Συστήματα Υπολογισμού Υψηλών Επιδόσεων (ECE 415)
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας

Διδάσκων: Χρήστος Δ. Αντωνόπουλος

1η Εργαστηριακή Άσκηση – Προθεσμία 14/10/2024, 23:59

1 Στόχοι

- Βηματική βελτιστοποίηση ακολουθιακής εφαρμογής σε CPU.
- Εξοικείωση με βασικές αρχές πειραματισμού και μετρήσεων σε υπολογιστικά συστήματα.
- Εξοικείωση με τον μεταγλωττιστή και τις βελτιστοποιήσεις.

2 Εισαγωγή

Η ανίχνευση ακμών είναι μια πολύ συνηθισμένη διαδικασία στην επεξεργασία εικόνας. Στα πλαίσια αυτής της εργασίας σας δίνεται έτοιμος κώδικας ο οποίος ανιχνεύει ακμές σε εικόνα σε τόνους του γκρι (grayscale) με χρήση του φίλτρου Sobel. Ο κώδικας διαβάζει μια εικόνα από αρχείο εισόδου, την επεξεργάζεται και παράγει την εικόνα εξόδου την οποία επίσης αποθηκεύει σε αρχείο. Τέλος, συγκρίνει το αποτέλεσμα που παρήγαγε σε σχέση με αυτό που παράγεται από μια σωστή υλοποίηση

(επίσης σας δίνεται ως υπόδειγμα) και υπολογίζει το PSNR.

Εσείς θα πρέπει να υλοποιήσετε διαδοχικά μια σειρά πολύ συνηθισμένων βελτιστοποιήσεων στον κώδικα που σας δίνεται. Για κάθε βελτιστοποίηση που εφαρμόζετε θα πρέπει να μετράτε το χρόνο εκτέλεσης και να τον συγκρίνετε με αυτόν της αμέσως προηγούμενης έκδοσης. Αυτή η διαδικασία θα πρέπει να επαναληφθεί τόσο με απενεργοποιημένες όσο και με ενεργοποιημένες τις βελτιστοποιήσεις του μεταγλωττιστή. Εξυπακούεται ότι μετά από κάθε βήμα θα πρέπει να επαληθεύετε την ορθότητα της υλοποίησής σας.

3 Υπόβαθρο

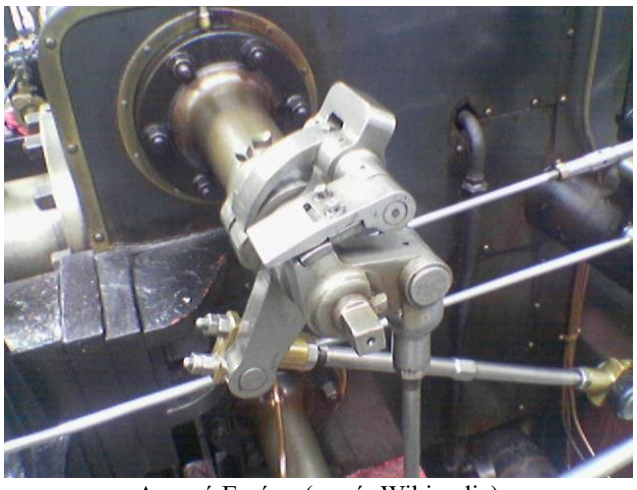
3.1 Τελεστής Sobel

Ο τελεστής Sobel χρησιμοποιείται στην επεξεργασία εικόνας για την υλοποίηση τεχνικών ανίχνευσης ακμών. Πρόκειται για έναν τελεστή διακριτής διαφόρισης. Σε κάθε σημείο (pixel) της εικόνας που εφαρμόζεται ο τελεστής, το αποτέλεσμα είναι η παράγωγος της φωτεινότητας (ή το μέτρο της) στο συγκεκριμένο σημείο.

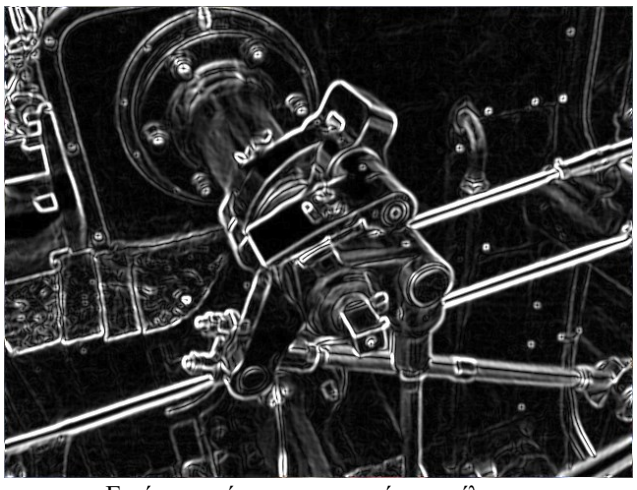
Ο τελεστής χρησιμοποιεί δύο πίνακες 3x3 οι οποίοι συνελίσσονται με την αρχική εικόνα. Έναν ο οποίος εντοπίζει τις αλλαγές (και τις ακμές) στην οριζόντια κατεύθυνση και ένας στην κατακόρυφη. Έστω A η αρχική εικόνα, O_x, O_y οι 2 πίνακες, * ο τελεστής δισδιάστατης συνέλιξης και G_x, G_y τα αποτελέσματα της εφαρμογής του O_x και O_y αντίστοιχα στην αρχική εικόνα:

$O_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} O_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} G_x = O_x \cdot A, G_y = O_y \cdot A$ενώ το αποτέλεσμα (μέτρο της παραγώγου) μπορεί να υπολογιστεί ως $Y = \sqrt{{G_x}^2 + {G_y}^2}$

Δείτε για παράδειγμα το αποτέλεσμα της εφαρμογής του φίλτρου Sobel στην παρακάτω εικόνα:



Αρχική Εικόνα (πηγή: Wikipedia)



Εικόνα μετά την εφαρμογή του φίλτρου

3.2 PSNR

Το PSNR (peak signal to noise ratio) ποσοτικοποιεί το λόγο μεταξύ της ισχύος ενός σήματος και της ισχύος του θορύβου που υποβαθμίζει την ποιότητά του. Το PSNR χρησιμοποιείται πολύ συχνά για τη σύγκριση της ποιότητας εικόνων, ειδικά για την ποσοτικοποίηση των ιδιοτήτων αλγορίθμων συμπίεσης. Έστω μια grayscale εικόνα I, μεγέθους m x n, χωρίς θόρυβο, η οποία χρησιμοποιείται ως πρότυπο σύγκρισης, και έστω μια grayscale εικόνα K την οποία θέλουμε να συγκρίνουμε με την I.

Το μέσο τετραγωνικό σφάλμα μεταξύ τους υπολογίζεται ως

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

ενώ το PSNR ορίζεται ως $PSNR = 10\log_{10}\left(\frac{{MAX_I}^2}{MSE}\right)$με MAX_I^2 να είναι η μέγιστη πιθανή τιμή ενός pixel της εικόνας (για grayscale εικόνες όπου κάθε pixel αναπαρίσταται με ένα byte η τιμή αυτή είναι 255). Προφανώς, εάν οι 2 εικόνες είναι απολύτως ίδιες, το MSE είναι 0 και PSNR άπειρο.

3.3 Βελτιστοποιήσεις Κώδικα

Οι βελτιστοποιήσεις κώδικα σκοπό έχουν κατά κανόνα σκοπό να παράξουν ταχύτερο κώδικα. Σε κάθε περίπτωση, ο βελτιστοποιημένος κώδικας θα πρέπει να παράγει τα ίδια ακριβώς αποτελέσματα με τον αρχικό. Παρακάτω παρουσιάζονται συνοπτικά ορισμένες συχνά χρησιμοποιούμενες βελτιστοποιήσεις κώδικα. Οι βελτιστοποιήσεις αυτές μπορούν να γίνουν από τον προγραμματιστή, αλλά συχνά δοκιμάζει να τις εφαρμόσει και ο μεταγλωττιστής.

3.3.1 Loop Interchange (εναλλαγή βρόχων)

Είναι η διαδικασία εναλλαγής 2 φωλιασμένων (nested) loops. Παράδειγμα:

for (i=0; i < 10; i++)
  for (j = 0; j < 20; j++)
  a[i, j] = i + j;
	for (j=0; j < 20; j++)
	  for (i=0; i < 10; i++)
	  a[i, j] = i + j;

Αρχικός κώδικας	Μετά την εναλλαγή loops

Η βελτιστοποίηση αυτή συνήθως χρησιμοποιείται για να βελτιώσει την τοπικότητα στη μνήμη, να διασφαλίσει δηλαδή ότι τα στοιχεία πινάκων θα προσπελάζονται με τη σειρά με την οποία είναι αποθηκευμένα στη μνήμη και όχι με άλματα.

Το loop interchange αλλάζει τη σειρά με την οποία γίνονται οι αναθέσεις νέων τιμών στη μνήμη, συνεπώς ΔΕΝ είναι πάντα επιτρεπτή. Θα πρέπει κανείς να ελέγξει αν οι εντολές κάποιας επανάληψης έχουν εξαρτήσεις από εντολές άλλων επαναλήψεων (αν δηλαδή υπάρχει περίπτωση μια επανάληψη να γράφει ή να διαβάσει δεδομένα που γράφει και κάποια άλλη επανάληψη, καθώς στην περίπτωση αυτή δεν επιτρέπεται να αλλάξει η σχετική σειρά εκτέλεσης αυτών των επαναλήψεων). Σίγουρα πάντως δεν υπάρχουν εξαρτήσεις και η εναλλαγή είναι αποδεκτή κάθε σύμβολο (μεταβλητή ή πίνακας) εμφανίζεται μέσα στον βρόχο είτε αριστερά, είτε δεξιά από τις εντολές ανάθεσης, και όχι άλλοτε στο αριστερό και άλλοτε στο δεξί μέλος.

3.3.2 Loop Unrolling (ξεδίπλωμα βρόχων)

Κατά το loop unrolling επαναλαμβάνουμε πολλαπλές φορές τον κώδικα του σώματος ενός loop, ώστε να μειώσουμε τον αριθμό των επαναλήψεών του. Για παράδειγμα:

int x;
 	
for (x=0; x<100; x++)
  delete(x);
	int x;
	
	for (x=0; x<100; x+=5) {
	  delete(x);
	  delete(x+1);
	  delete(x+2);
	  delete(x+3);
	  delete(x+4);
	}

Αρχικός κώδικας	Μετά το ξεδίπλωμα του loop κατά 5

Το αποτέλεσμα του ξεδιπλώματος είναι να μειωθεί σημαντικά ο αριθμός των εντολών που αφορούν τη διαχείριση του loop (αυξήσεις μετρητών, έλεγχοι, άλματα) και οι αντίστοιχες επιβαρύνσεις. Επιπλέον, επειδή αυξάνεται ο κώδικας εντός του σώματος του loop είναι δυνατόν ο επεξεργαστής ή ο μεταγλωττιστής να αναδιοργανώσει πιο αποτελεσματικά τη σειρά εκτέλεσης των εντολών. Στον αντίποδα βέβαια, αυξάνει το μέγεθος του τελικού εκτελέσιμου και δυνητικά είναι δυνατόν να αυξήσει και τις αστοχίες στην instruction cache. Επίσης, τείνει να αυξήσει τον αριθμό των απαιτούμενων καταχωρητών.

Το ξεδίπλωμα ενός loop μπορεί να είναι πλήρες (και να εξαφανιστεί τελείως το loop) αν το loop είναι μικρό και γνωστού μεγέθους κατά το χρόνο συγγραφής του κώδικα. Σε αντίθετη περίπτωση αρκούμαστε σε μερικό ξεδίπλωμα του βρόχου (π.χ. κατά 4, 8, 16, 20 κλπ). Εσείς πειραματιστείτε με ξεδίπλωμα έως το πολύ 16 φορών. Σημειώστε ότι στην περίπτωση μερικού ξεδιπλώματος του loop απαιτείται προσοχή αν δεν είμαστε βέβαιοι ότι ο συνολικός αριθμός επαναλήψεων διαιρείται ακριβώς με το βαθμό ξεδιπλώματος που έχουμε επιλέξει.

Σημειώστε ότι η συγκεκριμένη βελτιστοποίηση είναι πάντα νόμιμη.

3.3.3 Loop Fusion

Δύο διαδοχικά loops, τα οποία διατρέχουν το ίδιο εύρος συνενώνονται σε ένα, όπως φαίνεται στο παρακάτω παράδειγμα

int i, a[100], b[100];
 	
for (i = 0; i < 100; i++)
  a[i] = 1;
for (i = 0; i < 100; i++)
  b[i] = 2;
	int i, a[100], b[100];
	
	for (i = 0; i < 100; i++) {
	  a[i] = 1;
	  b[i] = 2;
	}

Αρχικός κώδικας	Μετά τη συνένωση των 2 διαδοχικών loops

Το loop fusion δεν είναι δυνατό αν τα loops δε διατρέχουν το ίδιο εύρος επαναλήψεων. Ενδέχεται να μην είναι δυνατό αν μεταξύ τους υπάρχει κώδικας που δε μπορεί να μετακινηθεί. Επίσης, δεν είναι επιτρεπτό αν η επανάληψη i του 2ου loop αναφέρεται σε δεδομένα που παράγονται στο 1ο loop σε επανάληψη μεταγενέστερη της i, γιατί εάν τα loops συνενωθούν θα υπάρχει παραβίαση των εξαρτήσεων.

Το loop fusion μειώνει τη διαχειριστική επιβάρυνση για την εκτέλεση των loops και παρέχει περισσότερο κώδικα στο σώμα του loop για τυχόν μεταγενέστερες βελτιστοποιήσεις από τον μεταγλωττιστή ή κατά την εκτέλεση του κώδικα. Δεν είναι πάντως απαραίτητο ότι το loop fusion θα βελτιώσει την επίδοση (π.χ. μπορεί να τη χειροτερέψει σε περιπτώσεις που η συνένωση χαλάσει την τοπικότητα των αναφορών στη μνήμη).

Υπάρχει και η αντίστροφη διαδικασία από το loop fusion, η οποία καλείται loop fission. Αφορά τη διάσπαση ενός loop σε περισσότερα διαδοχικά.

3.3.4 Function Inlining

Αντικαθιστά μια κλήση συνάρτησης με το σώμα της συνάρτησης στο σημείο της κλήσης. Είναι πάντα επιτρεπτή (με την επιφύλαξη των αναδρομικών συναρτήσεων). Παράδειγμα function inlining φαίνεται παρακάτω:

```
int pred(int x) {
  if (x == 0)
    return 0;
  else
    return x - 1;
}


int f(int y) {
  return pred(y) + pred(0) + pred(y+1);
}
```

Αρχικός κώδικας

```
int f(int y) {
  int temp;
  if (y == 0)
    temp  = 0;
  else
    temp  = y - 1; /* (1) */
  if (0 == 0)
    temp += 0;
  else
    temp += 0 - 1; /* (2) */
  if (y+1 == 0)
    temp += 0;
  else
    temp += (y + 1) - 1; /* (3) */

  return temp;
}
```

Μετά το inlining της συνάρτησης pred στην f

Πέρα από το προφανές όφελος της μείωσης του αριθμού των κλήσεων συναρτήσεων (και των επιβαρύνσεων που προκύπτουν), η βελτιστοποίηση αυτή καθιστά δυνατό πλήθος άλλων βελτιστοποιήσεων που δε θα ήταν εφικτό να εφαρμοστούν σε πολλαπλές συναρτήσεις. Για παράδειγμα, στον παραπάνω κώδικα είναι σαφές ότι ο έλεγχος (0 == 0) οδηγεί σε αποτέλεσμα πάντα αληθές και άρα δε χρειάζεται να γίνει. Επίσης δε χρειάζεται να εκτελεστούν οι προσθέσεις temp += 0 ενώ μπορεί να απλοποιηθεί και η πράξη temp += (y + 1) – 1.

Τα μειονεκτήματα είναι αντίστοιχα με αυτά του ξεδιπλώματος βρόχων.

3.3.5 Loop Invariant code motion

Αφορά τη μετακίνηση τμημάτων κώδικα έξω από το σώμα loops, εάν αφορούν υπολογισμούς που δεν επηρεάζονται από τις επαναλήψεις του loop. Για παράδειγμα:

```
for (int i = 0; i < n; i++) {
   x = y + z;
   a[i] = 6 * i + x * x;
}
```

Αρχικός κώδικας

```
x = y + z;
t1 = x * x;
for (int i = 0; i < n; i++) {
        a[i] = 6 * i + t1;
}
```

Μετά την αφαίρεση από το loop των invariants

3.3.6 Common Subexpression Elimination

Πρόκειται για την αναζήτηση εκφράσεων / υποεκφράσεων των οποίων ο υπολογισμός τείνει να επαναλαμβάνεται. Οι εκφράσεις υπολογίζονται μία μόνο φορά και χρησιμοποιούνται στη συνέχεια. Για παράδειγμα:

```
a = b * c + g;
d = b * c * e;
```

Αρχικός κώδικας

```
tmp = b * c;
a = tmp + g;
d = tmp * e;
```

Μετά την επαναχρησιμοποίηση του γινομένου b*c

Σημαντικές ευκαιρίες εφαρμογής της παραπάνω βελτιστοποίησης προκύπτουν κατά τον υπολογισμό της απομάκρυνσης από την αρχή του πίνακα για να προσπελάσουμε το στοιχείο σε συγκεκριμένες συντεταγμένες ενός πολυδιάστατου πίνακα.

Το μειονέκτημα είναι ότι ενδεχομένως να οδηγηθούμε σε μεγαλύτερη ζήτηση για registers για την αποθήκευση των ενδιάμεσων αποτελεσμάτων.

3.3.7 Strength Reduction

Πρόκειται για την αντικατάσταση "ακριβών" πράξεων με άλλες "φθηνότερες" με ισοδύναμο αποτέλεσμα. Για παράδειγμα:

c = 8;
for (i = 0; i < N; i++) {
  y[i] = c * i;
}
	c = 8;
	k = 0;
	for (i = 0; i < N; i++) {
	  y[i] = k;
	  k = k + c;
	}

Αρχικός κώδικας	Μετά την αντικατάσταση πολλαπλασιασμών από προσθέσεις

Συχνά χρησιμοποιούμενες αντικαταστάσεις είναι διαιρέσεις / πολλαπλασιασμοί με δυνάμεις του 2 από shifts, δυνάμεις από διαδοχικούς πολλαπλασιασμούς κλπ.

Ακόμα πιο περίπλοκες αριθμητικές πράξεις μπορούν να αντικατασταθούν από lookup tables, ειδικά αν εφαρμόζονται σε περιορισμένα εύρη διακριτών τιμών (π.χ. σε ένα μικρό εύρος ακεραίων). Σε αυτή την περίπτωση, τα αποτελέσματα των περίπλοκων αυτών αριθμητικών πράξεων υπολογίζονται εκ των προτέρων για όλες τις τιμές στις οποίες υπάρχει περίπτωση να εφαρμοστούν (συνήθως κατά το χρόνο της συγγραφής του κώδικα) και αποθηκεύονται σε έναν πίνακα. Κατά το χρόνο εκτέλεσης, αντί να εκτελούνται οι αντίστοιχες πράξεις, χρησιμοποιείται το προϋπολογισμένο αποτέλεσμα από τον πίνακα.

3.3.8 Υποβοήθιση Μεταγλωττιστή

Σημαντική βελτίωση ταχύτητας μπορεί να προκύψει αν δώσουμε πληροφορίες στον μεταγλωττιστή που δεν μπορεί να τις εξάγει από ανάλυση, προκειμένου να τον βοηθήσουμε στις αυτόματες βελτιστοποιήσεις του. Για παράδειγμα, θα μπορούσαμε να σημειώσουμε ορίσματα συναρτήσεων που δε θα αλλάξουν μέσα στη συνάρτηση με const (π.χ. const char *name για να δηλώσουμε ότι τα περιεχόμενα στη θέση μνήμης που δείχνει το name δε θα αλλάξουν μέσα στη συνάρτηση), να βάλουμε το πρόθεμα register μπροστά από τις δηλώσεις μεταβλητών που θα κρίναμε σωστό να μπουν σε καταχωρητές (π.χ. register int val), να χρησιμοποιήσουμε τη λέξη restrict στη δήλωση δεικτών για να δηλώσουμε ότι οι θέσεις μνήμης στην οποία δείχνουν θα προσπελαστούν αποκλειστικά και μόνο μέσω του συγκεκριμένου pointer (π.χ. int * restrict foo για να δηλώσετε ότι η μνήμη στην οποία δείχνει ο foo δεν υπάρχει περίπτωση να προσπελαστεί με άλλο τρόπο παρά μόνο μέσω του foo) κλπ.

3.4 Μετρήσεις Χρόνου

Στα πλαίσια του μαθήματος χρειάζεται να μετράτε το χρόνο εκτέλεσης του κώδικά σας. Υπάρχουν διαφορετικές μέθοδοι μέτρησης χρόνου, με διαφορετική ανάλυση. Θα πρέπει κάθε φορά να επιλέγετε μέθοδο κατάλληλης ανάλυσης ανάλογα με το χρονική διάρκεια εκτέλεσης του τμήματος κώδικα που θέλετε να μετρήσετε. Για παράδειγμα, θα ήταν άστοχο να χρησιμοποιήσετε μια μέθοδο με ανάλυση 100msec για να μετρήσετε κώδικα που περιμένετε να έχει διάρκεια εκτέλεσης της τάξης των 120msec, διότι η ακρίβεια της μέτρησής σας θα είναι κακή. Για τους σκοπούς της άσκησής μας θα χρησιμοποιήσουμε τη συνάρτηση clock_gettime() η οποία έχει ανάλυση τουλάχιστον 1μsec.

Κατά τη διάρκεια των μετρήσεών σας είναι σημαντικό να βεβαιωθείτε ότι δεν υπάρχουν άλλοι

χρήστες που εκτελούν υπολογισμούς στο μηχάνημα (αν αυτό είναι κοινόχρηστο) και ότι δεν τρέχετε οι ίδιοι κάποια άλλη υπολογιστική διεργασία. Η εντολή top στο τερματικό σας δείχνει μια λίστα με τις ενεργές διεργασίες, ταξινομημένες με βάση το ποσοστό χρήσης του επεξεργαστή. Η λίστα αυτή ανανεώνεται περιοδικά. Μπορείτε να βγείτε από την εντολή top πατώντας το πλήκτρο q (quit).

Όπως θα διαπιστώσετε, κάθε φορά που τρέχετε το πρόγραμμά σας θα παίρνετε ελαφρά διαφορετικές μετρήσεις χρόνου. Σε ελάχιστες περιπτώσεις, μπορεί να έχετε και σημαντική απόκλιση από τις υπόλοιπες εκτελέσεις, συνήθως προς το χειρότερο, επειδή π.χ. το σύστημα συνέβη τη συγκεκριμένη στιγμή να είναι απασχολημένο με κάποια διαχειριστική λειτουργία. Για το λόγο αυτό είναι κρίσιμο να μην εκτελέσετε κάθε πείραμα μία μόνο φορά. Μια καλή πρακτική είναι να εκτελέσετε κάθε πείραμα 12 φορές, να πετάξετε το χαμηλότερο και τον υψηλότερο χρόνο και να αναφέρετε το μέσο όρο και την τυπική απόκλιση των υπολοίπων (ΠΡΟΣΟΧΗ: ΠΑΝΤΑ μαζί με το μέσο όρο θα πρέπει να αναφέρετε και την τυπική απόκλιση).

3.5 Χρήση Μεταγλωττιστή

Για την εργασία θα χρησιμοποιήσουμε το μεταγλωττιστή icx της Intel. Ο τρόπος κλήσης του μοιάζει πολύ με τον τρόπο κλήσης του gcc. Για να μεταγλωττίσετε π.χ. ένα πρόγραμμα από το αρχείο κώδικα test_prog.c και να παράξετε το εκτελέσιμο test_prog, ενώ ταυτόχρονα να δείτε όλα τα warnings, θα δώσετε:

```
icx -Wall test_prog.c -o test_prog
```

Αν θέλετε να προσθέσετε την απαραίτητη πληροφορία για να μπορείτε να χρησιμοποιήσετε debugger, χρησιμοποιείτε κατά τα γνωστά και το flag -g:

```
icx -Wall -g test_prog.c -o test_prog
```

Ένα πολύ σημαντικό flag είναι το -O, το οποίο ελέγχει το βαθμό βελτιστοποιήσεων που θα προσπαθήσει να κάνει αυτόματα ο μεταγλωττιστής. Με -O0 δηλώνουμε ότι δε θέλουμε να γίνουν καθόλου βελτιστοποιήσεις:

```
icx -Wall -O0 test_prog.c -o test_prog
```

Δίπλα από το -O μπορείτε να χρησιμοποιήσετε τον αριθμό 0, 1, 2 ή 3, για να διαλέξετε το βαθμό βελτιστοποιήσεων (-O3 είναι ο μέγιστος βαθμός βελτιστοποιήσεων). Δεν είναι απαραίτητο ότι πηγαίνοντας σε υψηλότερο επίπεδο βελτιστοποιήσεων θα επιταχύνετε τον κώδικα. Είναι πιθανό να συμβεί και το αντίθετο, καθώς δεν είναι απαραίτητα ευεργετικές όλες οι βελτιστοποιήσεις σε όλους τους κώδικες. Εναλλακτικά, μπορείτε να δώσετε το flag -fast για να ζητήσετε από τον μεταγλωττιστή να εφαρμόσει εκείνες τις βελτιστοποιήσεις που θεωρεί ότι θα βελτιώσουν την ταχύτητα του κώδικα:

```
icx -Wall -fast test_prog.c -o test_prog
```

4 Ζητούμενα

Θα πρέπει, ξεκινώντας από τον κώδικα που σας δίνεται να εφαρμόσετε διαδοχικά βελτιστοποιήσεις ώστε να επιταχύνετε την εκτέλεση του κώδικα. Μπορείτε να εφαρμόσετε τις βελτιστοποιήσεις με όποια σειρά κρίνετε πιο πρόσφορη, αν και η επίδοση που παρατηρείτε και ο τελικός κώδικας που θα προκύψει συχνά εξαρτάται από τη σειρά. Ο κώδικας που προκύπτει μετά από κάθε βελτιστοποίηση θα αποθηκεύεται σε αρχείο με διαφορετικό όνομα. Σημειώστε ότι κάθε νέα βελτιστοποίηση θα εφαρμόζεται πάνω στον κώδικα που προέκυψε από την προηγούμενη, με την προϋπόθεση βέβαια ότι η προηγούμενη οδήγησε σε βελτίωση (ή πάντως όχι σε αύξηση) του χρόνου εκτέλεσης. Θυμηθείτε ότι πιθανόν οι βελτιστοποιήσεις να κάνουν τον κώδικα πιο δυσανάγνωστο. Αυτό είναι φυσιολογικό, μη διστάσετε :-).

Για κάθε νέα έκδοση κώδικα που προκύπτει θα πρέπει να τον μεταγλωττίσετε και να επιβεβαιώσετε

την ορθότητά του. Κατόπιν, θα μετρήσετε την επίδοση εκτελώντας έναν αριθμό πειραμάτων σύμφωνα με όσα συζητήθηκαν στην παράγραφο 3.4.

Θα πρέπει να εκτελέσετε 2 ανεξάρτητες σειρές πειραμάτων: Μία στην οποία η μεταγλώττιση έχει γίνει χωρίς την εφαρμογή αυτόματων βελτιστοποιήσεων από την πλευρά του μεταγλωττιστή (flag -O0) κα μία στην οποία έχουν εφαρμοστεί αυτόματες βελτιστοποιήσεις με στόχο την ταχύτητα (flag -fast).

5 Παράδοση

Πρέπει να παραδώσετε:

- Τον κώδικα που προκύπτει από τη διαδοχική εφαρμογή των βελτιστοποιήσεων (είτε ως καταλόγους με τις διαφορετικές εκδόσεις του κώδικα, είτε με τη μορφή ενός καταλόγου που περιλαμβάνει ένα git repo με ένα commit (με αντίστοιχο σχόλιο) ανά βήμα βελτιστοποίησης). **Μη συμπεριλάβετε αρχεία εισόδου και εξόδου, ούτε και εκτελέσιμα.**
- Το makefile όπως χρειάστηκε να το μετατρέψετε για να μεταγλωττίζονται όλα τα αρχεία και να παράγονται όλα τα εκτελέσιμα.
- Αναφορά στην οποία:
  - Θα περιγράφετε τα χαρακτηριστικά του συστήματος που χρησιμοποιήσατε (επεξεργαστής, μνήμη, έκδοση λειτουργικού, έκδοση πυρήνα, μεταγλωττιστής και έκδοσή του)
  - Θα περιγράφετε συνοπτικά τα σημεία στα οποία εφαρμόσατε κάθε βελτιστοποίηση και θα αναλύετε τα πειραματικά αποτελέσματα (μέσες τιμές / τυπικές αποκλίσεις) και τις παρατηρήσεις σας. Μην παραλείψετε να αναφέρετε και τυχόν βελτιστοποιήσεις που δοκιμάσατε χωρίς καλά αποτελέσματα στο χρόνο εκτέλεσης. Εξυπακούεται ότι θα πρέπει να υπάρχουν τα σχετικά διαγράμματα για τα πειραματικά αποτελέσματα.
- Spreadsheet (excel ή openoffice/libreoffice) στο οποίο θα υπάρχουν τα αποτελέσματα από τα επιμέρους πειράματα (οργανωμένα σε πίνακες), τα αποτελέσματα της στατιστικής τους επεξεργασίας για κάθε πείραμα (μέση τιμή και τυπική απόκλιση) και τα σχετικά διαγράμματα.

Δημιουργήστε ένα αρχείο .tar.gz με τα παραπάνω περιεχόμενα και όνομα <όνομα1>_<AEM1>_<όνομα2>_<AEM2>_lab1.tar.gz. Ανεβάστε το εμπρόθεσμα στο e-class.

6 Παράρτημα: Εγκατάσταση Intel OneAPI

Για αυτή την εργασία θα χρειαστεί να εγκαταστήσετε στον υπολογιστή σας το OneAPI της Intel. Πιο συγκεκριμένα, θα χρειαστεί να εγκαταστήσετε το Base toolkit (είναι 0δωρεάν). Το OneAPI παρέχει ένα σύνολο από εργαλεία για την ανάπτυξη, αποσφαλμάτωση, παραλληλοποίηση και μελέτη επίδοσης εφαρμογών σε CPUs, GPUs και FPGAs.