Παράρτημα Α΄

Ασκήσεις σε Σχεδιασμό Ενσωματωμένων Συστημάτων

Σχεδιασμός Ενσωματωμένων Συστημάτων: Σύντομη θεωρία και υποδειγματικά λυμένη εφαρμογή πολυμέσων

Α΄.1 Εισαγωγή

Το Παράρτημα αυτό γράφτηκε με τέτοιο τρόπο, ώστε να μπορεί να αποτελέσει ένα αυτόνομο κεφάλαιο με στόχο τους διπλωματούχους μηχανικούς που θέλουν να αποκτήσουν μια σύντομη, αλλά και εμπεριστατωμένη γνώση στο αντικείμενο της διαχείρισης μνήμης πολυμεσικών εφαρμογών.

Α΄.2 ΜΕΘΟΔΟΛΟΓΙΑ ΕΠΙΛΥΣΗΣ ΑΣΚΗΣΕΩΝ

Α΄.2.1 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.

Α΄.2.2 Σχεδιασμός Εφαρμογών

Οι εφαρμογές που σχεδιάζονται από τον προγραμματιστή δεν είναι πάντα καλογραμμένες εφαρμογές. Όταν ένας προγραμματιστής προσπαθεί να υλο-

ποιήσει έναν αλγόριθμο, η σκέψη του επικεντρώνεται κατά κύριο λόγο στην ανάλυση της εφαρμογής και στην επίλυση του προβλήματος (δηλ. στη σωστή *λογική λειτουργία* της εφαρμογής) και όχι στον τρόπο με τον οποίο θα τρέχει (ή θα υλοποιηθεί) η εφαρμογή πάνω στον προγραμματιζόμενο επεξεργαστή. Το γεγονός αυτό έχει ως 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, ώστε να πετύχουμε π.χ., εκτέλεση σε πραγματικό χρόνο (real-time) για εφαρμογές βίντεο. Επίσης, σε εφαρμογές που πρόκειται να χρησιμοποιηθούν σε φορητές συσκευές, η κατανάλωση ισχύος πρέπει να περιοριστεί στο ελάχιστο, ώστε να αυξηθεί η αυτονομία του συστήματος, καθώς και οι δύο παραπάνω παράγοντες πρέπει να λαμβάνονται σοβαρά υπόψη κατά τον σχεδιασμό εφαρμογών με τη χρήση ενσωματωμένων συστημάτων.

Α΄.2.3 Μεθοδολογία βελτιστοποίησης αλγορίθμων για χαμηλή κατανάλωση ενέργειας και υψηλή απόδοση

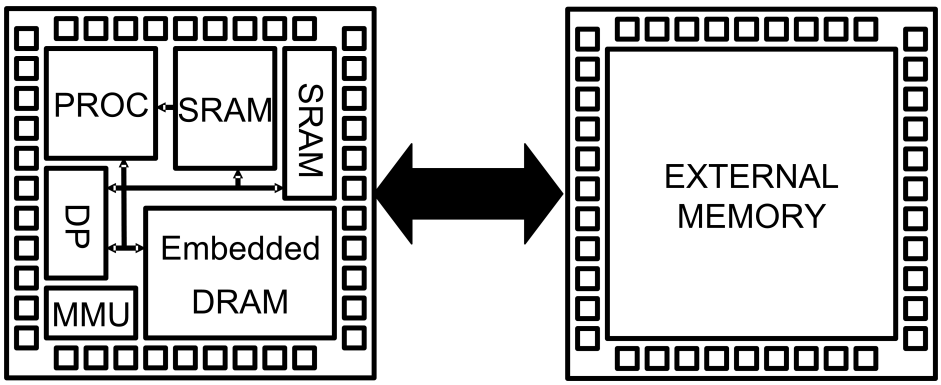
Επειδή ο προγραμματιστής κατά στην υλοποίηση αλγορίθμων σε μια γλώσσα υψηλού επιπέδου δεν λαμβάνει υπόψη του το υπολογιστικό σύστημα που θα εκτελέσει την εφαρμογή, δεν εκμεταλλεύεται τα χαρακτηριστικά του συστήματος που του προσφέρονται. Ο κύριος λόγος της αργής εκτέλεσης σε εφαρμογές πολυμέσων και δικτύων οφείλεται στον μεγάλο όγκο μεταφοράς δεδομένων από και προς τις μνήμες δεδομένων (ή κύρια μνήμη). Για τη βελτιστοποίηση αλγορίθμων με βάση την αρχιτεκτονική υλοποίησης έχει αναπτυχθεί μια Μεθοδολογία Εξερεύνησης Μεταφορών και Αποθήκευσης Δεδομένων (DTSE, Data Transfer & Storage Exploration[1]). Η μεθοδολογία αυτή βασίζεται σε αλγοριθμικούς μετασχηματισμούς, με στόχο τη 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 μεγα-

[1]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5613&rep=rep1&type=pdf

λύτερες καθυστερήσεις και εκείνων των σημείων, όπου γίνονται οι περισσότερες προσπελάσεις στη μνήμη (πίνακες δεδομένων). Η ανάλυση αυτή ονομάζεται *σκιαγράφηση* (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 τη βελτίωση αυτών των σημείων.

Στην συνέχεια ακολουθεί το στάδιο των μετασχηματισμών βρόχου (global loop) και το τρίτο στάδιο των μετασχηματισμών επαναχρησιμοποίησης δεδομένων (data-reuse transformations), όπως παρουσιάζονται συνοπτικά στις επόμενες παραγράφους.



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

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, και loop interchange. Παρακάτω παρουσιάζονται όλοι οι μετασχηματισμοί με παραδείγματα για να γίνουν ευκολότερα κατανοητοί.

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

```
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 άλλους 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.

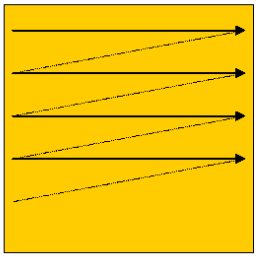
```
for (i=0;i<N;i++)
        a[i]=b[i]*c[i];
for (j=0;j<N;j++)
        d[j]=c[j]*e[j];
```

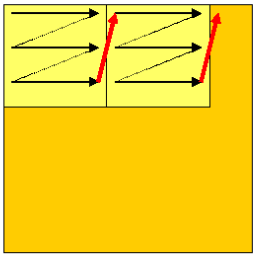
```
for (i=0;i<N;i++)
{
  a[i]=b[i]*c[i];
  d[i]=c[i]*e[i];
}
```

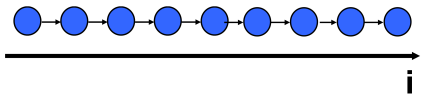
c) Loop tilling: Χωρίζει ένα βρόχο σε δυο ή περισσότερους συγχωνευμένους βρόχους, αλλάζει την σειρά των προσπελάσεων σε κάθε πίνακα και αλλάζει τη συμπεριφορά της cache μνήμης.

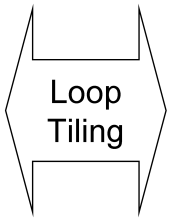
```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        c[i] = a[i,j]*b[i];
```

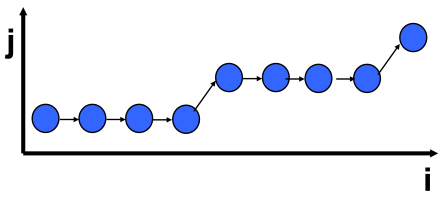
```
for (i=0; i<N; i+=2)
  for (j=0; j<N; j+=2)
    for (ii=0; ii<min(i+2,N); ii++)
      for (jj=0; jj<min(j+2,N); jj++)
        c[ii] = a[ii,jj]*b[ii];
```












```
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 Reverse: Απαλείφονται οι εξαρτήσεις

Υπάρχουν εξαρτήσεις μέσα στο βρόγχο.

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

N−i > i ⇒ merging not possible
(δεν επιτρέπεται η συγχώνευση των βρόχων)

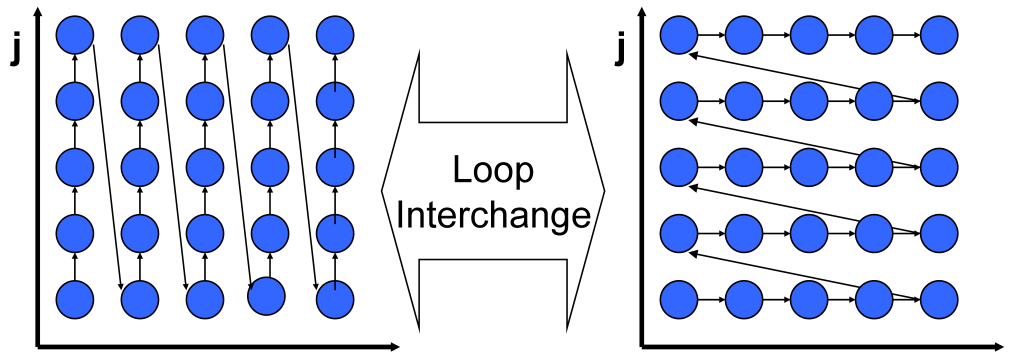
Loop Reverse

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

Loop Merge

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

h) Loop Interchange: Βασικός Μετασχηματισμός




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

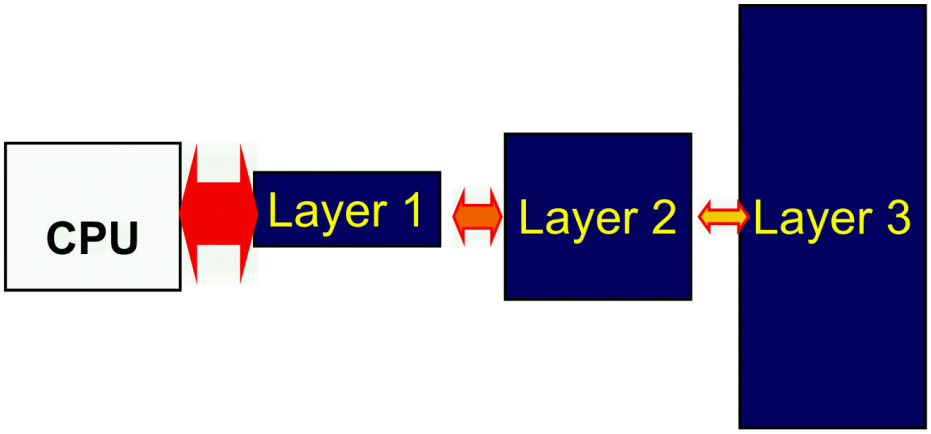
```
for(j=0; j<H; j++)
  for(i=0; i<W; i++)
    A[i][j] = ...;
```

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.

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 Α΄.2: Ιεραρχία μνήμης με τρία επίπεδα.

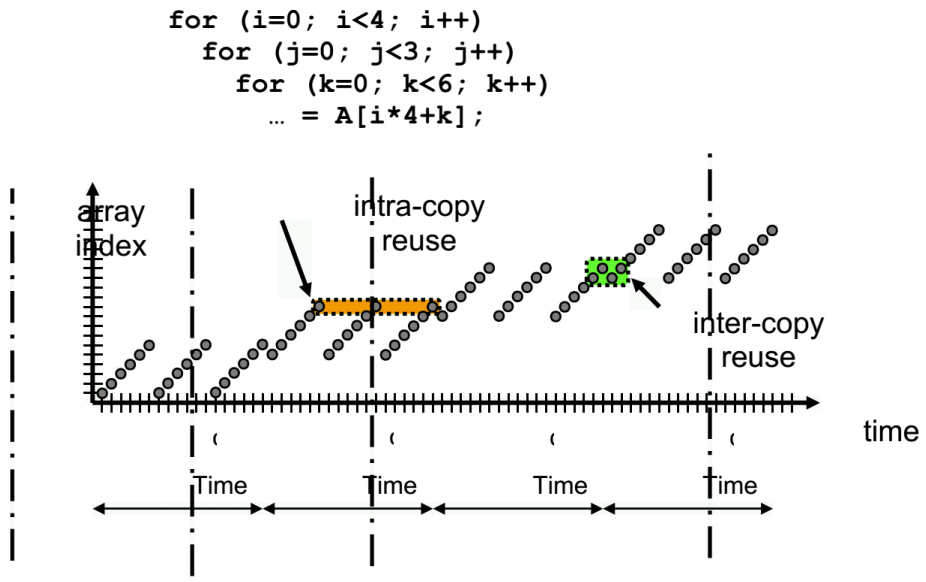
Βήμα 2: Καθορισμός των δυνατών συνδυασμών ιεραρχίας μνήμης

Α΄.3 ΠΑΡΑΔΕΙΓΜΑ

Σε αυτή την παράγραφο θα περιγράψουμε αναλυτικά τα στάδια για το σχεδιασμό μιας εφαρμογής πολυμέσων. Στη συνέχεια, στον αρχικό κώδικα που θα δοθεί, θα εφαρμοστούν μερικοί από τους μετασχηματισμούς με στόχο τη βελτιστοποίηση της εφαρμογής ως προς την ταχύτητα εκτέλεσης αλλά και την κατανάλωση ισχύος.

Α΄.3.1 Περιγραφή της άσκησης

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



Σχήμα Α΄.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.

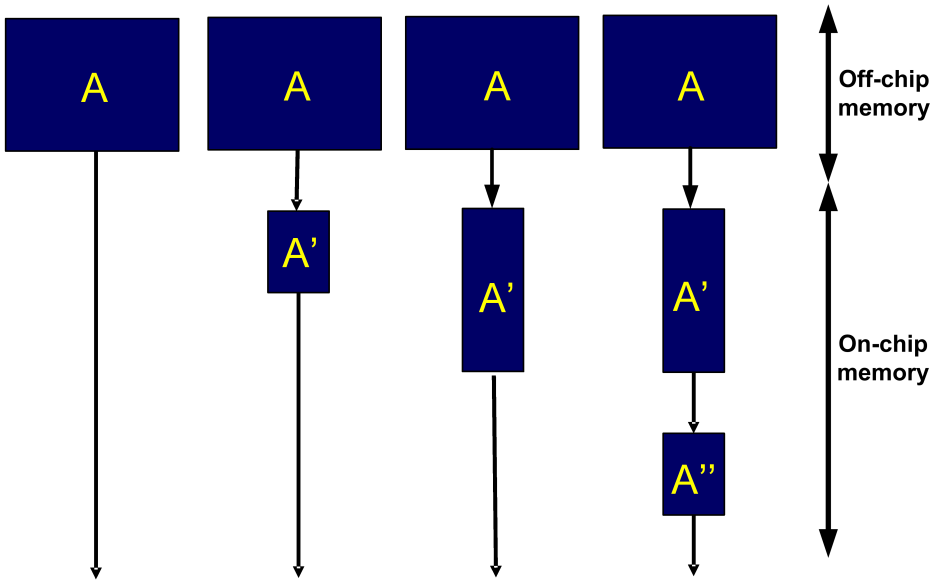
frame. Βάσει ενός συγκεκριμένου κριτηρίου επιλέγεται το μπλοκ εκείνο που εμφανίζει την μικρότερη τιμή στο κριτήριο.

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])
```



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

```
{
 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);
}
```

Α΄.3.2 Μέθοδος Επίλυσης

Εφαρμογή των μετασχηματισμών Global Loop

Η μεθοδολογία βελτιστοποίησης θα εφαρμοστεί σταδιακά στο κύριο τμήμα της εφαρμογής. Εδώ πρέπει να αναφερθεί ότι δεν θα γίνουν βελτιστοποιήσεις στις συναρτήσεις εισόδου. Το τμήμα του κώδικα που αποτελεί αντικείμενο μελέτης και εφαρμογής των μετασχηματισμών είναι η συνάρτηση «phods_motion_estimation».

Ο μετασχηματισμός Global Loop στους βρόχους με δείκτη i Στο πρώτο στάδιο θα εφαρμοστεί ο μετασχηματισμός loop merging για τους δύο βρόχους for(i=-S;i<S+1;i+=S). Επίσης στο διπλό βρόχο αρχικοποίησης των μεταβλητών vectors_x και vectors_y αλλάζουμε τις μεταβλητές των βρόχων από i, j σε x, y, ώστε να μπορεί να γίνει μετασχηματισμός συγχώνευσης με τους βρόχους που ακολουθούν. Ο μετασχηματισμός αυτός για να εφαρμοστεί θα πρέπει να μην παραβιάζονται οι εξαρτήσεις δεδομένων. Εξάρτηση δεδομένων έχουμε όταν κάποιες τιμές ενός πίνακα που δημιουργούνται και χρησιμοποιούνται στη συνέχεια του αλγορίθμου. Παραβίαση θα έχουμε όταν χρησιμοποιήσουμε μια τιμή ενός πίνακα πριν αυτή δημιουργηθεί. Στο συγκεκριμένο μετασχηματισμό δεν υπάρχει κάποια εξάρτηση, οπότε μπορούμε να τον εφαρμόσουμε. Ο αλγόριθμος μετά την εφαρμογή του μετασχηματισμού είναι ο ακόλουθος:

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

     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;
    }
   }
  }
```

Ο μετασχηματισμός Global Loop στους βρόχους με δείκτη k Στο επόμενο βήμα, με ένα δεύτερο μετασχηματισμό loop merging συγχωνεύουμε τις εντολές που περιέχουν οι βρόχοι με δείκτες k. Έτσι οι εντολές των δύο ακολουθιακών βρόχων με δείκτη k ενσωματώνονται σε ένα βρόχο που εμπεριέχει τις εντολές των δύο αρχικών.

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

Ο μετασχηματισμός Global Loop στους βρόχους με δείκτη l Παρατηρώντας τον κώδικα με λεπτομέρεια στα σημεία με έντονη γραμματοσειρά βλέπουμε ότι υπάρχουν δυο βρόχοι με δείκτη l και μπορούν να συγχωνευθούν σε ένα βρόχο. Έτσι μεταφέρονται οι εντολές του δευτέρου βρόχου στο εσωτερικό του πρώτου, με συνέπεια ο αλγόριθμος να παίρνει την παρακάτω μορφή. Επίσης βλέπουμε ότι η εντολή ανάγνωσης p1=current[B*x+k][B*y+l]; εμφανίζεται δύο φορές μέσα στις εντολές του νέου βρόχου (μία φορά σε κάθε ένα από του αρχικούς) οπότε αυτόματα διαγράφεται η επανάληψη της εντολής.

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

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

        S=S/2;
        vectors_x[x][y]+=bestx;
```

```
      vectors_y[x][y]+=besty;
     }
    }
   }
```

Ο αλγόριθμος μετά την εφαρμογή των μετασχηματισμών Global Loop Μια πιο συμπτυγμένη έκδοση του αλγορίθμου παρουσιάζεται παρακάτω.

```
   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,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;

    for(x=0;x<N/B;x++)       /* For all blocks in the current frame */
     for(y=0;y<M/B;y++)
     {
      vectors_x[x][y]=0;
      vectors_y[x][y]=0;
      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;
      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]+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];

              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];

        distx+=abs(p1-p2);
        disty+=abs(p1-q2);
       }

       if(distx<min1)
       {
        min1=distx;
        bestx=i;

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

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

Αποκοπή των περιττών εκτελέσεων Στον παραπάνω αλγόριθμο παρατηρούμε ότι για i=0 το p2 είναι ίσο με το q2 οπότε δεν είναι απαραίτητο να το υπολογίσουμε ξανά. Για να μειώσουμε τις περιττές προσπελάσεις στη μνήμη αλλά και τις περιττές εκτελέσεις εντολών εισάγουμε μια εντολή ελέγχου if(i==0) disty=distx; Οπότε αυτόματα παρακάμπτει τον υπολογισμό του disty όταν το i==0. Έτσι ο αλγόριθμος μετασχηματίζεται ως εξής:

```
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,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;

 for(x=0;x<N/B;x++)       /* For all blocks in the current frame */
  for(y=0;y<M/B;y++)
  {
```

```
vectors_x[x][y]=0;
vectors_y[x][y]=0;
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;
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]+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(i==0)
   disty=distx;
   else
   {
        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(distx<min1)
  {
   min1=distx;
   bestx=i;
```

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

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

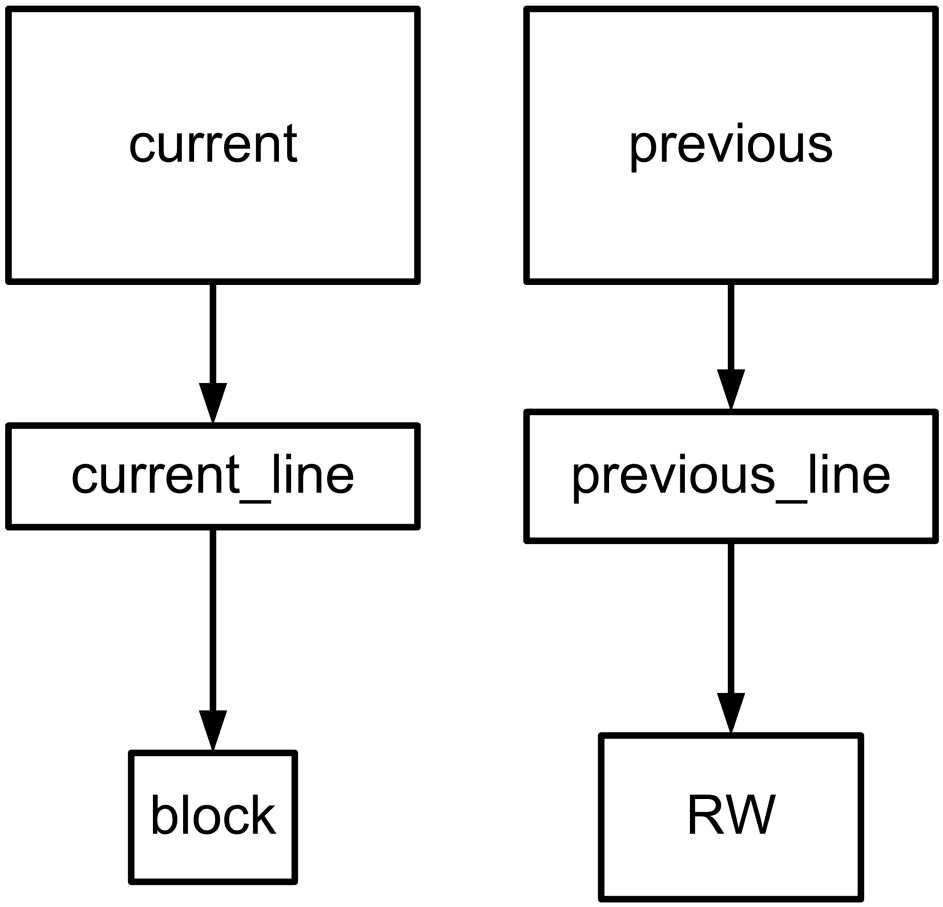
Εφαρμογή των μετασχηματισμών επαναχρησιμοποίησης δεδομένων

Μετά την εφαρμογή των μετασχηματισμών βρόχου ο αλγόριθμος έχει πάρει την κανονικοποιημένη μορφή των συγχωνευμένων βρόχων. Το επόμενο στάδιο της μεθοδολογίας είναι η εφαρμογή των μετασχηματισμών επαναχρησιμοποίησης δεδομένων. Αυτό που θέλουμε να πετύχουμε είναι να μειωθούν οι προσπελάσεις στους πίνακες δεδομένων current και previous οι οποίοι έχουν μεγάλο μέγεθος. Η μείωση θα γίνει με την εισαγωγή μικρότερου μεγέθους πινάκων δεδομένων για την προσωρινή αποθήκευση τμημάτων των αρχικών πινάκων. Οι νέοι πίνακες θα εισαχθούν στα σημεία του αλγορίθμου μετά από κάθε βρόχο και θα αποθηκεύουν τμήμα των δεδομένων ενός από τους αρχικούς πίνακες δεδομένων, τα οποία θα χρησιμοποιηθούν στο αντίστοιχο βρόχο.

Σε κάθε εκτέλεση του βρόχου με δείκτη x, τα δεδομένα που χρησιμοποιούνται από τους δύο πίνακες δεδομένων (current και previous) μπορούν να αποθηκευθούν σε δύο buffers με διαστάσεις M×B για τον current πίνακα και τον ονομάζουμε current_line. Ενώ για τον previous πίνακα εισάγουμε έναν buffer με διαστάσεις M×(B+2*p) (όπου p=S+S/2+S/4) και τον ονομάζουμε previous_line.

Εσωτερικά που βρόχου με δείκτη y, τα δεδομένα που χρησιμοποιούνται από τους αρχικούς πίνακες δεδομένων μπορούν να αποθηκευτούν προσωρινά σε δυο buffers διαστάσεων B×B και (B+2*p×(B+2*p). Αντίστοιχα, μελετώντας τους εσωτερικότερους βρόχους δημιουργούμε μικρότερου μεγέθους buffers για την προσωρινή αποθήκευση τμημάτων δεδομένων από τους αρχικούς πίνακες. Τέλος, το μικρότερο μέγεθος buffer που μπορεί να χρησιμοποιηθεί για την προσωρινή αποθήκευση του πίνακα current είναι το block με διαστάσεις B×B. Με την ολοκλήρωση της διερεύνησης πιθανών πινάκων για την προσωρινή αποθήκευση δεδομένων σχηματίζουμε οι αλυσίδες περιγραφής των πιθανών δομών μνήμης

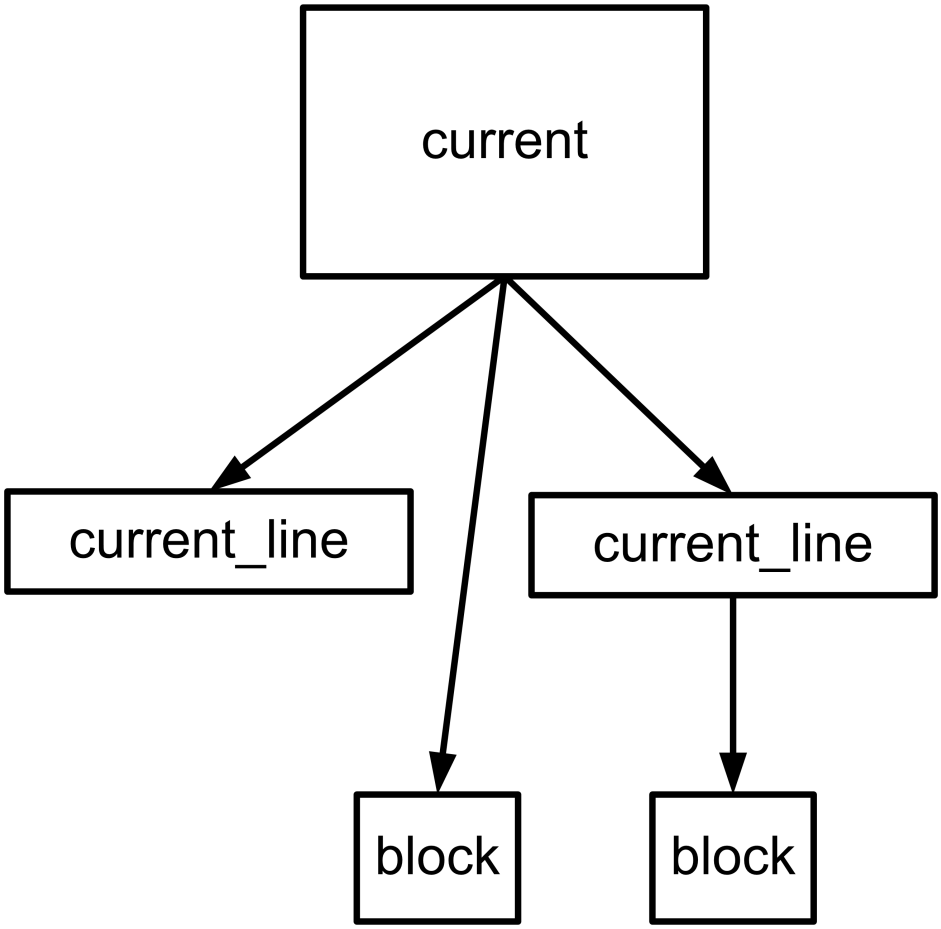
(Σχήμα 5). Με την διερεύνηση όλων των πιθανών τρόπων χρησιμοποίησης των προσωρινών πινάκων αποθήκευσης δημιουργούνται τα δέντρα αντιγραφής και τα οποίο παρουσιάζονται στα Σχήματα 6 και 7 για το κλάδο του current και previous πίνακα αντίστοιχα.



Σχήμα Α΄.5: Οι αλυσίδες αντιγραφής δεδομένων από του αρχικούς πίνακες (current και previous) δεδομένων σε μικρότερου μεγέθους πίνακες δεδομένων τους current_line, block και τους previous_line, RW αντίστοιχα.

Η εφαρμογή των μετασχηματισμών επαναχρησιμοποίησης δεδομένων θα γίνει για κάθε ένα κλάδο των δένδρων αντιγραφής δεδομένων που παρουσιάζονται στα Σχήματα 6 και 7. Έτσι κάθε κλάδος αποτελεί ένα ξεχωριστό μετασχηματισμό επαναχρησιμοποίησης δεδομένων.

Ο μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή του πίνακα current_line Αρχίζοντας τους μετασχηματισμούς από το πίνακα



Σχήμα Α΄.6: Το δέντρο με όλους τους δυνατούς συνδυασμούς αντιγραφής δεδομένων του πίνακα current.

δεδομένων current ο πρώτος μετασχηματισμός θα εισάγει το πίνακα (buffer) current_line (Σχήμα 6). Η διαδικασία είναι η ακόλουθη, τμήμα των δεδομένων από το πίνακα current θα αντιγραφούν στο πίνακα current_line και στην εφαρμογή θα χρησιμοποιηθούν μέσω του πίνακα current_line που έχει μικρότερο μέγεθος. Ο αλγόριθμος του πρώτου μετασχηματισμού επαναχρησιμοποίησης δεδομένων θα είναι ο εξής:

```
    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,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;
     int current_line[M][B];
```

```
for(x=0;x<N/B;x++)       /* For all blocks in the current frame */

 for(j=0;j<M;j++)    /* Copy data from current to buffer current_line */
  for(i=0;i<B;i++)
   current_line[i][j]=current[B*x+i][j];

 for(y=0;y<M/B;y++)
 {
  vectors_x[x][y]=0;
  vectors_y[x][y]=0;
  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;
  disty=0;
    for(k=0;k<B;k++)     /* For all pixels in the block */
   for(l=0;l<B;l++)
     {
      p1=current_line[k][B*y+l]; /* Read the data from buffer current_line
*/

        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(i==0)
        disty=distx;
        else
        {
              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(distx<min1)
        {
         min1=distx;
         bestx=i;

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

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

Ο μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή του πίνακα block Ο επόμενος μετασχηματισμός επαναχρησιμοποίησης δεδομένων εισάγει το πίνακα πρόχειρης αποθήκευσης block. Ο πίνακας block θα αποθηκεύει προσωρινά τμήμα δεδομένων από το current και θα το χρησιμοποιεί ο επεξεργαστής μέσω του πίνακα buffer. Οπότε ο αλγόριθμος μετασχηματίζεται ως ακολούθως:

```
    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,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;
     int block[B][B];

     for(x=0;x<N/B;x++)       /* For all blocks in the current frame */
      for(y=0;y<M/B;y++)
      {
       vectors_x[x][y]=0;
       vectors_y[x][y]=0;
       S=4;
```

```
while(S>0)
{
 min1=255*B*B;
 min2=255*B*B;

for(k=0;k<B;k++)    /* Copy data from current to buffer block */
 for(l=0;l<B;l++)
  block[k][l]=current[B*x+k][B*y+l];


 for(i=-S;i<S+1;i+=S)     /* For all candidate blocks in X dimension */
 {
  distx=0;
disty=0;

  for(k=0;k<B;k++)     /* For all pixels in the block */
 for(l=0;l<B;l++)
   {
    p1=block[k][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(i==0)
    disty=distx;
    else
    {
          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(distx<min1)
```

```
        {
         min1=distx;
         bestx=i;

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

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

Ο μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή των πινάκων current_line και block Ο τρίτος μετασχηματισμός επαναχρησιμοποίησης δεδομένων εισάγει και τους δυο προηγούμενους πίνακες δεδομένων των current_line και τον block. Η διαδικασία αντιγραφής δεδομένων θα ακολουθεί την ιεραρχία μνήμης, έτσι τμήμα δεδομένων από τον πίνακα current θα αντιγράφεται στο current_line και στη συνέχεια, τμήμα από το current_line θα αντιγράφεται στο block. Μέσω του πίνακα block θα γίνεται η ανάγνωση των δεδομένων από το κριτήριο του αλγορίθμου. Ο αλγόριθμος του τρίτου μετασχηματισμού θα είναι ο παρακάτω.

```
    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,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;
     int current_line[M][B];
     int block[B][B];

     for(x=0;x<N/B;x++)       /* For all blocks in the current frame */

      for(j=0;j<M;j++)    /* Copy data from current to buffer current_line */
       for(i=0;i<B;i++)
        current_line[i][j]=current[B*x+i][j];
```

```
      for(y=0;y<M/B;y++)
      {
       vectors_x[x][y]=0;
       vectors_y[x][y]=0;
       S=4;

       for(k=0;k<B;k++)    /* Copy data from current to buffer block */
        for(l=0;l<B;l++)
         block[k][l]=current_line[k][B*y+l];

     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;
      disty=0;
        for(k=0;k<B;k++)     /* For all pixels in the block */
       for(l=0;l<B;l++)
         {
          p1=current_line[k][B*y+l]; /* Read the data from buffer current_line
*/

            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(i==0)
          disty=distx;
          else
          {
                  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(distx<min1)
        {
         min1=distx;
         bestx=i;

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

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

Ο μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή του πίνακα previous_line Με τον ίδιο τρόπο γίνεται η εφαρμογή των μετασχηματισμών επαναχρησιμοποίησης δεδομένων με τον πίνακα δεδομένων previous. Πρώτη μας επιλογή είναι η εισαγωγή του πίνακα previous_line. Η διάσταση του πίνακα αυτού είναι M×(2*p+B), p=B+S+S/2+S/4. Η εισαγωγή του πίνακα αυτού είναι όμοια με την αντίστοιχη του current_line για τον current πίνακα. Ο αλγόριθμος μετασχηματίζεται στον ακόλουθο.

```
    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,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;
     int previous_line[B+2*p][M];

     for(x=0;x<N/B;x++)       /* For all blocks in the current frame */
     {

      for(i=0;i<2*p+B;i++)
```

```
        for(j=0;j<M;j++)
        {
         if(x==0)
         {
          if(i<p) previous_line[i][j]=0;
         else   previous_line[i][j]=previous[i-p][j]; /* Copy from previous array
*/
         }
         else
         {
           if(i<2*p) previous_line[i][j]=previous_line[i+B][j]; /* Reuse from the
same array*/
          else
          {
           if(x==N/B-1 && i>B+p)  previous_line[i][j]=0;
           else previous_line[i][j]=previous[B*x-p+i][j];
          }
         }
        }


       for(y=0;y<M/B;y++)
       {
        vectors_x[x][y]=0;
        vectors_y[x][y]=0;
        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;
       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]+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_line[vectors_x[x][y]+i+k+p][B*y+vectors_y[x][y]+l];

         distx+=abs(p1-p2);

         if(i==0)
           disty=distx;
           else
           {
                  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_line[vectors_x[x][y]+k+p][B*y+vectors_y[x][y]+i+l];

            disty+=abs(p1-q2);
           }
          }

          if(distx<min1)
          {
           min1=distx;
           bestx=i;

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

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

Ο μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή του πίνακα previous_line Ο μετασχηματισμός αυτός εισάγει τον πίνακα RW διαστάσεων (B+2*p)×(B+2*p) για την προσωρινή αποθήκευση τμήμα δεδομένων του πίνακας previous. Ο αρχικός αλγόριθμος PHODS μετασχηματίζεται στον παρακάτω.

```
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,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;
 int RW[B+2*p][B+2*p];

 for(x=0;x<N/B;x++)       /* For all blocks in the current frame */
  for(y=0;y<M/B;y++)
  {
   vectors_x[x][y]=0;
   vectors_y[x][y]=0;


    for(k=0;k<B+2*p;k++) /* Copy data from previous to RW */
     for(l=0;l<B+2*p;l++)
       {
        if((B*x+k-p)<0 || (B*x+k-p)>(N-1) || (B*y+l-p)<0 || (B*y+l-p)>(M-1))
          rw[k][l]=0;
        else
          {
           if(l>2*p-1 || y==0) rw[k][l]=previous[B*x+k-p][B*y+l-p];
           else rw[k][l]=rw[k][l+B];
          }
       }


   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;
   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]+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=rw[vectors_x[x][y]+i+k+p][vectors_y[x][y]+l+p]; /* Read from
RW */

         distx+=abs(p1-p2);

         if(i==0)
           disty=distx;
           else
           {
                  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= rw[vectors_x[x][y]+k+p][vectors_y[x][y]+l+p+i]; /* Read from
RW */

            disty+=abs(p1-q2);
           }
          }

          if(distx<min1)
          {
           min1=distx;
           bestx=i;

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

        S=S/2;
        vectors_x[x][y]+=bestx;
```

```
        vectors_y[x][y]+=besty;
       }
      }
     }
```

Ο μετασχηματισμός επαναχρησιμοποίησης δεδομένων με την εισαγωγή των πινάκων previous_line και RW Ο τελευταίος μετασχηματισμός επαναχρησιμοποίησης εισάγει και τους δύο πίνακες previous_line και RW στον κλάδο του πίνακα previous.

```
    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,k,l,p1,p2,q2,distx=0,disty=0,S,min1,min2,bestx,besty;
     int previous_line[B+2*p][M];
     int RW[B+2*p][B+2*p];

     for(x=0;x<N/B;x++)       /* For all blocks in the current frame */
     {

      for(i=0;i<2*p+B;i++)
       for(j=0;j<M;j++)
       {
        if(x==0)
        {
         if(i<p) previous_line[i][j]=0;
        else   previous_line[i][j]=previous[i-p][j]; /* Copy from previous array
*/
        }
        else
        {
          if(i<2*p) previous_line[i][j]=previous_line[i+B][j]; /* Reuse from the
same array*/
         else
         {
          if(x==N/B-1 && i>B+p)  previous_line[i][j]=0;
          else previous_line[i][j]=previous[B*x-p+i][j];
         }
        }
       }
```

```
for(y=0;y<M/B;y++)
{
 vectors_x[x][y]=0;
 vectors_y[x][y]=0;

 for(k=0;k<B+2*p;k++) /* Copy data from previous_line to RW */
  for(l=0;l<B+2*p;l++)
    {
     if((B*x+k-p)<0 || (B*x+k-p)>(N-1) || (B*y+l-p)<0 || (B*y+l-p)>(M-1))
       rw[k][l]=0;
     else
      {
       if(l>2*p-1 || y==0) rw[k][l]=previous_line[k-p][B*y+l-p];
/* Copy from previous_line array */
       else rw[k][l]=rw[k][l+B];   /* Reuse from the same array*/
      }
    }

 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;
  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]+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=rw[vectors_x[x][y]+i+k+p][vectors_y[x][y]+l+p]; /* Read from
RW */

  distx+=abs(p1-p2);

  if(i==0)
```

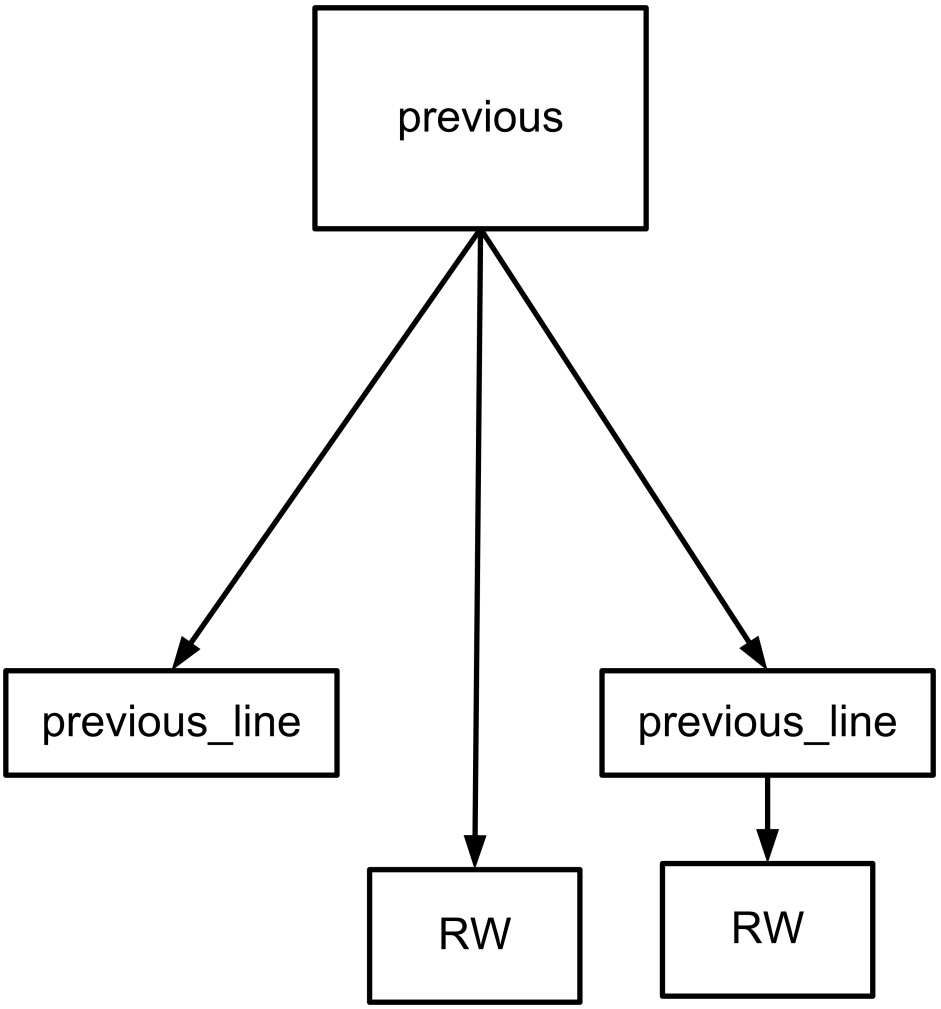
```
          disty=distx;
          else
          {
                if((B*x+vectors_x[x][y]+k)<0  ||  (B*x+vectors_x[x][y]+k)>(N-1)  ||
(B*y+vectors_y[x][y]+i+l)<0 || (B*y+vectors_y[x][y]+i+l)>(M-1))
            q2=0;
           else
            q2= rw[vectors_x[x][y]+k+p][vectors_y[x][y]+l+p+i]; /* Read from
RW */
           disty+=abs(p1-q2);
          }
         }

         if(distx<min1)
         {
          min1=distx;
          bestx=i;

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

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

Α΄.3.3 Εξαγωγή μετρήσεων

Για να αποδείξει κανείς την επιτυχία ή όχι της εφαρμογής των μετασχηματισμών πρέπει να πραγματοποιηθούν κάποιες μετρήσεις. Οι μετρήσεις των προσπελάσεων στην μνήμη δεδομένων μπορεί να γίνει με την χρήση του εργαλείου Atomium (εφόσον είναι διαθέσιμο) ή ακόμα εισάγοντας απαριθμητές (counters) σε κάθε σημείο που έχουμε ανάγνωση από κάποιο πίνακα δεδομένων. Οι μετρήσεις για την ταχύτητα εκτέλεσης (performance) και τις εντολές

που εκτελούνται μπορούν να επιτευχθούν με το προσομοιωτή ενός επεξεργαστή π.χ., ARM. Δημιουργώντας project με το εργαλείο ARMulator[2] μπορούμε να κάνουμε προσομοίωση του αλγορίθμου και να δούμε τα στατιστικά αποτελέσματα, το συνολικό αριθμό των κύκλων που απαιτούνται για την ολοκλήρωση της εκτέλεσης. Επίσης, υπολογίζει τον συνολικό αριθμό των εντολών που εκτελούνται και την εκτέλεση της εφαρμογής.

[2]http://sourceforge.net/projects/armulator/



Σχήμα Α΄.7: Το δέντρο με όλους τους δυνατούς συνδυασμούς αντιγραφής δεδομένων του πίνακα previous.