UNIVERSITY OF ATHENS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΘΠ04 - **Παράλληλα Συστήματα**

Εργασία - Game of Life

Α.Μ.: 1115201600141 – Βασίλειος Πουλόπουλος

Α.Μ.: 1115201300202 – Κωνσταντίνος Χατζόπουλος

Εισαγωγή

Για την υλοποίηση της εργασίας, δημιουργήσαμε αρχεία εισόδου για τις μετρήσεις. Για κάθε τμήμα της εργασίας δημιουργήσαμε ξεχωριστο φάκελο στον οποίο υπάρχει ο κώδικας, οι μετρήσεις, τα αποτελέσματα του profiling, τα input files και κάποια bash scripts που υλοποιήσαμε για να αυτοματοποιήσουμε τη διαδικασία των compiling, run, συλλογή μετρήσεων και υπολογισμό των speedup και efficiency. Τα scripts αρχικά μεταγλωτίζουν το πρόγραμμα με τις αντίστοιχες παραμέτρους κάθε φορά (μέγεθος προβλήματος N, αριθμό διεργασιών/nodes/cpus/threads/gpus) και στη συνέχεια για να το τρέξουν το τοποθετούν στην ουρά με την εντολή qsub.

Για όσο βρίσκεται στην ουρά και εκτελείται ή περιμένει να εκτελεστεί, το script συνεχώς ελέγχει με την εντολή qstat αν ολοκληρώθηκε το job και αμέσως μετά, με την εντολή append του bash (>>) δημιουργεί (αν δεν υπάρχουν ήδη) αρχεία που περιέχουν μόνο τους χρόνους με όνομα times.txt. Τέλος για τα script που αφορούν το MPI ή MPI+OpenMp, το script διαβάζει τα αντίστοιχα αρχεία χρόνων και δημιουργεί τα speedup.txt και efficiency.txt.

Επίσης, υλοποιήσαμε script που δημιουργεί επίπεδα αρχεία εισόδου (όλος ο δισδιάστατος πίνακας χαρακτήρων σε μια γραμμή) τα οποία είναι ίδια με τα παραγόμενα αρχεία κάθε βήματος. Επιπλέον για να ελέγξουμε αν κάθε βήμα έχει υπολογιστεί σωστά υλοποιήσαμε script που μετατρέπει τα επιπεδα αρχεία εισόδου/εξόδου σε δισδιάστατη αναπαράσταση μαύρων ή άσπρων κουτιών (μαύρο κουτί αν η τιμή είναι 0 και άσπρο αν είναι 1).

Στο mpi και openmp + mpi χρησιμοποιούμε το command line για να πάρουμε τα εξής:

- path για το input file
- path για το output file
- αριθμό γραμμών
- αριθμό στηλών

MPI

Η επιλογή των κόμβων και των διεργασιών έγινε με το σκεπτικό να γεμίζει ο κάθε κόμβος πριν χρησιμοποιήσουμε κάποιο καινούργιο. Αυτό το κάνουμε για να έχουμε όσο το δυνατό λιγότερη επικοινωνία ανάμεσα στους κόμβους ώστε να αποφεύγονται άσκοπες καθυστερήσεις. Δοκιμάζοντας και τις 2 προσεγγίσεις (πληρότητα κόμβων, διασπορά διεργασιών σε κόμβους) καταλήξαμε στην πρώτη διότι παρατηρήσαμε ότι ήταν πιο γρήγορη, πράγμα το οποίο είναι λογικό αφού στην αρχιτεκτονική κατανεμημένης μνήμης ο κάθε κόμβος δεν έχει πρόσβαση στη μνήμη άλλου κόμβου. Αυτό έχει ως αποτέλεσμα όσο τους αυξάνουμε να αυξάνεται και η επικοινωνία μεταξύ των κόμβων μέσω μηνυμάτων και συνεπως να αυξάνεται η καθυστέρηση.

Σχεδιασμός

Ο δισδιάστατος πίνακας εισόδου θεωρείται περιοδικός, π.χ. αν είμαστε στην τελευταία γραμμή του δισδιάστατου και θέλουμε να πάμε στην ακριβώς από κάτω του, τότε αυτή θα είναι η πρώτη γραμμή κ.ο.κ.

Για την καλύτερη επίτευξη επικοινωνίας μεταξύ των κόμβων, δημιουργούμε μια τοπολογία όπου περιέχει ένα δισδιάστατο πλέγμα από μπλοκς/διεργασίες διαστάσεων sqrt(#of processes). Κάθε διεργασία είναι στοιχισμένη σε κάποια θέση στο δισδιάστατο πλέγμα. Κάθε διεργασία επεξεργάζεται ένα μπλοκ.

Σχεδιασμός MPI κώδικα

Για να μπορέσει να τρέξει το MPI πρόγραμμα, πρέπει να γίνει αρχικοποίηση. Για να γινει αυτό καλούμε τη συνάρτηση **MPI_Init()**. Αμέσως μετά δημιουργούμε την τοπολογία μέσω της συνάρτησης **setupGrid** όπου αρχικοποιεί τη δομή **gridInfo** με τα δεδομένα της αντίστοιχης διεργασίας.

```
typedef struct gridInfo {
   MPI_Comm gridComm;      // communicator for entire grid
   Neighbors neighbors;    // neighbor processes
   int processes;          // total number of processes
   int gridRank;           // rank of current process in gridComm
   int gridDims[2];        // grid dimensions
   int gridCoords[2];      // grid coordinates
   int blockDims[2];       // block dimensions
   int localBlockDims[2];  // local block dimensions
   int stepGlobalChanges;
   int stepLocalChanges;
} GridInfo;
```

Τα δεδομένα τα αρχικοποιεί η παραπάνω συνάρτηση σε κάθε διεργασία είναι ο communicator που ανήκει στο πλέγμα, οι γειτονικές διεργασίες, ο συνολικός αριθμός διεργασιών, η τάξη της διεργασίας στο πλέγμα, τις διαστάσεις του πλέγματος, τις συντεταγμένες της στο πλέγμα καθώς και τις διαστάσεις του συνολικού πίνακα αλλά και του τοπικού (δικού της) πίνακα. Για να δημιουργηθεί η τοπολογία καλούμε την συνάρτηση **MPI_Cart_create()** και για να λάβουμε τις συντεταγμένες κάθε διεργασίας στην τοπολογία καλούμε την **MPI_Cart_coords()**.

Για τον υπολογισμό κάθε βήματος χρειάζεται να ξέρουμε την προηγούμενη κατάσταση του παιχνιδιού για υπολογίσουμε την τρέχουσα. Για το λόγο αυτό αρχικοποιούμε δυναμικά (malloc) 2 δισδιάστατους πίνακες χαρακτήρων (old, current).

Χρειαζόμασταν ένα τρόπο με τον οποίο θα παίρναμε τις ακριανές γραμμές και στήλες των γειτόνων κάθε μπλοκ, για να τις αποθηκευσουμε στο μπλοκ ώστε να μπορούμε να κάνουμε υπολογισμούς για τα ακριανά κύτταρα του κάθε μπλοκ.

Για αυτό το λόγο δημιουργήσαμε datatypes με τις κατάλληλες συναρτήσεις (MPI_Type_vector, MPI_Type_commit) μέσω των οποίων μπορούσαμε να πάρουμε τις γραμμές και τις στήλες.

Επειδή στο mpi για τα datatypes θεωρείται πως τα δεδομένα είναι σε συνεχόμενες θέσεις μνήμης, και τα offset για γραμμές και στήλες είναι σταθερα, χρειαζόταν οι δισδιάστατοι πίνακες old και current να είναι σε συνεχόμενες θέσεις μνήμης. Γι αυτό το λόγο υλοποιήσαμε την allocate2darray με τέτοιο τρόπο ώστε να δημιουργεί ένα δισδιάστατο πίνακα σε συνεχόμενες θέσεις μνήμης.

Επιπλέον δημιουργήσαμε ένα subarray datatype.

Αν υπάρχει αρχείο εισόδου χρησιμοποιούμε το subarray datatype για την ανάγνωση των σωστών περιοχών/τμημάτων του αρχείου εισόδου από το αντίστοιχο process μέσω της **MPI_File_set_view()** και το διαβάζουμε μέσω της **MPI_File_read()**.

Σε αντίθετη περίπτωση, δημιουργείται ένας τυχαίος πίνακας με την αρχική κατάσταση του παιχνιδιού.

Για την επίτευξη της αποστολής των 8 μηνυμάτων (ένα για κάθε γείτονα) στις γειτονικές διεργασίες καθώς και τη λήψη άλλων 8, χωρίς την επιβάρυνση της κεντρικής επανάληψης με επιπλέον υπολογισμούς (offsets, κτλ), απαιτείται η αρχικοποίηση τους μέσω της συνάρτησης **MPI_Send_init()** ώστε να δημιουργηθούν 16 requests τα οποία θα είναι έτοιμα προς εκκίνηση όταν αυτό χρειαστεί με τη χρήση της συνάρτησης **MPI_Startall()**.

Στη συνέχεια γίνονται οι υπολογισμοί των εσωτερικών κυττάρων και κατόπιν, καλείται η **MPI_Waitall()** ωστε πριν υπολογιστούν και τα εξωτερικά κύτταρα να έχουν σταλθεί όλα τα γειτονικά σε κάθε μπλοκ για να γίνουν σωστά οι υπολογισμοί.

Η συνάρτηση **calculate()** που έχουμε υλοποιήσει για τους υπολογισμούς είναι **inline** για να μην υπάρχουν καθυστερήσεις, επιπλέον, η συνάρτηση αυτή δέχεται ώς όρισμα ένα δείκτη στην ακέραια μεταβλητή **changes** όπου αν υπάρξει αλλαγή στο συγκεκριμένο κύτταρο που εξετάζεται εκείνη τη στιγμή, η μεταβλητή αυτή αυξάνεται κατά 1.

Στο τέλος της επανάληψης, το αποτέλεσμα που υπολογίστηκε από τον πίνακα old, θα αποθηκευτεί στον πινακα current. Έτσι, στην επόμενη επανάληψη θα πρέπει η παλιά κατάσταση του παιχνιδιού να αντικατασταθεί με την current του προηγούμενου βήματος. Για το λόγο αυτό, πρέπει να γίνει swap του old με τον current.

4. Μετρήσεις χρόνων εκτέλεσης, υπολογισμός speedup, efficiency παρουσίαση αποτελεσμάτων (με έλεγχο τερματισμού και
χωρίς). Μελέτη κλιμάκωσης δεδομένων και επεξεργαστών. Σταθερός αριθμός επαναλήψεων για όλες τις μετρήσεις, ώστε να
παίρνετε μετρήσιμους χρόνους, π.χ ακολουθιακό πρόγραμμα όχι πάνω από ~20 δευτερόλεπτα. Χρήση mpiP για τον
εντοπισμό καθυστερήσεων και την βελτιστοποίηση του κώδικα MPI. Μετρήσεις για
a. MPI πρόγραμμα μόνο με τοπική επικοινωνία, χωρίς allreduce για έλεγχο κενού πλέγματος ή μη αλλαγής πλέγματος.
b. MPI πρόγραμμα με επιπλέον έλεγχο κενού πλέγματος ή μη αλλαγής. Προσοχή, ακόμη και να διαπιστώσετε συνθήκη τέλους να εκτελείτε όλες τις επαναλήψεις γενεών. Allreduce κάθε 10 επαναλήψεις.
c. Να ενσωματωθούν στο MPI+allreduce εντολές OpenMp για παραλληλοποίηση υπολογισμών (π.χ εσωτερικών στοιχείων), ώστε να αναπτυχθεί υβριδικό πρόγραμμα. Συγκρίσεις με καθαρό MPI.

Μετρήσεις χρόνων εκτέλεσης

	1	**4**	**16**	**64**
320320	0.745	0.216	0.202	0.292
640x640	3.354	0.770	0.313	0.310
1280x1280	13.268	3.383	0.851	0.391

2560x2560	53.610	13.459	3.466	0.917
5120x5120	213.743	53.943	18.438	8.652
10240x10240	-	215.087	54.659	13.750
20480x20480	-	-	217.871	57.988
40960x40960	-	-	-	218.213

υπολογισμός speedup

	1	**4**	**16**	**64**
320320	1.0	3.449	3.681	2.542
640x640	1.0	4.357	10.705	10.815
1280x1280	1.0	3.921	15.600	33.949
2560x2560	1.0	3.983	15.468	58.478
5120x5120	1.0	3.962	11.593	24.704

υπολογισμός efficiency

	1	**4**	**16**	**64**
320320	1.0	1.724	0.920	0.318
640x640	1.0	2.178	2.676	1.352
1280x1280	1.0	1.961	3.900	4.244
2560x2560	1.0	1.992	3.867	7.310
5120x5120	1.0	1.981	2.898	3.088