Ειδικά Θέματα Παράλληλου Προγραμματισμού 2023-24

Σετ Ασκήσεων 2

Παναγιώτης Τριανταφυλλίδης

AM: 3200199

p3200199@aueb.gr

Το αντικείμενο της εργασίας είναι η φόρτωση και επεξεργασία εικόνων. Κάθε εικόνα αποτελείται από έναν αριθμό pixels όπου το χρώμα του κάθε πίξελ αναπαρίσταται από τις τιμές των 3 βασικών χρωμάτων Red , Green , Blue . Συνήθως σε αυτά τα χρωματικά κανάλια προσθέτουμε και ένα τέταρτο κανάλι, Alpha , το οποίο αναπαριστά το ποσοστό της διαφάνειας του pixel. Τα κανάλια αποθηκεύονται ως μεταβλητές τύπου unsigned char , που παίρνουν τιμές από το 0 (καθόλου συνεισφορά) έως το 255 (μέγιστη συνεισφορά). Η κάθε εικόνα αποθηκεύεται ως ένας πίνακας από unsigned chars μεγέθους [width * height * number_of_channels].

Οι μετρήσεις έγιναν στο παρακάτω σύστημα:

Επεξεργαστής	AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
Πλήθος πυρήνων	4
Πλήθος λογικών πυρήνων	8
Λειτουργικό σύστημα	Windows 10
Μεταφραστής	MSVC v143

Άσκηση 1

Το πρόβλημα

Στην πρώτη εργασία (HW1) μας είχε δοθεί η συνάρτηση gaussian_blur_separate_serial() , η οποία εφαρμόζει την τεχνική δύο περασμάτων από Gaussian Blur ώστε να επιταχύνει το αποτέλεσμα της θόλωσης της εικόνας “street_night.jpg”. Μας ζητείται να δημιουργήσουμε μία νέα συνάρτηση (gaussian_blur_separate_parallel()), στην οποία να φορτώνεται η ίδια εικόνα και έπειτα να γίνεται η θόλωση παράλληλα, χρησιμοποιώντας τις οδηγίες παραλληλοποίησης των for loops της OpenMP. Τέλος, να αποθηκεύσουμε την εικόνα σε ένα αρχείο με όνομα “blurred_image_parallel.jpg”.

Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα της gaussian_blur_separate_serial για την δομή της gaussian_blur_separate_parallel.

Για τον αρχικό ορισμό των νημάτων χρησιμοποιήθηκαν οι οδηγίες παραλληλοποίησης της OpenMP. Για την εισαγωγή τους στο πρόγραμμα χρειάστηκε η εντολή:

```
#include <omp.h>
```

Αρχικά χρειάζεται να παραλληλοποιήσουμε την θόλωση της εικόνας στον οριζόντιο άξονα. Για να το κάνουμε αυτό, θα χρειαστεί να χρησιμοποιήσουμε την οδηγία:

```
#pragma omp parallel for
```

Αυτή η οδηγία ενημερώνει τον compiler να εκτελέσει το παρακάτω for loop παράλληλα, για πολλά νήματα (χωρίς καθορισμένο αριθμό νημάτων.

Ωστόσο βλέπουμε πως δεν έχουμε ένα αλλά τρία for loops και ευτυχώς με την βιβλιοθήκη της OpenMP μπορούμε να συνδυάσουμε τις τρεις επαναλήψεις σε μια, ώστε με αυτόν τον τρόπο να έχουμε μια συνδυασμένη τελική επανάληψη που εκτελεί κάθε νήμα με την εντολή:

```
collapse(3)
```

Τέλος, για να ισομοιράσουμε το φόρτο εργασίας μεταξύ των νημάτων, χρησιμοποιούμε την οδηγία:

```
schedule(static)
```

Έτσι κάθε νήμα, εκτελώντας κάθε φορά ένα συνδυασμό τριών for loop, με ισάριθμο φορτίο εργασίας, θολώνει την εικόνας στον οριζόντιο άξονα. Ο τελικό κώδικας της οριζόντιας θόλωσης είναι ον εξής:

```
// Horizontal Blur
#pragma omp parallel for collapse(3) schedule(static)
for (int y = 0; y < height; y++)
{
	for (int x = 0; x < width; x++)
	{
		int pixel = y * width + x;
		for (int channel = 0; channel < 4; channel++)
		{
			img_horizontal_blur[4 * pixel + channel] = blurAxis(x, y,
channel, 0, img_in, width, height);
		}
	}
}
```

Αντίστοιχη είναι και η προσέγγιση μας για την παραλληλοποίηση της θόλωσης της εικόνας στον κατακόρυφο άξονα:

```
// Vertical Blur
#pragma omp parallel for collapse(3) schedule(static)
for (int y = 0; y < height; y++)
{
	for (int x = 0; x < width; x++)
	{
		int pixel = y * width + x;
		for (int channel = 0; channel < 4; channel++)
		{
			img_out[4 * pixel + channel] = blurAxis(x, y, channel, 1,
img_horizontal_blur, width, height);
		}
	}
}
```

Πρέπει όμως να επισημανθεί ότι πριν την θόλωση της εικόνας κατακόρυφα, όσο και πριν τον υπολογισμού του τελικού χρόνου και την εγγραφή της blurred_image_parallel, έχουν διεκπεραιωθεί οι προηγούμενες εγρασίες ώστε να επιτύχουμε τον κατάλληλο συγχρονισμό με τα barriers. Αυτό συμβαίνει προκαθορισμένα από τις οδηγίες της OpenMP. Για την απενεργοποίηση των barriers απαιτείται το “nowait”.

Πειραματικά αποτελέσματα – μετρήσεις

Η χρονομέτρηση έγινε με τις εντολές :

```
// Timer to measure performance
auto start = std::chrono::high_resolution_clock::now();

auto end = std::chrono::high_resolution_clock::now();

// Computation time in milliseconds
int time = (int)std::chrono::duration_cast<std::chrono::milliseconds>(end -
start).count();
```

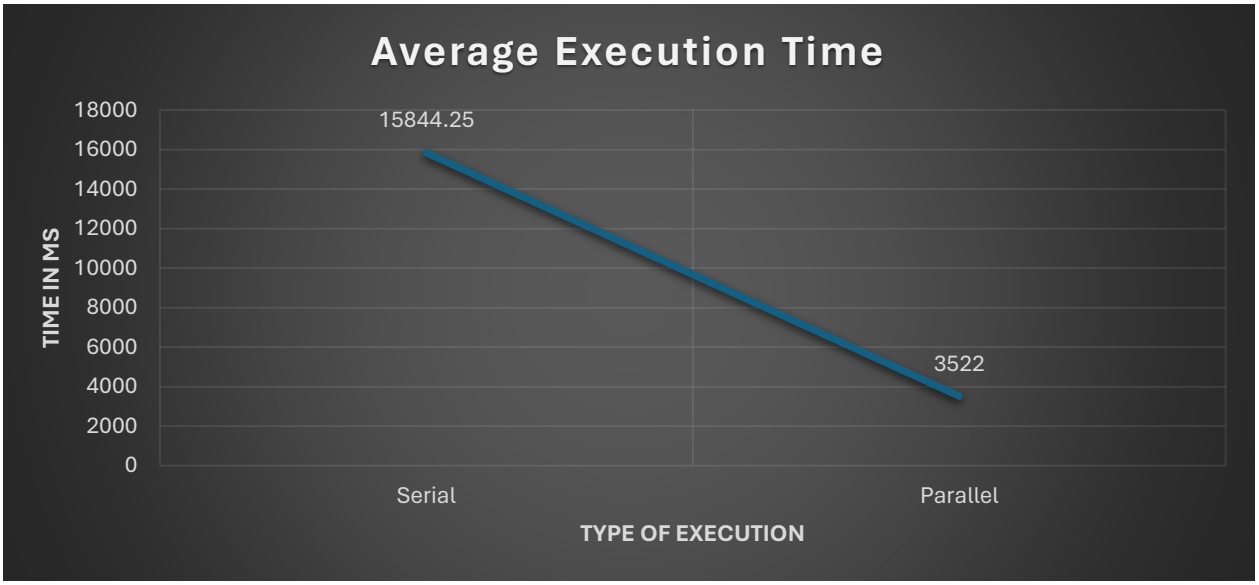
Τόσο στην σειριακή, όσο και στην παράλληλη έκδοση, χρονομετρήθηκε πριν την έναρξη της θόλωσης της εικόνας, μέχρι και το πέρας της εκτέλεσης της (πριν το γράψιμο στο τελικό αρχείο. Τόσο το πρόγραμμα της σειριακής εκτέλεσης, όσο και της παράλληλης εκτελέστηκε 4 φορές και πάρθηκε ο μέσος χρόνος των εκτελέσεων.

Serial	**Parallel**
16401	3463
15606	3535
15678	3595
15692	3495

Και οι μέσοι χρόνοι είναι οι εξής:

Serial	Parallel
15844.25	3522

Διαγραμματικά:



Σχόλια

Παρατηρούμε ότι ο μέσος συνολικός χρόνος από την σειριακή στην παράλληλη εκτέλεση γνώρισε σημαντική πτώση. Συγκεκριμένα, στην παράλληλη εκτέλεση έφτασε κοντά στο 1/5 της σειριακής εκτέλεσης. Αυτό είναι φυσιολογικό, καθώς τα νήματα αναλαμβάνουν σημαντικό και ισορροπημένο φόρτο εργασίας το καθένα. Επίσης η επιτάχυνση της εργασίας οφείλεται και στο ότι η βιβλιοθήκη OpenMP δεν προσδιορίζει πόσα νήματα θα χρησιμοποιηθούν στις οδηγίες της και ορίζει δυναμικά τον δυνατότερο αριθμό νημάτων που μπορεί να υποστηρίξει το σύστημα. Αυτό σε συνδυασμό με τον ορισμό των οδηγιών της OpenMP δείχνει τα οφέλη της βιβλιοθήκης στην τόσο στην μείωση της θόλωσης της εικόνας, αλλά και στην οργανωμένη σύνταξη του κώδικα.

Άσκηση 2

Το πρόβλημα

Μας ζητείται να δημιουργήσουμε μια συνάρτηση με όνομα bloom_parallel(), η οποία θα φορτώνει την εικόνα “street_night.jpg” και θα υλοποιεί το φίλτρο bloom, παραλληλοποιώντας τις εργασίες της με χρήση της OpenMP, μέσα σε ένα παράλληλο block της OpenMP (#pragma omp parallel).

Μέθοδος παραλληλοποίησης

Η bloom_parallel() χωρίζεται σε συγκεκριμένα σημεία, κρίσιμα για την παραλληλοποίηση του προγράμματος. Εισαγωγικά όμως, όλα τα στάδια εκτέλεσης εκτελούνται μέσα σε ένα παράλληλο block της βιβλιοθήκης OpenMP:

```
#pragma omp parallel
```

Με αυτόν τον τρόπο δημιουργούνται όλα τα νήματα που δύναται από τον μεταγλωττιστή.

Αρχικά, απαιτείται ο υπολογισμός της μέγιστης τιμής της φωτεινότητας “luminance” του κάθε pixel. Η φωτεινότητα υπολογίζεται από την μέση τιμή των 3 channels του pixel (red , green , blue). Για να παραλληλοποιήσουμε την εκτέλεση του κώδικα, ουσιαστικά χρησιμοποιούμε το μπλοκ της OpenMP:

```
#pragma omp for collapse(2) schedule(static)
```

Με αυτόν τον τρόπο οι δύο βρόχοι συμπτύσσονται σε έναν μόνο βρόχο για παράλληλη εκτέλεση με συνδυασμένο χώρο επανάληψης και ισοκατανομής φόρτου μεταξύ των νημάτων. Έτσι για κάθε pixel βρίσκουμε το luminance του:

```
// Calculate luminance for each pixel
#pragma omp for collapse(2) schedule(static)
for (int y = 0; y < height; y++) {
	for (int x = 0; x < width; x++) {
		int pixel = y * width + x;
		unsigned char red = img_in[4 * pixel + 0];
		unsigned char green = img_in[4 * pixel + 1];
		unsigned char blue = img_in[4 * pixel + 2];
		float luminance = (red + green + blue) / 3.0f;
		luminance_array[pixel] = luminance;
	}
}
```

Για τον υπολογισμό του maximum luminance θα χρησιμοποιήσουμε έναν συνδυασμό του #pragma omp for και #pragma omp reduction() ώστε να συνδυάσουμε τις τελικές τιμές της

μέγιστης φωτεινότητας κάθε παράλληλης εκτέλεσης σε μια τελική τιμή για να επιταχύνουμε τον υπολογισμό της:

```
// Find maximum luminance
#pragma omp for reduction(max:max_luminance)
for (int i = 0; i < width * height; i++) {
	if (luminance_array[i] > max_luminance) {
		max_luminance = luminance_array[i];
	}
}
```

Επίσης χωρίς την εισαγωγή του “nowait” έχουμε την ύπαρξη ενός barrier που μας εγγυάται ότι η εκτέλεση του προγράμματος θα συνεχιστεί μετά το πέρας όλων των νημάτων στο παραπάνω μπλοκ.

Έπειτα, κάθε νήμα εκτυπώνει την τιμή του maximum luminance, έχοντας εξασφαλίσει ότι θα είναι κοινή για όλα τα νήματα:

```
std::cout << "Max luminance: " << max_luminance << std::endl;
```

Στην περίπτωση μας η μέγιστη φωτεινότητα βγαίνει 191.

Στην συνέχεια δημιουργούμε μια νέα εικόνα (bloom_mask), ίδιου μεγέθους με την αρχική, στην οποία το κάθε pixel θα έχει το χρώμα του pixel της αρχικής εικόνας εφόσον η φωτεινότητα του αρχικού pixel είναι πάνω από το 90% της μέγιστης φωτεινότητας . Σε αντίθετη περίπτωση, το pixel θα πάρει το μαύρο χρώμα.

Για την παραλληλοποίηση της δημιουργίας της εικόνας ακολουθούμε την ίδια τακτική με την άσκηση 1 για τους ίδιους ακριβώς λόγους:

```
// Create bloom_mask
#pragma omp for collapse(3) schedule(static)
for (int y = 0; y < height; y++) {
	for (int x = 0; x < width; x++) {
		for (int channel = 0; channel < 4; channel++) {
			int pixel = (y * width + x);
			if (luminance_array[pixel] > 0.9 * max_luminance) {
				// Keep the original color because it is higher than
90%
				bloom_mask[pixel * 4 + channel] = img_in[pixel * 4 +
channel];
			}
			else {
				// Set to black -> 0
				bloom_mask[pixel * 4 + channel] = 0;
			}
		}
	}
}
```

Το επόμενο βήμα είναι θόλωση της εικόνας bloom_mask στον οριζόντιο άξονα και μετά στον κάθετο άξονα με την τεχνική του Gaussian Blur δύο περασμάτων. Πάλι θα χρησιμοποιήσουμε τις τεχνικές της άσκησης 1 με τον κώδικα να είναι ο εξής:

```
// Horizontal Blur
#pragma omp for collapse(3) schedule(static)
for (int y = 0; y < height; y++)
{
	for (int x = 0; x < width; x++)
	{
		for (int channel = 0; channel < 4; channel++)
		{
			int pixel = (y * width + x) * 4 + channel;
			bloom_mask[pixel] = blurAxis(x, y, channel, 0, bloom_mask,
width, height);
		}
	}
}

// Vertical Blur
#pragma omp for collapse(3) schedule(static)
for (int y = 0; y < height; y++)
{
	for (int x = 0; x < width; x++)
	{
		for (int channel = 0; channel < 4; channel++)
		{
			int pixel = (y * width + x) * 4 + channel;
			bloom_mask[pixel] = blurAxis(x, y, channel, 1, bloom_mask,
width, height);
		}
	}
}
```

Σε αυτό το σημείο θα χρειαστεί να γράψουμε την εικόνα blurred_image σε ένα αρχείο. ωστόσο αντί να χρησιμοποιήσουμε όλα τα νήματα θα χρησιμοποιήσουμε ένα νήμα, με την βοήθεια της οδηγίας #pragma omp single. Επίσης προσθέτουμε το “nowait” ώστε τα υπόλοιπα νήματα να συνεχίσουν την εκτέλεση, χωρίς να περιμένουν την εγγραφή της εικόνας.

```
// Write the blurred image to a file (only from one thread)
#pragma omp single nowait
{
	stbi_write_jpg("bloom_blurred.jpg", width, height, 4, bloom_mask, 90);
}
```

Η stbi_write_jpg αφού δεν αλλάζει την bloom_mask δεν θα δημιουργήσει κάποιο πρόβλημα στην εγγραφή της εικόνας στα επόμενα βήματα παρακάτω.

Για την τελική εικόνα, το κάθε pixel θα αποτελείται από την πρόσθεση των χρωμάτων των pixel της αρχικής και θολωμένης εικόνας. (Προσοχή : Η μέγιστη τιμή που μπορεί να έχει το κάθε pixel είναι το 255 διότι αποθηκεύεται σε μεταβλητή τύπου unsigned char). Σε αυτό το βήμα θα συμπτύξουμε τα τρία εμφωλευμένα for loops και θα ισομοιράσουμε το φορτίο εργασίας στα νήματα:

```
// Final Image Creation
#pragma omp for collapse(3) schedule(static)
for (int y = 0; y < height; y++) {
	for (int x = 0; x < width; x++) {
		for (int channel = 0; channel < 4; channel++) {
			int pixel = (y * width + x) * 4 + channel;
			// Combine colors of original and blurred images
			img_lum[pixel] = img_in[pixel] + bloom_mask[pixel];
			// Ensure maximum value does not exceed 255
			if (img_in[pixel] + bloom_mask[pixel] > 255) {
				img_lum[pixel] = 255;
			}
			else {
				img_lum[pixel] = img_in[pixel] + bloom_mask[pixel];
			}
		}
	}
}
```

Τέλος θα εγγράψουμε την εικόνα στην μνήμη πάλι με το ίδιο μοτίβο, καθώς έχουμε σιγουρευτεί με την ύπαρξη του προκαθορισμένου barrier παραπάνω ότι θα έχουν τελειώσει όλα τα νήματα:

```
// Write the final image to a file (only from one thread)
#pragma omp single nowait
{
	stbi_write_jpg("bloom_final.jpg", width, height, 4, img_lum, 90);
}
```

Σχόλια

Γίνεται αντιληπτό πως με την χρήση της OpenMP σε αυτήν την άσκηση καταφέραμε να απλοποιήσουμε την σύνταξη του κώδικα και συγχρόνως να εντάξουμε την έννοια της παραλληλοποίησης. Η τελική εικόνα του κώδικά μας είναι ίδια με την αρχική αλλά πιο φωτεινή σε ορισμένα σημεία.

** *Στα συμπιεσμένα αρχεία συμπεριλαμβάνεται το βιβλίο εργασίας excel που περιέχει τις μετρήσεις και για την 1η άσκηση, καθώς και screenshots της εκτέλεσης με το output των προγραμμάτων από το terminal της μηχανής στον φακελο resources. Η εκτέλεση έγινε σε περιβάλλον IDE: Microsoft Visual Studio και σε γλώσσα C++ στην έκδοση v20. Οι παραγόμενες εικόνες βρίσκονται στην ίδιο φάκελο με τον πηγαίο κώδικα.*