Παράλληλα Συστήματα

Εισαγωγή

Η παρούσα εργασία αποτελεί την από κοινού προσπάθεια των φοιτητών Πάσχος Νικόλαος (8360) και Μπέκος Χριστόφορος (8311). Για την εκπόνηση της εργασίας πραγματοποιήθηκε παραλληλοποίηση του αλγορίθμου sorting “Bitonic Sort”. Ο αλγόριθμος εκτελείται σε 2 μορφές, imperative και recursive. Για την παραλληλοποίηση των μεθόδων αυτών έγινε χρήση OpenMP, Pthreads καθώς και Cilk. Και για τις 3 μεθόδους τα σημεία παραλληλοποίησης είναι κοινά, η υλοποίηση με την κάθε μέθοδο όμως διαφέρει. Τα αποτελέσματα από μέθοδο σε μέθοδο ποικίλουν και θα εξεταστούν στη συνέχεια.

Imperative Bitonic Sort

Στην imperative εκδοχή του αλγορίθμου παρατηρώ πως υπάρχουν 3 βρόγχοι for. Για την παραλληλοποίηση επιλέχθηκε ο εσωτερικότερος βρόγχος αφού στους υπολοίπους υπάρχει εξάρτηση δεδομένων σε κάθε επανάληψη.

- Cilk:

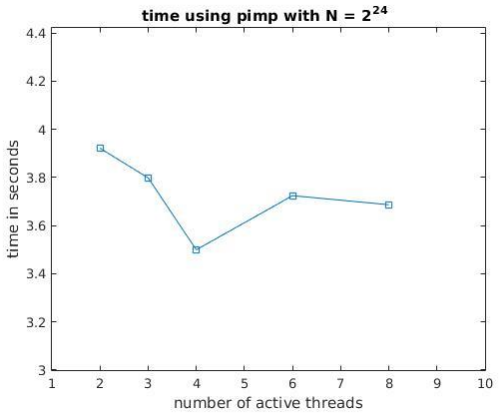
Για τον προσδιορισμό από τον χρήστη του αριθμού των workers έγινε χρήση των παρακάτω εντολών:

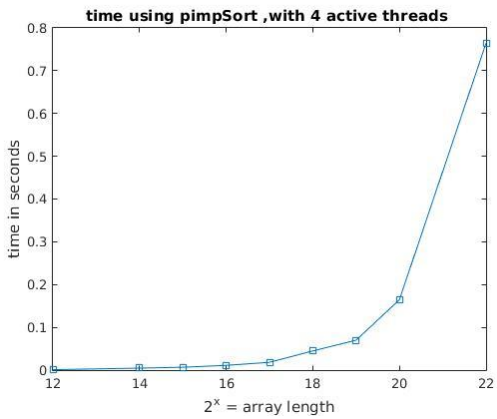
```
__cilkrts_init();

 char threads_cilk = num_of_threads + '0';

 __cilkrts_set_param("nworkers", &(threads_cilk));
```

Σημειώνεται πως ο πραγματικός αριθμός των workers δεν είναι αναγκαστικά ο αριθμός που ορίζει ο χρήσης. Το παραπάνω code block αποτελεί μία προτροπή για τον ορισμό των workers. Σε περίπτωση που κριθεί ότι το overhead είναι πολύ μεγάλο ή για εσωτερικούς λόγους της cilk, ο αριθμός των wokers





Πάσχος Νικόλαος 8360
Μπέκος Χριστόφορος 8311

τίθεται αυτόματα. Εμπειρικά υπολογίστηκε πως ο αριθμός αυτός βρίσκεται, όπως φαίνεται και από το διάγραμμα, για p = 2^5 workers.

Η παραλληλοποίηση έγινε μέσω της εντολής cilk_for όπως φαίνεται παρακάτω:

cilk_for(i=0; i<N; i++) (γραμμή 258 του αρχείου bitonic.c στον φάκελο cilk)

Το compile έγινε με χρήση του icc ο οποίος για τη χρήση του ορίζεται σαν source μέσω της εντολής (που βρίσκεται στα σχόλια του αντίστοιχου makefile) «source /opt/intel/bin/compilervars.sh intel64» 7

- OpenMP:

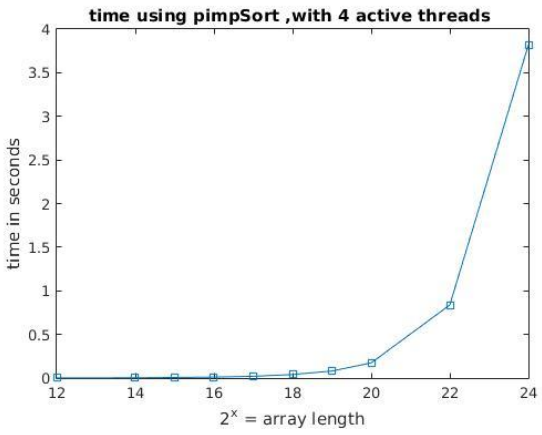
Για τον προσδιορισμό του αριθμού των threads στην OpenMP έγινε χρήση του παρακάτω block κώδικα:

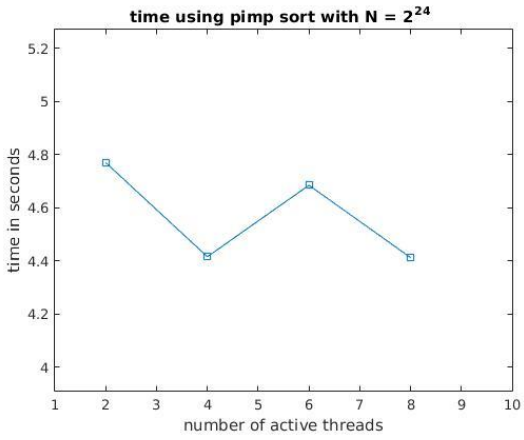
omp_set_dynamic(0);

#pragma omp parallel for num_threads(num_of_threads) default(shared) private(i)

for (i=0; i<N; i++)

Με τις παραπάνω εντολές απενεργοποιείται ο αυτόματος καθορισμός του αριθμού των threads, και με το parallel statement καθορίζεται πως η εσωτερικότερη for θα τρέξει παράλληλα με χρήση num_of_threads αριθμού threads, με private μεταβλητή μεταξύ των ενεργών threads την i. Αυτό αποτελεί ως ένα βαθμό πλεονασμό αλλά καθιστά τον κώδικα πιο κατανοητό, αφού αυτόματα η "i" αποτελεί private μεταβλητή.





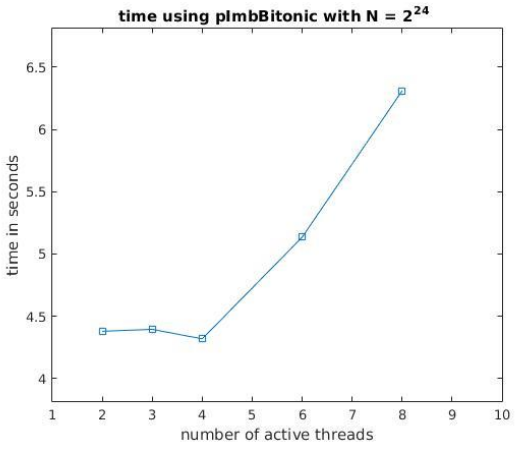
- Pthreads:

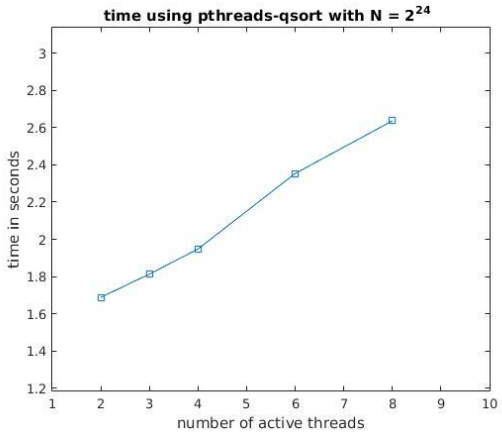
Για την παραλληλοποίηση με pthreads υλοποιήθηκε η εξής διαδικασία. Σε κάθε επανάληψη της $2^{ης}$ for δημιουργείται ένας πίνακας ο οποίος αποθηκεύει όλα τα threads που θα δημιουργηθούν. Στη συνέχεια δημιουργείται ένα struct pthreads_data για κάθε νήμα που θα σηκωθεί. Για κάθε ένα νήμα δίνονται

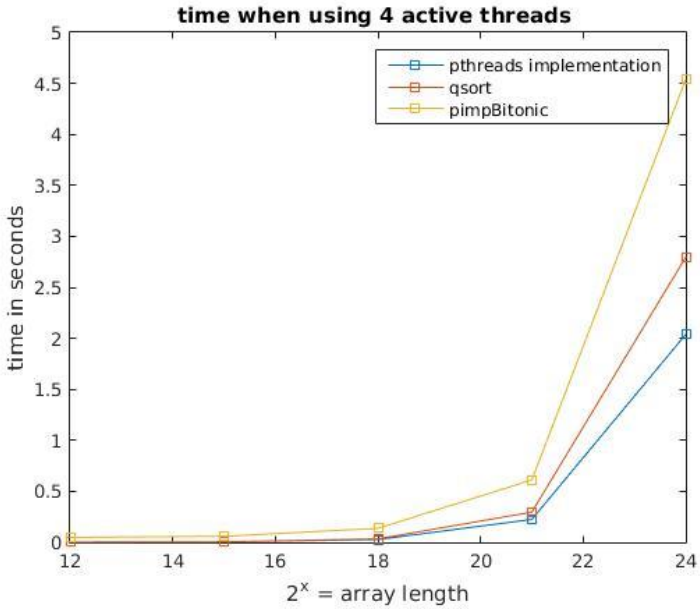
στο αντίστοιχο struct οι κατάλληλες τιμές που αφορούν τα άκρα του διαστήματος του global πίνακα που τίθεται προς sorting. Τέλος σηκώνονται τα threads και το thread της main περιμένει να ολοκληρώσουν την εκτέλεσή τους τα threads που σηκώθηκαν.

Εσωτερικά ο αλγόριθμος που εκτελείται σε κάθε thread ορίζεται από την συνάρτηση void* pImpBSort(void* data). Στη συνάρτηση αυτή γίνονται τα απαραίτητα typecasts για διαβαστούν σωστά τα δεδομένα που δίνονται σαν όρισμα σε αυτήν και στη συνέχεια εκτελείται η ίδια λογική με τον σειριακό imperative bitonic sort, αλλά αυτή τη φορά τοπικά στα άκρα που ορίζονται από το pthreads_data struct.

Σε όλα τα σημεία του αλγορίθμου γίνεται έλεγχος για την επιτυχή δημιουργία των threads από την pthread_create καθώς και για την επιτυχή έξοδό τους μέσω του status code που επιστρέφεται στην pthread_join.







Πάσχος Νικόλαος 8360
Μπέκος Χριστόφορος 8311

Recursive Bitonic Sort

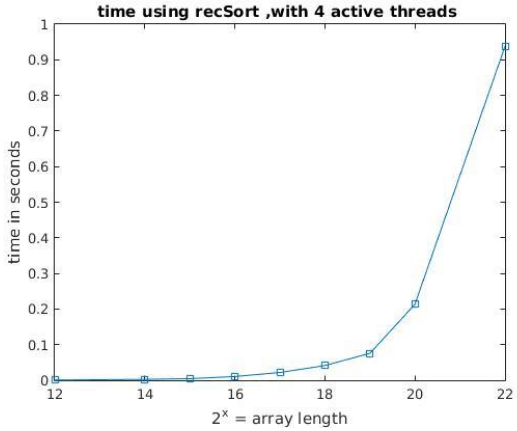
Για την παραλληλοποίηση της εκδοχής αυτής παρουσιάζονται 3 σημεία ενδιαφέροντος. Αρχικά ο πίνακας χωρίζεται αναδρομικά σε μικρότερους πίνακες και ανάλογα τη θέση του ταξινομείται σε αύξουσα ή φθίνουσα σειρά. Στη συνέχεια εκτελείται η bitonicMerge η οποία επίσης αναδρομικά ταξινομεί έναν πίνακα που έχει τη μορφή αύξουσα-φθίνουσα με αλλαγή μονοτονίας στο μέσο του, σε αύξουσα σειρά. Τα 3 πιθανά σημεία παραλληλοποίησης αφορούν τα παραπάνω σημεία. Τα 3 βήματα του αλγορίθμου είναι η ταξινόμηση σε αύξουσα σειρά, η ταξινόμηση σε φθίνουσα και τέλος η ένωση (ταξινόμηση δηλαδή σε αύξουσα σειρά) των επιμέρους τμημάτων. Λογική εξάρτηση δεδομένων παρουσιάζεται μόνο μεταξύ της ένωσης των τμημάτων. Συνεπώς ο αλγόριθμος μπορεί να παραλληλοποιηθεί με τους εξής τρόπους:

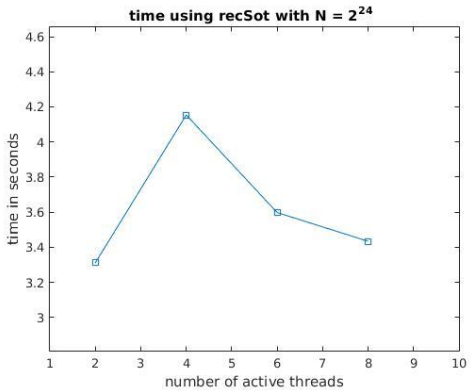
1. Παραλληλοποίηση σε 3 βήματα: 2^p threads για αύξουσα ταξινόμηση, 2^p threads για φθίνουσα των υπολοίπων τμημάτων και 2^p threads για bitonicMerge.
2. Παραλληλοποίηση σε 2 βήματα: 2^p threads για αύξουσα και φθίνουσα ταξινόμηση. 2^p threads για bitonicMerge.

Είναι προφανές ότι στην περίπτωση 1 χρειάζεται να σηκωθούνε 3 φορές 2^p threads καθώς και να περιμένει το thread της main 3 φορές για την ολοκλήρωση της εργασίας τους. 1 ανάμεσα σε κάθε διαδικασία και μία μετά την bitonicMerge. Αυτό αποδείχθηκε μη αποτελεσματικό και έτσι υιοθετήθηκε η δεύτερη μέθοδος για την παραλληλοποίηση.

- Cilk:

Έγινε κλήση της cilk_spawn για την παράλληλη εκτέλεση της recBitonicSort. Μετά την εκτέλεση της recBitonicSort με όρισμα DESCENDING χρησιμοποιείται η εντολή cilk_sync για τον συγχρονισμό των νημάτων. Πριν από την διαδικασία αυτή ακολουθήθηκε η λογική που περιεγράφηκε στην imperative cilk υλοποίηση για τον καθορισμό του αριθμού των workers.

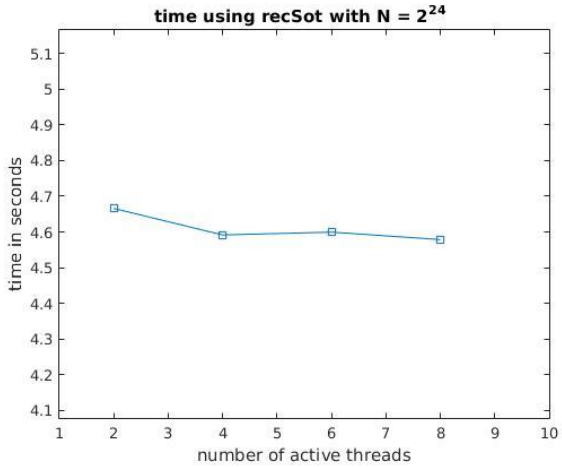


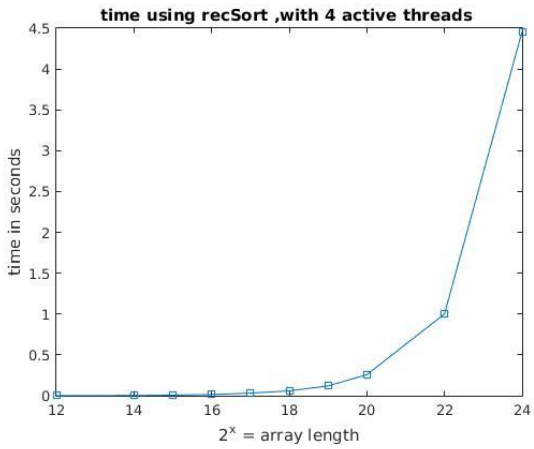


Πάσχος Νικόλαος 8360
Μπέκος Χριστόφορος 8311

- OpenMP:

Για την παραλληλοποίηση με OpenMP χρησιμοποιήθηκαν tasks. Έγινε αρχική κλήση της recBitonicSort με όρισμα όλο τον πίνακα N σαν 1 νήμα τύπου single nowait. Στη συνέχεια μέσα σε κάθε recursive call το κάθε task εκτελείται παράλληλα μέχρι και πριν από την bitonicMerge της οποίας το task είναι τύπου single nowait, συνεπώς περιμένει να τελειώσουν τα υπόλοιπα threads.





- Pthreads:

Για τα pthreads υλοποιήθηκε συνδυασμός pthreads με std::qsort για καλύτερα αποτελέσματα. Στον αλγόριθμο αυτό ο πίνακας χωρίζεται σε batch = p/N κομμάτια. Κάθε batch στη συνέχεια μέσω της std::qsort γίνεται sort σε ascend ή descend αναλόγως το τι απαιτείται κάθε φορά. Το νήμα της main περιμένει να τελειώσει η εκτέλεση της παραπάνω λογικής. Στη συνέχεια δημιουργεί κατάλληλο αριθμό threads τα οποία recursively εκτελούν merge στα μέρη του πίνακα που δημιουργήθηκαν προηγουμένως. Το merge αυτό σημειώνεται πως δεν γίνεται αναγκαστικά σε μορφή ascend, αλλά σε ότι χρειάζεται ανάλογα την θέση του batch ώστε να διατηρεί ο πίνακας την μορφή ascend-descend ώστε να είναι εφικτή η κλήση της bitonicMerge ξανά από νέα νήματα. Σε κάθε επανάληψη το νήμα της main και πάλι περιμένει να τελειώσει η εκτέλεση των pthreads.

EXTRA! Qsort/SelectionSort Hybrid using Cilk

Ως επιπλέον απασχόληση υλοποιήθηκε και δίνεται ο κώδικας παράλληλης υλοποίησης της qsort σε συνδυασμό με selectionSort για μικρούς πίνακες με τη χρήση cilk. Εμπειρικά παρατηρήθηκε ότι η μεγαλύτερη επιτάχυνση δημιουργείται όταν η selection sort εκτελείται για πίνακες μήκους 32.

Πάσχος Νικόλαος 8360
Μπέκος Χριστόφορος 8311

Σχολιασμός Αποτελεσμάτων

Όπως είναι αναμενόμενο όσο μεγαλώνει το μέγεθος του πίνακα N με τον αριθμό των νημάτων σταθερό, τόσο περισσότερος χρόνος απαιτείται για την ταξινόμηση. Συγκριτικά με την std::qsort οι χρόνοι πλησιάζουν αλλά δεν γίνονται οριακά καλύτεροι. Η μόνη περίπτωση που γίνονται καλύτεροι είναι στον συνδυασμό pthreads με qsort για 2^24 δεδομένα με χρήση 2^2 μέχρι και 2^4 threads. Μεταξύ των μεθόδων παρατηρείται βέλτιστη παραλληλοποίηση με τη χρήση cilk για p = 2^[2:4]. Σε οποιαδήποτε άλλη περίπτωση εκτός της cilk, όσο ανεβαίνει ο αριθμός των threads ο χρόνος εκτέλεσης μειώνεται αλλά στη συνέχεια αρχίζει να αυξάνεται. Αυτό είναι λογικό αφού η εναλλαγή των threads είναι διαδικασία χρονοβόρα για το σύστημα και απαιτεί χρόνο. Έτσι η εναλλαγή αυτή καθιστά τον αλγόριθμο εξαιρετικά αργό και κάθε κέρδος από την παράλληλη εκτέλεση χάνεται. Τα πειράματα έγιναν σε Ubuntu 16.04 με CPU Intel i5 7200U @2.5GHz. Τα αποτελέσματα που δίνονται βγήκαν από τον μέσο όρο 10 μετρήσεων για κάθε συνδυασμό νημάτων και μεγέθους πινάκων. Αγνοήθηκαν οι 2 πρώτες τιμές ώστε να φορτωθεί ο κώδικας στην cache. Τα αποτελέσματα κρίνονται ορθά αφού γίνεται χρήση της συνάρτησης test() η οποία εξετάζει εάν ο πίνακας είναι ταξινομημένος στην ζητούμενη μορφή.

Πάσχος Νικόλαος 8360
Μπέκος Χριστόφορος 8311