ΔΗΜΟΚΡΙΤΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΡΑΚΗΣ

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εργαστήριο Αρχιτεκτονικής Υπολογιστών και Συστημάτων Υψηλών Επιδόσεων

Παράλληλοι Αλγόριθμοι και Υπολογιστική Πολυπλοκότητα

“Δυαδικοποίηση εικόνων κατα Otsu ”

Συντάκτες:

Φαίδρα Μπογιάνογλου Βραχνού	**5ο ακάδ. έτος**
Ιάκωβος Μιχαηλίδης	**5ο ακάδ. έτος**

- Ξάνθη, 2024 -

Περιεχόμενα

Εισαγωγή στον αλγόριθμο Otsu

Η μέθοδος Otsu είναι ένας δημοφιλής αλγόριθμος για την αυτόματη διχοτόμηση εικόνων (image thresholding), η οποία πήρε το όνομά της από τον Ιάπωνα Nobuyuki Otsu. Χρησιμοποιείται κυρίως για την επεξεργασία και ανάλυση εικόνων. Στόχος του αλγόριθμου είναι να χωρίζει μια εικόνα σε δύο μέρη: το φόντο (background) και το προσκήνιο (foreground) με βάση το ιστόγραμμα της εικόνας. Η τοποθέτηση των pixel της εικόνας σε μία από τις δύο κατηγορίες γίνεται με τον ορισμό ενός κατωφλίου. Η διαδικασία αυτή γίνεται αυτόματα και ο αλγόριθμος προσπαθεί να βρει την καλύτερη δυνατή τιμή για τον διαχωρισμό. Η τιμή αυτή θα πρέπει να ελαχιστοποιεί την διακύμανση των τιμών μέσα σε κάθε ομάδα αλλά ταυτόχρονα να μεγιστοποιεί την διαφορά ανάμεσα στις δύο ομάδες.

Οι εικόνες που θα χρησιμοποιηθούν για την υλοποίηση του προγράμματος είναι σε μορφή grayscale, οπότε το ιστόγραμμα τους αντιπροσωπεύεται από 256 τιμές. Κάθε pixel έχει 8-bit, άρα το ιστόγραμμα παίρνει τιμές από το 0 μέχρι το 255. Ανάλογα την συχνότητα που εμφανίζεται μία τιμή στην εικόνα τόσο μεγαλύτερο είναι το ιστόγραμμα σε αυτήν την τιμή. Σκοπός του αλγορίθμου Otsu είναι η επιλογή της κατάλληλης τιμής T κατωφλίου του ιστογράμματος ώστε να χωριστούν τα pixel σε δύο κατηγορίες αυτήν με τιμές κάτω από το T (δηλαδή το background) και τιμές πάνω από το T (δηλαδή το foreground).

Ο αλγόριθμος προσπαθεί να βρει το καλύτερο δυνατό threshold ώστε να διαχωρίσει την εικόνα με τον βέλτιστο τρόπο. Για αυτόν τον λόγο χρησιμοποιείται μία επαναληπτική μέθοδος, ώστε να εξετάσει όλες τις δυνατές τιμές του T. Ο αλγόριθμος κάνει χρήση πιθανοτήτων και υπολογίζει ένα σταθμισμένο άθροισμα των διακυμάνσεων των δύο ομάδων. Ο υπολογισμός αυτός γίνεται για κάθε τιμή από το 0 έως το 255 και το τελικό T είναι αυτό που μεγιστοποιεί το αθροισμα αυτό. Συγκεκριμένα ο υπολογισμός γίνεται με την παρακάτω εξίσωση :

$$\sigma_{\omega}^{2}(t) = \omega_{0}(t)\sigma_{0}^{2}(t) + \omega_{1}(t)\sigma_{1}^{2}(t)$$

Ορίζουμε αρχικά:

- $p(i)$: πιθανότητα ένα εικονοστοιχείο να έχει ένταση i
- $\omega_1(\mathrm{T})$: αθροιστική πιθανότητα της ομάδας 1 (0 έως το T).
- $\omega_2(\mathrm{T})$: αθροιστική πιθανότητα της ομάδας 2 ($\mathrm{T}+1$ έως το 255).

$$\omega_1(\mathrm{T}) = \sum_{i=0}^{T} p(i), \quad \omega_2(\mathrm{T}) = \sum_{i=T+1}^{255} p(i) \quad (1)$$

Οι μέσες τιμές έντασης για τις δύο ομάδες:

$$\mu_1(\mathrm{T}) = \frac{\sum_{i=0}^{T} i \cdot p(i)}{\omega_1(\mathrm{T})}, \quad \mu_2(\mathrm{T}) = \frac{\sum_{i=T+1}^{255} i \cdot p(i)}{\omega_2(\mathrm{T})} \quad (2)$$

Διασπορά μεταξύ κλάσεων:

Η διασπορά μεταξύ των κλάσεων $\sigma_b^2(T)$ υπολογίζεται ως εξής:

$$\sigma_b^2(T) = \omega_1(\mathrm{T}) \cdot \omega_2(\mathrm{T}) \cdot (\mu_1(\mathrm{T}) - \mu_2(\mathrm{T}))^2 \quad (3)$$

Αυτή η τιμή μετρά το πόσο διαχωρισμένες είναι οι δύο κλάσεις. Όσο μεγαλύτερη η τιμή, τόσο καλύτερος ο διαχωρισμός. Ο αλγόριθμος δοκιμάζει όλα τα πιθανά κατώφλια T (από 0 έως 255) και υπολογίζει $\sigma_b^2(T)$ για το καθένα. Το βέλτιστο κατώφλι Tc είναι αυτό που μεγιστοποιεί τη $\sigma_b^2(T)$.

$$\mathrm{T}c = argmax_T \sigma_b^2(T) \quad (4)$$

Σειριακή υλοποίηση του αλγορίθμου Otsu

Στην σειριακή εκδοχή του αλγορίθμου όλα εκτελούνται από μία και μόνο διεργασία. Οπότε αρχικά υπολογίζεται το ιστόγραμμα της εικόνας και έπειτα γίνεται η επιλογή του κατάλληλου κατωφλίου. Αν η εικόνα έχει N*N στοιχεία τότε ο υπολογισμός του ιστογράμματος είναι γραμμικός ως προς τον αριθμό των pixel και η πολυπλοκότητα θα είναι O(N^2), γενικά θα είναι ίση η πολυπλοκότητα με τον αριθμό των pixel της εικόνας. Ο χωρισμός των pixel με βάση το κατώφλι προσθέτει μία επιπλέον πολυπλοκότητα της τάξης O(K) όπου K=256 όσα δηλαδή και τα πιθανά κατώφλια. Συνολικά οπότε, η πολυπλοκότητα δίνεται από την πρόσθεση των δύο και θα είναι ίση με O(N^2 + K). Αν όμως ο αριθμός των pixel είναι αρκετά μεγάλος τότε το K μπορεί να παραληφθεί για λόγους μιας ολοκληρωμένης εικόνας της απόδοσης του αλγορίθμου δεν θα πραγματοποιηθεί αυτή η απλοποίηση.

Αρκετά απλή είναι όμως η υλοποίηση της μεθόδου αρκεί να εφαρμοστούν οι εξισώσεις που προαναφέρθηκαν. Παρακάτω δίνεται ο ψευδοκώδικας που βοήθησε στο τελικό αποτέλεσμα του σειριακού κώδικα.

```
algorithm OtsusMethod(I):
    // INPUT
    //    I = Grayscale image
    // OUTPUT
    //    T_opt = Optimal threshold value

    Compute grayscale histogram H(i)

    for T <- 0 to 255:
        Compute P_0(T) and P_1(T) using Eq. (1)
        Compute m_0(T) and m_1(T) using Eq. (2)
        Compute between-class variance var(T) using Eq. (3)

    Find the optimal threshold value T_opt using Eq. (4)
    Apply threshold T_opt to image I to obtain binary image.
```

Ουσιαστικά εφαρμόζονται τέσσερα απλά βήματα με έναν επαναληπτικό βρόγχο. Αρχικά υπολογίζεται το ιστόγραμμα και οι πιθανότητες για κάθε τιμή. Ορίζονται αρχικές τιμές στα ω και μ . Εκτελείται ο επαναληπτικός βρόχος για όλα τα διαφορετικά T ανανεώνοντας τα ω και μ και υπολογίζοντας το σ . Τέλος επιλέγεται το κατάλληλο T για την μεγιστοποίηση του σ.

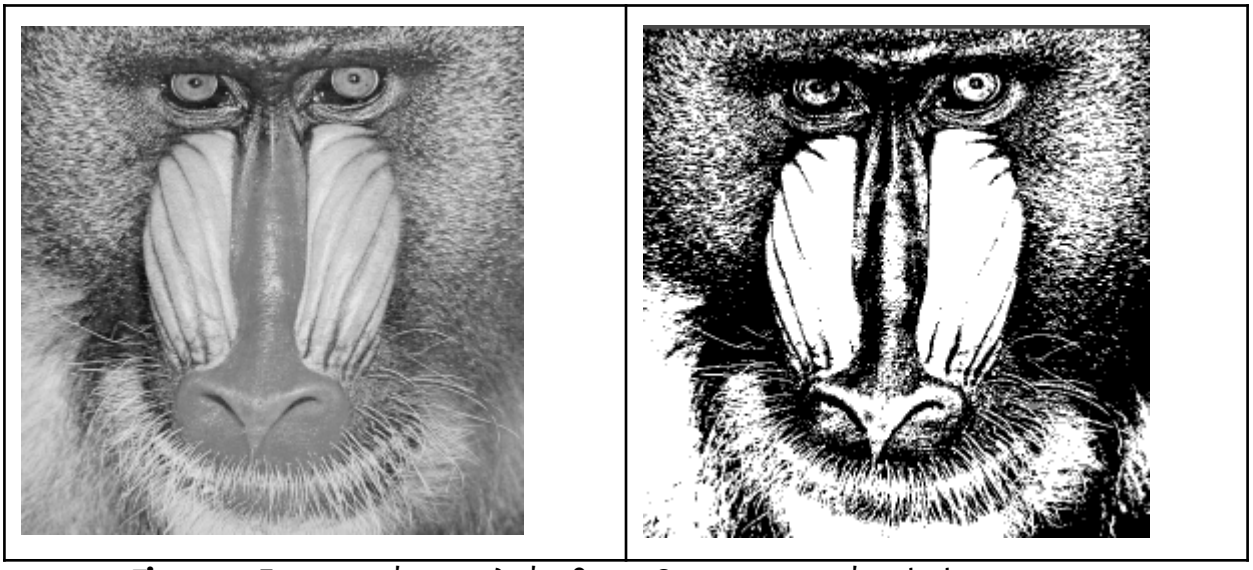

Figure 1: Εφαρμογή του αλγόριθμου Otsu στην εικόνα baboon514.raw

Παράλληλη υλοποίηση του αλγορίθμου Otsu (MPI)

Ο αλγόριθμος Otsu μπορεί αρκετά εύκολα να παραλληλοποιηθεί με την βοήθεια του MPI ή του OpenMP. Στο πρώτο μία κεντρική διεργασία με id=0 διαβάζει την εικόνα και έπειτα με την χρήση της Scatter η εικόνα μοιράζεται σε όλες τις διεργασίες. Οι διαφορετικές εργασίες υπολογίζουν το τοπικό ιστόγραμμα και με την χρήση της reduce προστίθενται για να δημιουργήσουν το global_histogram το οποίο αντιγράφεται σε όλους με την χρήση της MPI_Bcast. Αφού τώρα όλες οι διεργασίες κατέχουν όλο το ιστόγραμμα κάθε διεργασία ελέγχει συγκεκριμένες τιμές threshold εφαρμόζοντας τοπικά τον επαναληπτικό αλγόριθμο. Η εκτέλεση του αλγορίθμου τερματίζεται με την επιλογή του καλύτερου threshold με σύγκριση των αποτελεσμάτων από όλες τις διεργασίες και εφαρμογή του στην εικόνα παράγοντας την έξοδο. Αναλυτικότερα:

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

#define GRAY_LEVELS 256

void calculate_histogram(unsigned char *image, int size, int *histogram) {
    for (int i = 0; i < size; i++) {
        histogram[image[i]]++;
    }
}

int otsu_threshold(int *histogram, int total_pixels) {
    int sum = 0, sumB = 0, q1 = 0, q2 = 0;
    float max_var = 0.0, threshold = 0.0;

    for (int i = 0; i < GRAY_LEVELS; i++) {
        sum += i * histogram[i];
    }
    for (int i = 0; i < GRAY_LEVELS; i++) {
        q1 += histogram[i];
        if (q1 == 0) continue;

        q2 = total_pixels - q1;
        if (q2 == 0) break;

        sumB += i * histogram[i];
        float m1 = (float)sumB / q1;
        float m2 = (float)(sum - sumB) / q2;
        float var_between = (float)q1 * q2 * (m1 - m2) * (m1 - m2);

        if (var_between > max_var) {
            max_var = var_between;
            threshold = i;
        }
    }
    return (int)threshold;
}

void apply_threshold(unsigned char *image, int size, int threshold) {
    for (int i = 0; i < size; i++) {
        image[i] = (image[i] > threshold) ? 255 : 0;
    }
}
```

Ο κύριος επεξεργαστής (`rank == 0`) διαβάζει την εικόνα σε μορφή raw και τη μετατρέπει σε μονοδιάστατο πίνακα για την επεξεργασία της

```
    if (rank == 0) {
        io_start = MPI_Wtime(); // Χρόνος έναρξης I/O
        image = (unsigned char **)malloc(height * sizeof(unsigned char *));
        for (i = 0; i < height; i++) {
            image[i] = (unsigned char *)malloc(width * sizeof(unsigned
char));
        }
        read_rawimage(input_file, height, width, image);
        io_end = MPI_Wtime(); // Χρόνος λήξης I/O
    }
```

Ο πίνακας εικόνας χωρίζεται σε ίσα μέρη και αποστέλλεται στους επεξεργαστές (`MPI_Scatter`). Κάθε επεξεργαστής λαμβάνει ένα υποσύνολο των δεδομένων για επεξεργασία.

```
    scatter_start = MPI_Wtime(); // Χρόνος έναρξης Scatter
    if (rank == 0) {
        unsigned char *flat_image = (unsigned char *)malloc(total_pixels *
sizeof(unsigned char));
        for (i = 0; i < height; i++) {
            memcpy(flat_image + i * width, image[i], width);
        }
        MPI_Scatter(flat_image, local_size, MPI_UNSIGNED_CHAR, local_image,
local_size, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD);
        free(flat_image);
    } else {
        MPI_Scatter(NULL, local_size, MPI_UNSIGNED_CHAR, local_image,
local_size, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD);
    }
    scatter_end = MPI_Wtime(); // Χρόνος λήξης Scatter
```

Κάθε επεξεργαστής υπολογίζει το ιστόγραμμα για το τμήμα της εικόνας που του ανατέθηκε. Το ιστόγραμμα καταγράφει τη συχνότητα εμφάνισης των αποχρώσεων του γκρι.

```
    compute_start = MPI_Wtime(); // Χρόνος έναρξης Υπολογισμού
    local_histogram = (int *)calloc(GRAY_LEVELS, sizeof(int));
    calculate_histogram(local_image, local_size, local_histogram);
```

Τα τοπικά ιστογράμματα συγχωνεύονται (`MPI_Reduce`) σε ένα συνολικό ιστόγραμμα από τον κύριο επεξεργαστή.

```
    if (rank == 0) global_histogram = (int *)calloc(GRAY_LEVELS,
sizeof(int));
    MPI_Reduce(local_histogram, global_histogram, GRAY_LEVELS, MPI_INT,
MPI_SUM, 0, MPI_COMM_WORLD);
```

Ο κύριος επεξεργαστής (`rank == 0`) εφαρμόζει τον αλγόριθμο Otsu για να βρει το βέλτιστο κατώφλι που διαχωρίζει την εικόνα σε δύο κατηγορίες: φωτεινές και σκοτεινές περιοχές. Ύστερα, το κατώφλι μεταδίδεται σε όλους τους επεξεργαστές(`MPI_Bcast`), οι οποίοι το εφαρμόζουν στα τμήματα της εικόνας που επεξεργάζονται.

```
    int threshold = 0;
    if (rank == 0) {
        threshold = otsu_threshold(global_histogram, total_pixels);
    }
    MPI_Bcast(&threshold, 1, MPI_INT, 0, MPI_COMM_WORLD);
    apply_threshold(local_image, local_size, threshold);
    compute_end = MPI_Wtime(); // Χρόνος λήξης Υπολογισμού
```

Τα επεξεργασμένα τμήματα της εικόνας επιστρέφουν στον κύριο επεξεργαστή με την χρήση της `MPI_Gather`, και ανασυνθέτει ο ίδιος την τελική εικόνα.

```
    gather_start = MPI_Wtime(); // Χρόνος έναρξης Gather
    if (rank == 0) {
        unsigned char *flat_image = (unsigned char *)malloc(total_pixels *
sizeof(unsigned char));
        MPI_Gather(local_image, local_size, MPI_UNSIGNED_CHAR, flat_image,
local_size, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD);
        for (i = 0; i < height; i++) {
            memcpy(image[i], flat_image + i * width, width);
        }
        free(flat_image);
        write_rawimage(output_file, height, width, image);
        for (i = 0; i < height; i++) free(image[i]);
        free(image);
        free(global_histogram);
    } else {
```

```
        MPI_Gather(local_image, local_size, MPI_UNSIGNED_CHAR, NULL,
local_size, MPI_UNSIGNED_CHAR, 0, MPI_COMM_WORLD);
    }
    gather_end = MPI_Wtime(); // Χρόνος λήξης Gather
```

Μετρικές από την υλοποίηση σε διαφορετικά μεγέθη εικόνων

Εφαρμόσαμε τον αλγόριθμό σειριακά και παράλληλα για πυρήνες 1,2,4 και 8, πάνω στην ίδια εικόνα drone.raw σε διαφορετικά μεγέθη και λάβαμε τις εξής μετρήσεις:

N	Metric	Serial	P = 1	P = 2	P = 4	P = 8
512x512	Time	0.045	0.022497	0.021713	0.023614	0.02507
	Speedup	-	2	2.072	1.905	1.795
	Efficiency	-	2	1.036	0.476	0.224
1024x1024	Time	0.095	0.077373	0.081846	0.090834	0.121479
	Speedup	-	1.22	1.161	1.045	0.782
	Efficiency	-	1.22	0.580	0.261	0.0977
2048x2048	Time	0.182	0.235403	0.248703	0.231536	0.248227
	Speedup	-	0.773	0.732	0.786	0.733
	Efficiency	-	0.773	0.366	0.196	0.092

Figure 2: Εφαρμογή του αλγόριθμου Otsu στην εικόνα drone514.raw

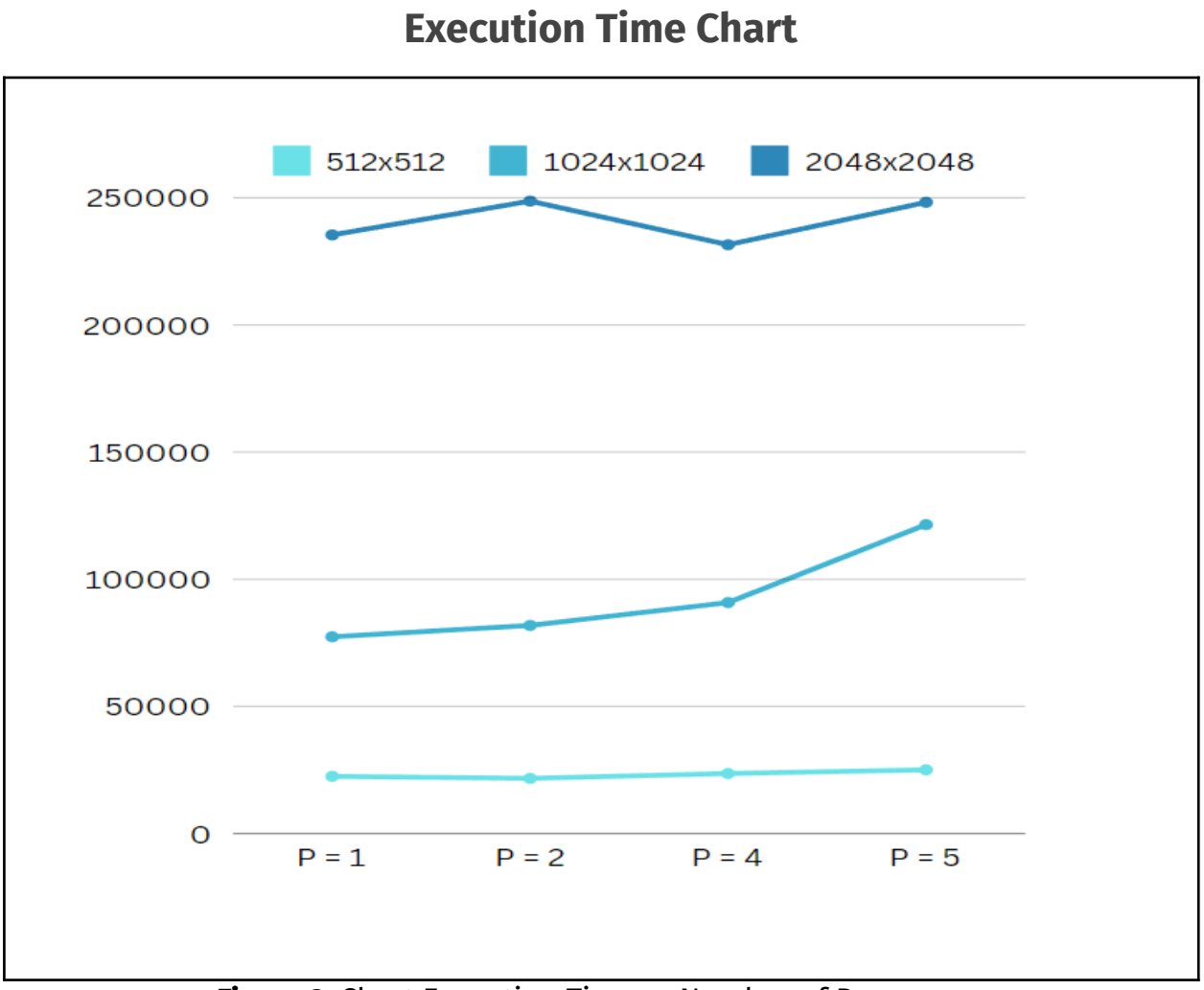

Figure 3: Chart Execution Time vs Number of Processors

Speed Up Chart

Figure 4: Chart Speedup vs Number of Processors

Efficiency Chart

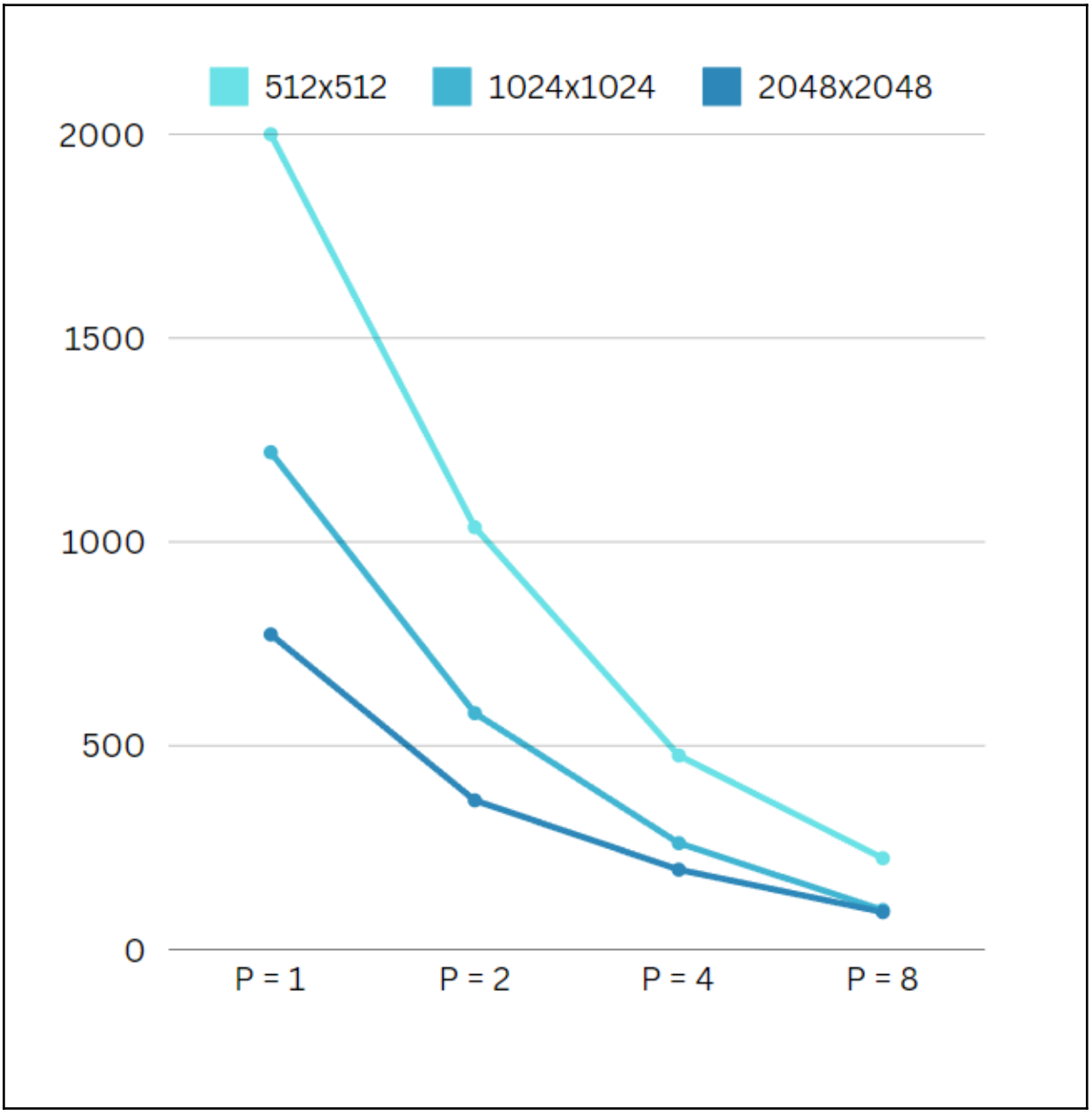

Figure 5: Chart Efficiency vs Number of Processors

Συμπερασματα

Τα αποτελέσματα των μετρήσεων δείχνουν ότι η αποδοτικότητα της παραλληλίας εξαρτάται σε μεγάλο βαθμό από το μέγεθος της εικόνας και τον αριθμό των επεξεργαστών που χρησιμοποιούνται. Στις μικρότερες εικόνες, παρατηρείται σημαντική βελτίωση στους χρόνους εκτέλεσης με την αύξηση του αριθμού των επεξεργαστών, ιδιαίτερα όταν ο αριθμός τους είναι περιορισμένος. Ωστόσο, καθώς ο αριθμός των επεξεργαστών αυξάνεται, η αποδοτικότητα μειώνεται σημαντικά, γεγονός που υποδεικνύει αυξημένα επικοινωνιακά κόστη και φαινόμενα κορεσμού.

Σε μεσαία μεγέθη εικόνων, η παραλληλία δεν επιτυγχάνει την επιθυμητή κλιμάκωση, καθώς οι χρόνοι εκτέλεσης δεν μειώνονται ικανοποιητικά, και σε ορισμένες περιπτώσεις οι επιδόσεις είναι συγκρίσιμες ή χειρότερες από τη σειριακή εκτέλεση. Αυτό μπορεί να οφείλεται σε αδυναμία του αλγορίθμου να διαχειριστεί αποτελεσματικά τη διανομή του φόρτου ή σε υψηλά κόστη επικοινωνίας που υπερβαίνουν τα οφέλη της διαίρεσης του έργου.

Σε πολύ μεγάλα μεγέθη εικόνων, η παραλληλία αποτυγχάνει να αποδώσει αποδοτικά, καθώς παρατηρείται έντονη πτώση της επιτάχυνσης και της αποδοτικότητας. Οι χρόνοι εκτέλεσης δεν επωφελούνται από την αύξηση των επεξεργαστών, γεγονός που υποδεικνύει ότι οι επικοινωνιακές καθυστερήσεις και οι εξαρτήσεις του αλγορίθμου περιορίζουν δραστικά την απόδοσή του. Γενικά, τα αποτελέσματα δείχνουν ότι η αποδοτικότητα της παραλληλίας μειώνεται όσο αυξάνεται το μέγεθος του προβλήματος και ο αριθμός των επεξεργαστών, υποδεικνύοντας την ανάγκη για βελτιστοποίηση του αλγορίθμου και των επικοινωνιακών μηχανισμών, ώστε να αξιοποιηθούν πλήρως οι διαθέσιμοι πόροι.

Βιβλιογραφία

---

1. https://en.m.wikipedia.org/wiki/Otsu's_method (θεωρία)
2. N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62-66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4310076&isnumber=4310064 (εξισώσεις)
3. https://www.baeldung.com/cs/otsu-segmentation (ψευδοκωδικας)
4. https://www.ipol.im/pub/art/2016/158/article_lr.pdf
5. https://medium.com/@vignesh.g1609/image-segmentation-using-otsu-threshold-selection-method-856ccdacf22