Παράλληλα Συστήματα
Εργασία 2016-17: Conway's Game of Life

Ομάδα:

Παναγιώτης Κοκκινάκος - 1115201400069
Θοδωρής Στέφου - 1115201400193
Γιώργος Ρούβαλης - 1115201400173

Περιεχόμενα:

Εισαγωγή

Την εργασία υλοποιήσαμε 3 άτομα ,γι' αυτό το λόγο, ως έξτρα εργασία, φτιάχνουμε αρχεία .txt με την αρχική δομή του πίνακα που διαμοιράζονται με parallel I/O (δεν τα διαμοιράζει δηλαδή μια διεργασία στις υπόλοιπες). Έχουν γραφτεί 2 , λοιπόν, επιπλέον προγράμματα i) RandomArrayGenerator και ii) ConvertVert. Το πρώτο δημιουργεί ένα αρχείο με μια σειρά και τον πίνακα από 0 και 1 για το κύριο πρόγραμμα και το δεύτερο, δίνει τη δυνατότητα να παραχθεί ένα αρχείο παίρνοντας σαν είσοδο την έξοδο του RandomArrayGenerator και να το παρουσιάσει κάθετα, σαν ένα κανονικό nxn πίνακα.
Επίσης, έχει γραφτεί και το πρόγραμμα της εργασίας σε σειριακή μορφή, ώστε να χρησιμοποιηθεί για τον υπολογισμό των μεγεθών speedup και efficiency.

Κάθε διεργασία αφού ανοίξει το αρχείο εισόδου, θέτει την “όψη” του αρχείου (συνάρτηση setview()) έτσι ώστε όποτε διαβάζει από το αρχείο να παίρνει μόνο τα δεδομένα που την αφορούν. Το διάβασμα από το αρχείο γίνεται με iread() ώστε να έχουμε τη δυνατότητα να αρχικοποιήσουμε παραμέτρους κλπ μέχρι να τελειώσει, δηλαδή το I/O είναι non-blocking.

Το όνομα του αρχείου από το οποίο διαβάζεται η αρχική κατάσταση του πίνακα, ορίζεται με #define στην αρχή του προγράμματος, όπως επίσης και η διάσταση του πίνακα με τα κελιά.

Σχεδιασμός Διαμοιρασμού Δεδομένων

Έχει δημιουργηθεί μία καινούρια τοπολογία ώστε το σύστημα να υπολογίζει τον καλύτερο διαμοιρασμό τον διεργασιών στους πυρήνες για τη βέλτιστη επικοινωνία μεταξύ τους.
Η τοπολογία δημιουργεί ένα δισδιάστατο πλέγμα το οποίο είναι περιοδικό προς τα πάνω, κάτω, αριστερά, και δεξιά.
Κάθε MPI διεργασία που δημιουργείται αναλαμβάνει ένα block του συνολικού πίνακα. Άρα, ο πίνακας χωρίζεται σε blocks, ίσα στο πλήθος με τις διεργασίες. Κάθε διεργασία έχει έναν ιδιωτικό πίνακα ο οποίος έχει μέγεθος όσο και τα κελιά για τα οποία είναι υπεύθυνη συν μια εξωτερική περίμετρο στην οποία φυλάγονται τα κελιά των γειτονικών block τα οποία χρειάζονται για τον υπολογισμό της επόμενης γενιάς των κελιών που βρίσκονται στην περίμετρο του πραγματικού block της διεργασίας.
Κάθε διεργασία σε κάθε επανάληψη γεμίζει όλη την έξτρα περίμετρο λαμβάνοντας τα δεδομένα από τα 8 γειτονικά της block.

Σχεδιαστικές Επιλογές

Έχουμε δημιουργήσει δύο καινούρια Data Types από τα οποία το ένα αναπαριστά τις γραμμές και το άλλο τις στήλες, ώστε να γίνουν οι ανταλλαγές των δεδομένων με μία μόνο κλήση συνάρτησης και χωρίς περιττές αντιγραφές στοιχείων. Για την προετοιμασία της αποστολής των δεδομένων χρησιμοποιούμε την συνάρτηση MPI_Send_init() ενώ για την προετοιμασία της λήψης την MPI_Recv_init() επειδή οι παράμετροι δεν αλλάζουν σε κανένα σημείο του κώδικα. Έχουν γίνει δύο προετοιμασίες αποστολής και λήψης, μία για τον πίνακα a και μία για τον πίνακα b, γιατί σε κάθε επανάληψη αλλάζει η διεύθυνση στην οποία γράφονται τα εξωτερικά βοηθητικά κελιά.

Όταν καλείται η συνάρτηση StartAll() που σηματοδοτεί την έναρξη της επικοινωνίας μεταξύ των διεργασιών δεν περιμένουμε να τελειώσει η επικοινωνία δηλαδή να γεμίσει η περίμετρος του πίνακα κάθε διεργασίας, αλλά μέχρι να λάβει τα δεδομένα κάνει τους εσωτερικούς υπολογισμούς. Όταν υπολογιστούν τα εσωτερικά κελιά, οι διεργασίες συγχρονίζονται -μπλοκάρουν μέχρι να τελειώσουν όλες οι επικοινωνίες και μετά υπολογίζεται και η περίμετρος του πραγματικού πίνακα. (όχι η έξτρα δηλαδή)

Σε κάθε βήμα της επανάληψης κάθε διεργασία μετράει τον αριθμό των ζωντανών κελιών του block της στην επόμενη γενιά. Επίσης κάθε διεργασία έχει ένα flag το οποίο γίνεται 1 αν κάποιο κελί έγινε 0 στην επόμενη γενιά ενώ ήταν 1 ή το αντίστροφο. Κάθε 10 επαναλήψεις, πραγματοποιείται ένα AllReduce μέσω του οποίου διαμοιράζεται σε όλες τις διεργασίες το άθροισμα των ζωντανών κελιών, και το άθροισμα των προαναφερθέντων flags. Αν το άθροισμα των ζωντανών κελιών είναι 0, τότε τερματίζεται το πρόγραμμα. Επίσης αν το άθροισμα των flags είναι 0, σημαίνει ότι σε καμία διεργασία δεν άλλαξε η κατάσταση κάποιου κελιού, άρα το πρόγραμμα τερματίζει και πάλι. Ο λόγος που δεν συγκρίνουμε το άθροισμα των κελιών στην προηγούμενη και επόμενη γενιά,και αν είναι ίδιο να τερματίσουμε το πρόγραμμα, είναι επειδή υπάρχουν περιπτώσεις όπου ο αριθμός των ζωντανών κελιών μένει ίδιος, αλλά ο πίνακας έχει αλλάξει, δηλαδή έχουν δημιουργηθεί τόσα κελιά όσα είναι αυτά που έχουν πεθάνει.

Στο τέλος του υπολογισμού κάθε γενιάς, όλες οι διεργασίες γράφουν σε ένα κοινό αρχείο την κατάσταση της γενιάς αυτής και πάλι με setview().

Μετρήσεις

Οι πίνακες με τις μετρήσεις χρόνων, speedup και efficiency υπάρχουν σε ξεχωριστό αρχείο. Έχουν γίνει μετρήσεις για διαστάσεις (τετραγωνικών) πινάκων 240,480,960,1920,3840, για 4 9 και 16 διεργασίες και για κάθε έναν από τους συνδυασμούς αυτούς έχει γίνει μέτρηση και με τη χρήση AllReduce και χωρίς. Λόγω της κατάστασης των μηχανημάτων και του γεγονότος ότι την χρονική στιγμή που έγιναν οι μετρήσεις (Σάββατο 7-10), τα μηχανήματα χρησιμοποιούνταν από πολλά άτομα για τον ίδιο ακριβώς λόγο, οι μετρήσεις για τα MPI προγράμματα (και Open MP παρακάτω) δεν είναι αντιπροσωπευτικές, συγκρίνοντάς τις βέβαια και με μετρήσεις που έχουν γίνει στα δικά μας μηχανήματα, όπου εκεί παρουσιάζεται θετική επιτάχυνση και αποτελεσματικότητα και ισχυρή κλιμάκωση.

Για σχετικά μικρά δεδομένα, όπως παραδείγματος χάρη τον πίνακα 240x240, η παραλληλοποίηση δεν προσφέρει κανένα κέρδος και δεν συμφέρει επειδή το κόστος επικοινωνίας είναι πολύ μεγαλύτερο από τον χρόνο που κερδίζουμε λόγω της παράλληλης εκτέλεσης.

Για διαστάσεις πίνακα από 960x960 και πάνω, όσο αυξάνονται οι διεργασίες μειώνεται και ο χρόνος, δηλαδή αυξάνεται το speedup για τον αριθμό πυρήνων που εξετάστηκε.

Γενικά, το κόστος επικοινωνίας λόγω του Reduce αυξάνεται, άρα και ο χρόνος και αυτό φαίνεται στις περισσότερες από τις μετρήσεις που κάναμε.

MPI + OpenMP

Ο κώδικας για το MPI + OpenMP είναι ίδιος με αυτόν του MPI, με τη διαφορά ότι παραλληλοποιούμε τις επαναλήψεις στις οποίες βρίσκουμε τον αριθμό των γειτόνων ενός κελιού και υπολογίζουμε την επόμενη κατάστασή του. Μετά από δοκιμές καταλήξαμε ότι σχεδιαστικά είναι καλύτερα να χρησιμοποιήσουμε κυκλική ανάθεση των επαναλήψεων στα threads με τη χρήση του schedule(). Για τον υπολογισμό του συνολικού αθροίσματος των ζωντανών κελιών στην επόμενη κατάσταση και της τιμής της flag που καθορίζει αν άλλαξε η κατάσταση κάποιου κελιού, χρησιμοποιείται το reduction του OpenMP σε επίπεδο thread, και όπως πριν, το AllReduce του MPI. Για μείωση του overhead του reduction του open mp, η μεταβλητή που θα γίνει reduce δεν προσπελαύνεται κάθε φορά που ικανοποιούνται τα αντίστοιχα κριτήρια για να αυξηθεί (στην περίπτωση του αθροίσματος) ή να γίνει 1 (στην περίπτωση του flag), αλλά κάθε thread κρατάει private μεταβλητές για τον σκοπό αυτό, και στο τέλος του for προσπελαύνονται και αλλάζουν οι reduced μεταβλητές 1 μόνο φορά.

OpenMP Μετρήσεις

Για τις μετρήσεις του OpenMP+MPI, λόγω του ότι όπως ειπώθηκε στο μάθημα πρέπει να αναθέσουμε σε κάθε μηχάνημα 1 διεργασία μέσω του machines file, αλλά τα διαθέσιμα μηχανήματα για OpenMP είναι 7, χρησιμοποιήσαμε μόνο 1 και 4 διεργασίες MPI. Επίσης χρησιμοποιήσαμε τους ίδιους πίνακες με το MPI, 2,4 και 8 OpenMP threads και ο συνδυασμός των παραπάνω έγινε και με τη χρήση reduce και χωρίς. Για τους ίδιους λόγους που αναφέρθηκαν στο MPI οι μετρήσεις δεν είναι αντιπροσωπευτικές, και δεν μπορούν να βγουν τα σωστά συμπεράσματα. Υπήρχαν για παράδειγμα περιπτώσεις στις οποίες για τα ίδια ακριβώς δεδομένα, διεργασίες και OpenMP threads, χωρίς καμία αλλαγή σε καμία παράμετρο, μια μέτρηση έπαιρνε παραπάνω από 120 δευτερόλεπτα για να τελειώσει, και άλλη μέτρηση κάτι λιγότερο από 3 δευτερόλεπτα.

Για μικρά δεδομένα (όπως π.χ. 240x240) είναι προτιμότερο να έχουμε 1 διεργασία MPI και 4 OpenMP threads γιατί ο χρόνος που κάνουν τα threads για να δημιουργηθούν είναι μεγαλύτερος από τον χρόνο που εξοικοινομείται με τη χρήση τους.

Για μεσαίου μεγέθους δεδομένα (όπως π.χ. 1920x1920) είναι προτιμότερο να έχουμε 4 διεργασίες MPI και 4 OpenMP threads.

Για μεγάλου μεγέθους δεδομένα (όπως π.χ. 3840x3840) είναι καλό να χρησιμοποιούμε αρκετά μεγάλο αριθμό MPI διεργασιών και OpenMP threads.

Σχετικά με το reduction, επειδή πρέπει να προστεθεί και reduction σε επίπεδο thread , και υπάρχει ήδη reduction σε επίπεδο διεργασιών MPI, οι χρόνοι αυξάνονται πολύ σε σχέση με όταν το πρόγραμμα τρέχει χωρίς reduction.

Cuda

Για την υλοποίηση του προγράμματος σε cuda ακολουθήσαμε την ίδια προσέγγιση που ακολουθήσαμε και στο ακολουθιακό, δηλαδή για να επιτύχουμε την περιοδικότητα στον πίνακα προσθέσαμε μια έξτρα περίμετρο. Στο host δημιουργούμε ένα μονοδιάστατο πίνακα με την έξτρα περίμετρο, ενώ στο device αποθηκεύουμε 2 πίνακες (επίσης με την έξτρα περίμετρο), από τους οποίους ο πρώτος αναπαριστά την προηγούμενη γενιά του παιχνιδιού και ο δεύτερος την επόμενη. Έχουμε δημιουργήσει 3 kernels που εκτελούνται στην gpu. Οι 2 πρώτοι καλούνται στην αρχή κάθε επανάληψης, έτσι ώστε τα threads που δημιουργούνται να γεμίσουν τις γραμμές και τις στήλες της έξτρα περιμέτρου του πίνακα αντίστοιχα. Ύστερα καλείται ο kernel ο οποίος περιέχει το βασικό βήμα της προσομοίωσης. Κάθε block αποτελείται απο 256 threads, και είναι δισδιάστατο ώστε να ταιριάζει στη φύση του προβλήματος. Όταν κληθεί, κάθε thread υπολογίζει το μοναδικό id του σε σχέση με όλα τα thread των block, και το id του σε σχέση με τα threads του block στο οποίο βρίσκεται. Σε κάθε block υπάρχει shared memory, έτσι ώστε ο υπολογισμός γειτόνων κάθε κελιού να γίνεται γρηγορότερα, αφού με αυτόν τον τρόπο το thread δε διαβάζει συνεχώς από τη global memory, αλλά απο τη shared. Στην αρχή του kernel κάθε block, εφ'όσον είναι στα επιθυμητά όρια του πίνακα, φορτώνει στην shared memory από την global memory, την τιμή του κελιού που του αντιστοιχεί. Όταν το κάνουν αυτό όλα τα threads του ίδιου block, κάθε thread υπολογίζει την επόμενη κατάσταση του κελιού για το οποίο είναι υπεύθυνο. Όταν η επόμενη γενιά του πίνακα έχει υπολογιστεί, ο πίνακας που την αναπαριστά αντιγράφεται στον πίνακα που υπάρχει στον host, αυτός αντιγράφεται σε ένα αρχείο και προχωράμε στην επόμενη επανάληψη. Έχει προστεθεί και υλοποίηση του reduction χρησιμοποιώντας την συνάρτηση atomicAdd(), για να δειχθεί ότι η χρήση της αυξάνει κατά πολύ το χρόνο του προγράμματος, ειδικά όταν χρησιμοποιείται από πολλά threads , αφού το πρόγραμμα έτσι σειριοποιείται σε πολύ μεγάλο βαθμό. Για να γίνει αυτό αντιγράφονται στην gpu 2 επιπλέον int μεταβλητές, από τις οποίες η μία μετράει τον αριθμό των ζωντανών κελιών στην επόμενη γενιά, και η άλλη αν δεν υπήρξε καμία αλλαγή κατάστασης σε κανένα κελί του πίνακα (όπως ακριβώς δηλαδή και στο MPI και OpenMP). Ο έλεγχος είναι και πάλι ο ίδιος, δηλαδή το πρόγραμμα τερματίζει πρόωρα αν κάποια από αυτές τις μεταβλητές είναι 0.

Cuda Μετρήσεις

Οι μετρήσεις cuda έγιναν με κάρτα GeForce GT 240 η οποία διαθέτει 96 πυρήνες cuda, και το speedup βρέθηκε σχετικά με μετρήσεις του σειριακού προγράμματος οι οποίες έγιναν στον ίδιο υπολογιστή, για μεγαλύτερη αντικειμενικότητα. Χρησιμοποιήθηκαν οι ίδιοι πίνακες όπως και στο MPI και OpenMP, δηλαδή με διαστάσεις 240,480,960,1920 και 3840. Αρχικά έγιναν μετρήσεις χρησιμοποιώντας 256 threads ανά block και ύστερα με 1024 threads ανα μπλοκ. Επίσης έγιναν μετρήσεις χρησιμοποιώντας το reduction με το atomicAdd() όπως περιγράφηκε πιο πάνω.

Παρατηρούμε ότι στις μετρήσεις χωρίς reduction με 256 threads/block , υπάρχει μεγάλο speedup το οποίο ανάλογα με τη διάσταση του πίνακα κυμαίνεται από 14 έως και 25. Όσο αυξάνεται το μέγεθος του πίνακα τόσο αυξάνεται και το speedup. Όταν χρησιμοποιούμε 1024 threads ανά μπλοκ παρατηρούμε ότι το speedup είναι τουλάχιστον διπλάσιο, και ότι, όπως και πριν, όσο αυξάνεται το μέγεθος των δεδομένων τόσο το speedup αυξάνεται.

Όταν χρησιμοποιούμε reduction, εξαιτίας της atomicAdd() το πρόγραμμα σειριοποιείται σε πολύ μεγάλο βαθμό. Οι μετρήσεις έγιναν για διαστάσεις 240,480,960 και 256 threads ανά block , γιατί για μεγαλύτερα δεδομένα παρουσιαζόταν πρόβλημα στον υπολογιστή λόγω του χρόνου που χρειαζόταν για τον τερματισμό του προγράμματος. Παρατηρούμε ότι και για τις 3 διαστάσεις, το speedup είναι αρνητικό, άρα συμπεραίνουμε ότι η χρήση της atomicAdd() δεν πρέπει να γίνεται συχνά, ειδικά όταν καλείται από όλα τα threads, γιατι έχουμε τα αντίθετα αποτελέσματα από αυτά που θέλουμε.

Συμπεράσματα

(Τα συμπεράσματα είναι με βάση τις μετρήσεις MPI και OpenMP που έγιναν στα μηχανήματα της σχολής, οι οποίες όπως έχει ήδη προαναφερθεί δεν είναι αντιπροσωπευτικές λόγω του μεγάλου αριθμού ατόμων που πραγματοποιούσαν μετρήσεις την ίδια χρονική περίοδο.)

Γενικά, παρατηρούμε ότι ανάμεσα στο MPI, MPI+OpenMP και Cuda, τη μεγαλύτερη επιτάχυνση την επιτυγχάνουμε χρησιμοποιώντας το τελευταίο. Και στις 3 περιπτώσεις η χρήση reduction καθυστερεί την εκτέλεση του προγράμματος, ειδικά στην περίπτωση του OpenMP+MPI όπου γίνεται σε 2 επίπεδα: σε επίπεδο thread και σε επίπεδο διεργασιών, και καλό θα ήταν να αποφεύγεται. Η χρήση μεγάλου πλήθους διεργασιών MPI πρέπει να γίνεται σε συνδυασμό με μεγάλα δεδομένα, αλλιώς αν χρησιμοποιούμε πολλές διεργασίες για μικρο πλήθος δεδομένων, το χρονικό κόστος επικοινωνίας ανάμεσά τους είναι μεγαλύτερο από το χρόνο που εξοικονομείται με τη χρήση τους.

Η χρήση του OpenMp σε συνδυασμό με το MPI επιταχύνει την εκτέλεση του προγράμματος στις περισσότερες περιπτώσεις, Πρέπει όμως σε μικρά δεδομένα να μην χρησιμοποιούμε μεγάλο πλήθος OpenMp threads, γιατί το κόστος δημιουργίας και καταστροφής τους είναι μεγαλύτερο από το χρόνο που εξοικονομείται Αντίθετα σε μεγάλα δεδομένα, μεγαλύτερο πλήθος OpenMP threads επιταχύνει την εκτέλεση του προγράμματος. Γι’αυτό ανάλογα με τα δεδομένα μας καλό θα ήταν να ελέγχουμε κάθε συνδυασμό MPI διεργασιών και OpenMP threads ώστε να καταλάβουμε ποιος είναι ο πιο αποτελεσματικός.

Γενικά-Οδηγίες

Όπως προαναφέρθηκε, στις μετρήσεις δεν συμπεριλάβαμε το βήμα στο οποίο κάθε γενιά γράφεται σε αρχείο, για μεγαλύτερη αντικειμενικότητα και αληθοφάνεια. Οι σχετικές γραμμές κώδικα έχουν σχολιαστεί, όπως και οι σχετικές δηλώσεις μεταβλητών που χρησιμοποιούνται σε αυτά τα σημεία. Ο πηγαίος κώδικας είναι χωρισμένος σε φακέλους για MPI, OpenMP και Cuda, υπάρχει φάκελος με τα test files που χρησιμοποιήθηκαν για τις μετρήσεις, όπως επίσης και τα βοηθητικά προγράμματα που αναφέρθηκαν στην εισαγωγή. Επίσης έχουμε συμπεριλάβει και τον φάκελο που προκύπτει μετά από εκτέλεση του προγράμματος χρησιμοποιώντας τον πίνακα 240x240, που περιέχει και τις 150 γενιές του πίνακα σε τετραγωνική μορφή.

Όλα τα προγράμματα τρέχουν χρησιμοποιώντας σαν input τον πίνακα με διάσταση 960. Το MPI με την εκτέλεσή του δημιουργεί φάκελο στον οποίο γράφονται σε αρχεία .txt όλες οι γενιές του πίνακα. Σε όλα τα υπόλοιπα προγράμματα οι σχετικές γραμμές έχουν σχολιαστεί.

Σειριακό: Μεταγλωττίζεται με make και η εκτέλεση γίνεται ως εξής : ./golSerial

MPI: Έχουμε συμπεριλάβει και τις γραμμές κώδικα με τις οποίες γίνεται το reduction. Μεταγλωττίζεται με make και η εκτέλεση γίνεται ως εξής : mpiexec -f machines -n <Processes> ./golMpi

OpenMP : Δεν έχουμε συμπεριλάβει τις γραμμές κώδικα με τις οποίες γίνεται το reduction. Μεταγλωττίζεται με make και η εκτέλεση γίνεται ως εξής : mpiexec -f machines -n <Processes> ./golOpenMP

Cuda: Δεν έχουμε συμπεριλάβει τις γραμμές κώδικα με τις οποίες γίνεται το reduction. Μεταγλωττίζεται με make και η εκτέλεση γίνεται ως εξής : ./golCuda

Οι μετρήσεις βρίσκονται σε έγγραφο excel, το οποίο υπάρχει στο φάκελο, αλλά μπορείτε και να το βρείτε στο παρακάτω link:
https://docs.google.com/spreadsheets/d/1Q2sFb9o32b08Hrz6mlxgcG2f9TSaxE9CqQl2_pok18k/edit#gid=1626113614

Υπάρχουν 8 διαφορετικά sheets: 2 για κάθε ένα από τα MPI, OpenMP και Cuda, από τα οποία το 1 περιέχει μετρήσεις με το reduction και το άλλο χωρίς, και 2 sheets με μετρήσεις για το σειριακό πρόγραμμα από τα οποία το 1 αφορά τις μετρήσεις στα μηχανήματα της σχολής και το άλλο τις μετρήσεις σε δικό μας μηχάνημα.

Παραπάνω παρατηρήσεις και λεπτομέρειες για τον κώδικα υπάρχουν μέσα στα πηγαία αρχεία με τη μορφή σχολίων.