Δημοκρίτειο Πανεπιστήμιο Θράκης

Τμήμα: Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο: Αρχιτεκτονικής Υπολογιστών και Συστημάτων Υψηλών Επιδόσεων

Εργασία για το μάθημα Παράλληλοι Αλγόριθμοι και Υπολογιστική Πολυπλοκότητα

Θέμα: **Εφαρμογή συνελικτικών φίλτρων και maxpooling σε RGB εικόνες**

Έτος: 2023-2024

Ομάδα 8

Ονοματεπώνυμο:

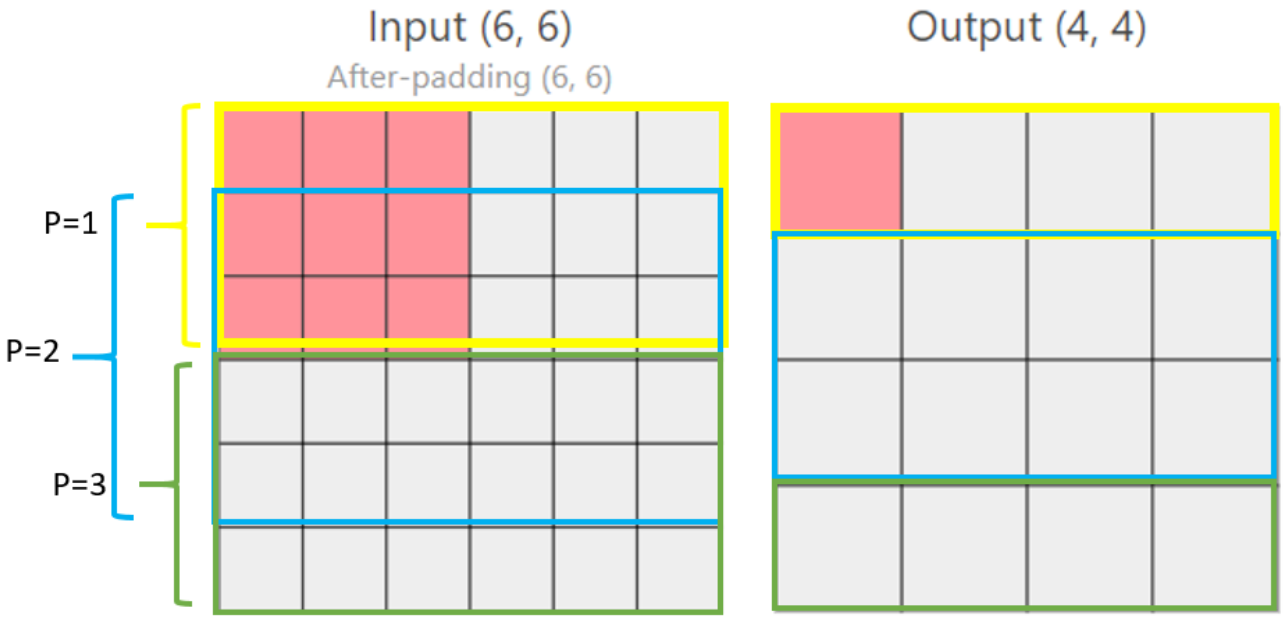
Γιάμπαστος Πουλέκας Δημήτριος ee: 58147

Μωραΐτη Παναγιώτα ee: 58054

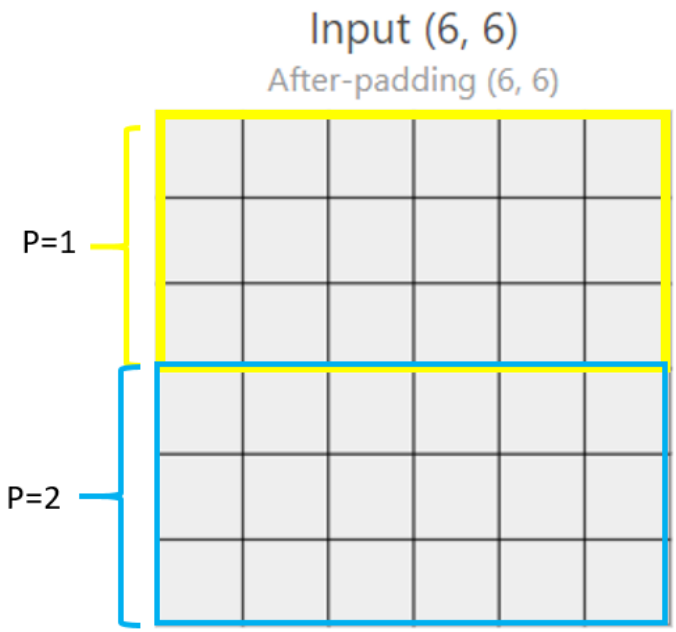
Μεθοδολογία της παράλληλης επίλυσης

Αρχικά, θέλουμε για την παραγωγή του feature map, οι γραμμές που θα παραχθούν με τη διαδικασία της συνέλιξης να μοιραστούν στις διεργασίες. Η πρώτη και η τελευταία διεργασία στην έξοδο (feature map) θα παράγουν μία γραμμή λιγότερη από τις υπόλοιπες, επειδή δεν έχουμε padding. Οπότε, η διάσταση της εικόνας εξόδου θα είναι μικρότερη από την αρχική εικόνα και για kernel size 3x3, είναι μικρότερη κατά δύο γραμμές και στήλες.

Οι γραμμές που θα πρέπει να παίρνει κάθε διεργασία από τον αρχικό πίνακα για να δημιουργήσει το αποτέλεσμα επικαλύπτονται μεταξύ τους. Για παράδειγμα, στην παρακάτω εικόνα αν θεωρήσουμε ότι έχουμε 3 διεργασίες, η διεργασία 1 πρέπει να έχει τις γραμμές 1, 2, 3 και να παράξει την $1^η$ γραμμή της συνελιγμένης εικόνας. Η διεργασία 2, πρέπει να έχει τις γραμμές 2, 3, 4, 5 και να παράξει τις γραμμές 2 και 3 της συνελιγμένης εικόνας. Η διεργασία 3, πρέπει να έχει τις γραμμές 4, 5, 6 και να παράξει την $4^η$ γραμμή της συνελιγμένης εικόνας. Οπότε, λόγω της ύπαρξης ίδιων γραμμών του αρχικού πίνακα σε διαφορετικές διεργασίες δεν μπορεί να χρησιμοποιηθεί η συνάρτηση scatter για να μοιραστούν οι γραμμές. Εμείς επιλέξαμε να στέλνουμε όλον τον πίνακα κάθε φορά σε όλες τις διεργασίες, δηλαδή όλη την εικόνα. Μία βελτίωση που θα μπορούσε να γίνει είναι να χρησιμοποιήσουμε Send/Receive σε συνδυασμό με scatter και να στέλνουμε μόνο τα κομμάτια της εικόνας που είναι απαραίτητα για κάθε διεργασία, ώστε να εξοικονομήσουμε χρόνο στην επικοινωνία.

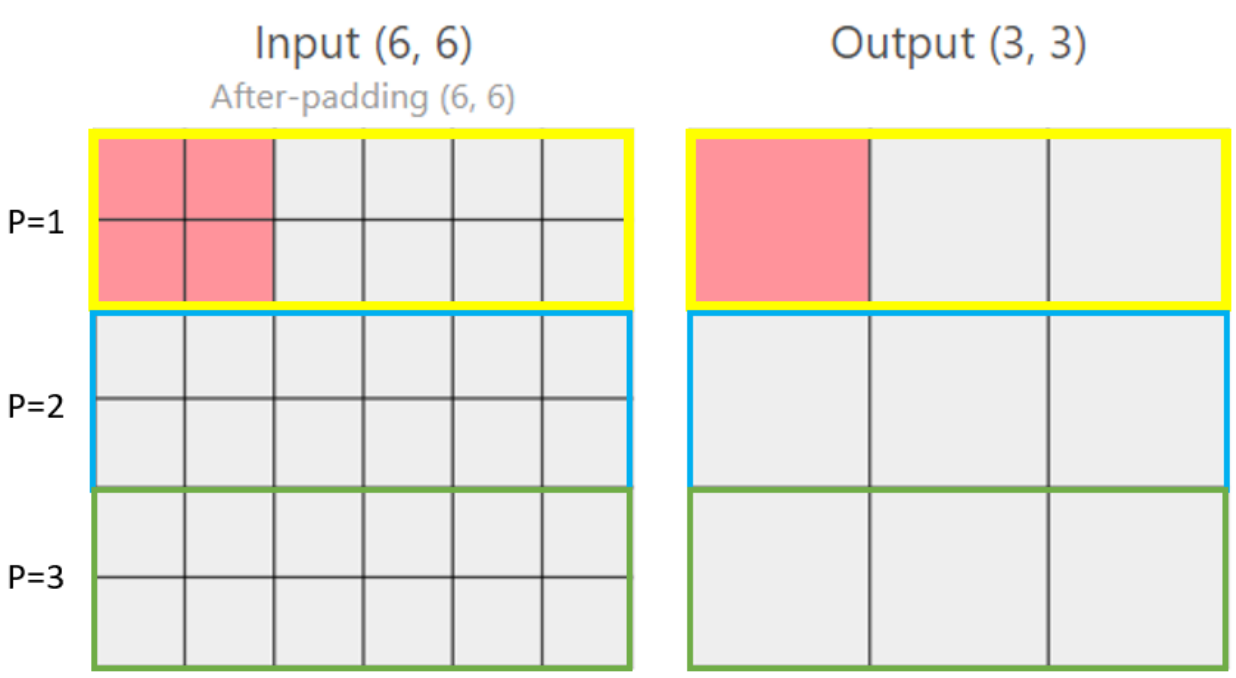


Σε περίπτωση που το stride (βήμα) είναι ίσο με το kernel size, μόνο τότε μπορεί να χρησιμοποιηθεί η scatter. Αν όμως γίνει αυτό έχουμε μεγάλη μείωση στις διαστάσεις της εικόνας εξόδου. Για kernel size 3 και stride 3, αν η αρχική εικόνα είναι 6x6, η εικόνα εξόδου έχει διάσταση 2x2.



Επίσης κάθε διεργασία (εκτός από την 1η και την τελευταία) αναλαμβάνει να δημιουργήσει **Height/P** γραμμές της τελικής εικόνας. Η 1η και η τελευταία διεργασία αναλαμβάνουν **Height/P-1** γραμμές (δεν υπάρχει padding). Αν **Height/P** δεν είναι ακέραιος αριθμός, τότε τις γραμμές που απομένουν θα τις αναλάβει η 1η διεργασία.

Τέλος η 1η διεργασία (root), μαζεύει όλες τις γραμμές της τελικής εικόνας εξόδου και εκτελεί την πράξη του maxpooling. Το maxpooling δεν πραγματοποιείται παράλληλα, επειδή πρέπει πάλι να στείλουμε κομμάτια της εικόνας σε όλες τις διεργασίες κάτι που θεωρείται αρκετά χρονοβόρο, ενώ το maxpooling είναι μια σχετικά απλή πράξη.



Μια βελτίωση θα ήταν να υλοποιηθεί το maxpooling παράλληλα σε όλες τις διεργασίες. Επειδή δεν υπάρχουν επικαλύψεις μπορεί να χρησιμοποιηθεί η scatter.

Έλεγχος ορθότητας κώδικα

Σε αυτή την περίπτωση, όπως προαναφέραμε, δημιουργούμε εμείς τυχαία πίνακες με τις διαστάσεις που θέλουμε με σκοπό να επαληθεύσουμε τα αποτελέσματα των πράξεων.

Πριν περάσουμε στην εκτέλεση των παραπάνω όμως, θα ορίσουμε μικρές διαστάσεις Width = 6 και Height = 12, για να μπορούμε να επαληθεύσουμε το σωστό αποτέλεσμα των πράξεων μεταξύ σειριακού και παράλληλου κώδικα. Παρακάτω φαίνονται τα αποτελέσματα των πινάκων.

```
Featuremap Serial:
56      26      0       59
19      0       34      30
21      0       113     0
0       40      39      5
92      0       33      33
0       72      50      72
80      8       80      0
0       34      0       0
26      4       38      0
67      20      0       50

Featuremap Parallel:
56      26      0       59
19      0       34      30
21      0       113     0
0       40      39      5
92      0       33      33
0       72      50      72
80      8       80      0
0       34      0       0
26      4       38      0
67      20      0       50

Featuremap Serial After Pooling:
56      59
40      113
92      72
80      80
67      50

Featuremap Parallel After Pooling:
56      59
40      113
92      72
80      80
67      50

Correct answer!!!!!!!!!!!
```

Βλέπουμε ότι τα αποτελέσματα σειριακού και παράλληλου κώδικα είναι ίδια. Επιπλέον, με σκοπό να επαληθεύσουμε ότι οι τιμές που βγάζουν οι αλγόριθμοι από τις πράξεις touconvolution, συνθέσαμε τον ίδιο αλγόριθμο σε python χρησιμοποιώντας την έτοιμη συνάρτηση correlate2D. Τα αποτελέσματα που

πήραμε για τις ίδιες εισόδους καναλιών RGB, όπως και για τα ίδια kernel και bias ήταν τα ίδια.

```
# Perform 2D convolutions with bias
red_result = correlate2d(matrixR, kernel1, mode='valid')
green_result = correlate2d(matrixG, kernel2, mode='valid')
blue_result = correlate2d(matrixB, kernel3, mode='valid')

# Combine the results by element-wise addition
# feature_map = red_result + green_result + blue_result + bias
feature_map = np.round(red_result + green_result + blue_result + bias).astype(int)
feature_map[feature_map < 0] = 0
print(feature_map)
```

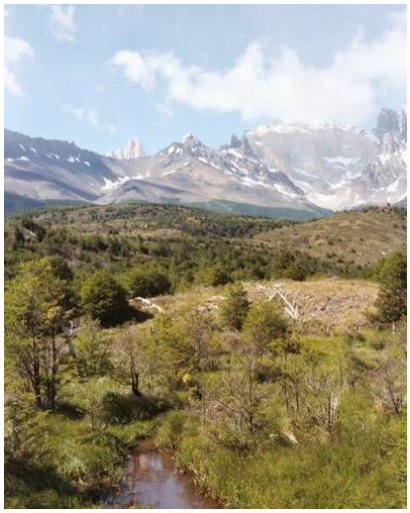
```
[[ 56  26   0  59]
 [ 19   0  34  30]
 [ 21   0 113   0]
 [  0  40  39   5]
 [ 92   0  33  33]
 [  0  72  50  72]
 [ 80   8  80   0]
 [  0  34   0   0]
 [ 26   4  38   0]
 [ 67  20   0  50]]
```

Συνεπώς τα αποτελέσματα του κώδικα μας είναι σωστά.

Περιγραφή των πειραματικών αποτελεσμάτων

Η περιγραφή των πειραματικών αποτελεσμάτων έγινε για εφαρμογή του φίλτρου σε πραγματικές εικόνες RGB διάφορων μεγεθών.

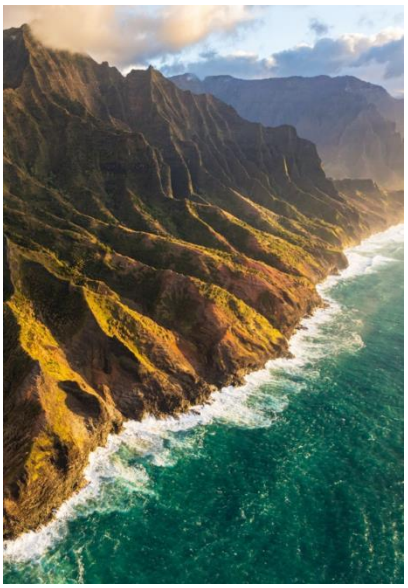
Ο αλγόριθμος εφαρμόστηκε σε διάφορα μεγέθη εικόνων, και πιο συγκεκριμένα στα παρακάτω 4 μεγέθη προβλήματος.



Mountain (400 x 600) px

Elephant (2040 x 1356) px

Flower (4000 x 3000) px



Sea (4000 x 6000) px

Το πρώτο βήμα είναι να κατεβάσουμε τις εικόνες από το διαδίκτυο στις διαστάσεις που αναγράφονται και έπειτα να τις μετατρέψουμε από ‘ .jpg ’ σε ‘ .raw ‘ αρχεία για να μπορούμε να τις διαβάσουμε σαν πίνακες των καναλιών RGB. Έτσι λοιπόν, ως input θα έχουμε τρείς δισδιάστατους πίνακες, matrixR, matrixG και matrixB, με διαστάσεις width x height της αντίστοιχης εικόνας.

Αφού τρέξαμε τον αλγόριθμο που περιγράφηκε παραπάνω για κάθε εικόνα και για διαφορετικό αριθμό διεργασιών, πήραμε σαν output το ‘ .raw ‘ αρχείο της τελικής μορφοποιημένης εικόνας αλλά και το ‘ .raw ‘ αρχείο της εικόνας πριν την εφαρμογή του max pooling (δηλαδή το feature map).

Τελευταίο βήμα αποτελεί η μετατροπή των ‘ .raw ‘ αρχείων σε ‘ .png ‘ αρχεία για να μπορέσουμε να διακρίνουμε τα οπτικά αποτελέσματα. Παρακάτω φαίνονται της εικόνας 2 (flower).

Feature_map_flower.png

Feature_map_after_pooling_flower.png

Όπως μπορούμε να παρατηρήσουμε, το output μας δίνεται σε grayscale χρωματισμό και ιδιαίτερα με την εφαρμογή του max pooling, στην τελική εικόνα επιλέγονται να τονιστούν μονό οι μεγαλύτερες τιμές του αντίστοιχου 2x2 kernel, τονίζοντας τα έντονα pixel περισσότερο.

Πέρα από τα οπτικά αποτελέσματα, μετρήσαμε τον χρόνο εκτέλεσης του σειριακού υπολογισμού του αποτελέσματος καθώς και της παράλληλης υλοποίησης σε MPI που αναφέρθηκε λεπτομερώς νωρίτερα. Επίσης υπολογίσαμε την επιτάχυνση (speedup) σε κάθε βήμα και τα αποτελέσματα φαίνονται παρακάτω. Με κίτρινο υπογραμμίζονται οι υψηλότερες τιμές του speedup που πετύχαμε.

Dimensions (WxH)	Metric (s)	Serial (P=1)	P=4	P=8	P=12
400 x 600	Time total	0,107126	0,212022	0,243764	0,305757
	Time without broadcast	-	0,068897	0,054103	0,045929
	Time without communication	-	0,058061	0,041206	0,032421
	Speedup	-	0,505259	0,438166	0,366595
	Speedup without broadcast	-	1,554872	1,974179	2,440484
2040 x 1356	Time total	1,348418	1,648286	2,156589	2,17859
	Time without broadcast	-	0,810268	0,59138	0,571792
	Time without communication	-	0,696328	0,460575	0,43244
	Speedup	-	0,818073	0,608853	0,59345
	Speedup without broadcast	-	1,664163	2,220308	2,261109
4000 x 3000	Time total	5,694383	7,439295	9,146094	8,704094
	Time without broadcast	-	3,335905	2,471598	2,204195
	Time without communication	-	2,849251	1,914505	1,617856
	Speedup	-	0,765447	0,619804	0,652758
	Speedup without broadcast	-	1,706998	2,293572	2,57766

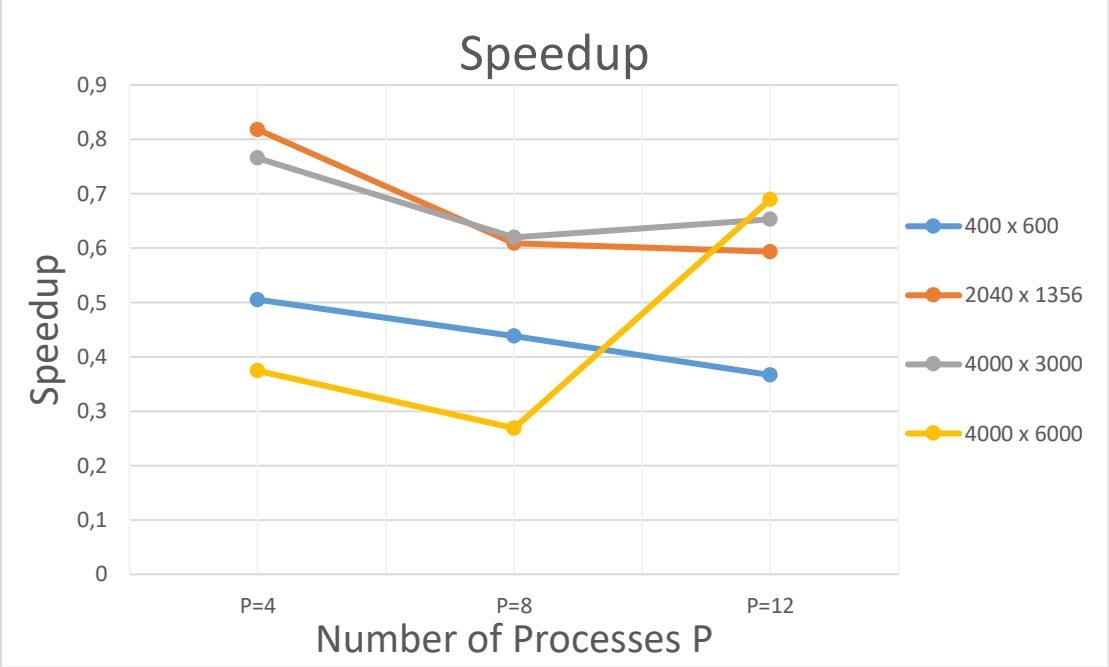
4000 x 6000	Time total	25,205195	67,295631	75,110108	16,736715
	Time without broadcast	-	51,103321	33,919305	4,398881
	Time without communication	-	43,40802	29,682362	3,248605
	Speedup	-	0,374544	0,26886	0,68983
	Speedup without broadcast	-	0,49322	0,595357	2,624641

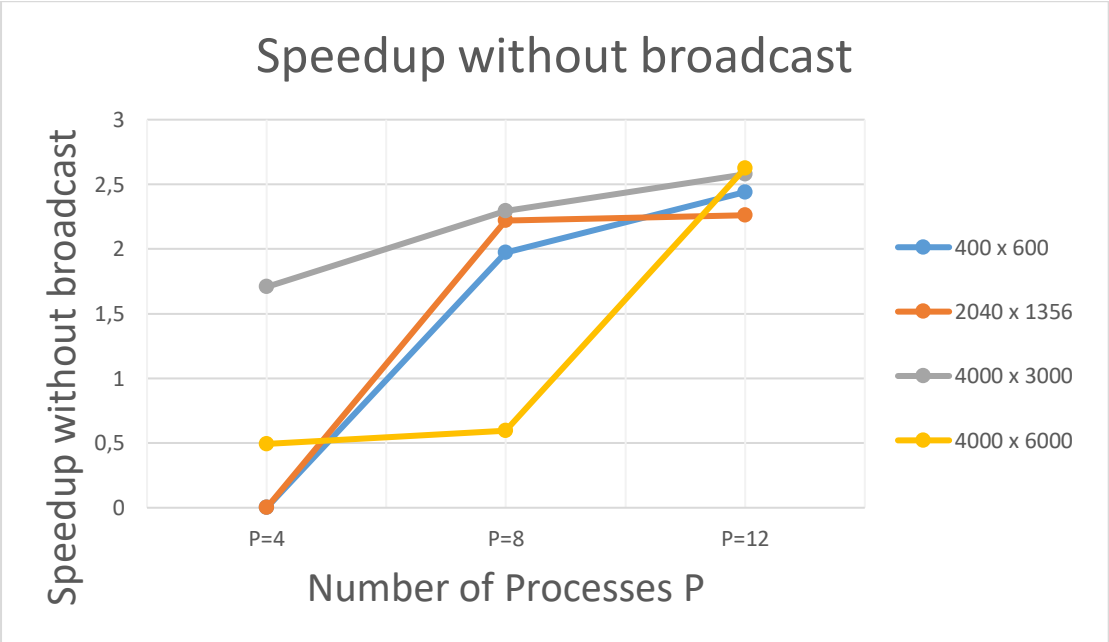
Παρουσιάζονται παρακάτω οι χρόνοι για την επικοινωνία μεταξύ των διεργασιών και δίνεται έμφαση στην τεράστια επιρροή που έχει η Broadcast στον συνολικό χρόνο επικοινωνίας σε αντίθεση με την Gather.

Dimensions (WxH)	Metric (s)	P=4	P=8	P=16
400 x 600	Total time for communication	0,153961	0,202558	0,273336
	Time for Broadcast	0,143125	0,189661	0,259828
	Time for Gather	0,010836	0,012897	0,013508
2040 x 1356	Total time for communication	0,951958	1,696014	1,74615
	Time for Broadcast	0,838018	1,565209	1,606798
	Time for Gather	0,11394	0,130805	0,139352
4000 x 3000	Total time for communication	4,590044	7,231589	7,086238
	Time for Broadcast	4,10339	6,674496	6,499899
	Time for Gather	0,486654	0,557093	0,586339
4000 x 6000	Total time for communication	23,887611	45,427746	13,48811
	Time for Broadcast	16,19231	41,190803	12,337834
	Time for Gather	7,695301	4,236943	1,150276

Βλέπουμε ότι η Broadcast καλύπτει το **91%** του συνολικού χρόνου επικοινωνίας.

Παρουσιάζονται επίσης και τα διαγράμματα της επιτάχυνσης για άμεση σύγκριση της παραλληλίας με γνώμονα το πλήθος των διεργασιών.





<u>Συμπεράσματα</u>

Παρατηρούμε ότι το κόστος τη επικοινωνίας για την αποστολή ολόκληρης της εικόνας σε όλες τις διεργασίες καταλαμβάνει πολλές φορές περισσότερο χρόνο από το υπόλοιπο πρόγραμμα (δεν υπάρχει speedup). Βέβαια, αν λάβουμε υπόψιν ότι όλοι οι υπολογιστές έχουν την εικόνα (δεν λαμβάνω υπόψιν το χρόνο για το broadcast), τότε υπάρχει επιτάχυνση με τη χρήση πολλών διεργασιών. Επίσης, αν θεωρήσουμε ότι στέλνουμε την εικόνα μόνο μία φορά και εφαρμόζουμε διαδοχικές συνελίξεις, πάλι θα υπάρχει

επιτάχυνση. Πάντως η δυσκολία που υπάρχει στη χρήση της scatter για το διαμοιρασμό των γραμμών δεν καθιστά εύκολη την υλοποίηση της πράξης της συνέλιξης.

Τέλος, όσο αυξάνεται ο αριθμός των διεργασιών, τόσο μεγαλύτερη είναι η επιτάχυνση. Για την μεγαλύτερη εικόνα στις περισσότερες διεργασίες έχουμε το μεγαλύτερο speedup, κάτι που δείχνει ότι για μεγαλύτερα μεγέθη προβλήματος υπάρχουν περιθώρια αύξησης του speedup.

Βιβλιογραφία

https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26

https://poloclub.github.io/cnn-explainer/