



ΔΗΜΟΚΡΙΤΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΡΑΚΗΣ

ΤΜΗΜΑ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ



Δημοκρίτειο Πανεπιστήμιο Θράκης

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών

Τομέας Λογισμικού και Ανάπτυξης Εφαρμογών

Εργαστήριο Προγραμματισμού και Επεξεργασίας Πληροφοριών

Εργασία με την χρήση της βιβλιοθήκης Open MP & MPI

Υπολογισμός των Ροπών H_u σε εικόνες

Παράλληλοι Αλγόριθμοι και Υπολογιστική Πολυπλοκότητα

Μαρία Αρετή Γερμανού
57807

Παναγιώτης Μαζιάνης
57915

1. Θεωρητική παρουσίαση του αλγορίθμου εύρεσης ροπών Hu:

Οι ροπές Hu είναι 7 μετρικές που χρησιμοποιούνται στην όραση υπολογιστών για ανάλυση εικόνων. Ο αλγόριθμος χρησιμοποιεί σαν είσοδο τις πληροφορίες των pixels (π.χ. χρώμα ή ένταση) και χρησιμοποιώντας κυρίως τον σταθμισμένο μέσο όρο των pixels εξάγει 7 πολύ χρήσιμες μετρικές. Αυτές οι μετρικές μας δίνουν πληροφορίες για το σχήμα και περίγραμμα ενός αντικειμένου στην εικόνα. Είναι σημαντικό να σημειωθεί ότι αυτές οι μετρικές παραμένουν αναλλοίωτες σε αλλαγές όπως μέγεθος, περιστροφή και αντανάκλαση αντικειμένου.

Ο κώδικας ακολουθεί κάποια συγκεκριμένα μαθηματικά βήματα για να καταλήξει στις ροπές Hu:

- Αρχικά βρίσκονται οι κεντρικές ροπές M_{00} , M_{01} και M_{10} χρησιμοποιώντας τον παρακάτω τύπο:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

οι κεντρικές ροπές χρησιμοποιούνται για να βρεθούν οι συντεταγμένες του κέντρου βάρους της εικόνας.

$$\bar{x} = \frac{M_{10}}{M_{00}}$$
$$\bar{y} = \frac{M_{01}}{M_{00}}$$

Η παραπάνω διαδικασία εκτελείται στη συνάρτηση `processImage()`:

```
for (i = 0; i < height; i++) {
    for (j = 0; j < width; j++) {
        // Υπολογισμός Centroids
        sum_x += j * binaryVector[i*width+j];
        sum_y += i * binaryVector[i*width+j];
        total += binaryVector[i*width+j];
    }
}

centroid_x = sum_x / total;
centroid_y = sum_y / total;
```

- Στη συνέχεια πρέπει να βρεθούν οι ζητούμενες κεντρικές ροπές χρησιμοποιώντας τον τύπο:

$$\mu_{ij} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j I(x, y)$$

και να κοινωνικοποιηθούν με τον τύπο:

$$\eta_{ij} = \frac{\mu_{i,j}}{\mu_{00}^{(i+j)/2+1}}$$

Η παραπάνω διαδικασία εκτελείται από την Calculate_Hu_central_moment() η οποία δέχεται εκτός των άλλων την τάξη των ροπών (p,q) και παράγει την συγκεκριμένη κεντρική ροπή κανονικοποιημένη:

```
double Calculate_Hu_central_moment(...){
    unsigned int central_moment_pq=0, central_moment_00=0;
    unsigned long i,j;
    double central_moment_pq_norm;

    for (i = 0; i < height; i++) {
        for (j = 0; j < width; j++) {
            central_moment_00 += (j-centroid_x) * (i-centroid_y) *
binaryVector[i*width+j];
            central_moment_pq += pow((j-centroid_x),p) * pow((i-centroid_y),q)
* binaryVector[i*width+j];
        }
    }

    central_moment_pq_norm = (double)(central_moment_pq) /
pow((double)(central_moment_00), 1+(p+q) / 2.0);
    return central_moment_pq_norm;
}
```

- Τέλος χρησιμοποιώντας τις παραγόμενες κεντρικές ροπές εξάγονται τα τελικά αποτελέσματα με τους τύπους Hu και κανονικοποιούνται λογαριθμικά:

$$\begin{aligned}
\varphi_1 &= \eta_{20} + \eta_{02} \\
\varphi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
\varphi_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
\varphi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
\varphi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12}) [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
\varphi_6 &= (\eta_{20} - \eta_{02}) [(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
&\quad + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
\varphi_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12}) [(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03}) [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned}$$

Στον κώδικα, καλείται πρώτα η `Calculate_Hu_central_moment()` και εξάγουμε τις κεντρικές ροπές που χρειαζόμαστε.

```
// Υπολογισμός κεντρικών ροπών και κανονικοποίηση τιμών
eta00 =
Calculate_Hu_central_moment(binaryVector,height,width,centroid_x,centroid_
y,0,0);
eta20 =
Calculate_Hu_central_moment(binaryVector,height,width,centroid_x,centroid_
y,2,0)/(eta00 * eta00);
eta02 =
Calculate_Hu_central_moment(binaryVector,height,width,centroid_x,centroid_
y,0,2)/(eta00 * eta00);
```

Και έπειτα, υπολογίζονται και κανονικοποιούνται οι ροπές Hu:

```
// Υπολογισμός Hu ροπών - Moments και κανονικοποίηση
Hu1 = eta20 + eta02;
Hu2 = (eta20 - eta02) * (eta20 - eta02) + 4 * eta11 * eta11;
Hu3 = (eta30 - 3 * eta12) * (eta30 - 3 * eta12) + (3 * eta21 - eta03) *
(3 * eta21 - eta03);
. . .

norm_hu1 = -logf(fabsf(Hu1));
norm_hu2 = -logf(fabsf(Hu2));
```

```

    norm_hu3 = -logf(fabsf(Hu3));

. . .

    // Εκτύπωση Hu moments
    printf("Hu Moments:\n");
    printf("Hu1: %.20f\n", norm_hu1);
    printf("Hu2: %.20f\n", norm_hu2);
    printf("Hu3: %.20f\n", norm_hu3);

. . .

```

2. Πολυπλοκότητα αλγορίθμου

Αρχικά αντιστοιχίζουμε τον μέγιστο χρόνο που θα χρειαστεί κάθε επανάληψη να διατρέξει κάθε στοιχείο και κάθε εντολή που καλείται μέσα σε αυτήν. Στην περίπτωση του υπάρχοντα αλγορίθμου είναι οι εμφωλευμένες for επαναλήψεις που εκτελούν πράξεις σταθερής πολυπλοκότητας σε κάθε εικονοστοιχείο της εικόνας για να βρεθούν οι κεντρικές ροπές.

```

for (i = 0; i < height; i++) {
    for (j = 0; j < width; j++) {
        .
        .
        .
        //O(1) operations
    }
}

```

Επομένως η πολυπλοκότητα υπολογίζεται ως το μέγεθος των επαναλήψεων (loop iterations) του μέγιστου χρόνου που αναφέρθηκε, πολλαπλασιασμένο με την πολυπλοκότητα του σώματος της επανάληψης.

Η πολυπλοκότητα εύρεσης μιας ροπής είναι $O(m*n)*O(1) = O(m*n)$. Όμως οι κεντρικές ροπές που χρειάζονται για την εύρεση των ροπών Hu είναι 9. Επίσης η εύρεση των συντεταγμένων του κέντρου βάρους απαιτεί ακόμα μία επανάληψη. Συνεπώς, η συνολική πολυπλοκότητα του αλγόριθμου είναι $O(m*n)*9+1 = 10 O(m*n) = O(10m*n)$. Όπου n και m ο αριθμός των επαναλήψεων (στη συγκεκριμένη περίπτωση height και width)

3. Μέθοδος δυναμικής δέσμευσης μνήμης

Κατά το άνοιγμα αρχείων εικόνων τα εικονοστοιχεία με την πληροφορία τους, αποθηκεύονται σε έναν δισδιάστατο πίνακα δυναμικής δέσμησης μνήμης. Ωστόσο, η επεξεργασία της εικόνας, μέχρι την τελική αποθήκευσή της σε αντίστοιχο δισδιάστατο πίνακα, για την δημιουργία του τελικού αρχείου εικόνας, γίνεται σε δέσμηση μονοδιάστατου χώρου. Επομένως γίνεται χρήση μονού δείκτη για δέσμηση 1Δ διανύσματος και χρήση αριθμητικής δεικτών 1Δ χώρου και στη συνέχεια απαιτείται η χρήση αριθμητικής δεικτών γιατί δε μπορεί να χρησιμοποιηθεί η σύνταξη `arr[i][j]`, εφόσον το `arr` είναι 1Δ. Λόγω της row wise αποθήκευσης των πινάκων στη C, το στοιχείο `[i][j]` ενός 2Δ πίνακα βρίσκεται στη θέση `[i*cols + j]` ενός 1Δ διανύσματος.

4. Βήματα επεξεργασίας εικόνας

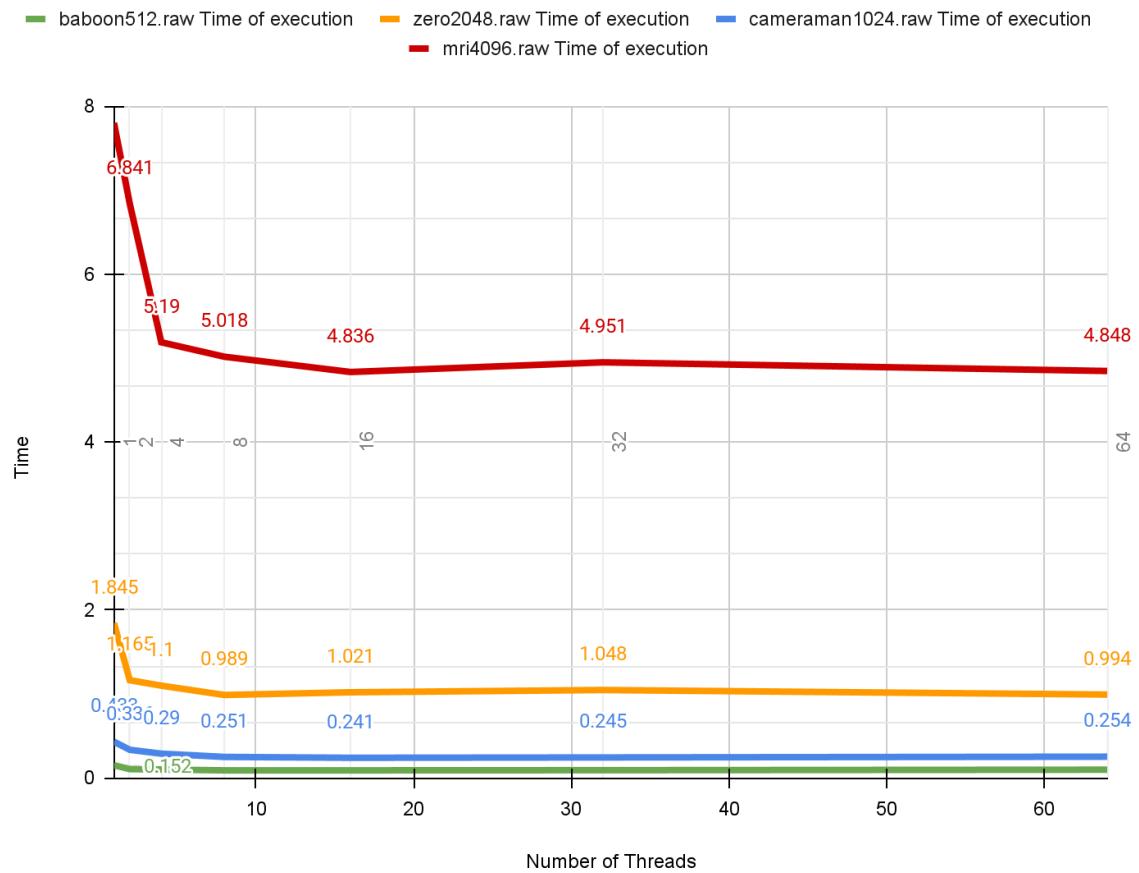
1. Αποθήκευση αρχείου εικόνας σε δισδιάστατο πίνακα με δέσμηση δυναμικής μνήμης.
2. Μεταφορά πληροφορίας εικονοστοιχείων από δισδιάστατο πίνακα σε διάνυσμα ίδιου μήκους με τον πίνακα με την χρήση αριθμητικής δεικτών μονοδιάστατου χώρου.
3. Μετατροπή εικόνας σε grayscale, αν χρειαστεί, καθώς η πληροφορία των εικόνων είναι κατά βάση η ένταση των εικονοστοιχείων που χρειάζεται για να αναπαραχθεί η grayscale εικόνα.
4. Μετατροπή της εικόνας σε δυαδικής μορφής binary έτσι ώστε να υπάρχουν μηδενικής ή μέγιστης έντασης εικονοστοιχεία (0 ή 255). Το threshold ορίζεται στην μέση τιμή του εύρους έντασης.
5. Χρήση της δυαδικής εικόνας για τους υπολογισμούς που απαιτούνται για τις ροπές H_u .
6. Υπολογισμός κεντρικών ροπών M_{00}, M_{01} και M_{10} .
7. Υπολογισμός κέντρου βάρους x και y συντεταγμένης.
8. Υπολογισμός κεντρικών ροπών M_{ij} .
9. Υπολογισμός H_u ροπών.
10. Λογαριθμική κανονικοποίηση H_u ροπών.

5. Αποτελέσματα Μετρήσεων

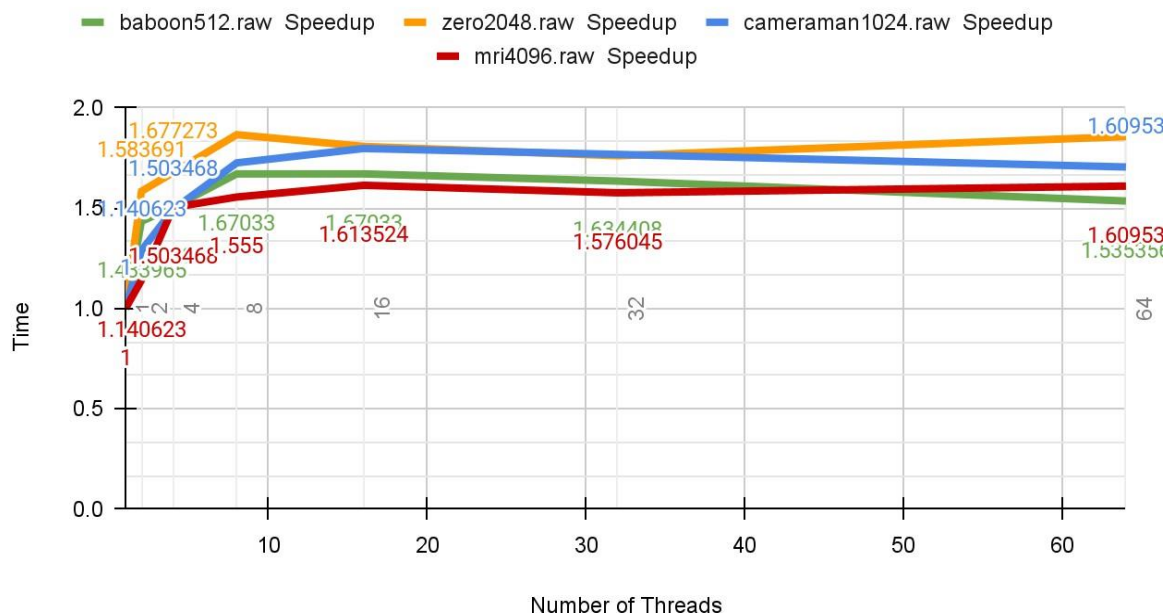
Image Tested	Number of threads							
		nt=1	nt=2	nt=4	nt=8	nt=16	nt=32	nt=64
baboon512.raw (512x512)	Time (seconds)	0.152	0.106	0.101	0.091	0.091	0.093	0.099
	Speedup	1.000	1.433 965	1.504 955	1.6703 30	1.6703 30	1.6344 08	1.5353 56
zero2048.raw (2048x2048)	Time (seconds)	1.845	1.165	1.100	0.989	1.021	1.048	0.994
	Speedup	1.000	1.583 691	1.677 273	1.8655 21	1.8070 52	1.7604 96	1.8561 36
cameraman1024.raw(1024x1024)	Time (seconds)	0.433	0.336	0.290	0.251	0.241	0.2450	0.254
	Speedup	1.000	1.288 691	1.493 1	1.7251	1.7966 81	1.7673 46	1.7047 24
mri4096.raw (4096x4096)	Time (seconds)	7.803 0	6.841 0	5.19	5.018	4.836	4.951	4.848
	Speedup	1.000	1.140 623	1.503 468	1.555	1.6135 24	1.5760 45	1.6095 30

Ακολουθούν διαγράμματα μετρήσεων για το 1ο παράδειγμα όπως και η έξοδος στην κονσόλα με τις εκτυπωμένες τιμές και μετρήσεις.

Time of execution and Number of Threads



Speedup and Number of Threads



Terminal Console Output:

Height: 512

Width: 512

Execution time: 0.152000 seconds

Speedup: 1.000000, Number of cores: 1

Height: 512

Width: 512

Execution time: 0.106000 seconds

Speedup: 1.433965, Number of cores: 2

Height: 512

Width: 512

Execution time: 0.101000 seconds

Speedup: 1.504955, Number of cores: 4

Height: 512

Width: 512

Execution time: 0.091000 seconds

Speedup: 1.670330, Number of cores: 8

Height: 512

Width: 512

Execution time: 0.091000 seconds

Speedup: 1.670330, Number of cores: 16

Height: 512

Width: 512

Execution time: 0.093000 seconds

Speedup: 1.634408, Number of cores: 32

Height: 512

Width: 512

```
Execution time: 0.099000 seconds
Speedup: 1.535356, Number of cores: 64
```

6. Χρήση MPI

Με την χρήση του εργαλείου MPI, πραγματοποιείται η παραλληλοποίηση εργασιών σε καθορισμένο αριθμό πυρήνων. Όσον αφορά τον κώδικα δεν αλλάζει η ροή του προγράμματος, δηλαδή διατηρείται η ίδια μέθοδος υπολογισμών με το πρώτο μέρος. Γίνεται χρήση των συναρτήσεων Scatterv - Gatherv και της μεθόδου εξάλειψης υπολοίπου από την παραλληλοποίηση που χωρίζει τον πίνακα ή διάνυσμα ή εικόνα σε ακέραια αρχικά τμήματα. Έπειτα ακολουθούν οι μετρήσεις με τις ίδιες εικόνες που χρησιμοποιήθηκαν στο πρώτο μέρος.

Προσθήκες στον κώδικα, “Main”:

Απαραίτητες εντολές επικοινωνίας του εργαλείου MPI. Έχουν προστεθεί το MPI_INIT για την αρχικοποίηση του MPI, το MPI_Comm_rank που επιστρέφει το rank, αριθμό πυρήνα διεργασίας, του communicator MPI_COMM_WORLD και το MPI_Comm_size που επιστρέφει το πλήθος του συνολικού αριθμού πυρήνων του communicator.

```
int rank, size;

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

// Ensure at least 2 processes are available
if (size < 2) {
    fprintf(stderr, "This program requires at least 2 processes.\n");
    MPI_Abort(MPI_COMM_WORLD, 1);
}
```

```
MPI_Finalize();
```

Προσθήκες στον κώδικα, “ProcessImage()”:

Απαραίτητες εντολές επικοινωνίας του εργαλείου MPI.

```
int rank, size, *sendcounts, *displs;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Απαραίτητες εντολές λήψης χρόνου του εργαλείου MPI. Το MPI_Barrier είναι μια σημαντική εντολή της mpi που λειτουργεί σαν ένα σημείο συγχρονισμού για τις διεργασίες εντός ενός communicator. Χρησιμοποιείται για να διασφαλιστεί ότι όλες οι διεργασίες που

συμμετέχουν στην επικοινωνία έχουν φτάσει στο συγκεκριμένο σημείο του προγράμματος πριν οποιαδήποτε από αυτές προχωρήσει περαιτέρω. Χρησιμοποιείται πριν από κάθε εντολή λήψης χρόνου.

```
MPI_Barrier(MPI_COMM_WORLD);
start_allocation_time = MPI_Wtime();
```

Εντολές που εκτελούνται μόνο στον κύριο πυρήνα του εργαλείου MPI. Στην προκειμένη δηλώνεται το διάνυσμα της εικόνας στην οποία θα μεταφερθεί ο δισδιάστατος πίνακας εικόνας για να διανεμηθεί στους πυρήνες.

```
if (rank == 0) {
    input_img_vector = (unsigned char *)malloc(height * width *
sizeof(unsigned char));

    if (input_img_vector == NULL) {
        perror("Error allocating memory for input image");
        exit(-1);
    }

    for (i = 0; i < height; i++) {
        for (j = 0; j < width; j++) {
            input_img_vector[i * width + j] = input_img[i][j];
        }
    }
}
```

Δήλωση τοπικών buffers μήκους όσο και οι διεργασίες που διανέμονται χρήσιμες για την παραλληλοποίηση.

```
// Δέσμευση μνήμης rec_buffer σε όλες τις διεργασίες
rec_buffer = (unsigned char *)malloc(width * height / size *
sizeof(unsigned char));
// Δέσμευση μνήμης sent_buffer σε όλες τις διεργασίες
sent_buffer = (unsigned int *)malloc(width * height / size *
sizeof(unsigned int));
```

Απαραίτητες εντολές λήψης χρόνου του εργαλείου MPI.

```
MPI_Barrier(MPI_COMM_WORLD);
end_allocation_time = MPI_Wtime();
start_time_parallel = MPI_Wtime();
```

Διανύσματα υπεύθυνα για την διαχείριση υπολοίπου κατά την παραλληλοποίηση και για

την ποσότητα εργασιών για τον κάθε πυρήνα.

```
// Μέθοδος διαχείρισης υπολοίπου δεδομένων μεταξύ του διαμοιρασμού εικόνας
στον αριθμό των πυρήνων
// Ορισμός sendcounts and displs για MPI_Scatterv και MPI_Gatherv
sendcounts = (int *)malloc(size * sizeof(int));
displs = (int *)malloc(size * sizeof(int));

// Μέτρηση αριθμού σειρών που θα αναλάβει κάθε διεργασία
int remaining_rows = height % size;
int base_rows = height / size;

// Προσάρμοσε sendcounts και displs για τις σειρές που υπολοίπονται
for (i = 0; i < size; i++) {
    sendcounts[i] = base_rows * width;
    if (i < remaining_rows) {
        sendcounts[i] += width;
    }

    displs[i] = i * base_rows * width;
    if (i < remaining_rows) {
        displs[i] += i * width;
    } else {
        displs[i] += remaining_rows * width;
    }
}
```

Απαραίτητες εντολές για την παραλληλοποίηση του διανύσματος εικόνας που αποστέλλεται τμηματικά σε buffer τοπικό.

```
MPI_Scatterv(input_img_vector, sendcounts, displs, MPI_UNSIGNED_CHAR,
rec_buffer, sendcounts[rack], MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD);
```

Αλλαγή ποσότητας επαναλήψεων για την παραλληλοποίηση.

```
for (i = 0; i < height / size; i++) {
    for (j = 0; j < width; j++) {...}
}
```

Απαραίτητες εντολές για την παραλληλοποίηση του διανύσματος εικόνας που συλλέγεται τμηματικά από τον δεύτερο buffer τοπικό.

```
// Gather the results back to the root process
```

```

MPI_Gatherv(sent_buffer, sendcounts[rank], MPI_UNSIGNED, binaryVector,
sendcounts, displs, MPI_UNSIGNED, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);
end_time_parallel = MPI_Wtime();

```

Απαραίτητες εντολές για την παραλληλοποίηση του διανύσματος εικόνας που συλλέγεται τμηματικά από τον δεύτερο buffer τοπικό.

```

if (rank == 0) {
    time_parallel = end_time_parallel - start_time_parallel;
    printf("\n\nElapsed time for allocation only: %f seconds\n",
end_allocation_time - start_allocation_time);
    printf("Elapsed parallel time: %f seconds\n\n", time_parallel);
    free(input_img_vector);
    free(rec_buffer);
    free(sent_buffer);
}

```

Απαραίτητες εντολές για την αποδέσμευση μνήμης.

```

free(sendcounts);
free(displs);

```

Αποτελέσματα Μετρήσεων MPI παράλληλου κώδικα με χρήση υπολογιστών:

Image Tested	Number of threads		
	Rank = 0	-n = 4	-n = 8
baboon512.raw(512x512)	Time (seconds)	Elapsed parallel time: 0.201861 seconds	Elapsed parallel time: 0.110902 seconds
zero2048.raw (2048x2048)	Time (seconds)	Elapsed parallel time: 4.141434 seconds	Elapsed parallel time: 1.737315 seconds
cameraman1024.raw(1024x1024)	Time (seconds)	Elapsed parallel time: 0.802493 seconds	Elapsed parallel time: 0.441347 seconds
mri4096.raw (4096x4096)	Time (seconds)	Elapsed parallel time: 15.553390 seconds	Elapsed parallel time: 6.923559 seconds

```
[marigerm1@diopsis3 Project1]$ mpirun -machinefile m3 -np 4 test baboon512.raw 512
```

512

Debug Point 1 - Process 0 of 4
Debug Point 2 - Process 0: After Scatter
Debug Point 3 - Process 0: After Data Processing
Debug Point 4 - Process 0: Before MPI_Gatherv

Elapsed time for allocation only: 0.002490 seconds
Elapsed parallel time: 0.201861 seconds

[marigerm1@diopsis3 Project1]\$ mpirun -machinefile m3 -np 4 test cameraman1024.raw 1024 1024

Debug Point 1 - Process 0 of 4
Debug Point 2 - Process 0: After Scatter
Debug Point 3 - Process 0: After Data Processing
Debug Point 4 - Process 0: Before MPI_Gatherv

Elapsed time for allocation only: 0.008969 seconds
Elapsed parallel time: 0.802493 seconds

[marigerm1@diopsis3 Project1]\$ mpirun -machinefile m3 -np 4 test mri4096.raw 4096 4096

Debug Point 1 - Process 0 of 4
Debug Point 2 - Process 0: After Scatter
Debug Point 3 - Process 0: After Data Processing
Debug Point 4 - Process 0: Before MPI_Gatherv

Elapsed time for allocation only: 0.193912 seconds
Elapsed parallel time: 15.553390 seconds

Elapsed time for allocation only: 0.002474 seconds
Elapsed parallel time: 0.110902 seconds

[marigerm1@diopsis3 Project1]\$ mpirun -machinefile m3 -np 4 test zero2048.raw 2048 2048

Debug Point 1 - Process 0 of 4
Debug Point 2 - Process 0: After Scatter
Debug Point 3 - Process 0: After Data Processing
Debug Point 4 - Process 0: Before MPI_Gatherv

Elapsed time for allocation only: 0.045517 seconds
Elapsed parallel time: 4.141434 seconds

```
[marigerm1@dioptis3 Project1]$ mpirun -machinefile m3 -np 8 test baboon512.raw 512 512
```

```
Debug Point 1 - Process 0 of 8
```

```
Debug Point 2 - Process 0: After Scatter
```

```
Debug Point 3 - Process 0: After Data Processing
```

```
Debug Point 4 - Process 0: Before MPI_Gatherv
```

```
[marigerm1@dioptis3 Project1]$ mpirun -machinefile m3 -np 8 test zero2048.raw 2048 2048
```

```
Debug Point 1 - Process 0 of 8
```

```
Debug Point 2 - Process 0: After Scatter
```

```
Debug Point 3 - Process 0: After Data Processing
```

```
Debug Point 4 - Process 0: Before MPI_Gatherv
```

```
Elapsed time for allocation only: 0.053850 seconds
```

```
Elapsed parallel time: 1.737315 seconds
```

```
[marigerm1@dioptis3 Project1]$ mpirun -machinefile m3 -np 8 test cameraman1024.raw 1024 1024
```

```
Debug Point 1 - Process 0 of 8
```

```
Debug Point 2 - Process 0: After Scatter
```

```
Debug Point 3 - Process 0: After Data Processing
```

```
Debug Point 4 - Process 0: Before MPI_Gatherv
```

```
Elapsed time for allocation only: 0.009018 seconds
```

```
Elapsed parallel time: 0.441347 seconds
```

```
[marigerm1@dioptis3 Project1]$ mpirun -machinefile m3 -np 8 test mri4096.raw 4096 4096
```

```
Debug Point 1 - Process 0 of 8
```

```
Debug Point 2 - Process 0: After Scatter
```

```
Debug Point 3 - Process 0: After Data Processing
```

```
Debug Point 4 - Process 0: Before MPI_Gatherv
```

```
Elapsed time for allocation only: 0.139365 seconds
```

```
Elapsed parallel time: 6.923559 seconds
```

Σε αυτό το σημείο θα ακολουθήσει ένα παράδειγμα από την ολοκληρωμένη έξοδο στο terminal της πρώτης εικόνας καλώντας τις παρακάτω εντολές:

```
ssh dioptis3
```

```
mpicc Project_MPI.c -o Project_MPI -lm
```

```
mpirun -machinefile m3 -np 8 Project1_MPI baboon512.raw 512 512
```

Έξοδος:

```
Debug Point 1 - Process 0 of 8
Debug Point 2 - Process 0: After Scatter
Debug Point 3 - Process 0: After Data Processing
Debug Point 4 - Process 0: Before MPI_Gatherv
```

Centroid X: 265.000000

Centroid Y: 30.000000

Total: 1709265

Hu Moments:

Hu1: 7.65137529373168945312

Hu2: 11.76259517669677734375

Hu3: 34.31407546997070312500

Hu4: 33.81059646606445312500

Hu5: 69.33482360839843750000

Hu6: 39.84848785400390625000

Hu7: 67.66077423095703125000

Elapsed time for allocation only: 0.012487 seconds

Elapsed parallel time: 0.118437 seconds

```
Debug Point 1 - Process 3 of 8
Debug Point 2 - Process 3: After Scatter
Debug Point 3 - Process 3: After Data Processing
Debug Point 4 - Process 3: Before MPI_Gatherv
```

Centroid X: 254.000000

Centroid Y: 30.000000

Total: 3207645

Hu Moments:

Hu1: 8.48170757293701171875

Hu2: 14.47630596160888671875

Hu3: 36.11396408081054687500

Hu4: 36.45029449462890625000

Hu5: 73.16221618652343750000

Hu6: 44.15835189819335937500

Hu7: 73.49655914306640625000

Elapsed time for allocation only: 0.011231 seconds
Elapsed parallel time: 0.118930 seconds

Debug Point 1 - Process 7 of 8
Debug Point 2 - Process 7: After Scatter
Debug Point 3 - Process 7: After Data Processing
Debug Point 4 - Process 7: Before MPI_Gatherv

Centroid X: 223.000000
Centroid Y: 32.000000
Total: 3035775
Hu Moments:
Hu1: 8.28418827056884765625
Hu2: 17.09061431884765625000
Hu3: 35.95250701904296875000
Hu4: 35.28196716308593750000
Hu5: 70.95727539062500000000
Hu6: 43.89405059814453125000
Hu7: 73.56116485595703125000

Elapsed time for allocation only: 0.003983 seconds
Elapsed parallel time: 0.119063 seconds

Debug Point 1 - Process 6 of 8
Debug Point 2 - Process 6: After Scatter
Debug Point 3 - Process 6: After Data Processing
Debug Point 4 - Process 6: Before MPI_Gatherv

Centroid X: 185.000000
Centroid Y: 35.000000
Total: 2441370
Hu Moments:
Hu1: 7.38097190856933593750
Hu2: 15.42175388336181640625
Hu3: 33.74687576293945312500
Hu4: 33.98311614990234375000
Hu5: 68.06264495849609375000
Hu6: 42.25194549560546875000
Hu7: 68.09020996093750000000

Elapsed time for allocation only: 0.002037 seconds
Elapsed parallel time: 0.118591 seconds

Debug Point 1 - Process 2 of 8
Debug Point 2 - Process 2: After Scatter
Debug Point 3 - Process 2: After Data Processing
Debug Point 4 - Process 2: Before MPI_Gatherv

Centroid X: 274.000000

Centroid Y: 32.000000

Total: 3443265

Hu Moments:

Hu1: 8.10659599304199218750

Hu2: 14.49409770965576171875

Hu3: 35.59945678710937500000

Hu4: 37.14772796630859375000

Hu5: 73.52196502685546875000

Hu6: 44.39495468139648437500

Hu7: 73.36870574951171875000

Elapsed time for allocation only: 0.011653 seconds

Elapsed parallel time: 0.118459 seconds

Debug Point 1 - Process 5 of 8
Debug Point 2 - Process 5: After Scatter
Debug Point 3 - Process 5: After Data Processing
Debug Point 4 - Process 5: Before MPI_Gatherv

Centroid X: 139.000000

Centroid Y: 31.000000

Total: 1731195

Hu Moments:

Hu1: 6.82292079925537109375

Hu2: 11.86564636230468750000

Hu3: 34.09579467773437500000

Hu4: 34.30266571044921875000

Hu5: 68.50956726074218750000

Hu6: 40.23649215698242187500

Hu7: 68.38932800292968750000

Elapsed time for allocation only: 0.002219 seconds

Elapsed parallel time: 0.119020 seconds

Debug Point 1 - Process 4 of 8
Debug Point 2 - Process 4: After Scatter

Debug Point 3 - Process 4: After Data Processing
Debug Point 4 - Process 4: Before MPI_Gatherv

Centroid X: 211.000000
Centroid Y: 29.000000
Total: 2984775
Hu Moments:
Hu1: 9.06102752685546875000
Hu2: 14.19275951385498046875
Hu3: 35.80618667602539062500
Hu4: 36.20885848999023437500
Hu5: 72.33327484130859375000
Hu6: 43.30578613281250000000
Hu7: 72.77593231201171875000

Elapsed time for allocation only: 0.003274 seconds
Elapsed parallel time: 0.118564 seconds

Debug Point 1 - Process 1 of 8
Debug Point 2 - Process 1: After Scatter
Debug Point 3 - Process 1: After Data Processing
Debug Point 4 - Process 1: Before MPI_Gatherv

Centroid X: 268.000000
Centroid Y: 32.000000
Total: 2566065
Hu Moments:
Hu1: 7.20260190963745117188
Hu2: 15.32122230529785156250
Hu3: 31.80839538574218750000
Hu4: 32.90592575073242187500
Hu5: 66.65700531005859375000
Hu6: 40.90423965454101562500
Hu7: 64.99311065673828125000

Elapsed time for allocation only: 0.011780 seconds
Elapsed parallel time: 0.118887 seconds

Αποτελέσματα Μετρήσεων MPI σύγκριση σειριακού και παράλληλου κώδικα με την χρήση 8 υπολογιστών:

Επιτάχυνση (S): Η επιτάχυνση μετράει πόσο πιο γρήγορα τρέχει ένα παράλληλο πρόγραμμα σε σύγκριση με τον σειριακό του αντίστοιχο. Ο τύπος για την επιτάχυνση είναι:

$$Speedup = \frac{T_{serial}}{T_{parallel}}$$

όπου T_{serial} είναι ο χρόνος εκτέλεσης του σειριακού κώδικα, και $T_{parallel}$ είναι ο χρόνος εκτέλεσης του παράλληλου κώδικα.

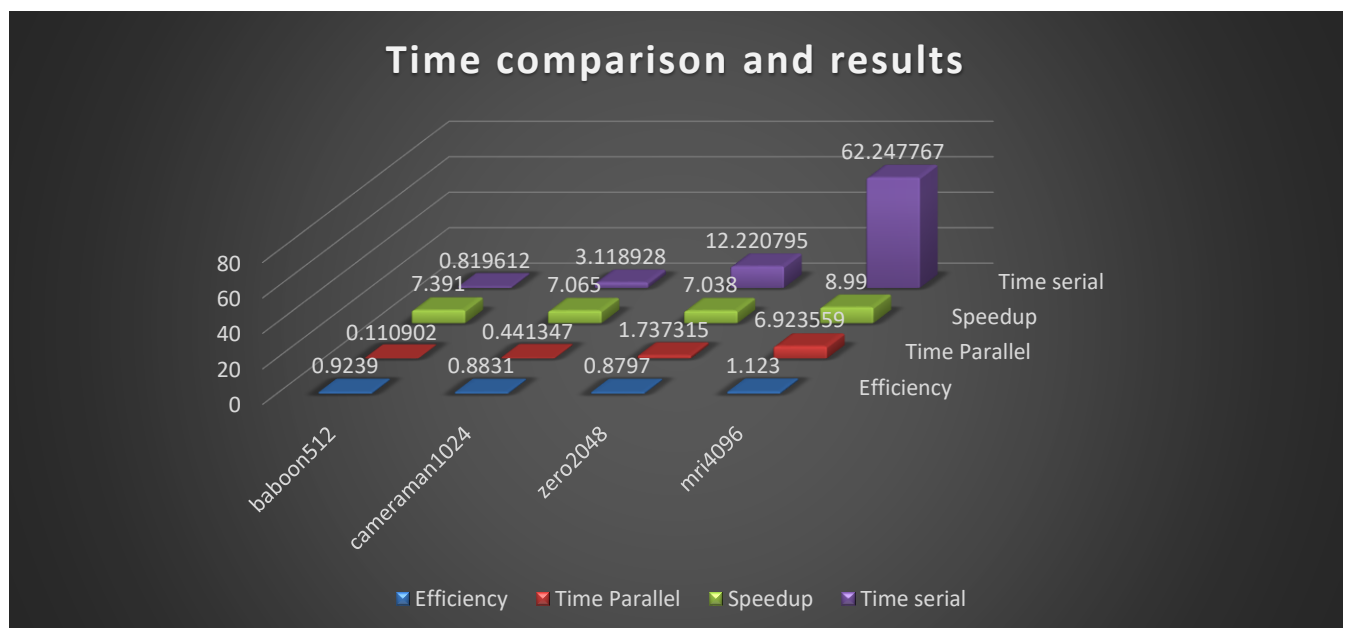
Αποτελεσματικότητα (E): Η αποτελεσματικότητα μετρά την εκμετάλλευση των υπολογιστικών πόρων σε ένα παράλληλο σύστημα. Ο τύπος για την αποτελεσματικότητα είναι:

$$E = \frac{Speedup}{P}$$

όπου $Speedup$ είναι η επιτάχυνση και P είναι ο αριθμός των επεξεργαστών (ή διεργασιών).

Image Tested	Time serial	Time Parallel	Speedup	Efficiency
baboon512.raw (512x512)	0.819612	0.110902	7.391	0.9239
zero2048.raw (2048x2048)	12.220795	1.737315	7.038	0.8797
cameraman1024.raw (1024x1024)	3.118928	0.441347	7.065	0.8831
mri4096.raw (4096x4096)	62.247767	6.923559	8.990	1.123

Είναι σημαντικό να σημειωθεί ότι παρόλο που η υπέρβαση του 1 είναι ασυνήθιστη, μπορεί να συμβεί σε κάποιες περιπτώσεις. Ωστόσο, σε πολλές περιπτώσεις, η αποτελεσματικότητα που πλησιάζει το 1 θεωρείται πολύ αποτελεσματική για παράλληλο υπολογισμό.



7. Βιβλιογραφία

- [1] [Analysis of Loop in Programming \(enjoyalgorithms.com\)](http://enjoyalgorithms.com)
- [2] [Chapter 5 Shared Memory Programming with OpenMP \(studylib.net\)](http://studylib.net)
- [3] [Find Center of a Blob \(Centroid\) Using OpenCV \(C++/Python\) | LearnOpenCV](#)
- [4] [Shape Matching using Hu Moments \(C++ / Python\) | LearnOpenCV](#)
- [5] [https://www.researchgate.net/publication/224146066 Analysis of Hu's moment invariants on image scaling and rotation](https://www.researchgate.net/publication/224146066)