1 ΜΕΘΟΔΟΛΟΓΙΑ ΕΠΙΛΥΣΗΣ ΑΣΚΗΣΕΩΝ

1.1 Περιγραφή Ενσωματωμένων συστημάτων

Ως ενσωματωμένο σύστημα ορίζεται κάθε συσκευή η οποία εμπεριέχει ένα προγραμματιζόμενο επεξεργαστή, αλλά δεν είναι από μόνο του ένας γενικού σκοπού υπολογιστής. Ο προγραμματισμός των ενσωματωμένων συστημάτων μπορεί να πραγματοποιηθεί σε γλώσσα μηχανής (assembly) ή σε κάποια υψηλότερη γλώσσα προγραμματισμού αν διατίθεται ο compiler της αντίστοιχης γλώσσας για το συγκεκριμένο επεξεργαστή.

1.2 Σχεδιασμός Εφαρμογών

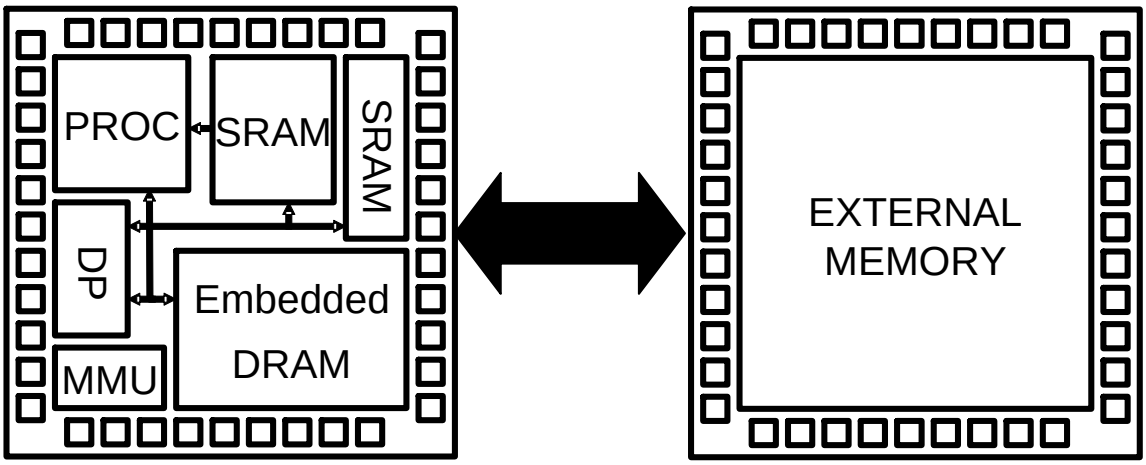
Οι εφαρμογές που σχεδιάζονται από τον προγραμματιστή δεν είναι πάντα καλογραμμένες εφαρμογές. Όταν ένας προγραμματιστής προσπαθεί να υλοποιήσει ένα αλγόριθμο, η σκέψη του επικεντρώνεται στην ανάλυση της εφαρμογής και στην επίλυση του προβλήματος και όχι στο τρόπο με τον οποίο θα τρέχει η εφαρμογή πάνω στο προγραμματιζόμενο επεξεργαστή. Το γεγονός αυτό έχει σαν συνέπεια τη μη-βέλτιστη υλοποίηση μιας εφαρμογής αφού δεν λαμβάνει υπόψη τα συγκεκριμένα χαρακτηριστικά ενός ενσωματωμένου συστήματος. Για να επιτευχθεί μια βέλτιστη υλοποίηση θα πρέπει να μειωθούν οι περιττές εκτελέσεις εντολών και οι άσκοπες προσπελάσεις στη μνήμη, οι οποίες έχουν σαν αποτέλεσμα την αργή εκτέλεση της εφαρμογής αλλά και την υψηλή κατανάλωση ενέργειας. Ο σχεδιασμός ενσωματωμένων συστημάτων απαιτεί ελαχιστοποίηση του χρόνου εκτέλεσης της εφαρμογής, για να πετύχουμε εκτέλεση πραγματικού χρόνου (real-time). Επίσης σε εφαρμογές που πρόκειται να χρησιμοποιηθούν σε φορητές συσκευές, πρέπει να περιοριστεί η κατανάλωση ενέργειας στο ελάχιστο ώστε να αυξηθεί η αυτονομία του συστήματος, καθώς και οι δύο παραπάνω παράγοντες πρέπει να λαμβάνονται σοβαρά υπόψη κατά το σχεδιασμό εφαρμογών με τη χρήση ενσωματωμένων συστημάτων.

1.3 Μεθοδολογία βελτιστοποίησης αλγορίθμων για χαμηλή κατανάλωση ενέργειας και υψηλή απόδοση

Επειδή ο προγραμματιστής κατά στην υλοποίηση αλγορίθμων σε μια γλώσσα υψηλού επιπέδου δεν λαμβάνει υπόψη του το υπολογιστικό σύστημα που θα εκτελέσει την εφαρμογή, δεν εκμεταλλεύεται τα χαρακτηριστικά του συστήματος που του προσφέρονται. Ο κύριος λόγος της αργής εκτέλεσης σε εφαρμογές πολυμέσων και δικτύων οφείλεται στο μεγάλο όγκο μεταφοράς δεδομένων από και προς τις μνήμες δεδομένων. Για την βελτιστοποίηση αλγορίθμων με βάση την αρχιτεκτονική υλοποίησης έχει αναπτυχθεί μια Μεθοδολογία Εξερεύνησης Μεταφορών και Αποθήκευσης Δεδομένων (DTSE, Data Transfer & Storage Exploration). Η μεθοδολογία αυτή βασίζεται σε αλγοριθμικούς μετασχηματισμούς, με στόχο την μείωση των προσπελάσεων στην εξωτερική μνήμη. Η μείωση αυτή επιτυγχάνεται αποθηκεύοντας σε μικρού μεγέθους μνήμης προσωρινά τμήματα των δεδομένων προς επεξεργασία. Οι μικρές προσωρινές μνήμες μπορούν να τοποθετηθούν πλησιέστερα στον υπολογιστικό πυρήνα (On-chip – cache μνήμες) που έχουν μικρότερο χρόνο προσπέλασης αλλά και χαμηλότερη κατανάλωση ενέργειας ανά προσπέλαση (Σχήμα 1).

Τα στάδια της βελτιστοποίησης μιας εφαρμογής είναι τρία. Το πρώτο στάδιο είναι ο εντοπισμός των σημείων της εφαρμογής όπου εμφανίζονται οι μεγαλύτερες καθυστερήσεις και εκείνων των σημείων όπου γίνονται οι περισσότερες προσπελάσεις στη μνήμη (πίνακες δεδομένων). Η ανάλυση αυτή ονομάζεται profiling και σε μικρού μεγέθους εφαρμογές ο εντοπισμός μπορεί να γίνει εύκολα, αντιθέτως σε μεγάλου μεγέθους αλγορίθμους η χρήση εργαλείων είναι απαραίτητη. Ένα τέτοιο εργαλείο που βοηθά το σχεδιαστή είναι το ATOMIUM (http://www.imec.be/design/atomium) το οποίο μετρά τον αριθμό των προσπελάσεων σε κάθε πίνακα δεδομένων της εφαρμογής και κατευθύνει το σχεδιαστή να επικεντρώσει την προσπάθεια και την βελτίωση αυτών των σημείων.

Στην συνέχεια ακολουθεί το στάδιο των μετασχηματισμών βρόχου (global loop) και τρίτο στάδιο των μετασχηματισμών επαναχρησιμοποίησης δεδομένων, όπως παρουσιάζονται αναλυτικά στις επόμενες παραγράφους.



Σχήμα 1. Ενσωματωμένο σύστημα με προσωρινές μνήμης αποθήκευση πάνω στο ολοκληρωμένο κύκλωμα του επεξεργαστή (SRAM, Embedded DRAM) για την προσωρινή αποθήκευση μικρού μεγέθους μεταβλητών, μειώνοντας τον αριθμό των προσπελάσεων στην εξωτερικό μνήμη.

1.3.1 Αλγοριθμικοί μετασχηματισμοί βελτιστοποίησης (Global Loop)

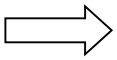
Οι κυριότεροι μετασχηματισμοί βρόχων (global loop) είναι οι loop unrolling, loop merging, loop tilling, loop bump, loop extend, loop body split, loop reverse, loop interchange. Παρακάτω παρουσιάζονται όλοι οι μετασχηματισμοί με παραδείγματα για να γίνουν ευκολότερα κατανοητοί.

Στόχος της εφαρμογής των μετασχηματισμών global loop είναι φέρουμε την μορφή της εφαρμογής σε κανονική δομή. Με τον όρο κανονική δομή εννοούμε τη μορφή του παρακάτω σχήματος. Δηλαδή στην μορφή συγχωνευμένων βρόχων, κάθε βρόχος θα έχει στο εσωτερικό του ένα βρόχο ο οποίος θα περιέχει ένα άλλο βρόχο εσωτερικά του και θα συνεχίζει όμοια.

```
for (i=0;i<N;i++)
{
  code into loop i
  for (j=0;j<M;j++)
  {
    code into loop j
    for (k=0;k<F;k++)
    {
      code into loop k
      for (l=0;l<G;l++)
      {
        code into loop l
        for (m=0;m<P;m++)
        code into loop m
      }
    }
  }
}
```

a) Loop unrolling: Μειώνει την πρόσθετη επιβάρυνση των βρόχων, και ενεργοποιεί άλλες βελτιστοποιήσεις (μετασχηματισμούς).

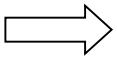
```
for
(i=0;i<4;i++)
 a[i]=b[i]*c[i];
```



```
for (i=0; i<2; i++)
{
   a[i*2]=b[i*2]*c[i*2];
   a[i*2+1]=b[i*2+1]*c[i*2+1];
}
```

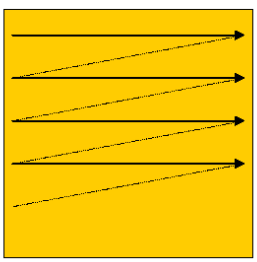
b) Loop merging: Μειώνει την πρόσθετη επιβάρυνση των βρόχων, και κανονικοποιεί τη δομή του αλγορίθμου. Παράλληλα μειώνει τις περιττές προσπελάσεις στη μνήμη δεδομένων.

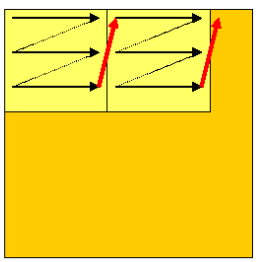
```
for
(i=0;i<N;i++)
 a[i]=b[i]*c[i];
for
(j=0;j<N;j++)
 d[j]=c[j]*e[j];
```

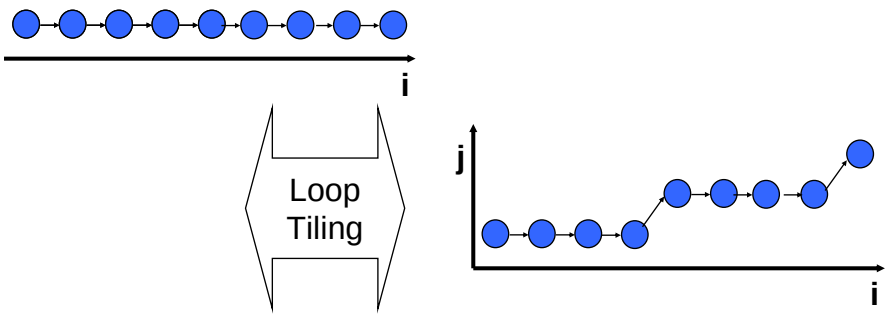


```
for (i=0;i<N;i++)
{
  a[i]=b[i]*c[i];
  d[i]=c[i]*e[i];
}
```

c) Loop tilling: Χωρίζει ένα βρόχο σε δυο ή περισσότερους συγχωνευμένους βρόχους, αλλάζει την σειρά των προσπελάσεων σε κάθε πίνακα και αλλάζει τη συμπεριφορά της cache μνήμης.








```
for(i=0;i<9;i++)
  A[i] = ...;
```

```
for(j=0; j<3; j++)
  for(i=4*j; i<4*j+4;i++)
    if (i<9)
      A[i] = ...;
```

d) Loop Bump: Ενεργοποιεί το μετασχηματισμό Loop Merging

```
for (i=2; i<N; i++)
   B[i] = f(A[i]);
for (i=0; i<N-2; i++)
   C[i] = g(B[i+2]);
```

i+2 > i ⇒ υπάρχουν εξαρτήσεις

```
for (i=2; i<N; i++)
   B[i] = f(A[i]);
for (i=2; i<N; i++)
   C[i-2] = g(B[i+2-2]);
```

i+2-2 = i ⇒ merging possible

Loop Merge

```
for (i=2; i<N; i++)
  B[i] = f(A[i]);
  C[i-2] = g(B[i]);
```

e) Loop Extend: Ενεργοποιεί το μετασχηματισμό Loop Merging

```
for (i=0; i<N; i++)
   B[i] = f(A[i]);
for (i=2; i<N+2; i++)
   C[i-2] = g(B[i]);
for (i=0; i<N+2; i++)
   if(i<N)
     B[i] = f(A[i]);
for (i=0; i<N+2; i++)
   if(i>=2)
      C[i-2] = g(B[i);
```

Loop Extend

Loop Merge

```
for (i=0; i<N+2; i++)
   if(i<N)
     B[i] = f(A[i]);
   if(i>=2)
      C[i-2] = g(B[i);
```

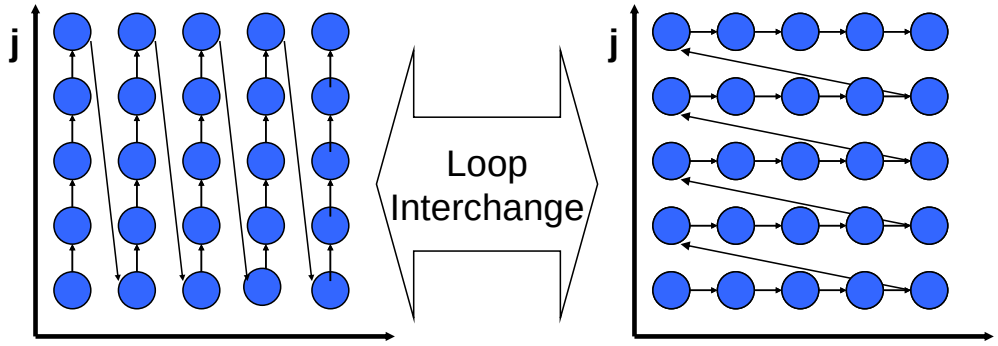
f) Loop Body Split: Ενεργοποιεί άλλους μετασχηματισμούς

```
for (i=0; i<N; i++)
 A[i] = f(A[i-1]);
 B[i] = g(in[i]);
for (j=0; j<N; j++)
 C[i] = h(B[i],A[N]);
```

```
for (i=0; i<N; i++)
 A[i] = f(A[i-1]);
for (k=0; k<N; k++)
 B[k] = g(in[k]);
for (j=0; j<N; j++)
 C[j] = h(B[j],A[N]);
```

```
for (i=0; i<N; i++)
 A[i] = f(A[i-1]);
for (j=0; j<N; j++)
 B[j] = g(in[j]);
 C[j] = h(B[j],A[N]);
```

g) Loop Interchange: Βασικός Μετασχηματισμός

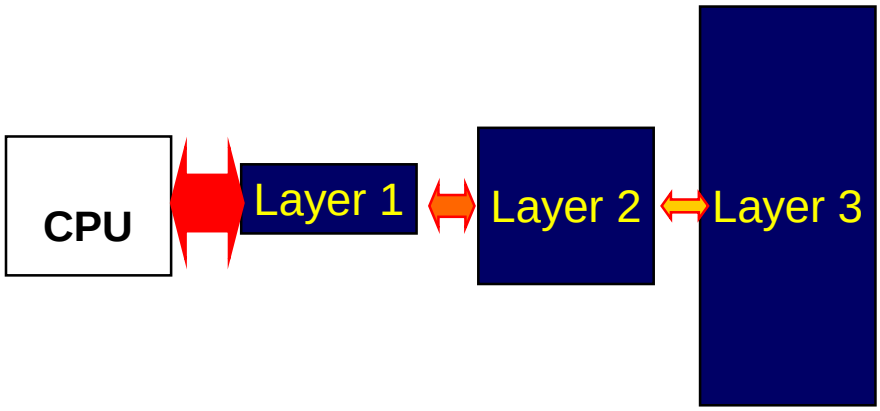



```
for(i=0; i<W; i++)
  for(j=0; j<H; j++)
    A[i][j] = …;
```

```
for(j=0; j<H; j++)
  for(i=0; i<W; i++)
    A[i][j] = …;
```

1.3.2 Μετασχηματισμοί Επαναχρησιμοποίησης δεδομένων

Η δεύτερη κατηγορία μετασχηματισμών που πρέπει να εφαρμοστούν στο αλγόριθμο είναι οι μετασχηματισμοί επαναχρησιμοποίησης δεδομένων (Data Reuse Transformations). Ο βασικός στόχος των μετασχηματισμών επαναχρησιμοποίησης δεδομένων είναι να μειώσουν τις περιττές προσπελάσεις στη μνήμη δεδομένων εισάγοντας μικρότερου μεγέθους μνήμες (για την προσωρινή αποθήκευση δεδομένων) που μπορούν να τοποθετηθούν πλησιέστερα στον επεξεργαστή. Σε αυτά τα επίπεδα θα τοποθετηθούν οι μεταβλητές οι οποίες εμφανίζουν το μεγαλύτερο αριθμό επαναχρησιμοποίησης και άρα μειώνονται οι προσπελάσεις στην εξωτερική μνήμη. Έτσι, οι μεταφορές των δεδομένων από τη μνήμη προς τον επεξεργαστή και αντίστροφα θα πραγματοποιούνται μέσω των επιπέδων που βρίσκονται κοντά στον επεξεργαστή και οι οποίες είναι πιο γρήγορες αλλά και το κόστος σε ενέργεια ανά προσπέλαση είναι μικρότερο. Έτσι μειώνεται ο χρόνος εκτέλεσης, παράλληλα όμως θα μειώνεται και η κατανάλωση της ενέργειας, αφού μειώνονται οι προσπελάσεις στην εξωτερική μνήμη. Στο σχήμα που ακολουθεί παρουσιάζεται μια αρχιτεκτονική με 3 επίπεδα μνήμης.



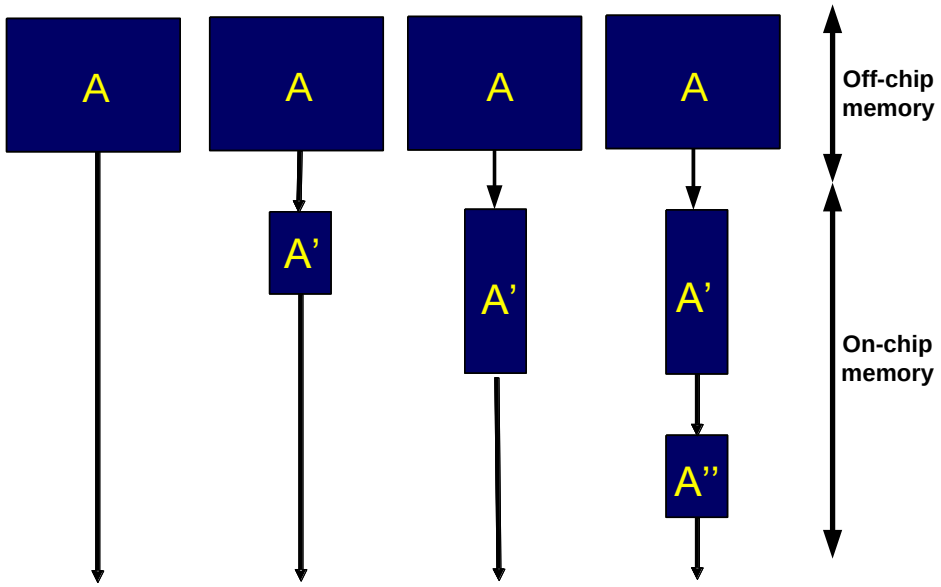
Σχήμα 2. Ιεραρχία μνήμης με τρία επίπεδα.

Τα βήματα που πρέπει να ακολουθήσουμε για την εφαρμογή των μετασχηματισμών επαναχρησιμοποίησης δεδομένων είναι τα ακόλουθα:

Βήμα 1: Αναγνώριση των πινάκων δεδομένων στους οποίους μπορεί να γίνει επαναχρησιμοποίηση δεδομένων.

Σχήμα 3. Time frame είναι η περίοδος κατά την οποία τμήμα δεδομένων από ένα μεγάλο πίνακα μπορούν να αντιγραφούν προσωρινά σε ένα μικρότερο ώστε να χρησιμοποιηθούν από το μικρό πίνακα έναντι του μεγάλου.

Βήμα 2: Καθορισμός των δυνατών συνδυασμών ιεραρχίας μνήμης



Σχήμα 4. Έχοντας στην διάθεση τρία μπλοκ μνήμης (A, A΄, A΄΄), δημιουργούνται τέσσερις ιεραρχίες μνήμης δεδομένων

2 DESIGN SPACE EXPLORATION

Η εξερεύνηση χώρου σχεδιασμού (Design Space Exploration – DSE) παίζει κεντρικό ρόλο στον μοντέρνο σχεδιασμό ενσωματωμένων συστημάτων. Πιο συγκεκριμένα, κατά τον σχεδιασμό υπάρχουν πολλές διαφορετικές επιλογές χαρακτηριστικών που ικανοποιούν τις προδιαγραφές του συστήματος. Ο ρόλος του DSE είναι να αναδειχθεί ένας συστηματικός τρόπος ώστε οι διαφορετικές διαμορφώσεις (configurations) του συστήματος, να μπορούν να βρεθούν και να αξιολογηθούν ώστε να επιλεχθεί μια τελική διαμόρφωση με βάση μια σειρά από κριτήρια. Πολύ σημαντική παράμετρος του προβλήματος είναι ότι χαρακτηριστικά του συστήματος μπορεί να είναι αντικρουόμενα και η επιλογή τους να δημιουργεί ένα trade-off. Για παράδειγμα, η υψηλή απόδοση ενός συστήματος είναι ένας συνηθισμένος σχεδιαστικός στόχος ο οποίος κατά κανόνα έρχεται σε αντίθεση με την σχεδιαστικό στόχο της χαμηλής κατανάλωσης ισχύος του συστήματος υπό σχεδιασμό. Η παρακάτω εικόνα συνοψίζει τα προαναφερθέντα trade-offs στον σχεδιασμό συστημάτων που χρησιμοποιούνται ευρέως σήμερα.

Σχήμα 5. Συνδυασμός σχεδιαστικών trade-offs για την παραγωγή ενός τελικού συστήματος

Ένα πιο συγκεκριμένο DSE πρόβλημα είναι το ακόλουθο. Κατά το σχεδιασμό ενός συστήματος σε ψηφίδα (System on Chip – SoC), πρέπει να ληφθεί απόφαση για το υποσύστημα μνήμης cache που θα συμπεριληφθεί στο σύστημα. Υποθέτουμε ότι το SoC θα χρησιμοποιήσει για την εκτέλεση εφαρμογών οι οποίες μπορούν να προσομοιωθούν επαρκώς από μια σουίτα από benchmarks η οποία και θα χρησιμοποιηθεί για την αξιολόγηση των διαφορετικών cache configurations. Στο παράδειγμα αυτό, περιοριζόμαστε μόνο σε L1 data cache και οι σχεδιαστικές μας επιλογές περιλαμβάνουν

το μέγεθος της cache, το associativity level , το block size και την στρατηγική αντικατάστασης εσωτερικά της cache. Ο σχεδιαστικός στόχος είναι να βρεθεί ένα configuration το οποίο:

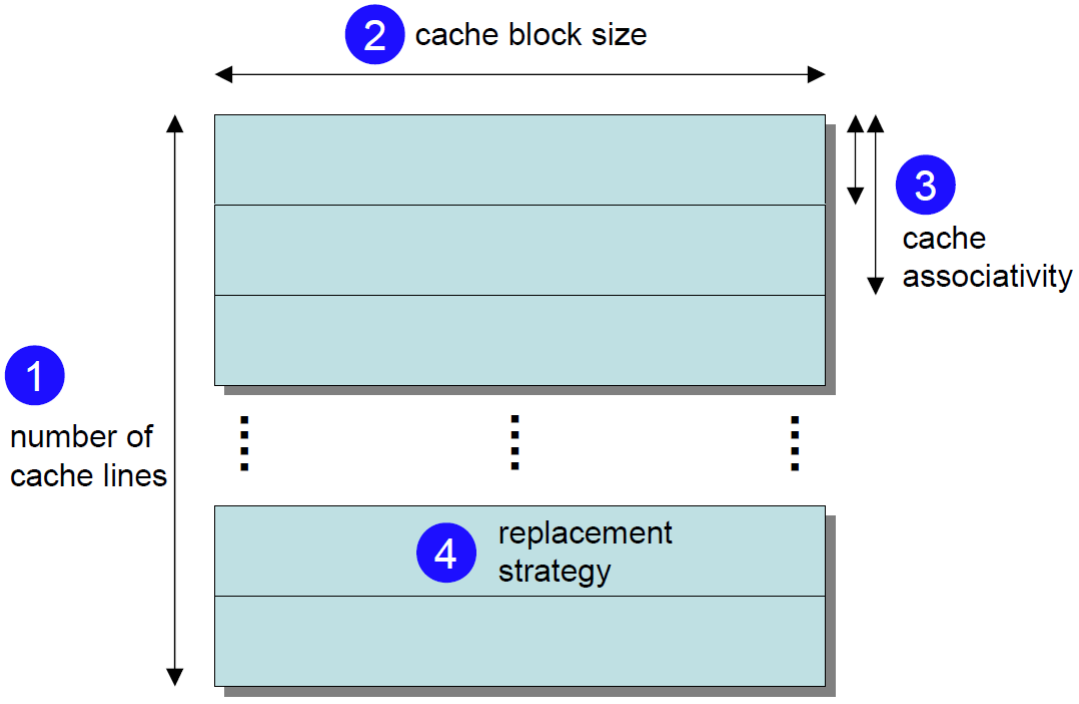
1. Μεγιστοποιεί την υπολογιστική απόδοση του SoC κατά την εκτέλεση της προαναφερθείσας σουίτας εφαρμογών. Η μετρική που χρησιμοποιείται για την ποσοτικοποίηση της απόδοσης είναι οι κύκλοι ανά εντολή (Cycles per instruction – CPI) του επεξεργαστή.
2. Ελαχιστοποιεί την επιφάνεια (area) που θα καταλάβει η μνήμη πάνω στο chip.

Ο παρακάτω πίνακας ομαδοποιεί τις σχεδιαστικές παραμέτρους και το εύρος των διαθέσιμων τιμών.

Αριθμός	Παράμετρος	Εύρος τιμών
1	Αριθμός γραμμών cache	2^k, k = 6…14
2	Μέγεθος Block	2^k Bytes, k = 3…7
3	Associativity	2^k, k = 0…5
4	Στρατηγική αντικατάστασης	LRU or FIFO

Πίνακας 1. Παράμετροι για τον προσδιορισμό της αρχιτεκτονικής της μνήμης cache

Η εικόνα 6 απεικονίζει τις σχεδιαστικές επιλογές:

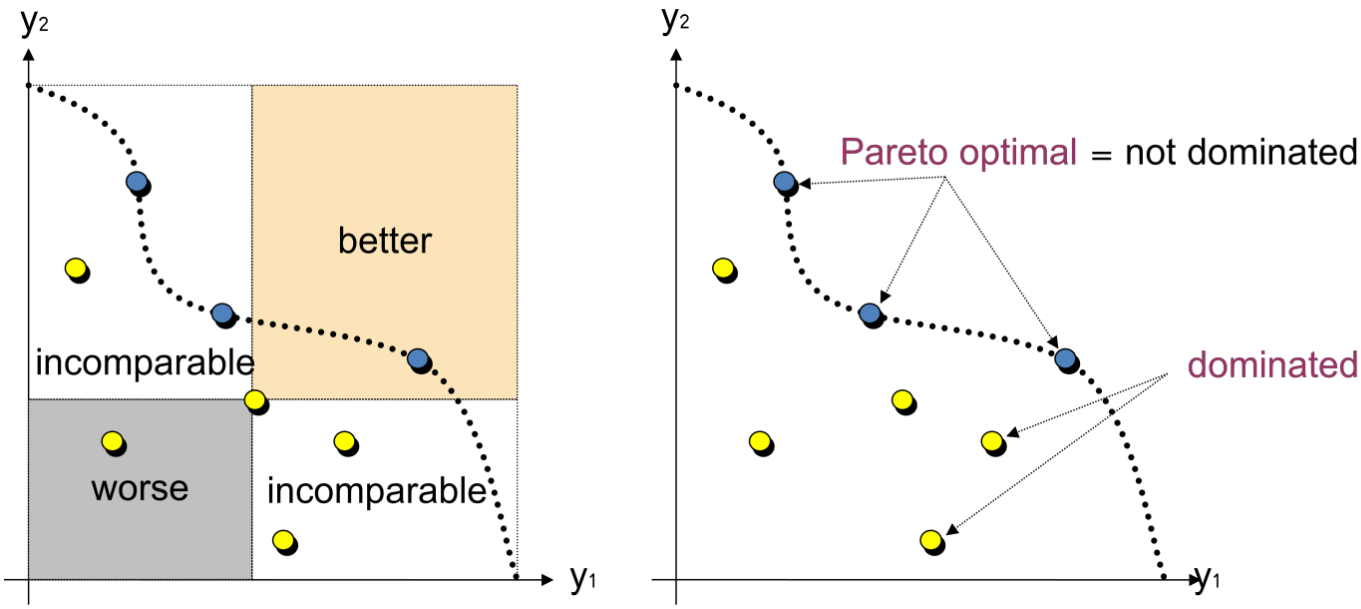


Σχήμα 6. Απεικόνιση των διαθέσιμων σχεδιαστικών επιλογών για την αρχιτεκτονική της L1 μνήμης cache.

Παρά την απλή φύση του σχεδιαστικού προβλήματος, οι δύο σχεδιαστικοί στόχοι (μεγιστοποίηση του CPI – ελαχιστοποίηση του area) συνθέτουν ένα performance vs area trade-off. Το trade-off αυτό, δεν μπορεί να επιλυθεί βρίσκοντας μια μοναδική βέλτιστη λύση αλλά υπόκειται σε μια σειρά από βέλτιστες κατά Pareto λύσεις (Pareto-optimal solutions). Για να οριστεί η βέλτιστη κατά Pareto λύση πρέπει πρώτα να οριστεί η έννοια της κυριαρχίας ενός σημείου στον χώρο σχεδιασμού. Συγκεκριμένα, ένα σημείο K_i του χώρου σχεδιασμού κυριαρχείται από ένα άλλο σημείο K_j αν

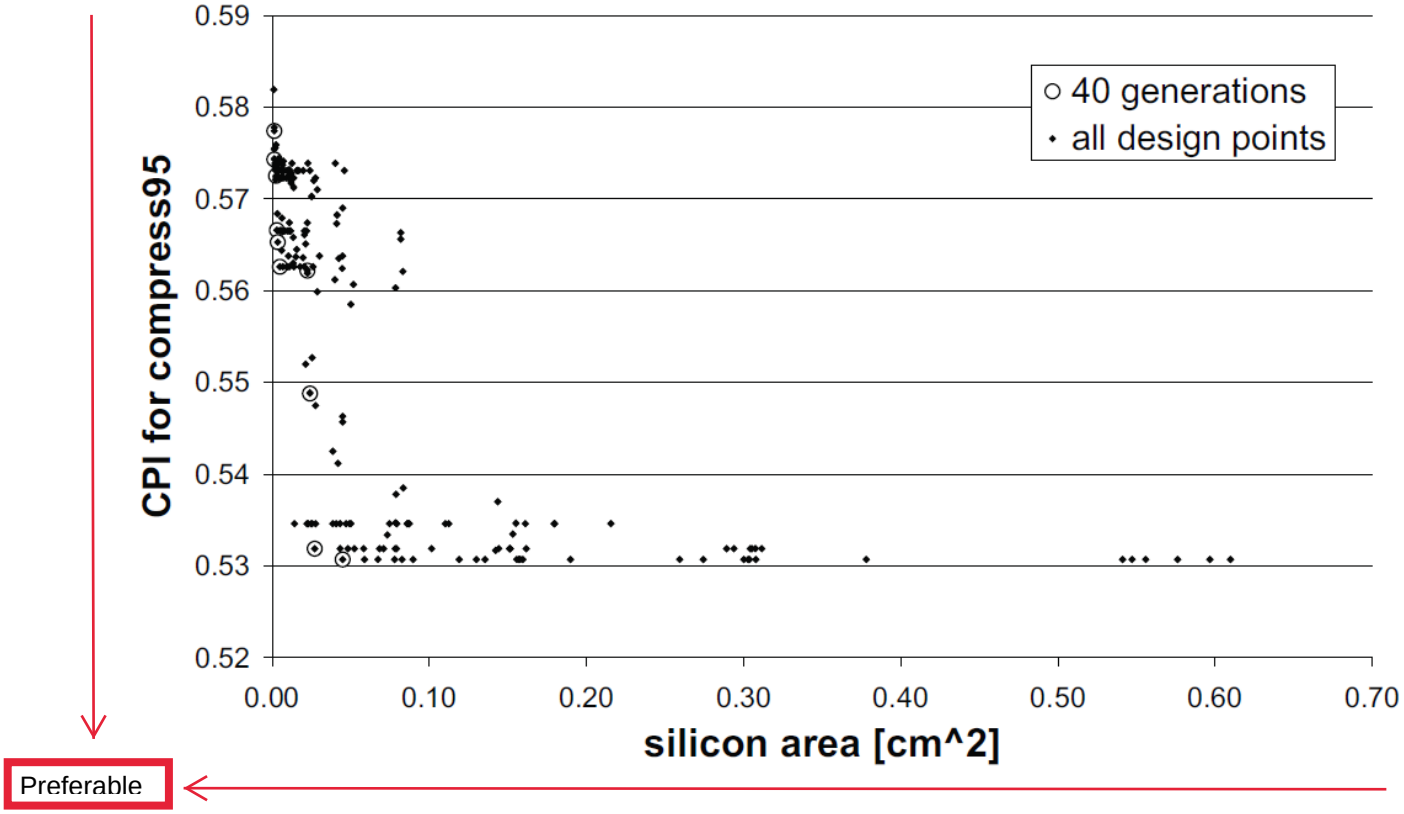
i. Το K_j είναι καλύτερο ή ίσο από το K_i για όλα τα σχεδιαστικά κριτήρια και
ii. Το K_j είναι καλύτερο από το K_i για τουλάχιστον ένα σχεδιαστικό κριτήριο

Ένα σημείο του χώρου σχεδιασμού είναι βέλτιστο κατά Pareto αν δεν υπάρχει άλλο σημείου του χώρου το οποίο να κυριαρχεί επί αυτού. Η εικόνα 7 αναπαριστά γραφικά τις ανωτέρω σχέσεις για ένα σχεδιαστικό χώρο με κριτήρια y_1 και y_2.



Σχήμα 7. Κυριαρχία κατά Pareto και Pareto βέλτιστα σημεία σε ένα σχεδιαστικό χώρο με κριτήρια y_1 και y_2.

Ο σχεδιαστικός χώρος για το παράδειγμα του σχεδιασμού της μνήμης cache όπως προέκυψε από ένα εργαλείο για DSE, φαίνεται στην παρακάτω εικόνα. Ο σχεδιαστής πλέον μπορεί να επιλέξει την διαμόρφωση που πληροί καλύτερα τις προδιαγραφές του υπό σχεδιασμού συστήματος.



Σχήμα 8. Γραφική αναπαράσταση και των 540 πιθανών σχεδιαστικών σημείων όπως προσδιορίστηκαν από εξαντλητική αναζήτηση. Στο διάγραμμα φαίνονται ακόμα τα σχεδιαστικά σημεία που προέκυψαν από των αλγόριθμο αναζήτησης SPEA2.

Σε πιο σύνθετα σχεδιαστικά προβλήματα DSE όπου πολλοί και διαφορετικοί στόχοι πρέπει να επιτευχθούν και ο χώρος περιορίζεται από βέλτιστες κατά Pareto λύσεις, η εξερεύνηση των διαθέσιμων σχεδιαστικών επιλογών αποκτά συστηματικό χαρακτήρα και υπάρχουν μια σειρά από επιλογές για την προσέγγιση της εξερεύνησης:

i. **Εξαντλητική αναζήτηση (Exhaustive Search)** του χώρου σχεδιασμού. Η πρακτική αυτή εξασφαλίζει το βέλτιστο αποτέλεσμα όμως είναι μη αποδοτική και κατά πάσα πιθανότητα ανέφικτη να εφαρμοστεί σε πολύπλοκα συστήματα όπου ο αριθμός των διαθέσιμων επιλογών είναι απαγορευτικά μεγάλος.
ii. **Αναγωγή του προβλήματος σε ένα μόνο σχεδιαστικό στόχο (Reduction to a single objective)**. Για DSE με πολλαπλά αντικρουόμενα κριτήρια, υπάρχουν αρκετές προσεγγίσεις διαθέσιμες για την αναγωγή του προβλήματος σε ένα σετ από προβλήματα με μοναδικό κριτήριο βελτιστοποίησης. Έπειτα το πρόβλημα μπορεί να αντιμετωπιστεί με κλασσικούς τρόπους βελτιστοποίησης όπως simulated annealing ή ακέραιο γραμμικό προγραμματισμό.
iii. **Τυχαιοποιημένη αναζήτηση μαύρου κουτιού (Black-Box Randomized Search).** Ο χώρος σχεδιασμού δειγματοληπτείται και εξερευνάται κάνοντας χρήση βελτιστοποίησης μαύρου κουτιού που σημαίνει ότι νέα σχεδιαστικά σημεία δημιουργούνται με βάση τις πληροφορίες που έχουν συλλεχθεί μέχρι τώρα και κάνοντας χρήση μιας ορισμένης συνάρτησης για την παραγωγή

επόμενου σημείου προς έλεγχο. Τα νέα σημεία που δημιουργούνται ελέγχονται εκ νέου δίνοντας παρέχοντας νέες πληροφορίες για τον χώρο σχεδιασμού. Στην βιβλιογραφία υπάρχουν εργασίες για Pareto Simulated Annealing, Pareto Tabu Search, evolutionary multi-objective optimization και άλλα.

Η ίδια η φύση του προβλήματος σχεδιασμού μπορεί να βοηθήσει στον περιορισμό του δέντρου των επιλογών του χώρου σχεδιασμού. Για παράδειγμα, μπορεί κάποιες παράμετροι να μην σχετίζονται μεταξύ τους ή η αναζήτηση να μπορεί να περιοριστεί σε ένα υποσύνολο του χώρου όπου υπάρχουν με μεγάλη πιθανότητα ελκυστικές λύσεις. Εφόσον ο χώρος σχεδιασμού περιοριστεί, εφαρμόζονται οι προαναφερθείσες τεχνικές ανάλυσης για την παραγωγή του τελικού επιθυμητού configuration.

Παράδειγμα γενετικού αλγορίθμου: NSGA-II

Ο αλγόριθμος **Non-dominated Sorting Genetic Algorithm II (NSGA-II)** αναπτύχθηκε από τον καθηγητή Deb και τους συνεργάτες του στο Kanpur Genetic Algorithms Laboratory. Ο NSGA-II είναι ένας γρήγορος και ελιτιστικός multi-objective evolutionary αλγόριθμος, ο οποίος έχει την ίδια δομή με όλους τους γενετικούς αλγορίθμους που βασίζονται σε πληθυσμούς.

Ο βασικός μηχανισμός μπορεί να συνοψιστεί ως εξής: ξεκινώντας από έναν πληθυσμό - γονέα, κάποια άτομα επιλέγονται προκειμένου να παράξουν έναν πληθυσμό - παιδί. Ο αλγόριθμος εφαρμόζει τελεστές που δουλεύουν πάνω στις μεταβλητές εισόδου των επιλεγμένων ατόμων, προσπαθώντας να βελτιώσουν τις εξόδους τους: η μεταλλαγή και η διασταύρωση είναι κλασικές επιλογές για τον NSGA-II. Αυτή η διαδικασία επαναλαμβάνεται για τον ζητούμενο αριθμό από παραγωγές - generations.Ο αλγόριθμος έχει κάποια εξαιρετικά και δυνατά χαρακτηριστικά:

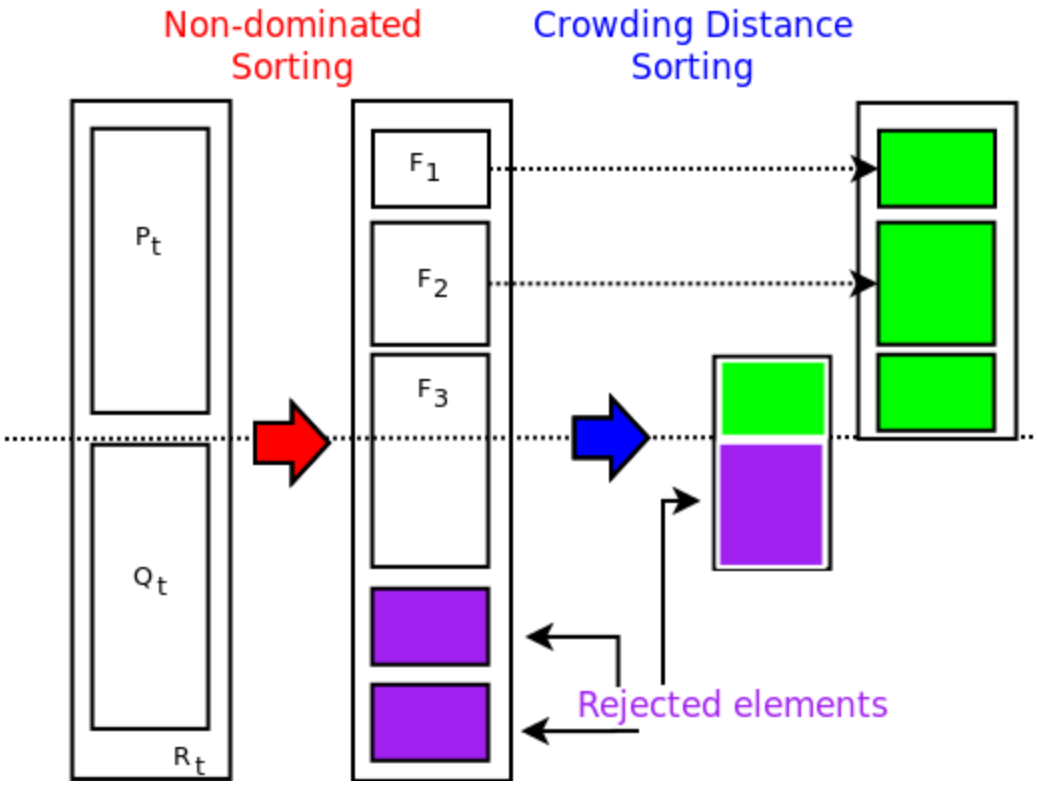
- Περιλαμβάνει μία γρήγορη \non-dominated} διαδικασία ταξινόμησης. Το να ταξινομήσεις τα άτομα από ένα δοσμένο πληθυσμό με βάση το επίπεδο της μή-κυριαρχίας (non-domination) είναι μία περίπλοκη διαδικασία, γεγονός το οποίο κάνει τους γενικούς non dominated αλγορίθμους ταξινόμησης αρκετά ακριβούς για μεγαλύτερη μεγέθη πληθυσμών. Η λύση που υιοθετήθηκε για τον NSGA-II εφαρμόζει μία έξυπνη στατηγική ταξινόμησης.
- Η multi-objective αναζήτηση περιλαμβάνει ελιτισμό. Ο NSGA-II υλοποιεί την multi-objective αναζήτηση χρησιμοποιώντας μία προσέγγιση, η οποία πρακτικά περιλαμβάνει την αποθήκευση όλων των non-dominated λύσεων που έχουν ανακαλυφθεί ως τώρα, αρχής γενομένης από τον αρχικό πληθυσμό. Ο ελιτισμός ενισχύει τις ιδιότητες σύγκλισης προς το πραγματικά Pareto-optimal σύνολο.
- Υιοθετείται ένας μηχανισμός που συντηρεί τις διαφορετικότητα των παραμέτρων. Η διαφορετικότητα καθώς και η διασπορά των λύσεων είναι εγγυημένες χωρίς τη χρήση από επιπλέον παραμέτρους. Ο NSGA-II υιοθετεί ένα κατάλληλο ξεχώρισμα παραμέτρων, το οποίο καλείται πληθυσμιακή απόσταση (crowding distance), το οποίο εκτιμά την

πυκνότητα των λύσεων πάνω στον αντικειμενικό χώρο, καθώς και έναν τελεστή για σύγκριση πληθυσμών ο οποίος καθορίζει την επιλογή της διαδικασίας που θα ακολουθηθεί προς ένα ομοιόμορφα κατανεμημένο μέτωπο Pareto.

- Η μέθοδος χειρισμού των περιορισμών δεν χρησιμοποιεί παραμέτρους ποινής. Ο αλγόριθμος υλοποιεί έναν τροποποιημένο ορισμό της κυριαρχίας προκειμένου να λύσει τα multi-objective προβλήματα, κάτω από τους περιορισμούς που έχουν οριστεί, αποδοτικά: η συνήθης κυριαρχία είναι το κριτήρια για να ταξινομήσεις feasible σημεία. Ένα feasible σημείο είναι πάντα προτιμότερο από ένα unfeasible. Τα unfeasible στοιχεία ταξινομούνται με βάση το άθροισμα των περιορισμών που παραβιάζουν.

Η διαδικασία του NSGA-II περιγράφεται γραφικά στο Σχήμα 9. Τα άτομα του πληθυσμού-γονέα P_t μεγέθους N και ο νέος πληθυσμός Q_t του ίδιου μεγέθους (που δημιουργήθηκε εφαρμόζοντας διάφορες εναλλαγές των τελεστών μετάλλαξης και διασταύρωσης πάνω σε άτομα του P_t) ομαδοποιούνται μαζί. Ο συνδυασμένος πληθυσμός R_t κατόπιν, ταξινομείται με βάση το επίπεδο της μη-κυριαρχίας. Έτσι, καταλήγουμε με σύνολα από *non-dominated* λύσεις, του τύπου $(F_1, F_2, ...)$. Ο νέος πληθυσμός P_{t+1} δημιουργείται διαλέγοντας τα καλύτερα *non-dominated* σύνολα, τα οποία μπορούν να εισαχθούν ολόκληρα μέσα στο νέο πληθυσμό (με ένα συνδυασμένο μέγεθος μικρότερο ή ίσο του N) συν τα μέλη από το τελευταίο σύνολο που επιλέγεται (το οποίο δεν μπορεί να εξυπηρετηθεί πλήρως), χρησιμοποιώντας τον crowded comparison τελεστή.

Ο NSGA-II επιτρέπει τόσο συνεχείς (real-coded) όσο και διακριτές (binary-coded) μεταβλητές σχεδίασης. Ειδικοί τελεστής μετάλλαξης και διασταύρωσης εφαρμόζονται σε κάθε είδος μεταβλητών. Χειριζόμαστε τις κατηγορηματικές (μή-ταξινομημένες διακριτές) μεταβλητές σαν απλές διακριτές μεταβλητές. Αυτό το ελάττωμα, ωστόσο, αναπληρώνεται αυξάνοντας τις δυνατότητες εξερεύνησης του αλγορίθμου επιτρέποντας μία μεγαλύτερη μετάλλαξη. Ο NSGA-II ελέγχει μία μεγαλύτερη γκάμα από υποψήφιες λύσεις και οι ρουτίνες επιλογής και ελιτισμού του οδηγούν σταδιακά στο σχηματισμό του μετώπου Pareto.



Σχήμα 9. NSGA-II sorting procedure

3 ΑΣΚΗΣΗ

3.1 Περιγραφή της υπό εξέταση αλγορίθμου

Σε αυτή την παράγραφο θα αναλυθεί ο αλγόριθμος Parallel Hierarchical One Dimensional Search (PHODS), ένας αλγόριθμος που ανήκει στην περιοχή των πολυμέσων. Ο αλγόριθμος PHODS είναι ένας αλγόριθμος εκτίμησης κίνησης (Motion Estimation), ο οποίος έχει στόχο να ανιχνεύσει τη κίνηση των αντικειμένων μεταξύ δύο διαδοχικών εικόνων (frame) του βίντεο. Οι αλγόριθμοι ανίχνευσης της κίνησης είναι καθοριστικοί για την συμπίεση βίντεο και αποτελούν τον πυρήνα κάθε εφαρμογής που περιέχει βίντεο. Ο PHODS έχει σαν είσοδο δύο διαδοχικές εικόνες από μια ακολουθία εικόνων βίντεο (διαστάσεων M×N), χωρίζει τις εικόνες σε block (διαστάσεων B×B pixel) σε κάθε μια από τις εικόνες αυτές και προσπαθεί να βρει την μετατόπιση του κάθε block από τη μια εικόνα (frame) στην επόμενη (frame). Για την εύρεση της μετατόπισης κάθε μπλόκ από το ένα frame στο επόμενο frame εκτελείται σύγκριση του μπλόκ από το πρώτο frame με όλα τα μπλόκ που βρίσκονται στην γύρω περιοχή από το επόμενο frame. Βάσει ενός συγκεκριμένου κριτηρίου επιλέγεται το μπλοκ εκείνο που εμφανίζει την μικρότερη τιμή στο κριτήριο.

Ο αρχικός αλγόριθμος PHODS σε γλώσσα C παρουσιάζεται παρακάτω.

```
        /* Parallel Hierarchical One-Dimensional Search motion estimation - Initial algorithm */
        /* Used for simulation and profiling */

        #include <stdio.h>
        #include <math.h>
        #include <string.h>
        #include <stdlib.h>
        #include <time.h>
        #define N 144     /* frame dimension for QCIF format */
        #define M 176     /* frame dimension for QCIF format */
        #define B 16      /* Block size */
        #define p 7       /* Search space. Restricted in a [-p,p] region around the original
location of the block. */

        void read_sequence(unsigned char current[N][M],unsigned char previous[N][M])
        {
         FILE *picture0,*picture1;
         int i,j;

         if((picture0=fopen("akiyo0.y","rb"))==NULL)
         {
          printf("previous frame doesn't exist\n");
          exit(-1);
         }

         if((picture1=fopen("akiyo1.y","rb"))==NULL)
         {
          printf("current frame doesn't exist\n");
          exit(-1);
         }

        /* Input for the previous frame */
          for(i=0;i<N;i++)
           for(j=0;j<M;j++)
            previous[i][j]=fgetc(picture0);

        /* Input for the current frame */
          for(i=0;i<N;i++)
           for(j=0;j<M;j++)
            current[i][j]=fgetc(picture1);

         fclose(picture0);
         fclose(picture1);
        }


        void phods_motion_estimation(int current[N][M],int previous[N][M],int
vectors_x[N/B][M/B],int vectors_y[N/B][M/B])
        {
         int x,y,i,j,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;
          for(i=0;i<N/B;i++)
           for(j=0;j<M/B;j++)
           {
            vectors_x[i][j]=0;
            vectors_y[i][j]=0;
           }

         for(x=0;x<N/B;x++)       /* For all blocks in the current frame */
          for(y=0;y<M/B;y++)
          {
           S=4;
           while(S>0)
           {
            min1=255*B*B;
            min2=255*B*B;
            for(i=-S;i<S+1;i+=S)     /* For all candidate blocks in X dimension */
            {
             distx=0;
             for(k=0;k<B;k++)     /* For all pixels in the block */
              for(l=0;l<B;l++)
              {
```

```
               p1=current[B*x+k][B*y+l];

                   if((B*x+vectors_x[x][y]+i+k)<0 || (B*x+vectors_x[x][y]+i+k)>(N-1) ||
(B*y+vectors_y[x][y]+l)<0 || (B*y+vectors_y[x][y]+l)>(M-1))
                     p2=0;
                   else
                    p2=previous[B*x+vectors_x[x][y]+i+k][B*y+vectors_y[x][y]+l];

                   distx+=abs(p1-p2);
                  }
                  if(distx<min1)
                  {
                   min1=distx;
                   bestx=i;
                  }
                 }

                 for(i=-S;i<S+1;i+=S)     /* For all candidate blocks in X dimension */
                 {
                  disty=0;
                  for(k=0;k<B;k++)     /* For all pixels in the block */
                   for(l=0;l<B;l++)
                   {
                    p1=current[B*x+k][B*y+l];

                   if((B*x+vectors_x[x][y]+k)<0 || (B*x+vectors_x[x][y]+k)>(N-1) ||
(B*y+vectors_y[x][y]+i+l)<0 || (B*y+vectors_y[x][y]+i+l)>(M-1))
                    q2=0;
                   else
                    q2=previous[B*x+vectors_x[x][y]+k][B*y+vectors_y[x][y]+i+l];

                   disty+=abs(p1-q2);
                  }

                  if(disty<min2)
                  {
                   min2=disty;
                   besty=i;
                  }
                 }

                 S=S/2;
                 vectors_x[x][y]+=bestx;
                 vectors_y[x][y]+=besty;
                }
               }
              }


              int main()
              {
               unsigned char current[N][M],previous[N][M];
               int motion_vectors_x[N/B][M/B],motion_vectors_y[N/B][M/B];

               read_sequence(current,previous);
               phods_motion_estimation(current,previous,motion_vectors_x,motion_vectors_y);
              }
```

3.2 Ζητούμενα

Στο **προσομοιωτή συστημάτων qemu**, για αρχιτεκτονική ARM επεξεργαστή να υλοποιηθούν τα παρακάτω.

1. Μετασχηματίστε τον δοθέντα κώδικα ώστε να μετράται ο χρόνος που χρειάζεται η συνάρτηση phods_motion_estimation για να εκτελεστεί. Η

μέτρηση του χρόνου να γίνει με την συνάρτηση gettimeofday με ακρίβεια μικροδευτερολέπτων. Παρουσιάστε τα αποτελέσματα της μέτρησης.

2. Με δεδομένη την ύπαρξη της υποδομής για την μέτρηση του χρόνου που χρειάζεται το κρίσιμο κομμάτι της εφαρμογής για να εκτελεστεί, εφαρμόστε μετασχηματισμούς στον κώδικα ώστε να μειωθεί ο χρόνος εκτέλεσης.

3. Στον κώδικα που προέκυψε από το ερώτημα 2, εφαρμόστε Design Space Exploration αναφορικά με την εύρεση του βέλτιστου μεγέθους μπλοκ B (μεταβλητή στον κώδικα) για την συγκεκριμένη ARM αρχιτεκτονική. Η αναζήτηση να πραγματοποιηθεί με εξαντλητική αναζήτηση και μόνο για τιμές του B που είναι κοινοί διαιρέτες του M και του N.

4. Στον κώδικα που προέκυψε από το ερώτημα 2, εφαρμόστε Design Space Exploration αναφορικά με την εύρεση του βέλτιστου μεγέθους μπλοκ θεωρώντας ορθογώνιο μπλοκ διαστάσεων B_x και B_y για την συγκεκριμένη ARM αρχιτεκτονική. Το μέγεθος του B_x να είναι διαιρέτης του N και το μέγεθος B_y να είναι διαιρέτης του M. Εφαρμόστε εξαντλητική αναζήτηση και προτείνετε κάποιο ευριστικό τρόπο περιορισμού της αναζήτησης αν θεωρείτε ότι υπάρχει.

Υποσημείωση 1: Όλα τα προγράμματα να γίνουν compile με flag –O0 ώστε να μην γίνονται optimizations από τον gcc στον κώδικα σας.

Υποσημείωση 2: Στα ερωτήματα 3 και 4 η εκτέλεση των διαφορετικών περιπτώσεων πρέπει να γίνεται με χρήση κάποιου script που θα καλεί το τελικό πρόγραμμα για τα διάφορα configurations. Το script θεωρείται μέρος της άσκησης και θα παραδοθεί και αυτό με τον τελικό κώδικα. Μη αυτόματη εκτέλεση των προγραμμάτων θεωρείται μη αποδοτική και δεν θα πάρει τον πλήρη βαθμό.