Τσεσμελής Δημήτρης 1115201400208

Φλωράκης Απόστολος 11150201400217

*Η εργασία που παραδίδουμε το Σεπτέμβρη είναι βελτίωση της εργασίας που παραδώσαμε κατά τη διάρκεια του χειμερινού εξαμήνου.
Συγκεκριμένα έχουν γίνει οι παρακάτω αλλαγές:

- Προσθήκη Paraver και σχολιασμού αποτελεσμάτων.
- Προσθήκη χρόνων εκτέλεσης Mpi+All_Reduce και Mpi+OpenMp και σχολιασμού αποτελεσμάτων.
- Υλοποίηση και Παρουσίαση.

MPI:

Γενικές παρατηρήσεις για το σχεδιασμό:

Στην εργασία μας έχουν υλοποιηθεί όσα συζητήθηκαν κατά τη διάρκεια των μαθημάτων και της παρουσίασης της άσκησης και συγκεκριμένα διαχωρισμός σε block, ασύγχρονη επικοινωνία, data types, τοπολογίες και χρήση MPI_Allreduce.

Αναλυτικότερα:

- Αναφορικά με την επικοινωνία μεταξύ των διεργασιών, σε κάθε γενεά στέλνονται στους γείτονες οι απαραίτητες πληροφορίες με τη χρήση της MPI_Isend και αναμένονται οι αντίστοιχες πληροφορίες από αυτούς με τη χρήση των MPI_Irecv και MPI_Wait, ενώ παράλληλα γίνεται υπολογισμός του εσωτερικού πίνακα.
- Οι πληροφορίες που αναφέρονται παραπάνω αναπαριστώνται με τη χρήση των data types. Για την αποστολή μιας γραμμής στέλνεται δείκτης στην πρώτη θέση της γραμμής καθώς τα δεδομένα αποθηκεύονται σε συνεχόμενες θέσεις μνήμης. Για την αποστολή στήλης υλοποιήθηκε ο τύπος "col_type" με τη χρήση MPI_Type_vector. Τα δεδομένα της στήλης δεν αποθηκεύονται συνεχόμενα καθώς μεταξύ 2 στοιχείων υπάρχουν n-2 στοιχεία.
- Με τη σωστή τοπολογία επιτυγχάνουμε ανάθεση των γειτονικών διεργασιών σε "γειτονικούς" επεξεργαστές ώστε να μειώσουμε το overhead της επικοινωνίας. Συγκεκριμένα με τη χρήση του MPI_Cart_create δημιουργούμε ένα νέο δίαυλο επικοινωνίας που μοιάζει με τη μορφή του προβλήματός μας, δηλαδή με ένα πλέγμα n x n (όπου n η τετραγωνική ρίζα του αριθμού των διεργασιών).
- Τέλος για τον έλεγχο τερματισμού μεταξύ των γενεών χρησιμοποιήθηκε το MPI_Allreduce. Συγκεκριμένα, σε κάθε επανάληψη η κάθε διεργασία στέλνει 1 αν άλλαξε κάτι μεταξύ της τρέχουσας και της προηγούμενης γενεάς αλλιώς στέλνει 0. Αυτά αθροίζονται συνολικά και στην περίπτωση που το συνολικό άθροισμα είναι 0 γίνεται τερματισμός του προγράμματος χωρίς να ολοκληρωθούν οι συνολικές επαναλήψεις.
- Μετά την διαδικασία εύρεσης των γειτόνων κάθε διεργασίας ακολουθεί ένα MPI_Barrier για να διασφαλιστεί ότι κάθε μία διεργασία έχει υπολογίσει σωστά το id όλων των γειτονικών.

Ακολουθούν κάποιοι πίνακες με τα αποτελέσματα από τις εκτελέσεις που έγιναν στο cluster της σχολής.

Τα μηχανήματα που χρησιμοποιήθηκαν βρίσκονται στο φάκελο machines.

(15 μηχανήματα και 100 γενεές).

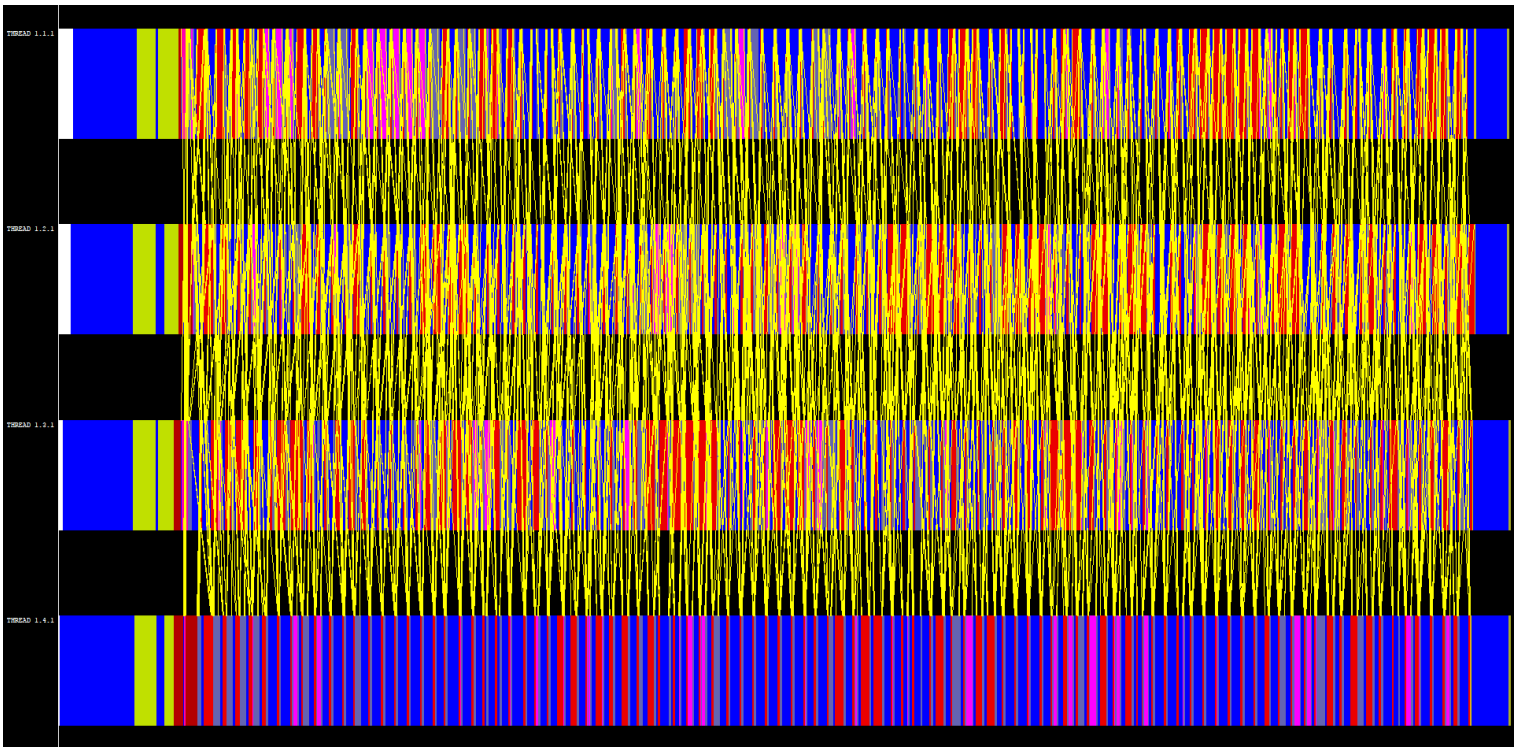
χρόνοι	210	420	840	1680	3360
1	0.13	0.51	2.06	8.2	33
4	0.07	0.2	0.7	3.1	25
9	0.04	0.1	0.3	1.4	7.1
16	0.1	0.1	0.4	1.3	5.3
25	0.2	0.2	0.4	0.8	3.4
36	1.2	1.2	1.2	1.2	2.9
49	1.3	1.5	1.5	1.3	2.5

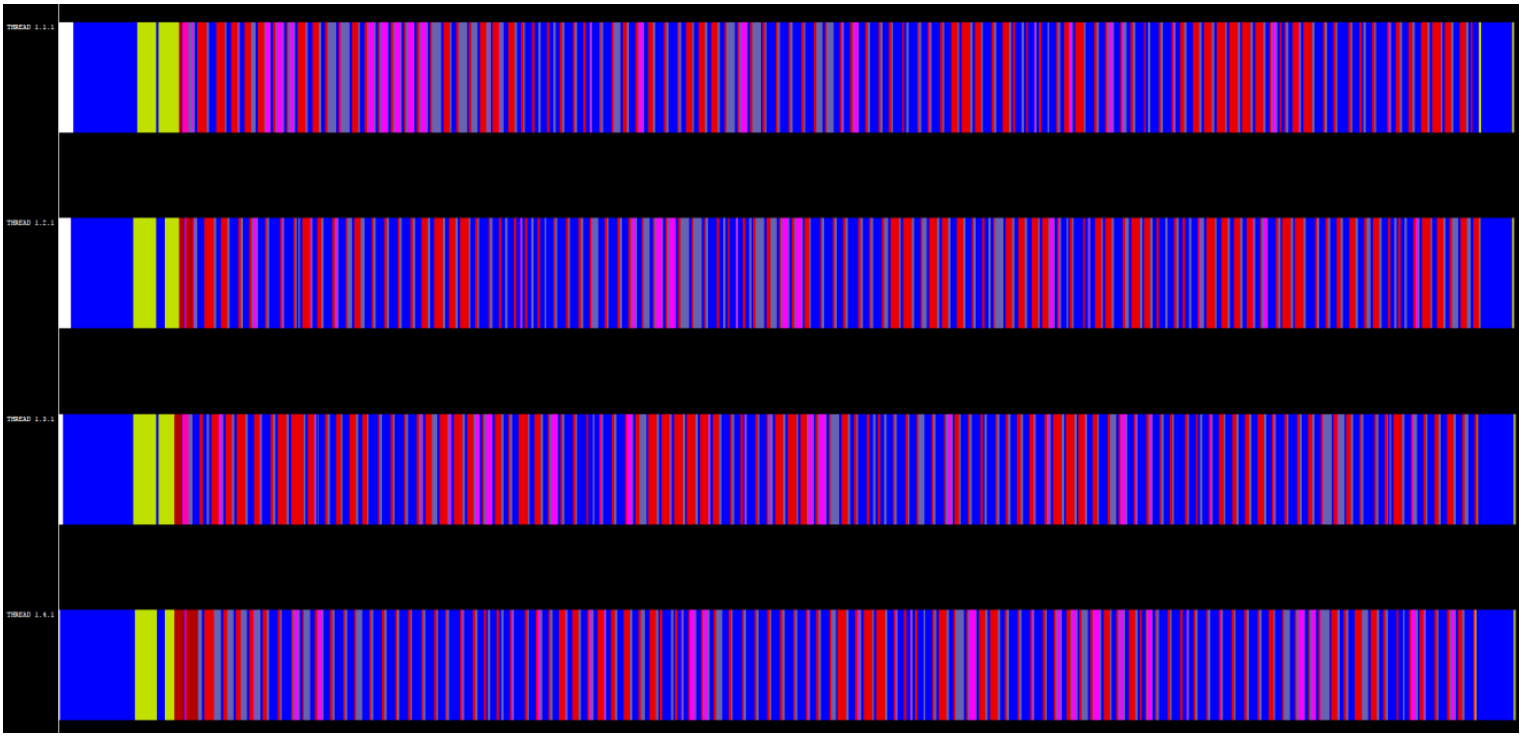
Speed-up	210	420	840	1680	3360
1	1	1	1	1	1
4	1.85	2.55	2.94	2.67	1.32
9	3.25	5.1	6.8	5.8	4.64
16	1.3	5.6	5.15	6.3	6.2
25		2.55	5.15	10.25	9.7
36			1.71	6.8	11.37
49			1.37	6.3	13.2

efficiency	210	420	840	1680	3360
1	1	1	1	1	1
4	0.46	0.63	0.73	0.66	0.33
9	0.36	0.56	0.75	0.64	0.51
16	0.08	0.35	0.32	0.39	0.38
25		0.1	0.2	0.41	0.38
36			0.04	0.18	0.31
49			0.02	0.12	0.27

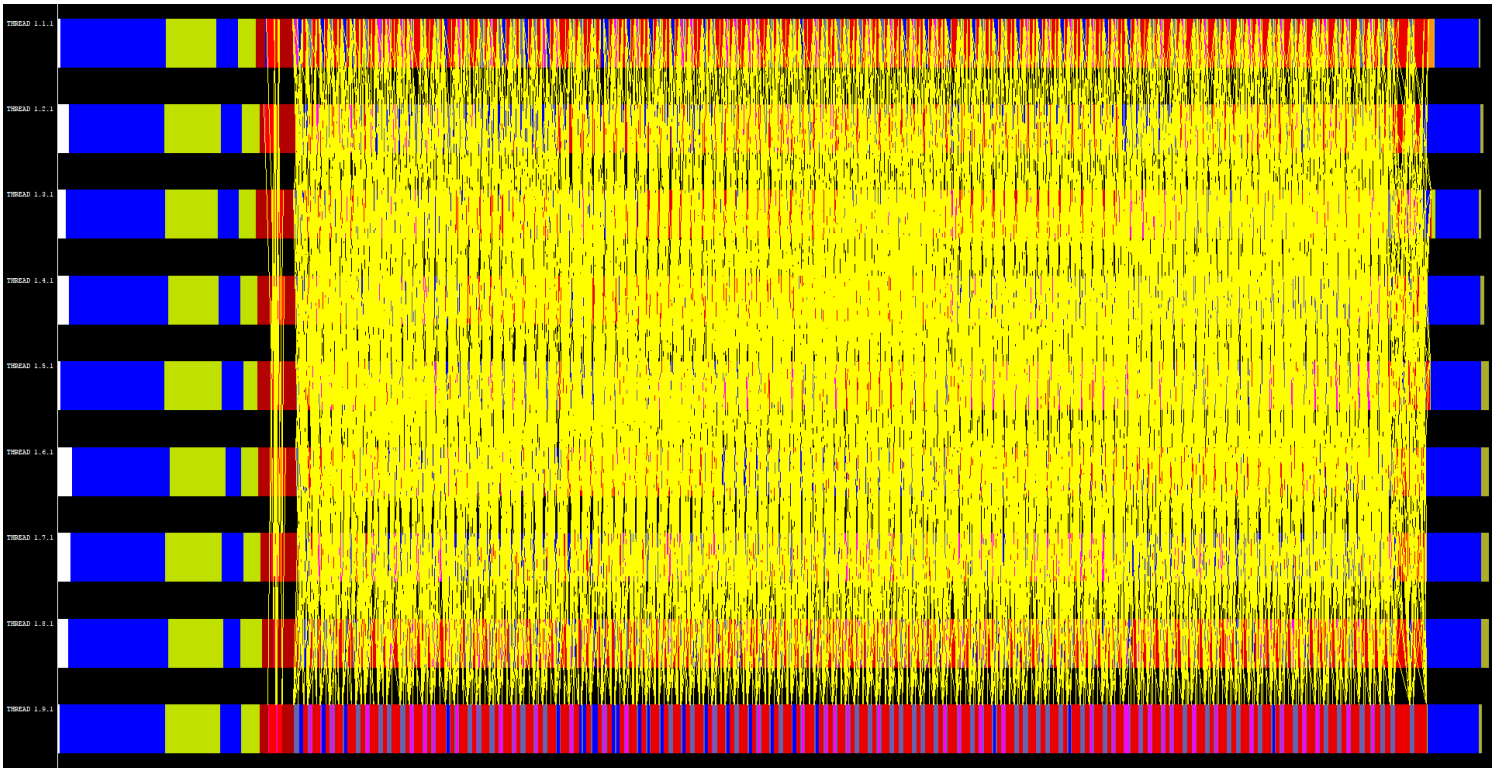
Paraver-Ανάλυση επικοινωνίας:

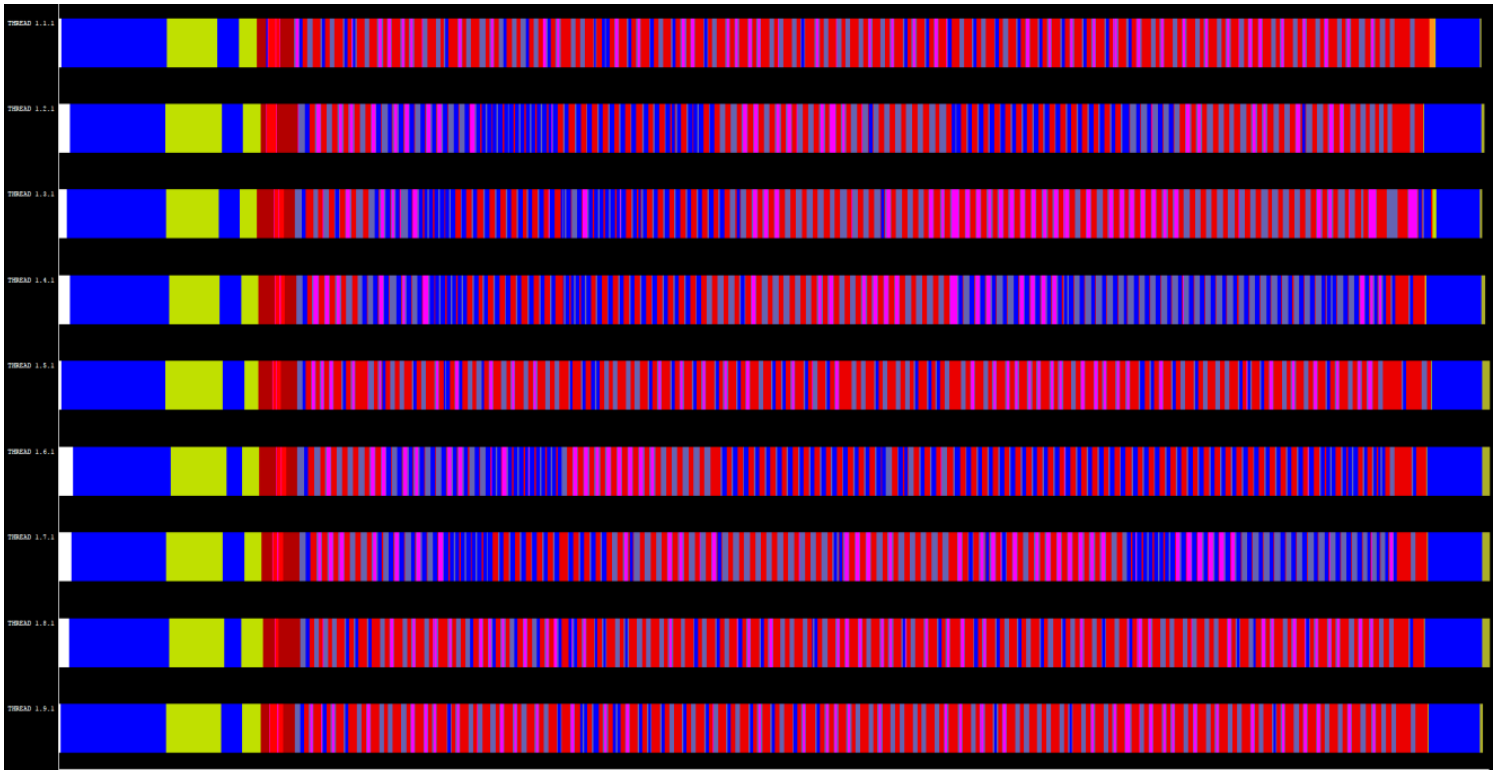
Για τη συγκεκριμένη εκτέλεση χρησιμοποιήθηκε πίνακας μεγέθους 840x840 και 4 διεργασίες όπου και σημειώθηκε το μεγαλύτερο efficiency.
Από το πρώτο διάγραμμα βλέπουμε πως υπάρχει αυξημένη επικοινωνία (κίτρινες γραμμές) μεταξύ των διεργασιών, καθώς σε κάθε γενεά η κάθε διεργασία στέλνει και λαμβάνει μηνύματα από 8 διεργασίες.
Από το δεύτερο διάγραμμα παρατηρούμε πως οι παράγοντες που επιβραδύνουν το πρόγραμμά είναι το κόκκινο χρώμα (waiting a message) το οποίο είναι αρκετά περιορισμένο. Επίσης βλέπουμε πως υπάρχουν αρκετά κομμάτια μπλε χρώμματος (running) κατά τη διάρκεια των οποίων γίνεται ο υπολογισμός του πίνακα της επόμενης γενεάς από κάθε διεργασία. Τέλος, τα χρώματα που επικρατούν μαζί με το μπλε είναι το ροζ (immediate send) και το γκρι (immediate receive) τα οποία μας δείχνουν πως οι ανταλλαγές των μηνυμάτων μεταξύ των διεργασιών γίνονται αμέσως και δεν προσθέτουν καθυστέρηση στην επικοινωνία, χάρη στη χρήση Isend και Ireceive.





Οι παρακάτω δύο πίνακες αντιστοιχούν σε μέγεθος πίνακα 840x840 και 9 διεργασίες.
Συγκριτικά με την προηγούμενη εκτέλεση (4 διεργασίες) παρατηρούμε πως η επικοινωνία μεταξύ των διεργασιών αυξάνεται αισθητά.
Παρόλα αυτά, το γεγονός ότι ο αριθμός των immediate send αυξήθηκε αρκετά σε σχέση με την προηγούμενη εκτέλεση είχε σαν αποτέλεσμα την αύξηση του speed up και την επίτευξη του υψηλότερου efficiency (75%).





Χρόνοι με Mpi_AllReduce: (Τα παρακάτω αποτελέσματα είναι με χρήση 11 μηχανημάτων)

Speed-up	210	420	840	1680	3360
1	1	1	1	1	1
4	1.9	2.55	2.6	2.5	1.14
9	2.6	3.4	5.15	5.5	4.08
16	0.33	1.02	4.12	5.5	5.7
25			0.6	3.15	6.4
36			0.4	1.6	6.7
49			0.31	1.2	6.6

χρόνοι	210	420	840	1680	3360
1	0.13	0.51	2.06	8.2	33
4	0.07	0.2	0.8	3.3	29
9	0.05	0.15	0.4	1.5	8.1
16	0.4	0.5	0.5	1.5	5.8
25	2.7	2.9	3.5	3	5.5
36	5.4	5.8	5.7	5.1	4.9
49	6.2	6.5	6.6	6.8	5

efficiency	210	420	840	1680	3360
1	1	1	1	1	1
4	0.48	0.63	0.65	0.62	0.30
9	0.3	0.38	0.58	0.61	0.50
16	0.02	0.06	0.26	0.34	0.40
25			0.24	0.13	0.30
36			0.01	0.04	0.19
49			0.006	0.05	0.14

Από τα παραπάνω αποτελέσματα προκύπτει ότι η προσθήκη του Mpi_AllReduce δεν επιτάχυνε τη διαδικασία υπολογισμού καθώς στις περισσότερες περιπτώσεις η επιτάχυνση που παρατηρήθηκε ήταν μικρότερη από εκείνη το απλού Mpi.
Να σημειωθεί βέβαια ότι οι χρόνοι αυτοί προέκυψαν μετά τη χρήση 11 μηχανημάτων και όχι 14 όπως στο από Mpi καθώς αυτά υπήρχαν διαθέσιμα την περίοδο που κάναμε τους παραπάνω υπολογισμούς.

Χρόνοι με Mpi+OpenMp: (Τα παρακάτω αποτελέσματα είναι με χρήση 11 μηχανημάτων)

χρόνοι	210	420	840	1680	3360
1	0.13	0.51	2.06	8.2	33
4	0.06	0.2	0.8	3.1	28
9	0.03	0.1	0.35	1.4	6.8
16	0.3	0.2	0.5	1.3	5.3
25	1.1	1.2	1.2	1.3	4.3
36	1.3	1.3	1.35	1.3	3.9
49	1.4	1.5	1.5	1.6	3.5

Speed-up	210	420	840	1680	3360
1	1	1	1	1	1
4	2.17	2.6	3.4	2.7	1.18
9	4.3	5.1	5.9	5.9	4.9
16	0.43	2.6	4.12	6.3	6.3
25			1.71	6.3	7.7
36			1.5	6.3	8.5
49			1.4	5.13	9.4

efficiency	210	420	840	1680	3360
1	1	1	1	1	1
4	0.54	0.65	0.85	0.68	0.30
9	0.47	0.56	0.66	0.65	0.54
16	0.03	0.16	0.26	0.39	0.24
25			0.07	0.25	0.30
36			0.04	0.18	0.24
49			0.03	0.10	0.19

Συγκρίνοντας το efficiency του απλού Mpi με αυτό του Mpi+OpenMp παρατηρούμε ότι για 4 και 9 διεργασίες το efficiency του δεύτερου είναι καλύτερο ενώ για 16-49 διεργασίες τα δύο efficiency είναι περίπου ίδια. Αυτό συμβαίνει πιθανώς διότι τα αποτελέσματα του Mpi+OpenMp προέκυψαν από την εκτέλεση σε 11 μήχανήματα έναντι 14 του απλού Mpi.

Συνεπώς η χρήση OpenMp βελτιώνει σε γενικές γραμμές τους χρόνους εκτέλεσης

CUDA:

Γενικές παρατηρήσεις για το σχεδιασμό:

Για την αναπαράσταση του πλέγματος έχει χρησιμοποιηθεί ένας μονοδιάστατος πίνακας (τόσο στη cpu όσο και στην gpu) με σκοπό την αποθήκευση κάθε γενεάς σε συνεχόμενες θέσεις μνήμης.
Για δεδομένο μέγεθος προβλήματος (έστω NxN) χρησιμοποιείται πίνακας μεγέθους (N+2)x(N+2), προσθέτοντας έτσι 2 γραμμές και 2 στήλες, δηλαδή μία γραμμή πριν την πρώτη γραμμή και μία γραμμή μετά την τελευταία και αντίστοιχα για τις στήλες.
Οι επιπρόσθετες αυτές γραμμές/στήλες περιγράφονται στον κώδικα ως "**virtual**" και σκοπό έχουν την αναπαράσταση ενός κυκλικού πλέγματος εξασφαλίζοντας την ύπαρξη 8 γειτόνων για τα στοιχεία που βρίσκονται στο περιθώριο του πίνακα.
Ο εσωτερικός πίνακας αρχικοποιείται τυχαία με 0/1 ενώ οι virtual γραμμές/στήλες με 0.
Σε κάθε επανάληψη και πριν τον υπολογισμό της επόμενης γενεάς γίνεται ενημέρωση των virtual γραμμών/στηλών με τις kernel συναρτήσεις **updateVirtualRows**, **updateVirtualColumns** και **updateVirtualCorners.**

Για τον υπολογισμό της επόμενης γενεάς χρησιμοποιείται η kernel συνάρτηση **calculateNextGen.**
Για να επιταχύνουμε τον υπολογισμό χρησιμοποιούμε shared memory μεταξύ των νημάτων του ίδιου block.
Έτσι, αρχικά δημιουργούμε ένα τοπικό πίνακα (local) σε κάθε block μεγέθους (BLSIZE*CELLS_PER_THREAD+2)x(BLSIZE*CELLS_PER_THREAD+2) στον οποίο αντιγράφουμε από τη global memory το κατάλληλο κομμάτι του αρχικού πίνακα.
Στη συνέχεια, χρησιμοποιούμε για το συγχρονισμό των νημάτων ένα **__syncthreads()** ώστε να έχει ολοκληρωθεί η μεταφορά των δεδομένων από τον global στους local πίνακες πριν αρχίσει ο υπολογισμός των ζωντανών γειτόνων.
Τέλος με βάση τους 8 γείτονες γίνεται ο υπολογισμός της επόμενης κατάστασης κάθε στοιχείου και ενημερώνεται κατάλληλο το στοιχείο στον πίνακα της επόμενης γενεάς.

Οδηγίες εκτέλεσης:

Αναφορικά με τα δεδομένα παραμετροποίησης του προβλήματος, τροποποιήσεις μπορούν να γίνουν από το αρχείο **header.cuh.**
Συγκεκριμένα στο αρχείο αυτό υπάρχει το μέγεθος του προβλήματος (**N**), ο αριθμός των γενεών (**GENERATION_NUM**), η σταθερά **CELLS_PER_THREAD** που αφορά το μέγεθος των δεδομένων που επεξεργάζεται κάθε thread καθώς και η σταθερά **BLSIZE** που περιγράφει το μέγεθος του κάθε block από threads στην gpu.

Generation number = 1000
Cell per Thread = 1
Block Size = 16

Table Size	Time (msec)	Bandwith (GB/sec)
100	11.38	54.497
500	76.67	195.909912
1000	242.45	246.831 GB
2000	906.29	263.597

Generation number = 1000
Cell per Thread = 2
Block Size = 16

Table Size	Time (msec)	Bandwith (GB/sec)
100	16.83	36.838
500	78.78	190.663
1000	256.81	250.570
2000	911.92	266.327

<u>Σχολιασμός αποτελεσμάτων και σύγκριση με απλό Mpi:</u>

Συμπερασματικά, δεδομένης της μορφής του προβλήματος Game Of Life καθώς και των χρόνων εκτέλεσης τόσο με τη χρήση απλού Mpi όσο και με τη χρήση Cuda, προκύπτει ότι το πρόβλημα ταιριάζει καλύτερα στο περιβάλλον Cuda.
Αυτό προκύπτει από το γεγονός ότι ο χρόνος εκτέλεσης που πετύχαμε το καλύτερο efficiency με απλό Mpi (size: 840x840, generation_number: 100, execution_time: 0.3sec) είναι αρκετά μεγαλύτερος από τον αντίστοιχο χρόνο εκτέλεσης με Cuda (size: 1000x1000, generation_number: 1000, execution_time: 0.242sec) αν αναλογιστούμε ότι ο αριθμός των γενεών στην εκτέλεση του Cuda είναι 10 φορές μεγαλύτερος από αυτόν του Mpi.

Πιο συγκεκριμένα, το γεγονός ότι το παιχνίδι σε κάθε γενεά απαιτεί υπολογισμό απλών πράξεων, δηλαδή 8 προσθέσεων, καθώς και ελάχιστους ελέγχους, καθιστά το Cuda καλύτερη επιλογή. Αυτό συμβαίνει, διότι το κάθε thread στην gpu μπορεί να εκτελεί πολύ γρήγορα απλές πράξεις αλλά καθυστερεί αρκετά στον υπολογισμό συνθηκών.
Επίσης, η εκτέλεση του παιχνιδιού σε παράλληλα περιβάλλοντα καθιστά απαραίτητη την αυξημένη επικοινωνία μεταξύ των διαφορετικών επεξεργαστών/νημάτων. Η διαδικασία αυτή καθυστερεί αισθητά σε περιβάλλοντα όπως το Mpi όπου οι επεξεργαστές ‘’απέχουν’’ αρκετά και οι ανταλλαγές μηνυμάτων κοστίζουν ενώ σε περιβάλλοντα όπως το Cuda όπου οι μεταφορές δεδομένων μεταξύ της shared και της global memory γίνονται πολύ πιο αποτελεσματικά.