Πανεπιστήμιο Ιωαννίνων

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Προπτυχιακό Μάθημα:

«Παράλληλα Συστήματα και Προγραμματισμός»

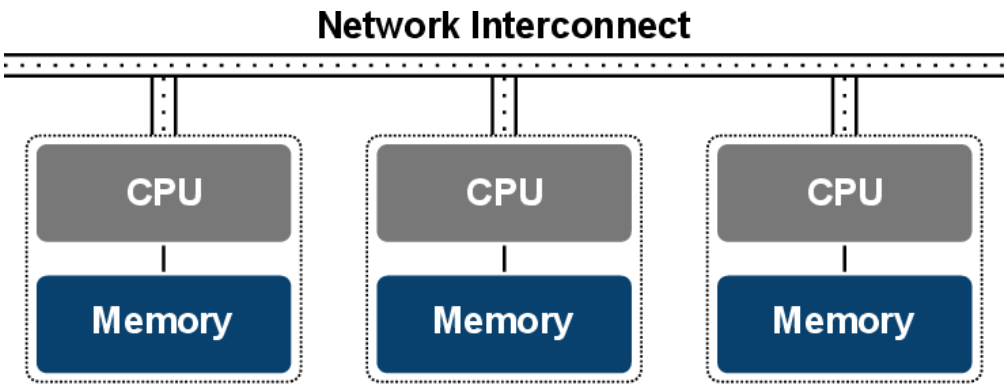
Τρίτο Σετ Προγραμματιστικών Ασκήσεων

Όνομα Φοιτητή – Α.Μ.:

Γεώργιος Κρομμύδας – 3260

E-mail Φοιτητή:

cs03260@uoi.gr



ΙΩΑΝΝΙΝΑ,

2022

Πίνακας περιεχομένων

1.Εισαγωγή:

Το 3ο σετ ασκήσεων αφορά στον παράλληλο προγραμματισμό με το μοντέλο κατανεμημένου χώρου διευθύνσεων μέσω του προτύπου ***OpenMPI***. Ζητείται η παραλληλοποίηση δύο εφαρμογών. Η μία είναι η θόλωση εικόνων με το μοντέλο ***MPI*** και η άλλη αφορά τον πολλαπλασιασμό πινάκων με χρήση του υβριδικού μοντέλου ***MPI/OpenMP*** με χρήση των συλλογικών επικοινωνιών.

Όλες οι μετρήσεις έγιναν στο παρακάτω σύστημα:

Όνομα Υπολογιστή	opti7020ws01
Επεξεργαστής	Intel Core i5-4590 3.3Ghz
Πλήθος Πυρήνων	4
Μεταγλωττιστής	gcc/mpicc v.7.5.0

Πίνακας 1: Πληροφορίες Συστήματος

2.Άσκηση – 1

2.1 Το Πρόβλημα

Στην άσκηση αυτή ζητείτε να παραλληλοποιηθεί ο αλγόριθμος θόλωσης εικόνων με το μοντέλο του ***MPI*** σε συνδυασμό με διαφορετικές εικονικές μηχανές (hostfiles) που θα περιέχουν διαφορετικό πλήθος πυρήνων. Επίσης, κάθε διεργασία που κάνει εκτελεί το δικό του μπλοκ γραμμών της εικόνας, θα τις επιστρέφει στον κεντρικό κόμβο ώστε να εμφανίζει το τελικό αποτέλεσμα.

2.2 Μέθοδοι Παραλληλοποίησης

Για την παραλληλοποίηση, χρησιμοποιήθηκε το σειριακό πρόγραμμα που υπήρχε στην ιστοσελίδα του μαθήματος (gaussian-blur.c). Σε αυτό το πρόγραμμα τροποποιήθηκε η σειριακή εκδοχή του αλγορίθμου gaussian_blur_serial(). Συγκεκριμένα, δημιουργήθηκε η συνάρτηση gaussian_blur_mpi(), η οποία εκτός από τα κλασικά τρία ορίσματα, παίρνει και ως ορίσματα τον αριθμό των διεργασιών και την ταυτότητα του κάθε κόμβου. Πριν ξεκινήσει η ανταλλαγή των μηνυμάτων μεταξύ

των κόμβων, γίνεται broadcast σε όλους του κόμβους το μέγεθος της ακτίνας θόλωσης με την εντολή

MPI_Bcast(&radius, 1, MPI_INT, 0, MPI_COMM_WORLD);

στην συνάρτηση ***main()***. Επίσης, ορίζονται και οι διεργασίες με την συνάρτηση ***MPI_Comm_Rank()*** και το πλήθος των διεργασιών με την ***MPI_Comm_Size()***.

Για να γίνει η ανταλλαγή μηνυμάτων μεταξύ των διαθέσιμων κόμβων, θα πρέπει πρώτα να οριστεί η μεταβλητή ***WORK*** η οποία είναι το μέγεθος της εικόνας ως προς το πλήθος των διεργασιών. Το επόμενο βήμα είναι να μεταδώσουμε τα συνεχόμενα μπλοκ της εικόνας στις διεργασίες. Στη συνέχεια ορίζονται οι buffers των χρωμάτων με χρήση της ***malloc***, με μέγεθος ***height*width*sizeof(unsigned char)***, καθώς θα περιέχουν όλη την πληροφορία των διεργασιών των άλλων κόμβων που θα πρέπει να σταλθούν στην διεργασία 0. Αυτό που γίνεται στην ουσία είναι ο τεμαχισμός της εικόνας σε ***ID*WORK*** συνεχόμενα μπλοκ.

Αρχικά, η διεργασία 0 επεξεργάζεται το κομμάτι από το 0 έως ***WORK*** της εικόνας, με την παραλληλοποίηση να γίνεται πάντα στον εξωτερικό βρόχο. Όταν τελειώσει την επεξεργασία η διεργασία 0, τότε λαμβάνει μηνύματα από όλες τις υπόλοιπες διεργασίες και για τα τρία κανάλια χρώματος. Στη συνέχεια τα αποτελέσματα των τριών buffer τα αποθηκεύει στην εικόνα εξόδου ***imgout*** στις αντίστοιχες θέσεις ανάλογα με την διεργασία και στην αντίστοιχη θέση. Δηλαδή θα έχουμε τις εξής ακολουθίες εντολών:

```
1. for(k = 1; k < nproc; k++)
2.   MPI_Recv(red_buffer, WORK*width, MPI_UNSIGNED_CHAR, k, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
3.  MPI_Recv(green_buffer, WORK*width, MPI_UNSIGNED_CHAR, k, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
4.  MPI_Recv(blue_buffer, WORK*width, MPI_UNSIGNED_CHAR, k, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
5.  for(i = k*WORK; i < (k+1)*WORK; i++)
6.    for(j = 0; j < width; j++){
7.          imgout->red[i*width+j] = red_buffer[i*width+j];
8.          imgout->green[i*width+j]=green_buffer[i*width+j];
9.          imgout->blue[i*width+j] = blue_buffer[i*width+j];
```

Κώδικας 1: Εντολές της διεργασίας 0

Το επόμενο βήμα είναι να δούμε τι κάνουν οι υπόλοιπες διεργασίες. Πρώτα από όλα η κάθε διεργασία εκτελεί τον αλγόριθμο της θόλωσης με διαφορετικό πλήθος block. Παραλληλοποιήται το εξωτερικό ***for loop*** το οποίο θα γίνει ***for(i = ID*WORK; i < (ID+1)*WORK; i++)*** και στην συνέχεια η κάθε διεργασία θα αποθηκεύει τα αποτελέσματα στους τρεις buffers. Τέλος, η κάθε διεργασία τα στέλνει στην διεργασία 0 μέσω της συνάρτησης ***MPI_Send()***. Πιο αναλυτικά φαίνεται παρακάτω:

```
1.   for(i = ID*WORK; i < (ID+1)*WORK; i++)
2.     for(j = 0; j < width; j++)
3.       red_buffer[i*width+j] = imgout->red[i*width+j];
4.       green_buffer[i*width+j] = imgout->green[i*width+j];
5.       blue_buffer[i*width+j] = imgout->blue[i*width+j];
6.    MPI_Send(red_buffer,   WORK*width,   MPI_UNSIGNED_CHAR,   0,   0,
MPI_COMM_WORLD);
7.    MPI_Send(green_buffer,   WORK*width,   MPI_UNSIGNED_CHAR,   0,   0,
MPI_COMM_WORLD);
8.    MPI_Send(blue_buffer,   WORK*width,   MPI_UNSIGNED_CHAR,   0,   0,
MPI_COMM_WORLD);
```

Κώδικας 2: Εντολές των Υπόλοιπων διεργασιών

Κατά το πέρας της επεξεργασίας από όλους τους κόμβους ανασυντίθεται η τελική θολωμένη εικόνα τελειώνουν και οι επικοινωνίες μεταξύ των κόμβων.

2.3 Πειραματικά Αποτελέσματα – Μετρήσεις

Καθώς η λειτουργία της θόλωσης γίνεται επιτυχώς και παράγονται τα επιθυμητά αποτελέσματα, η χρονομέτρησή του βασίστηκε σε τρεις εικονικές μηχανές με διαφορετικό πλήθος κόμβων και διεργασιών. Αυτές οι εικονικές μηχανές είναι τα αρχεία ***nodes_2***, ***nodes_4*** και ***nodes_8***, με 2, 4 και 8 υπολογιστές αντίστοιχα η κάθε μία. Οι υπολογιστές οι οποίοι χρησιμοποιήθηκαν ήταν ο τοπικός και οι υπολογιστές από τον ***opti7020ws04*** έως ***opti7020ws10***. Όλοι οι παραπάνω υπολογιστές έχουν τα ίδια χαρακτηρίστηκα τα οποία αναφέρθηκαν στο κεφάλαιο της εισαγωγής στον *πίνακα 1*. Επιπλέον, ο αριθμός των διεργασιών που εφαρμόστηκε είναι 4xY = X, όπου το X θα είναι το συνολικός πλήθος των διεργασιών και το Y θα είναι το πλήθος των κόμβων.

Ο παραπάνω υπολογισμός γίνεται καθώς πρέπει να αξιοποιηθούν όλοι οι πυρήνες της εικονικής μηχανής, όπου κάθε μία αποτελείτε από 4 πυρήνες.

Η χρονομέτρηση έγινε με χρήση της συνάρτησης ***MPI_Wtime()*** και χρονομετρήθηκε ο συνολικός χρόνος εκτέλεσης του προγράμματος μαζί με τον χρόνο των overheads, από τον οποίο αφαιρείται ο χρόνος υπολογισμών της εφαρμογής. Κάθε πείραμα εκτελέστηκε τέσσερις φορές και υπολογίστηκαν οι μέσοι χρόνοι. Προσοχή δόθηκε να μην χρονομετρηθεί το κομμάτι που διαβάζει τις εικόνες από τον δίσκο και τις ετοιμάζει για την επεξεργασία μαζί και με το κομμάτι που κάνει εγγραφή στον δίσκο της θολωμένης εκδοχής της εικόνας μαζί με την αποδέσμευση χώρου μνήμης. Τα πειράματα έγιναν με βάσει της ακτίνας $r = 8$ και της εικόνας 1500.bmp.

Τα αποτελέσματα δίνονται στους παρακάτω *πίνακες 2,3,4* (οι χρόνοι είναι σε sec):

Πυρήνες – Διεργασίες	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μέσος Όρος
8	3,394101	3,340548	3,354567	3,405423	3,37365975
16	2,212597	2,032547	2,124595	2,125479	2,1238045
32	1,995621	1,956321	1,842654	1,795642	1,8975595

Πίνακας 2: Gaussian Blur Filter MPI Total Execution Time

Πυρήνες – Διεργασίες	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μέσος Όρος
8	0,207015	0,204587	0,195248	0,203647	0,20262425
16	0,503214	0,516583	0,565487	0,542148	0,531858
32	1,084622	1,201548	0,995423	0,958743	1,060084

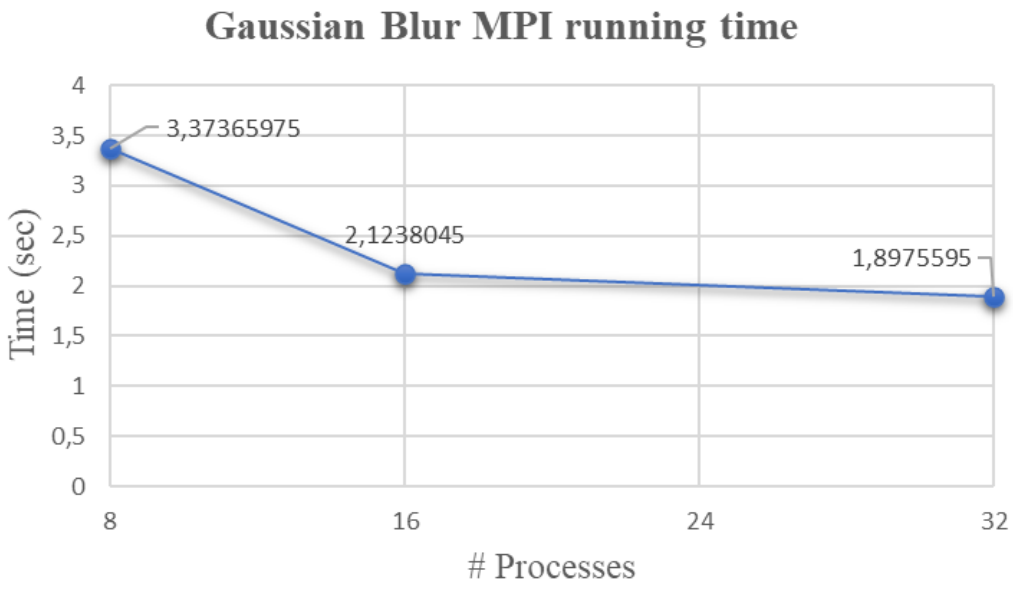
Πίνακας 3: Gaussian Blur Filter MPI Overheads Time

Πυρήνες – Διεργασίες	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μέσος Όρος
8	3,187087	3,135961	3,162087	3,201776	3,17172775
16	1,709383	1,515964	1,559108	1,583331	1,5919465
32	0,910999	0,754773	0,847231	0,836899	0,8374755

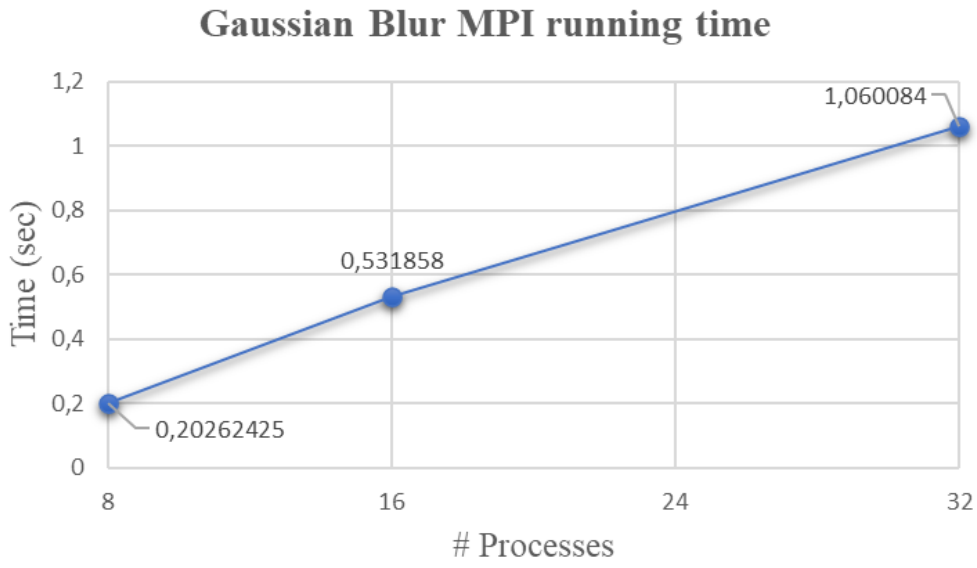
Πίνακας 4: Gaussian Blur Filter MPI Computation Time

Σειριακός Χρόνος Εκτέλεσης = 21,46479525 sec

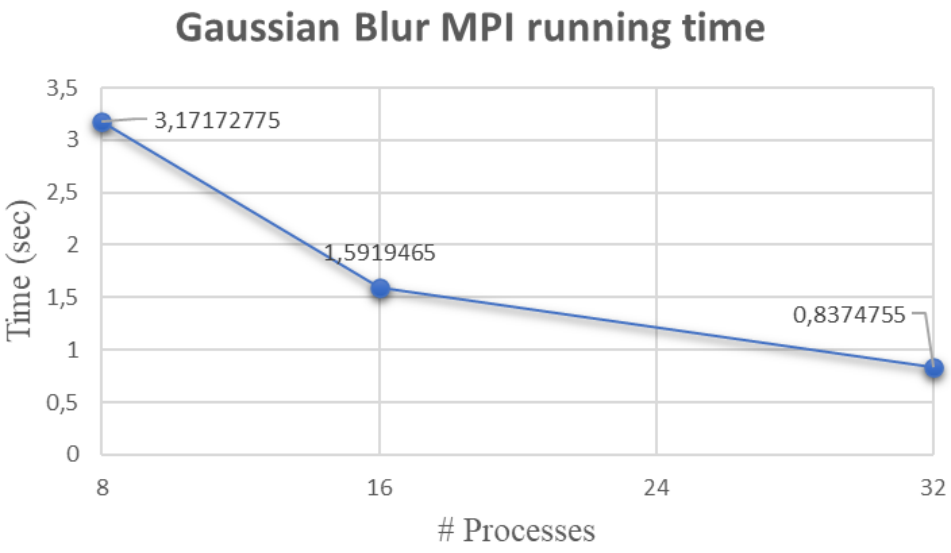
Τα αποτελέσματα των *πινάκων 2,3,4* φαίνονται στα παρακάτω *γραφήματα 1,2,3:*



Γράφημα 1: Gaussian Blur Total Execution Time Graph



Γράφημα 2: Gaussian Blur Overheads Time Graph



Γράφημα 3: Gaussian Blur Computation Time Graph

2.4 Σχόλια

Με βάση τα παραπάνω αποτελέσματα, παρατηρείται πως η αύξηση των διεργασιών επιφέρει μία σημαντική πτώση στον χρόνο εκτέλεσης και μάλιστα περίπου ιδανικά ανάλογα με τον αριθμό των διεργασιών. Επίσης, παρατηρείται πως με την αύξηση των κόμβων αυξάνεται και ο χρόνος των overheads. Αυτό είναι λογικό, καθώς με μεγαλύτερο πλήθος κόμβων εγκαθιδρύονται περισσότερες επικοινωνίες μεταξύ των υπολογιστών. Έτσι ο χρόνος θα αυξάνεται, παρόλο που ο χρόνος υπολογισμών μειώνεται.

3.Άσκηση – 2

3.1 Το Πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί ο πολλαπλασιασμός πινάκων με την χρήση του υβριδικού προγραμματισμού ***MPI/OpenMP***. Αυτό θα γίνει παραλληλοποιώντας μόνο τον εξωτερικό βρόχο για να διατρέχονται μόνο οι γραμμές και κάθε κόμβος θα αναλαμβάνει τον υπολογισμό μιας «λωρίδας» συνεχόμενων γραμμών. Τέλος, θα γίνει η χρήση συλλογικών επικοινωνιών για να μεταφέρονται τα δεδομένα ισόποσα στους κόμβους και μετά οι υπολογισμοί θα γίνονται από τους πυρήνες κάθε κόμβου παράλληλα.

3.2 Μέθοδοι Παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα (matmul_serial.c) από την ιστοσελίδα και τροποποιήθηκε κατάλληλα σε ένα νέο αρχείο το matmul_par.c το οποίο χρησιμοποιεί το υβριδικό μοντέλο ***MPI/OpenMP*** και με χρήση των συλλογικών επικοινωνιών. Αρχικά, ορίζεται στην αρχή της ***main()*** το ***MPI*** με την ***MPI_Init(&argc, &argv)*** για να πάρει τα κατάλληλα ορίσματα από το τερματικό. Στην συνέχεια, αρχικοποιείται ο αριθμός των κόμβων με την εντολή ***MPI_Comm_rank()*** και ο αριθμός των διεργασιών με την ***MPI_Comm_size()***. Καθώς έχουν διαβαστεί οι προηγούμενες πληροφορίες, το επόμενο βήμα είναι να οριστεί ο αριθμός των νημάτων που θα χρησιμοποιήσει το ***OpenMP***, και να υπολογιστεί και ο φόρτος ***WORK*** ο οποίος θα είναι στην ουσία ο φόρτος που θα έχει κάθε διεργασία να εκτελέσει. Το φόρτο ορίζεται ως ο λόγος ***N/nproc***, όπου ***N*** είναι η διαστάσεις του πίνακα και ***nproc*** ο αριθμός των διεργασιών. Έπειτα, η διεργασία με ***myid = 0*** είναι υπεύθυνη για να διαβάζει τους πίνακες ***A*** και ***B*** από τον δίσκο. Το επόμενο βήμα είναι να διαχωρίσει τον πίνακα ***A*** σε ισομερή κομμάτια, μεγέθους ***WORK*N,*** για να διασκορπιστούν στις υπόλοιπες διεργασίες με την συνάρτηση ***MPI_Scatter()***. Επιπλέον, ο πίνακας ***B*** εκπέμπεται σε όλους τους κόμβους της εικονικής μηχανής, με μέγεθος ***N*N***, με την εντολή ***MPI_Bcast()***.

Στη συνέχεια, αφού μεταδοθούν όλα τα κομμάτια των πινάκων στους εκάστοτε κόμβους, ξεκινάει η παράλληλη περιοχή και έχει την εξής εντολή πριν το πρώτο ***for loop***, όπου στην ουσία γίνονται οι υπολογισμοί. Η εντολή θα είναι η :

```
1. omp_set_num_threads(_NUM_THREADS);
2. #pragma omp parallel for private(j, k, sum) shared(Aa,B, Cc)
```

Κώδικας 3: Κώδικας OpenMP

όπου οι μεταβλητές ***j***, ***k*** και ***sum*** θα είναι ιδιωτικές ώστε κάθε νήμα να κάνει τους υπολογισμούς στο δικό του χώρο χωρίς να επηρεάζει τα υπόλοιπα ενώ οι πίνακες ***Aa***, ***B*** και ***Cc*** είναι κοινόχρηστοι. Στην προκειμένη περίπτωση δεν απαιτείται αμοιβαίος αποκλεισμός μιας και κάθε νήμα επηρεάζει διαφορετικά τα στοιχεία του υποπίνακα ***Cc***. Επίσης, το πρώτο ***for loop*** τροποποιήθηκε έτσι ώστε κάθε νήμα να επεξεργάζεται ένα μέρος της πράξης που ανατίθεται από την μεταβλητή ***WORK***. Επιπλέον, σε κάθε επιμέρους άθροισμα, χρησιμοποιείται ο αντίστοιχος υποπίνακας ***Aa*** με την πίνακα ***B*** και το αποτέλεσμα αποθηκεύεται στο κελί του κάθε υποπίνακα ***Cc***.

Κατά το πέρας της παράλληλης περιοχής, και αφού έχουν εκτελεστεί όλοι οι υπολογισμοί, αυτοί αποθηκεύονται σε έναν πίνακα ***Cc***, μεγέθους ***WORK*N***, ο οποίος έχει αρχικοποιηθεί πριν την έναρξη των επικοινωνιών και της μετάδοσης των πινάκων. Με την συνάρτηση ***MPI_Gather()*** η διεργασία ***myid = 0*** συλλέγει στον πίνακα ***C*** όλα τα κομμάτια του πίνακα τα οποία έχουν υπολογιστεί. Τέλος, η διεργασία ***myid = 0*** αναλαμβάνει την εγγραφή στον δίσκο τα αποτελέσματα από την προηγούμενη διαδικασία καθώς και να υπολογίσει όλους του χρόνους εκτέλεσης του προγράμματος. Συνολικά, οι εντολές που εφαρμόστηκαν για την μετάδοση και την λήψη των μηνυμάτων και της παράλληλης περιοχής για τους υπολογισμούς, φαίνονται παρακάτω:

```
1. MPI_Scatter(A, WORK*N, MPI_INT, Aa, WORK*N, MPI_INT, 0, MPI_COMM_WORLD);
2. MPI_Bcast(B, N*N, MPI_INT, 0, MPI_COMM_WORLD);
3. omp_set_num_threads(_NUM_THREADS);
4. #pragma omp parallel for private(j, k, sum) shared(Aa,B, Cc)
    for loop
5. MPI_Gather(Cc, WORK*N, MPI_INT, C, WORK*N, MPI_INT, 0, MPI_COMM_WORLD);
```

Κώδικας 4: Κώδικας Επικοινωνιών και υπολογισμών

3.3 Πειραματικά Αποτελέσματα – Μετρήσεις

Για να ελεγχθεί η σωστή λειτουργία του παράλληλου προγράμματος, δημιουργήθηκε και ένα απλό πρόγραμμα το οποίο συγκρίνει το αποτέλεσμα του σειριακού κώδικα με το αποτέλεσμα του παράλληλου. Καθώς, έχει βεβαιωθεί η σωστή λειτουργία του παράλληλου προγράμματος, το επόμενο βήμα είναι να χρονομετρηθεί το πρόγραμμα σε τρεις διαφορετικές εικονικές μηχανές.

Αυτές οι εικονικές μηχανές είναι τα αρχεία ***nodes_2***, ***nodes_4*** και ***nodes_8***, με 2, 4 και 8 υπολογιστές αντίστοιχα η κάθε μία. Οι υπολογιστές οι οποίοι χρησιμοποιήθηκαν ήταν ο τοπικός και οι υπολογιστές από τον ***opti7020ws04*** έως ***opti7020ws10***. Όλοι οι παραπάνω υπολογιστές έχουν τα ίδια χαρακτηρίστηκα τα οποία αναφέρθηκαν στο κεφάλαιο της εισαγωγής στον *πίνακα 1*. Επιπλέον, ο αριθμός των διεργασιών που εφαρμόστηκε είναι ο ίδιος με το πλήθος των κόμβων, διότι χρησιμοποιείται και το πλήθος νημάτων του ***OpenMP*** στην παράλληλη περιοχή. Επιπλέον, για την εκτέλεση των προγραμμάτων χρησιμοποιήθηκαν τα αρχεία των πινάκων με διάσταση 2048x2048.

Η χρονομέτρηση έγινε με χρήση της συνάρτησης ***MPI_Wtime()*** και χρονομετρήθηκε ο συνολικός χρόνος εκτέλεσης του προγράμματος μαζί με τον χρόνο των overheads, από τον οποίο αφαιρείται ο χρόνος υπολογισμών της εφαρμογής. Κάθε πείραμα εκτελέστηκε τέσσερις φορές και υπολογίστηκαν οι μέσοι χρόνοι. Προσοχή δόθηκε να μην χρονομετρηθεί το κομμάτι που διαβάζει τα αρχεία από τον δίσκο και το κομμάτι που κάνει εγγραφή στον δίσκο.

Τα αποτελέσματα δίνονται στους παρακάτω *πίνακες 5,6,7* (οι χρόνοι είναι σε sec):

Πυρήνες – Διεργασίες	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μέσος Όρος
2	40,428211	40,439598	40,082047	41,081391	40,50781175
4	37,288962	36,580768	37,369402	38,11525	37,3385955
8	19,99346	22,239154	21,376969	21,407199	21,2541955

Πίνακας 5: Hybrid Matrix Multiplication Total Execution Time

Πυρήνες – Διεργασίες	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μέσος Όρος
2	0,136200	0,139760	0,139353	0,135490	0,13770075
4	0,150854	0,150854	0,154359	0,152486	0,15213825
8	1,827866	4,447389	3,861109	3,482132	3,404624

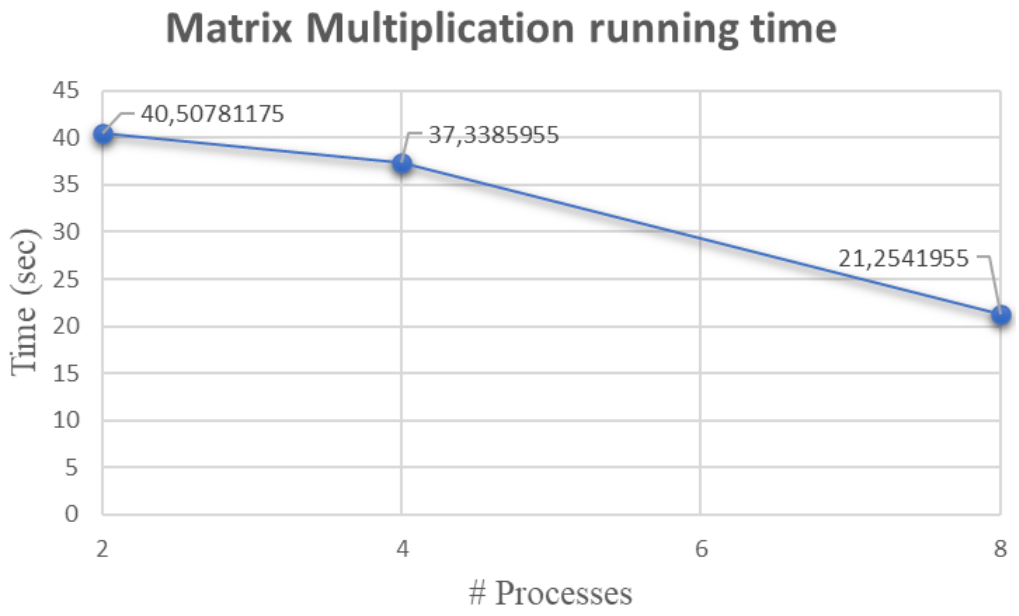
Πίνακας 6: Hybrid Matrix Multiplication Overheads Time

Πυρήνες – Διεργασίες	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μέσος Όρος
2	40,29201	40,299837	39,942694	40,945901	40,3701105
4	37,138107	36,415578	37,215043	37,962764	37,182873
8	18,165593	17,791766	18,771522	17,925067	18,163487

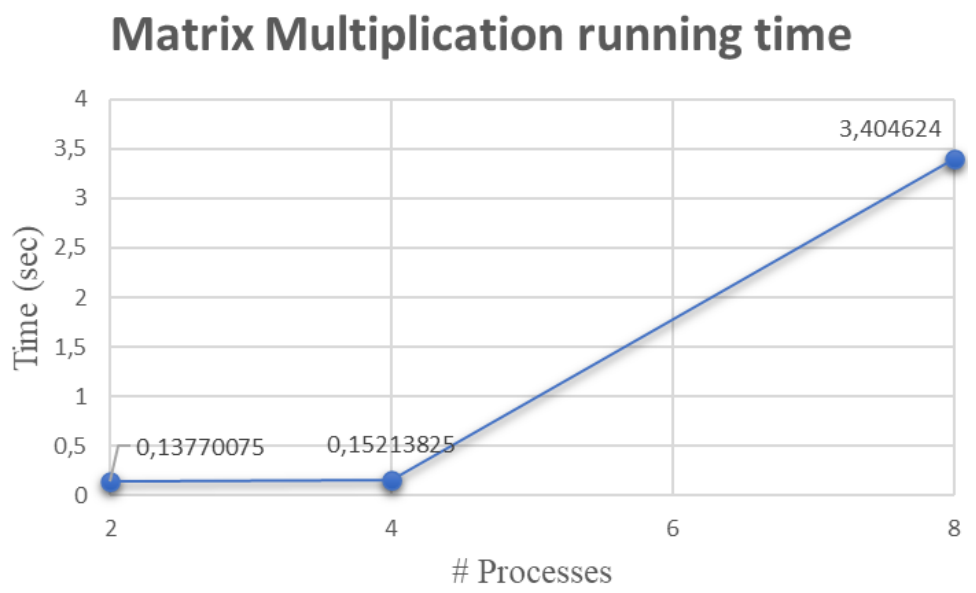
Πίνακας 7: Hybrid Matrix Multiplication Computation Time

Σειριακός Χρόνος Εκτέλεσης = 107.97993525 sec

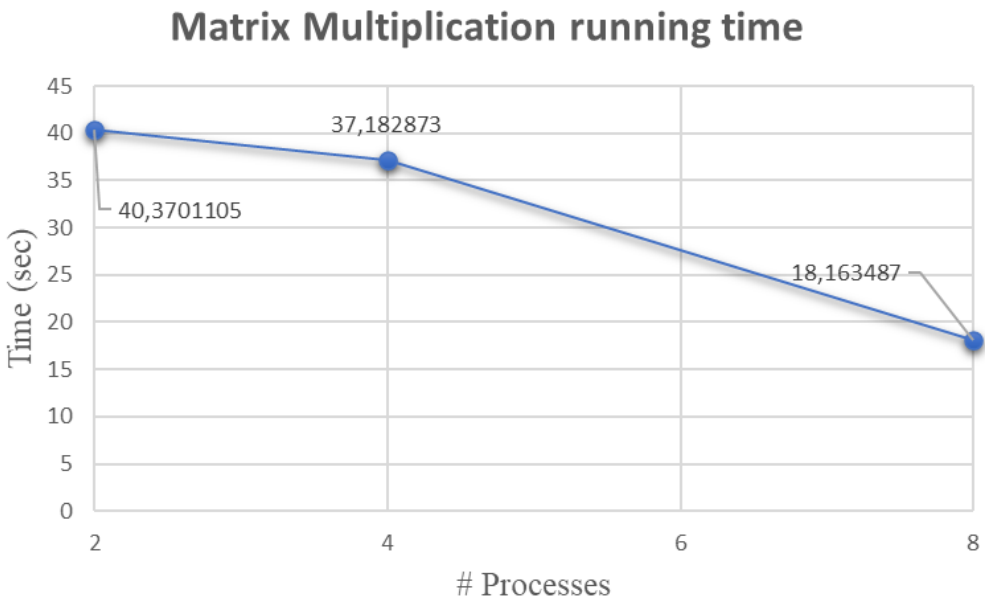
Τα αποτελέσματα των *πινάκων 5,6,7* φαίνονται στα παρακάτω *γραφήματα 4,5,6*:



Γράφημα 4: Hybrid Matrix Multiplication Total Time Graph



Γράφημα 5: Hybrid Matrix Multiplication Overheads Time Graph



Γράφημα 6: Hybrid Matrix Multiplication Computation Time Graph

3.4 Σχόλια

Με βάση τα παραπάνω αποτελέσματα, παρατηρείται πως η αύξηση των διεργασιών επιφέρει μία σημαντική πτώση στον χρόνο εκτέλεσης και μάλιστα περίπου στο μισό από τον σειριακό χρόνο. Ωστόσο, η εικονική μηχανή με 4 διεργασίες έχει περίπου ίδιο χρόνο με αυτόν της εικονικής μηχανής των 2. Αυτό οφείλεται στο γεγονός, πως οι δύο εικονικές μηχανές με 2 και 4 διεργασίες έχουν τον ίδιο χρόνο επικοινωνιών και τον σχεδόν τον ίδιο χρόνο υπολογισμών. Όμως, για την εικονική μηχανή με 8 διεργασίες, παρατηρείται πως ο συνολικός της χρόνος μειώνεται περίπου στο μισό συγκριτικά με τις άλλες δύο. Αυτό συμβαίνει διότι υπάρχει μεγαλύτερος αριθμός διεργασιών και σε σχέση με τα νήματα που εκτελούν τους υπολογισμούς γίνεται γρηγορότερα. Αυτό οφείλεται και στο γεγονός πως η κατανομή των πόρων γίνεται πιο δίκαιη συγκριτικά με τις υπόλοιπες εικονικές μηχανές. Το μόνο ελάττωμα της συγκεκριμένης εικονικής μηχανής είναι η ραγδαία αύξηση του χρόνου των επικοινωνιών το οποίο είναι λογικό, καθώς ο αριθμός των διεργασιών έχει αυξηθεί.

Βιβλιογραφία

[1] Β. Δημακόπουλος, *Παράλληλα Συστήματα και Προγραμματισμός*, Εκδόσεις ΣΕΑΒ, Μαρ. 2017 (1η αναθεωρημένη έκδοση).

[2] P. Pacheco, *An Introduction to Parallel Programming*, Morgan Kaufmann Publishers - Elsevier, 2015.

[3] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw Hill Education, International Edition 2003.

[4] N. MacDonald, E. Minty, J. Malard, T. Harding, S. Brown, M. Antonioletti, *Writing Message Passing Parallel Programs with MPI: A Two-Day Course on MPI Usage*, Version 1.8.2, Edinburgh Parallel Computing Center, 2018.

[5] W. Gropp, E. Lusk, R. Thakur, *Using MPI-2: Advanced Features of the Message-Parsing Interface*, MIT Press, November 1999.

[6] V. Eijkhourt, *Parallel Programming for Science and Engineering: Using MPI, OpenMP, and the PETSc library*, 3rd Edition, May 2022.