙΑ


- Asanovic K., Bodik R., Catanzaro B.C., et al., (2006). The landscape of parallel computing research: A view from Berkeley. University of California, Berkeley. Technical Report No. UCB/EECS-2006-183.
- Blelloch G., (1996). Programming parallel algorithms. Comm. ACM. 39(3): 85-97.
- Garland M., et al., (2008). Parallel computing experiences with CUDA. Micro IEEE. 48(4): 13-27.
- Kessler C., (2004). Managing distributed shared arrays in a bulk-synchronous parallel environment. Concurrency - Pract. Exp., 16: 133-153.
- Kessler C., Keller K., (2007). Models for parallel computing: Review and Perspectives. PARS-Mitteilungen. 24: 13-29.
- Marowka A., (2007). Parallel computing on any desktop. Communication of the ACM. 50(9): 75-78.
- Mattson T., (1995). Programming environments for parallel and distributed computing: A comparison of p4, PVM, Linda and TCGMSG," The International Journal of Supercomputing. 9: 138-161.
- Shan H., Singh J.P., (2001). A comparison of MPI, SHMEM, and Cache-Coherent Shared Address Space Programming Models on a Tightly-Coupled Multiprocessor. International Journal of Parallel Programming, 29(3): 283-318.
- Shan H., Singh J.P., (2002). Comparison of Three Programming Models for Adaptive Applications on the Origin 2000. Journal of Parallel and Distributed Computing, 62: 241-266.
- Silva L.M., Buyya R., (1999). Parallel programming models and paradigms. Cluster comuting. 2: 4-27.

## ΙΣΤΟΓΡΑΦΙΑ


- http://en.wikipedia.org/wiki/Automatic_parallelization
- http://en.wikipedia.org/wiki/Beowulf_%28computing%29
- http://en.wikipedia.org/wiki/Blade_server
- http://en.wikipedia.org/wiki/Cloud_computing
- http://en.wikipedia.org/wiki/Computer_cluster
- http://en.wikipedia.org/wiki/CUDA
- http://en.wikipedia.org/wiki/GPGPU
- http://en.wikipedia.org/wiki/Grid_computing
- http://en.wikipedia.org/wiki/InfiniBand
- http://en.wikipedia.org/wiki/Message_Passing_Interface
- http://en.wikipedia.org/wiki/Parallel_computing
- http://en.wikipedia.org/wiki/Symmetric_multiprocessing
- http://www.grnet.gr/default.asp?pid=89&la=1
- http://www.hellasgrid.gr/
- http://www.shocksolution.com/2012/12/installing-and-configuring-infiniband-on-a-red-hat-system/
- http://people.sc.fsu.edu/~jburkardt/cpp_src/prime_mpi/prime_mpi.html