OpenMP

Πρόσθεσα OMP directives για πολυνηματική εκτέλεση τόσο στη MPI όσο και στη σειριακή υλοποίηση. Για 4 επεξεργαστές με ένα thread ανά επεξεργαστή η εκτέλεση διαρκεί κατά μέσο όρο 414940 μs ενώ με 2 νήματα ανά επεξεργαστή διαρκεί 303261 μs, παρατηρούμε δηλαδή 37% επιτάχυνση. Για ένα επεξεργαστή με ένα thread ο χρόνος είναι 1207244 ενώ με δύο threads πέφτει στα 609050 μs, σχεδόν στο μισό. Η διαφορά είναι τόσο μεγάλη σε σχέση με τους 4 επεξεργαστές επειδή στη σειριακή υλοποίηση δεν έχουμε το overhead του message passing και άρα όποιο overhead έχουμε είναι ελάχιστο μπροστά στον όγκο της παραλληλοποιήσιμης με νήματα δουλειάς. Για τα παραπάνω αποτελέσματα χρησιμοποιήθηκαν grayscale 2520*960 εικόνα και 5 iterations. Για την MPI υλοποίηση χρειάστηκε να αλλαχθούν το πλήθος των διεργασιών ανά υπολογιστή στο αρχείο machines ώστε να αντιστοιχισθεί κάθε process σε διαφορετικό επεξεργαστή και να επιτραπεί με αυτό τον τρόπο πρόσβαση σε παραπάνω από 1 thread σε κάθε μία τους.

Ακολουθεί το doc του MPI.

Εισαγωγή

Το παρόν πρόγραμμα προορίζεται για εκτέλεση σε linux cluster με εγκατεστημένη την MPICH υλοποίηση του MPI.

Για compile χρησιμοποιούμε το συνοδευόμενο makefile. Για τρέξιμο στο linux cluster της σχολής: mpiexec –machinefile machines –np <num_of_cores> executable <input's file name> <output's file name> <input height> <input width> <*superblock's edge's size*[1]> <num of iterations> <num of colors>

Όπου num_of_colors είναι 1 για grayscale και 3 για RGB και πρέπει input height >= input width. Το input file πρέπει να βρίσκεται στο τρέχον directory.

Δομή κώδικα

Όσον αφορά την παράλληλη υλοποίηση, το πλέγμα των pixels χωρίζεται σε τετράγωνα superblocks των οποίων το μέγεθος ακμής ορίζεται από command line argument. Τα superblocks χωρίζονται σε τετράγωνα blocks ίσου πλήθους με τις διεργασίες της κάθε εκτέλεσης και σε κάθε διεργασία αντιστοιχείται από ένα block για κάθε superblock (το $1^{ο}$ block κάθε superblock στην $1^{η}$ διεργασία, το $2^{ο}$ στη $2^{η}$ etc.). Η $1^{η}$ διεργασία του MPI communicator ανοίγει το διαβάζει από μια γραμμή superblock τη φορά την οποία διασπείρει (scatter) σε όλες τις διεργασίες του communicator όπως περιγράφεται παραπάνω. Με αυτό τον τρόπο το grid διαμοιράζεται ισόποσα σε όλες τις διεργασίες. Οι διεργασίες κατασκευάζουν 3 ακόμα arrays η κάθε μία στα οποία φυλάνε πολλαπλάσια των στοιχείων του αρχικού τους array όπως προβλέπει το φίλτρο h και στέλνουν μέρος αυτών στις (περιοδικά) καρτεσιανά γειτονικές τους διεργασίες. Αφού ο σχηματισμός των τριών intermediate arrays έχει ολοκληρωθεί η κάθε διεργασία υπολογίζει το κομμάτι που της αναλογεί του νέου grid. Στη συνέχεια

[1] Προτεινόμενο μέγεθος = 120. Παραλείπεται όταν έχουμε num_of_cores = 1, δηλαδή σειριακή εκτέλεση.

με τη χρήση μίας MPI reduction operation υπολογίζεται και η ποσοστιαία διαφορά του παλαιού και του νέου grid. Η παραπάνω διαδικασία επαναλαμβάνεται όσες φορές έχει ορίσει ο χρήστης μέσω command line argument ή έως ότου η διαφορά παλαιού – νέου grid κριθεί αμελητέα. Τέλος τα κομμάτια του grid όλων των διεργασιών συγκεντρώνονται σε μία η οποία παράγει το anti-aliased αρχείο εξόδου. Όταν το αρχείο εισόδου είναι RGB, τότε εφόσον το no_channels έχει οριστεί ορθά (3) μέσω command line argument, παράγονται 3 προσωρινά αρχεία, ένα για κάθε χρώμα και η παραπάνω διαδικασία εκτελείται χωριστά για κάθε ένα εξ αυτών. Οι έξοδοί τους αντικαθιστούν το περιεχόμενο των $3^{ων}$ προσωρινών αρχείων και στη συνέχεια αυτά ενοποιούνται σε ένα μόνιμο που αποτελεί και το επιθυμητό RGB anti-aliased αποτέλεσμα. (ναι αρκετά άκομψο, συγνώμη)

Για εκτέλεση σε έναν πυρήνα η υλοποίηση είναι αρκετά πιο straightforward χωρίς διαχωρισμό του πλέγματος σε superblocks.

Στις χρονικές μετρήσεις δε συμπεριλαμβάνονται ο χρόνος διαβάσματος και εγγραφής αρχείων και ο χρόνος των scattering και gathering από και προς την reader/printer process της παράλληλης υλοποίησης.

Λεπτομέρειες σχετικές με βελτιστοποίηση

Για τη μεταφορά δεδομένων μεταξύ των διεργασιών χρησιμοποιούνται MPI vectors και indexed types που περιγράφονται με comments στο source. Η πληροφορία κάθε pixel δεσμεύει χώρο size short κατά τη μεταφορά και αποθήκευση στα intermediate arrays και αυτό γιατί δύναται να περιέχουν τιμές μεγαλύτερες του 255 αφού έχουν πολλαπλασιαστεί με 2 ή 4. Κατά τον υπολογισμό των intermediate arrays, υπολογίζονται πρώτα τα στοιχεία που αντιστοιχούν στις γωνίες και τα πλευρικά όρια των blocks (τα οποία και είναι απαιτητά από τις γειτονικές τους διεργασίες για την ολοκλήρωση των δικών τους intermediate arrays. Έτσι ακριβώς μετά τον υπολογισμό μιας πλευράς, στέλνεται στην αντίστοιχη γειτονική διεργασία και ζητείται η απέναντί της αντίστοιχα από την απέναντι διεργασία (π.χ. calculate TOP_LEFT, send TOP_LEFT, receive BOTTOM_RIGHT) και αυτό εκτελείται παράλληλα σε κάθε διεργασία, ταυτόχρονα για όλα τα blocks. Ο υπολογισμός των γωνιών προηγείται της αποστολής των πλευρών διότι σε αυτές περιλαμβάνονται και οι γωνίες. Τα sends και receives είναι non-blocking και ο υπολογισμός των πλευρών των αργότερων εξ αυτών επικαλύπτει τα νωρίτερα αλλά η μεγαλύτερη επικάλυψη γίνεται από τον υπολογισμό των εσωτερικών μερών (δηλαδή εξαιρουμένων των πλευρών) των blocks. Στη συνέχεια έχουμε waitall για όλα τα προηγούμενα sends receives αφού πλέον χρειαζόμαστε τα δεδομένα τους ώστε να προχωρήσουμε στον υπολογισμό των τμημάτων του νέου grid. Το superblock_size δηλαδή το μέγεθος της πλευράς των superblocks είναι καλό να είναι όσο το δυνατό μεγαλύτερο (με την προϋπόθεση ότι διαιρεί το γινόμενο width * height) ώστε ο όγκος (χρόνος) του σειριακού υπολογισμού των εσωτερικών των blocks να υπερβαίνει το χρόνο των μεταφορών. Σε αντίθετη περίπτωση η χρήση περισσότερων πυρήνων μπορεί να καθυστερεί την εκτέλεση αντί να την επισπεύδει. Προτεινόμενο superblock_size για τα μεγέθη εικόνων που ζητά η άσκηση (2520*1920, 2520*960, 2520*480, 5040*1920, 10080*1920) είναι το 120. Το ύψος πρέπει να είναι πάντα μεγαλύτερο από το πλάτος.

Μετρήσεις

Αρχικά η εκτέλεση με grayscale εικόνα ύψους 2520 και πλάτους 1920, με 5 iterations και ένα πυρήνα διήρκησε 2252459 microseconds, με 4 πυρήνες 656718 microseconds ενώ για 9 πυρήνες 505766 microseconds. Επομένως για 4 επεξεργαστές έχουμε speedup = 3.43, efficiency = 0.86 και cost = 2626872 ενώ για 9 επεξεργαστές speedup = 4.45, efficiency = 0.49 και cost = 45681894. Το μεγαλύτερο speedup είχα για 9 επεξεργαστές επομένως έκανα και άλλες μετρήσεις για διαφορετικά μεγέθη εικόνων. Για εικόνα ανάλυσης 5040*1920 διήρκησε 911130 μs, για 10080*1920 1988474 μs, για 2520*960 329841μs και για 2520*480 189412 μs.

2520*480	2520*960	2520*1920	5040*1920	10080*1920
189412 μs	329841μs	505766 μs	911130 μs	1988474 μs

Ο λόγοι αύξησης μεγέθους προς την αύξηση χρόνου εκτέλεσης για κάθε εικόνα και τη διπλάσια διαδοχική της, ξεκινώντας από τη μικρότερη, είναι οι εξής 1.15, 1.3, 1.11 και 0.91. Δεδομένου ότι επιθυμούμαι οι παραπάνω τιμές να είναι ≥ 1 ώστε να έχουμε καλό scaling, παρατηρούμε ότι έχουμε αρχικά άνοδο μέχρι το τέλος που παρατηρείται κακό scaling με 0.91 δηλαδή για διπλασιασμό μεγέθους προβλήματος έχουμε υπερδιπλασιασμό του χρόνου εκτέλεσης.

Αντίστοιχα για RGB εικόνα 2520*1920, 5 iterations με ένα επεξεργαστή είχαμε 7505407 μs, με τέσσερις 2060610 μs και με εννέα 1512068 μs. Επομένως για 4 επεξεργαστές έχουμε speedup = 3.64, efficiency = 0.91 και cost = 8242440 ενώ για 9 έχουμε speedup = 4.96, efficiency = 0.55 και cost = 13608612. Για 9 επεξεργαστές 5 iterations έχουμε τις παρακάτω αναλύσεις και αποτελέσματα:

2520*480	2520*960	2520*1920
546665 μs	1094803 μs	1512068 μs

Από 2520*480 σε 2520*960 έχουμε διπλασιασμό μεγέθους και χρόνου ενώ στη συνέχεια για 2520*1920 όπου πάλι έχουμε διπλασιασμό μεγέθους προβλήματος ο χρόνος αυξάνεται κατά μόλις 38%.

Για κάθε μια από τις παραπάνω μετρήσεις είχα 3, 4 ή 5 εκτελέσεις από τις οποίες κρατούσα το μέσο όρο των χρόνων εκτέλεσης. Δε δοκίμασα για μεγαλύτερες από 2520*1920 RGB εικόνες όπως έκανα για τις grayscale διότι είχα περιορισμένο quota στα linux της σχολής και δε χώραγε κάτι μεγαλύτερο (εδώ για την 2520*1920 χρειάστηκε να σβήσω σχεδόν ό,τι είχα). Οι περισσότεροι επεξεργαστές που κατάφερα να χρησιμοποιήσω ήταν 9 διότι οι υπολογιστές από τον linux07 και πάνω ήταν unresponsive (δοκίμασα και ping και δεν επέστρεφε τίποτα). Η οπτική σύγκριση μεταξύ input και output εικόνων έγινε με το πρόγραμμα ThumbsPlus Pro 10.

Συμπεράσματα

Το πρόγραμμα τρέχει και δίνει ικανοποιητικά αποτελέσματα σε ανεκτό χρόνο. Τα αποτελέσματα που πήρα είναι πιθανώς μη αντιπροσωπευτικά άλλων εκτελέσεων καθώς η επεξεργαστική ισχύς στο cluster είναι κοινόχρηστη και δε δύναμαι να γνωρίζω πόσο load υπάρχει λόγω άλλων χρηστών, άρα και πόση μου αναλογεί κάθε φορά.