1η εργαστηριακή άσκηση

Σχεδιασμός Ενσωματωμένων Συστημάτων

Αρβανίτης Χρήστος: 03114622
Μπαγάκης Μάνος: 03114157

Ερώτημα 1

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

#include <stdio.h>
#include <stdlib.h>

//Erwthma 1 includes for gettimeofday
#include <time.h>
#include <sys/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(int current[N][M], int 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, disty, S, min1, min2, bestx,
besty;

	distx = 0;
	disty = 0;

	/*Initialize the vector motion matrices*/
	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 all blocks in the current frame*/
	for (x = 0; x < N / B; x++)
	{
	for (y = 0; y < M / B; y++)
	{
		S = 4;

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

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

			/*For all pixels in the block*/
			for (k = 0; k < B; k++)
			{
				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 all candidate blocks in Y dimension*/
                    for (i = -S; i < S + 1; i += S)
                    {
                    disty = 0;

                    /*For all pixels in the block*/
                    for (k = 0; k < B; k++)
                    {
                          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()
{
        int current[N][M], previous[N][M], motion_vectors_x[N / B][M / B],
        motion_vectors_y[N / B][M / B], i, j;

        struct timeval start, end;

        read_sequence(current, previous);
        gettimeofday(&start, NULL);
        phods_motion_estimation(current, previous, motion_vectors_x,
motion_vectors_y);
        gettimeofday(&end, NULL);

//usecond precision through 10^6 mult
        printf("%ld\n", ((end.tv_sec * 1000000 + end.tv_usec)
                    - (start.tv_sec * 1000000 + start.tv_usec)));
        return 0;
}
```

Για το πρωτο ερωτημα οι γραμμες της main συνάρτησης σχετικά με τη μέτρηση του χρόνου σε microseconds είναι, όπως φαίνεται και παραπάνω:

```
1. struct timeval start, end;
2. gettimeofday(&start, NULL);
3. gettimeofday(&end, NULL);
4. printf("%ld\n", ((end.tv_sec * 1000000 + end.tv_usec)
                      - (start.tv_sec * 1000000 + start.tv_usec)));
```

Αφού κάνουμε compile το παραπάνω πρόγραμμα, απενεργοποιώντας τα compiler optimizations του gcc, εκτελούμε το αποτέλεσμα οπότε και έχουμε την εξής μέτρηση:

~/embedded/ex1 % ./phods.out
92757 μs

Ερώτημα 2

Ο τελικά βελτιστοποιημένος κώδικας (31430.4 μs) είναι ο παρακάτω:

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

#include <stdio.h>
#include <stdlib.h>

//Ερώτημα 1 includes for gettimeofday
#include <time.h>
#include <sys/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.*/

#define NDIVB (N/B)
#define MDIVB (M/B)
#define N1 (N-1)
#define M1 (M-1)



#define bar(S) \
              min1 = min_init;\
              min2 = min_init;\
\
              /*For all candidate blocks in X dimension*/\
              for (i = -S; i < S + 1; i += S)\
              {\
                     distx = 0;\
                     disty = 0;\
\
                     /*For all pixels in the block*/\
                     for (k = 0; k < B; k++)\
                     {\
                     vectors_x_cur_k_Bx = vectors_x_cur + k + Bx;\
                     vectors_x_cur_k_Bx_i = vectors_x_cur_k_Bx + i;\
                     for (l = 0; l < B; l++)\
                     {\
                            p1 = current[k][l];\
```

```
                        vectors_y_cur_l_By = vectors_y_cur + l + By;\
                        if (i==0) \
                        {\
                        if ((vectors_x_cur_k_Bx) < 0 ||\
                               (vectors_x_cur_k_Bx) > (N1) ||\
                               (vectors_y_cur_l_By) < 0 ||\
                               (vectors_y_cur_l_By) > (M1))\
                        {\
                               p2 = 0;\
                        } else {\
                               p2 = previous[vectors_x_cur_k_Bx][vectors_y_cur_l_By];\
                        }\
\
                        inc = abs(p1 - p2); \
                        distx += inc;\
                        disty += inc;\
                        continue;\
                        }\
                        vectors_y_cur_l_By_i = vectors_y_cur_l_By + i;\
                        if ((vectors_x_cur_k_Bx_i) < 0 ||\
                               (vectors_x_cur_k_Bx_i) > (N1) ||\
                               (vectors_y_cur_l_By) < 0 ||\
                               (vectors_y_cur_l_By) > (M1))\
                        {\
                        p2 = 0;\
                        } else {\
                        p2 = previous[vectors_x_cur_k_Bx_i][vectors_y_cur_l_By];\
                        }\
\
                        distx += abs(p1 - p2);\
                        \
                        if ((vectors_x_cur_k_Bx) < 0 ||\
                               (vectors_x_cur_k_Bx) > (N1) ||\
                               (vectors_y_cur_l_By_i) < 0 ||\
                               (vectors_y_cur_l_By_i) > (M1))\
                        {\
                        q2 = 0;\
                        } else {\
                        q2 = previous[vectors_x_cur_k_Bx][vectors_y_cur_l_By_i];\
                        }\
\
                        disty += abs(p1 - q2);\
                }\
                }\
\
                if (distx < min1)\
                {\
                min1 = distx;\
                bestx = i;\
                }\
```

```
\
                        if (disty < min2)\
                        {\
                        min2 = disty;\
                        besty = i;\
                        }\
                }\
\
                vectors_x_cur += bestx;\
                vectors_y_cur += besty;


void read_sequence(int current[N][M], int 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[NDIVB][MDIVB], int vectors_y[NDIVB][MDIVB])
{
        int x, y, i, j, k, l, p1, p2, q2, distx, disty, S, min1, min2, bestx, besty;
        int Bx, By, line_fetch_x, line_fetch_y, vectors_y_cur, vectors_x_cur,inc;
        int block_fetch_x,block_fetch_y;

        distx = 0;
        disty = 0;

        int NB = N/B;
        int MB = M/B;
        int min_init = (1 << 8)*B*B;

        int vectors_x_cur_k_Bx,vectors_y_cur_l_By;
        int vectors_x_cur_k_Bx_i,vectors_y_cur_l_By_i;

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


        for (y = 0; y < MB; y++)
        {
                vectors_x_cur = 0;
                vectors_y_cur = 0;

                By = B*y;

                bar(4);

                bar(2);

                bar(1);

                //they should be updated before the xy pair is changed for valid code
                vectors_x[x][y] = vectors_x_cur;
                vectors_y[x][y] = vectors_y_cur;
        }

        }
}

int main()
{
        int current[N][M], previous[N][M], motion_vectors_x[N / B][M / B],
        motion_vectors_y[N / B][M / B], i, j;
```

```
        struct timeval start, end;

        read_sequence(current, previous);
        gettimeofday(&start, NULL);
        phods_motion_estimation(current, previous, motion_vectors_x, motion_vectors_y);
        gettimeofday(&end, NULL);

//usecond precision through 10^6 mult
        printf("%ld\n", ((end.tv_sec * 1000000 + end.tv_usec)
                        - (start.tv_sec * 1000000 + start.tv_usec)));
        return 0;
}
```

Σε σχέση με τον αρχικό κώδικα έγιναν οι εξής αλλαγές:

1. Παρατηρούμε πως υπάρχουν τα δύο ακόλουθα loops με ίδια άκρα και ανεξάρτητα δεδομένα εντός τους, συνεπώς και μπορούμε να τις συνενώσουμε (loop merging):

```
for (i = -S; i < S + 1; i += S)
```

   Με αυτόν τον τρόπο γλιτώνουμε επιβαρύνσεις απο τους επαναληπτικούς βρόχους.

2. Ακριβώς όπως πριν, παρατηρούμε πως οι βρόχοι με δείκτη τον k εντός του προηγούμενου loop επεξεργάζονται ανεξάρτητα δεδομένα μπορούν επίσης να συγχωνευθούν (loop merging):

```
for (k = 0; k < B; k++)
```

3. Οι βρόχοι εντός του προηγούμενου συγχωνευμένου loop μπορούν επίσης να γίνουν merge:

```
for (l = 0; l < B; l++)
```

Εδώ αξίζει να παρατηρήσουμε πως το p1 έχει την ίδια τιμή και τις δύο φορές οπότε δεν χρειάζεται να το υπολογίσουμε εις διπλούν.

4. Παρατηρώντας την μορφή του κώδικα, γίνεται εμφανές πως στη περίπτωση που i=0 τα p2 και q2 είναι ίσα οπότε μπορούμε να διατυπώσουμε μια ξεχωριστή περίπτωση ώστε να αποφευχθεί ο διπλός υπολογισμός.

```
if (i==0) //if i==0 p2 == q2
{
        if ((Bx + vectors_x_cur + k) < 0 ||
        (Bx + vectors_x_cur + k) > (N - 1) ||
        (By + vectors_y_cur + l) < 0 ||
        (By + vectors_y_cur + l) > (M - 1))
        {
        p2 = 0;
        } else {
```

```
        p2 = prev_block[p + vectors_x_cur + k][vectors_y_cur + l + p];
        q2=p2;
        }
inc = abs(p1 - p2);
        distx += inc;
        disty += inc;
        continue;
}
```

5. Σε κάθε iteration x,y παρατηρούμε πως χρησιμοποιούνται μόνο οι τιμές vectors_x[x][y] και vectors_y[x][y] και μόνο για διάβασμα. Επίσης παρατηρούμε πως οι τιμές τους αλλάζουν μόνο για κάθε πιθανή τιμή του S (4,2,1). Συνεπώς μπορούμε να χρησιμοποιούμε μια buffer προσωρινή μεταβλητή για κάθε πίνακα vectors_* (vectors_x_cur, vectors_y_cur), ώστε να αποφεύγουμε τις περιττές προσβάσεις στους πίνακες αυτούς, και να ενημερώνουμε κατάλληλα τους vectors_* σε κάθε αλλαγή του S.

```
        while (S > 0)
        {
        min1 = min_init;
        min2 = min_init;

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

                /*For all pixels in the block*/
                for (k = 0; k < B; k++)
                {
                for (l = 0; l < B; l++)
                {
                        p1 = cur_block[k][l];

                        if (i==0) //if i==0 p2 == q2
                        {
                        if ((Bx + vectors_x_cur + k) < 0 ||
                                (Bx + vectors_x_cur + k) > (N - 1) ||
                                (By + vectors_y_cur + l) < 0 ||
                                (By + vectors_y_cur + l) > (M - 1))
                        {
                                p2 = 0;
                        } else {
                                p2 = prev_block[p + vectors_x_cur + k][vectors_y_cur + l
+ p];

                                q2=p2;
                        }
```

```
                                inc = abs(p1 - p2);
                                distx += inc;
                                disty += inc;
                                continue;
                                }

                                if ((Bx + vectors_x_cur + i + k) < 0 ||
                                       (Bx + vectors_x_cur + i + k) > (N - 1) ||
                                       (By + vectors_y_cur + l) < 0 ||
                                       (By + vectors_y_cur + l) > (M - 1))
                                {
                                p2 = 0;
                                } else {
                                p2 = prev_line[vectors_x_cur + i + k + p][p + vectors_y_cur +
l];
                                }

                                distx += abs(p1 - p2);

                                if ((Bx + vectors_x_cur + k) < 0 ||
                                       (Bx + vectors_x_cur + k) > (N - 1) ||
                                       (By + vectors_y_cur + i + l) < 0 ||
                                       (By + vectors_y_cur + i + l) > (M - 1))
                                {
                                q2 = 0;
                                } else {
                                q2 = prev_line[p + vectors_x_cur + k][p + vectors_y_cur + i +
l];
                                }

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

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

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

                S = (S >> 1);

                vectors_x_cur += bestx;
```

```
            vectors_y_cur += besty;
            }

            //they should be updated before the xy pair is changed for valid code
            vectors_x[x][y] = vectors_x_cur;
            vectors_y[x][y] = vectors_y_cur;
    }
```

Η προηγούμενη αλλαγή συνεπάγεται πως δεν χρειάζεται να αρχικοποιήσουμε τους vectors_x και vectors_y πίνακες, αφού αρχικοποιούμε τις αντίστοιχες buffer μεταβλητές.

7. Στον κώδικα, παρατηρούνται κάποιες αριθμητικές παραστάσεις οι οποίες επαναλαμβάνονται και αποτελούν ορίσματα για άλλες παραστάσεις (δηλαδή χρησιμοποιούνται μονάχα για ανάγνωση), συνεπώς θα μπορούσαμε να τις υπολογίζουμε μία φορά σε μια buffer μεταβλητή, και στη συνέχεια να χρησιμοποιούμε αυτή για τους υπολογισμούς. Αυτές οι πράξεις είναι οι ακόλουθες:
    a. B*x => μεταβλητή Bx
    b. B*y => μεταβλητή By
    c. N/B => NDIVB macro
    d. M/B => MDIVB macro
    e. vectors_x_cur + k + Bx => μεταβλητή vectors_x_cur_k_Bx
    f. vectors_x_cur_k_Bx + i => μεταβλητή vectors_x_cur_k_Bx_i
    g. vectors_y_cur + l + By => μεταβλητή vectors_y_cur_l_By
    h. vectors_y_cur_l_By + i => μεταβλητή vectors_y_cur_l_By_i
    i. 255*B*B => μεταβλητή min_init

Οι παραπάνω buffers χρησιμοποιούνται με τον ακόλουθο τρόπο και οι αρχικοποιήσεις τους γίνονται σε τέτοια σημεία ώστε να αποφεύγονται περιττοί υπολογισμοί:

```
            min1 = min_init;
        min2 = min_init;

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

                /*For all pixels in the block*/
                for (k = 0; k < B; k++)
                {
                vectors_x_cur_k_Bx = vectors_x_cur + k + Bx;
                vectors_x_cur_k_Bx_i = vectors_x_cur_k_Bx + i;
                for (l = 0; l < B; l++)
                {
                        p1 = current[k][l];
                        vectors_y_cur_l_By = vectors_y_cur + l + By;
                        if (i==0)
                        {
```

```
        if ((vectors_x_cur_k_Bx) < 0 ||
               (vectors_x_cur_k_Bx) > (N1) ||
               (vectors_y_cur_l_By) < 0 ||
               (vectors_y_cur_l_By) > (M1))
        {
               p2 = 0;
        } else {
               p2 = previous[vectors_x_cur_k_Bx][vectors_y_cur_l_By];
        }

        inc = abs(p1 - p2);
        distx += inc;
        disty += inc;
        continue;
        }
        vectors_y_cur_l_By_i = vectors_y_cur_l_By + i;
        if ((vectors_x_cur_k_Bx_i) < 0 ||
               (vectors_x_cur_k_Bx_i) > (N1) ||
               (vectors_y_cur_l_By) < 0 ||
               (vectors_y_cur_l_By) > (M1))
        {
        p2 = 0;
        } else {
        p2 = previous[vectors_x_cur_k_Bx_i][vectors_y_cur_l_By];
        }

        distx += abs(p1 - p2);

        if ((vectors_x_cur_k_Bx) < 0 ||
               (vectors_x_cur_k_Bx) > (N1) ||
               (vectors_y_cur_l_By_i) < 0 ||
               (vectors_y_cur_l_By_i) > (M1))
        {
        q2 = 0;
        } else {
        q2 = previous[vectors_x_cur_k_Bx][vectors_y_cur_l_By_i];
        }

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

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

if (disty < min2)
{
```

```
                min2 = disty;
                besty = i;
                }
        }

        vectors_x_cur += bestx;
        vectors_y_cur += besty;
```

8. Παρατηρούμε πως ο κώδικας μέσα στο loop while(S>0) εκτελείται τρεις φορές για κάθε xy συνδυασμό. Συγκεκριμένα το S παίρνει τις τιμές 4,2,1 για κάθε xy. Με loop unrolling θα μπορούσαμε να απεμπλακούμε από αυτό το while τοποθετώντας σταθερές τιμές για το S. Μία επιπλέον βελτιστοποίηση, όπως βλέπουμε και στον τελικά βελτιστοποιημένο κώδικα είναι να χρησιμοποιήσουμε macros και όχι συναρτήσεις όπου είναι δυνατό, ώστε να γλιτώσουμε το overhead της κλήσης και του περάσματος παραμέτρων.

```
/*For all blocks in the current frame*/
        for (x = 0; x < NB; x++){
            Bx = B*x;
        for (y = 0; y < MB; y++){
                vectors_x_cur = 0;
                vectors_y_cur = 0;

                By = B*y;

                bar(4); //bar is the macro for each S

                bar(2);

                bar(1);

                //they should be updated before the xy pair is changed for valid code
                vectors_x[x][y] = vectors_x_cur;
                vectors_y[x][y] = vectors_y_cur;
        }

        }
```

Ερώτημα 3

Για 100 επαναλήψεις σε κάθε block size, εξάγουμε τις ακόλουθες μετρήσεις:

Block Size	**Time (avg per 100 repeats)**

1	82827μs
2	50236μs
4	36139.5μs
8	32804.4μs
16	31430.4μs

Συνεπώς, και με βάση τα παραπάνω αποτελέσματα, συμπεραίνουμε πως το μέγεθος Block ίσο με 16 αποτελεί, για τη συγκεκριμένη αρχιτεκτονική ARM που προσομοιώνουμε στο QEMU, το βέλτιστο δυνατό. Για την πραγματοποιήση των προσομοιώσεων χρησιμοποιήθηκε το ακόλουθο script, το οποίο δέχεται ως όρισμα το πλήθος των επαναλήψεων ανα μέγεθος block size και στη συνέχεια για κάθε τιμή B (1,2,4,8,16) εκτελεί τόσες επαναλήψεις υπολογίζοντας το μέσο όρο χρόνου εκτέλεσης:

```
#INIT DEFINE B = 0 !!!!!
#COUNTER -> NUMBER OF ITERATION PER SIMULATION FOR ACCURACY PURPOSES
prev=0
for B in 1 2 4 8 16
do

    sed -i "13s/$prev/$B/" phods2.c && gcc phods2.c -o a.out ;

        COUNTER=$1;
        while [  $COUNTER -gt 0 ];
        do
        ./a.out >> results.txt;
        let COUNTER=COUNTER-1
        done
       echo “Block size: $B” >> mean.txt
        awk '{ total += $1; count++ } END { print total/count }' results.txt >> mean.txt &&
rm results.txt



    prev=$B;
done
sed -i "13s/$B/0/" phods2.c
```

Ερώτημα 4

```
#INIT DEFINE B = 0 !!!!!
#COUNTER -> NUMBER OF ITERATION PER SIMULATION FOR ACCURACY PURPOSES
prev1=0
prev2=0
for B1 in 1 2 3 4 6 8 9 12 16 18 24 36 48 72 144
do
    for B2 in 1 2 4 8 11 16 22 44 88 176
    do

         sed -i "13s/$prev1/$B1/" phods3.c && sed -i "14s/$prev2/$B2/" phods3.c && gcc
phods3.c -o a.out ;

        COUNTER=$1;
        while [  $COUNTER -gt 0 ];
        do
                ./a.out >> results.txt;
                let COUNTER=COUNTER-1
        done
        awk '{ total += $1; count++ } END { print total/count }' results.txt >> mean.txt &&
rm results.txt




         prev2=$B2;
    done
    prev1=$B1;
done
sed -i "13s/$B1/0/" phods3.c
sed -i "14s/$B2/0/" phods3.c
```

Bx	By	Exec time
1	1	77169.3
1	2	52902.8
1	4	40776.7
1	8	36341.3
1	11	34461.1

1	16	33372
1	22	32650.8
1	44	30902.1
1	88	30504.6
1	176	32097.5
2	1	85081.6
2	2	45511.2
2	4	39739.1
2	8	38813.9
2	11	35487.8
2	16	33925.8
2	22	32933.9
2	44	32822.1
2	88	31546.6
2	176	32175.1
3	1	53571.9
3	2	45066.9
3	4	43503.5
3	8	39432.1
3	11	35852.7
3	16	37929.4
3	22	32959.9
3	44	36817.6
3	88	33478

3	176	31098.4
4	1	48200
4	2	39378.5
4	4	34709.4
4	8	31852.1
4	11	33469.9
4	16	31507
4	22	31477.5
4	44	30553.7
4	88	32926.1
4	176	32270.8
6	1	48539.3
6	2	38017
6	4	35107.2
6	8	31170.5
6	11	32194.9
6	16	30536.5
6	22	31646.5
6	44	30568.4
6	88	30664.2
6	**176**	**29773.9**
8	1	43130.7
8	2	36677.1
8	4	34024.1

8	8	31590.7
8	11	33588.7
8	16	33394.4
8	22	33032.3
8	44	34404.5
8	88	31552.5
8	176	31804.7
9	1	45885.3
9	2	39325.2
9	4	35272.2
9	8	33885.7
9	11	32059.2
9	16	35231.6
9	22	33788.4
9	44	32913.1
9	88	35980.2
9	176	37626.5
12	1	43277.6
12	2	36625.2
12	4	35011.6
12	8	36902.3
12	11	32199.7
12	16	31621.6
12	22	33267

12	44	31074.8
12	88	32634.5
12	176	32753.7
16	1	41632.8
16	2	36575.4
16	4	32955
16	8	34511.8
16	11	32038.6
16	16	37051.7
16	22	31021.9
16	44	36725.5
16	88	36292.6
16	176	32482.4
18	1	44804.9
18	2	37643.9
18	4	35041.8
18	8	33808.4
18	11	31269.5
18	16	31451.5
18	22	31921
18	44	33093.9
18	88	36767
18	176	30831.2
24	1	39976.4

24	2	37604.5
24	4	36623.7
24	8	33183.3
24	11	31822.6
24	16	30957
24	**22**	**29927.3**
24	44	31688
24	**88**	**29938.8**
24	176	32865.4
36	1	42609.3
36	2	37258
36	4	34052.8
36	8	32073.3
36	11	33714.1
36	16	31693.4
36	22	32047.2
36	44	31925
36	88	32090.8
36	176	31435.6
48	1	40304.4
48	2	35258.9
48	4	33759.7
48	8	31331.5
48	11	32651.7

48	16	31446.3
48	22	31471.4
48	44	30613.3
48	88	33409.4
48	176	31027.1
72	1	43664.1
72	2	37470.8
72	4	34037
72	8	31757.4
72	11	33874.7
72	16	31293.5
72	22	30844.7
72	44	30795.1
72	88	30710
72	176	50750.9
144	1	44586.1
144	2	35695.1
144	4	36285.1
144	8	33954.7
144	11	33200.8
144	16	36387.9
144	22	32047.8
144	44	31859.3
144	88	31682.3

144	176	33008.5

Όπως παρατηρούμε στα προηγούμενα δεδομένα, για σταθερό Bx, δηλαδή για σταθερό αριθμό γραμμών, το σύστημα εκτελείται γρηγορότερα για μεγαλύτερο μέγεθος γραμμών (οπότε αριθμό από στήλες By). Αυτο δικαιολογείται ακριβώς από το γεγονός ότι το iteration των current και previous γίνεται με fixed γραμμή και μεταβολή της στήλης, για κάθε μία από τις γραμμές.

```
for x{
        for y
}
```

Συνεπώς υφίστανται λιγότερα cache misses και μειώνεται ο χρόνος εκτέλεσης του κρίσιμου κομματιού κώδικα.

Με τον ίδιο ακριβώς τρόπο δικαιολογείται το γεγονός πως δεν υπάρχει κανένα pattern για σταθερές στήλες και μεταβλητές γραμμές, αφού δεν συνάδει με τον τρόπο που διαπερνάμε τους πίνακες που έρχονται στη μνήμη.

Όσον αφορά την ευριστική προσέγγιση, θα χρησιμοποιήσουμε τον εξής αλγόριθμο:

- Βήμα 1 : Κάνε τις επαναλήψεις για μία τυχαία γραμμή x (δηλαδή κρατάμε σταθερή τη μία διάσταση του μπλοκ).
- Βήμα 2 : Βρες τον καλύτερο συνδυασμό x,y από αυτή τη στήλη.
- Βήμα 3 : Κρατώντας τώρα σταθερό το καλύτερο y που βρήκαμε επανέλαβε το βήμα 1 τις προσομοιώσεις με σταθερο το y.
- Βήμα 4: Όταν ελέγξουμε μια γραμμή ή στήλη παραμέτρων και δεν βρούμε καλύτερο αποτέλεσμα από το στοιχείο που ξεκινήσαμε, ο αλγόριθμος τερματίζει (δεν θα ξανακοιτάμε τις ήδη υπολογισμένες περιπτώσεις αφού ξέρουμε ότι δεν είναι οι καλύτεροι χρόνοι, αποθηκεύοντας τα ήδη ελεγμένα configurations, προκειμένου να αποφευχθούν περιττές επαναλήψεις)

Εφαρμογή :

Έστω ότι ξεκινάμε με την πρώτη γραμμή και ελέγχουμε όλους τους χρόνους της μορφής 1,y.

1 - 1 : 77169.3
1 - 2 : 52902.8
1 - 4 : 40776.7
1 - 8 : 36341.3
1 - 11 : 34461.1
1 - 16 : 33372
1 - 22 : 32650.8
1 - 44 : 30902.1
1 - 88 : 30504.6
1 - 176 : 32097.5

Παρατηρούμε ότι y=88 το καλύτερο, επομένως τρέχουμε τις προσομοιώσεις με y=88

1 - 88 : 30504.6
2 - 88 : 31546.6
3 - 88 : 33478
4 - 88 : 32926.1
6 - 88 : 30664.2
8 - 88 : 31552.5
9 - 88 : 35980.2
12 - 88 : 32634.5
16 - 88 : 36292.6
18 - 88 : 36767
24 - 88 : 29938.8
36 - 88 : 32090.8
48 - 88 : 33409.4
72 - 88 : 30710
144 - 88 : 31682.3

Παρατηρούμε ότι x=24 το καλύτερο, επομένως τρέχουμε τις προσομοιώσεις με x=24

24 - 1 : 39976.4
24 - 2 : 37604.5
24 - 4 : 36623.7
24 - 8 : 33183.3
24 - 11 : 31822.6
24 - 16 : 30957
24 - 22 : 29927.3
24 - 44 : 31688
24 - 88 : 29938.8
24 - 176 : 32865.4

Παρατηρούμε ότι y=22 το καλύτερο, επομένως τρέχουμε τις προσομοιώσεις με y=22

2 - 22 : 32933.9
3 - 22 : 32959.9
4 - 22 : 31477.5
6 - 22 : 31646.5
8 - 22 : 33032.3
9 - 22 : 33788.4
12 - 22 : 33267
16 - 22 : 31021.9
18 - 22 : 31921
24 - 22 : 29927.3
36 - 22 : 32047.2
48 - 22 : 31471.4
72 - 22 : 30844.7
144 - 22 : 32047.8

Παρατηρούμε ότι δεν βρήκαμε μικρότερη τιμή οπότε ο αλγόριθμος σταματάει εδώ.

Το αποτέλεσμα που βρήκαμε είναι η δεύτερη καλύτερη τιμή από όλες που παρατηρήσαμε (τρέξαμε 10 φορές κάθε σετ παραμέτρων του μπλοκ και υπολογίζουμε την μέση τιμή με ακρίβεια ενός δεκαδικού ψηφίου).