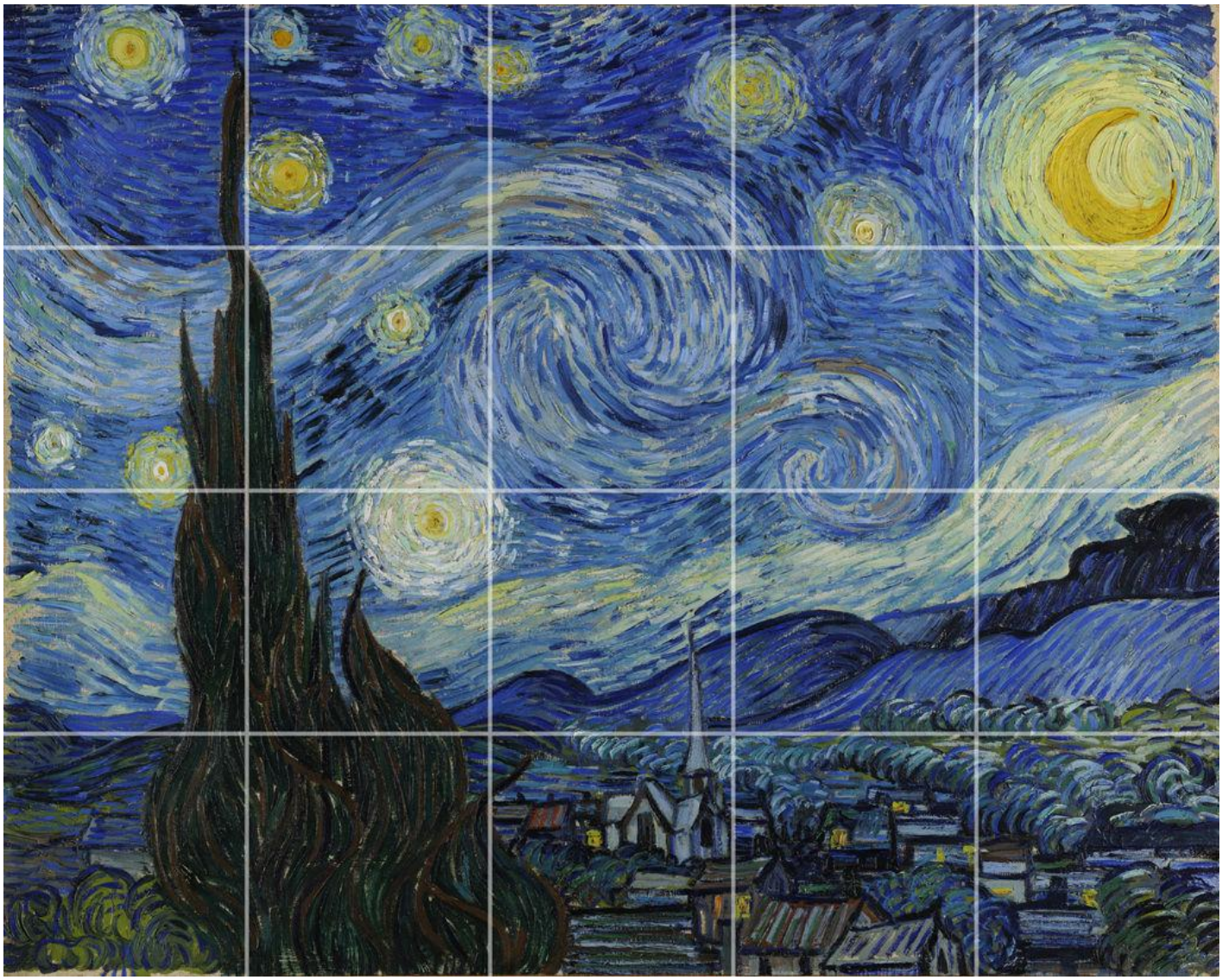
ΣΧΕΔΙΑΣΜΟΣ, ΑΝΑΠΤΥΞΗ ΚΑΙ ΑΠΟΤΙΜΗΣΗ ΠΑΡΑΛΛΗΛΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ ΓΙΑ ΣΥΝΕΛΙΞΗ ΔΙΣΔΙΑΣΤΑΤΩΝ ΕΙΚΟΝΩΝ

ΜΑΘΗΜΑ: *Παράλληλα Συστήματα* **ΕΤΟΣ:** *2015*

ΔΙΔΑΣΚΩΝ: *Ιωάννης Κοτρώνης*



ΟΝΟΜΑ: *Ορέστης Μελκονιάν* **ΑΜ:** *1115201000128*

Εισαγωγή

Στόχος

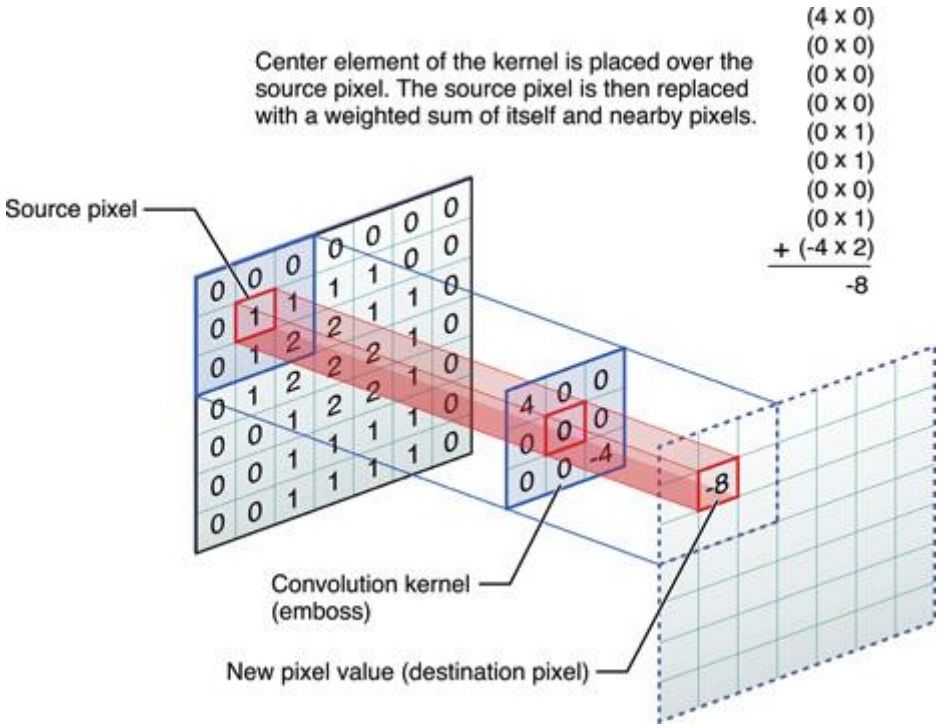
Αντικείμενο της παρούσας εργασίας είναι η συγγραφή παράλληλων προγραμμάτων που υλοποιούν τη συνέλιξη μιας εικόνας με ένα φίλτρο 3x3 με τρεις διαφορετικές τεχνολογίες-βιβλιοθήκες:

1. MPI
2. MPI + OpenMP
3. CUDA

Συνέλιξη

Η συνέλιξη αποτελεί εφαρμογή ενός φίλτρου πάνω σε κάθε εικονοστοιχείο(pixel) μιας δυσδιάστατης εικόνας, η οποία έχει ως αποτέλεσμα μία δεύτερη αλλοιωμένη εικόνα.

Η εφαρμογή του φίλτρου πάνω σε ένα pixel είναι η αντικατάσταση του συγκεκριμένου pixel με το σταθμισμένο άθροισμα αυτού και των 8 γειτόνων του, σύμφωνα με τους συντελεστές που ορίζονται από το φίλτρο.



Δομή Κώδικα

File-system

/MPI/: Υλοποίηση σε σκέτο MPI (1 process/machine)

/MPI_OpenMP/: Προσθήκη νημάτων σε κάθε διεργασία του MPI

/CUDA/: Υλοποίηση σε CUDA

/images/: Οι εικόνες όλων των διαστάσεων

MPI

Η βιβλιοθήκη MPI (Message Passing Interface) μας προσφέρει τη δυνατότητα συγγραφής προγραμμάτων που τρέχουν παράλληλα σε πολλά μηχανήματα. Η επικοινωνία μεταξύ των διαφορετικών διεργασιών γίνεται εξ' ολοκλήρου μέσω μηνυμάτων.

Στη συγκεκριμένη εφαρμογή, ανατίθεται σε κάθε διεργασία ένα τμήμα της αρχικής εικόνας. Έπειτα, κάθε διεργασία αναλαμβάνει την συνέλιξη του αντίστοιχου τμήματος. Προφανώς, η επικοινωνία είναι απαραίτητη μόνο στα οριακά pixel κάθε τμήματος.

Για να υλοποιηθεί αποδοτικά και κομψά η επικοινωνία, ορίστηκε μια καρτεσιανή τοπολογία των διεργασιών. Η τοπολογία αυτή ταιριάζει με το πρόβλημα, διότι κάθε επικοινωνία χρειάζεται να επικοινωνήσει μόνο με τους γείτονες του στη καρτεσιανή τοπολογία. Επίσης, δίνουμε στη βιβλιοθήκη δυνατότητα να κάνει reorder τις διεργασίες, ώστε να βελτιστοποιήσει όσο περισσότερο μπορεί την ανάθεση διεργασιών σε φυσικούς επεξεργαστές.

```
/* Cartesian topology setup */
MPI_Comm old_comm, new_comm;
old_comm = MPI_COMM_WORLD;
int dim = sqrt(comm_size);
int dim_size[2] = {dim, dim}, periods[2] = {1, 1};
MPI_Cart_create(old_comm, 2, dim_size, periods, 1, &new_comm);
```

Για την αποφυγή χρήσης buffers και γενικότερη κομψότητα χρησιμοποιήθηκαν MPI_Datatypes. Συγκεκριμένα, για την αποστολή γραμμών της εικόνας ορίστηκε ο τύπος *Row*, για την αποστολή στηλών της εικόνας ορίστηκε ο τύπος *Column* και για την διάσπαση της εικόνας σε block ορίστηκε ο τύπος *InnerArray*.

```
/* Derived Datatypes */
// ROW
MPI_Datatype Row;
MPI_Type_vector(subWidth, bpp, bpp, MPI_UNSIGNED_CHAR, &Row);
MPI_Type_commit(&Row);
// COLUMN
MPI_Datatype Column;
MPI_Type_vector(subHeight, bpp, bpp*(subWidth+2), MPI_UNSIGNED_CHAR,
&Column);
MPI_Type_commit(&Column);
// INNER_ARRAY
MPI_Datatype InnerArray;
int arrayOfSizes[2] = {args.height, args.width * bpp};
int arrayOfSubsizes[2] = {subHeight, subWidth * bpp};
int arrayOfStarts[2] = {me[0] * subHeight, me[1] * subWidth * bpp};

MPI_Type_create_subarray(2, arrayOfSizes, arrayOfSubsizes, arrayOfStarts,
MPI_ORDER_C,    MPI_UNSIGNED_CHAR, &InnerArray);
MPI_Type_commit(&InnerArray);
```

Η επικοινωνία γίνεται ασύγχρονα μέσω των συναρτήσεων *Isend* και *Irecv*, διότι ενόσω μια διεργασία περιμένει μηνύματα για να μπορεί να εφαρμόσει τη συνέλιξη στα οριακά pixel, μπορεί να αξιοποιήσει το χρόνο για να υπολογίσει την συνέλιξη των εσωτερικών pixel.

```
/* Receive & Send messages */
for (int i = 0; i < 8; i++) {
        MPI_Start(&sendReq[i][index]);
        MPI_Start(&recvReq[i][index]);
}

/* Convolute inner pixels */
for (int i = 2; i < h; i++)
        for (int j = 2; j < w; j++)
                convolute(image, buffer, i, j, w+2, bpp);

/* Wait on receive requests */
for (int i = 0; i < 8; i++)
        MPI_Wait(&recvReq[i][index], MPI_STATUS_IGNORE);
```

Για το έλεγχο σύγκλισης χρησιμοποιήθηκε η συνάρτηση *MPI_Allreduce.*

```
/* Check if we reached fixpoint (only every CHECK loops) */
if (loop % CHECK == 0) {
        int myFinished = 1;
        for (int i = 1; i < subHeight + 1; i++)
        for (int j = 1; j < subWidth + 1; j++)
        if (*indexAt(image,subWidth+2,bpp,i,j) != *indexAt(buffer,subWidth+2,bpp,i,j))  {
                        myFinished = 0;
                        break;
        }
        MPI_Allreduce(&myFinished, &allFinished, 1, MPI_INT, MPI_MIN, new_comm);
}
```

MPI + OpenMP

Σε αυτή την υλοποίηση ξεκινάμε μόνο 1 διεργασία σε κάθε μηχάνημα, αλλά κάθε διεργασία χωρίζεται σε νήματα που εκτελούνται παράλληλα μέσω της βιβλιοθήκης OpenMP.

Το OpenMP μας παρέχει ένα πολύ αφαιρετικό τρόπο να μετατρέψουμε ένα σειριακό πρόγραμμα σε παράλληλο μέσω των preprocessor directives (#pragma). Μετά από μετρήσεις χρόνου εκτέλεσης διαπίστωσα ότι το μόνο μέρος που αξίζει να εκτελεστεί παράλληλα με threads είναι η εφαρμογή της συνέλιξης στα εσωτερικά pixel σε κάθε επανάληψη. Σε κάθε άλλη περίπτωση, ο χρόνος που κερδίζουμε στον υπολογισμό είναι είτε αμελητέος, είτε μικρότερος από το overhead που δημιουργείται για δημιουργία, εναλλαγή και καταστροφή των νημάτων.

```
/* Convolute inner pixels */
#pragma omp parallel for shared(buffer) collapse(2) schedule(static)
for (int i = 2; i < h; i++)
        for (int j = 2; j < w; j++)
                convolute(image, buffer, i, j, w+2, bpp);
```

Για παράδειγμα, η παραλληλοποίηση της συνέλιξης για κάθε χρώμα(στην περίπτωση RGB εικόνας) αποδείχθηκε μοιραία για τον χρόνο εκτέλεσης.

```
/* Calculates the convolution of two 2D matrices for given position: (row, column) */
void convolute(unsigned char *image, unsigned char *buffer, int row, int col, int width, int bpp) {
        #pragma omp parallel for num_threads(bpp) shared(buffer) schedule(static)
        for (int offset = 0; offset < bpp; offset++) {
                ……
        }
……
```

CUDA

Η υλοποίηση σε CUDA είναι πολύ πιο απλοϊκή διότι δεν πραγματοποιείται κάποια επικοινωνία μεταξύ των threads. Απλώς, ανατίθεται σε κάθε thread ένα pixel και καθένα από αυτά, έχοντας πρόσβαση στην global memory που έχουμε αποθηκεύσει την αρχική εικόνα, υπολογίζει την επόμενη τιμή του pixel. Στην περίπτωση πολλαπλών blocks, τα οριακά pixel μένουν αμετάβλητα. Επίσης, δεν έχει υλοποιηθεί έλεγχος σύγκλισης όπως στις προηγούμενες δύο υλοποιήσεις.

```
/* Main Loop */
for (int loop = 0; loop < loops; loop++) {
        // Convolution on GPU
        convolute<<<grid, block>>>(image, buffer, sum, height, width, bpp);
        CUDA_SAFE_CALL(cudaGetLastError());
        // Synchronize threads
        CUDA_SAFE_CALL(cudaThreadSynchronize());
        // Swap buffers
        unsigned char *temp = image;
        image = buffer;
        buffer = temp;
}
```

Μετρήσεις και Επιδόσεις

Περιβάλλον

Για τις υλοποιήσεις σε MPI, οι μετρήσεις πραγματοποιήθηκαν στα μηχανήματα linux της σχολής . Χρησιμοποιήθηκαν οι συναρτήσεις *MPI_Wtime* και *MPI_Reduce*.

Για την υλοποίηση σε CUDA, λόγω αδυναμίας σύνδεσης στον server που αναφέρεται στο φυλλάδιο του αντίστοιχου εργαστηρίου, οι μετρήσεις πραγματοποιήθηκαν στον προσωπικό μου υπολογιστή που έχει κάρτα γραφικών την GeForce GTX-260 με 192 cuda cores και 896MB μνήμης.

Οι μετρήσεις έγιναν αυτόματα με bash scripts, που βρίσκονται μέσα στον φάκελο της εκάστοτε υλοποίησης που μετρήθηκε.

Πέρα από τις αρχικές εικόνες που μας δόθηκαν, δημιουργήθηκαν και εικόνες με πολλαπλάσιες διαστάσεις της αρχικής [x(1/4), x(1/2), x(2), x(4)].

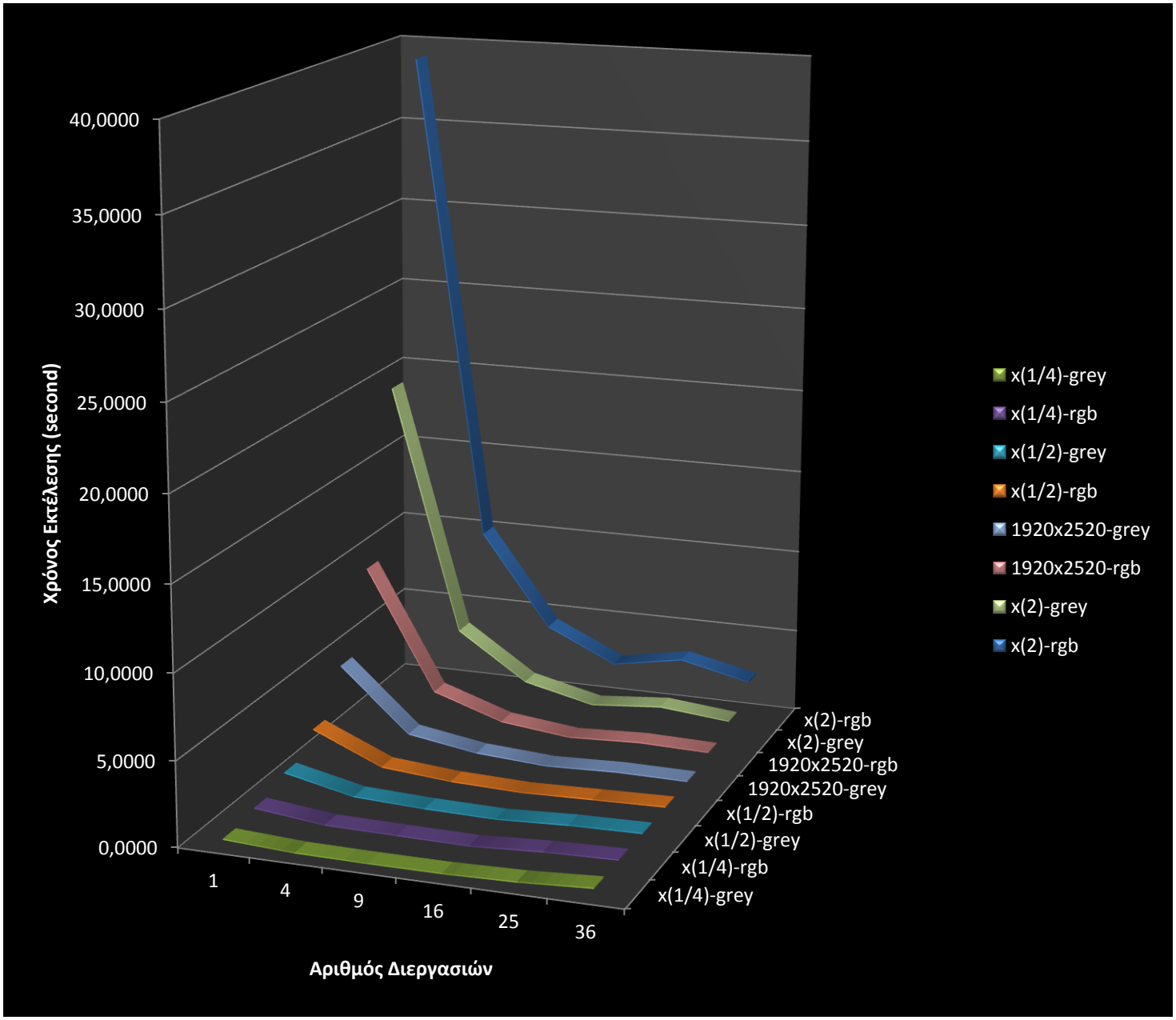
Οι μετρικές που θα παρουσιάσω είναι οι εξής:

- **Συνολικός χρόνος εκτέλεσης**
  Ο χρόνος καθαρού υπολογισμού της συνέλιξης, παραβλέποντας το χρόνο setup του συστήματος, κλπ.
- **Speedup**
  $$Speedup = \frac{T_{serial}}{T_{parallel}}$$
- **Efficiency**
  $$Efficiency = \frac{Speedup}{\#processes}$$

MPI

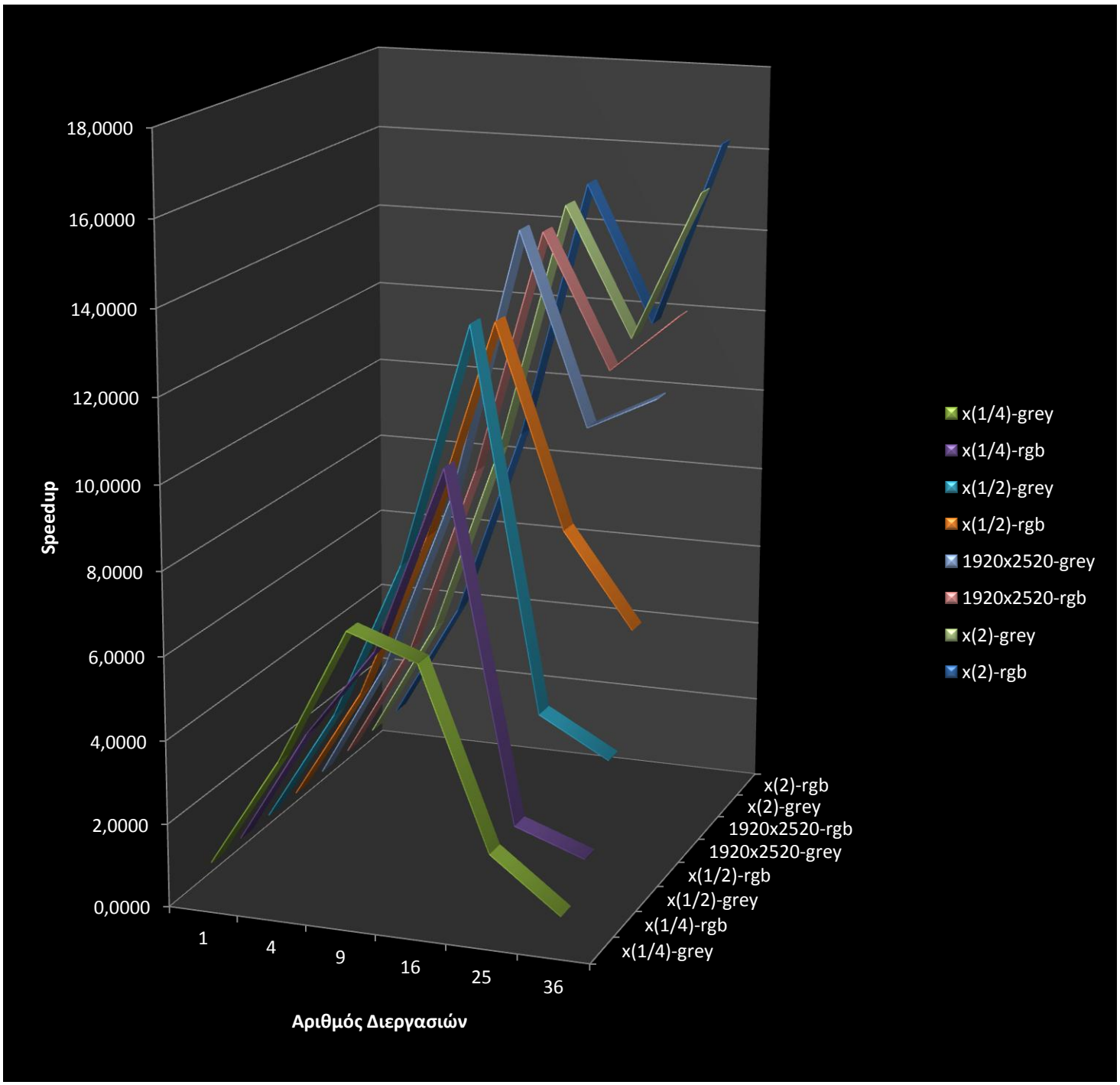
Συνολικός Χρόνος Εκτέλεσης(sec)

Εικόνες / Διεργασίες	1	4	9	16	25	36
x(1/4)-grey	0,3129	0,0862	0,0454	0,0495	0,1518	0,3891
x(1/4)-rgb	0,6167	0,1641	0,1042	0,0595	0,2871	0,3951
x(1/2)-grey	1,2477	0,3381	0,1664	0,0939	0,2916	0,3711
x(1/2)-rgb	2,4471	0,6605	0,3162	0,1882	0,2981	0,4091
1920x2520-grey	4,9913	1,2630	0,6031	0,3364	0,4863	0,4500
1920x2520-rgb	9,7591	2,5315	1,1467	0,6733	0,8659	0,7680
x(2)-grey	19,8390	5,0754	2,3033	1,3367	1,6957	1,2921
x(2)-rgb	39,0645	10,0499	4,5149	2,5896	3,3323	2,4021
x(4)-grey	Quota Exceeded					
x(4)-rgb						



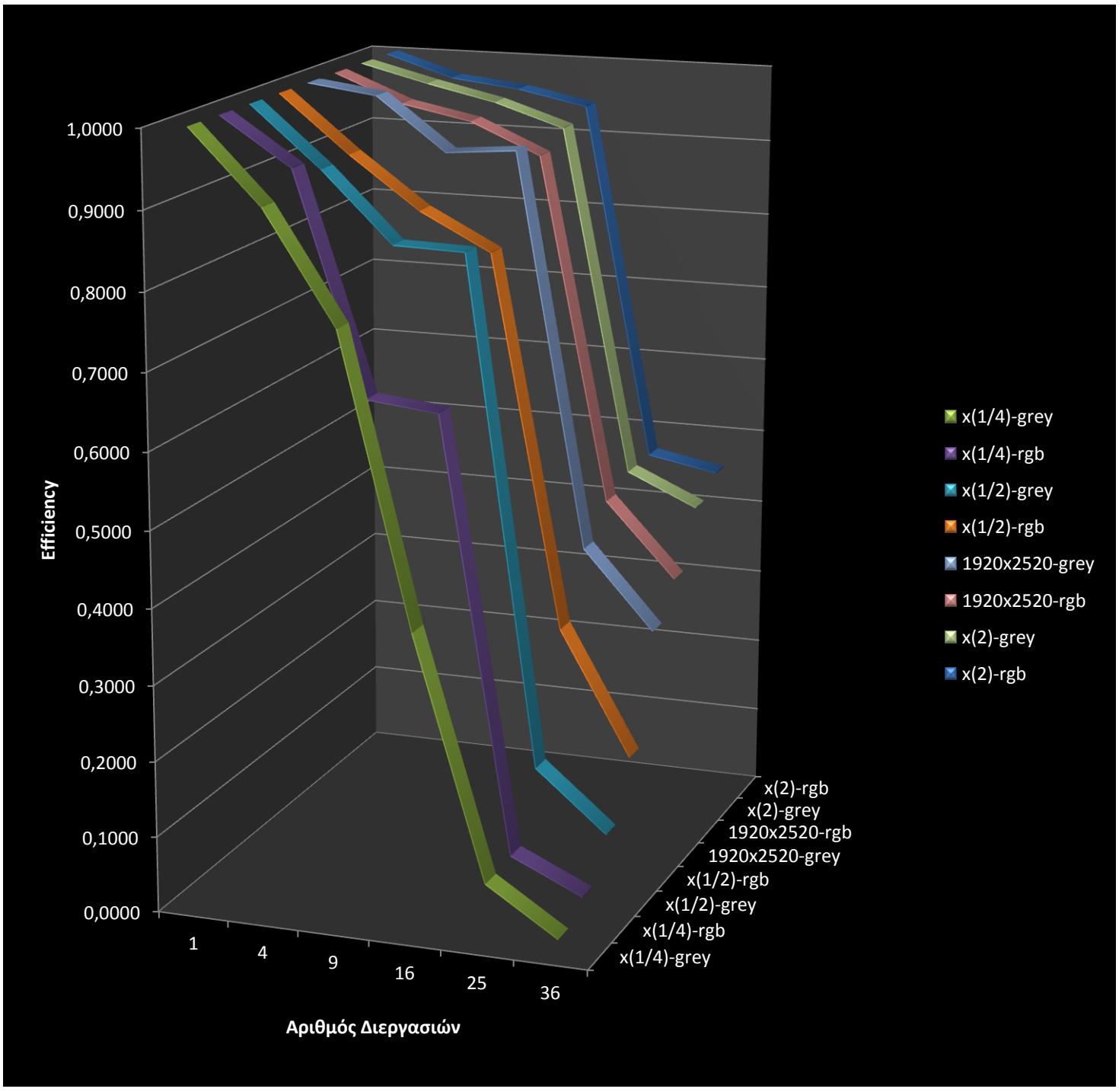
Speedup

Εικόνες / Διεργασίες	1	4	9	16	25	36
x(1/4)-grey	1,0000	3,6299	6,8921	6,3212	2,0613	0,8042
x(1/4)-rgb	1,0000	3,7581	5,9184	10,3647	2,1480	1,5609
x(1/2)-grey	1,0000	3,6903	7,4982	13,2875	4,2788	3,3622
x(1/2)-rgb	1,0000	3,7049	7,7391	13,0027	8,2090	5,9817
1920x2520-grey	1,0000	3,9519	8,2761	14,8374	10,2638	11,0918
1920x2520-rgb	1,0000	3,8551	8,5106	14,4944	11,2705	12,7072
x(2)-grey	1,0000	3,9089	8,6133	14,8418	11,6996	15,3541
x(2)-rgb	1,0000	3,8871	8,6524	15,0851	11,7230	16,2626
x(4)-grey	Quota Exceeded					
x(4)-rgb						



Efficiency

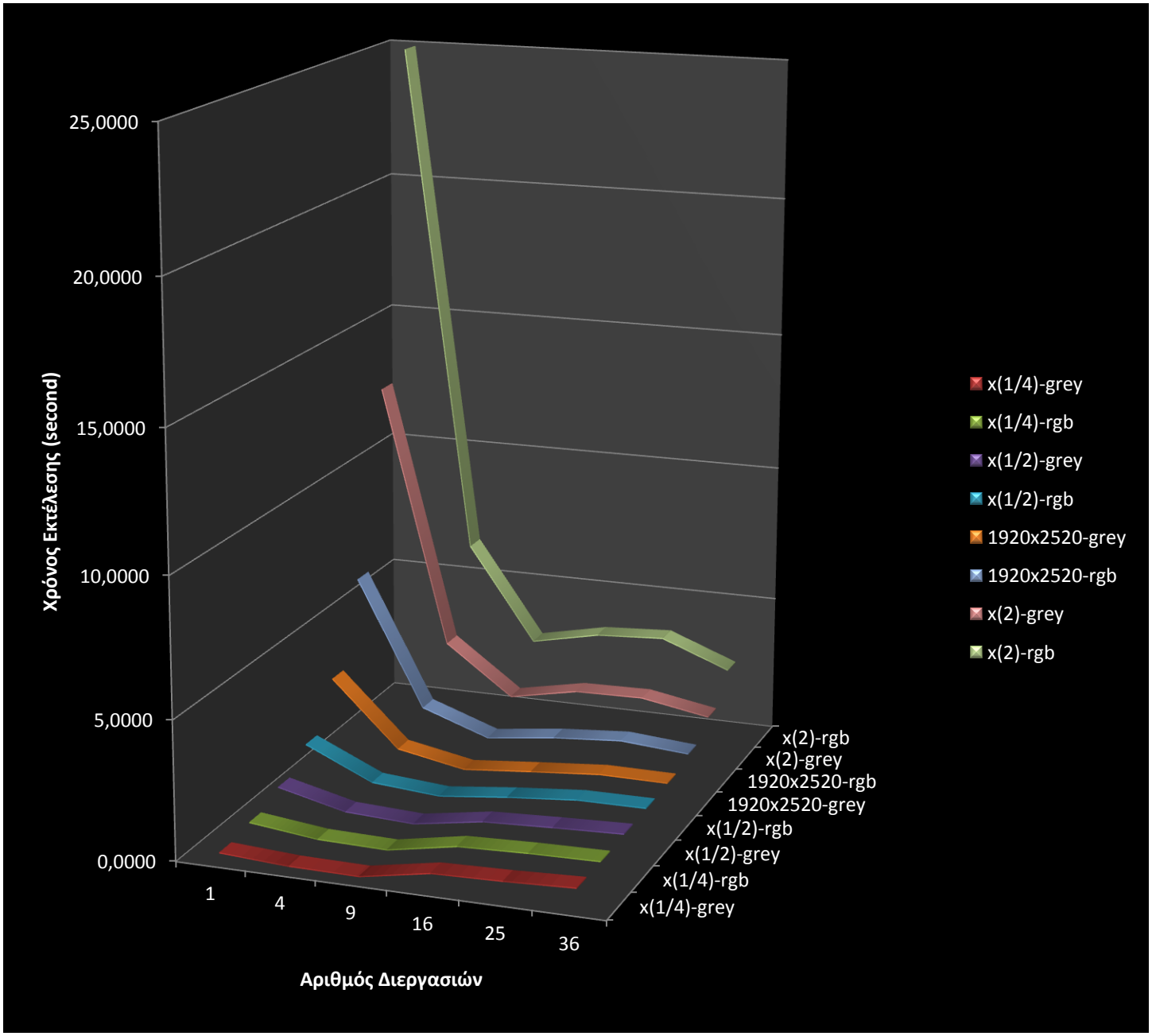
Εικόνες / Διεργασίες	1	4	9	16	25	36
x(1/4)-grey	1,0000	0,9075	0,7658	0,3951	0,0825	0,0223
x(1/4)-rgb	1,0000	0,9395	0,6576	0,6478	0,0859	0,0434
x(1/2)-grey	1,0000	0,9226	0,8331	0,8305	0,1712	0,0934
x(1/2)-rgb	1,0000	0,92623	0,8599	0,81267	0,32836	0,16616
1920x2520-grey	1,0000	0,9880	0,9196	0,9273	0,4106	0,3081
1920x2520-rgb	1,0000	0,9638	0,9456	0,9059	0,4508	0,3530
x(2)-grey	1,0000	0,9772	0,9570	0,9276	0,4680	0,4265
x(2)-rgb	1,0000	0,9718	0,9614	0,9428	0,4689	0,4517
x(4)-grey	Quota Exceeded					
x(4)-rgb						



MPI + OpenMP

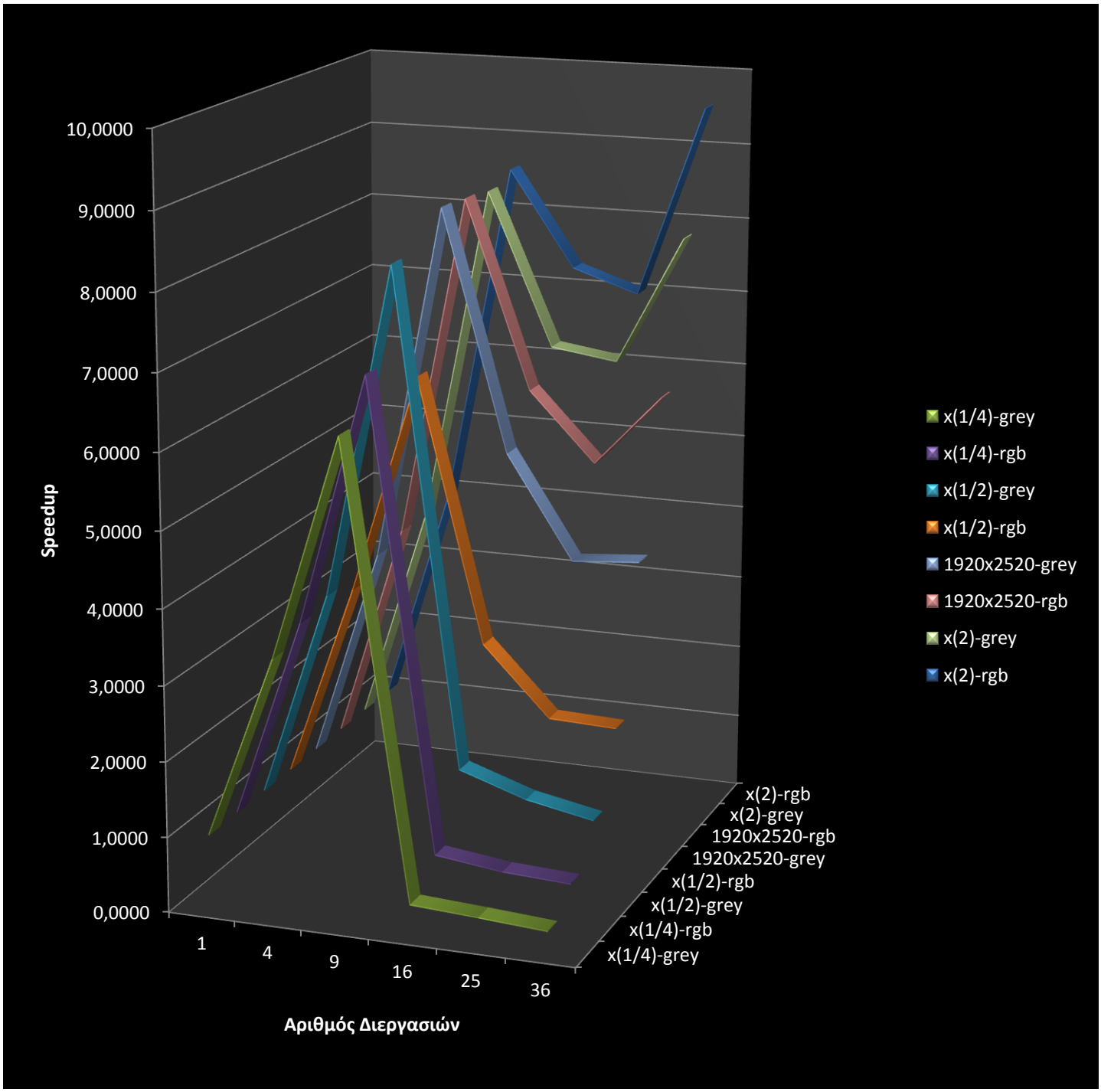
Συνολικός Χρόνος Εκτέλεσης(sec)

Εικόνες / Διεργασίες	1	4	9	16	25	36
x(1/4)-grey	0,2028	0,0592	0,0320	0,4838	0,5509	0,6557
x(1/4)-rgb	0,3906	0,1077	0,0566	0,5132	0,6010	0,6465
x(1/2)-grey	0,8037	0,2152	0,0991	0,5012	0,6140	0,7069
x(1/2)-rgb	1,5446	0,4326	0,2384	0,512	0,7304	0,7428
1920x2520-grey	3,2130	0,8451	0,3773	0,6015	0,8072	0,7932
1920x2520-rgb	6,2416	1,6009	0,7363	1,0431	1,2302	1,0347
x(2)-grey	12,7930	3,3113	1,5188	2,0048	2,0466	1,6068
x(2)-rgb	24,9407	6,3812	2,9100	3,4244	3,5681	2,6134
x(4)-grey	Quota Exceeded					
x(4)-rgb						



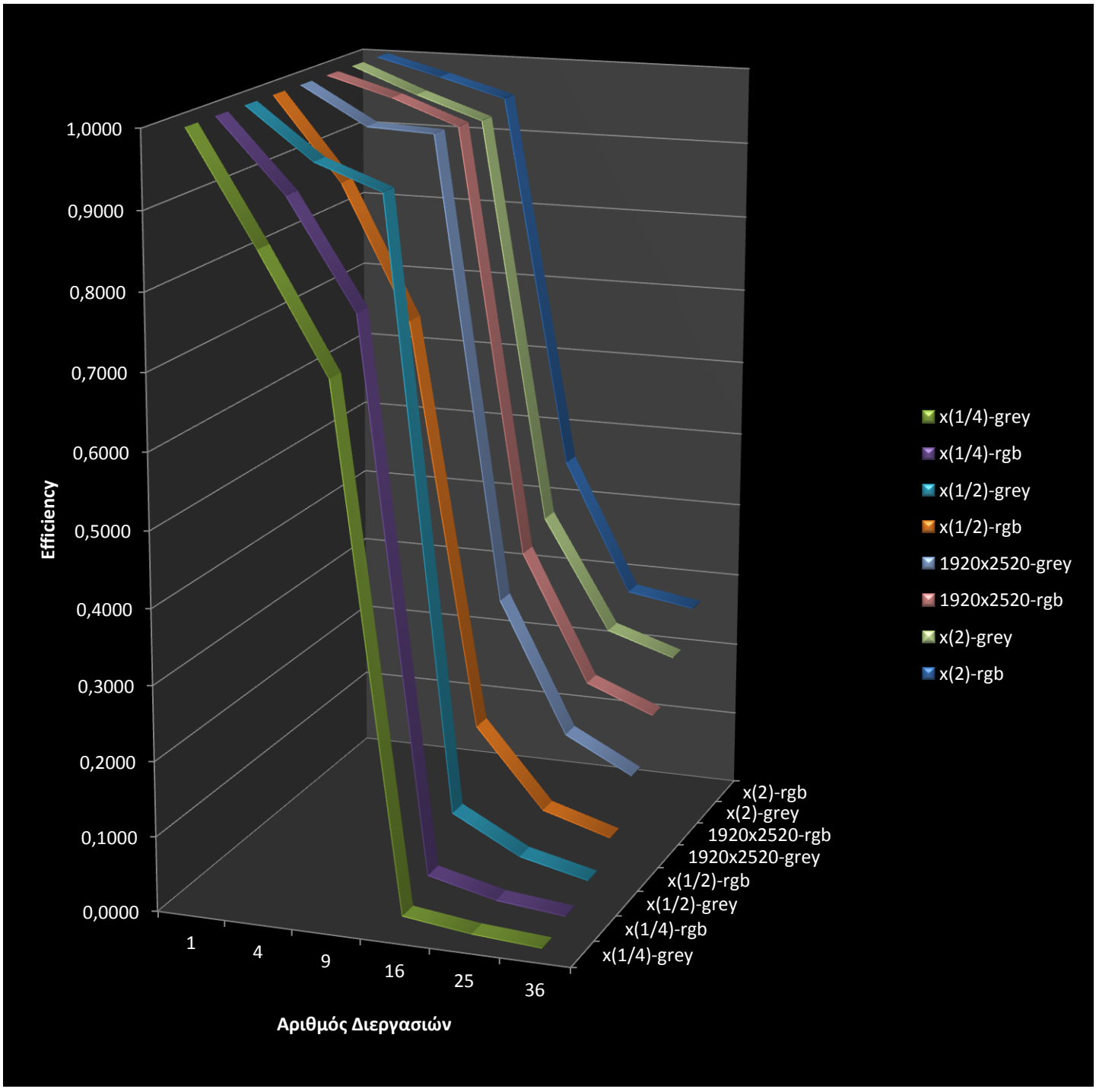
Speedup

Εικόνες / Διεργασίες	1	4	9	16	25	36
x(1/4)-grey	1,0000	3,4257	6,3375	0,4192	0,3681	0,3093
x(1/4)-rgb	1,0000	3,6267	6,9011	0,7611	0,6499	0,6042
x(1/2)-grey	1,0000	3,7347	8,1100	1,6036	1,3090	1,1369
x(1/2)-rgb	1,0000	3,5705	6,47903	3,0168	2,11473	2,07943
1920x2520-grey	1,0000	3,8019	8,5158	5,3416	3,9804	4,0507
1920x2520-rgb	1,0000	3,8988	8,4770	5,9837	5,0736	6,0323
x(2)-grey	1,0000	3,8634	8,4231	6,3812	6,2509	7,9618
x(2)-rgb	1,0000	3,9085	8,5707	7,2832	6,9899	9,5434
x(4)-grey	Quota Exceeded					
x(4)-rgb						



Efficiency

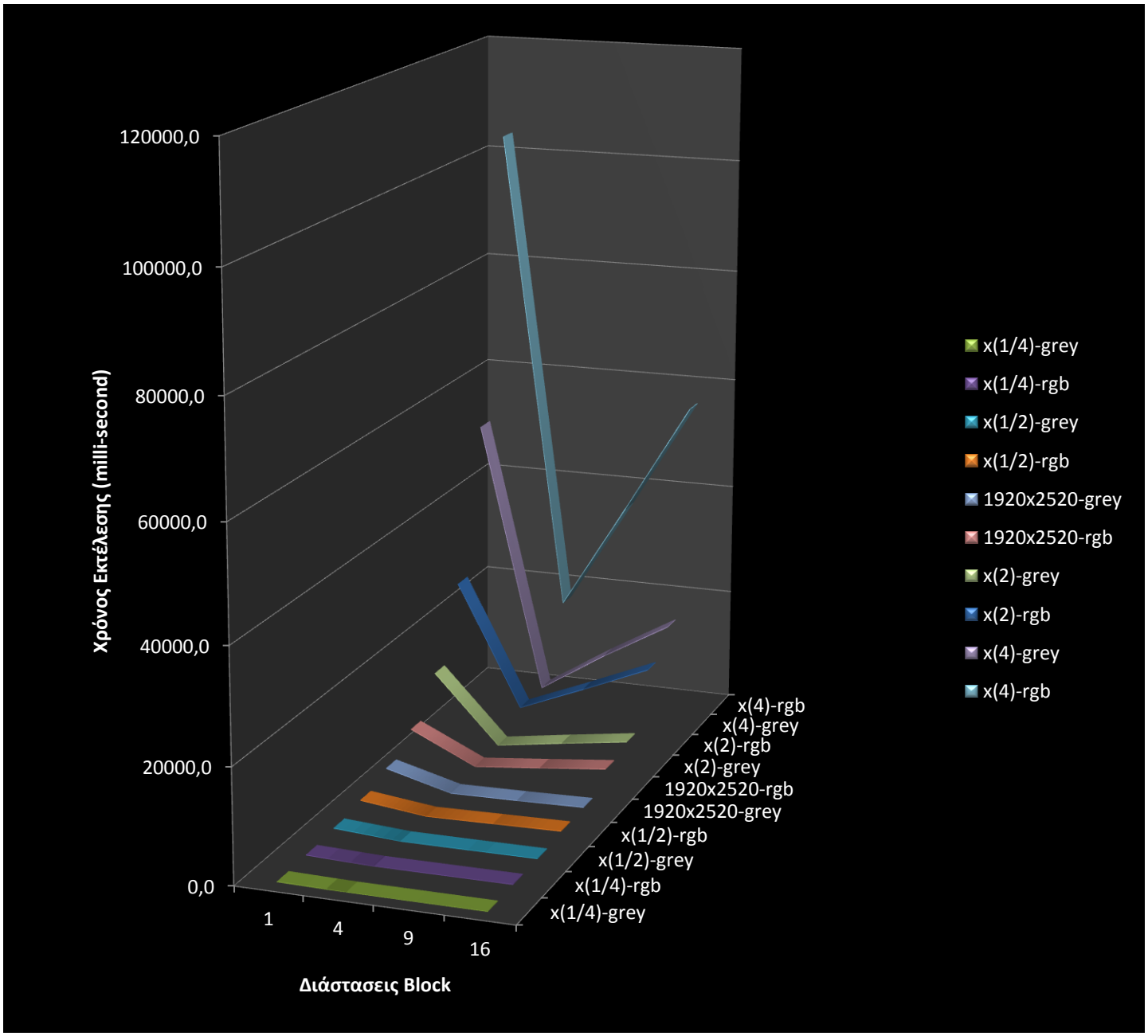
Εικόνες / Διεργασίες	1	4	9	16	25	36
x(1/4)-grey	1,0000	0,8564	0,7042	0,0262	0,0147	0,0086
x(1/4)-rgb	1,0000	0,9067	0,7668	0,0476	0,0260	0,0168
x(1/2)-grey	1,0000	0,9337	0,9011	0,1002	0,0524	0,0316
x(1/2)-rgb	1,0000	0,89263	0,71989	0,18855	0,08459	0,05776
1920x2520-grey	1,0000	0,9505	0,9462	0,3339	0,1592	0,1125
1920x2520-rgb	1,0000	0,9747	0,9419	0,3740	0,2029	0,1676
x(2)-grey	1,0000	0,9659	0,9359	0,3988	0,2500	0,2212
x(2)-rgb	1,0000	0,9771	0,9523	0,4552	0,2796	0,2651
x(4)-grey	Quota Exceeded					
x(4)-rgb						



CUDA

Συνολικός Χρόνος Εκτέλεσης(millisec)

Εικόνες / Διαστάσεις Block	1	4	9	16
x(1/4)-grey	201,0	13,8	21,0	34,5
x(1/4)-rgb	395,7	45,8	83,7	125,6
x(1/2)-grey	807,6	67,5	101,2	167,8
x(1/2)-rgb	1595,5	235,0	459,6	671,6
1920x2520-grey	3273,3	233,6	426,2	701,9
1920x2520-rgb	6482,0	1077,2	2272,4	3506,2
x(2)-grey	13096,3	1072,7	2844,2	4523,1
x(2)-rgb	26234,3	4293,5	9073,4	14015,6
x(4)-grey	52362,1	4325,8	11728,7	18469,9
x(4)-rgb	103306,0	17092,9	36312,5	55922,0



Παρατηρήσεις

- Ο έλεγχος για σύγκλιση πραγματοποιείται μόνο κάθε <CHECK> φορές.
- Μόνο 12 μηχανήματα με εγκατεστημένο MPI λειτουργούσαν την περίοδο των μετρήσεων, με συνέπεια να μην μπορούνε να αξιοποιηθούν οι εκτελέσεις με αριθμό διεργασιών ίσο με 36 και, όπως φαίνεται και στα γραφήματα, παρουσιάζεται αύξηση του χρόνου εκτέλεσης και επομένως μείωση του efficiency.
- Επίσης, λόγω της δημόσιας πρόσβασης στα μηχανήματα, οι χρόνοι επηρεάζονται από την εκάστοτε κίνηση στο δίκτυο, με αποτέλεσμα οι μετρήσεις να μην είναι 100% αξιόπιστες.
- Είναι φανερό ότι με μεγαλύτερες διαστάσεις εικόνας αξιοποιούμε πολύ καλύτερα τα μηχανήματα, οπότε παίρνουμε καλύτερο speedup και efficiency.
- Αντιθέτως, για μικρές διαστάσεις εικόνας βλέπουμε αύξηση του χρόνου εκτέλεσης με περισσότερες διεργασίες, λόγω του overhead που δημιουργείται για δημιουργία/καταστροφή και επικοινωνία μεταξύ των διεργασιών.
- Με την προσθήκη νημάτων με OpenMP στην υλοποίηση MPI βλέπουμε σημαντική βελτίωση χρόνου εκτέλεσης. Βέβαια, στις οριακές περιπτώσεις, που δεν αξιοποιούμε πλήρως τα μηχανήματα, βλέπουμε ακόμα μεγαλύτερο overhead, οπότε και πιο αργές εκτελέσεις.
- Στην υλοποίηση CUDA, παρατηρούμε δραστική μείωση του χρόνου εκτελέσεις με αύξηση των διαστάσεων των block. Όμως, όταν το μέγεθος του block ξεπεράσει τα διαθέσιμα cuda cores (block dim > 9), ξαναεμφανίζεται αύξηση του χρόνου εκτέλεσης, λόγω επιπλέον ρυθμίσεων από τη βιβλιοθήκη, χωρίς βέβαια προσφορά περισσότερης υπολογιστικής δύναμης.
- Γενικότερα, βλέπουμε καλύτερη συμπεριφορά στις υλοποιήσεις σε MPI από την υλοποίηση σε CUDA. Δεν έγινε όμως αναλυτικότερη σύγκριση, διότι οι μετρήσεις έγιναν σε διαφορετικό hardware.

Επεκτάσεις

Parallel I/O

Το διάβασμα/γράψιμο της εικόνας από/στη μνήμη στις υλοποιήσεις MPI έγιναν με *parallel I/O* μέσω συναρτήσεων που μας προσφέρει η βιβλιοθήκη MPI. Για να γίνει αυτό εφικτό, χρησιμοποιήθηκε ο τύπος *InnerArray*(βλ. Δομή Κώδικα) και η συνάρτηση MPI_*File_set_view* που επιτρέπει σε κάθε διεργασία να έχει παράλληλη πρόσβαση στο τμήμα της εικόνας που έχει αναλάβει.

```
/* Read from input image file */
MPI_File file;
MPI_File_open(new_comm, args.inputImage, MPI_MODE_RDONLY, MPI_INFO_NULL,
&file);
MPI_File_set_view(file, 0, MPI_UNSIGNED_CHAR, InnerArray, "native",
MPI_INFO_NULL);
MPI_File_read_all(file, image, arrayOfSubsizes[0] * arrayOfSubsizes[1],
MPI_UNSIGNED_CHAR,                    MPI_STATUS_IGNORE);
MPI_File_close(&file);
……
/* Write final result to output image file*/
MPI_File_open(new_comm, args.outputImage, MPI_MODE_WRONLY |
MPI_MODE_CREATE, MPI_INFO_NULL, &file);
MPI_File_set_view(file, 0, MPI_UNSIGNED_CHAR, InnerArray, "native",
MPI_INFO_NULL);
MPI_File_write_all(file, image, arrayOfSubsizes[0] * arrayOfSubsizes[1],
MPI_UNSIGNED_CHAR, MPI_STATUS_IGNORE);
MPI_File_close(&file);
```

Persistent Communication

Μιας και σε κάθε επανάληψη τα μηνύματα που στέλνονται και παραλαμβάνονται είναι τα ίδια, χρησιμοποιήθηκε η δυνατότητα *persistent communication* που μας προσφέρει η βιβλιοθήκη MPI, έτσι ώστε να ελαχιστοποιηθεί το overhead που δημιουργείται από την επαναληπτική ρύθμιση των μηνυμάτων.

```
MPI_Recv_init(indexAt(image,w+2,bpp,0,1), 1, Row, up, 1, new_comm, &recvReq[0][0]);
MPI_Send_init(indexAt(image,w+2,bpp,1,1), 1, Row, up, 2, new_comm, &sendReq[0][0]);
…..
MPI_Start(&sendReq[i][index]);
MPI_Start(&recvReq[i][index]);
```

CUDA Contant Memory

Για την αποθήκευση του φίλτρου χρησιμοποιήθηκε η *constant memory* της κάρτας γραφικών (GPU). Αυτό δίνει τη δυνατότητα στη βιβλιοθήκη CUDA να κάνει περαιτέρω βελτιστοποιήσεις σχετικά με τη πρόσβαση των νημάτων σε αυτή τη μνήμη.

```
/* Filter on GPU constant memory */
__constant__ int filterGPU[9];

.....

/* Copy filter from CPU to GPU (constant memory) */
CUDA_SAFE_CALL(cudaMemcpyToSymbol(filterGPU, filterCPU, 9 * sizeof(int)));
```

CUDA Events

Για τη μέτρηση εκτέλεσης του προγράμματος CUDA χρησιμοποιήθηκαν τα *Events* που μας προσφέρει η βιβλιοθήκη CUDA, έτσι ώστε να μην επηρεάζεται από εξωγενής παράγοντες (CPU thread scheduling, etc…).

```
CUDA_SAFE_CALL(cudaEventCreate(&start));
CUDA_SAFE_CALL(cudaEventCreate(&stop));
…..
/* Start time */
CUDA_SAFE_CALL(cudaEventRecord(start, 0));
…..
/* Stop time */
CUDA_SAFE_CALL(cudaEventRecord(stop, 0));
CUDA_SAFE_CALL(cudaEventSynchronize(stop));

float elapsedTime;
CUDA_SAFE_CALL(cudaEventElapsedTime(&elapsedTime, start, stop));
printf("%3.1f ms\n", elapsedTime);
```

Συμπεράσματα

Η τεχνική της συνέλιξης προσφέρει γόνιμο έδαφος για να αναδείξουμε την δυνατότητα βελτιστοποίησης ενός προγράμματος μέσω παραλληλισμού.

Χρησιμοποιήθηκαν τρεις διαφορετικοί τρόποι, όπου όλοι αποδείχθηκαν αποδοτικοί.

Οι μετρήσεις απέδειξαν τις υποθέσεις μας, δηλαδή την μεγαλύτερη βελτιστοποίηση με αύξηση του μεγέθους της εικόνας καθώς και την αύξηση των διεργασιών/νημάτων.

Δυστυχώς, ο περιορισμένος αριθμός των μηχανημάτων έκανε αδύνατη την εξέταση κλιμάκωσης(*scalability*) των μεθόδων.

Παράρτημα: Οδηγίες Εκτέλεσης

Τα ορίσματα της εκτέλεσης είναι τα ίδια για όλες τις υλοποιήσεις:

./a.out -i <inputImage> -o <outputImage> -w <width> -h <height> -bpp <bitsPerPixel> -l <iterations>

- Υποστηρίζονται μόνο εικόνες τύπου .raw
- <input/outputImage>: εικόνα για ανάγνωση/εγγραφή
- <width/height>: το πλάτος/ύψος της εικόνας εισόδου
- <bpp> = 1 για ασπρόμαυρες εικόνες, 3 για έγχρωμες εικόνες
- <iterations>: Πόσες φορές θα εφαρμοστεί η συνέλιξη

*Στην εκτέλεση της υλοποίησης CUDA πρέπει να δώσουμε ένα επιπλέον όρισμα –b <blockSize>.

MPI / MPI + OpenMP / CUDA

1. cd MPI/

   MPI_OpenMP/

   CUDA/

2. make
3. make run
   ή
   manual run με τα επιθυμητά ορίσματα

Βιβλιογραφία

- An Introduction to Parallel Programming, Peter S. Pacheco
- Introduction to Parallel Computing, Grama-Gupta-Karypis-Kumar
- MPI Data Types, Research Computing (University of Colorado)
- Parallel Programming for Multicore Machines using MPI and OpenMP, MIT
- A “hands-on” Introduction to OpenMP, Mattson-Wrinn
- Common Mistakes in OpenMP and How to Avoid Them, Suß-Leopold
- Hybrid MPI + OpenMP Programming, CSC Prace
- CUDA by Example, Sanders-Kandrot
- CUDA Thread Basics, Samuel S. Cho