Finite Impulse Response “FIR” SIMD Optimization using intrinsic

Compilers

7/1/2012

Papoutsis Georgios Msc I.S.H/w&S/w 193
papoutsis@ceid.upatras.gr

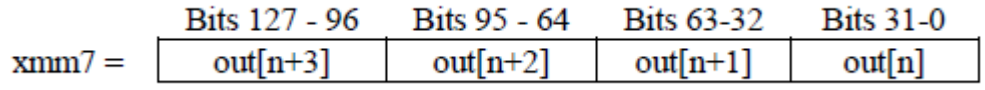
Σκοπός

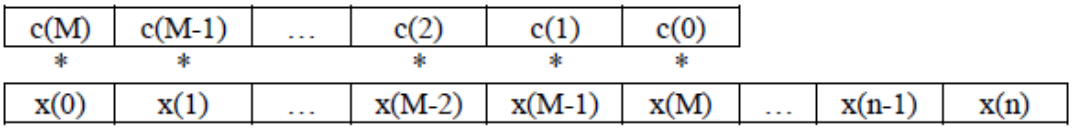
Αυτή η εργασία έχει ως σκοπό την βελτιστοποίηση του FIR φίλτρου με χρήση SIMD instruction set.Γενικά ένα FIR φίλτρο εκφράζετε από την εξίσωση: $y(n) = \sum_{k=0}^{M} bk\, x(n-k)$ με το x(n) να εκφράζει το διάνυσμα εισόδου μήκους n,το bk να εκφράζει το διάνυσμα συντελεστών (k-taps) μήκους k και το y(n) το διάνυσμα εξόδου μήκους n.

Ανάλυση Αλγορίθμου ως προς τις εξαρτήσεις

- Κάθε πολλαπλασιασμός είναι ανεξάρτητος του άλλου (προηγούμενου ή επόμενου).Επομένως υπάρχει παραλληλοποίηση.
- Όμως κάθε πολλαπλασιασμός σχετίζεται με μια πρόσθεση και για το λόγο αυτό θα εκτελούνται μαζί.

Για να εκμεταλλευτούμε τις δυνατότητες που μας παρέχει η xmm μπορούμε σε κάθε επανάληψη να διαχειριζόμαστε τα δεδομένα εισόδου (αφού προηγουμένως έχουν στοιχηθεί στην μνήμη) και συντελεστών ανα 4.Πιο αναλυτικά αυτό σημαίνει ότι όπως δείχνουμε στο παρακάτω σχήμα σε κάθε επανάληψη θα πρέπει να κρατάμε ένα συσσωρευτικό άθροισμα των γινομένων.

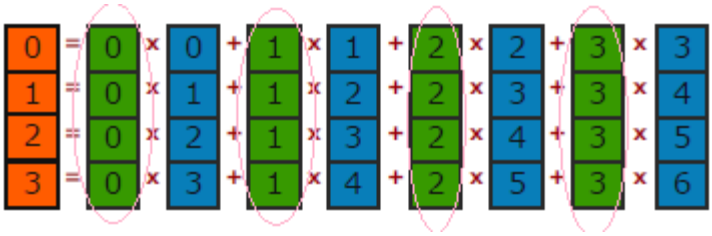




Έτσι το y(M) υπολογίζεται από τον τύπο:

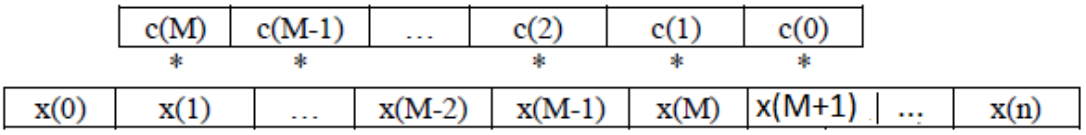
$$y(M) = x(M) * c(0) + x(M-1) * c(1) + \ldots + x(0) * c(M)$$

Παρατηρώντας ότι οι συντελεστές (taps) έχουν μεγάλη επαναχρησιμοποίηση και σε συνδυασμό με την εκμετάλλευση της xmm αρχιτεκτονικής τοποθετούμε σε κάθε επανάληψη στις τέσσερεις θέσεις του xmm register τον ίδιο συντελεστή. Στην παρακάτω εικόνα διακρίνουμε την κατά τέσσερα χρησιμοποίηση των συντελεστών του FIR φίλτρου.



Εδώ πρέπει να αναφέρουμε ότι παίρνουμε τους συντελεστές με αντίστροφη σειρά και υπολογίζουμε πρώτα την τελευταία έξοδο προκειμένου να επιτύχουμε την βέλτιστη επαναχρησιμοποίηση των συντελεστών σε κάθε επανάληψη.

Όμοια για να υπολογίσουμε το y(M+1)=x(M+1)*c(0)+…..x(1)*c(M).Δηλαδή ολισθαίνουμε τους συντελεστές κατά μια θέση ως προς το διάνυσμα εισόδου.



Μια άλλη παράμετρος που θα πρέπει να λάβουμε υπόψη είναι ότι: επειδή προσπελαύνουμε τους συντελεστές ανά τέσσερις προκειμένου τότε να είναι σωστή η αντιστοίχησή τους στον πολλαπλασιασμό με την είσοδο θα πρέπει να επαυξήσουμε το διάνυσμα κατάλληλα με κάποια μηδενικά. Για παράδειγμα εάν έχουμε 16 συντελεστές c0…c15 τότε για 4*8=32 bit fp θέλουμε 4 (οκτάμπιτα) μηδενικά.

Ο κώδικάς μας υλοποιείτε υπό τις παραδοχές:

1) Το FIR φίλτρο είναι πραγματικό
2) Το διάνυσμα εισόδου είναι στην κρυφή μνήμη

Ο κώδικας χωρίς την xmm βελτιστοποίηση είναι:

```
void main(){
	float*  input;
	int	length;
	float*  taps;
	int	num_taps;
	float*  output;

	float accum = 0.0;
	int index = 0;
	for (int n = 0; n < length; n++)
	{
		accum = 0.0;
		for (int m = 0; m < num_taps; m++)
		{
			index = n - m;
			if (index >= 0)
				accum += taps[m] * input[index];
		}
		output[n] = accum;
	}
}
```

Ο κώδικας με την xmm βελτιστοποίηση είναι:

```
for (i=0;i!=N;i+=4){
mmY =_mm_load_ps(Y + i );//kanoume load ta 4 stoixia eksodou

for (j=0;j!=M;j++){
    k=i+j;
mmX1 =_mm_load_ps(X + k);//kanoume load ta 4 stoixia eisodou
	mmTaps = _mm_load_ps(Taps + j);//kamoume load ta 4 taps
//prokimenou apoktisi se ka8e epanalipsi ta swsta x(i),x(i+1),x(i+2),x(i+3)
//mmX = _mm_alignr_epi8(mmX2,mmX1,j));
//antigrafume to idio tap
	mmTaps = _mm_shuffle_ps(mmTaps,mmTaps,_MM_SHUFFLE(j,j,j,j

	mmX = _mm_mul_ps(mmX,mmTaps);
	mmY = _mm_add_ps(mmY,mmX);
}


_mm_store_ps((float *)Y , mmY);
```

```
_mm_empty();

}
```