Παράλληλα & Κατανεμημένα Συστήματα Υπολογιστών

Εργασία 2

Μουρούζης Χρίστος
ΑΕΜ:7571

ΕΡΓΑΣΙΑ 2: ΥΛΟΠΟΙΗΣΗ IMPERATIVE BITONIC SORT ΣΕ MPI

ΠΕΡΙΓΡΑΦΗ:

Σε αυτή την εργασία μας ζητήθηκε να γράψουμε κατανεμημένο πρόγραμμα σε MPI που να διατάσσει, με αύξουσα σειρά, $N=2^{p+q}$ ακέραιους αριθμούς, χρησιμοποιώντας τον αλγόριθμο Imperative Bitonic Sort για την υλοποίηση των επικοινωνιών μεταξύ των διεργασιών. Οι 2^p διεργασίες έχουν από ένα πίνακα με 2^q στοιχεία τα οποία ορίζονται τυχαία μέσω της συνάρτησης rand(). Σκοπός του προγράμματος είναι να διατάσσει σε αύξουσα σειρά όλους τους αριθμούς όλων των πινάκων συνολικά. Παρακάτω αναλύεται η διαδικασία που το πρόγραμμα επιτυγχάνει τον πιο πάνω σκοπό μέσω συγκεκριμένων συναρτήσεων.

ΑΝΑΛΥΣΗ:

Συνάρτηση:

- **init():** Η συγκεκριμένη συνάρτηση αρχικοποιεί τον πίνακα κάθε διεργασίας δίνοντας του σαν τιμές τυχαίους αριθμούς από 0-2^q. Επίσης τυπώνει για πρακτικούς λόγους τα τελευταία 16 στοιχεία του πίνακα της τελευταίας διαδικασίας. Το πρόγραμμα για να συνεχίσει τη διαδικασία περιμένει όλες τις διεργασίες να τελειώσουν την αρχικοποίηση των πινάκων τους μέσω της εντολής MPI_BARRIER.
- **impBitonicSort():** Αυτή η συνάρτηση διατάσσει με τον επαναληπτικό Bitonic Sort αλγόριθμο τα στοιχεία όλων των πινάκων όλων των διεργασιών. Συγκεκριμένα αν το rank της διεργασίας είναι το ίδιο με το rank της συνάδελφης διεργασίας τότε ο έλεγχος γίνεται σε ένα πίνακα αφού μιλάμε για την ίδια διεργασία και ο πινάκας αυτός διατάσσεται τοπικά σε αύξουσα σειρά μέσω της συνάρτησης **exchange.** Αν το rank της διεργασίας είναι διαφορετικό με της συνάδελφης διεργασίας τότε με τις εντολές MPI_SEND και MPI_RECV έχουμε αντίστοιχα τον πίνακα της «χ» διεργασίας και της «κ» διεργασίας να ελέγχονται μεταξύ τους και να διατάσσονται μεταξύ τους μέσω της συνάρτησης **exchange_with_partner().**
- **test():** Η συνάρτηση test ελέγχει τον κάθε πίνακα ξεχωριστά αν είναι σε αύξουσα σειρά και τυπώνει αν είναι επιτυχής ο έλεγχος ή όχι. Ο λόγος που το κάνουμε αυτό είναι για να ελέγξουμε τυχόν πίνακες που δεν διατάχθηκαν σωστά ή μπορεί να «ξεχάστηκαν κάπου στη μέση».
- **test_final():** Αφού όλοι οι πίνακες πέρασαν τον τοπικό έλεγχο ορθότητας με τη συνάρτηση test_final ελέγχουμε, αρχίζοντας από την διεργασία 0 εως και rank-1 διαδοχικά, το τελευταίο στοιχείο κάθε πίνακα με το αρχικό στοιχείο του

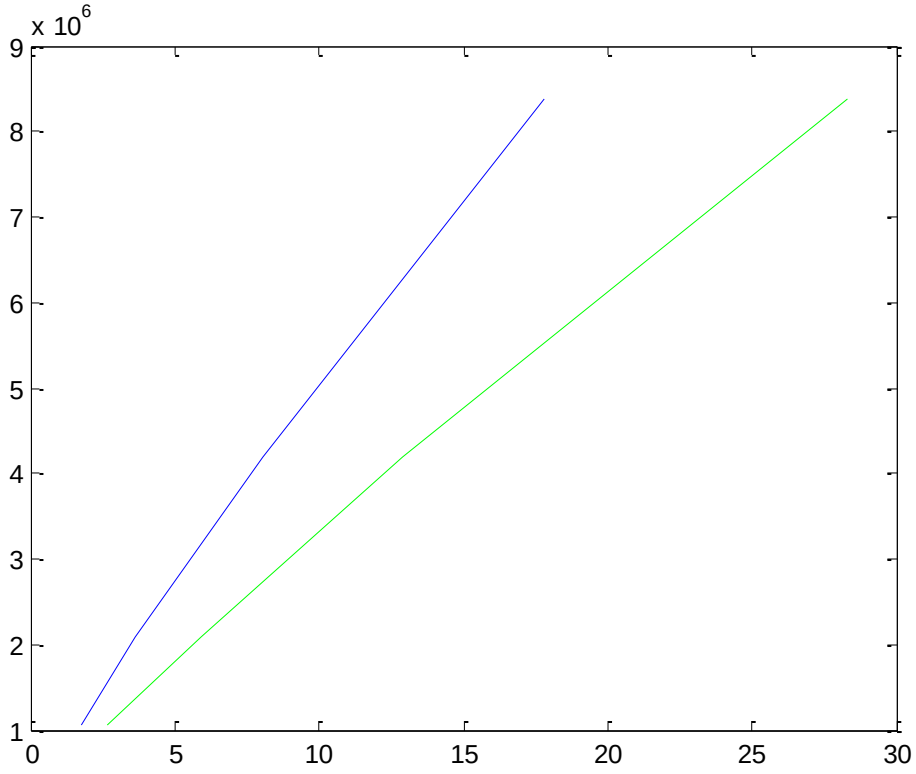
προηγούμενου του. Αν ο έλεγχος είναι ορθός τότε έχουμε όλα τα στοιχεία συνολικά διατεταγμένα σε αύξουσα σειρά.

- **print():** Την εκτέλεση αυτής της συνάρτησης αναλαμβάνει η τελευταία διεργασία αφού θέλουμε να τυπώσουμε τα τελευταία 16 στοιχεία του συνόλου των στοιχείων (για πρακτικούς λόγους).
- Τέλος το πρόγραμμα τερματίζει με την εντολή MPI_FINALIZE.

ΣΥΓΚΡΙΣΗ ΠΑΡΑΛΛΗΛΟΥ ΚΩΔΙΚΑ ΜΕ ΣΕΙΡΙΑΚΟ:

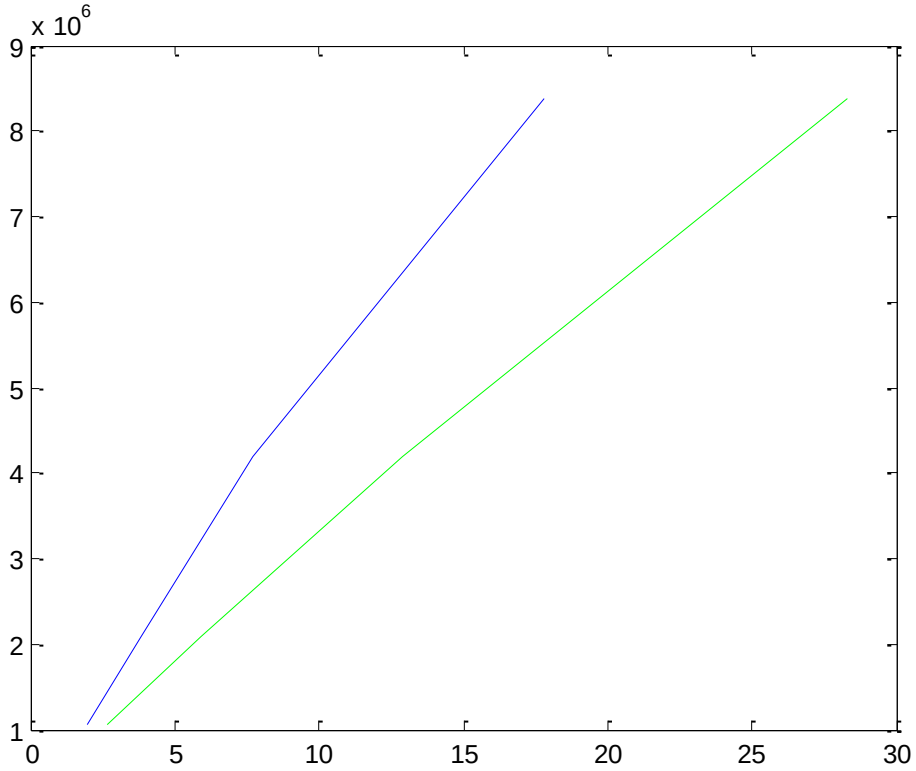
Παρακάτω παρατίθενται γραφικές παραστάσεις **(N στοιχεία-time in secs)** που δείχνουν ότι όντως ο παράλληλος κώδικας που υλοποιήθηκε πιο πάνω είναι πιο «γρήγορος» από το σειριακό. Οι εκτελέσεις των προγραμμάτων έγιναν στο HellasGrid.

- Εδώ θεωρούμε σταθερό τον αριθμό των διεργασιών και αλλάζουμε διαδοχικά τον αριθμό των στοιχείων πίνακα έτσι ώστε να προκύπτει **N=2^{20}, 2^{21}, 2^{22}, 2^{23}** και το συγκρίνουμε αντίστοιχα με τον σειριακό. Συγκεκριμένα, **διεργασίες**=2^3=8 και σ**τοιχεία**= 2^{17}, 2^{18}, 2^{19}, 2^{20}. Με πράσινο χρώμα είναι ο σειριακός κώδικας και με μπλε χρώμα ο παράλληλος. Η γραφική παράσταση είναι N στοιχεία συναρτήσει του χρόνου σε secs.



Σχ.1 Σύγκριση παράλληλου κώδικα (σταθερές διεργασίες) με το σειριακό κώδικα.

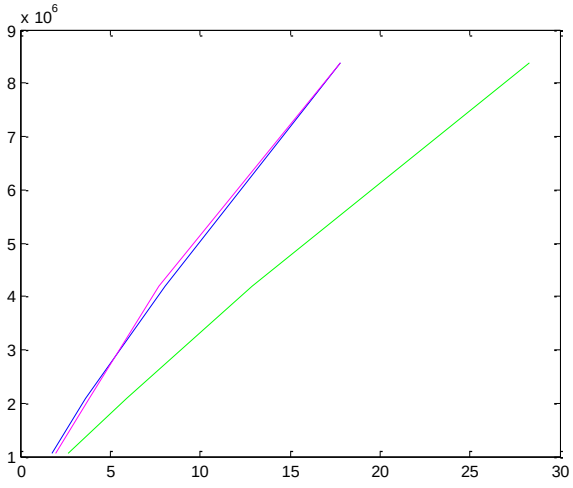
- Εδώ θεωρούμε σταθερό τον αριθμό των στοιχείων ενός πίνακα και αλλάζουμε διαδοχικά τον αριθμό των διεργασιών έτσι ώστε να προκύπτει **N=2^{20}, 2^{21}, 2^{22}, 2^{23}** και το συγκρίνουμε αντίστοιχα με τον σειριακό. Συγκεκριμένα, **στοιχεία**=2^{20} και **διεργασίες**= 2^0, 2^1, 2^2, 2^3. Με πράσινο χρώμα είναι ο σειριακός κώδικας και με μπλε χρώμα ο παράλληλος. Η γραφική παράσταση είναι Ν στοιχεία συναρτήσει του χρόνου σε secs.



Σχ.2 Σύγκριση παράλληλου κώδικα (σταθερά στοιχεία πίνακα) με το σειριακό κώδικα.

Παρατηρήσεις:

Δεξιά φαίνονται οι δύο γραφικές παραστάσεις στο ίδιο σχήμα. Βλέπουμε ότι είτε έχουμε σταθερό αριθμό διεργασιών είτε σταθερό αριθμό στοιχείων κάθε πίνακα ο παράλληλος κώδικας είναι κατά πολύ πιο γρήγορος από τον σειριακό.



Τέλος παρατίθεται ένα παράδειγμα εκτυπώσεως αποτελεσμάτων:

```
Number of procs=8
Elements of Each Array=65536
Elements to sort (ProcsxN)=524288

------->Initialize Array (Last 16 Elements)<-------
52965
23579
29685
57214
21419
3923
8213
9196
35479
40419
21214
24756
30937
43306
17204
173

 TEST of process 1 Passed
 TEST of process 2 Passed
 TEST of process 3 Passed
 TEST of process 4 Passed
 TEST of process 5 Passed
 TEST of process 6 Passed
 TEST of process 7 Passed
 TEST of process 8 Passed
 Test all elements. Test: PASSED

------->Final Array (Last 16 Elements)<-------
65534
65534
65534
65534
65534
65534
65534
65534
65535
65535
65535
65535
65535
65535
65535
65535

Time for bitonic sort = 0.708923
```

ΚΑΛΑ ΧΡΙΣΤΟΥΓΕΝΝΑ ΚΑΙ
ΕΥΤΥΧΙΣΜΕΝΟ ΤΟ
ΚΑΙΝΟΥΡΓΙΟ ΕΤΟΣ 2015