Heap's algorithm Permutations σε ARM v8-A

Ηρόδοτος Δημητρίου
Τμήμα Πληροφορικής Πανεπιστημίου Κύπρου
Λευκωσία , Κύπρος
Email: edemet01@cs.ucy.ac.cy

***Περίληψη*—Η πιο κάτω αναφορά παρουσιάζει την σχεδίαση και υλοποίηση του «Heap's algorithm», ο οποίος αφορά την παραγωγή αναγραμματισμών μια λέξης n ψηφίων. Η εργασία αυτή έχει υλοποιηθεί στην συμβολική γλώσσα προγραμματισμού ARM v8 – A η οποία στηρίζεται στην αρχιτεκτονική AArch64. Ο αλγόριθμος χρησιμοποιεί μια αναδρομική συνάρτηση για την εύρεση όλων των πιθανών συνδυασμών που θα προκύψουν και τους τυπώνει στην οθόνη. Ενδιαφέρον στην λύση του προβλήματος με αυτό τον αλγόριθμο αποτελεί το γεγονός ότι οι επόμενοι αναγραμματισμοί δημιουργούνται από τον προηγούμενο αναγραμματισμό χωρίς να παρατηρούνται οποιεσδήποτε συγκρούσεις μεταξύ των αναγραμματισμών.**

Λέξεις κλειδιά: — ARM v8 – A; Heap's algorithm; αναδρομή;

ΕΙΣΑΓΩΓΗ

Στην καθημερινή μας ζωή όλοι έχουμε έρθει αντιμέτωποι με περιστατικά όπου κάποιος μας αναφέρει ένα τηλεφωνικό νούμερο αλλά εμείς από αμέλεια σημειώνουμε αντίστροφα κάποιους από τους αριθμούς του. Η λύση σε αυτό το πρόβλημα δίνεται με την χρήση αναγραμματισμών (διατάξεων) των ψηφίων του τηλεφωνικού νούμερου.

Στα μαθηματικά, η έννοια της διάταξης, σχετίζεται με την τακτοποίηση όλων των μελών ενός αριθμητικού συνόλου σε κάποια ακολουθία ή αλλιώς σειρά. Οι διατάξεις διαφέρουν από τους συνδυασμούς οι οποίοι αντίθετα αποτελούν τυχαίες επιλογές μελών του συνόλου. Υπόθεση ύπαρξης του προβλήματος των διατάξεων αποτελεί το γεγονός ότι όλα τα στοιχεία του συνόλου θεωρούνται μοναδικά.

Οι μεταβολές που μπορούν να υπάρξουν στην διάταξη των στοιχείων ενός αριθμητικού συνόλου δίνονται από τον ακόλουθο γενικευμένο τύπο.

$$P(n,r) = \frac{n!}{(n-r)!}$$

Στην περίπτωσή μας το r είναι ίσο με το n και γι' αυτό τον λόγο οι μέγιστες δυνατές διατάξεις σε ένα σύνολο με n στοιχεία είναι n!.

Το πρόβλημα αυτό είναι ένα τετριμμένο παράδειγμα χρήσης ηλεκτρονικών υπολογιστών με συνδυαστικά μαθηματικά και προκαλεί ενδιαφέρον η μελέτη του αφού πολλαπλές προσεγγίσεις μπορούν να εφαρμοστούν. Πολλοί μαθηματικοί και επιστήμονες της Πληροφορικής έχουν εξετάσει το ζήτημα ενδελεχώς τις τελευταίες δεκαετίες και μπορούμε να συμπεράνουμε χωρίς καμία αμφιβολία ότι ίσως ο πιο απλός αλγόριθμος αναπαραγωγής ανακατατάξεων των στοιχείων ενός συνόλου είναι ο «Heap's algorithm» τον οποίο πρότεινε για πρώτη φορά στο περιοδικό «The Computer Journal» ο «B. R. Heap» το 1963.

Παρατήρηση : ασχέτως με ποιον αλγόριθμο χρησιμοποιήσουμε για να λύσουμε το πρόβλημα αυτό, το υπολογιστικό κόστος αναπαραγωγής διατάξεων για 17 στοιχεία ενός συνόλου (υποθέτουμε ότι κάθε διάταξη για να υπολογιστεί χρειαζόταν 1 microsecond) ανέρχεται στα 10 χρόνια έτσι ώστε να παραχθούν όλες οι διατάξεις.

Χρόνος που χρειάζεται για την παραγωγή όλων των διατάξεων συνόλου με N στοιχεία. (1 μSec ανά διάταξη)

N	**N!**	**Χρόνος**
1	1	< sec
2	2	< sec
3	6	< sec
4	24	< sec
5	120	< sec
6	720	< sec
7	5040	< sec
8	40320	< sec
9	362880	< sec
10	3628800	3 sec
11	39916800	40 sec
12	479001600	8 min
13	6227020800	2 hours
14	87178291200	1 day
15	1307674368000	14 days
16	20922789888000	8 months
17	355689428096000	10 years

ΠΕΡΙΣΓΡΑΦΗ ΤΟΥ ΑΛΟΡΙΘΜΟΥ ΣΕ ΓΛΩΣΣΑ ΥΨΗΛΟΥ ΕΠΙΠΕΔΟΥ

Ο αρχικός κώδικας ο οποίος χρησιμοποιήθηκε για την υλοποίηση του αλγόριθμου ήταν σε γλώσσα υψηλού επιπέδου C++ και για λόγους ευελιξίας μεταφράστηκε σε C.

```
int heapPermutation(int *a, int size, int n){
        if (size == 1){
           for (int j=0; j<n; j++)
                printf("%d ", a[j]);
          printf("\n");
        }else{
          for (int i=0; i<size; i++){
          heapPermutation(a,size-1,n);
          int temp,x,y;
          if (size%2==1){
                swap(a[0], a[size-1]);
                temp = a[0];
                a[0]    = a[size-1];
                a[size-1]    = temp;
          }else{
                swap(a[i], a[size-1]);
                temp = a[i];
                a[i]    = a[size-1];
                a[size-1]    = temp;
          }
      }
}
```

Το πιο πάνω απόσπασμα κώδικα δείχνει την συνάρτηση η οποία παράγει διατάξεις ενός αριθμητικού συνόλου ακεραίων αριθμών που βρίσκονται σε ένα πίνακα.

Αρχικά η συνάρτηση λαμβάνει ως παραμέτρους ένα πίνακα από ακέραιους αριθμούς (αριθμητικό σύνολο), την τιμή του ορίου όπου θα γίνονται οι μεταθέσεις στοιχείων και την παράμετρο n η οποία είναι ίση με το μέγεθος του αρχικού πίνακα αλλά δεν δέχεται οποιαδήποτε αλλαγή κατά την εκτέλεση της συνάρτησης. Παραμένει δηλαδή σταθερή.

Μετέπειτα ελέγχουμε αν η τιμή του ορίου είναι ίση με 1. Δηλαδή εάν υπάρχει μόνο ένα στοιχείο για να διατάξουμε. Το γεγονός αυτό υποδηλώνει και το τέλος της διαδικασίας εύρεσης της επόμενης διάταξης. Έτσι τυπώνουμε το περιεχόμενο του πίνακά μας στοιχείο προς στοιχείο με την χρήση ενός βρόγχου από το 0 μέχρι το αρχικό μέγεθος του πίνακα (n).

Σε περίπτωση που το όριο δεν είναι ίσο με 1 η συνάρτηση υλοποιεί ένα βρόγχο με αρχή το 0 και τέλος το υφιστάμενο μέγεθος του ορίου.

Στον βρόγχο υλοποιούνται οι πιο κάτω διαδικασίες.

Αρχικά καλείται αναδρομικά η ίδια συνάρτηση με την μόνη διαφορά ότι το όριο μεταθέσεων μειώνεται κατά 1. Σύμφωνα με μαθηματικούς υπολογισμούς η συνάρτηση πρέπει να κληθεί N! φορές.

Αφού τυπωθεί η επόμενη διάταξη η συνάρτηση επιστρέφει στο σημείο που κλήθηκε. Εκεί ελέγχουμε αν το μέγεθος του υφιστάμενου ορίου είναι ζυγός αριθμός.

Στην περίπτωση που είναι μονός ανταλλάζουμε μεταξύ τους το πρώτο ([0]) και n-οστό στοιχείο του πίνακα.

Στην περίπτωση που είναι ζυγός ανταλλάζουμε μεταξύ τους το i-οστό ([i]) και n-1 οστό στοιχείο του πίνακα.

Έτσι δημιουργούμε μια καινούρια διάταξη των στοιχείων του πίνακα. Η προηγούμενη διάταξη των στοιχείων δεν μας ενδιαφέρει, αφού έχει τυπωθεί και η επόμενη κατάσταση του προγράμματός μας εξαρτάται μόνο από την παρούσα κατάσταση και όχι από τις πιο παλιές.

Συνολικά θα πραγματοποιηθούν N! – 1 μεταθέσεις αφού η πρώτη διάταξη είναι έτοιμη και είναι η σειρά με την οποία βρίσκονται αρχικά τα στοιχεία του πίνακα αποθηκευμένα.

```
1 2 3
2 1 3
3 1 2
1 3 2
2 3 1
3 2 1
```

Παράδειγμα
εκτέλεσης για N=3

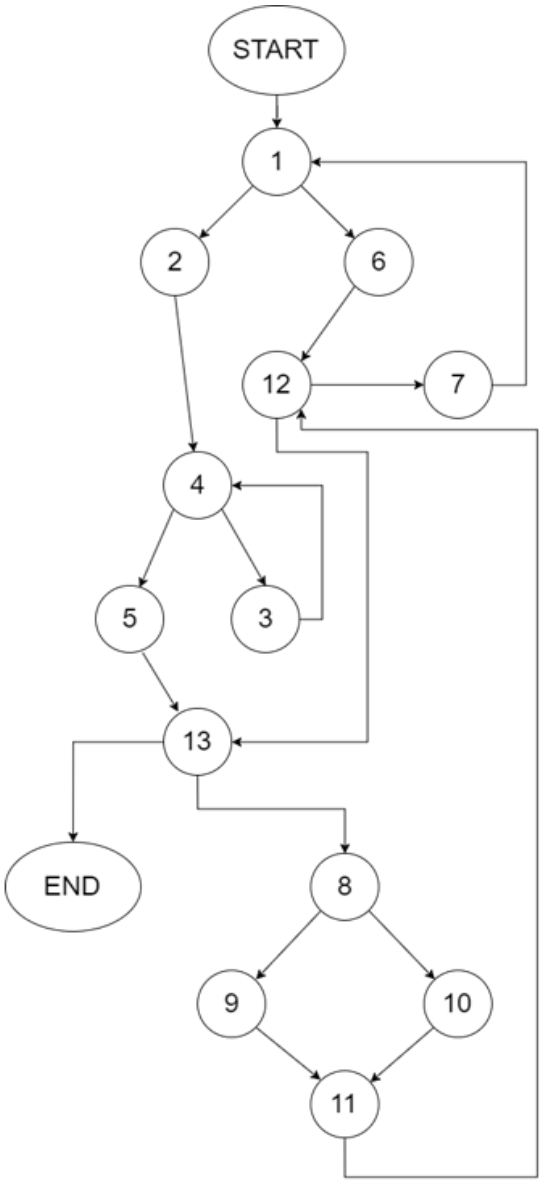
```
1 2 3 4    2 3 4 1    2 4 1 3
2 1 3 4    3 2 4 1    4 2 1 3
3 1 2 4    4 1 3 2    1 2 4 3
1 3 2 4    1 4 3 2    2 1 4 3
2 3 1 4               1 4 3 2
3 2 1 4               4 3 1 2
4 2 3 1               1 3 4 2
2 4 3 1               3 1 4 2
3 4 2 1               4 1 2 3
4 3 2 1               1 4 2 3
```

Παράδειγμα
εκτέλεσης για N=4

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ ΤΗΣ ΕΠΙΛΥΣΗΣ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΣΕ ΣΥΜΒΟΛΙΚΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΠΕΡΙΓΡΑΦΗ ΤΟΥ

Στο πιο κάτω διάγραμμα μπορούμε να δούμε την ροή της συνάρτησης την οποία χωρίσαμε σε BB. Η ετικέτα START υποδηλώνει την εισαγωγή δεδομένων στην συνάρτηση για πρώτη φορά ενώ η ετικέτα END υποδηλώνει τον οριστικό τερματισμό της συνάρτησης. Το BB13 μπορεί είτε να συνεχίσει στο BB8 όπου η συνάρτηση συνεχίζει την λειτουργία της μέσα στον επαναληπτικό βρόγχο που την κάλεσε είτε να πάει στο END όπου επιστρέφει στην κύρια συνάρτηση φόρτωσής της (main).

Στα BB1,2,6 καθώς και στα BB4,5,3 παρατηρούμε την υλοποίηση βρόγχων όπως προαναφέρθηκαν, ενώ στα BB8,9,10,11 παρατηρούμε την υλοποίηση υποθετικής συνθήκης.



ΠΕΡΙΓΡΑΦΗ ΚΩΔΙΚΑ ΣΥΜΒΟΛΙΚΗΣ ΓΛΩΣΣΑΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Πιο κάτω περιγράφονται οι λειτουργίες που πραγματοποιούνται μέσα στην συνάρτηση ανά BB. Η συνάρτηση main δεν περιγράφεται κάπου στο δοκίμιο για τον λόγο ότι δεν υλοποιεί κάτι σημαντικό που αφορά τον αλγόριθμο. Στην main διαβάζεται από αρχείο δοκιμής το αριθμητικό σύνολο που θα λάβει σαν είσοδο η συνάρτηση και δημιουργείται ένας πίνακας στον σωρό όπου αποθηκεύονται τα δεδομένα που διαβάστηκαν. Επίσης, η main προνοεί έτσι ώστε να λάβει τα σωστά ορίσματα η συνάρτηση του αλγόριθμου. Μπορείτε να δείτε τον κώδικα της main στο Παράρτημα «Α».

Ακολουθεί περιγραφή των BB των οποίων ο κώδικας βρίσκεται στο Παράρτημα «Β».

BB1 : Σε αυτό το Basic Block αρχικά δεσμεύεται χώρος στην στοίβα για την λειτουργία της συνάρτησης. Ακολούθως αποθηκεύονται στην στοίβα οι παράμετροι που λαμβάνει η συνάρτηση (table address , size και n). Επίσης ελέγχεται αν το size είναι άνισο με 1. Στην περίπτωση που η συνθήκη είναι αληθής πάει στο BB6 αλλιώς πάει στο BB2.

BB2 : Σε αυτό το Basic Block αρχικοποιείται στην στοίβα με την τιμή 0, ο μετρητής j του βρόγχου που τυπώνει τα στοιχεία του πίνακα. Μετά πάει στο BB4.

BB3 : Σε αυτό το Basic Block φορτώνεται από την στοίβα η διεύθυνση του πίνακα με τα στοιχεία που θέλουμε να τυπώσουμε και ακολούθως ανακτείται και τυπώνεται το επόμενο στοιχείο του πίνακα. Τέλος αυξάνεται κατά 1 ο μετρητής j και αποθηκεύεται η νέα του τιμή στην στοίβα. Μετά πάει στο BB4.

BB4 : Σε αυτό το Basic φορτώνονται από την στοίβα ξανά ο μετρητής j και το n. Ακολούθως ελέγχουμε αν το j είναι μικρότερο από το n. Στην περίπτωση που η συνθήκη είναι αληθής πάει στο BB3, αλλιώς πάει στο BB5.

BB5 : Σε αυτό το Basic Block αλλάζουμε γραμμή αφού τυπώσαμε την προηγούμενη διάταξη και πάει στο BB13.

BB6 : Σε αυτό το Basic Block αρχικοποιείται στην στοίβα με την τιμή 0 ο μετρητής i του βρόγχου που καλεί αναδρομικά την συνάρτηση και υλοποιεί τις μεταθέσεις των στοιχείων του πίνακα. Μετά πάει στο BB12.

BB7 : Σε αυτό το Basic Block αφαιρούμε από size 1 (size-1) και καλούμε τον εαυτό της συνάρτησής μας. (Αρχίζει από την αρχή. Δηλαδή πάει στο BB1).

BB8 : Σε αυτό το Basic Block φορτώνουμε την τιμή του size αφού η συνάρτηση συνεχίζει από το σημείο που επέστρεψε αφού κάλεσε τον εαυτό της. Μετέπειτα βρίσκει αν η τιμή του size είναι μονός αριθμός. Αν ναι, τότε πάει στο BB9 αλλιώς πάει στο BB 10.

BB9: Σε αυτό το Basic Block φορτώνεται από την στοίβα η τιμή του size ενώ ταυτόχρονα αφαιρούμε από αυτήν 1 για να υπολογίσουμε σωστά το offset στο οποίο βρίσκεται το στοιχείο table[size-1]. Ακολούθως υλοποιείται ανταλλαγή (swap) των στοιχείων table[0] και table[size-1] του πίνακα. Μετά πάει στο BB11.

BB10: Σε αυτό το Basic Block φορτώνονται από τη στοίβα η διεύθυνση του πίνακα, το size και η τιμή του μετρητή i. Υπολογίζουμε τα offset για τα στοιχεία του πίνακα

table[size-1] και table[i] και ακολούθως τα ανταλλάζουμε μεταξύ τους. Μετά πάει στο BB11.

BB11: Σε αυτό το Basic Block αυξάνουμε την τιμή του μετρητή i κατά 1 και την αποθηκεύουμε στην στοίβα. Μετά πάει στο BB12.

BB12: Σε αυτό το Basic Block φορτώνουμε την τιμή του μετρητή i και του size και τις συγκρίνουμε μεταξύ τους. Αν το i είναι μικρότερο από το size τότε πάει στο BB7 αλλιώς πάει στο BB13.

BB13: Σε αυτό το Basic Block αποδεσμεύεται ο χώρος που κρατήθηκε για την συνάρτηση στην στοίβα και επιστέφει πίσω στο σημείο που κλήθηκε η συνάρτηση.

ΠΡΟΣΠΑΘΕΙΕΣ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ ΤΟΥ ΚΩΔΙΚΑ ΣΥΜΒΟΛΙΚΗΣ ΓΛΩΣΣΑΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αφού ο αλγόριθμος υλοποιήθηκε σε C και μετέπειτα σε Assembly ARM v8-A εφαρμόστηκαν σε αυτό τροποποιήσεις έτσι ώστε να βελτιστοποιηθεί ο χρόνος εκτέλεσης του προγράμματος.

Στον κώδικα της Assembly πραγματοποιήθηκαν 3 είδη βελτιστοποιήσεων:

(α) Λογική λειτουργίας του κώδικα – χωρίς αλλαγές στον αρχικό αλγόριθμο.

(β) Χρήση περισσότερων προσωρινών καταχωριτών για αποφυγή συνεχούς φόρτωσης και αποθήκευσης δεδομένων καθώς και αποφυγή Hazards.

(γ) Χρήση εναλλακτικών πιο γρήγορων εντολών.

Ακολουθούν περιγραφές για τις βελτιστοποιήσεις που έγιναν ανά BB των οποίων ο κώδικας βρίσκεται στο Παράρτημα «Γ».

BB1 : Σε αυτό το Basic Block αφαιρέθηκαν οι αντίστοιχες γραμμές του «Πίνακα με basic blocks κώδικα assembly πριν τις βελτιστοποιήσεις» με αριθμό 7,8,9. Η υλοποίησή τους δεν ήταν απαραίτητη αφού οι τιμές των temporary registers στην συνάρτηση δεν επηρεάζονταν από κάποια ενδιάμεση πράξη έτσι ώστε να ξαναφορτώσουμε τις τιμές του από το Stack.

BB2 : Το BB δεν δύναται για περεταίρω βελτιστοποίηση.

BB3 : Το BB δεν δύναται για περεταίρω βελτιστοποίηση.

BB4 : Το BB δεν δύναται για περεταίρω βελτιστοποίηση.

BB5 : Το BB δεν δύναται για περεταίρω βελτιστοποίηση.

BB6 : Το BB δεν δύναται για περεταίρω βελτιστοποίηση.

BB7 : Σε αυτό το Basic Block αφαιρέθηκαν οι αντίστοιχες γραμμές του «Πίνακα με basic blocks κώδικα assembly πριν τις βελτιστοποιήσεις» με αριθμό 43,44. Η υλοποίησή τους δεν ήταν απαραίτητη αφού οι καταχωρητές στους οποίους φορτωνόταν η τιμή τους είχαν την σωστή τιμή. Δηλαδή οι προηγούμενες πράξεις δεν επηρέαζαν το περιεχόμενό τους.

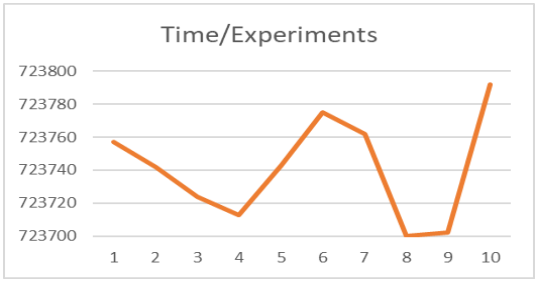
BB8 : Σε αυτό το Basic Block έγιναν αλλαγές ως προς την διευκόλυνση διεξαγωγής του Pipeline. Δηλαδή στις αντίστοιχες γραμμές του «Πίνακα με basic blocks κώδικα assembly πριν τις βελτιστοποιήσεις» με αριθμό 52,53,54 χρησιμοποιήθηκαν περισσότεροι καταχωρητές για να αποφευχθούν πιθανά Hazards. Η βελτιστοποίηση αυτή έφερε αμυδρά αποτελέσματα στην βελτίωση του χρόνου για τον λόγο ότι ο επεξεργαστής έχει το δικό του Forwarding Unit το οποίο μεριμνά για τέτοιες περιπτώσεις. Επίσης σε αυτό το Basic Block προστέθηκαν 2 γραμμές κώδικα όπως φαίνεται στον «Πίνακας με basic blocks κώδικα assembly μετά τις βελτιστοποιήσεις (συμπεριλαμβάνονται μόνο τα βελτιστοποιημένα basic blocks)» στις γραμμές 16 , 17. Αυτό είναι μια αλλαγή τύπου λογικής. Πιο αναλυτικά αντί να φορτώνουμε σε δύο διαφορετικά BB τις τιμές από την στοίβα, τις φορτώνουμε μία φορά πριν διαλέξω σε πιο κλαδί του προγράμματός μου θα μεταφερθώ. Έτσι την ίδια στιγμή μειώνεται και η συνολική έκταση που λαμβάνει ο κώδικας του προγράμματος.

BB9: Σε αυτό το Basic Block έχουν αφαιρεθεί όπως αναφέρθηκε στο προηγούμενο BB οι αντίστοιχες γραμμές του «Πίνακα με basic blocks κώδικα assembly πριν τις βελτιστοποιήσεις» με αριθμό 58, 59. Επίσης αφαιρέθηκαν οι γραμμές 61, 62, 63 και η εντολή της γραμμής 64 αντικαταστάθηκε με την εντολή
ldr w6,[x0, x1 , lsl 2]. Η εντολή αυτή προτιμήθηκε περισσότερο για τον λόγο ότι προκαλεί το ίδιο αποτέλεσμα που προκαλούσαν οι προηγούμενες τρείς γραμμές μαζί, καθώς επίσης και για τον λόγο ότι η εύρεση του offset του πίνακα με την χρήση της lsl είναι πιο ταχύς διαδικασία σύμφωνα με το εγχειρίδιο βελτιστοποιήσεων της ARM κατά 2 μsec. Επίσης στην περίπτωση υλοποίησης της νέας συνδυαστικής εντολής παραλείπουμε την εντολή add x1,x1,x0 με αποτέλεσμα να εξοικονομούμε 1 μsec.

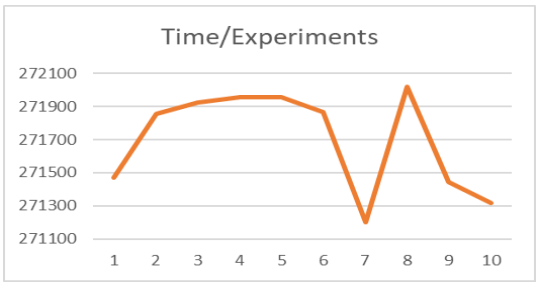
BB10: Σε αυτό το Basic Block έγιναν οι ανάλογες αλλαγές οι οποίες προαναφέρθηκαν στο πιο πάνω BB. Οι λειτουργίες αυτών των BB είναι πανομοιότυπες με την μόνη διαφορά ότι βρίσκουν διαφορετικό αριθμό offset.

ΣΥΓΚΡΙΣΗ ΤΟΥ ΚΩΔΙΚΑ ΣΥΜΒΟΛΙΚΗΣ ΓΛΩΣΣΑΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΜΕ ΤΟΝ ΑΝΤΙΣΤΟΙΧΟ ΚΩΔΙΚΑ ΠΟΥ ΠΑΡΑΓΕΙ Ο GCC

Τα αποτελέσματα που προκύπτουν στις πιο κάτω γραφικές παραστάσεις αφορούν μέγεθος συνόλου διατάξεων 10. Δηλαδή για N = 10, άρα ο αλγόριθμος επαναλαμβάνεται για 10! Φορές = 3628800.

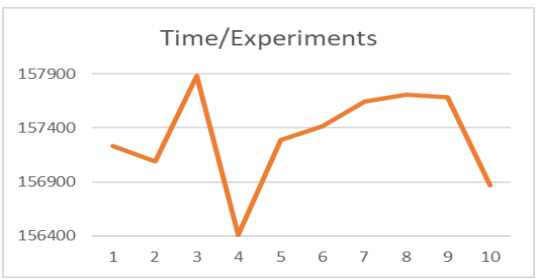


Η πιο πάνω γραφική παράσταση μας δείχνει τον χρόνο εκτέλεσής του προγράμματος με gcc -O0 για 10 πειράματα. Ο μέσος χρόνος εκτέλεσης του προγράμματος είναι 723741 μsec.



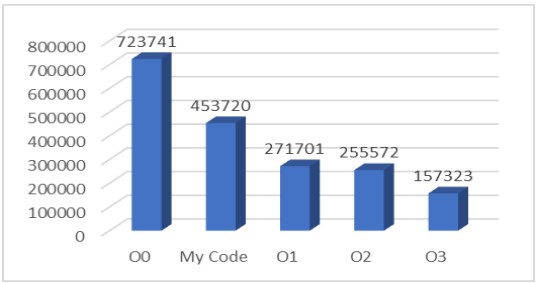
Η πιο πάνω γραφική παράσταση μας δείχνει τον χρόνο εκτέλεσής του προγράμματος με gcc -O1 για 10 πειράματα. Ο μέσος χρόνος εκτέλεσης του προγράμματος είναι 271701 μsec.

Η πιο πάνω γραφική παράσταση μας δείχνει τον χρόνο εκτέλεσής του προγράμματος με gcc -O2 για 10 πειράματα. Ο μέσος χρόνος εκτέλεσης του προγράμματος είναι 255572 μsec.

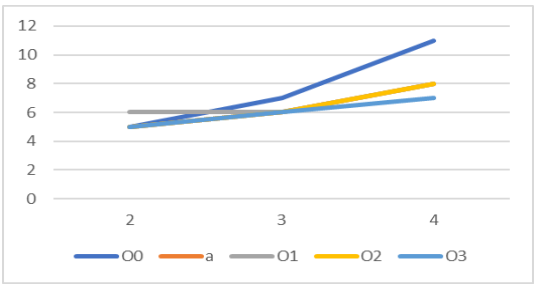


Η πιο πάνω γραφική παράσταση μας δείχνει τον χρόνο εκτέλεσής του προγράμματος με gcc -O3 για 10 πειράματα. Ο μέσος χρόνος εκτέλεσης του προγράμματος είναι 157323 μsec.

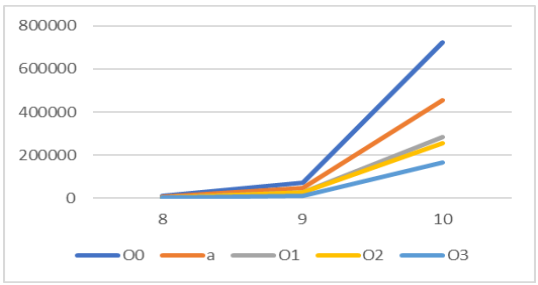
Η πιο πάνω γραφική παράσταση μας δείχνει τον χρόνο εκτέλεσής του προγράμματος που δεν δημιουργήθηκε από την μηχανή για 10 πειράματα. Ο μέσος χρόνος εκτέλεσης του προγράμματος είναι 453720 μsec.



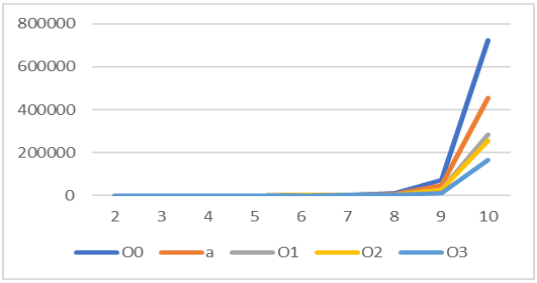
Στην πιο πάνω γραφική παράσταση παρουσιάζονται αναλυτικά οι χρόνοι που χρειάζονται οι 4 περιπτώσεις κώδικα.



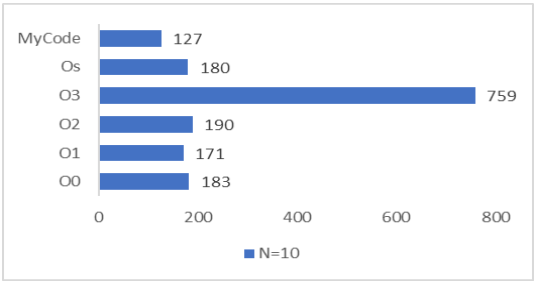
Η πιο πάνω γραφική παράσταση μας δείχνει τον χρόνο (σε μsec) που χρειάζονται τα προγράμματα ανά τύπο βελτιστοποίησης για να εκτελεστούν για N =1 , 2 , 3. (όπου α MyCode).



Η πιο πάνω γραφική παράσταση μας δείχνει τον χρόνο (σε μsec) που χρειάζονται τα προγράμματα ανά τύπο βελτιστοποίησης για να εκτελεστούν για N =8 , 9 , 10 . (όπου α MyCode).



Η πιο πάνω γραφική παράσταση μας δείχνει τον χρόνο (σε μsec) που χρειάζονται τα προγράμματα ανά τύπο βελτιστοποίησης για να εκτελεστούν για N =1 , 2 , 3, 4, 5, 6, 7, 8, 9, 10 . (όπου α MyCode).



Η πιο πάνω γραφική παράσταση παρουσιάζει τον τύπο βελτιστοποίησης σε σχέση με την έκταση του κώδικα σε γραμμές. Σε αυτό το γράφημα προστέθηκε η βελτιστοποίηση τύπου Os για τον λόγο ότι αφορά μόνο την έκταση σε κείμενο του προγράμματος. Δεν συμπεριλήφθηκε στις προηγούμενες μετρήσεις χρόνου αφού το Os δεν αφορά την επίδοση του προγράμματος σε χρόνο. Παραταύτα για λόγους σύγκρισης χρόνου και έκτασης προγράμματος έγινε πείραμα όπου το Os χρειάζεται κατά μέσο όρο 350600 μsec.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Σύμφωνα με τις πιο πάνω μετρήσεις παρατηρούμε ότι οι μέσοι χρόνοι εκτέλεσης των προγραμμάτων για O0 μέχρι O1 παρουσιάζουν μείωση κατά 62,5% , για O1 μέχρι O2 παρουσιάζουν μείωση κατά 6% και για O2 μέχρι O3 παρουσιάζουν μείωση κατά 38%. Είναι φανερό πως υπάρχει μικρή διαφορά μεταξύ του O1 και του O2 ενώ μεγάλη διαφορά παρουσιάζεται στο O1 σε σχέση με το O0 και στο O3 σε σχέση με το O2. Επίσης, ο O1 είναι προτιμότερος από τον Os επειδή πετυχαίνει καλύτερα αποτελέσματα σε χρόνο ενώ ταυτόχρονα έχει πιο μικρή έκταση. Ο MyCode έχει την μικρότερη έκταση από όλους τους κώδικες και μια ενδιάμεση επίδοση ανάμεσα στους O0 και O1. Ακόμη παρατηρούμε πως επαληθεύεται ο πίνακας της πρώτης σελίδας αφού οι χρόνοι εκτέλεσης ακολουθούν εκθετική μορφή ανάλογα με την αύξηση του N το οποίο εμείς κάναμε πειράματα για N = 10. Τελικά, αξίζει να σημειώσουμε ότι ο η καλύτερη έκδοση βελτιστοποίησης είναι η O2 αφού ο O3 για μεγάλα N θα χρειαστεί να υλοποιήσει τεχνικές loop unrolling πολλές φορές με αποτέλεσμα να αυξάνεται το μέγεθος του κώδικα εκθετικά , γεγονός που ίσως να προκαλέσει προβλήματα .

ΑΝΑΦΟΡΕΣ

[1] R. Sedgewick, "Permutation Generation Methods", ACM Computing Surveys, vol. 9, no. 2, pp. 137-164, 1977.

[2] B. Heap, "Permutations by Interchanges", The Computer Journal, vol. 6, no. 3, pp. 293-298, 1963.

[3] "Why does Heap's algorithm work?", Ruslanledesma.com, 2017. [Online]. Available: http://ruslanledesma.com/2016/06/17/why-does-heap-work.html. [Accessed: 26- Nov- 2017].

[4] "Heap's Algorithm – Crooked Code", Crookedcode.com, 2017. [Online]. Available: http://crookedcode.com/tag/heaps-algorithm/. [Accessed: 26- Nov- 2017].

[5] "Tim Robinson", Partario.com, 2017. [Online]. Available: http://www.partario.com/blog/archive/2009/08/1/. [Accessed: 26- Nov- 2017].

[6] "Heap's Algorithm for generating permutations - GeeksforGeeks", GeeksforGeeks, 2017. [Online]. Available: http://www.geeksforgeeks.org/heaps-algorithm-for-generating-permutations/. [Accessed: 26- Nov- 2017].

ΠΑΡΑΡΤΗΜΑ «Α»

ΠΙΝΑΚΑΣ ΜΕ BASIC BLOCKS ΚΩΔΙΚΑ ASSEMBLY ΠΡΙΝ ΤΙΣ ΒΕΛΤΙΣΤΟΠΟΙΗΣΕΙΣ

```
//*****************************| bb1 |*****************************
[1]  permutation:
[2]    stp    x29, x30, [sp, -48]!      //First (!) move the stack pointer 32
                                         //bytes down and then store double
                                         //x29 and x30
[3]    add    x29, sp, 0                //Copy the stack pointer into the
                                         //frame pointer
[4]    str    x0,[x29,24]               //Store the table address in the stack
[5]    str    w1,[x29,20]               //Store the Size in the stack
[6]    str    w2,[x29,16]               //Store the n in the stack
[7]    ldr    x0,[x29,24]               //Load the table address
[8]    ldr    w1,[x29,20]               //Load the Size
[9]    ldr    w2,[x29,16]               //Load n
[10]   cmp    w1,1                      //Compare Size to 1
[11]   bne    sinexia_loop              //If not equals go to sinexia_loop
[12] //*************************| bb2 |*****************************
[13]   str     wzr,[x29,40]             //Store 0 in stack ( j counter)
[14]   b       elexos_j                 //Go to label elexos_j
[15] //*************************| bb3 |*****************************
[16] print_array:                       //Start of label
[17]   ldrsw  x0, [x29, 40]             //Load the counter j
[18]   lsl    x0, x0, 2                 //Find the offset of j element in array
[19]   ldr    x1, [x29, 24]             //Load the address of the table
[20]   add    x0, x1, x0                //Find the total memory offset
[21]   ldr    w1, [x0]                  //Load the element table[j]
[22]   adr    x0, check1                //Load to x0 the label check1
[23]   bl     printf                    //Print the element
[24]   ldr    w0, [x29, 40]             //Load j from the stack
[25]   add    w0, w0, 1                 //Add to j 1
[26]   str    w0, [x29, 40]             //Store the new j
[27] //*************************| bb4 |*****************************
[28] elexos_j:                          //Start of label
[29]   ldr    w1, [x29, 40]             //Load j from stack
[30]   ldr    w0, [x29, 16]             //Load n from stack
[31]   cmp    w1, w0                    //Compare j and n
[32]   blt    print_array               //If I is less than n go to print_array
[33] //*************************| bb5 |*****************************
[34]   adr    x0,newline                //Load to x0 the label newLine
[35]   bl     printf                    //Call printf fuction to change line
[36]   b      exodos                    //Go to label exodos
[37] //*************************| bb6 |*****************************
[38] sinexia_loop:                      //Start of label
[39]   str    wzr,[x29,44]              //Store to stack counter i = 0
[40]   b      elexos_i                  //Go to elexos_i
[41] //*************************| bb7 |*****************************
[42] loop2:                             //Start of label
[43]   ldr x0,[x29,24]                  //Load the table address
[44]   ldr w1,[x29,20]                  //Load Size
[45]   sub w1,w1,1                      // Size = Size -1
[46]   ldr w2,[x29,16]                  //Load n
[47]   bl permutation                   //Call function permutation
[48] //*************************| bb8 |*****************************
[49]   ldr    w1,[x29,20]               //Load Size
[50]   mov    w11,2                     //Mov to w11 the number 2
[51]   // The following process finds the modulo Size
[52]   sdiv   w12,w1,w11
[53]   mul    w12,w12,w11
[54]   sub    w12,w1,w12
[55]   cmp    w12,1                     //Compare the modulo to 1 in order to
                                         //find if size is even or odd number
[56]   bne    else_swap                 //If is not odd go to else_swap
[57] //*************************| bb9 |*****************************
[58]   ldr    w0,[x29,24]               //Load address of the table from stack
[59]   ldr    x1,[x29,20]               //Load Size from stack
[60]   add    w1,w1,-1                  //Size = Size -1
[61]   mov    x19,4                     //Move to x19 number  4
[62]   mul    x1,x1,x19                 //x1 =  x1 * x19
[63]   add    x1,x1,x0                  //x1 = x1 + x0
[64]   ldr    w5,[x0]                   //Load to w5 table[0]
[65]   ldr    w6,[x1]                   //Loaf to w6 table[Size-1]
[66]   str    w5,[x1]                   //Store w5 to  table[Size-1]
[67]   str    w6,[x0]                   //Store w6 to  table[0]
[68]   b      allagi_x10                //Go to allagi_x10
[69] //*************************| bb10 |*****************************
[70] else_swap:                         //Start of label
[71]   ldr    w0,[x29,24]               //Load table address from stack
[72]   ldr    x1,[x29,20]               //Load Size
[73]   ldr    w3,[x29,44]               //Load I to w3
[74]   mov    x19,4                     //Move to x19 number 4
[75]   mul    x3,x3,x19                 //x3 = x3 * x19
[76]   add    x3,x3,x0                  //x0 = x0 + x3
[77]   add    w1,w1,-1                  //Size = Size -1
[78]   mul    x1,x1,x19                 //x1 = x1 * x19
[79]   add    x1,x1,x0                  //x1 = x1 + x0
[80]   ldr    w5,[x3]                   //Load to w5 table[i]
[81]   ldr    w6,[x1]                   //Loaf to w6 table[Size-1]
[82]   str    w5,[x1]                   //Store w5 to  table[Size-1]
[83]   str    w6,[x3]                   //Store w to  table[i]
[84] //*************************| bb11 |*****************************
[85] allagi_x10:                        //Start of label
[86]   ldr    w10,[x29,44]              //Load counter i
[87]   add    w10,w10,1                 //Increase counter by 1
[88]   str    w10,[x29,44]              //Store the new counter to stack
[89] //*************************| bb12 |*****************************
[90] elexos_i:                          //Start label
[91]   ldr    w10,[x29,44]              //Load counter i from stack
[92]   ldr    w1,[x29,20]               //Load size
[93]   cmp    w10,w1                    //Compare i and size
[94]   blt    loop2                     //If counter i is less than size goto
                                         //loop2
[95] //*************************| bb13 |*************************
[96] exodos:                            //Start label
[97]   ldp    x29, x30, [sp], 48        //Release reserved space from stack
[98] ret                                //Return to the loading function
```

ΠΑΡΑΡΤΗΜΑ «Β»

ΠΙΝΑΚΑΣ ΜΕ BASIC BLOCKS ΚΩΔΙΚΑ ASSEMBLY ΜΕΤΑ ΤΙΣ ΒΕΛΤΙΣΤΟΠΟΙΗΣΕΙΣ (ΣΥΜΠΕΡΙΛΑΜΒΑΝΟΝΤΑΙ ΜΟΝΟ ΤΑ ΒΕΛΤΙΣΤΟΠΟΙΗΜΕΝΑ BASIC BLOACKS)

```
[1]  //**********************| bb1 |*******************************
[2]  permutation:
[3]    stp     x29, x30, [sp, -48]!        //First (!) move the stack pointer 32
                                           //bytes down and then store double
                                           //x29 and x30
[4]    add     x29, sp, 0                  //Copy the stack pointer into the
                                           //frame pointer
[5]    str x0,[x29,24]                     //Store the table address in the stack
[6]    str w1,[x29,20]                     //Store the Size in the stack
[7]    str w2,[x29,16]                     //Store the n in the stack
[8]    cmp    w1,1                         //Compare Size to 1
[9]    bne    sinexia_loop                 //If not equals go to sinexia_loop
[10] //**********************| bb7 |*******************************
[11] loop2:                                //Start of label
[12]   sub w1,w1,1                         // Size = Size -1
[13]   ldr w2,[x29,16]                     //Load n
[14]   bl permutation                      //Call function permutation
[15] //**********************| bb8 |*******************************
[16]   ldr     w0,[x29,24]                 //Load Size
[17]   ldr     w1,[x29,20]                 //Load Size
[18]   mov    w11,2                        //Move to w11 the number 2
[19]   // The following process finds the modulo Size
[20]   sdiv    w12,w1,w11
[21]   mul     w13,w12,w11
[22]   sub     w14,w1,w13
[23]   cmp    w14,1                        //Compare the modulo to 1 in order to
                                           //find if size is even or odd number
[24]   bne     else_swap                   //If is not odd go to else_swap
[25] //**********************| bb9 |*******************************
[26]   add     w1,w1,-1                    //Size = Size -1
[27]   ldr     w5,[x0]                     //Load to w5 table[0]
[28]   ldr     w6,[x0, x1 , lsl 2]         //Loaf to w6 table[Size-1]
[29]   str     w5,[x0,x1,lsl 2]            //Store w5 to  table[Size-1]
[30]   str     w6,[x0]                     //Store w6 to  table[0]
[31]   b       allagi_x10                  //Go to allagi_x10
[32] //**********************| bb10 |******************************
[33] else_swap:                            //Start of label
[34] ldr       w3,[x29,44]                 //Load I to w3
[35] add     w1,w1,-1                      //Size = Size -1
[36] ldr     w5,[x0, x3, lsl 2]            //Load to w5 table[i]
[37] ldr     w6,[x0, x1, lsl 2]            //Loaf to w6 table[Size-1]
[38] str     w5,[x0,x1, lsl 2]             //Store w5 to  table[Size-1]
[39]  str     w6,[x0,x3, lsl 2]            //Store w to  table[i]
```

ΠΑΡΑΡΤΗΜΑ «Γ»

ΠΙΝΑΚΑΣ ΜΕ ΚΩΔΙΚΑ ASSEMBLY ΠΟΥ ΑΦΟΡΑ ΤΗΝ ΣΥΝΑΡΤΗΣΗ ΦΟΡΤΩΣΗΣ MAIN

```
[1]            .data
[2]  FileOpenMode:  .string "r"          //File mode read
[3]  FileName:      .string "test.txt"   //The file to read
[4]  scanner:       .string "%d"         //Scanner to read data
[5]  check1:        .string "%d "        //Output format
[6]  newLine:       .string "\n"         //Output new line
[7]            .text
[8]            .global      main
[9]  main:
[10]   stp     x29, x30, [sp, -32]!      //First (!) move the stack pointer 32
                                         //bytes down and then store double
                                         //x29 and x30
[11]
[12]   add     x29, sp, 0                //Copy the stack pointer into the
                                         //frame pointer
[13]   // Open the File Pointer
[14]   adr     x1, FileOpenMode          //Load the file mode
[15]   adr     x0, FileName              //Load the file name
[16]   bl      fopen                     //Open the file
[17]   mov x20, x0                       // Store the file pointer into x20
[18]   mov     x0,40                     // Set the size of array where will
                                         //store the readen data to 40 bytes
[19]   bl      malloc                    //Call function malloc to create array
                                         //in heap memory
[20]   mov     x21,x0                    //Save a copy of the array address to
                                         //x21
[21]   mov     x22,x0                    //Save a copy of the array address to
                                         //x22
[22]   mov x19, 10                       //Move to x19 10 which is the
                                         //Number of data we will read from
                                         //file.
[23]   mov     x24, x19                  //Save a copy of x19 to
                                         //x24
[24]  loop:                              //Start of loop
[25]   add     x2, x21, 0                //Set the location where scanned data
                                         //will be stored
[26]   add     x21,x21,4                 //Increase x21 by 4. (next location)
[27]   adr     x1, scanner               //Load the scanner
[28]   mov     x0, x20                   // Move to x0 the pointer
[29]   bl      fscanf                    //Call function fscanf
[30]   add x19, x19, -1                  //Decrease x19 by 1.( Counter )
[31]   cbnz x19, loop                    //If x19 is not 0 goto label loop
[32]   mov     x0, x20                   // Get the File pointer
[33]   bl      fclose                    // Close the file.
[34]   mov     x0,x22                    //Get the address of the table[] in w0
[35]   mov     w1,w24                    //Get te value of variable Size in w1
[36]   mov     w2,w24                    //Get the value of n in w2
[37]   bl      permutation               //Call function permutation
[38]   ldp     x29, x30, [sp], 32        //Release reserved space from stack
[39]    Ret                              //Return to the loading function
```