**Πανεπιστήμιο Ιωαννίνων**

**Πολυτεχνική Σχολή**

**Τμήμα Μηχανικών Η/Υ και Πληροφορικής**

**Προπτυχιακό Μάθημα:**

**«Παράλληλα Συστήματα και Προγραμματισμός»**

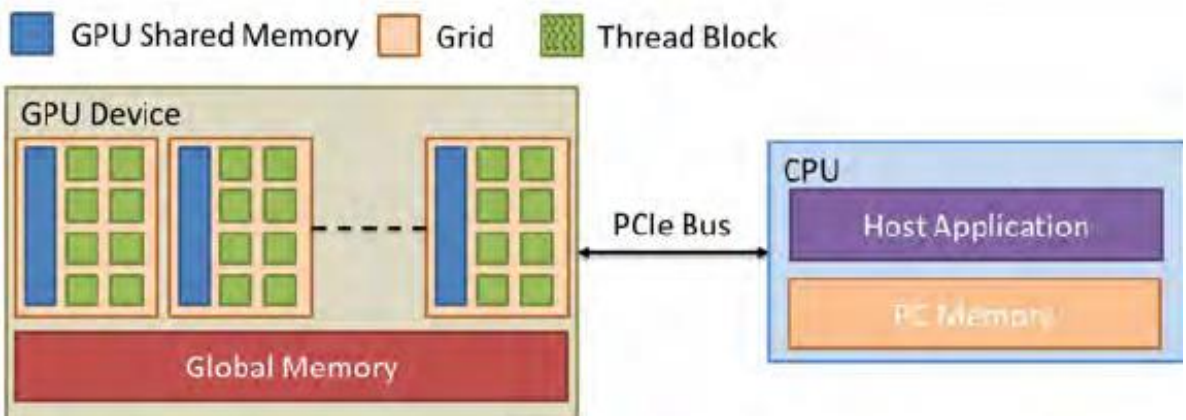
**Δεύτερο Σετ Προγραμματιστικών Ασκήσεων**

**Όνομα Φοιτητή – Α.Μ.:**

**Γεώργιος Κρομμύδας – 3260**

**E-mail Φοιτητή:**

**cs03260@uoi.gr**




*ΙΩΑΝΝΙΝΑ,*

*2022*

# Πίνακας περιεχομένων

**1.Εισαγωγή:** ..... 3

**2.Άσκηση – 1:** ..... 3

**2.1. Πληροφορίες Συστήματος:** ..... 3

**2.2. Ερώτημα:** ..... 4

**3. Άσκηση-2:** ..... 5

**3.1. Το πρόβλημα:** ..... 5

**3.2. Μέθοδοι Παραλληλοποίησης:** ..... 5

**3.3. Πειραματικά Αποτελέσματα – Μετρήσεις:** ..... 7

**3.4. Σχόλια:** ..... 17

**Βιβλιογραφία** ..... 19

# 1.Εισαγωγή:

Το 2ο σετ ασκήσεων αφορά τον παράλληλο προγραμματισμό με την χρήση της κάρτας γραφικών (***GPU***) και συγκεκριμένα το μοντέλο ***CUDA*** της ***NVIDIA***. Επίσης, γίνεται και χρήση του μοντέλου κοινόχρηστης μνήμης μέσω του προτύπου ***OpenMP***. Ζητείται η εύρεση πληροφοριών σχετικά με την κάρτα γραφικών που βρίσκεται στο σύστημα parallax του τμήματος και η παραλληλοποίηση της εφαρμογής θόλωσης εικόνων με χρήση του μοντέλου ***OpenMP*** και την εκτέλεσή του σε ***CUDA***.

Όλες οι μετρήσεις έγιναν στο παρακάτω σύστημα:

| **Όνομα Συστήματος** | Parallax |
|---|---|
| **Κάρτα Γραφικών** | GPU NVIDIA TESLA P40 |
| **Πλήθος Πυρήνων** | 3.840 |
| **Μεταγλωττιστές** | nvcc v.11.6 \| LLVM/Clang v.11.0 |

*Πίνακας 1: Λεπτομέρειες Συστήματος*

# 2.Άσκηση – 1:

## 2.1. Πληροφορίες Συστήματος:

Στην άσκηση αυτή ζητείται η εύρεση πληροφοριών της κάρτας γραφικών του συστήματος ***parallax*** μέσω του ***CUDA toolkit 11.6***, και μάλιστα με την χρήση της βιβλιοθήκης <*cuda_runtime.h*>. Χρησιμοποιήθηκε το πρόγραμμα από την σελίδα του μαθήματος (cuinfo.cu). Επεκτάθηκε το ***for loop*** της συνάρτησης ***cuinfo_print_devinfo()*** και προστέθηκε η εντολή ***cudaGetDeviceProperties(&dev_prop, i)***, έτσι ώστε να παρθούν οι λεπτομέρειες της κάρτας γραφικών. Εν συνεχεία τοποθετήθηκαν τα κατάλληλα ***printf(.)*** και εμφανίζονται οι επιθυμητές πληροφορίες οι οποίες είναι αποθηκευμένες στο ***struct cudaDeviceProp *dev_prop.***

Επομένως το αποτέλεσμα θα είναι το εξής που φαίνεται στην εικόνα 1.

```
---------------------------------------------------------
Information for CUDA Device - 1
---------------------------------------------------------
Device Name: Tesla P40
Device CUDA compute capability: 6.1
Device number of streaming multiprocessors: 30
Device max number of threads per block: 1024
Device size of global memory: 24032444416 bytes (= 24 Gbytes)
Device size of shared memory per block: 49152 bytes
---------------------------------------------------------
```

*Εικόνα 1: Πληροφορίες Κάρτας Γραφικών*

Για το όνομα χρησιμοποιήθηκε το πεδίο ***dev_prop.name***. Για την εύρεση του ***CUDA compute capability*** χρησιμοποιήθηκε το πεδίο ***dev_prop.major*** και ***dev_prop.minor***. Για την εύρεση του πλήθους των *SM* χρησιμοποιήθηκε το πεδίο ***dev_prop.multiProcessorCount***. Για την εύρεση του μέγιστου πλήθους νημάτων ανά μπλοκ χρησιμοποιήθηκε το πεδίο ***dev_prop.maxThreadsPerBlock***. Για την εύρεση της καθολικής μνήμης της συσκευής χρησιμοποιήθηκε το πεδίο ***dev_prop.totalGlobalMem*** και για την εύρεση της κοινόχρηστης μνήμης ανά μπλοκ χρησιμοποιήθηκε το πεδίο ***dev_prop.sharedMemPerBlock***.

### 2.2. Ερώτημα:

Σε μία ***NVIDIA GPU*** μπορούμε να υπολογίσουμε των πλήθος των cores που υπάρχουν συνολικά. Αυτό θα γίνει αρχικά με βάσει του αριθμού ***compute capability*** της κάρτας γραφικών και εν συνεχεία θα πολλαπλασιαστεί με το συνολικό πλήθος των ***streaming multiprocessors***. Στην συγκεκριμένη συσκευή γνωρίζουμε πως το ***compute capability*** είναι 6.1. Συνεπώς κάθε ***SM*** θα έχει 128 cores, καθώς το ***compute capability*** συσχετίζεται με το υλικό και την αρχιτεκτονική της κάρτας, με το συγκεκριμένο μοντέλο να έχει αυτό το πλήθος πυρήνων. Αυτό οφείλεται και στον minor αριθμό .1. Οπότε επί το συνολικό πλήθος των SM, η κάρτα γραφικών θα έχει συνολικά 3.840 cores.

# 3. Άσκηση-2:

## 3.1. Το πρόβλημα:

Σε αυτή την άσκηση ζητείται η παραλληλοποίηση του αλγορίθμου θόλωσης εικόνων με την μέθοδο του ***Gauss***. Αυτό θα γίνει με συνδυασμό της κάρτας γραφικών ***NVIDIA TESLA P40*** με το πρότυπο παραλληλοποίησης ***OpenMP***. Αυτό θα γίνει με το σωστό *offloading* του προγράμματος στην συσκευή και μετέπειτα την παραλληλοποίηση των βρόχων, έτσι ώστε να αξιοποιούνται όλοι οι πυρήνες κατά την εκτέλεση του προγράμματος.

## 3.2. 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, υλοποιήθηκε η συνάρτηση gaussian_blur_omp_device() η οποία χρησιμοποιεί το πρότυπο OpenMP σε συνδυασμό με την κάρτα γραφικών. Το πρώτο βήμα είναι να γίνει ***offloading*** (μεταφορά προγράμματος και δεδομένων) στην συσκευή(δηλαδή την κάρτα γραφικών). Αυτό θα γίνει με την εντολή πριν τον πρώτο βρόχο ***for*** η οποία θα είναι:

$$\#pragma\ omp\ target\ teams\ distribute\ parallel\ for\ collapse(2)\backslash$$

$$num\_teams(numOfTeams)\ num\_threads(numOfThreads)\ private(row, col)\backslash$$

$$firstprivate(redSum, greenSum, blueSum, weightSum)\backslash$$

$$map(to: imgin{->}red[0: height * width], imgin{->}green[0: height * width], imgin{->}blue[0: height * width])\backslash$$

$$map(tofrom: imgout{->}red[0: height * width], imgout{->}green[0: height * width], imgout{->}blue[0: height * width])$$

όπου η υπό-οδηγία ***target teams distribute*** αρχικά θα δημιουργήσει έναν αριθμό από ομάδες και θα διαμοιράσει τις επαναλήψεις στους αρχηγούς των εκάστοτε *CUDA blocks*. Η εντολή ***target*** κάνει το *offloading* στην συσκευή(GPU), η εντολή ***teams*** δημιουργεί τις ομάδες στο σύστημα και η εντολή ***distribute*** κατανέμει τον φόρτο

ισόποσα στους αρχηγούς των *block*. Εν συνεχεία, η υπο-οδηγία ***parallel for*** θα δημιουργήσει ένα πλήθος νημάτων ανά ομάδα και τα νήματα θα εκτελούν παράλληλα τις επαναλήψεις των αρχηγών. Αυτά τα νήματα των ομάδων στην ουσία θα είναι τα νήματα των *CUDA blocks*. Η υπο-οδηγία ***collapse(2)*** θα ενώσει τα δύο πρώτα ***for loop*** σε ένα και θα αυξήσει τον παραλληλισμό των νημάτων. Η υπο-οδηγία ***num_teams(numOfTeams)*** καθορίζει τον αριθμό των ομάδων στην gpu. Η υπο-οδηγία ***num_threads(numOfThreads)*** καθορίζει το πλήθος των νημάτων στις ομάδες και μάλιστα θα πρέπει να είναι πολλαπλάσιο του 32. Επίσης, ο συνδυασμός των δύο παραπάνω θα πρέπει να είναι τέτοιος ώστε να αξιοποιούνται όλοι οι πυρήνες της κάρτας γραφικών.

Οι μεταβλητές *row* και *col* θα πρέπει να είναι ιδιωτικές, διότι κάθε νήμα πρέπει να κάνει τους υπολογισμούς στο δικό του χώρο, δίχως να επηρεάζει τα υπόλοιπα. Οι μεταβλητές *redSum, greenSum, blueSum* και *weigthSum* πρέπει να είναι *firstprivate*, έτσι ώστε το κάθε νήμα να κάνει τους υπολογισμούς στο δικό του χώρο χωρίς να επηρεάζονται τα υπόλοιπα και μάλιστα θα πρέπει να παίρνουν τις αρχικές τιμές, με τις οποίες ορίστηκαν.

Εκτός από τις προηγούμενες μεταβλητές, χρειάζεται και η συσκευή τα δεδομένα που θα επεξεργαστεί. Τα δεδομένα στην προκειμένη περίπτωση είναι τα pixels της εικόνας, τα οποία αποτυπώνονται σε τρία κανάλια λόγω του χρώματος. Επομένως, θα χρησιμοποιήσουμε την εντολή ***map(to: list)*** όπως αναφέρεται παραπάνω στην εντολή για την είσοδο των δεδομένων στην συσκευή. Θα αντιγράψει όλα τα δεδομένα και από τα τρία κανάλια της εικόνας στην συσκευή. Αντίστοιχα, με την οδηγία ***map(tofrom: list)*** που αναφέρθηκε προηγουμένως, θα γίνει απλή δέσμευση μνήμης στην συσκευή για τις μεταβλητές εξόδου των καναλιών της εικόνας, όπου θα περιέχουν την θολωμένη εκδοχή. Τα δεδομένα αυτά θα αντικαταστήσουν τις τιμές τους στη CPU με το πέρας του target και το τέλος της παράλληλης περιοχής.

### 3.3. Πειραματικά Αποτελέσματα – Μετρήσεις:

Το πρόγραμμα εκτελέστηκε στο σύστημα που αναφέρθηκε στην εισαγωγή στον πίνακα 1 και η χρονομέτρηση έγινε με την συνάρτηση ***gettimeofday(struct timeval *, struct tzp*)***. Η διαδικασία της χρονομέτρησης γίνεται στην συνάρτηση ***timeit()*** που δέχεται ως όρισμα την συνάρτηση επεξεργασίας και τις αντίστοιχες παραμέτρους της, δηλαδή την ακτίνα θόλωσης και την εικόνα ως είσοδο μαζί με την θολωμένη εκδοχή της στην έξοδο.

Ο αριθμός των νημάτων που χρησιμοποιήθηκε ήταν πολλαπλάσιο του 32 και μάλιστα οι αριθμοί ήταν οι 128, 256, 512 και 1024, για να μπορέσουν να αξιοποιηθούν όλοι οι πυρήνες του συστήματος. Επίσης, ο αριθμός των ομάδων θα πρέπει να είναι πολλαπλάσιο του 30, διότι έτσι θα αξιοποιούνται και όλοι οι multiprocessors του συστήματος σε συνδυασμό με τους πυρήνες του κάθε multiprocessor που στην περίπτωση μας είναι 128. Επιπλέον, τα παρακάτω πειράματα έγιναν με ακτίνα θόλωσης $r = 8$ και με την εικόνα 1500.bmp.

Κάθε πείραμα εκτελέστηκε 4 φορές και υπολογίστηκαν οι μέσοι όροι. Τα αποτελέσματα δίνονται στους παρακάτω πίνακες (οι χρόνοι είναι σε sec).

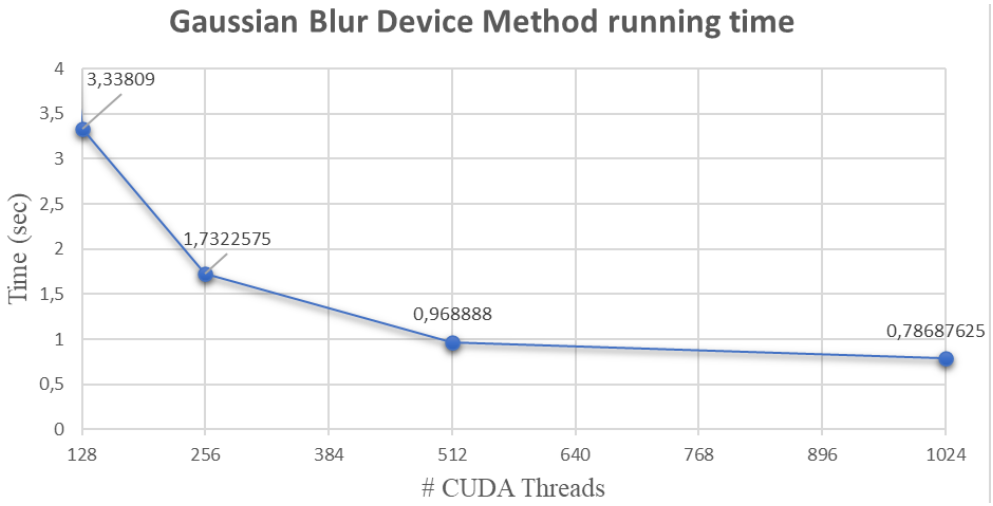
Για αριθμό ομάδων (*blocks*) 30 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 3,343657 | 3,355987 | 3,311677 | 3,341039 | 3,33809 |
| 256 | 1,736364 | 1,738567 | 1,710661 | 1,743438 | 1,7322575 |
| 512 | 0,969323 | 0,982476 | 0,975900 | 0,947853 | 0,968888 |
| 1024 | 0,788173 | 0,797850 | 0,793418 | 0,768064 | 0,78687625 |

*Πίνακας 2: Αποτελέσματα Gaussian Blur Device Method NumTeams(30)*

**Σειριακός Χρόνος Προγράμματος** = 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 2, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 1.




*Γράφημα 1: Γράφημα Μέσου Χρόνου Μεθόδου με 30 Ομάδες*

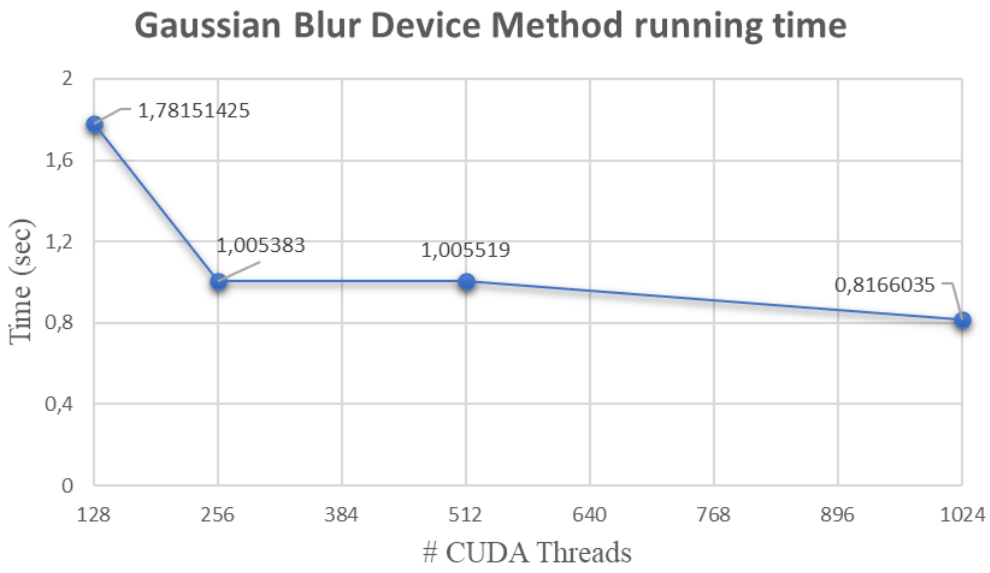
Για αριθμό ομάδων (*blocks*) 60 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 1,786164 | 1,761496 | 1,778251 | 1,800146 | 1,78151425 |
| 256 | 0,989033 | 1,021406 | 1,023242 | 0,987851 | 1,005383 |
| 512 | 0,997989 | 0,988556 | 1,013259 | 1,022272 | 1,005519 |
| 1024 | 0,827915 | 0,808147 | 0,821066 | 0,809286 | 0,8166035 |

*Πίνακας 3: Αποτελέσματα Gaussian Blur Device Method NumTeams(60)*

**Σειριακός Χρόνος Προγράμματος** = 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 3, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 2.




*Γράφημα 2: Γράφημα Μέσου Χρόνου Μεθόδου με 60 Ομάδες*

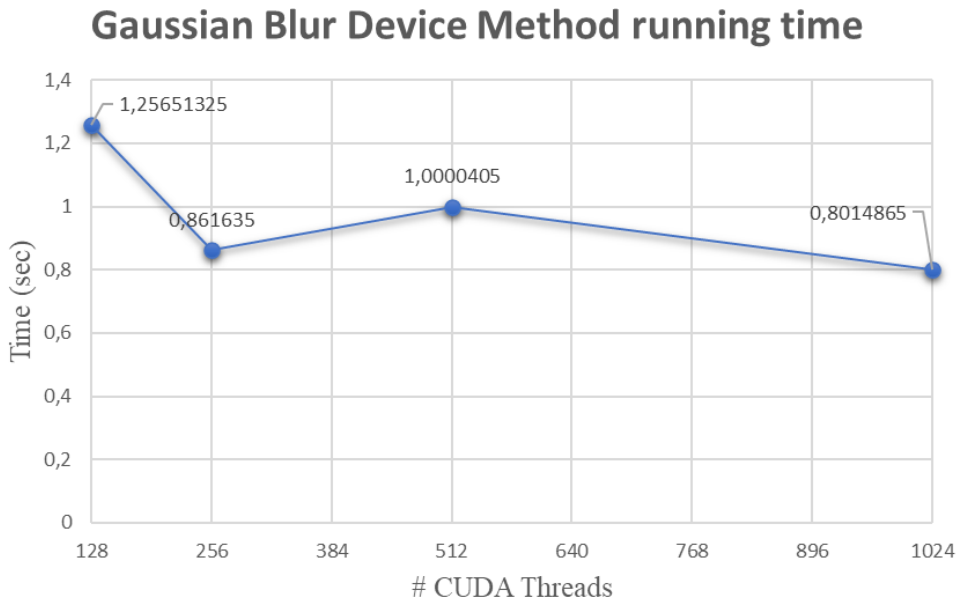
Για αριθμό ομάδων (*blocks*) 90 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 1,263242 | 1,260392 | 1,265933 | 1,236486 | 1,25651325 |
| 256 | 0,862357 | 0,877822 | 0,844290 | 0,862071 | 0,861635 |
| 512 | 0,991398 | 0,993914 | 1,005687 | 1,009163 | 1,0000405 |
| 1024 | 0,794757 | 0,806980 | 0,804145 | 0,800064 | 0,8014865 |

*Πίνακας 4: Αποτελέσματα Gaussian Blur Device Method NumTeams(90)*

**Σειριακός Χρόνος Προγράμματος =** 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 4, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 3.




*Γράφημα 3: Γράφημα Μέσου Χρόνου Μεθόδου με 90 Ομάδες*

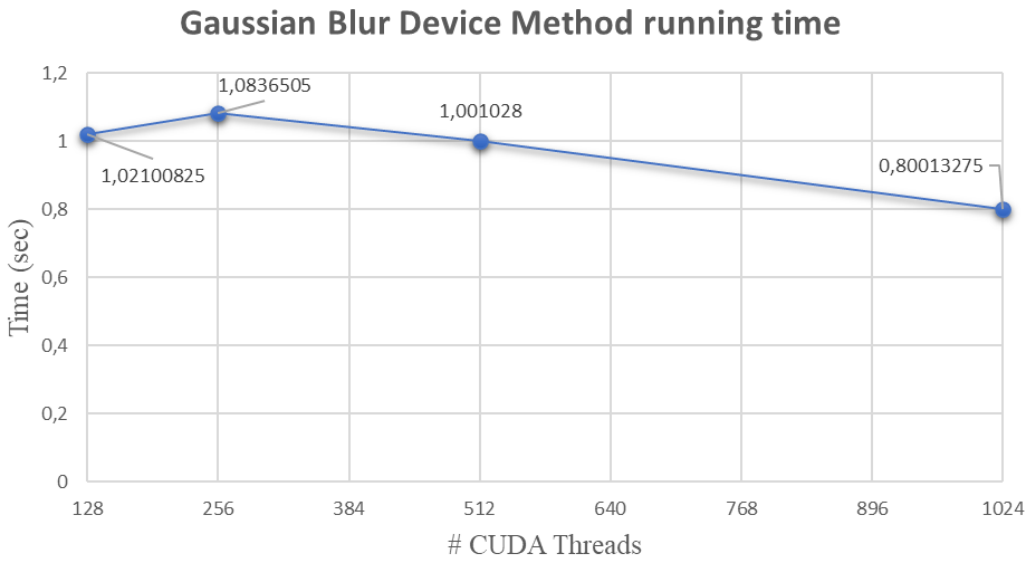
Για αριθμό ομάδων (*blocks*) 120 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 1,017831 | 1,016788 | 1,021954 | 1,02746 | 1,02100825 |
| 256 | 1,072432 | 1,084587 | 1,085767 | 1,091816 | 1,0836505 |
| 512 | 1,006943 | 1,014854 | 0,988405 | 0,993910 | 1,001028 |
| 1024 | 0,804650 | 0,788899 | 0,813222 | 0,793760 | 0,80013275 |

*Πίνακας 5: Αποτελέσματα Gaussian Blur Device Method NumTeams(120)*

**Σειριακός Χρόνος Προγράμματος =** 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 5, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 4.




*Γράφημα 4: Γράφημα Μέσου Χρόνου Μεθόδου με 120 Ομάδες*

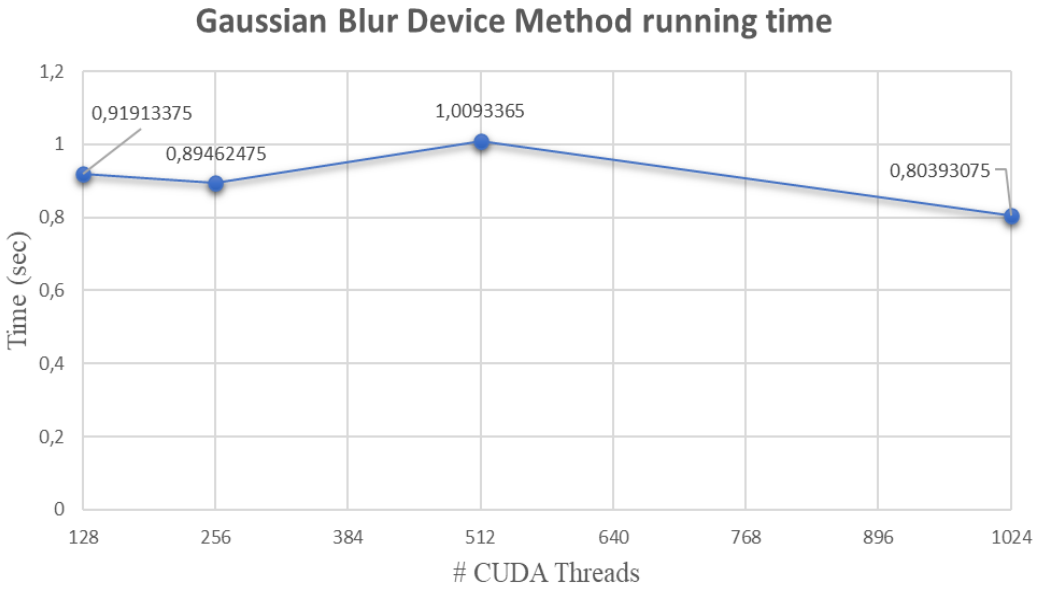
Για αριθμό ομάδων (*blocks*) 150 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 0,922292 | 0,923513 | 0,898636 | 0,932094 | 0,91913375 |
| 256 | 0,899653 | 0,892725 | 0,891547 | 0,894574 | 0,89462475 |
| 512 | 1,027436 | 1,018883 | 0,98774 | 1,003287 | 1,0093365 |
| 1024 | 0,794208 | 0,796683 | 0,820845 | 0,803987 | 0,80393075 |

*Πίνακας 6: Αποτελέσματα Gaussian Blur Device Method NumTeams(150)*

**Σειριακός Χρόνος Προγράμματος =** 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 6, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 5.




*Γράφημα 5: Γράφημα Μέσου Χρόνου Μεθόδου με 150 Ομάδες*

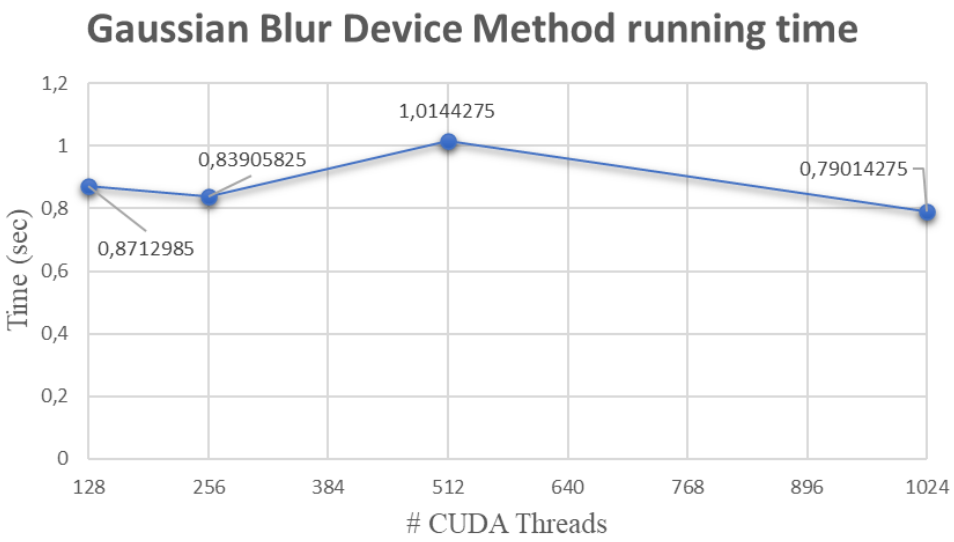
Για αριθμό ομάδων (*blocks*) 180 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 0,870364 | 0,876129 | 0,879278 | 0,859423 | 0,8712985 |
| 256 | 0,856164 | 0,848641 | 0,852797 | 0,798631 | 0,83905825 |
| 512 | 1,025626 | 1,022661 | 0,995363 | 1,014060 | 1,0144275 |
| 1024 | 0,819724 | 0,793623 | 0,791934 | 0,755290 | 0,79014275 |

*Πίνακας 7: Αποτελέσματα Gaussian Blur Device Method NumTeams(180)*

**Σειριακός Χρόνος Προγράμματος =** 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 7, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 6.




*Γράφημα 6: Γράφημα Μέσου Χρόνου Μεθόδου με 180 Ομάδες*

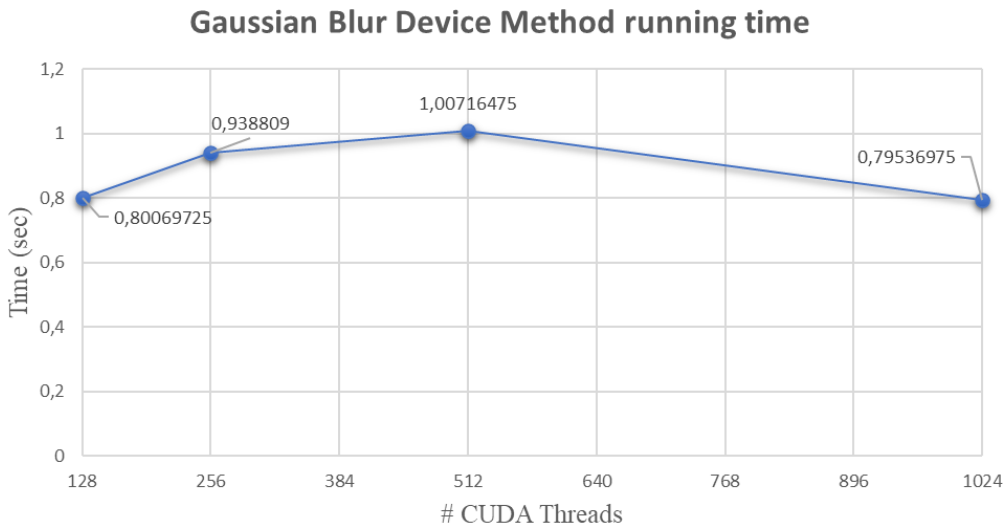
Για αριθμό ομάδων (*blocks*) 210 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 0,797447 | 0,812114 | 0,786870 | 0,806358 | 0,80069725 |
| 256 | 0,960521 | 0,958287 | 0,852797 | 0,983631 | 0,938809 |
| 512 | 1,018341 | 0,993481 | 1,013705 | 1,003132 | 1,00716475 |
| 1024 | 0,801173 | 0,814548 | 0,754613 | 0,811145 | 0,79536975 |

*Πίνακας 8: Αποτελέσματα Gaussian Blur Device Method NumTeams(210)*

**Σειριακός Χρόνος Προγράμματος =** 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 8, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 7.




*Γράφημα 7: Γράφημα Μέσου Χρόνου Μεθόδου με 210 Ομάδες*

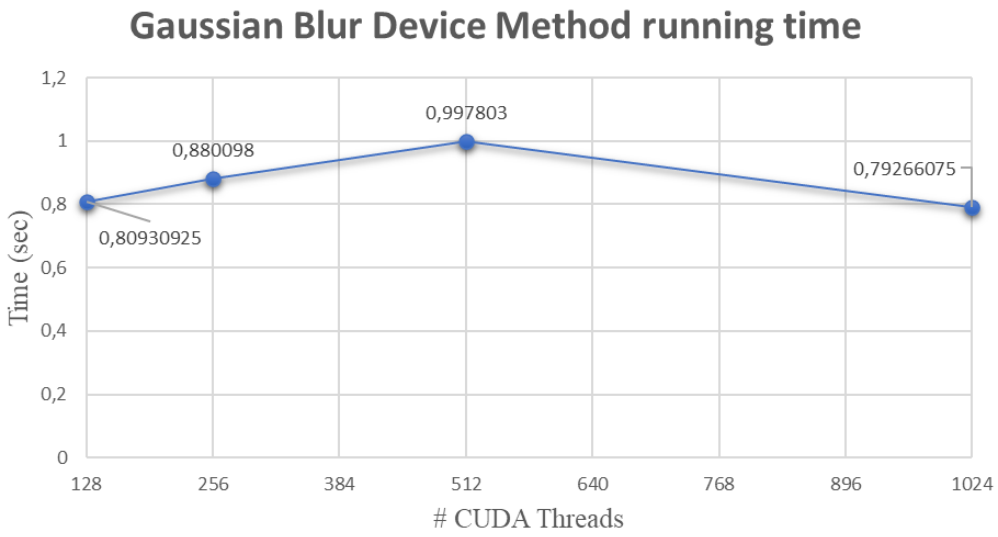
Για αριθμό ομάδων (*blocks*) 240 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 0,809484 | 0,814508 | 0,809146 | 0,804099 | 0,80930925 |
| 256 | 0,867786 | 0,877116 | 0,888373 | 0,887117 | 0,880098 |
| 512 | 1,019177 | 0,945067 | 1,029521 | 0,997447 | 0,997803 |
| 1024 | 0,751694 | 0,806120 | 0,818881 | 0,793948 | 0,79266075 |

*Πίνακας 9: Αποτελέσματα Gaussian Blur Device Method NumTeams(240)*

**Σειριακός Χρόνος Προγράμματος =** 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 9, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 8.




*Γράφημα 8: Γράφημα Μέσου Χρόνου Μεθόδου με 240 Ομάδες*

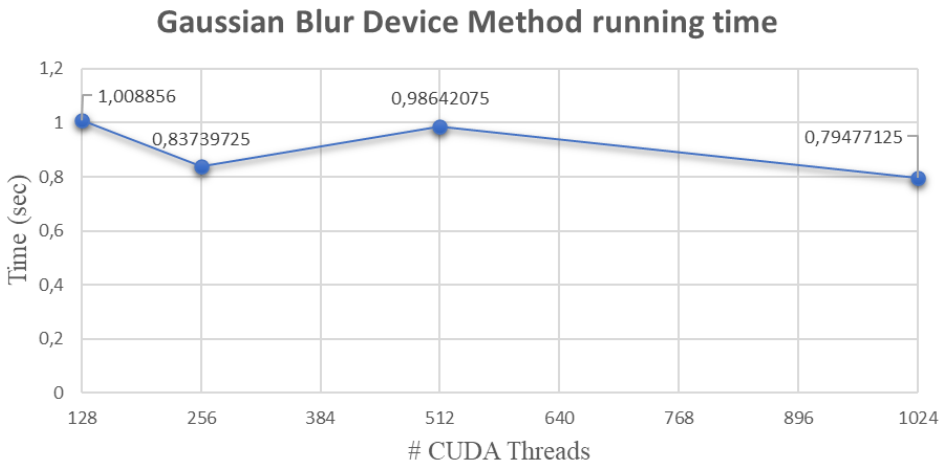
Για αριθμό ομάδων (*blocks*) 270 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 0,969543 | 1,013652 | 1,02981 | 1,022419 | 1,008856 |
| 256 | 0,859618 | 0,799587 | 0,840298 | 0,850086 | 0,83739725 |
| 512 | 0,943463 | 0,98433 | 1,009366 | 1,008524 | 0,98642075 |
| 1024 | 0,829043 | 0,749942 | 0,799623 | 0,800477 | 0,79477125 |

*Πίνακας 10: Αποτελέσματα Gaussian Blur Device Method NumTeams(270)*

**Σειριακός Χρόνος Προγράμματος =** 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 10, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 9.




*Γράφημα 9: Γράφημα Μέσου Χρόνου Μεθόδου με 270 Ομάδες*

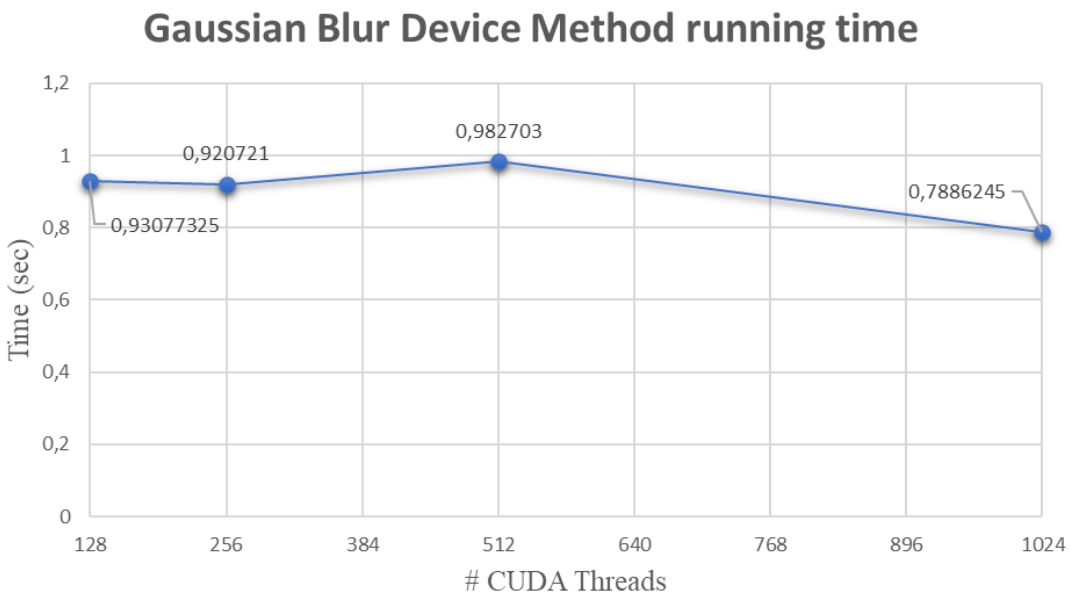
Για αριθμό ομάδων (*blocks*) 270 προκύπτει ο εξής πίνακας:

| **Νήματα CUDA** | **1η Εκτέλεση** | **2η Εκτέλεση** | **3η Εκτέλεση** | **4η Εκτέλεση** | **Μέσος Χρόνος** |
|---|---|---|---|---|---|
| 128 | 0,937297 | 0,890993 | 0,955592 | 0,939211 | 0,93077325 |
| 256 | 0,931795 | 0,950400 | 0,925642 | 0,875047 | 0,920721 |
| 512 | 0,992277 | 0,986719 | 1,00733 | 0,944486 | 0,982703 |
| 1024 | 0,824067 | 0,796183 | 0,786156 | 0,748092 | 0,7886245 |

*Πίνακας 11: Αποτελέσματα Gaussian Blur Device Method NumTeams(300)*

**Σειριακός Χρόνος Προγράμματος** = 18.963817 sec

Με βάσει τις παραπάνω μετρήσεις του πίνακα 11, προκύπτει το εξής γράφημα μέσου χρόνου στο γράφημα 10.




*Γράφημα 10: Γράφημα Μέσου Χρόνου Μεθόδου με 300 Ομάδες*

### 3.4. Σχόλια:

Με βάση τα παραπάνω αποτελέσματα, παρατηρούμε πως ο κατάλληλος συνδυασμός των αριθμών ομάδων και νημάτων μπορούν να μειώσουν σημαντικά το χρόνο εκτέλεσης της εφαρμογής. Πολύ σημαντικός παράγοντας στην επίδοση αποτελεί και το γεγονός πως αξιοποιούνται όλοι οι streaming multiprocessors σε συνδυασμό με όλους τους πυρήνες. Αυτό συμβαίνει διότι το πλήθος των ομάδων έχει τέτοιο νούμερο ώστε να αποτελεί πολλαπλάσιο των 30 streaming multiprocessors για να αξιοποιούνται όλοι κατά την εκτέλεση της εφαρμογής. Επίσης, στην απόδοση βοηθάει και ο αριθμός των νημάτων που χρησιμοποιούνται καθώς είναι πολλαπλάσια του 32 με το μέγιστο πλήθος να είναι τι 1024, το οποίο βάσει αρχιτεκτονικής της GPU χρησιμοποιεί όλους τους διαθέσιμους πυρήνες της συσκευής.

Επιπλέον, με βάση τις γραφικές παραστάσεις της προηγούμενης ενότητας παρατηρούμε πως κατά την αύξηση των ομάδων ξεκινάει να μειώνεται ο χρόνος μέχρι συγκεκριμένες τιμές όσο και να αυξηθούν οι παράμετροι. Αυτό είναι αναμενόμενο διότι αξιοποιείται όλη η συσκευή και επιφέρει ταχύτερα αποτελέσματα.

Σύμφωνα με τα παραπάνω αποτελέσματα, ο καλύτερος συνδυασμός των δύο παραμέτρων είναι να έχουμε αριθμό ομάδων 30 και αριθμό νημάτων 1024. Καθώς η συσκευή μας είχε 30 streaming multiprocessors, τότε ανατίθεται από ένα block στον

καθένα. Εν συνεχεία, κάθε block θα περιέχει 1024 νήματα τα οποία θα εκτελούν τις εκάστοτε διεργασίες.

Αυτό το αποτέλεσμα είναι αναμενόμενο να συμβεί, διότι οι επαναλήψεις των βρόχων μοιράζονται ισόποσα από τους αρχηγούς νήματα των εκάστοτε ομάδων σε όλα τα υπόλοιπα νήματα του κάθε μπλοκ και το καθένα έχει παρόμοιο φόρτο υπολογισμών. Επομένως, η αύξηση του αριθμού των ομάδων μειώνει ραγδαία τον χρόνο εκτέλεσης σε μία GPU, συγκριτικά με την ίδια μέθοδο εκτέλεσης σε μία CPU. Μάλιστα συγκριτικά με τα αποτελέσματα που είχε η μέθοδος με τα ***for loops*** και με τα ***tasks*** στην CPU, διαπιστώνεται πως είναι προτιμότερο η χρήση της συσκευής για να παρθούν γρήγορα τα αποτελέσματα σε μία κρίσιμη εφαρμογή.

# Βιβλιογραφία

[1] J. Cheng, M. Grossman, Ty McKercher, *Professional CUDA C Programming*, John Wiley & Sons Publishing, 2014.

[2] S. Cook, *CUDA Programming A Developer's Guide To Parallel Computing With GPUs*, Morgan Kaufmann Publishing, 2012.

[3] *NVIDIA CUDA Toolkit*, 11.6 Release Notes, January 2022.

[4] *Clang Compiler*, 11.0 Release Notes, October 2020.

[5] D. Storti, M. Yurtoglu, *CUDA for Engineers: An Introduction to High-Performance Parallel Computing*, Addison-Wesley, 2015.

[6] *CUDA C PROGRAMMING GUIDE*, October 2012.