Εργασία Παράλληλων Συστημάτων 2019-20

README_CODE για τον κώδικα (Το README για τις μετρήσεις βρίσκεται στο άλλο README)

Game of Life

Μέλη ομάδας:
Φαραώ Γεώργιος 1115201700177

Στρόμπολας Σοφοκλής 1115201700153

1. Αρχεία

   Ο κώδικας βρίσκεται στα αρχεία **gol_mpi.c** και **gol_mpi_omp.c .**
   Αναφορικά τα υπόλοιπα αρχεία της εργασίας:

   - Το αρχείο **gol_mpi.c** υλοποιεί το Conway's game of life με MPI (με allreduce, και χωρίς, ανάλογα το όρισμα).
   - Το αρχείο **gol_mpi_omp.c** υλοποιεί το Conway's game of life με υβριδικό προγραμματισμός MPI+OpenMp (με allreduce, και χωρίς, ανάλογα το όρισμα).
   - Στο αρχείο **README** βρίσκεται η Τεκμηρίωση, η Παρουσίαση και τα Αποτελέσματα των προγραμμάτων
   - Στο αρχείο **mpiPresult3840grid-64p-500g** βρίσκεται το αποτέλεσμα του mpiP για το gol_mpi.c για το grid 3840, με 64 processes και 500 generations.

2. Μεταγλώττιση και Εκτέλεση προγράμματος στο Μηχάνημα Argo

   Για να μεταγλωτησουμε το MPI:
   mpicc -O3 gol_mpi.c -o gol_mpi.x -lm
   Και με mpiP:
   mpicc -O3 -g gol_mpi.c -L$MPIP_DIR/lib -lmpiP - lbfd -lunwind -o gol_mpi.x
   Για να το τρέξουμε στο script γράφουμε:
   mpirun -np (processes) ./gol_mpi.x -g (generations) -sz (array size) -sc (boolean flag for sanity check)

Για το υβριδικό MPI+OpenMp, για την μεταγλώττιση:
mpicc -O3 -fopenmp gol_mpi_omp.c -o gol_mpi_omp.x -lm
Για να το τρέξουμε στο script:
mpirun -np (processes) ./gol_mpi_omp.x -g (generations) -sz (array size) -sc (boolean flag for sanity check) -th (number of threads)

3. Λειτουργία προγράμματος gol_mpi.c

- Αρχικά κάνουμε έλεγχο για το αν έχει δοθεί σωστός αριθμός ορισμάτων. Στην συνέχεια τα τοποθετούμε σε μεταβλητές. Ξεκινάμε με MPI_Init και δημιουργούμε το αρχικό μας grid.
- Επειτα μοιράζουμε το grid στα blocks, που είναι ανάλογα με τον αριθμό των διεργασιών, με την απαίτηση ο αριθμός τους να είναι τέλειο τετράγωνο. Μετά υπολογίζουμε σε ποιο σημείο του αρχικου grid θα ξεκινάει καθε block και δημιουργουμε blocks με μέγεθος πλευράς array_size/(processes) και φτιάχνουμε ένα νέο datatype για τα blocks αυτά.
- Φτιάχνουμε ένα 2D καρτεσιανό επίπεδο με τα blocks και βρισκουμε τους γείτονές τους για north, south, east, west, north east, north west, south east, south west.
- Δημιουργούμε τους πίνακες send_line και send_side, που χρησιμοποιούνται για να στέλνουμε σε γειτονικά blocks γραμμές και στήλες αντίστοιχα, καθως και τον recv_array όπου αποθηκεύουμε τις τιμές που λαμβάνουμε από τα γειτονικά block.
- Ξεκινάει η επανάληψη για τα generations. Αρχικά κανουμε τα MPI_Irecv και έπειτα τα MPI_Isend.
- Υπολογίζουμε την τιμή των στοιχείων του block που είναι εσωτερικά και κάνουμε MPI_Waitall για τα receive.
- Μετα θα υπολογίσουμε τις τιμές των εξωτερικών στοιχείων με βάσει τις τιμές που λάβαμε από τα γειτονικά block, πρώτα για τις δύο γραμμές και μετά για τις δύο στήλες.
- Στο σημείο αυτο αν έχουμε allreduce γίνεται έλεγχος (κάθε 10 generations) για το αν το grid μας είναι ολο μηδέν ή το ίδιο με την προηγούμενη γενεά.
- Κάνουμε swap τα παλιά grid με τα καινούργια που δημιουργήσαμε και wait για τα send.
- Βγαίνοντας από την επανάληψη κάνουμε MPI_Gatherv για να σχηματίσουμε το τελικό grid μας και βρίσκουμε τον τελικό χρόνο.
- Υπολογίζουμε τον μέγιστο, ελάχιστο και μέσο χρόνο και τους εκτυπώνουμε, μαζί με το τελικό grid.

4. Λειτουργία gol_mpi_omp.c

- Η λογική είναι ίδια με του απλού mpi προγράμματος με μερικές προσθήκες.
- Κανουμε include το omp.h
- Στην γραμμή εντολών δίνουμε με -th τον αριθμό των threads.
- Προσθέτουμε # pragma omp parallel num_threads και #pragma omp for για να παραλληλοποιήσουμε την διπλή for που υπολογίζει τα εσωτερικά στοιχεία και τις 4 for που υπολογίζουν τα εξωτερικά.