ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Τμήμα Ψηφιακών Συστημάτων

ΠΜΣ:ΜΗΧΑΝΙΚΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΔΙΑΔΙΚΤΥΑΚΕΣ ΚΑΙ ΦΟΡΗΤΕΣ ΕΦΑΡΜΟΓΕΣ

ΕΙΚΟΝΙΚΕΣ ΤΟΠΟΛΟΓΙΕΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗ ΑΡΧΕΙΩΝ ΜΕ MPI

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΚΩΤΟΥΛΑΣ ΕΥΑΓΓΕΛΟΣ (ΑΜ: Μ013121015)

Επιβλέπων: ***ΗΛΙΑΣ Κ. ΣΑΒΒΑΣ, ΚΑΘΗΓΗΤΗΣ***

ΛΑΡΙΣΑ, ΙΟΥΛΙΟΣ 2022

UNIVERSITY OF THESSALY

Digital Systems Department

Postgraduate: Software Engineering for Web and Mobile Application

Virtual Topologies and File Management in MPI

POSTGRADUATE WORK

EVANGELOS KOTOULAS (RN: M013121015)

Supervisor: ***Hlias Savvas, Professor***

LARISSA, July 2022

Υπεύθυνη Δήλωση περί Ακαδημαϊκής Δεοντολογίας και Πνευματικών Δικαιωμάτων

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα μεταπτυχιακή εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον, και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών

(Υπογραφή)

Ονοματεπώνυμο Φοιτητή

Κωτούλας Ευάγγελος

Ημερομηνία

18/07/2022

Εγκρίνεται από την Επιτροπή Εξέτασης:

Επιβλέπων/πουσα	**Ονοματεπώνυμο Επιβλέποντα** Βαθμίδα/ιδιότητα επιβλέποντα, Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Θεσσαλίας
Μέλος	**Ονοματεπώνυμο Μέλους 1** Βαθμίδα/ιδιότητα μέλους 1, Τμήμα/Ιδρυμα μέλους 1
Μέλος	**Ονοματεπώνυμο Μέλους 2** Βαθμίδα/ιδιότητα μέλους 2, Τμήμα/Ιδρυμα μέλους 2

Ημερομηνία έγκρισης: dd-mm-yyyy

Περίληψη

Η συγκεκριμένη μεταπτυχιακή εργασία έχει ως στόχο την ανάλυση των εικονικών τοπολογιών και την διαχείριση αρχείων σχετικά με τον παράλληλο προγραμματισμό στο πρότυπο MPI(Message Passing Interface). Το κύριο θέμα είναι η ανάλυση των συναρτήσεων των εικονικών τοπολογιών και επίσης η ανάλυση των συναρτήσεων της διαχείρισης αρχείων σύμφωνα με το πρότυπο του MPI. Επιπλέον στο θέμα της διαχείρισης αρχείων θα αναλύσουμε εκείνες τις συναρτήσεις που επιτρέπουν την ανάγνωση και εγγραφή που επιτρέπουν την μεταφορά των δεδομένων ανάμεσα στα αρχεία του συστήματος και στις διεργασίες των εφαρμογών του MPI. Υπάρχουν τρείς διαφορετικές παράμετροι: ανάλογα με τον τρόπο καθορισμού της θέσης εντός του αρχείου, ανάλογα με το χαρακτηριστικό της συνάρτησης (παρεμποδιστική η μη παρεμποδιστική), ανάλογα με την χρήση της συνάρτησης από όλες τις διεργασίες της τρέχουσας ομάδας ή μόνο από την τρέχουσα διεργασία.

Γενικότερα το πρότυπο MPI έχει ως βασική μονάδα την διεργασία. Υπάρχουν διάφορες ομάδες διεργασιών οι οποίες επικοινωνούν μεταξύ τους με την χρήση εξειδικευμένων περιβαλλόντων επικοινωνίας που ονομάζονται communicators.

Η συγκεκριμένη μεταπτυχιακή εργασία χωρίζεται σε τρία κεφάλαια. Στο πρώτο κεφάλαιο παρουσιάζουμε τα βασικά στοιχεία του προτύπου του MPI καθώς και τις βασικές συναρτήσεις. Στο δεύτερο κεφάλαιο αναλύουμε τα βασικά των χαρακτηριστικά των εικονικών τοπολογιών καθώς και τις βασικές συναρτήσεις και παραδείγματα από κώδικα. Τέλος στο τελευταίο κεφάλαιο αναλύουμε τα βασικά χαρακτηριστικά της διαχείρισης αρχείων σύμφωνα με το MPI τις βασικές συναρτήσεις καθώς και παραδείγματα από κώδικα. Τέλος, στο παράρτημα παρουσιάζεται ο τρόπος εγκατάστασης του MPICH 3.2 στο λειτουργικό σύστημα Linux. Τέλος η σημερινή έκδοση του MPICH είναι το 4.1.

Abstract

This master's thesis aims at the analysis of virtual topologies and file management regarding parallel programming in the MPI (Message Passing Interface) standard. The main topic is the analysis of virtual topologies functions and also the analysis of file management functions according to the MPI standard. In addition to the topic of file management we will analyze those functions that allow reading and writing, that allow the transfer of data between system files and MPI application processes. There are three different parameters: depending on how the position within the file is defined, depending on the feature of the function (blocking or non-blocking), depending on whether the function is used by all processes in the current group or only from the current process. More generally, the MPI standard has as its basic unit the process. There are various groups of processes that communicate with each other using specialized communication environments called communicators.
This master thesis is divided into three chapters. In the first chapter we represent the basic elements of the MPI standard as well as the basic functions. In the second chapter we analyze the basic characteristics of virtual topologies as well as the basic functions and examples from code. Finally, in the last chapter we analyze the basic features of file management according to MPI, the basic functions as well as examples from code. Finally, the appendix shows how to install MPICH 3.2 on the Linux operating system. Finally, the current version of MPICH is 4.1.

Ευχαριστίες

Τελειώνοντας την μεταπτυχιακή μου διατριβή θα ήθελα να ευχαριστήσω τον κύριο Ηλία Σάββα που μου εμπιστεύτηκε αυτό το θέμα καθώς και τις πολύτιμες συμβουλές και την βοήθεια που μου παρείχε σε οποιαδήποτε απορία που είχα. Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου για την υπομονή και την στήριξη που μου παρείχε.

Κωτούλας Ευάγγελος

18/07/2022

Περιεχόμενα

Πίνακας Εικόνων

Περιεχόμενα Πινάκων:

1 Εισαγωγή

Το MPI είναι ένα προγραμματιστικό μοντέλο μεταβίβασης μηνυμάτων που χρησιμοποιείται για την ανάπτυξη παράλληλων εφαρμογών σε παράλληλους υπολογιστές κατανεμημένης μνήμης (MIMD). Ορίζει τους κανόνες επικοινωνίας των διεργασιών μέσω κατάλληλα διαμορφωμένων μηνυμάτων και περιλαμβάνει μια βιβλιοθήκη συναρτήσεων που χρησιμοποιούνται σε εφαρμογές που είναι γραμμένα σε γλώσσες προγραμματισμού C, C++ Fortran.

Στο σειριακό μοντέλο προγραμματισμού πραγματοποιείται η χρήση ενός και μοναδικού επεξεργαστή σε αντίθεση με το παράλληλο μοντέλο όπου υπάρχει η χρήση πολλών διαφορετικών επεξεργαστών που χρησιμοποιεί την δική του περιοχή μνήμης και τις δικές του μεταβλητές του προγράμματος.

1.1 Ιστορική αναδρομή του MPI

Ξεκίνησε το 1992 στο συνέδριο Workshop on Standards for Message Passing Interface in a Distributed Memory Environment που πραγματοποιήθηκε στις 29-30 Απριλίου του ίδιου έτους. Ο στόχος αυτού του συνεδρίου ήταν η ανταλλαγή απόψεων σχετικά με τα βασικά του χαρακτηριστικά ενός νέου αναθεωρημένου προγραμματιστικού μοντέλου που στηρίζονταν στα ήδη υπάρχοντα μοντέλα όπως είναι το PVM, το Express, το Parmacs, το PVX/2 και το p4. Με την πρώτη διατύπωση των κανόνων του προτύπου το συνέδριο αυτό δημιούργησε μια ομάδα εργασίας που είναι γνωστή και ως MPI Forum όπου όριζε τις θεματικές ενότητες εργασίας και τα πρόσωπα που θα δούλευαν πάνω σε αυτές. Η τελική έκδοση του προτύπου παρουσιάστηκε το 1994 και αφορούσε την πρώτη έκδοση του προτύπου που είναι το MPI 1.0.

1.2 Βασικοί στόχοι του MPI

Το MPI δεν σχεδιάστηκε ως ένα κατανεμημένο λειτουργικό σύστημα αλλά ως βιβλιοθήκη συναρτήσεων που επιτρέπουν την ανάπτυξη παράλληλων εφαρμογών. Παρόλα αυτά επιτρέπει την πραγματοποίηση κάποιων διαδικασιών ενός λειτουργικού συστήματος όπως είναι η δυναμική διαχείριση διεργασιών. Όμως με την ταχεία εξέλιξη της τε-

χνολογίας των παράλληλων αρχιτεκτονικών η σχεδίαση του προτύπου γίνεται με τέτοιο τρόπο ώστε να χαρακτηρίζεται από εύκολες διαδικασίες επέκτασης και παραμετροποίησης που παραπέμπει σε μία δομημένη αρχιτεκτονική που έχει ως στόχο να καλύψει ανάγκες που θα προκύψουν στο μέλλον. Επίσης θα πρέπει να υποστηρίζει την ανάπτυξη εφαρμογών υψηλής απόδοσης που μπορούν να εκτελεστούν σε μία ποικιλία παράλληλων συστημάτων. Και τέλος ο πιο σημαντικός στόχος του MPI είναι η δυνατότητα εκτέλεσης σε ετερογενή συστήματα δηλαδή συστήματα που αποτελούνται από υπολογιστικούς κόμβους διαφορετικού τύπου.

1.3 Σημαντικά χαρακτηριστικά του προτύπου MPI 1.0

- **Επικοινωνίες από σημείο σε σημείο**: επιτρέπουν την ανταλλαγή δεδομένων ανάμεσα σε 2 διεργασίες όπου η πρώτη διεργασία είναι η διεργασία αποστολέας και η δεύτερη διεργασία είναι η διεργασία παραλήπτης.
- **Συλλογικές Επικοινωνίες**: συμμετέχουν όλες οι επικοινωνίες που ανήκουν σε κάποια ομάδα διεργασιών. Τυπικές μορφές είναι η εκμπομπή, η συλλογή, η διασπορά και ο υποβιβασμός των δεδομένων.
- **Ομάδες διεργασιών**: Το MPI επιτρέπει την ομαδοποίηση των διεργασιών σε μία η περισσότερες ομάδες όπου οι ομάδες αυτές είτε μπορούν να συνεργαστούν μεταξύ τους ανταλλάσσοντας δεδομένα η να λειτουργήσουν ανεξάρτητα η μία από την άλλη.
- **Περιβάλλοντα Επικοινωνίας**: η διακίνηση των δεδομένων στηρίζεται στην χρήση του κατάλληλου περιβάλλοντος επικοινωνίας και σε ένα ειδικό αντικείμενο που ονομάζεται communicator.Υπάρχουν 2 είδη ειδικών αντικειμένων. Το πρώτο είναι ο intracommunicator όπου επιτρέπουν την επικοινωνία των διεργασιών της ίδιας ομάδας. Το δεύτερο είναι ο intercommunicator όπου επιτρέπουν την ανταλλαγή δεδομένων σε διεργασίες που ανήκουν σε διαφορετικές ομάδες.
- **Τοπολογίες διεργασιών**: Οι διεργασίες της εφαρμογής οργανώνονται σε κατάλληλα διαμορφωμένες τοπολογίες για να επιλύσουν το πρόβλημα με αποδοτικότερο τρόπο. Υπάρχουν 2 είδη τοπολογιών. Η πρώτη τοπολογία είναι η τοπολογία γραφήματος όπου οι διεργασίες της εφαρμογής γίνονται με αυθαίρετο τρόπο. Η δεύτερη είναι οι καρτεσιανές τοπολογίες όπου οι διεργασίες βρίσκονται στους κόμβους ενός πλέγματος.

- **Διαχείριση Περιβάλλοντος**: επιτρέπουν την πραγματοποίηση των διεργασιών μέσω συναρτήσεων που σχετίζονται με την εκκίνηση, τον τερματισμό και την διαχείριση σφαλμάτων των διεργασιών.
- **Διαδικασίες καταγραφής και αξιολόγησης**: παρέχει συναρτήσεις δημιουργίας και χρήσης των αρχείων καταγραφής με σκοπό την μέτρηση και την αξιολόγηση της απόδοσης της παράλληλης εφαρμογής.
- **Fortran και C bindings**: σύνολα κανόνων που καθορίζουν την ανάπτυξη εφαρμογών C και Fortran.

1.4 Σημαντικά χαρακτηριστικά του προτύπου MPI 2.0

- **Δυναμική Διαχείριση Διεργασιών**: αυτές οι συναρτήσεις επιτρέπουν την δυναμική δημιουργία και διαχείριση των διεργασιών και χρήση διεργασιών μέσα από μία παράλληλη εφαρμογή.
- **Συναρτήσεις Εισόδου-Εξόδου**: Υποστηρίζει συλλογικές διαδικασίες ανάγνωσης και εγγραφής του αρχείου μέσω διαφορετικών όψεων αρχείων με την βοήθεια των παραγόμενων τύπων δεδομένων.
- **Λειτουργίες απομακρυσμένης προσπέλασης μνήμης**: οι συναρτήσεις αυτές επιτρέπουν την απομακρυσμένη ανάγνωση και εγγραφή των δεδομένων σε περιοχές μνήμης των διεργασιών της εφαρμογής.
- **C++ bindings**: κανόνες που επιτρέπουν την ανάπτυξη των εφαρμογών στην γλώσσα ++.
- **Υποστήριξη threads**: οι συναρτήσεις αυτές επιτρέπουν την ανάπτυξη multi-threaded εφαρμογών.

1.5 Βασικές υλοποιήσεις του MPI

- **MPICH**: Αποτελεί από τις πιο γνωστές υλοποιήσεις του MPI. Αναπτύχθηκε από το Argonne National Laboratory και το Mississippi State University. Η αρχιτεκτονική στηρίζεται σε μια πολυεπίπεδη δομή ώστε η πολυπλοκότητα της αλληλεπίδρασης της εφαρμογής με το υλικό να μην είναι ορατή από τον χρήστη. Αυτό επιτυγχάνεται με την βοήθεια του επιπέδου ADI.

- **LAM:** Αναπτύχθηκε στο Ohaio SuperComputer Center. Η βασική του λειτουργία είναι η μετατροπή ενός δικτύου σταθμού εργασίας σε έναν εικονικό παράλληλο υπολογιστή. Αυτή η διαδικασία στηρίζεται στην εκτέλεση σε κάθε έναν από τους δικτυακούς κόμβους ενός daemon (LAM daemon) που επιτρέπει την πραγματοποίηση διαδικασιών (π.χ η δημιουργία διεργασιών και η διαχείριση υποσυστήματος εισόδου-εξόδου). Η υλοποίηση LAM διατίθεται ελεύθερα στο παγκόσμιο διαδίκτυο.
- **IBM MPI**: η υλοποίηση από την IBM έγινε για να καλύψει τις ανάγκες λειτουργίας των συστημάτων SP, για να αντικαταστήσει την βιβλιοθήκη MPI, που χρησιμοποιούνταν για την υλοποίηση της μεταβίβασης μηνυμάτων στα συστήματα SP. Στην υλοποίηση αυτή τα υπολογιστικά συστήματα μπορούν να διαιρεθούν σε 2 κατηγορίες. Στην πρώτη κατηγορία είναι ότι η εφαρμογή προσπελάυνει άμεσα το στοιχείο μεταγωγής που σημαίνει ότι συμβάλλει σε υψηλή τιμή απόδοσης. Στην δεύτερη κατηγορία οι διεργασίες της παράλληλης εφαρμογής επικοινωνούν μέσω από το πρωτόκολλο IP.
- **Sun MPI**: η υλοποίηση της Sun βασίστηκε στο MPICH και έχει ενσωματωθεί στο περιβάλλον Sun HPC. Η χρήση αυτού του περιβάλλοντος επιτρέπει στους χρήστες να υλοποιήσουν διάφορες διαδικασίες όπως εκκίνηση/τερματισμός διεργασιών και είσοδο/έξοδο του προτύπου MPI-2.

1.6 Δομικές μονάδες και επικοινωνία μεταξύ τους

Στοιχειώδης μονάδα είναι η **Διεργασία (Process)** η οποία δημιουργείται και εκτελείται **ανεξάρτητα** από τις υπόλοιπες και επίσης έχει τους δικούς της πόρους. Ταυτοποιείται από τον **κωδικό διεργασίας (PID)** που ονομάζεται **τάξη (rank)**, ενώ η χρήση κάθε ομάδας διεργασιών **(group)** γίνεται με το δικό της ID.

Η ανταλλαγή μηνυμάτων μεταξύ των διεργασιών γίνεται με το μέσο επικοινωνίας **(communicator)** αλλά για να επικοινωνήσουν δύο διεργασίες πρέπει να ανήκουν στον ίδιο communicator. Μια διεργασία, μπορεί να στείλει μήνυμα στον εαυτό της, αυτό γίνεται αποθηκεύοντας το μήνυμα σε ένα ενδιάμεσο (αν και δεν συνιστάται). Όταν 2 διεργασίες θέλουν να ανταλλάξουν κάποιο μήνυμα θα πρέπει να ανήκουν στον ίδιο communicator. Αυτός έχει πάντα κάποιον κωδικό ο οποίος μπορεί να ανακτηθεί με την χρήση συναρτήσεων του MPI. Έτσι διασφαλίζεται ο σωστός τρόπος διακίνησης της πληροφορίας και αποφεύγεται η εμφάνιση των προβλημάτων.

Η επικοινωνία στο πρότυπο MPI γίνεται με 2 τρόπους:

- **Επικοινωνία από σημείο σε σημείο:** επικοινωνία μεταξύ 2 διεργασιών όπου η μία διεργασία στέλνει ένα μήνυμα ενώ μία άλλη το παραλαμβάνει. Το κάθε ένα ταυτοποιείται από μία ετικέτα (tag) και έχει μία συγκεκριμένη τυποποίηση που είναι ιδιαίτερα σημαντική ειδικά σε ετερογενή συστήματα. Επίσης, το μήνυμα παρέχει και διάφορες πληροφορίες που είναι απαραίτητες για την επιτυχή μετάδοση του μηνύματος: το πλήθος και τον τύπο δεδομένων, τις τάξεις των διεργασιών αποστολής και λήψης, την ετικέτα του μηνύματος και το όνομα του communicator. Όλες αυτές οι πληροφορίες δίνονται στις συναρτήσεις αποστολής και παραλαβής του μηνύματος. Οι μορφές των συναρτήσεων είναι οι ακόλουθες:
  - *Παρεμποδιστική:* λέγεται η συνάρτηση όταν η κλήση της μέσα από κάποια διεργασία αναστέλλει την λειτουργία αυτής της διεργασίας μέχρι την ολοκλήρωση της λειτουργίας της συνάρτησης. Ένα παράδειγμα είναι η συνάρτηση αποστολής send(), για όσο αποστέλλονται τα δεδομένα η διεργασία που κάλεσε την send() περιμένει. Μόλις σταλούν τα δεδομένα τερματίζει η συνάρτηση send() και συνεχίζει η διεργασία αποστολέα. Ένα άλλο παράδειγμα είναι η συνάρτηση που διαβάζει το μήνυμα (receive()), όσο η receive() διαβάζει τα δεδομένα η διεργασία παραλήπτη περιμένει. Όταν παραληφθούν όλα τα δεδομένα η receive() τερματίζει και ο έλεγχος επιστρέφει στην διεργασία παραλήπτη.
  - *Μη παρεμποδιστική:* όταν η κλήση της συνάρτησης δεν προκαλεί την αναστολή της λειτουργίας της διεργασίας. Η κλήση της send() δεν αναστέλλει την διεργασία αποστολέα. Η κλήση της receive() δεν αναστέλλει την διεργασία παραλήπτη. Ενδέχεται να ανακύψουν προβλήματα εάν έχουμε διαδοχικές κλήσεις της send() σε μικρό χρονικό διάστημα.
- **Συλλογικές Επικοινωνίες:** σε αυτό το είδος έχουμε περισσότερες από 2 διαδικασίες. Υπάρχει πάντα μία κεντρική διεργασία που ονομάζεται διεργασία ρίζα (root process) η οποία είτε αποστέλλει τα δεδομένα της εφαρμογής είτε συλλέγει μηνύματα από τις υπόλοιπες διεργασίες. Οι πιο σημαντικές είναι οι ακόλουθες:
    - *Εκμπομπή (Broadcasting):* ένα μήνυμα αποστέλλεται σε όλες τις διεργασίες.

- *Διασπορά (Scattering):* το μήνυμα σπάει σε μικρότερα κομμάτια και το καθένα στέλνεται σε διαφορετική διεργασία.
    - *Συλλογή (gather):* τα μηνύματα όλων των μελών μια ομάδας διεργασιών παραλαμβάνονται από μία διεργασία.
    - *Υποβιβασμός (reduce):* μία διεργασία συλλέγει δεδομένα από τις άλλες και τα χρησιμοποιεί για κάποιον υπολογισμό.

1.7 Οι τύποι δεδομένων του MPI

Οι συναρτήσεις του MPI καλούνται από ένα πρόγραμμα που είναι γραμμένο στην γλώσσα προγραμματισμού C (υποστηρίζονται επίσης και οι γλώσσες προγραμματισμού C++ και Fortran). Οι τύποι δεδομένων των ορισμάτων των συναρτήσεων δεν είναι ίδιοι με τους τύπους δεδομένων που χρησιμοποιούνται στην γλώσσα C, γιατί το MPI χρησιμοποιεί τους δικούς του τύπους δεδομένων. Παρόλα αυτά κανείς δεν απαγορεύει την χρήση των γνωστών τύπων δεδομένων της γλώσσας C αλλά προτείνεται η χρήση των τύπων δεδομένων του MPI.

Πίνακας 1: τύποι δεδομένων του MPI (Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

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

1.8 Οι βασικές συναρτήσεις του MPI

Για να χρησιμοποιήσουμε αυτές τις συναρτήσεις θα πρέπει να εισάγουμε την βιβλιοθήκη <mpi.h>.

Επομένως οι βασικές συναρτήσεις είναι οι εξής:

1.8.1 Η συνάρτηση MPI_Init ()

Η συνάρτηση αυτή αρχικοποιεί το περιβάλλον MPI και δημιουργεί όλες τις απαραίτητες δομές δεδομένων. Θα πρέπει να κληθεί μία και μόνο φορά πριν από την κλήση οποιαδήποτε άλλης συνάρτησης του MPI. Το πρωτότυπο της συνάρτησης έχει την μορφή:

int MPI Init(int * argc, char **argv[]);

Για παράδειγμα θα μπορούσαμε να καλέσουμε αυτή την συνάρτηση ως εξής:

```
MPI_Init(&argc, &argv);
```

1.8.2 Η συνάρτηση MPI_Finalize()

Ο ρόλος της είναι ο τερματισμός της λειτουργίας του περιβάλλοντος MPI και πρέπει να κληθεί ως τελευταία συνάρτηση. Αποδεσμεύει όλες τις δομές δεδομένων του προτύπου.

Το πρωτότυπο της συνάρτησης είναι:

int MPI_Finalize(void);

1.8.3 Η συνάρτηση MPI_Comm_rank()

Επιστρέφει την τάξη της τρέχουσας διεργασίας. Το πρωτότυπο της συνάρτησης είναι:

int MPI_Comm_rank(MPI_Comm comm, int *rank);

με την μεταβλητή comm να καθορίζει τον communicator που ανήκει η διεργασία και το rank που περιέχει την τάξη της τρέχουσας διεργασίας. Ένα παράδειγμα κλήσης είναι:

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

1.8.4 Η συνάρτηση MPI_Comm_size()

Επιστρέφει το μέγεθος ενός communicator, δηλαδή το πλήθος των διεργασιών που περιλαμβάνονται στην ομάδα διεργασιών που συσχετίζονται με αυτόν. Το πρωτότυπο της συνάρτησης είναι:

int MPI_Comm_size (MPI_Comm comm, int *size);

όπου το όρισμα comm χρησιμοποιείται για τον καθορισμό του communicator για το μέγεθος που θέλουμε να ανακτήσουμε και το όρισμα size περιέχει το πλήθος διεργασιών στον τρέχοντα communicator.

1.8.5 Η συνάρτηση MPI_Send()

Η συνάρτηση αυτή κάνει αποστολή μηνύματος σε κάποια από τις διεργασίες στον τρέχοντα communicator. Το πρωτότυπο της συνάρτησης είναι:

int MPI_Send (void*buf, int count, MPI Datatype datatype, int destRank, int messageTag, MPI_Comm comm);

όπου το όρισμα buf περιέχει την διεύθυνση της περιοχής μνήμης, το όρισμα count περιέχει το πλήθος των στοιχείων προς αποστολή, το όρισμα datatype περιέχει τον τύπο δεδομένων των στοιχείων, το όρισμα destRank είναι η τάξη της διεργασίας που πρόκειται να σταλεί το μήνυμα, το όρισμα messageTag είναι η ετικέτα του μηνύματος που μας επιτρέπει να ξεχωρίσουμε αυτό το μήνυμα από τα υπόλοιπα (MPI_ANY_TAG) και τέλος το όρισμα comm είναι ο communicator όπου ανήκουν οι διεργασίες αποστολέας και παραλήπτης.

Ένα παράδειγμα κλήσης θα μπορούσε να είναι το παρακάτω:

```
MPI_Send(buffer, 100, MPI_CHAR, 1,0, MPI_COMM_WORLD);
```

1.8.6 Η συνάρτηση MPI_Recv()

Η συνάρτηση αυτή κάνει παραλαβή ενός μηνύματος σε κάποια από τις διεργασίες από τον τρέχοντα communicator.

Το πρωτότυπο της συνάρτησης είναι:

int MPI_Recv (void*buf, int count, MPI Datatype datatype, int destRank, int messageTag, MPI_Comm comm, MPI_Status status);

Ένα παράδειγμα κλήσης θα μπορούσε να είναι το παρακάτω:

```
MPI_Recv (buffer, 50, MPI_DOUBLE, 4, MPI_ANY_TAG, MPI_COMM_WORLD, status);
```

1.8.7 Η συνάρτηση MPI_Get_count()

Η συνάρτηση αυτή κάνει ανάκτηση του μεγέθους των μηνυμάτων που έχει παραληφθεί από την MPI_Recv. Το πρωτότυπο της συνάρτησης είναι:

int MPI_Get_count (MPI_Status *status, MPI_Datatype datatype, int * count);

Ένα παράδειγμα που συνδυάζονται η MPI_Recv και MPI_Get_count είναι το παρακάτω:

```
MPI_Recv (&value, 1, MPI_INT, rank-1, 0, MPI_COMM_WORLD, &status);
MPI_Get_count (status, MPI_INT, &items);
```

1.8.8 Η συνάρτηση MPI_Sendrecv()

Η συνάρτηση αυτή λειτουργεί με παρεμποδιστικό τρόπο. Αυτό που κάνει είναι να συνδυάζει την αποστολή του μηνύματος προς μία διεργασία παραλήπτη και την παραλαβή του μηνύματος από μια διαδικασία αποστολέα σε μία και μοναδική κλήση.

Το πρωτότυπο της είναι:

int MPI_Sendrecv (void * sendBuf, int sendCount, MPI_Datatype sendType, int dest, int sendTag, void *recvBuf, int recvCount, MPI_Datatype recvType, int source, int recvTag, MPI_Comm comm, MPI_Status *status);

Το βασικό χαρακτηριστικό της συνάρτησης είναι ότι οι περιοχές αποθήκευσης για την αποστολή και την παραλαβή των δεδομένων θα πρέπει να είναι διαφορετικές.

1.8.9 Η συνάρτηση MPI_Isend()

Η συνάρτηση αυτή χρησιμοποιείται για την αποστολή ενός μηνύματος. Έχει πολύ μεγάλες ομοιότητες σε σχέση με την συνάρτηση MPI_Send().

Το πρωτότυπο της είναι το παρακάτω: **int MPI_Isend (void * buf, int count, MPI_Datatype datatype, int destRank, int messageTag, MPI_Comm comm, MPI_Request request);**

Το όρισμα buf αντιστοιχεί στο όνομα της περιοχής μνήμης, το όρισμα count αντιστοιχεί στο πλήθος των δεδομένων, το όρισμα datatype περιέχει τον τύπο των δεδομένων, το όρισμα destRank είναι η τάξη που παραλαμβάνει το μήνυμα, το όρισμα messageTag χρησιμοποιείται για τον καθορισμό της ετικέτας του μηνύματος προς αποστολή, το όρισμα comm χρησιμοποιείται για την επικοινωνία μεταξύ 2 διεργασιών και το όρισμα

request χρησιμοποιείται για την πραγματοποίηση μεγάλου εύρους διαδικασιών που συσχετίζονται με την μη παρεμποδιστική διαδικασία.

Ένα παράδειγμα κλήσης θα μπορούσε να είναι το παρακάτω:

```
MPI_Isend (buf, 15, MPI_DOUBLE, 5, MPI_ANY_TAG, MPI_COMM_WORLD, request);
```

1.8.10 Η συνάρτηση MPI_Irecv()

Χρησιμοποιείται για την μη παρεμποδιστική διαδικασία παραλαβής μηνύματος.

Το πρωτότυπο της είναι:

int MPI_Irecv (void * buf, int count, MPI_Datatype datatype, int srcRank, int messageTag, MPI_Comm comm, MPI_Request request);

Τα ορίσματα αυτής της συνάρτησης είναι ίδια με την συνάρτηση MPI_Isend().

Στην δήλωση της συνάρτησης δεν περιέχει το όρισμα MPI_Status σε σχέση με την συνάρτηση MPI_Recv().

Ένα παράδειγμα θα μπορούσε να είναι το παρακάτω:

```
MPI_Irecv (buf, 10, MPI_INT, 3, 0, MPI_COMM_WORLD, request);
```

1.8.11 Η συνάρτηση MPI_Bcast()

Η συνάρτηση αυτή αποστέλλει ένα μήνυμα από κάποια διεργασία (διεργασία ρίζα) σε όλες τις υπόλοιπες διεργασίες.

Η δήλωσή της είναι της μορφής:

int MPI_Bcast (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm);

Ένα παράδειγμα της συνάρτησης θα μπορούσε να είναι το παρακάτω:

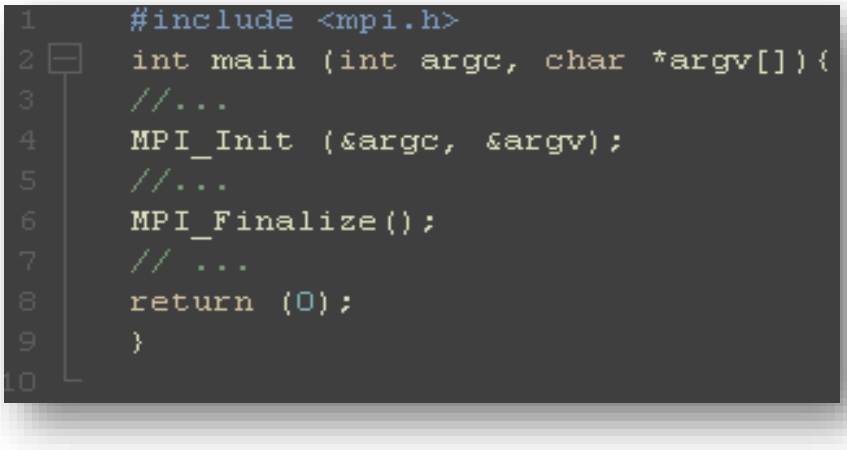
```
MPI_Bcast (buffer, 500, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

1.9 Δομή προγράμματος που χρησιμοποιεί τις βασικές συναρτήσεις του MPI

Το MPI θεωρείται ως ένα πολύπλοκο μοντέλο επικοινωνίας. Το σύνολο των παράλληλων εφαρμογών που στηρίζεται σε αυτό το πρότυπο μπορεί να υλοποιηθεί από ένα μικρό πλήθος συναρτήσεων. Αυτές οι συναρτήσεις βρίσκονται ορισμένες στο αρχείο επικεφαλίδας mpi.h που χρησιμοποιείται σε όλες τις παράλληλες εφαρμογές με αυτό το πρότυπο. Η κλήση αυτού του αρχείου επικεφαλίδας γίνεται με αυτό τον τρόπο: #include <mpi.h>. Επίσης θα πρέπει να λάβει χώρα η αρχικοποίηση του περιβάλλοντος που γίνε-

ται από την συνάρτηση MPI_Init().Αυτό σημαίνει ότι θα πρέπει να καλείται πρώτη πριν από κάθε άλλη συνάρτηση. Επιπλέον θα πρέπει να τερματίζεται και η λειτουργία του περιβάλλοντος. Η συνάρτηση που το κάνει είναι η συνάρτηση MPI_Finalize(). Η συνάρτηση αυτή θα πρέπει να καλείται πάντοτε τελευταία μέσα από τον κώδικα της εφαρμογής.

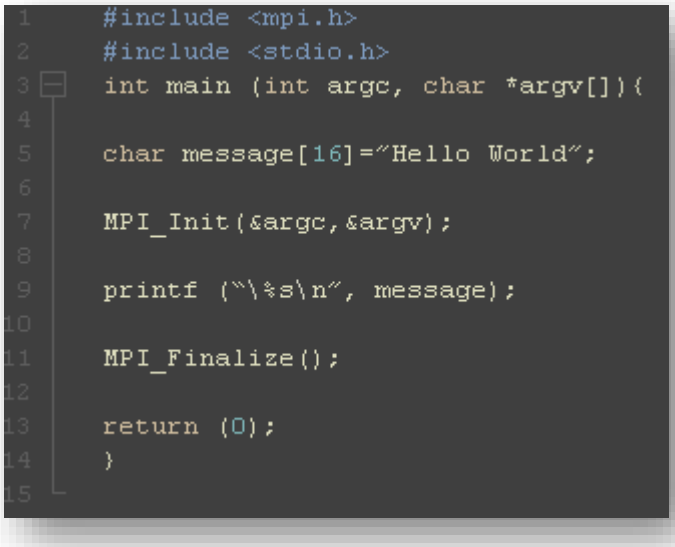
Βασική Δομή



Εικόνα 1: Η βασική δομή ενός προγράμματος MPI (Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

Παράδειγμα Προγράμματος: Hello World

Εάν θέλουμε να γράψουμε ένα πρόγραμμα το οποίο θα αρχικοποιεί το περιβάλλον MPI, θα εκτυπώνει το μήνυμα Hello World και θα τερματίζει τελικά το περιβάλλον, ο κώδικας φαίνεται στην εικόνα 2:



```
#include <mpi.h>
#include <stdio.h>
int main (int argc, char *argv[]){

char message[16]="Hello World";

MPI_Init(&argc,&argv);

printf ("\%s\n", message);

MPI_Finalize();

return (0);
}
```

Εικόνα 2: Παράδειγμα που χρησιμοποιεί τις συναρτήσεις MPI_Init και MPI_Finalize και εκτυπώνει το μήνυμα Hello World (Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

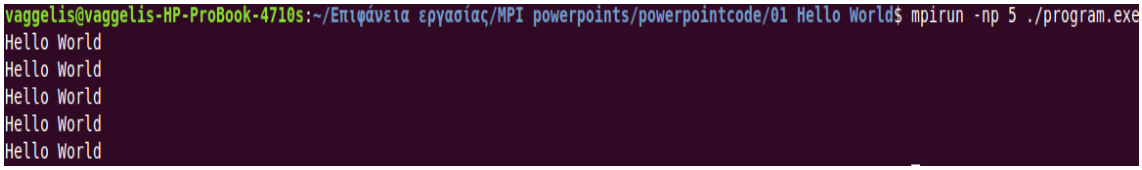
Εκτέλεση του κώδικα

Για να τρέξουμε το παραπάνω πρόγραμμα θα χρειαστούμε 3 βήματα:

- *Πρώτο βήμα*: Θα χρειαστεί να χρησιμοποιήσουμε έναν μεταγλωττιστή που φέρει το όνομα mpicc. Αυτός ο μεταγλωττιστής περιέχει λειτουργίες που είναι εξειδικευμένες και σχετίζονται με το περιβάλλον του MPI. Για να κατασκευάσουμε το αρχείο του αντικειμενικού κώδικα θα πρέπει να χρησιμοποιήσουμε την εντολή *mpicc -c program.c* όπου program.c είναι το όνομα που αποθηκεύσαμε.
- *Δεύτερο βήμα*: Για την κατασκευή του εκτελέσιμου αρχείου η εντολή που πρέπει να δώσουμε την εντολή θα είναι *mpicc -o program.exe program.o* όπου με αυτή την εντολή θα έχουμε το εκτελέσιμο αρχείο program.exe
- *Τρίτο βήμα*: Για να δημιουργήσουμε ένα πλήθος N διεργασιών θα πρέπει να χρησιμοποιήσουμε -np με το πλήθος των διεργασιών που θέλουμε να δημιουργήσουμε. Η εντολή αυτή είναι *mpirun -np 5 program.exe* όπου με αυτή την εντολή μπορούμε να δημιουργήσουμε 5 διεργασίες όπου οι τιμές των τάξεων τους κυμαίνονται από τις τιμές 0 έως 4.

Έξοδος του κώδικα:

Εάν τρέξουμε τον παραπάνω κώδικα στην κονσόλα το αποτέλεσμα φαίνεται στην Εικόνα 3:



Εικόνα 3: Έξοδος του κώδικα για N=5 (Έξοδος από το λειτουργικό σύστημα Ubuntu)

1.10 Οι συναρτήσεις MPI_Test() , MPI_Wait() και MPI_Reduce()

Η συνάρτηση MPI_Wait()

Ελέγχει εάν μια διαδικασία έχει ολοκληρωθεί ή είναι ακόμα σε εξέλιξη. Δέχεται ως όρισμα την μεταβλητή της τιμής request που ανήκει στον ίδιο τύπο δεδομένων MPI_Request

Το πρωτότυπο της είναι:

int MPI_Wait (MPI_Request *request, MPI_Status * status);

όπου το request είναι η τιμή του ορίσματος αυτού που έχει επιστραφεί από την συναρτήσεις MPI_Isend η MPI_Irecv και το όρισμα status περιέχει την ζητούμενη πληροφορία σχετικά με την κατάσταση της διαδικασίας.

Η συνάρτηση MPI_Test()

Εξετάζει εάν η διαδικασία έχει ολοκληρώσει την εκτέλεσή της η όχι. Η δήλωση της είναι το παρακάτω:

int MPI_Test (MPI_Request request, int *flag, MPI_Status * status);

όπου το όρισμα flag περιέχει την τιμή true εάν η διαδικασία μετάδοσης από το όρισμα request έχει ολοκληρωθεί και false εάν δεν έχει ολοκληρωθεί.

Η συνάρτηση MPI_Reduce()

Η συνάρτηση αυτή δέχεται ως είσοδο έναν πίνακα τιμών που έχουν αποσταλλεί από άλλες διεργασίες. Αυτό που κάνει είναι ότι υπολογίζει την τιμή μιας απλής ποσότητας εφαρμόζοντας μια αριθμητική η λογική πράξη (του συστήματος ή του χρήστη).

Η δήλωσή της θα είναι η παρακάτω:

int MPI_Reduce (void * sendBuffer, void * recvBuffer, int count, MPI_Datatype dataType, MPI_Op operation, int root, MPI_Comm comm);

2 Εικονικές Τοπολογίες

Σε αυτό το κεφάλαιο θα αναλύσουμε τις εικονικές τοπολογίες. Το MPI υποστηρίζει 2 είδη τοπολογιών τις καρτεσιανές τοπολογίες και τις τοπολογίες γραφήματος. Στο παρόν κεφάλαιο θα αναλύσουμε κάποια θεωρητικά στοιχεία των τοπολογιών αυτών και θα δούμε τις βασικές συναρτήσεις.

2.1 Εισαγωγή

Σημαντικό χαρακτηριστικό της βιβλιοθήκης του MPI είναι η δυνατότητα οργάνωσης διεργασιών μιας παράλληλης εφαρμογής σε κατάλληλα διαμορφωμένες τοπολογίες που προσφέρουν πολλά πλεονεκτήματα. Η δυνατότητα οργάνωσης των διεργασιών σε κατάλληλα διαμορφωμένες τοπολογίες είναι πολύ σημαντική λόγω ότι οι διεργασίες που ανταλλάσσουν δεδομένα σε σχέση με τις υπόλοιπες διεργασίες θα είναι γειτονικές.

Υπάρχουν 2 είδη τοπολογιών:

- ***Καρτεσιανές Τοπολογίες***: οι διεργασίες τοποθετούνται πάνω στους κόμβους ενός πλέγματος d διαστάσεων.
- ***Τοπολογίες γραφήματος***: Το επικοινωνιακό περιβάλλον συσχετίζεται με ένα μη προσανατολισμένο γράφημα.

2.2 Βασικά χαρακτηριστικά των εικονικών και των καρτεσιανών τοπολογιών

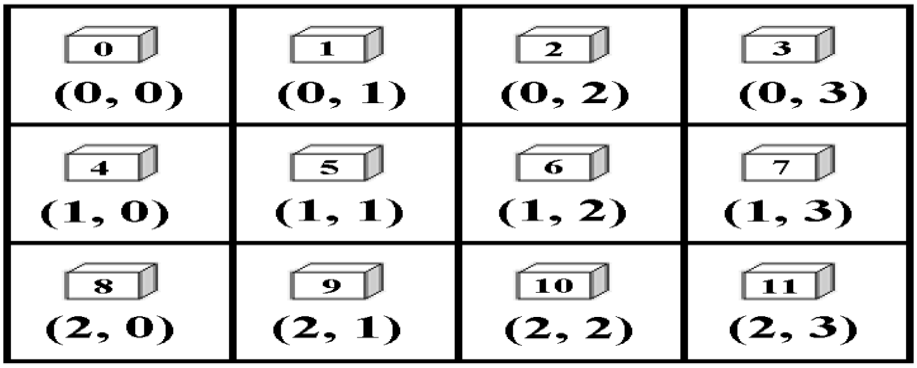
Εικονικές Τοπολογίες

Μία ιδιότητα που συσχετίζεται με τις εικονικές τοπολογίες είναι η αναδιάταξη των διεργασιών για τις τιμές της τάξης τους. Η αναδιάταξη αυτή επιτρέπει την χρήση διαφορετικής τιμής τάξης για την διεργασία τόσο στον παλιό όσο και στον νέο τρόπο οργάνωσης. Για παράδειγμα εάν k η αρχική τάξη της διεργασίας και k΄ η τιμή τάξης του communicator που συσχετίζεται με την νέα τοπολογία τότε οι δύο τιμές θα είναι διαφο-

ρετικές μεταξύ τους. Και τέλος τα μειονεκτήματα των εικονικών τοπολογιών είναι πως μία ανακατανομή δεδομένων στις διεργασίες της εφαρμογής δεν θα μπορέσει να πραγματοποιηθεί με συλλογικό τρόπο και επίσης οι εικονικές τοπολογίες παρέχουν μια στατική περιγραφή του περιβάλλοντος επικοινωνίας.

Καρτεσιανές Τοπολογίες:

Οι καρτεσιανές τοπολογίες διεργασιών είναι μέρος του συνόλου μιας μερικής περίπτωσης των τοπολογιών γραφήματος. Η δημιουργία και η διαχείριση τους λαμβάνει χώρα μέσω ενός συνόλου εντολών του MPI. Στις καρτεσιανές τοπολογίες η γραμμική οργάνωση των διεργασιών και η απόδοση των τιμών στο διάστημα [0, N-1] δεν υφίσταται. Οι διεργασίες τοποθετούνται πάνω στους κόμβους ενός πλέγματος. Δεν επαρκεί η γνώση της τάξης μίας διεργασίας για την ταυτοποίησή της: η κάθε διεργασία περιγράφεται από ένα σύνολο συντεταγμένων ίσο με τον αριθμό d (διάσταση πλέγματος). Η οργάνωση των διεργασιών στους κόμβους ενός δισδιάστατου πλέγματος στην καρτεσιανή τοπολογία μπορεί να φανεί στην παρακάτω εικόνα:



Εικόνα 4: Τυπική οργάνωση των διεργασιών στους κόμβους ενός δισδιάστατου πλέγματος

(Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

2.3 Βασικές συναρτήσεις καρτεσιανής τοπολογίας διεργασιών

Η συνάρτηση MPI_Cart_create()

Η συνάρτηση αυτή δημιουργεί μια καρτεσιανή τοπολογία.

Η δήλωσή της είναι:

int MPI_Cart_create (MPI_Comm comm, int nDims, int *dims, int *periods, int reorder, MPI_Comm *cartComm);

όπου το όρισμα comm είναι ο communicator οι διεργασίες του οποίου θα οργανωθούν σε μία καρτεσιανή τοπολογία και θα συσχετιστεί με τον νέο communicator cartComm. (Ο communicator θα είναι ο MPI_COMM_WORLD), το όρισμα nDims είναι το πλήθος των διεργασιών της καρτεσιανής τοπολογίας, το όρισμα dims είναι ένας πίνακας που περιλαμβάνει το πλήθος των διεργασιών της κάθε διάστασης και περιλαμβάνει nDims στοιχεία αυτής της μορφής, το όρισμα periods είναι πίνακας nDims λογικών μεταβλητών οι οποίες παίρνουν την τιμή true η false, και το όρισμα reorder παίρνει την τιμή true η false και καθορίζει εάν θα λάβει χώρα η επαναδιάταξη των διεργασιών, στον νέο communicator που συσχετίζεται με την καρτεσιανή τοπολογία.

Η συνάρτηση MPI_Dims_create()

Η συνάρτηση αυτή αποφασίζει το πλήθος των διεργασιών που θα τοποθετηθούν σε κάθε μία από τις διευθύνσεις του καρτεσιανού πλέγματος.

Η δήλωσή της είναι:

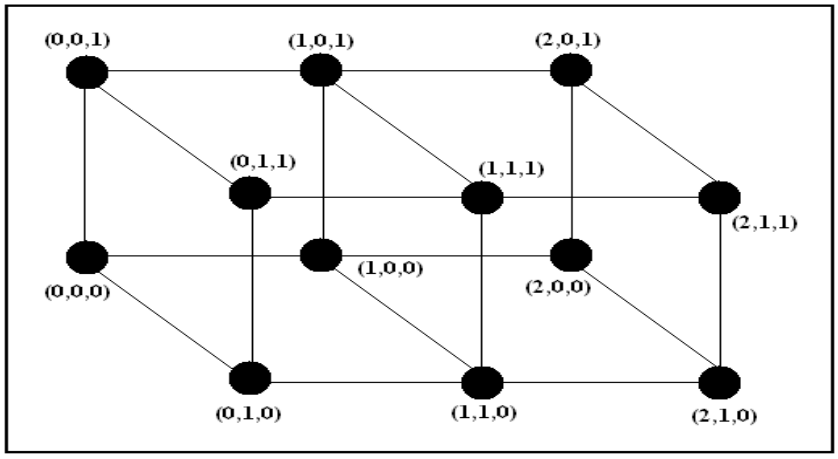
int MPI_Dims_create (int nNodes, int nDims, int *dims);

όπου το όρισμα nNodes είναι το πλήθος των κόμβων του πλέγματος, το όρισμα nDims είναι το πλήθος των διαστάσεων του πλέγματος, και το όρισμα dims είναι ένας πίνακας ακεραίων που περιέχει μετά την κλήση της συνάρτησης το πλήθος των διεργασιών που θα τοποθετηθούν σε κάθε μία από τις διευθύνσεις της καρτεσιανής τοπολογίας. Ένα παράδειγμα κώδικα της λειτουργίας των συναρτήσεων MPI_Cart_create() και MPI_Dims_create() είναι το παρακάτω. Το παράδειγμα αυτού του κώδικα είναι από την βιβλιογραφία *(Αθανάσιος Ι. Μάργαρης MPI Θεωρία & Εφαρμογές Εκδόσεις Τζιόλα 2014):*

```
int main (int argc, char ** argv) {
MPI_Comm cartComm;
int counter, rank, size;
int nDims = 3, dims [3] = {0,3,0};
int periodic [3] = {true, false, true};
MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);
MPI_Dims_create (size, nDims, dims);
if (rank==0) {
for (counter=0;counter<3;counter++)
printf ("Dims[%d]=%d ", counter, dims[counter]);
}
MPI_Cart_create (MPI_COMM_WORLD, 3, dims, periodic, true, &cartComm);
```

```
MPI_Finalize();
return (0); }
```

Από τον παραπάνω κώδικα γίνεται ανάκτηση του πλήθους των διεργασιών και αυτόματος υπολογισμός όπου η κάθε διεργασία μπορεί να τοποθετηθεί σε κάθε κατεύθυνση ενός καρτεσιανού πλέγματος τριών κατευθύνσεων. Η εκτύπωση της πληροφορίας γίνεται από την διεργασία R=0 και επίσης δημιουργείται μια καρτεσιανή τοπολογία που δημιουργείται από την συνάρτηση MPI_Cart_create(). Εάν έχουμε ένα σύνολο 12 διεργασιών με τιμές του πίνακα dims={3,2,2} μπορεί να αναπαρασταθεί σχηματικά στην παρακάτω εικόνα. Στην παρακάτω εικόνα επίσης εμφανίζονται και οι συντεταγμένες που ταυτοποιούν αυτή την νέα τοπολογία.



Εικόνα 5: Παράδειγμα καρτεσιανής τοπολογίας σε πλέγμα τριών διαστάσεων {dims=(3,2,2)}(Αθανάσιος Ι. Μάργαρης MPI Θεωρία & Εφαρμογές Εκδόσεις Τζιόλα 2014)

Η συνάρτηση MPI_Cartdim_get()

Η συνάρτηση αυτή επιστρέφει το πλήθος των διαστάσεων που έχει δημιουργηθεί από την καρτεσιανή τοπολογία.

Η δήλωσή της είναι:

int MPI_Cartdim_get (MPI_Comm comm, int *nDims);

όπου το όρισμα comm είναι ο communicator που συσχετίζεται με την καρτεσιανή τοπολογία το πλήθος του οποίου επιθυμούμε να ανακτήσουμε, και το όρισμα nDims είναι η τιμή του μεγέθους που βρίσκεται καταχωρημένη μετά την επιστροφή της συνάρτησης.

Ένα παράδειγμα κλήσης θα μπορούσε να είναι:

```
MPI_Cart_create (MPI_COMM_WORLD, nDims, dims, periodic, reorder, &cartComm) ;
MPI_Cartdim_get (cartComm, &nDims);
```

Η συνάρτηση MPI_Cart_get()

Η συνάρτηση αυτή κάνει ανάκτηση των υπόλοιπων παραμέτρων της καρτεσιανής τοπολογίας.

Η δήλωσή της είναι:

int MPI_Cart_get (MPI_Comm comm, int maxDims, int*dims, int *periods, int*coords);

όπου το όρισμα comm είναι ο communicator που συσχετίζεται με την τρέχουσα καρτεσιανή τοπολογία, το όρισμα maxDims περιέχει το πλήθος των διαστάσεων αυτής της τοπολογίας ενώ τα ορίσματα dims, periods: περιέχουν για κάθε διάσταση το πλήθος των διεργασιών και την τιμή της λογικής μεταβλητής που καθορίζει την περιοδικότητα της και το όρισμα coords είναι οι συντεταγμένες της τρέχουσας διεργασίας.

Παράδειγμα λειτουργίας των συναρτήσεων των MPI_Cartdim_get() και MPI_Cart_get()

Σε αυτό το παράδειγμα έχουμε δημιουργία μιας καρτεσιανής τοπολογίας όπου τα χαρακτηριστικά της εν συνεχεία ανακτώνται και εκτυπώνονται στην οθόνη. Το πλήθος των διεργασιών για κάθε διάσταση είναι 2,3,2 ενώ το πλήθος των διεργασιών που θα ανακτηθούν θα είναι ίσο με 12. Η βιβλιογραφία αυτού του κώδικα είναι από *(Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014).*Ο κώδικας παρουσιάζεται στην συνέχεια:

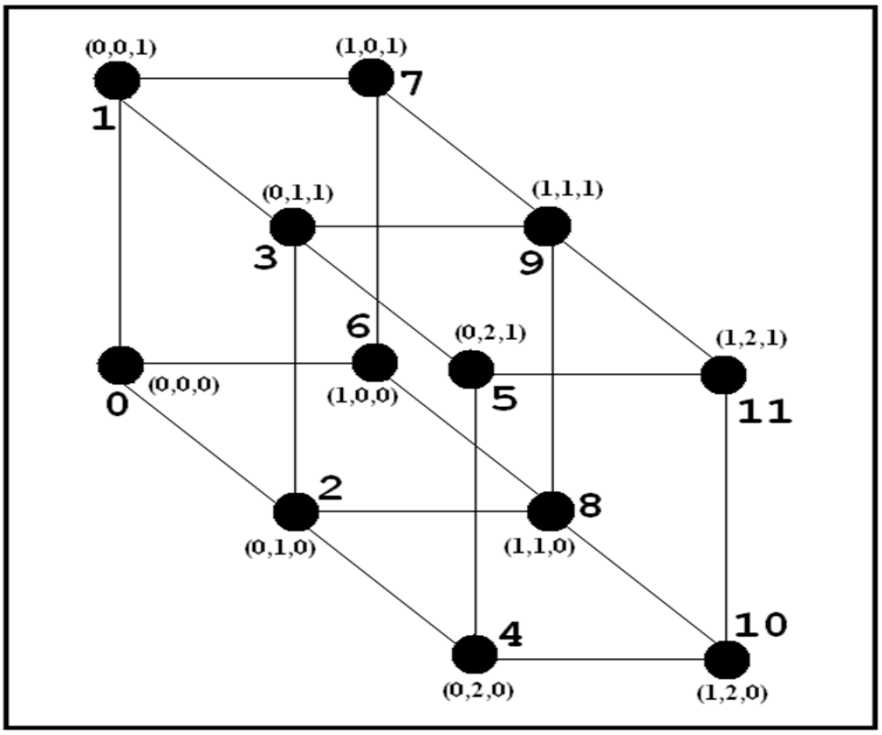
```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define DIM 3
int main (int argc, char ** argv) {
MPI_Comm cartComm;
int counter, rank, size;
int dims [DIM] = {2,3,2};
int periodic[DIM]={1,0,1};
int cartDim, *dim, *periods, *coords;
MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);
MPI_Cart_create (MPI_COMM_WORLD, DIM, dims, periodic, 1, &cartComm);
MPI_Cartdim_get (cartComm, &cartDim);

dim = (int *)malloc(cartDim*sizeof(int));
periods = (int *)malloc(cartDim*sizeof(int));
coords = (int *)malloc(cartDim*sizeof(int));

MPI_Cart_get (cartComm, cartDim, dim, periods, coords);

if (rank==0) {
printf ("Cartesian Topology Dimension is %d\n", cartDim);
for (counter=0;counter<cartDim;counter++)
printf ("Dim[%d]=%d ", dim[counter]);
printf ("\n");
for (counter=0;counter<cartDim;counter++)
printf ("Periods[%d]=%d ", periods[counter]);
printf ("\n");
}

printf ("Process %d Cartesian Coordinates\n", rank);
for (counter=0;counter<cartDim;counter++)
printf ("Coords[%d]=%d\n", counter, coords[counter]);
printf ("\n");
MPI_Finalize();
return (0);
}
```

Από τον παραπάνω κώδικα παρατηρούμε πως η διεργασία R=0 εκτυπώνει τα περιεχόμενα των πινάκων dims και periods ενώ η κάθε μία από τις υπόλοιπες διεργασίες εκτυπώνει τις συντεταγμένες της στην καρτεσιανή τοπολογία. Η έξοδος του προγράμματος αυτού θα είναι το παρακάτω:

```
Cartesian Topology Dimension is 3

Dim[2]=2 Dim[3]=3 Dim[2]=2
Periods[1]=1 Periods[0]=0 Periods[1]=1

Process 0 Cartesian Coordinates
Coords[0]=0
Coords[1]=0
Coords[2]=0

Process 1 Cartesian Coordinates
Coords[0]=0
Coords[1]=0
Coords[2]=1
```

```
Process 2 Cartesian Coordinates
Coords[0]=0
Coords[1]=1
Coords[2]=0

Process 3 Cartesian Coordinates
Coords[0]=0
Coords[1]=1
Coords[2]=1

Process 4 Cartesian Coordinates
Coords[0]=0
Coords[1]=2
Coords[2]=0
Process 5 Cartesian Coordinates
Coords[0]=0
Coords[1]=2
Coords[2]=1

Process 6 Cartesian Coordinates
Coords[0]=1
Coords[1]=0
Coords[2]=0

Process 7 Cartesian Coordinates
Coords[0]=1
Coords[1]=0
Coords[2]=1

Process 8 Cartesian Coordinates
Coords[0]=1
Coords[1]=1
Coords[2]=0

Process 9 Cartesian Coordinates
Coords[0]=1
Coords[1]=1
Coords[2]=1

Process 10 Cartesian Coordinates
Coords[0]=1
Coords[1]=2
Coords[2]=0

Process 11 Cartesian Coordinates
Coords[0]=1
Coords[1]=2
Coords[2]=1
```

Επίσης η καρτεσιανή τοπολογία του παραπάνω παραδείγματος μπορεί να φανεί σχηματικά και παρουσιάζεται στην παρακάτω εικόνα:



Εικόνα 5: Παράδειγμα καρτεσιανής τοπολογίας σε πλέγμα τριών διαστάσεων {dims=(2,3,2)}(Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

Η συνάρτηση MPI_Cart_rank()

Η συνάρτηση αυτή κάνει ανάκτηση της τιμής μιας διεργασίας όταν είναι γνωστές οι λογικές συντεταγμένες στην καρτεσιανή τοπολογία που ανήκει.

Η δήλωσή της είναι η παρακάτω:

int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank);

όπου το όρισμα rank είναι η τάξη της διεργασίας μέσα στην αρχική ομάδα διεργασιών που χρησιμοποιήθηκε για την κατασκευή της καρτεσιανής τοπολογίας.

Η συνάρτηση MPI_Cart_coords()

Η συνάρτηση αυτή επιστρέφει τις καρτεσιανές τοπολογίες μιας διεργασίας όταν είναι γνωστή η τιμή της τάξης της.

Η δήλωσή της είναι:

int MPI_Cart_coords (MPI_Comm comm, int rank, int maxDim, int *coords);

όπου το όρισμα coords είναι ένας πίνακας που αποτελείται maxDim στοιχεία και περιέχει τις λογικές συντεταγμένες της συνάρτησης της καρτεσιανής τοπολογίας.

Παράδειγμα της συνάρτησης MPI_Cart_coords()

Στο παράδειγμα αυτό γίνεται οργάνωση των 12 διεργασιών του αρχικού communicator MPI_COMM_WORLD σε μία τρισδιάστατη καρτεσιανή τοπολογία με πλήθος διεργασιών ανά διεύθυνση 2,3,2. Έπειτα η διεργασία με τάξη R=0 υπολογίζει τις καρτεσιανές τοπολογίες για κάθε μία από τις διεργασίες του συστήματος. (Η βιβλιογραφία αυτού του κώδικα είναι από : *(Αθανάσιος Ι. Μάργαρης MPI Θεωρία & Εφαρμογές Εκδόσεις Τζιόλα 2014)*.Ο πηγαίος κώδικας θα είναι ο παρακάτω:

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char ** argv) {
MPI_Comm cartComm;
int counter, rank, size;
int dims [3] = {2,3,2}, coords [3];
int periodic [3] = {1, 0, 1};

MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);

MPI_Cart_create (MPI_COMM_WORLD, 3, dims, periodic, 1, &cartComm);

if (rank==0) {
    for (counter=0;counter<size;counter++) {
            MPI_Cart_coords (cartComm, counter, 3, coords);
            printf ("Process %02d has coordinates (%d,%d,%d)\n", counter, coords[0], coords[1],
                    coords[2]);
            }
            }
MPI_Finalize();
return (0);
}
```

και η έξοδος αυτού του κώδικα θα είναι η παρακάτω:

```
Process 00 has coordinates (0,0,0)
Process 01 has coordinates (0,0,1)
Process 02 has coordinates (0,1,0)
Process 03 has coordinates (0,1,1)
Process 04 has coordinates (0,2,0)
Process 05 has coordinates (0,2,1)
Process 06 has coordinates (1,0,0)
Process 07 has coordinates (1,0,1)
Process 08 has coordinates (1,1,0)
Process 09 has coordinates (1,1,1)
Process 10 has coordinates (1,2,0)
Process 11 has coordinates (1,2,1)
```

Η συνάρτηση MPI_Cart_shift()

Η συνάρτηση αυτή επιστρέφει την τάξη των διεργασιών που θεωρούνται γειτονικές ως προς την συγκεκριμένη διεργασία.

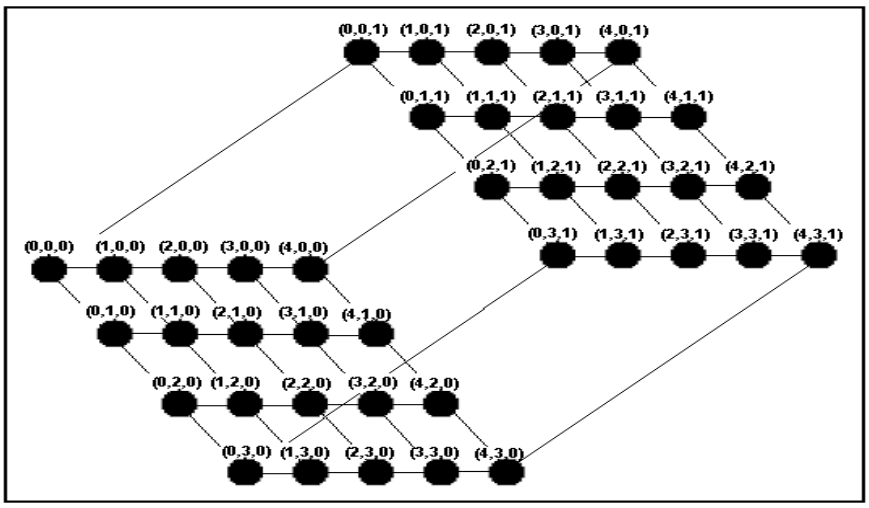
Η δήλωσή της είναι:

int MPI_Cart_shift (MPI_Comm comm, int direction, int displacement, int *sourceRank, int * destRank);

όπου το όρισμα comm είναι ο communicator που σχετίζεται με την τρέχουσα καρτεσιανή τοπολογία, το όρισμα direction είναι η διεύθυνση όπου αναζητούμε τις γειτονικές διεργασίες, το όρισμα displacement είναι η απόσταση των διεργασιών από την τρέχουσα διεργασία και τα ορίσματα sourceRank, destRank είναι οι τάξεις των διεργασιών που βρίσκονται καταχωρημένα στα ορίσματα αυτά μετά την επιστροφή της συνάρτησης.

Παράδειγμα της συνάρτησης MPI_Card_shift()

Έστω ότι έχουμε 40 διεργασίες που κατανέμονται στους κόμβους ενός τρισδιάστατου πλέγματος έτσι ώστε στην πρώτη διάσταση να τοποθετούνται 5 διεργασίες, στην δεύτερη διάσταση να τοποθετούνται 4 διεργασίες και στην τρίτη διάσταση να τοποθετούνται 2 διεργασίες. Αυτή η κατάσταση μπορεί να φανεί στην παρακάτω εικόνα:



Εικόνα 6: Τρισδιάστατη τοπολογία διεργασιών {dims=(5,4,2)}(Αθανάσιος Ι. Μάργαρης MPI Θεωρία & Εφαρμογές Εκδόσεις Τζιόλα 2014)

Από το παραπάνω σχήμα σε κάθε μία από τις διεργασίες της καρτεσιανής τοπολογίας, όπου σε κάθε μία από τις διεργασίες της καρτεσιανής τοπολογίας αναγράφονται οι τιμές των συντεταγμένων που ορίζουν μια διατεταγμένη τριάδα της μορφής (x,y,z). Εάν υποθέσουμε ότι για κάθε μία από τις παραπάνω διεργασίες θέλουμε να ταυτοποιήσουμε τους πιο κοντινούς γείτονες τότε θα πρέπει να καλέσουμε την συνάρτηση MPI_Card_shift τρείς φορές με την τιμή displacement=1. Στον κώδικα μας η κάθε διεργασία αναλαμβάνει να εντοπίσει τους πλησιέστερους γείτονές της και να εκτυπώσει τα

αποτελέσματα αναζήτησης στην οθόνη. (Η βιβλιογραφία αυτού του κώδικα είναι από : *Αθανάσιος Ι. Μάργαρης MPI Θεωρία & Εφαρμογές Εκδόσεις Τζιόλα 2014*)Ο κώδικας μας θα είναι ο παρακάτω:

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char ** argv) {
MPI_Comm cartComm;
int counter, rank, size;
int dims [3] = {5,4,2};
int periodic [3] = {1, 1, 1};
int sourceRank, destRank;

MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);

MPI_Cart_create (MPI_COMM_WORLD, 3, dims, periodic, 1, &cartComm);

for (counter=0;counter<3;counter++) {
MPI_Cart_shift (cartComm, counter, 1, &sourceRank, &destRank);
printf ("Process %02d: Source is %02d Destination is %02d for dimension %d\n", rank, sourceRank,
        destRank, counter);
}

MPI_Finalize();
return (0); }
```

Και η έξοδος του κώδικα φαίνεται στο Παράρτημα Β στην σελίδα 61.

Εάν θέλουμε να αλλάξουμε τον πίνακα περιοδικότητας με τις τιμές false, true, false θα πρέπει να αλλάξουμε μόνο τον πίνακα periodic. Ο υπόλοιπος κώδικας θα παραμείνει ο ίδιος:

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char ** argv) {
MPI_Comm cartComm;
int counter, rank, size;
int dims [3] = {5,4,2};
int periodic [3] = {0, 1, 0};
int sourceRank, destRank;

MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);

MPI_Cart_create (MPI_COMM_WORLD, 3, dims, periodic, 1, &cartComm);

for (counter=0;counter<3;counter++) {
MPI_Cart_shift (cartComm, counter, 1, &sourceRank, &destRank);
printf ("Process %02d: Source is %02d Destination is %02d for dimension %d\n", rank, sourceRank,
        destRank, counter);
}

MPI_Finalize();
return (0); }
```

Η έξοδος του κώδικα φαίνεται στην σελίδα 63.

2.4 Βασικές συναρτήσεις τοπολογίας γραφήματος

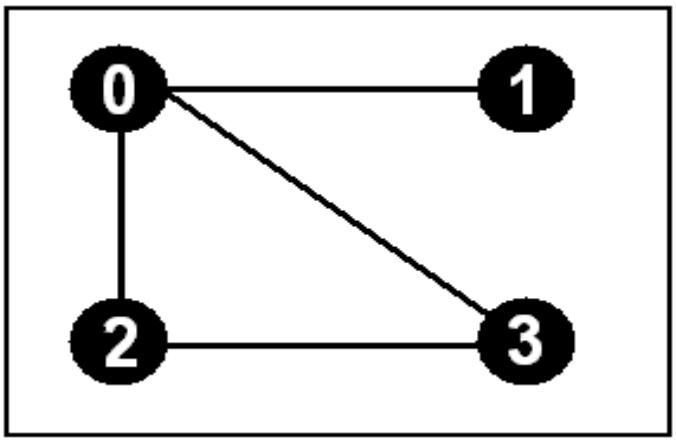
Η συνάρτηση MPI_Graph_create()

Η συνάρτηση αυτή επιτρέπει την δημιουργία και την διαχείριση τοπολογιών γραφήματος.

Η δήλωσή της είναι:

int MPI_Graph_create(MPI_Comm oldcomm, int nodeNumber, int *index, int *edges, int reorder, MPI_Comm *graphComm);

όπου το όρισμα oldcomm είναι ο αρχικός communicator όπου προκύπτει η τοπολογία γραφήματος και συσχετίζεται με τον νέο communicator graphComm, το όρισμα nodeNumber είναι το πλήθος των κόμβων του γραφήματος με τις τιμές 0,1,2,…, nodeNumber-1, το όρισμα index είναι το συνολικό πλήθος των γειτόνων των πρώτων k κόμβων του γραφήματος, το όρισμα edges είναι οι γείτονες του κάθε κόμβου που αποθηκεύονται στις συνεχόμενες θέσεις του πίνακα και το όρισμα reorder είναι η αναδιάταξη των διεργασιών στην νέα ομάδα που αφορά τις τιμές των τάξεων τους. Μία δημιουργία τοπολογίας γραφήματος φαίνεται στην παρακάτω εικόνα:



Εικόνα 7: Τοπολογία γραφήματος που έχει δημιουργηθεί μέσω της συνάρτησης MPI_Graph_create() (Αθανάσιος Ι. Μάργαρης MPI Θεωρία & Εφαρμογές Εκδόσεις Τζιόλα 2014)

Η συνάρτηση MPI_Graphdim_get()

Η συνάρτηση αυτή επιστρέφει μετά την κλήση της το πλήθος των κόμβων και των ακμών μιας τοπολογίας γραφήματος.

Η δήλωσή της είναι:

int MPI_Graphdim_get (MPI_Comm graphComm, int *nodeNumber, int *edgeNumber);

όπου το όρισμα graphComm είναι ο communicator της καρτεσιανής τοπολογίας, το όρισμα nodeNumber είναι το πλήθος των κόμβων της τρέχουσας τοπολογίας, και το όρισμα edgeNumber είναι το πλήθος των ακμών της τρέχουσας τοπολογίας.

Η συνάρτηση MPI_Graph_neighbors_count()

Η συνάρτηση αυτή ορίζει το πλήθος των γειτόνων της συγκεκριμένης διεργασίας.

Η δήλωσή της είναι:

int MPI_Graph_neighbors_count (MPI_Comm graphComm, int rank, int *neighborNumber);

όπου το όρισμα rank είναι η τάξη της διεργασίας που αναζητούμε που αναζητούμε το πλήθος των γειτόνων της και το όρισμα neighborNumber είναι η πληροφορία για την τάξη της διεργασίας που θα βρίσκεται καταχωρημένη σε αυτό το όρισμα μετά την επιστροφή της συνάρτησης.

Η συνάρτηση MPI_Graph_neighbors()

Με αυτή την συνάρτηση μπορούμε να ανακτήσουμε τις τιμές των τάξεων αυτών των γειτόνων.

Η δήλωσή της είναι:

int MPI_Graph_neighbors (MPI_Comm graphComm, int rank, int maxNeighbors, int * neighbors);

όπου το όρισμα rank καθορίζει την τάξη της τρέχουσας διεργασίας μέσα στο γράφημα, το όρισμα maxNeighbors καθορίζει την μέγιστη διάσταση του πίνακα neighbors μετά την επιστροφή της συνάρτησης και το όρισμα neighbors περιέχει τις τάξεις των διεργασιών, που είναι γειτονικές στην διεργασία με τάξη R μετά την επιστροφή της συνάρτησης.

Η συνάρτηση MPI_Graph_map()

Η συνάρτηση αυτή επιστρέφει την βέλτιστη τοπολογία γραφήματος για την δομή του παράλληλου φυσικού συστήματος.

Η δήλωσή της είναι:

int MPI_Graph_map (MPI_Comm comm, int nodeNumber, int * index, int * edges, int *newRank);

όπου το όρισμα newRank περιέχει την τάξη της τρέχουσας διεργασίας, στην νέα τοπολογία που προτείνεται από το MPI μετά την επιστροφή της συνάρτησης.

3 Διαχείριση αρχείων

Σε αυτό το κεφάλαιο θα αναλύσουμε τις βασικές συναρτήσεις των αρχείων του MPI. Πιο ειδικά θα αναλύσουμε τις συναρτήσεις εισόδου-εξόδου του MPI που επιτρέπουν την παράλληλη προσπέλαση αρχείων από το σύνολο των διεργασιών της εφαρμογής.

3.1 Εισαγωγή

Η ταχύτητα αύξησης ισχύος των επεξεργαστών είναι πολύ μεγαλύτερη από την αντίστοιχη αύξηση προσπέλασης των συσκευών εισόδου-εξόδου. Η διαφορά αυτή αυξάνει σημαντικά τον χρόνο εκτέλεσης των εφαρμογών και ελαττώνει αντίστοιχα την συνολική απόδοση ενός παράλληλου συστήματος. Αν θεωρήσουμε ένα σύνολο παράλληλων διεργασιών που χρησιμοποιούν τις λειτουργίες εισόδου-εξόδου, υπάρχουν τρείς διαφορετικές προσεγγίσεις για την υλοποίηση αυτών των λειτουργιών:

1. ***Συλλογική προσέγγιση 2 φάσεων***: σε αυτή την φάση ορίζουμε μία διεργασία ρίζα και αυτή η διεργασία θα πρέπει να υλοποιεί όλες τις λειτουργίες εισόδου-εξόδου. Συγκεκριμένα στην περίπτωση της ανάγνωσης η διεργασία ρίζα αναλαμβάνει την ανάγνωση των κατάλληλων δεδομένων και την διανομή τους στις διεργασίες που τρέχουν στο σύστημα. Αντίθετα στην περίπτωση της εγγραφής έχουμε ότι η κάθε διεργασία θα στείλει τα δικά της δεδομένα στην διεργασία ρίζα, η οποία θα τα συλλέξει και θα τα αποθηκεύσει στα κατάλληλα αρχεία του συστήματος. Το *πλεονέκτημα* της είναι ότι χαρακτηρίζεται από ικανοποιητικό ρυθμό απόδοσης (λίγες διεργασίες και μικρό όγκο δεδομένων) αλλά το *μειονέκτημα* της είναι η μεγάλη χρονική καθυστέρηση για μεγάλο πλήθος διεργασιών.

2. Προσέγγιση στην οποία κάθε διεργασία χρησιμοποιεί ***το δικό της αποκλειστικό αρχείο εξόδου*** για την εγγραφή των δεδομένων της: το *πλεονέκτημα* που έχει αυτή η προσέγγιση είναι ότι είναι καλή λύση όταν τα αρχεία εξόδου δημιουργούνται σε διατάξεις αποθήκευσης που θεωρούνται τοπικές σε σχέση με την διεργασία που τα δημιουργεί. Το *μειονέκτημα* που έχει είναι ότι η απόδοση μειώνεται σημαντικά εάν η τρέχουσα έξοδος που αποθηκεύεται στο αρχείο πρόκειται να χρησιμοποιηθεί στο μέλλον από άλλες τις διεργασίες της εφαρμογής και ακόμη περισσότερο εάν τα αρχεία αποθηκεύονται σε απομακρυσμένες συσκευές αποθήκευσης.
3. Προσέγγιση ***ταυτόχρονης αποθήκευσης εξόδου***: επιβάλει σε όλες τις διεργασίες της εφαρμογής να αποθηκεύσουν την έξοδό τους την ίδια χρονική στιγμή. Σε αυτή την προσέγγιση το αρχείο εξόδου είναι κοινό για όλες τις διεργασίες. Αυτό που κάνει η κάθε διεργασία είναι ότι ανοίγει το ίδιο αρχείο εξόδου, υπολογίζει την θέση μέσα στο αρχείο στην οποία πρέπει να αποθηκεύσει τα δεδομένα, μεταφέρεται σε αυτή την θέση και αποθηκεύει τα δεδομένα. Το *μειονέκτημα* αυτής της προσέγγισης είναι ότι όλες οι προσπελάσεις στο μέσο αποθήκευσης πρέπει να γίνουν ταυτόχρονα, κάτι το οποίο ενδεχομένως δεν υποστηρίζεται πάντα.

Τι γίνεται στην πράξη;

Από τις παραπάνω προσεγγίσεις που έχουν αναφερθεί καμία από τις τρείς προσεγγίσεις δεν διασφαλίζει την παράλληλη αποθήκευση δεδομένων από τις διεργασίες του συστήματος. Για αυτό τον λόγο χρησιμοποιούμε ένα σύνθετο μοντέλο προσπέλασης σε συσκευές εισόδου-εξόδου που συνδυάζουν την *ασύγχρονη* και την *παράλληλη μέθοδο προσπέλασης*. Μια διαδικασία εισόδου-εξόδου χαρακτηρίζεται ως ***ασύγχρονη*** όταν η προσπέλαση στο μέσο αποθήκευσης μπορεί να καθυστερήσει έτσι ώστε να συνεχίσει η διαδικασία της επεξεργασίας των δεδομένων ενώ το υποσύστημα εισόδου-εξόδου μεταφέρει ταυτόχρονα τα δεδομένα από την κεντρική μνήμη στις συσκευές δευτερεύουσας αποθήκευσης. Από την άλλη πλευρά στις ***παράλληλες*** διαδικασίες εισόδου-εξόδου το σύστημα διαθέτει πολλές συσκευές δευτερεύουσας αποθήκευσης όπου ένα αρχείο αποθηκεύεται παράλληλα με τέτοιο τρόπο ώστε κάθε τμήμα του να αποθηκεύεται σε διαφορετική διάταξη. Εάν το τμήμα κάθε αρχείου είναι αποθηκευμένο και σε διαφορετικό μέσο αποθήκευσης τότε είναι δυνατή η προσπέλαση διαφορετικών τμημάτων του ίδιου αρχείου στο σύνολο των διεργασιών του συστήματος.

3.2 Βασικές έννοιες συναρτήσεων εισόδου-εξόδου MPI

Αυτές οι έννοιες είναι η *απόσταση*, το *μέγεθος* και το *τέλος* του αρχείου, ο *δείκτης* του αρχείου και ο *χειριστής* του αρχείου. Η ***απόσταση (offset)*** χρησιμοποιείται για τον καθορισμό κάποιας θέσης εντός του αρχείου. Εξαρτάται άμεσα από τη όψη που χρησιμοποιείται σε κάθε περίπτωση. Είναι εκφρασμένο σε μονάδες eType. Για τον υπολογισμό της ποσότητας eType δεν λαμβάνονται υπόψιν οι κενές περιοχές (holes) που ενδεχομένως εμφανίζονται στον ορισμό του τύπου του αρχείου που σχετίζεται με την τρέχουσα όψη. Για παράδειγμα εάν μια τιμή της απόστασης είναι ίσο με το 0, τότε αυτό αντιστοιχεί στην θέση του πρώτου στοιχείου eType που εμφανίζεται στην τρέχουσα όψη, και προσπελαύνεται αγνοώντας την αρχική μετατόπιση και τις κενές περιοχές αυτής της όψης. Το ***μέγεθος του αρχείου*** είναι εκφρασμένο σε μονάδες bytes και μετράται σε σχέση με την αρχή του αρχείου. Ένα νέο αρχείο του MPI έχει μέγεθος 0 bytes. Χρησιμοποιώντας το μέγεθος του αρχείου ως την τιμή μετατόπισης στον ορισμό της όψης του, μπορούμε να βρούμε την θέση του byte που βρίσκεται αμέσως μετά το τελευταίο byte του αρχείου. Το ***τέλος του αρχείου*** ορίζεται ως η απόσταση του αμέσως επόμενου byte μετά το τέλος του αρχείου μετά το τέλος του αρχείου από την αρχή του αρχείου για την δεδομένη όψη που χρησιμοποιείται κάθε φορά. Ο ***δείκτης του αρχείου*** ορίζεται ως μία μεταβλητή που χρησιμοποιείται σε κάθε διαδικασία προσπέλασης του αρχείου. Για κάθε ***χειριστή του αρχείου*** το MPI διατηρεί και ενημερώνει έναν αποκλειστικό δείκτη αρχείου (individual file pointer) που ορίζεται τοπικά για την κάθε διεργασία, και έναν κοινόχρηστο δείκτη αρχείου (shared file pointer) που είναι κοινός για όλες τις διεργασίες της τρέχουσας ομάδας. Και τέλος ο ***χειριστής του αρχείου*** (file handle) ορίζεται ως ένα αδιαφανές αντικείμενο που δημιουργείται από την συνάρτηση MPI_File_open και αποδεσμεύεται από την συνάρτηση MPI_File_close. Σε σχέση με τις παραδοσιακές συναρτήσεις αρχείων των γλωσσών προγραμματισμού, οι πράξεις με αρχεία λαμβάνουν χώρα χρησιμοποιώντας τον εν λόγω χειριστή. Στην γλώσσα προγραμματισμού C ο χειριστής ενός αρχείου ανήκει στον ειδικό τύπο δεδομένων MPI_File.

3.3 Οι βασικές συναρτήσεις διαχείρισης αρχείων

Όνομα συνάρτησης	Περιγραφή λειτουργίας
MPI_File_open()	Χρησιμοποιείται για το άνοιγμα του αρχείου

	του MPI
MPI_File_close()	Χρησιμοποιείται για το κλείσιμο του αρχείου του MPI.
MPI_File_delete()	Χρησιμοποιείται για την διαγραφή του αρχείου του MPI
MPI_File_set_size()	Επιτρέπει τον καθορισμό του μεγέθους ενός αρχείου
MPI_File_preallocate()	Επιτρέπει την δέσμευση χώρου για ένα αρχείο
MPI_File_get_size()	Επιστρέφει το μέγεθος ενός αρχείου του MPI
MPI_File_get_group()	Επιστρέφει την ομάδα του αρχείου του MPI
MPI_File_get_info()	Επιστρέφει τα χαρακτηριστικά ενός αρχείου του MPI
MPI_File_set_info()	Επιτρέπει τον καθορισμό των χαρακτηριστικών του αρχείου

Πίνακας 2: Συναρτήσεις διαχείρισης αρχείων του MPI (Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

Η συνάρτηση MPI_File_open()

Η συνάρτηση αυτή επιτρέπει το άνοιγμα ενός αρχείου του MPI.

Η δήλωσή της είναι:

int MPI_File_open (MPI_Comm comm, char * fileName, int accessMode, MPI_Info info, MPI_File filehandle);

όπου το όρισμα comm είναι ο communicator στην ομάδα του οποίου ανήκουν οι διεργασίες που καλούν την συνάρτηση, το όρισμα filename είναι το όνομα του αρχείου που επιθυμούμε να ανοίξουμε, το όρισμα accessMode είναι η κατάσταση στην οποία θα ανοίξει το εν λόγω αρχείο, το όρισμα info περιέχει επιπρόσθετες πληροφορίες σχετικά με τα χαρακτηριστικά και τις ιδιότητες του αρχείου και το όρισμα filehandle θα περιέχει τον χειριστή του αρχείου για την διαχείριση και την προσπέλαση του αρχείου στις διερ-

γασίες της εφαρμογής μετά την επιστροφή της συνάρτησης. Το όρισμα accessMode μπορεί να πάρει να πάρει τις παρακάτω τιμές που φαίνονται στον παρακάτω πίνακα:

Όρισμα Συνάρτησης	Περιγραφή Λειτουργίας
MPI_MODE_RONLY	Το αρχείο ανοίγει μόνο σε κατάσταση ανάγνωσης
MPI_MODE_RDWR	Το αρχείο ανοίγει μόνο σε κατάσταση ανάγνωσης/εγγραφής
MPI_MODE_WRONLY	Το αρχείο ανοίγει μόνο σε κατάσταση εγγραφής
MPI_MODE_CREATE	Το αρχείο δημιουργείται εάν δεν υπάρχει
MPI_MODE_EXCL	Η συνάρτηση δημιουργεί σφάλμα εισόδου-εξόδου εάν δημιουργήσουμε υπάρχον αρχείο
MPI_MODE_DELETE_ON_CLOSE	Το αρχείο διαγράφεται αυτόματα αμέσως μετά τον τερματισμό της χρήσης του
MPI_MODE_UNIQUE_OPEN	Το αρχείο μπορεί να ανοίξει μόνο από μία διεργασία της ομάδας και όχι από πολλές διεργασίες ταυτόχρονα
MPI_MODE_SEQUENTIAL	Η προσπέλαση δεδομένων του αρχείου γίνεται με σειριακό τρόπο
MPI_MODE_APPEND	Το αρχείο ανοίγει σε κατάσταση εγγραφής με τον δείκτη του αρχείου μετά το τέλος του.

Πίνακας 3: Οι τιμές του ορίσματος accessMode (Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

Η συνάρτηση MPI_File_close()

Η συνάρτηση αυτή χρησιμοποιείται για το κλείσιμο ενός αρχείου του MPI.

Η δήλωσή της είναι:

int MPI_File_close (MPI_File * fileHandle);

όπου το όρισμα fileHandle είναι ο δείκτης αρχείου που επιστράφηκε από την συνάρτηση MPI_File_open κατά το άνοιγμα του αρχείου.

Η συνάρτηση MPI_File_delete()

Η συνάρτηση αυτή χρησιμοποιείται για την διαγραφή ενός αρχείου του MPI.

Η δήλωσή της είναι:

int MPI_File_delete (char * fileName, MPI_Info info);

όπου το όρισμα filename είναι το όνομα του αρχείου που επιθυμούμε να διαγράψουμε και το όρισμα info παρέχει πληροφορίες σχετικά με τις ιδιότητες και τα χαρακτηριστικά του εν λόγω αρχείου. Εάν δεν μας ενδιαφέρουν οι πληροφορίες μπορούμε να χρησιμοποιήσουμε την τιμή MPI_INFO_NULL για το πεδίο info.

Η συνάρτηση MPI_File_set_size()

Η συνάρτηση αυτή επιτρέπει τον καθορισμό και την ανάκτηση του αρχείου του MPI.

Η δήλωσή της είναι:

int MPI_File_set_size (MPI_File filehandle, MPI_Offset size);

όπου το όρισμα filehandle είναι ο χειριστής του αρχείου που έχει επιστραφεί από την συνάρτηση MPI_File_open, ενώ το όρισμα size περιέχει το νέο μέγεθος του αρχείου που είναι εκφρασμένο σε μονάδες bytes και ανήκει στον τύπο δεδομένων MPI_Offset. Αυτή η συνάρτηση καλείται από όλες τις διεργασίες της τρέχουσας ομάδας διεργασιών με την ίδια τιμή για το όρισμα size.

Η συνάρτηση MPI_File_get_size()

Η συνάρτηση αυτή επιστρέφει το μέγεθος ενός αρχείου του MPI.

Η δήλωσή της είναι:

int MPI_File_get_size (MPI_File filehandle, MPI_Offset * size);

όπου το όρισμα filehandle είναι ο χειριστής αρχείου όπου επιθυμούμε να ανακτήσουμε το μέγεθός του και το όρισμα size περιέχει το μέγεθος του αρχείου (μονάδες bytes) μετά την επιστροφή της συνάρτησης.

Η συνάρτηση MPI_File_preallocate()

Η συνάρτηση αυτή επιτρέπει την δέσμευση αποθηκευτικού χώρου για κάποιο αρχείο του MPI.

Η δήλωσή της είναι:

int MPI_File_preallocate (MPI_File filehandle, MPI_Offset size);

όπου το όρισμα filehandle αναφέρεται στον χειριστή του αρχείου όπου επιθυμούμε την δέσμευση αποθηκευτικού χώρου και το όρισμα size είναι το μέγεθος αυτού του χώρου εκφρασμένο σε μονάδες bytes.

Η συνάρτηση MPI_File_get_group()

Η συνάρτηση αυτή επιστρέφει την ομάδα διεργασιών που άνοιξαν αυτό το αρχείο.

Η δήλωσή της είναι:

int MPI_File_get_group (MPI_File filehandle, MPI_Group *group);

όπου το όρισμα group περιέχει ένα αντίγραφο της ομάδας διεργασιών που χρησιμοποιεί το αρχείο.

Η συνάρτηση MPI_File_get_amode()

Η συνάρτηση αυτή επιστρέφει σε κατάσταση πρόσβασης (access mode) στην οποία έχει ανοίξει το αρχείο.

Η δήλωσή της είναι:

int MPI_File_get_amode (MPI_File filehandle, int * accessMode);

όπου το όρισμα accessMode περιέχει την κατάσταση πρόσβασης που έχει διαβιβαστεί ως όρισμα στην συνάρτηση MPI_File_open που χρησιμοποιήθηκε για το άνοιγμα του αρχείου.

3.4 Συναρτήσεις εισόδου-εξόδου του MPI

3.4.1 Προσπέλαση με την βοήθεια της ακριβούς θέσης

Η πρώτη κατηγορία των συναρτήσεων είναι εκείνες οι συναρτήσεις που προσπελαύνουν δεδομένα με βάση τον ακριβή καθορισμό της θέσης του αρχείου όπου θα ξεκινήσει η ανάγνωση ή η αποθήκευση των δεδομένων. Υπάρχουν οκτώ τέτοιες συναρτήσεις όπου η έξι από αυτές συντάσσονται με το ίδιο τρόπο δηλαδή έχουν ακριβώς τα ίδια ορίσματα και η σημασία των ορισμάτων τους είναι ίδια σε όλες τις περιπτώσεις. Οι δύο

τελευταίες συναρτήσεις χαρακτηρίζονται ως split collective routines και θα παρουσιαστούν σε επόμενες παραγράφους. Η δήλωση των συναρτήσεων αυτών και τον ρόλο που παίζει η κάθε μία από αυτές παρουσιάζονται παρακάτω:

- **int MPI_File_read_at (MPI_File filehandle, MPI_Offset offset, void *buffer, int count, MPI_Datatype dataType, MPI_Status *status):** η συνάρτηση αυτή επιτρέπει την ανάγνωση δεδομένων από το αρχείο. Λειτουργεί με παρεμποδιστικό τρόπο και καλείται από την τρέχουσα διεργασία της εφαρμογής.
- **int MPI_File_write_at (MPI_File filehandle, MPI_Offset offset, void *buffer, int count, MPI_Datatype dataType, MPI_Status status);** η συνάρτηση αυτή επιτρέπει την εγγραφή δεδομένων σε αρχείο. Λειτουργεί με παρεμποδιστικό τρόπο και καλείται από την τρέχουσα διεργασία της εφαρμογής.
- **int MPI_File_read_at_all (MPI_File filehandle, MPI_Offset offset, void *buffer, int count, MPI_Datatype dataType, MPI_Status status);** η συνάρτηση αυτή επιτρέπει την ανάγνωση δεδομένων από το αρχείο. Λειτουργεί με παρεμποδιστικό τρόπο και καλείται από όλες τις διεργασίες της τρέχουσας ομάδας.
- **int MPI_File_write_at_all (MPI_File filehandle, MPI_Offset offset, void *buffer, int count, MPI_Datatype dataType, MPI_Status status);** η συνάρτηση αυτή επιτρέπει την εγγραφή δεδομένων σε αρχείο. Λειτουργεί με παρεμποδιστικό τρόπο και καλείται από όλες τις διεργασίες της τρέχουσας ομάδας.
- **int MPI_File_iread_at (MPI_File filehandle, MPI_Offset offset, void *buffer, int count, MPI_Datatype dataType, MPI_Request * request);** η συνάρτηση αυτή επιτρέπει την ανάγνωση δεδομένων από αρχείο. Λειτουργεί με μη παρεμποδιστικό τρόπο και καλείται από την τρέχουσα διεργασία της εφαρμογής.
- **int MPI_File_iwrite_at (MPI_File filehandle, MPI_Offset offset, void *buffer, int count, MPI_Datatype dataType, MPI_Request * request);** η συνάρτηση αυτή επιτρέπει την εγγραφή δεδομένων σε αρχείο. Λειτουργεί με μη παρεμποδιστικό τρόπο και καλείται από την τρέχουσα διεργασία της εφαρμογής.

Η σημασία των ορισμάτων είναι η εξής:

- το όρισμα ***filehandle*** είναι ο χειριστής προσπέλασης του αρχείου που έχει επιστραφεί από την συνάρτηση MPI_File_open,
- το όρισμα ***offset*** καθορίζει την θέση εντός του αρχείου όπου θα αρχίσει η ανάγνωση η εγγραφή των δεδομένων (σε μονάδες eType),
- το όρισμα ***buffer*** είναι η περιοχή μνήμης που θα αποθηκευτούν τα ανακτημένα δεδομένα,
- το όρισμα ***datatype*** είναι ο τύπος των δεδομένων,
- το όρισμα ***count*** είναι το πλήθος των δεδομένων προς ανάκτηση,
- το όρισμα ***status*** περιέχει πληροφορίες σχετικά με τα δεδομένα που έχουν αναγνωστεί η αποθηκευτεί (με τύπο δεδομένων MPI_Status) και
- το όρισμα ***request*** επιτρέπει τον έλεγχο ολοκλήρωσης των συναρτήσεων που είναι οι MPI_Wait και MPI_Test.

3.4.2 Προσπέλαση με την βοήθεια του ατομικού δείκτη

Στην κατηγορία αυτή δεν απαιτείται καθορισμός επακριβούς θέσης μέσα στο αρχείο. Οι συναρτήσεις στηρίζονται στην χρήση του ατομικού δείκτη αρχείου. Ο δείκτης είναι μοναδικός για κάθε χειριστή αρχείου (filehandle) και για κάθε διεργασία. Η ενημέρωση του γίνεται αυτόματα από το MPI μέσω του δείκτη αρχείου. Μετά την ολοκλήρωση της ανάγνωσης της εγγραφής ο δείκτης δείχνει αμέσως μετά το τελευταίο στοιχείο που προσπελάστηκε. Επίσης οι συναρτήσεις αυτές που ανήκουν σε αυτή την κατηγορία ενημερώνουν τον ατομικό δείκτη του αρχείου της τρέχουσας εγγραφής και όχι τον κοινόχρηστο δείκτη του αρχείου ο οποίος προσπελαύνεται με διαφορετικό τρόπο. Οι συναρτήσεις αυτές που ανήκουν σε αυτή την κατηγορία είναι οι παρακάτω:

- **int MPI_File_read (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Status * status);** Η συνάρτηση επιτρέπει την ανάγνωση δεδομένων από αρχείο, είναι παρεμποδιστική και καλείται από την τρέχουσα διεργασία.
- **int MPI_File_write (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Status *status);** Η συνάρτηση επιτρέπει την εγγραφή δεδομένων από αρχείο, είναι παρεμποδιστική και καλείται από την τρέχουσα διεργασία.
- **int MPI_File_read_all (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Status * status);** Η συνάρτηση επιτρέπει την

ανάγνωση δεδομένων από αρχείο, είναι μη παρεμποδιστική και καλείται από όλες τις διεργασίες.

- **int MPI_File_write_all (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Status * status);** Η συνάρτηση επιτρέπει την εγγραφή δεδομένων από αρχείο, είναι μη παρεμποδιστική και καλείται από όλες τις διεργασίες.
- **int MPI_File_iread (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Request * request);** Η συνάρτηση επιτρέπει την ανάγνωση δεδομένων από αρχείο, είναι μη παρεμποδιστική και καλείται από την τρέχουσα διεργασία.
- **int MPI_File_iwrite (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Request * request);** Η συνάρτηση επιτρέπει την εγγραφή δεδομένων από αρχείο, είναι μη παρεμποδιστική και καλείται από την τρέχουσα διεργασία. Η σημασία των ορισμάτων είναι ίδια όπως και στην προηγούμενη κατηγορία των συναρτήσεων. Η μόνη διαφορά στην σύνταξη αυτών των συναρτήσεων είναι ότι λείπει το όρισμα offset γιατί η θέση της τελευταίας προσπέλασης του αρχείου γίνεται από την τρέχουσα τιμή του ατομικού δείκτη όποτε δεν χρειάζεται να καθοριστεί μέσω του ορίσματος offset. Επιπρόσθετα υπάρχουν τρείς επιπλέον συναρτήσεις οι οποίες στηρίζονται στην χρήση του ατομικού δείκτη αρχείου:
- **int MPI_File_seek (MPI_File filehandle, MPI_Offset offset, int whence);** η συνάρτηση αυτή χρησιμοποιείται για την μεταβολή της τιμής του ατομικού δείκτη αρχείου. Το όρισμα offset χρησιμοποιείται για να καθορίσει την νέα θέση του ατομικού δείκτη αρχείου ενώ το όρισμα whence χρησιμοποιείται για τον τρόπο που θα λάβει χώρα η τιμή του μεγέθους. Η μεταβλητή whence παίρνει τις τιμές MPI_SEEK_SET, MPI_SEEK_CUR, MPI_SEEK_END. Εάν η τιμή της παραμέτρου είναι η σταθερά MPI_SEEK_SET τότε η νέα θέση του ατομικού δείκτη θα είναι αυτή που θα υπαγορεύεται από το όρισμα offset. Εάν η παράμετρος whence είναι η σταθερά MPI_SEEK_CUR τότε η νέα θέση προκύπτει εάν στην τρέχουσα τιμή προσθέσουμε το όρισμα offset.Τέλος εάν η τιμή της παραμέτρου whence είναι η σταθερά MPI_SEEK_END τότε ο ατομικός δείκτης αρχείου τοποθετείται offset μονάδες μετά το τέλος του αρχείου.

- **int MPI_File_get_position (MPI_File filehandle, MPI_Offset offset);** η συνάρτηση αυτή χρησιμοποιείται για την ανάκτηση της τρέχουσας τιμής του ατομικού δείκτη αρχείου. Το όρισμα filehandle χρησιμοποιείται στον χειριστή του αρχείου που έχει επιστραφεί από την συνάρτηση MPI_File_open() ενώ το όρισμα offset χρησιμοποιείται για την τρέχουσα τιμή του ατομικού δείκτη αρχείου που είναι εκφρασμένο σε μονάδες eType από την αρχή της τρέχουσας όψης.
- **int MPI_File_get_byte_offset (MPI_File filehandle, MPI_Offset offset, MPI_Offset *disp);** η συνάρτηση αυτή χρησιμοποιείται για την μετατροπή της τιμής της απόστασης της τρέχουσας θέσης από την αρχή της τρέχουσας όψης (σε απόλυτη απόσταση). Το όρισμα disp μετά την επιστροφή της συνάρτησης θα περιέχει την απόλυτη τιμή της απόστασης της τρέχουσας θέσης που είναι εκφρασμένη ως προς την αρχή του αρχείου.

3.4.3 Προσπέλαση με την βοήθεια του κοινόχρηστου δείκτη αρχείου

Οι συναρτήσεις αυτής της κατηγορίας ορίζονται ξεχωριστά για κάθε αρχείο και για κάθε διεργασία σε αντίθεση με τους ατομικούς δείκτες. Το MPI διατηρεί και ενημερώνει έναν και μοναδικό κοινόχρηστο δείκτη αρχείου. Για να είναι δυνατή η χρήση των συναρτήσεων που στηρίζονται στον κοινόχρηστο δείκτη θα πρέπει να χρησιμοποιούν την ίδια όψη για την προσπέλαση αρχείου, στην αντίθετη περίπτωση λαμβάνει χώρα η εμφάνιση σφαλμάτων εισόδου-εξόδου.

Κανόνες του τρόπου κλήσης των συναρτήσεων που χρησιμοποιούν τον κοινόχρηστο δείκτη για την προσπέλαση των δεδομένων

Η απόσταση (offset) εντός του αρχείου ορίζεται ως η τρέχουσα τιμή του κοινόχρηστου δείκτη αρχείου. Δηλαδή η τρέχουσα θέση με την οποία θα διαβάσουμε η θα ανακτήσουμε δεδομένα γίνεται αυτόματα από το MPI χωρίς να χρειάζεται να καθοριστεί από τον χρήστη. Και επιπλέον το αποτέλεσμα των πολλαπλών κλήσεων σε συναρτήσεις που χρησιμοποιούν τον κοινόχρηστο δείκτη αρχείου είναι το ίδιο εάν αυτές οι κλήσεις λαμβάνουν χώρα με σειριακό τρόπο.

Οι συναρτήσεις αυτές που υποστηρίζουν αυτή την κατηγορία είναι η εξής:

- **int MPI_File_read_shared (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Status * status);** Η συνάρτηση επιτρέπει την ανάγνωση δεδομένων από αρχείο, είναι παρεμποδιστική και καλείται από την τρέχουσα διεργασία.

- **int MPI_File_write_shared (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Status * status);** Η συνάρτηση επιτρέπει την εγγραφή δεδομένων από αρχείο, είναι παρεμποδιστική και καλείται από την τρέχουσα διεργασία.
- **int MPI_File_iread_shared (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Request * request);** Η συνάρτηση επιτρέπει την ανάγνωση δεδομένων από αρχείο, είναι μη παρεμποδιστική και καλείται από την τρέχουσα διεργασία.
- **int MPI_File_iwrite_shared (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Request * request);** Η συνάρτηση επιτρέπει την εγγραφή δεδομένων από αρχείο, είναι μη παρεμποδιστική και καλείται από την τρέχουσα διεργασία.
- **int MPI_File_read_ordered (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Status * status);** Η συνάρτηση επιτρέπει την ανάγνωση δεδομένων από αρχείο, είναι μη παρεμποδιστική και καλείται από την τρέχουσα διεργασία.
- **int MPI_File_write_ordered (MPI_File fHandle, void *buf, int count, MPI_Datatype datatype, MPI_Status * status);** Η συνάρτηση επιτρέπει την εγγραφή δεδομένων από αρχείο, είναι μη παρεμποδιστική και καλείται από την τρέχουσα διεργασία.
- **int MPI_File_seek_shared (MPI_File fileHandle, MPI_Offset offset, int whence);** επιστρέφει τον καθορισμό του κοινόχρηστου δείκτη αρχείου.
- **int MPI_File_get_position_shared (MPI_File filehandle, MPI_Offset * offset);** επιστρέφει την τιμή του κοινόχρηστου δείκτη αρχείου.

3.4.4 Split-Collective Routines

Γενικά

Οι συναρτήσεις αυτές μπορούν να θεωρηθούν ως μη παρεμποδιστικές συλλογικές συναρτήσεις εισόδου-εξόδου. Η λειτουργία τους αποτελείται από δύο φάσεις οι οποίες είναι η φάση εκκίνησης, η οποία συσχετίζεται με μία begin routine και η φάση τερματισμού που συσχετίζεται με μια end routine. Σε πλήρη αναλογία με τις συνήθεις μη παρε-

μποδιστικές ο χρήστης δεν μπορεί να προσπελάσει τα δεδομένα, εάν δεν ολοκληρωθεί η λειτουργία της διαδικασίας εκκίνησης. Μπορούμε να αντιστοιχήσουμε αυτές τις δύο διαδικασίες με τον συνήθη τρόπο λειτουργίας των μη παρεμποδιστικών διαδικασιών οι οποίες είναι η MPI_Iread και η MPI_Iwrite που είναι ρουτίνα εκκίνησης που εκκινούν μια διαδικασία αποστολής ή λήψης των δεδομένων και η MPI_Wait η MPI_Test οι οποίες είναι ρουτίνες τερματισμού που ελέγχουν την ολοκλήρωση η όχι της τρέχουσας μη παρεμποδιστικής διαδικασίας.

Κανόνες μη παρεμποδιστικών συλλογικών συναρτήσεων εισόδου-εξόδου

Οι μη παρεμποδιστικές συλλογικές συναρτήσεις εισόδου-εξόδου υποστηρίζονται από το MPI που δέχονται ως όρισμα τον χειριστή αρχείου που έχει επιστραφεί από την συνάρτηση MPI_File_open. Η λειτουργία των μη παρεμποδιστικών συναρτήσεων εισόδου-εξόδου στηρίζεται σε ένα σύνολο κανόνων που είναι οι ακόλουθοι:

- Σε κάθε διεργασία MPI, ο κάθε χειριστής αρχείου θα συσχετιστεί με μία το πολύ ενεργή μη παρεμποδιστική συλλογική διαδικασία εισόδου-εξόδου.
- Η κλήση της ρουτίνας εκκίνησης είναι μία συλλογική διαδικασία. Η χρονική σειρά της εκτέλεσής τους υπαγορεύεται από τους κανόνες των συλλογικών διαδικασιών.
- Ο τρόπος κλήσης των διαδικασιών τερματισμού εναρμονίζεται με τους κανόνες των συλλογικών διαδικασιών. Η κάθε κλήση διαδικασίας τερματισμού συσχετίζεται υποχρεωτικά με κάποια προγενέστερη κλήση διαδικασίας εκκίνησης. Το ζεύγος των διαδικασιών εκκίνησης και τερματισμού θεωρείται μια πλήρης διαδικασία προσπέλασης των δεδομένων.
- Οι ρουτίνες εκκίνησης και τερματισμού χρησιμοποιούν την ίδια ενδιάμεση περιοχή αποθήκευσης.
- Ανάμεσα στην ρουτίνα εκκίνησης και στην ρουτίνα τερματισμού δεν θα πρέπει να παρεμβάλλονται άλλες συλλογικές διαδικασίες αυτού του είδους. Στην αντίθετη περίπτωση θα λάβει χώρα η εμφάνιση σφαλμάτων εισόδου-εξόδου.
- Σε μία πολυνηματική εφαρμογή (multithreaded application) οι ρουτίνες εκκίνησης και τερματισμού μιας συλλογικής μη παρεμποδιστικής διαδικασίας εισόδου-εξόδου θα πρέπει να καλούνται μέσα από το ίδιο thread.

  Οι συναρτήσεις αυτής της κατηγορίας είναι οι εξής:

- **int MPI_File_read_at_all_begin (MPI_File *filehandle, MPI_Offset offset, void *buffer, int count, MPI_Datatype datatype);** η συνάρτηση αυτή είναι ρουτίνα ανάγνωσης που επιτρέπει την συλλογική ανάγνωση των δεδομένων και στηρίζεται στον επακριβή καθορισμό της θέσης ανάγνωσης εντός του αρχείου.
- **int MPI_File_write_at_all_begin (MPI_File *filehandle, MPI_Offset offset, void *buffer, int count, MPI_Datatype datatype);** η συνάρτηση αυτή είναι ρουτίνα εκκίνησης που επιτρέπει την συλλογική εγγραφή δεδομένων και στηρίζεται στον επακριβή καθορισμό της θέσης ανάγνωσης εντός του αρχείου.
- **int MPI_File_read_all_begin (MPI_File *filehandle, void *buffer, int count, MPI_Datatype datatype):** η συνάρτηση αυτή είναι ρουτίνα εκκίνησης που επιτρέπει την συλλογική ανάγνωση δεδομένων και στηρίζεται στην τιμή του αποκλειστικού δείκτη αρχείου.
- **int MPI_File_write_at_all_begin (MPI_File *filehandle, void *buffer, int count, MPI_Datatype datatype):** η συνάρτηση αυτή είναι ρουτίνα εκκίνησης που επιτρέπει την συλλογική εγγραφή δεδομένων. Στηρίζεται στην τιμή του αποκλειστικού δείκτη αρχείου.
- **int MPI_File_read_ordered_begin (MPI_File *filehandle, void *buffer, int count, MPI_Datatype datatype);** η συνάρτηση αυτή είναι ρουτίνα εκκίνησης που επιτρέπει την συλλογική ανάγνωση δεδομένων. Στηρίζεται στην τιμή του κοινόχρηστου δείκτη αρχείου.
- **int MPI_File_write_ordered_begin (MPI_File *filehandle, void *buffer, int count, MPI_Datatype datatype);** η συνάρτηση αυτή είναι ρουτίνα εκκίνησης που επιτρέπει την συνολική εγγραφή δεδομένων. Στηρίζεται στην τιμή του κοινόχρηστου δείκτη αρχείου.
- **int MPI_File_read_at_all_end (MPI_File *filehandle, void *buffer, MPI_Status* status);** η συνάρτηση αυτή είναι ρουτίνα τερματισμού που επιτρέπει την συλλογική ανάγνωση των δεδομένων. Στηρίζεται στον επακριβή καθορισμό της θέσης ανάγνωσης εντός του αρχείου.
- **int MPI_File_write_at_all_end (MPI_File *filehandle, void *buffer, MPI_Status status);** η συνάρτηση αυτή είναι ρουτίνα τερματισμού που επιτρέπει την συλλογική εγγραφή δεδομένων. Στηρίζεται στον επακριβή καθορισμό της θέσης ανάγνωσης εντός του αρχείου.

- **int MPI_File_read_all_end (MPI_File *filehandle, void *buffer, MPI_Status* status);** η συνάρτηση αυτή είναι ρουτίνα τερματισμού που επιτρέπει την συλλογική ανάγνωση των δεδομένων. Στηρίζεται στην τιμή του αποκλειστικού δείκτη αρχείου.
- **int MPI_File_write_all_end (MPI_File *filehandle, void *buffer, MPI_Status status);** η συνάρτηση αυτή είναι ρουτίνα τερματισμού που επιτρέπει την συλλογική εγγραφή δεδομένων. Στηρίζεται στην τιμή του αποκλειστικού δείκτη αρχείου.
- **int MPI_File_read_ordered_end (MPI_File *filehandle, void *buffer, MPI_Status * status);** η συνάρτηση αυτή είναι ρουτίνα τερματισμού που επιτρέπει την συλλογική ανάγνωση δεδομένων. Στηρίζεται στην τιμή του κοινόχρηστου δείκτη αρχείου.
- **int MPI_File_write_ordered_end (MPI_File *filehandle, void *buffer, MPI_Status status);** η συνάρτηση αυτή είναι ρουτίνα τερματισμού που επιτρέπει την συνολική εγγραφή δεδομένων. Στηρίζεται στην τιμή του κοινόχρηστου δείκτη αρχείου.

Οι συναρτήσεις αυτής της κατηγορίας που περιγράψαμε μπορούν να φανούν σε ένα συγκεντρωτικό πίνακα ο οποίος είναι ο παρακάτω:

POSITIONING	SYNCHRONISM	COORDINATION	
		NON-COLLECTIVE	COLLECTIVE
Explisit offsets	Blocking	MPI_File_read_at	MPI_File_read_at_all
		MPI_File_write_at	MPI_File_write_at_all
	Non Blocking and split collective	MPI_File_iread_at	MPI_File_read_at_all_begin
			MPI_File_read_at_all_end
		MPI_File_iwrite_at	MPI_File_write_at_all_begin
			MPI_File_write_at_all_end
Individual file pointers	Blocking	MPI_File_read	MPI_File_read_all
		MPI_File_write	MPI_File_write_all
	Non Blocking and split collective	MPI_File_iread	MPI_File_read_all_begin
			MPI_File_read_all_end
		MPI_File_iwrite	MPI_File_write_all_begin
			MPI_File_write_all_end
Shared file pointer	Blocking	MPI_File_read_shared	MPI_File_read_ordered
		MPI_File_write_shared	MPI_File_write_ordered
	Non Blocking and split collective	MPI_File_iread_shared	MPI_File_read_ordered_begin
			MPI_File_read_ordered_end
		MPI_File_iwrite_shared	MPI_File_write_ordered_begin
			MPI_File_write_ordered_end

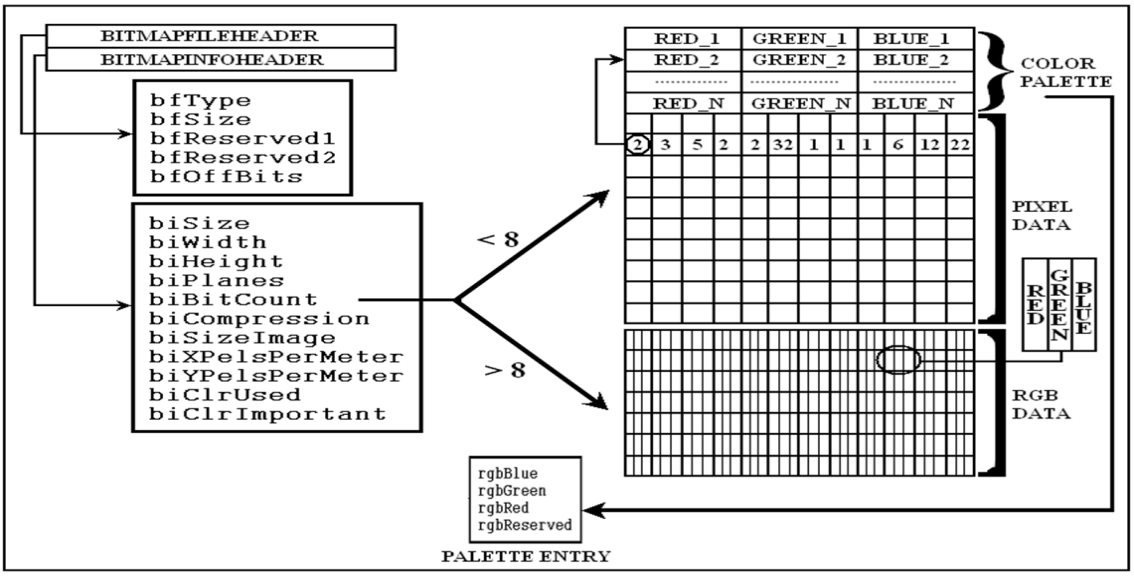
Πίνακας 4: Οι συναρτήσεις εισόδου-εξόδου του MPI (Αθανάσιος Ι. Μάργαρης MPI Θεωρία & Εφαρμογές Εκδόσεις Τζιόλα 2014)

3.5 Παραδείγματα χρήσης των συναρτήσεων εισόδου-εξόδου

3.5.1 Παράλληλη ανάγνωση στοιχείων εικόνας bmp

Το πρώτο μας παράδειγμα είναι η ανάγνωση των περιεχομένων ενός αρχείου εικόνας το οποίο βρίσκεται αποθηκευμένο σε μορφή BMP δηλαδή τα εικονοστοιχεία εκείνα που βρίσκονται στις διεργασίες του συστήματος. Με αυτή την τεχνική μπορούμε να υλοποιήσουμε παράλληλους αλγορίθμους επεξεργασίας εικόνας, γιατί η κάθε διεργασία θα επεξεργάζεται τα δικά της δεδομένα παράλληλα με τις υπόλοιπες διεργασίες της εφαρμογής. Στα αρχεία τύπου BMP υπάρχουν δύο επικεφαλίδες που περιέχουν πληροφορίες σχετικά με τα χαρακτηριστικά του αρχείου και της εικόνας που περιέχεται σε αυτό. Στην συνέχεια ακολουθούν τα χρωματικά δεδομένα η μορφή των οποίων εξαρτάται από το πλήθος των χρωμάτων που απαιτούνται για την σχεδίαση της εικόνας.

Η δομή ενός αρχείου εικόνας φαίνεται παρακάτω:



Εικόνα 8: Η δομή ενός αρχείου εικόνας BMP (Αθανάσιος Ι .Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

Στην γλώσσα C οι δύο επικεφαλίδες του αρχείου εικόνας ορίζονται ως ακολούθως

```
typedef struct tagBITMAPFILEHEADER
..UINT bfType;
  DWORD bfsize;
```

```
  UINT bfReserved1;
  UINT bfReserved2;
  DWORD bfOffBits;
 } BITMAPFILEHEADER;


typedef struct tagBITMAPINFOHEADER
  DWORD biSize;
  LONG biWidth;
  LONG biHeight;
  WORD biPlanes;
  WORD biBitCount;
  DWORD biCompression;
  DWORD bisizeImage;
  LONG biXPelsPerMeter;
  LONG biYPelsPerMeter;
  DWORD biClrUsed;
  DWORD biClrImportant;
  } BITMAPINFOHEADER;
```

Αναλυτικά τα πεδία των δύο δομών αναλύονται ως εξής:

1. Πεδία δομής BITMAPFILEHEADER
    - **BfType**: Περιέχει πάντα την **19778** που πιστοποιεί το τρέχον αρχείο εικόνας ως ένα έγκυρο **αρχείο BMP**.
    - **BfSize:** Περιέχει το **μέγεθος του αρχείου** σε bytes.
    - **BfReserved1:** Δεσμευμένο πεδίο που δεν χρησιμοποιείται και έχει πάντα την τιμή ίση με μηδέν.
    - **BdReserved2:** Δεσμευμένο πεδίο που δεν χρησιμοποιείται και έχει πάντα την τιμή ίση με μηδέν.
    - **BfOffBits:** Περιέχει την **απόσταση των χρωματικών δεδομένων** από την αρχή του αρχείου (σε bytes).
2. Πεδία δομής BITMAPINFOHEADER
    - **BiSize:** Περιέχει το μέγεθος της δομής BITMAPINFOHEADER δηλαδή την τιμή sizeof(BITMAPINFOHEADER).
    - **BiWidth:** Περιέχει το πλάτος της εικόνας σε pixels.
    - **BiHeight:** Περιέχει το ύψος της εικόνας σε pixels.
    - **Biplanes:** Περιέχει το πλήθος των χρωματικών επιπέδων εικόνας και έχει την τιμή ίση με την μονάδα.
    - **biBitCount:** Περιέχει το πλήθος των bits που χρησιμοποιούνται για την περιγραφή του κάθε χρώματος (χρωματικό βάθος).
    - **biCompression:** Καθορίζει το είδος της συμπίεσης της εικόνας.
    - **biSizeImage:** Περιέχει το πλήθος των pixels της εικόνας.

- **biXPelsPerMeter:** Περιέχει την ανάλυση της εικόνας κατά την οριζόντια διεύθυνση και μετράται σε μονάδες ppm (pixels per meter).
- **biYPelsPerMeter:** Περιέχει την ανάλυση της εικόνα κατά την κατακόρυφη διεύθυνση και μετράται σε μονάδες ppm (pixels per meter).
- **BiClrUsed**: Περιέχει το πλήθος των χρωμάτων της χρωματικής παλέτας που χρησιμοποιούνται για την σχεδίαση της εικόνας.
- **biClrImportant**: Περιέχει το πλήθος των χρωμάτων που είναι σημαντικά για την σχεδίαση της εικόνας.

Η δομή της χρωματικής παλέτας

Ένα οποιοδήποτε χρώμα μιας εικόνας μπορεί να περιγραφεί με την βοήθεια τριών βασικών χρωμάτων: το κόκκινο, το πράσινο και το μπλε. Η κάθε μία από τις χρωματικές συνιστώσες δέχεται μία τιμή στο διάστημα [0, 255] όποτε η αναπαράσταση της απαιτεί μια μεταβλητή τύπου char. Η δομή της ορίζεται ως εξής:

```
typedef struct tagRGBQUAD{
      BYTE rgbBlue;
      BYTE rgbGreen;
      BYTE rgbRed;
      BYTE rgbReserved;
      } RGBQUAD;
```

Η σημασία των πεδίων είναι η ακόλουθη:

- **rgbBlue, rgbGreen, rgbRed:** περιγράφει τις βασικές συνιστώσες του κάθε χρώματος.
- **rgbReserved:** δεν χρησιμοποιείται, αλλά εμφανίζεται προκειμένου το συνολικό μέγεθος του αρχείου της εικόνας να είναι ακέραιο πολλαπλάσιο του 4.

Διαδικασία ανάγνωσης και επεξεργασίας ενός αρχείου BMP

Αρχικά η διεργασία με τάξη R=0 ανακτά τις βασικές παραμέτρους της εικόνας που είναι τα bfOffBits, biWidth, biBitCount με χρήση των συναρτήσεων εισόδου-εξόδου. Στην συνέχεια αυτή η διεργασία διανέμει τις πληροφορίες σε άλλες διεργασίες της τρέχουσας ομάδας με την χρήση της συνάρτησης MPI_Bcast(). Αυτές οι διεργασίες θα ανοίξουν συλλογικά το αρχείο MPI_File_open, θα ανακτήσουν την χρωματική παλέτα και η κάθε μία θα διαβάσει ένα μέρος των δεδομένων που στην συνέχεια θα επεξεργαστεί. Τέλος, τα νέα δεδομένα θα εγγραφούν εκ νέου στο αρχείο, το οποίο θα περιέχει την εικόνα στην νέα της μορφή.

Ανάκτηση των τιμών των βασικών παραμέτρων της εικόνας

Ανάκτηση της τιμής του πεδίου bfOffbits με την συνάρτηση GetBitmapOffset

```
int GetBitmapDataOffset (char *filename) {
FILE * fp;
BITMAPFILEHEADER bmpheader={0};
fp= fopen (filename, "rb");
if (!fp){
printf ("Bitmap File could not be Opened\n");
return (-1); }
rewind (fp);
fread (&bmpHeader, sizeof (bmpHeader), 1, fp);
fclose(fp);
return (bmpHeader.bfOffBits);
```

Ανάκτηση της τιμής του πεδίου colorNumber με την συνάρτηση GetBitMapColor-Number

```
int GetBitmapColorNumber (char *filename) {
FILE * fp;
int colors;
BITMAPFILEHEADER bmpheader={0};
BITMAPINFOHEADER bmpInfo={0};
fp= fopen (filename, "rb");
if (!fp){
printf ("Bitmap File could not be Opened\n");
return (-1); }
rewind (fp);
fread (&bmpHeader, sizeof (bmpHeader), 1, fp);
fread (&bmpInfo, sizeof(bmpInfo), 1, fp);
fclose(fp);
if (bmpInfo.biBitCount<=8)
colors= (int)pow(2, bmpInfo.biBitCount);
else colors=0;
return (colors); }
```

Ανάκτηση της τιμής του πεδίου biWidth με την συνάρτηση GetBitmapWidth

```
int GetBitmapWidth (char *filename) {
FILE * fp;
BITMAPFILEHEADER bmpheader={0};
BITMAPINFOHEADER bmpInfo={0};
fp= fopen (filename, "rb");
if (!fp){
printf ("Bitmap File could not be Opened\n");
return (-1); }
rewind (fp);
fread (&bmpHeader, sizeof(bmpHeader), 1, fp);
fread (&bmpInfo, sizeof(bmpInfo), 1, fp);
fclose(fp);
return (bmpInfo.biWidth); }
```

Ανάκτηση της τιμής του πεδίου biHeight με την συνάρτηση GetBitmapHeight

```
int GetBitmapHeight (char *filename) {
FILE * fp;
BITMAPFILEHEADER bmpheader={0};
BITMAPINFOHEADER bmpInfo={0};
fp= fopen (filename, "rb");
if (!fp){
printf ("Bitmap File could not be Opened\n");
return (-1); }
rewind (fp);
fread (&bmpHeader, sizeof(bmpHeader), 1, fp);
fread (&bmpInfo, sizeof(bmpInfo), 1, fp);
fclose(fp);
return (bmpInfo.biHeight); }
```

Οι παραπάνω συναρτήσεις θα κληθούν από την διεργασία με τάξη R=0 και οι τιμές που θα επιστραφούν θα δοθούν στις υπόλοιπες διεργασίες με κλήση της συνάρτησης MPI_Bcast(). Η κάθε μία από τις άλλες διεργασίες θα εκτυπώσει τις παραμέτρους που παρέλαβε. Η διαδικασία φαίνεται στον ακόλουθο κώδικα:

```
if (rank==0){
colors= GetBitmapColorNumber (filename);
width= GetBitmapWidth (filename);
height= GetBitMapHeight (filename);
offset= GetBitmapDataOffset (filename);

MPI_Bcast (&colors, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&width, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&height, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&offset, 1, MPI_INT, 0, MPI_COMM_WORLD);
printf ("Process %d parameters:%d colors %d width %d height % d offset\n", rank, colors,
width, height, offset);
```

Εάν εκτελέσουμε αυτόν τον κώδικα για 5 διεργασίες και χρησιμοποιήσουμε ένα γνωστό αρχείο εικόνας των Windows με όνομα clouds.bmp, τα αποτέλεσμα είναι το παρακάτω:

```
Process 0 parameters : 256 colors 640 width 480 height 314 offset
Process 4 parameters : 256 colors 640 width 480 height 314 offset
Process 1 parameters : 256 colors 640 width 480 height 314 offset
Process 2 parameters : 256 colors 640 width 480 height 314 offset
Process 3 parameters : 256 colors 640 width 480 height 314 offset
```

Ανάκτηση χρωμάτων χρωματικής παλέτας

Μετά την ανάγνωση των παραμέτρων, η κάθε διεργασία πρέπει να ανακτήσει τα περιεχόμενα της χρωματικής παλέτας του αρχείου, Υπάρχουν 2 τρόποι να το κάνουμε και το καλύτερο είναι με την χρήση των συναρτήσεων εισόδου-εξόδου του MPI. Η διαδικασία φαίνεται στον παρακάτω κώδικα:

```
//Άνοιγμα του αρχείου:
MPI_File_open (MPI_COMM_WORLD, filename, MPI_MODE_RDWR, MPI_INFO_NULL, &fileHandle);

// Μεταφορά μετά τις επικεφαλίδες BITMAPFILEHEADER και BITMAPINFOHEADER
```

```
MPI_File_seek (fileHandle, disp, MPI_SEEK_SET);

//Ανάγνωση περιεχομένου χρωματικής παλέτας:
MPI_File_read (fileHandle, pallete, 4*colors, MPI_BYTE, &status);

//Υπολογισμός του πλήθους των στοιχείων που διαβάστηκαν:
MPI_Get_elements (&status, MPI_BYTE, &count);
```

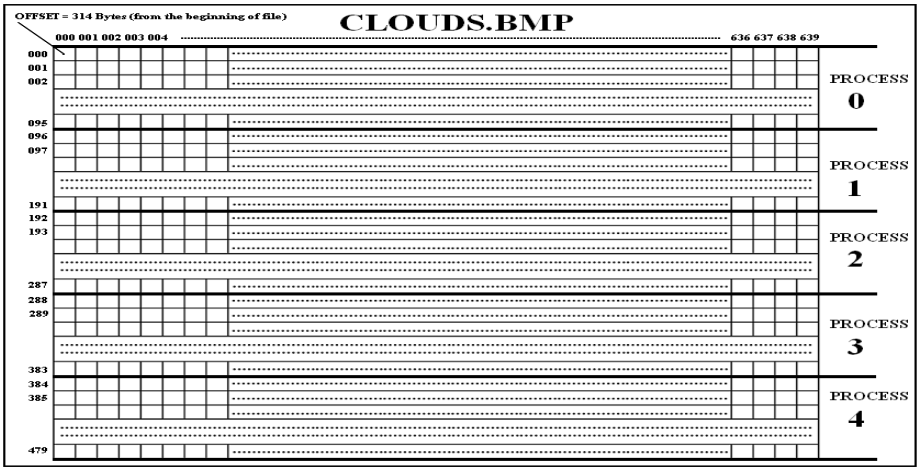
Και η έξοδος είναι η εξής:

```
Process 0 read 1024 items of color palette
Process 4 read 1024 items of color palette
Process 2 read 1024 items of color palette
Process 1 read 1024 items of color palette
Process 3 read 1024 items of color palette
```

Κατανομή των χρωματικών δεδομένων σε διαδικασίες

Μετά την ανάγνωση του αρχείου των πληροφοριών της χρωματικής παλέτας θα πρέπει να κατανέμουμε τα χρωματικά δεδομένα σε διαδικασίες. Αυτό γίνεται με 2 τρόπους:

1. να ορίσουμε μία *διαφορετική όψη για την κάθε διεργασία* έτσι ώστε αυτή να διαβάσει μόνο τα δικά της δεδομένα.
2. να γίνει *υπολογισμός της θέσης μέσα στο αρχείο* από την οποία η κάθε διεργασία θα ανακτήσει τα δικά της δεδομένα, στην μετάβαση του ατομικού δείκτη αρχείου στην θέση που υπολογίστηκε μέσω της συνάρτησης MPI_File_seek και στην ανάγνωση των κατάλληλων δεδομένων με χρήση της συνάρτησης MPI_File_read που στηρίζεται στον ατομικό δείκτη αρχείου. Η κατανομή των χρωματικών δεδομένων μπορεί να αναπαρασταθεί σχηματικά όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 9: Κατανομή των χρωματικών δεδομένων του αρχείου εικόνας clouds.bmp για 5 διεργασίες (Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

Το σχήμα αυτό αφορά την εικόνα clouds.bmp και θεωρούμε ότι τα χρωματικά δεδομένα σχηματίζουν ένα δισδιάστατο πίνακα 640X480. Η κάθε μία από τις 5 διεργασίες θα διαβάσει 96 γραμμές αυτού του πίνακα. Η τιμή του πεδίου bfOffBits της επικεφαλίδας BITMAPFILEHEADER είναι 314 που σημαίνει η διεργασία με τάξη 0 θα ξεκινήσει από αυτή την θέση την ανάγνωση των δεδομένων της και το πλήθος των στοιχείων που θα διαβάσει θα είναι 96X640=61440. Προφανώς, η διεργασία με τάξη 1 θα ξεκινήσει την ανάγνωση των δικών της δεδομένων από την θέση 61440+314=61754 και ανάλογα οι υπόλοιπες διεργασίες. Στο τέλος η τιμή του ατομικού δείκτη αρχείου θα είναι ίση με 246074+61440=307514 που είναι και το συνολικό μέγεθος του αρχείου σε bytes. Η τιμή αυτή είναι και η τιμή του πεδίου bfSize της δομής BITMAPFILEHEADER. Η διαδικασία ανάγνωσης των εικονοστοιχείων μπορεί να φανεί στον παρακάτω κώδικα:

```
rows = height/size;
pixels = (char *) malloc (rows*width*sizeof(char));
if (!pixels) {
printf ("No memory available\n");
MPI_Finalize ();
return (-1); }

MPI_File_seek (fileHandle, offset+rank*rows*width, MPI_SEEK_SET);
MPI_File_get_position (fileHandle, &offset1);
MPI_File_read (fileHandle, pixels, rows*width, MPI_BYTE, &status);
MPI_File_get_position (fileHandle, &offset2);
MPI_Get_elements (&status, MPI_BYTE, &count);
```

Με την εκτέλεση του παραπάνω κώδικα η κάθε διεργασία έχει διαβάσει την περιοχή μνήμης του πίνακα pixels 61440 στοιχεία τύπου MPI_BYTE. Εάν θέλουμε να εκτυπώσουμε την αρχική και τελική θέση ανάγνωσης της κάθε διεργασίας καθώς και το πλήθος των στοιχείων που έχουν αναγνωσθεί θα οδηγηθούμε σε κάποια αποτελέσματα που φαίνονται στον παρακάτω πίνακα:

Τάξη Διεργασίας	Αρχική θέση	Τελική θέση	Πλήθος στοιχείων
00	000314	061754	61440
01	061754	123194	61440
02	123194	184634	61440
03	184634	246074	61440
04	246074	307514	61440

Πίνακας 5: Η αρχική και η τελική θέση ανάγνωσης και το πλήθος των στοιχείων που έχουν ανακτηθεί από τις παράλληλες διεργασίες της εφαρμογής (Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

Εφόσον έχουμε διανέμει τα περιεχόμενα του αρχείου εικόνας στις διεργασίες του συστήματος, μπορούμε τώρα να τα χρησιμοποιήσουμε και να εφαρμόσουμε στην εικόνα clouds.bmp ένα εκτεταμένο σύνολο αλγορίθμων επεξεργασίας. Πιο συγκεκριμένα θα υλοποιήσουμε ένα σχετικά απλό αλγόριθμο που στηρίζεται στην **ανίχνευση ακμών (edge detection)** επί της τρέχουσας εικόνας. Η θεωρία αυτού του αλγορίθμου μας λέει πως ένα σύνολο εικονοστοιχείων θεωρείται πως ορίζει μια ακμή, όταν το χρώμα τους είναι αρκετά διαφορετικό από τα χρώματα των εικονοστοιχείων που βρίσκονται στις παρακείμενες περιοχές. Για να εκτιμήσουμε ποσοτικά την διαφορά αυτή που μπορεί να υπάρχει ανάμεσα στα δύο χρώματα μπορούμε να θεωρήσουμε το κάθε χρώμα ως ένα διάνυσμα με συντεταγμένες (R,G,B), όπου τα σύμβολα R, G και B είναι οι τρείς βασικές χρωματικές συνιστώσες που είναι το κόκκινο, το πράσινο και το μπλε χρώμα. Εάν θεωρήσουμε αυτή την διανυσματική αναπαράσταση δύο χρωμάτων με συντεταγμένες (R1, G1, B1) και (R2,B2, G2), μπορούμε να ορίσουμε την απόστασή τους από την γνωστή σχέση: $d(C_1,C_2)=\sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$ όπου αυτή η απόσταση είναι η γνωστή ***Ευκλείδεια απόσταση*** της γεωμετρίας. Εφόσον οριστεί αυτό το μέγεθος, μπορούμε να ταυτοποιήσουμε τα *οριακά εικονοστοιχεία (edge pixels)* τα οποία γίνονται ως εξής: ξεκινώντας από την αρχή του πίνακα που διατηρεί τα χρωματικά δεδομένα μπορούμε να διαβάζουμε για κάθε εικονοστοιχείο του (m,n), την αναφορά στο χρώμα της χρωματικής παλέτας με την οποία αυτό είναι σχεδιασμένο. Αυτή την πληροφορία, την ανακτούμε και για άλλα δύο εικονοστοιχεία όπου το πρώτο είναι το εικονοστοιχείο που βρίσκεται δεξιά από το προηγούμενο και αντιστοιχεί σε συντεταγμένες (m+1, n) και το δεύτερο είναι το εικονοστοιχείο που βρίσκεται ακριβώς από κάτω του και αντιστοιχεί σε συντεταγμένες (m, n+1). Εάν θεωρήσουμε πως οι τιμές αναφοράς που αντιστοιχούν σε αυτά τα τρία εικονοστοιχεία είναι οι C, C1 και C2, τότε μεταφερόμαστε στην χρωματική παλέτα, και ανακτούμε τις συνιστώσες των τριών χρωμάτων (R,G,B), (R1,G1,B1) και (R2, B2, G2) και υπολογίζουμε τις αποστάσεις d(C, C1) και d(C,C2) σύμφωνα με την εξίσωση που παραθέσαμε προηγουμένως. Εάν μία από αυτές τις τιμές είναι μεγαλύτερη από κάποια απόσταση κατωφλίου (threshold distance) K, η οποία καθορίζεται από τον χρήστη και περνιέται στον αλγόριθμο ως παράμετρος τότε το εικονοστοιχείο (m,n) χαρακτηρίζεται ως *οριακό εικονοστοιχείο (edge pixel).* Ο κώδικας που υλοποιεί την διαδικασία αυτή είναι ο παρακάτω:

```
localEdgePixels = 0;
for (counter=0;counter<rows-1;counter++) {
  for (index=0;index<width-1;index++) {
  color = pixels[counter*width+index];
```

```
  color1 = pixels[counter*width+index+1];
  color2 = pixels[(counter+1)*width+index];
  red = palette[color].rgbRed;
  green = palette[color].rgbGreen;
  blue = palette[color].rgbBlue;
  red1 = palette[color1].rgbRed;
  green1 = palette[color1].rgbGreen;
  blue1 = palette[color1].rgbBlue;
  red2 = palette[color2].rgbRed;
  green2 = palette[color2].rgbGreen;
  blue2 = palette[color2].rgbBlue;
  distance1 = sqrt(pow((red-red1),2)+pow((green-green1),2)+
  pow((blue-blue1),2));
  distance2 = sqrt(pow((red-red2),2)+pow((green-green2),2)+
  pow((blue-blue2),2));
  if ((distance1>=threshold)||(distance2>=threshold))
  localEdgePixels++; }}
```

Και τέλος καλούμε την συνάρτηση MPI_Reduce η οποία υπολογίζει το συνολικό πλήθος των οριακών εικονοστοιχείων που έχουν εντοπιστεί από όλες τις διεργασίες. Η τιμή αυτή εκτυπώνει στην οθόνη μας από την διεργασία με τιμή τάξης R=0. Το πρόγραμμα (η συνάρτηση main) που υλοποιεί όλη αυτή την διαδικασία παρουσιάζεται παρακάτω:

```
int main (int argc, char ** argv) {
int colors, width, height;
char * pixels; int offset;
int rank, size, rows, count, counter, index;
char fileName [64] = "clouds.bmp";
RGBQUAD * palette;
int localEdgePixels=0, totalEdgePixels;
MPI_File fileHandle; MPI_Status status;\

int disp = sizeof (BITMAPFILEHEADER) + sizeof (BITMAPINFOHEADER);

MPI_Offset offset1, offset2;
double threshold = 15.0;
double distance1, distance2;
int color, color1, color2;
int red, green, blue;
int red1, green1, blue1;
int red2, green2, blue2;

MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);

if (rank==0) {
colors = GetBitmapColorNumber (fileName);
width = GetBitmapWidth (fileName);
height = GetBitmapHeight (fileName);
offset = GetBitmapDataOffset (fileName); }

MPI_Bcast (&colors, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&width, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&height, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&offset, 1, MPI_INT, 0, MPI_COMM_WORLD);

MPI_File_open (MPI_COMM_WORLD, fileName, MPI_MODE_RDWR,
MPI_INFO_NULL, &fileHandle);
```

```
MPI_File_seek (fileHandle, disp, MPI_SEEK_SET);
palette=(RGBQUAD *)calloc(colors,sizeof(RGBQUAD));
MPI_File_read (fileHandle, palette, 4*colors, MPI_BYTE, &status);
MPI_Get_elements (&status, MPI_BYTE, &count);
if ((height%size)) {
if (rank==0)
printf ("Bitmap height must be an integer multiple of process
number\n");
MPI_Finalize ();
return (-1); }

rows = height/size;
pixels = (char *) malloc (rows*width*sizeof(char));
if (!pixels) {
printf ("No memory available\n");
MPI_Finalize ();
return (-2); }

MPI_File_seek (fileHandle, offset+rank*rows*width, MPI_SEEK_SET);
MPI_File_get_position (fileHandle, &offset1);
MPI_File_read (fileHandle, pixels, rows*width, MPI_BYTE, &status);
MPI_File_get_position (fileHandle, &offset2);
MPI_Get_elements (&status, MPI_BYTE, &count);
MPI_File_close (&fileHandle);

for (counter=0;counter<rows-1;counter++) {
for (index=0;index<width-1;index++) {
color = pixels[counter*width+index];
color1 = pixels[counter*width+index+1];
color2 = pixels[(counter+1)*width+index];
red = palette[color].rgbRed;
green = palette[color].rgbGreen;
blue = palette[color].rgbBlue;
red1 = palette[color1].rgbRed;
green1 = palette[color1].rgbGreen;
blue1 = palette[color1].rgbBlue;
red2 = palette[color2].rgbRed;
green2 = palette[color2].rgbGreen;
blue2 = palette[color2].rgbBlue;
distance1 = sqrt(pow((red-red1),2)+pow((green-green1),2)+
pow((blue-blue1),2));
distance2 = sqrt(pow((red-red2),2)+pow((green-green2),2)+
pow((blue-blue2),2));
if ((distance1>=threshold)||(distance2>=threshold))
localEdgePixels++; }}
printf ("%d edge pixels out of %d pixels detected from process %d\n",
localEdgePixels, (rows-1)*(width-1), rank);

MPI_Reduce (&localEdgePixels, &totalEdgePixels, 1, MPI_INT, MPI_SUM,
0, MPI_COMM_WORLD);
if (rank==0)
printf ("\n%ld edge pixels found in file %s out of %ld total pixels\n",
totalEdgePixels, fileName, (width-1)*(height-1));

free (pixels);
free (palette);

MPI_Finalize ();
return (0); }
```

Ο συνολικός κώδικας βρίσκεται στο Παράρτημα Α. Η έξοδος της εφαρμογής θα είναι η εξής:

```
2087 edge pixels out of 60705 pixels detected from process 0
3968 edge pixels out of 60705 pixels detected from process 4
3090 edge pixels out of 60705 pixels detected from process 2
3378 edge pixels out of 60705 pixels detected from process 1
3124 edge pixels out of 60705 pixels detected from process 3
15647 edge pixels found in file clouds.bmp out of 306081 total pixels
```

4 Συμπεράσματα

Όπως έχει ήδη αναφερθεί το MPI έχει πολλά χαρακτηριστικά και υποστηρίζονται από συγκεκριμένες συναρτήσεις όπου έχουν πολλές εφαρμογές στον κώδικα. Στην συγκεκριμένη μεταπτυχιακή εργασία αναλύσαμε 2 από αυτά τα χαρακτηριστικά που ήταν οι εικονικές τοπολογίες (καρτεσιανές και τοπολογίες γραφήματος), είδαμε κάποια χαρακτηριστικά τους, αναλύσαμε τις βασικές συναρτήσεις και κάποια παραδείγματα κώδικα των συναρτήσεων αυτών. Επιπλέον είδαμε και τις βασικές συναρτήσεις διαχείρισης αρχείων του MPI και είδαμε κάποια θεωρητικά χαρακτηριστικά. Το MPI έχει πολλά πλεονεκτήματα. Ένα σημαντικό πλεονέκτημα που μας προσφέρει το MPI είναι ο ετερογενής υπολογισμός. Ο ετερογενής υπολογισμός χρησιμοποιεί διαφορετικούς υπολογιστές συνδεδεμένους από ένα δίκτυο ώστε να επιλύσουμε ένα πρόβλημα με παράλληλη τρόπο. Επίσης ένα άλλο πλεονέκτημα που μας προσφέρει το MPI είναι ως προς τα ζητήματα σχεδιασμού. Ο πρώτος λόγος είναι ότι το MPI σχεδιάστηκε να είναι πλούσιο σε λειτουργικότητα. Αυτό αντικατοπτρίζεται στην υποστήριξη του MPI μέσω παραγόμενων τύπους δεδομένων τα οποία μπορούν να φανούν μέσω ενός communicator, μέσω τοπολογιών και μέσω ενός πλήρους εξοπλισμένου συνόλου ρουτινών συλλογικής επικοινωνίας. Ο δεύτερος λόγος είναι ότι το μέγεθος του MPI αντικατοπτρίζει την ποικιλομορφία και την πολυπλοκότητα για υπολογιστές υψηλής απόδοσης. Αξίζει να αναφερθεί πως η ταχύτατη ανάπτυξη και η διάδοση του προτύπου MPI λίγα χρόνια μετά την πρώτη του εμφάνιση, μας οδηγεί στο συμπέρασμα πως είναι ένα προγραμματιστικό μοντέλο που γνώρισε μεγάλη απήχηση στον χώρο της επιστημονικής κοινότητας και κάλυψε πάρα πολύ τις ανάγκες που έχουν οδηγήσει στην υλοποίηση του κάτι που ελπίζουμε να συνεχίσει να υφίσταται και στο μέλλον.

Βιβλιογραφία

[1] Αθανάσιος Ι. Μάργαρης, MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα, 2014

[2] Εικονικές Τοπολογίες: (url:www.it.uom.gr/teaching/wilkinson/mpi4.pdf)

[3] MPI: Message Passing Interface (url:www.cslab.ntua.gr/courses/pps/files/fall2019/MPIpresentation-Fall2019.pdf)

[4] Διεπαφή μεταβίβασης μηνυμάτων (url: http://pdplab.it.uom.gr/teaching/llnl-gr/Message%20Passing%20Interface%20(MPI).htm)

[5] Παράλληλοι υπολογισμοί Τμήμα Πληροφορικής και Τηλεπικοινωνιών ΕΚΠΑ 30 Μαρτίου 2009 (url:https://eclass.uoa.gr/document/file.php/D183/Par_Yp_(RB_SOR).pdf

[6] Εισαγωγή στον παράλληλο προγραμματισμό: (url:https://hpc.it.auth.gr/parallel-intro/#mpi-2.)

[7] Process Topologies: (url:https://www.codingame.com/playgrounds/47058/have-fun-with-mpi-in-c/mpi-process-topologies.)

[8] Virtual Topologies: (url:https://hpc-tutorials.llnl.gov/mpi/virtual_topologies.)

[9] William Gropp Process Topology and MPI (url:https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture28.pdf)

[10] Virtual Topologies: (url:https://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/node129.htm)

[11] Jan Thorbecke Programming with MPI: (url:https://janth.home.xs4all.nl/MPIcourse/PDF/07_MPI_Topology.pdf)

[12] Virtual Topology: (url:https://www.bu.edu/tech/support/research/training-consulting/online-tutorials/mpi/virtual-topology)

[13] Τοπολογίες: (url:www.rc.udf.edu/tutorials/classes/tutorial/mpi/chapter10.html)

[14] Edwin Laure Advanced MPI (url:https://indico.fysik.su.se/event/4384/contributions/5720/attachments/2881/3222/MPI-6.pdf.)

[15] Cartesian and graph topologies: (url:www.new-npac.org/projects/cdroms/cewes-1999-06-vol1/cps615course/mpi-examples/note2.html)

[16] Opening a file: (url:https://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-2.0/node175.htm)

[17] William Gropp Introduction to MPI I/O (url:https://wgropp.cs.illinois.edu./courses/cs598-s16/lectures/lecture32.pdf)

[18] Message Passing Interface (url:https://en.wikipedia.org/wiki/Message_Passing_interface)

[19] Parallel Programming with MPI and OpenMP (url:https://www.scss.tcd.ie/David.Gregg/cs3014/notes/lecture-25-MPI-OpenMP.pdf)

[20] Lucas Amudruz High Performance Computing for Science and Engineering (url:https://www.cse-lab.ethnz.ch/wp-content/uploads/2019/11/hpcse1-19_Lecture_10_IO_hybrid_MPI_3.pdf)

[21] MPI-2: Extensions to the Message Passing Interface (url: https://www.mpi-forum.org/docs/mpi-2.0/mpi-report.pdf)

[22] Parallel IO: basics and MPI2-IO (url: https://hpc-forge.cineca.it/files/CoursesDev/public/2017/Parallel_IO_and_management_of_large_scientific_data/Roma/MPI-IO_2017.pdf)

[23] Programming with Sun MPI I/O (url: https://docs.oracle.com/cd/E19061-01/hpc.cluster6/819-4133-10/io.html)

[24] Rolf Rabenseifner Parallel File I/O with MPI-2 (url: https://fs.hlrs.de/projects/par/par_prog_ws/pdf/mpi_io_1.pdf)

[25] MPI_File_open (url: https:www.mpich.org/static/docs/v3.3/www3/MPI_File_open.html)

[26] MPI_File_close (url: https://www.mpich.org/static/docs/v3.3/www3/MPI_File_close.html)

[27] Deleting a file (url: https://www.mpi-forum.org/docs/mpi-2.2/mpi22-report/node267.htm)

[28] Closing a File (url: https://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-2.0/node176.htm)

[29] MPI: The complete reference (url: http://cecs.wright.edu/~schung/ceg820/mpi-book.pdf)

[30] Marc Snir Steve oto, Steven Huss Landerman, David Walker Jack Dongarra MPI the complete reference published 28 September 1998

[31] MPI: The complete reference-Computing Systems Laboratory (url: www.cslab.ntua.gr/courses/common/mpi-book.pdf)

[32] MPI Process Topology (url: https://canvas.kth.se/courses/24375/pages/lecture-mpi-process-topology)

[33] Writing Message-Passing Parallel Programs with MPI (url: https://www.uni-muenster.de/imperia/md/content/physik_ct/pdf_intern/mpi_course.pdf)

[34] MPI_File_preallocate (url: https://www.mpich.org/static/docs/v3.2/www3/MPI_File_preallocate.html)

[35] MPI: A Message-Passing Interface Standard Version 3.0 (url: https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report-book.pdf)

[36] MPI_File_set_size (url: https://www.mpich.org/static/docs/v3.2/www3/MPI_File_set_size.html)

[37] MPI Functions (url: https://learn.microsoft.com/en-us/message-passing-interface/mpi-file-set-info-function)

[38] MPICH-High-Performance Portable MPI (url: https://www.mpich.org)

[39] MPICH (url: https://en.wikipedia.org/wiki/MPICH)

[40] LAM-MPI (url: https://en.wikipedia.org/wiki/LAM/MPI)

Παράρτημα Α-Κώδικας Προγραμμάτων

Κεφάλαιο 7: Παράλληλη ανάγνωση στοιχείων εικόνας BMP (Αθανάσιος Ι. Μάργαρης MPI Θεωρία &Εφαρμογές Εκδόσεις Τζιόλα 2014)

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef unsigned int UINT;
typedef unsigned long DWORD;
typedef unsigned short WORD;
typedef unsigned long LONG;
typedef unsigned char BYTE;


typedef struct tagBITMAPFILEHEADER {

UINT bfType;
DWORD bfSize;
UINT bfReserved1;
UINT bfReserved2;
DWORD bfOffBits;

} BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER {

DWORD   biSize;
LONG            biWidth;
LONG            biHeight;
WORD            biPlanes;
WORD            biBitCount;
DWORD   biCompression;
DWORD   biSizeImage;
LONG            biXPelsPerMeter;
LONG            biYPelsPerMeter;
DWORD   biClrUsed;
DWORD   biClrImportant;

} BITMAPINFOHEADER;

typedef struct tagRGBQUAD {
BYTE            rgbBlue;
BYTE            rgbGreen;
BYTE            rgbRed;
BYTE            rgbReserved;
} RGBQUAD;

int GetBitmapDataOffset (char * fileName) {
```

```
FILE * fp;
BITMAPFILEHEADER bmpHeader={0};
fp = fopen (fileName, "rb");
if (!fp) {
printf ("Bitmap File could not be Opened\n");
return (-1); }
rewind (fp);
fread (&bmpHeader, sizeof(bmpHeader), 1, fp);
fclose (fp);
return (bmpHeader.bfOffBits); }

int GetBitmapColorNumber (char * fileName) {
FILE * fp; int colors;
BITMAPFILEHEADER bmpHeader={0};
BITMAPINFOHEADER bmpInfo={0};
fp = fopen (fileName, "rb");
if (!fp) {
printf ("Bitmap File could not be Opened\n");
return (-1); }
rewind (fp);
fread (&bmpHeader, sizeof(bmpHeader), 1, fp);
fread (&bmpInfo,sizeof(bmpInfo),1,fp);
fclose (fp);
if (bmpInfo.biBitCount<=8)
colors = (int)pow(2,bmpInfo.biBitCount);
else colors = 0;
return (colors); }

int GetBitmapWidth (char * fileName) {
FILE * fp;
BITMAPFILEHEADER bmpHeader={0};
BITMAPINFOHEADER bmpInfo={0};
fp = fopen (fileName, "rb");
if (!fp) {
printf ("Bitmap File could not be Opened\n");
return (-1); }
rewind (fp);
fread (&bmpHeader, sizeof(bmpHeader), 1, fp);
fread (&bmpInfo,sizeof(bmpInfo),1,fp);
fclose (fp);
return (bmpInfo.biWidth); }

int GetBitmapHeight (char * fileName) {
FILE * fp;
BITMAPFILEHEADER bmpHeader={0};
BITMAPINFOHEADER bmpInfo={0};
fp = fopen (fileName, "rb");
if (!fp) {
printf ("Bitmap File could not be Opened\n");
return (-1); }
rewind (fp);
fread (&bmpHeader, sizeof(bmpHeader), 1, fp);
fread (&bmpInfo,sizeof(bmpInfo),1,fp);
fclose (fp);
return (bmpInfo.biHeight); }


RGBQUAD * GetBitmapColorPalette (char * fileName) {
FILE * fp; int colorNumber;
RGBQUAD * rgbTable = NULL;
BITMAPFILEHEADER bmpHeader={0};
```

```
BITMAPINFOHEADER bmpInfo={0};
fp = fopen (fileName, "rb");
if (!fp) {
printf ("Bitmap File could not be Opened\n");
return (NULL); }
rewind (fp);
fread (&bmpHeader, sizeof(bmpHeader), 1, fp);
fread (&bmpInfo,sizeof(bmpInfo),1,fp);
if (bmpInfo.biBitCount<=8)
colorNumber = (int) pow (2, bmpInfo.biBitCount);
else colorNumber = 0;
if (colorNumber) {
rgbTable=(RGBQUAD *)calloc(colorNumber,sizeof(RGBQUAD));
if (!rgbTable) return (NULL);
fread (rgbTable, sizeof(RGBQUAD), colorNumber, fp); }
fclose (fp);
if (!colorNumber) return (NULL);
return (rgbTable);}

int main (int argc, char ** argv) {
int colors, width, height;
char * pixels; int offset;
int rank, size, rows, count, counter, index;
char fileName [64] = "clouds.bmp";
RGBQUAD * palette;
int localEdgePixels=0, totalEdgePixels;
MPI_File fileHandle; MPI_Status status;\

int disp = sizeof (BITMAPFILEHEADER) +
                   sizeof (BITMAPINFOHEADER);

MPI_Offset offset1, offset2;
double threshold = 15.0;
double distance1, distance2;
int color, color1, color2;
int red, green, blue;
int red1, green1, blue1;
int red2, green2, blue2;

MPI_Init (&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);

if (rank==0) {
colors = GetBitmapColorNumber (fileName);
        width = GetBitmapWidth (fileName);
        height = GetBitmapHeight (fileName);
        offset = GetBitmapDataOffset (fileName); }

MPI_Bcast (&colors, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&width, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&height, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast (&offset, 1, MPI_INT, 0, MPI_COMM_WORLD);

MPI_File_open (MPI_COMM_WORLD, fileName, MPI_MODE_RDWR,
    MPI_INFO_NULL, &fileHandle);

MPI_File_seek (fileHandle, disp, MPI_SEEK_SET);
palette=(RGBQUAD *)calloc(colors,sizeof(RGBQUAD));
MPI_File_read (fileHandle, palette, 4*colors, MPI_BYTE, &status);
MPI_Get_elements (&status, MPI_BYTE, &count);
```

```
if ((height%size)) {
        if (rank==0)
      printf ("Bitmap height must be an integer multiple of process number\n");
              MPI_Finalize ();
              return (-1); }

rows = height/size;
pixels = (char *) malloc (rows*width*sizeof(char));
if (!pixels) {
        printf ("No memory available\n");
        MPI_Finalize ();
        return (-2); }

MPI_File_seek (fileHandle, offset+rank*rows*width, MPI_SEEK_SET);
MPI_File_get_position (fileHandle, &offset1);
MPI_File_read (fileHandle, pixels, rows*width, MPI_BYTE, &status);
MPI_File_get_position (fileHandle, &offset2);
MPI_Get_elements (&status, MPI_BYTE, &count);
MPI_File_close (&fileHandle);

localEdgePixels = 0;
for (counter=0;counter<rows-1;counter++) {
        for (index=0;index<width-1;index++) {
                color = pixels[counter*width+index];
                color1 = pixels[counter*width+index+1];
                color2 = pixels[(counter+1)*width+index];
                red = palette[color].rgbRed;
                green = palette[color].rgbGreen;
                blue = palette[color].rgbBlue;
                red1 = palette[color1].rgbRed;
                green1 = palette[color1].rgbGreen;
        blue1 = palette[color1].rgbBlue;
                red2 = palette[color2].rgbRed;
                green2 = palette[color2].rgbGreen;
                blue2 = palette[color2].rgbBlue;
                distance1 = sqrt(pow((red-red1),2)+pow((green-green1),2)+
                        pow((blue-blue1),2));
                distance2 = sqrt(pow((red-red2),2)+pow((green-green2),2)+
                        pow((blue-blue2),2));
                if ((distance1>=threshold)||(distance2>=threshold))
                        localEdgePixels++; }}

printf ("%d edge pixels out of %d pixels detected from process %d\n",
              localEdgePixels, (rows-1)*(width-1), rank);

MPI_Reduce (&localEdgePixels, &totalEdgePixels, 1, MPI_INT, MPI_SUM,
                           0, MPI_COMM_WORLD);
        if (rank==0)
             printf ("\n%d edge pixels found in file %s out of %d total pixels\n",
             totalEdgePixels, fileName, (width-1)*(height-1));

        free (pixels);
        free (palette);

MPI_Finalize ();
        return (0); }
```

Παράρτημα Β-Έξοδοι προγραμμά-των

Κεφάλαιο 2:

Παράδειγμα συνάρτησης MPI_Card_shift() (1)

```
Process 22: Source is 14 Destination is 30 for dimension 0
Process 22: Source is 20 Destination is 16 for dimension 1
Process 22: Source is 23 Destination is 23 for dimension 2
Process 30: Source is 22 Destination is 38 for dimension 0
Process 30: Source is 28 Destination is 24 for dimension 1
Process 30: Source is 31 Destination is 31 for dimension 2
Process 38: Source is 30 Destination is 06 for dimension 0
Process 38: Source is 36 Destination is 32 for dimension 1
Process 38: Source is 39 Destination is 39 for dimension 2
Process 17: Source is 09 Destination is 25 for dimension 0
Process 17: Source is 23 Destination is 19 for dimension 1
Process 17: Source is 16 Destination is 16 for dimension 2
Process 33: Source is 25 Destination is 01 for dimension 0
Process 33: Source is 39 Destination is 35 for dimension 1
Process 33: Source is 32 Destination is 32 for dimension 2
Process 25: Source is 17 Destination is 33 for dimension 0
Process 25: Source is 31 Destination is 27 for dimension 1
Process 25: Source is 24 Destination is 24 for dimension 2
Process 18: Source is 10 Destination is 26 for dimension 0
Process 18: Source is 16 Destination is 20 for dimension 1
Process 18: Source is 19 Destination is 19 for dimension 2
Process 34: Source is 26 Destination is 02 for dimension 0
Process 34: Source is 32 Destination is 36 for dimension 1
Process 34: Source is 35 Destination is 35 for dimension 2
Process 26: Source is 18 Destination is 34 for dimension 0
Process 26: Source is 24 Destination is 28 for dimension 1
Process 26: Source is 27 Destination is 27 for dimension 2
Process 03: Source is 35 Destination is 11 for dimension 0
Process 03: Source is 01 Destination is 05 for dimension 1
Process 03: Source is 02 Destination is 02 for dimension 2
Process 01: Source is 33 Destination is 09 for dimension 0
Process 01: Source is 07 Destination is 03 for dimension 1
Process 01: Source is 00 Destination is 00 for dimension 2
Process 00: Source is 32 Destination is 08 for dimension 0
Process 00: Source is 06 Destination is 02 for dimension 1
Process 00: Source is 01 Destination is 01 for dimension 2
Process 09: Source is 01 Destination is 17 for dimension 0
Process 09: Source is 15 Destination is 11 for dimension 1
Process 09: Source is 08 Destination is 08 for dimension 2
Process 20: Source is 12 Destination is 28 for dimension 0
Process 20: Source is 18 Destination is 22 for dimension 1
Process 20: Source is 21 Destination is 21 for dimension 2
Process 28: Source is 20 Destination is 36 for dimension 0
Process 28: Source is 26 Destination is 30 for dimension 1
Process 28: Source is 29 Destination is 29 for dimension 2
Process 08: Source is 00 Destination is 16 for dimension 0
```

```
Process 08: Source is 14 Destination is 10 for dimension 1
Process 08: Source is 09 Destination is 09 for dimension 2
Process 36: Source is 28 Destination is 04 for dimension 0
Process 36: Source is 34 Destination is 38 for dimension 1
Process 36: Source is 37 Destination is 37 for dimension 2
Process 31: Source is 23 Destination is 39 for dimension 0
Process 31: Source is 29 Destination is 25 for dimension 1
Process 31: Source is 30 Destination is 30 for dimension 2
Process 13: Source is 05 Destination is 21 for dimension 0
Process 13: Source is 11 Destination is 15 for dimension 1
Process 13: Source is 12 Destination is 12 for dimension 2
Process 12: Source is 04 Destination is 20 for dimension 0
Process 12: Source is 10 Destination is 14 for dimension 1
Process 12: Source is 13 Destination is 13 for dimension 2
Process 11: Source is 03 Destination is 19 for dimension 0
Process 11: Source is 09 Destination is 13 for dimension 1
Process 11: Source is 10 Destination is 10 for dimension 2
Process 10: Source is 02 Destination is 18 for dimension 0
Process 10: Source is 08 Destination is 12 for dimension 1
Process 10: Source is 11 Destination is 11 for dimension 2
Process 16: Source is 08 Destination is 24 for dimension 0
Process 16: Source is 22 Destination is 18 for dimension 1
Process 16: Source is 17 Destination is 17 for dimension 2
Process 32: Source is 24 Destination is 00 for dimension 0
Process 32: Source is 38 Destination is 34 for dimension 1
Process 32: Source is 33 Destination is 33 for dimension 2
Process 24: Source is 16 Destination is 32 for dimension 0
Process 24: Source is 30 Destination is 26 for dimension 1
Process 24: Source is 25 Destination is 25 for dimension 2
Process 02: Source is 34 Destination is 10 for dimension 0
Process 02: Source is 00 Destination is 04 for dimension 1
Process 02: Source is 03 Destination is 03 for dimension 2
Process 05: Source is 37 Destination is 13 for dimension 0
Process 05: Source is 03 Destination is 07 for dimension 1
Process 05: Source is 04 Destination is 04 for dimension 2
Process 15: Source is 07 Destination is 23 for dimension 0
Process 15: Source is 13 Destination is 09 for dimension 1
Process 15: Source is 14 Destination is 14 for dimension 2
Process 14: Source is 06 Destination is 22 for dimension 0
Process 14: Source is 12 Destination is 08 for dimension 1
Process 14: Source is 15 Destination is 15 for dimension 2
Process 07: Source is 39 Destination is 15 for dimension 0
Process 07: Source is 05 Destination is 01 for dimension 1
Process 07: Source is 06 Destination is 06 for dimension 2
Process 06: Source is 38 Destination is 14 for dimension 0
Process 06: Source is 04 Destination is 00 for dimension 1
Process 06: Source is 07 Destination is 07 for dimension 2
Process 23: Source is 15 Destination is 31 for dimension 0
Process 23: Source is 21 Destination is 17 for dimension 1
Process 23: Source is 22 Destination is 22 for dimension 2
Process 39: Source is 31 Destination is 07 for dimension 0
Process 39: Source is 37 Destination is 33 for dimension 1
Process 39: Source is 38 Destination is 38 for dimension 2
Process 21: Source is 13 Destination is 29 for dimension 0
Process 21: Source is 19 Destination is 23 for dimension 1
Process 21: Source is 20 Destination is 20 for dimension 2
Process 37: Source is 29 Destination is 05 for dimension 0
Process 37: Source is 35 Destination is 39 for dimension 1
Process 37: Source is 36 Destination is 36 for dimension 2
Process 04: Source is 36 Destination is 12 for dimension 0
Process 04: Source is 02 Destination is 06 for dimension 1
```

```
Process 04: Source is 05 Destination is 05 for dimension 2
Process 19: Source is 11 Destination is 27 for dimension 0
Process 19: Source is 17 Destination is 21 for dimension 1
Process 19: Source is 18 Destination is 18 for dimension 2
Process 35: Source is 27 Destination is 03 for dimension 0
Process 35: Source is 33 Destination is 37 for dimension 1
Process 35: Source is 34 Destination is 34 for dimension 2
Process 27: Source is 19 Destination is 35 for dimension 0
Process 27: Source is 25 Destination is 29 for dimension 1
Process 27: Source is 26 Destination is 26 for dimension 2
Process 29: Source is 21 Destination is 37 for dimension 0
Process 29: Source is 27 Destination is 31 for dimension 1
Process 29: Source is 28 Destination is 28 for dimension 2
```

Παράδειγμα MPI_Card_shift() (2)

```
Process 29: Source is 21 Destination is 37 for dimension 0
Process 29: Source is 27 Destination is 31 for dimension 1
Process 29: Source is 28 Destination is -1 for dimension 2
Process 20: Source is 12 Destination is 28 for dimension 0
Process 20: Source is 18 Destination is 22 for dimension 1
Process 20: Source is -1 Destination is 21 for dimension 2
Process 27: Source is 19 Destination is 35 for dimension 0
Process 27: Source is 25 Destination is 29 for dimension 1
Process 27: Source is 26 Destination is -1 for dimension 2
Process 35: Source is 27 Destination is -1 for dimension 0
Process 35: Source is 33 Destination is 37 for dimension 1
Process 35: Source is 34 Destination is -1 for dimension 2
Process 24: Source is 16 Destination is 32 for dimension 0
Process 24: Source is 30 Destination is 26 for dimension 1
Process 24: Source is -1 Destination is 25 for dimension 2
Process 32: Source is 24 Destination is -1 for dimension 0
Process 32: Source is 38 Destination is 34 for dimension 1
Process 32: Source is -1 Destination is 33 for dimension 2
Process 28: Source is 20 Destination is 36 for dimension 0
Process 28: Source is 26 Destination is 30 for dimension 1
Process 28: Source is -1 Destination is 29 for dimension 2
Process 22: Source is 14 Destination is 30 for dimension 0
Process 22: Source is 20 Destination is 16 for dimension 1
Process 22: Source is -1 Destination is 23 for dimension 2
Process 13: Source is 05 Destination is 21 for dimension 0
Process 13: Source is 11 Destination is 15 for dimension 1
Process 13: Source is 12 Destination is -1 for dimension 2
Process 12: Source is 04 Destination is 20 for dimension 0
Process 12: Source is 10 Destination is 14 for dimension 1
Process 12: Source is -1 Destination is 13 for dimension 2
Process 09: Source is 01 Destination is 17 for dimension 0
Process 09: Source is 15 Destination is 11 for dimension 1
Process 09: Source is 08 Destination is -1 for dimension 2
Process 08: Source is 00 Destination is 16 for dimension 0
Process 08: Source is 14 Destination is 10 for dimension 1
Process 08: Source is -1 Destination is 09 for dimension 2
Process 01: Source is -1 Destination is 09 for dimension 0
Process 01: Source is 07 Destination is 03 for dimension 1
Process 01: Source is 00 Destination is -1 for dimension 2
Process 39: Source is 31 Destination is -1 for dimension 0
Process 39: Source is 37 Destination is 33 for dimension 1
Process 39: Source is 38 Destination is -1 for dimension 2
Process 11: Source is 03 Destination is 19 for dimension 0
```

```
Process 11: Source is 09 Destination is 13 for dimension 1
Process 11: Source is 10 Destination is -1 for dimension 2
Process 10: Source is 02 Destination is 18 for dimension 0
Process 10: Source is 08 Destination is 12 for dimension 1
Process 10: Source is -1 Destination is 11 for dimension 2
Process 07: Source is -1 Destination is 15 for dimension 0
Process 07: Source is 05 Destination is 01 for dimension 1
Process 07: Source is 06 Destination is -1 for dimension 2
Process 06: Source is -1 Destination is 14 for dimension 0
Process 06: Source is 04 Destination is 00 for dimension 1
Process 06: Source is -1 Destination is 07 for dimension 2
Process 15: Source is 07 Destination is 23 for dimension 0
2
Process 15: Source is 13 Destination is 09 for dimension 1
Process 15: Source is 14 Destination is -1 for dimension 2
Process 31: Source is 23 Destination is 39 for dimension 0
Process 31: Source is 29 Destination is 25 for dimension 1
Process 31: Source is 30 Destination is -1 for dimension 2
Process 14: Source is 06 Destination is 22 for dimension 0
Process 14: Source is 12 Destination is 08 for dimension 1
Process 14: Source is -1 Destination is 15 for dimension 2
Process 16: Source is 08 Destination is 24 for dimension 0
Process 16: Source is 22 Destination is 18 for dimension 1
Process 16: Source is -1 Destination is 17 for dimension 2
Process 17: Source is 09 Destination is 25 for dimension 0
Process 17: Source is 23 Destination is 19 for dimension 1
Process 17: Source is 16 Destination is -1 for dimension 2
Process 33: Source is 25 Destination is -1 for dimension 0
Process 33: Source is 39 Destination is 35 for dimension 1
Process 33: Source is 32 Destination is -1 for dimension 2
Process 19: Source is 11 Destination is 27 for dimension 0
Process 19: Source is 17 Destination is 21 for dimension 1
Process 19: Source is 18 Destination is -1 for dimension 2
Process 21: Source is 13 Destination is 29 for dimension 0
Process 21: Source is 19 Destination is 23 for dimension 1
Process 21: Source is 20 Destination is -1 for dimension 2
Process 37: Source is 29 Destination is -1 for dimension 0
Process 37: Source is 35 Destination is 39 for dimension 1
Process 37: Source is 36 Destination is -1 for dimension 2
Process 26: Source is 18 Destination is 34 for dimension 0
Process 26: Source is 24 Destination is 28 for dimension 1
Process 26: Source is -1 Destination is 27 for dimension 2
Process 18: Source is 10 Destination is 26 for dimension 0
Process 18: Source is 16 Destination is 20 for dimension 1
Process 18: Source is -1 Destination is 19 for dimension 2
Process 34: Source is 26 Destination is -1 for dimension 0
Process 34: Source is 32 Destination is 36 for dimension 1
Process 34: Source is -1 Destination is 35 for dimension 2
Process 30: Source is 22 Destination is 38 for dimension 0
Process 30: Source is 28 Destination is 24 for dimension 1
Process 30: Source is -1 Destination is 31 for dimension 2
Process 36: Source is 28 Destination is -1 for dimension 0
Process 36: Source is 34 Destination is 38 for dimension 1
Process 36: Source is -1 Destination is 37 for dimension 2
Process 05: Source is -1 Destination is 13 for dimension 0
Process 05: Source is 03 Destination is 07 for dimension 1
Process 05: Source is 04 Destination is -1 for dimension 2
Process 23: Source is 15 Destination is 31 for dimension 0
Process 23: Source is 21 Destination is 17 for dimension 1
Process 23: Source is 22 Destination is -1 for dimension 2
Process 38: Source is 30 Destination is -1 for dimension 0
```

```
Process 38: Source is 36 Destination is 32 for dimension 1
Process 38: Source is -1 Destination is 39 for dimension 2
Process 25: Source is 17 Destination is 33 for dimension 0
Process 25: Source is 31 Destination is 27 for dimension 1
Process 25: Source is 24 Destination is -1 for dimension 2
Process 00: Source is -1 Destination is 08 for dimension 0
Process 00: Source is 06 Destination is 02 for dimension 1
Process 00: Source is -1 Destination is 01 for dimension 2
Process 03: Source is -1 Destination is 11 for dimension 0
Process 03: Source is 01 Destination is 05 for dimension 1
Process 03: Source is 02 Destination is -1 for dimension 2
Process 02: Source is -1 Destination is 10 for dimension 0
Process 02: Source is 00 Destination is 04 for dimension 1
Process 02: Source is -1 Destination is 03 for dimension 2
Process 04: Source is -1 Destination is 12 for dimension 0
Process 04: Source is 02 Destination is 06 for dimension 1
Process 04: Source is -1 Destination is 05 for dimension 2
```

Παράρτημα Γ-Εγκατάσταση MPI

Εάν θέλουμε να εγκαταστήσουμε το MPI στο λειτουργικό σύστημα Ubuntu τότε θα πρέπει να ακολουθήσουμε τα εξής βήματα:

1. Κάνουμε **download** τα απαραίτητα αρχεία **(mpich.tar.gz)** για την εγκατάσταση του MPI από το site http://www.mpich.org.
2. *Δημιουργούμε το αρχείο mpich.tar* με την εντολή **gzip -d mpich.tar.gz.**
3. Αποσυμπιέζουμε το *mpich.tar* με την εντολή **tar -xvf mpich.tar.**
4. Εισάγουμε τον φάκελο που δημιουργήθηκε σε ένα *directory (φάκελος)* με ***λατινικούς χαρακτήρες*.**
5. Μπαίνουμε στον φάκελο που δημιουργήσαμε.
6. *Διαμορφώνουμε τις παραμέτρους* που επιθυμούμε π.χ ορίζουμε τον φάκελο εγκατάστασης και ότι άλλο θέλουμε καθώς και την εγκατάσταση του mpe με την εντολή : **./configure -prefix=/home/vaggelis/mpi -with-mpe.**
7. Κάνουμε *build* με την εντολή **make.**
8. Κάνουμε *εγκατάσταση* με την εντολή **make install**.