Εργασία 4

Ονοματεπώνυμο:
Αριθμός Ταυτότητας:

Ερώτημα:

1) **PERIL-L τον ψευδο-κώδικα για μια συνάρτηση που θα υλοποίηση με threads την char_count (5%)(**Courier New 10**):**

```
char * array;
char character;
int length=0
int sum=0;
      forall threadID in (0..k-1){
            int i=0;
            local_count = 0;
            int length = size(array[]);
            int array_lcl[length] = localize(array[]);
            int begin = threadID * length;
       for (i=begin;  i< begin+length ; i++) {
       if (array_lcl[i] == character) {
             local_count++;
            }
            }
            sum += local_count;
      }
```

2)Υλοποίηση της char_count_pthreads_consecutive **και την** char_count_pthreads_interleaved**. (15% Κώδικας + 5% Σχόλια) (**Courier New 10**):**

```
void *pthread_count_consecutive(void *args){
//Getting the data of the struct and using them in local variables
 struct consec_data *data = (struct consec_data*) args;
 char *array = data->array;
 int count =0;
 char character = data->character;
 int length = data->length;
 pthread_mutex_t *pointer_mutex = data->pointer_mutex;

//Comparing every character of the data chunk if it matches the given character and
increasing the count if a match is found.

       int i=0;
       for (i=0; i<length; i++) {
          if (array[i] == character) {
              count++;
          }
}
//Protecting the global count with a mutex, ensuring that the global count is
synchronized among the threads.
pthread_mutex_lock(pointer_mutex);
global_count = global_count + count;
pthread_mutex_unlock(pointer_mutex);
return (void *) data;
}
```

```
int char_count_pthreads_consecutive(char * vector, int size, char c, int numThreads){
int sum =0;   //Local Counter for sum
//Array with structs based on number of threads
struct consec_data threadData[numThreads];
//Calculating the part of array for every thread that they will process
int chunk = size/numThreads;
//Array with threads based on number of Threads
pthread_t threads[numThreads];.
int i = 0;
//Mutex for protecting the counts;
pthread_mutex_t *mtx=&mutex;
//Initialising every struct with the needed data
        for (i=0; i<numThreads; i++) {
            threadData[i].array = vector + i*chunk;
            threadData[i].character = c;
            threadData[i].length = chunk;
            threadData[i].pointer_mutex = mtx;
//Creating and thread and calling the function of count with every struct. If struct
fails to be created then an error is returned.
 if (pthread_create(&threads[i],NULL,pthread_count_consecutive, (void
*)&threadData[i])){
                  printf("ERROR!!!\n");
                  exit(-1);
            }
        }

//Make sure that every struct completed its job otherwise, an error is returned.
       for (i=0;i<numThreads;i++){
            void *pv;
            if(0 != pthread_join(threads[i], &pv)){
                  printf("ERROR2\n");
                  exit(-1);
            }
      }
//Getting the sum of the global count.
      sum = global_count;
     return sum;
}


void *pthread_count_interleaved(void *args){
//Getting the data of the struct and using them in local variables
       struct interleaved_data *data = (struct interleaved_data*) args;
       char *array = data->array;
       int count =0;
       int distance = data->distance;
       char character = data->character;
       int length = data->length;
       pthread_mutex_t *pointer_mutex = data->pointer_mutex;

//Comparing every character of the data chunk if it matches the given character and
increasing the count if a match is found. The loop is getting each value of array
interleaved so that's why the I = i+distance.
       int i=0;
       for (i=0; i<length; i=i+distance) {
          if (array[i] == character) {
              count++;
          }
}
//Protecting the global count with a mutex, ensuring that the global count is
synchronized among the threads.
      pthread_mutex_lock(pointer_mutex);
```

```
      global_count = global_count + count;
      pthread_mutex_unlock(pointer_mutex);
return (void *) data;
}
int char_count_pthreads_interleaved(char * vector, int size, char c, int numThreads){

//Local Counter for sum
int sum =0;
//Ensuring that the global count is not getting values from the previous functions.
global_count =0;
//Array with structs based on number of threads
struct interleaved_data threadData[numThreads];
//Array with threads based on number of Threads
 pthread_t threads[numThreads];
 int i = 0;
pthread_mutex_t *mtx=&mutex;

        for (i=0; i<numThreads; i++) {
//The address that the thread is starting to process the chunk data
            threadData[i].array = vector+i;
            threadData[i].character = c;
            threadData[i].length = size;
//The next position of the array that is processed by thread.
            threadData[i].distance = numThreads;
            threadData[i].pointer_mutex = mtx;

//Creating and thread and calling the function of count with every struct. If struct
fails to be created then an error is returned.
  if (pthread_create(&threads[i],NULL,pthread_count_interleaved, (void
*)&threadData[i])){
                  printf("ERROR!!!\n");
                  exit(-1);
            }
        }
//Make sure that every struct completed its job otherwise, an error is returned.
       for (i=0;i<numThreads;i++){
            void *pv;
            if(0 != pthread_join(threads[i], &pv)){
                  printf("ERROR2\n");
                  exit(-1);
            }
      }
//Getting the sum of the global count.
      sum = global_count;
return sum;
}
```

3) Συνάρτηση `char_count_pthreads_consecutive_AVX2`**. (10% Κώδικας + 5% Σχόλια)** **(**`Courier New 10`**):**

```
void *char_count_consecutive_AVX2(void *args){

struct consec_data *data = (struct consec_data*) args;
//Getting the data of the struct and using them in local variables
char *array = data->array;
int count =0;
char character = data->character;
int length = data->length;
```

```
pthread_mutex_t *pointer_mutex = data->pointer_mutex;
//calling the AVX2 function
count = char_count_AVX2(array,length,character);
//Protecting the global count with a mutex, ensuring that the global count is
synchronized among the threads.
pthread_mutex_lock(pointer_mutex);
global_count = global_count + count;
pthread_mutex_unlock(pointer_mutex);
return (void *) data;
}



int char_count_pthreads_consecutive_AVX2(char * vector, int size, char c, int
numThreads){
//Ensuring that the global count is not getting values from the previous functions.
Global_count =0;
int sum =0;
//Array with structs based on number of threads
struct consec_data threadData[numThreads];
//Calculating the part of array for every thread that they will process
 int chunk = size/numThreads;
//Array with threads based on number of Threads
 pthread_t threads[numThreads];
 int i = 0;
      pthread_mutex_t *mtx=&mutex;


        for (i=0; i<numThreads; i++) {
//The address that the thread is starting to process the chunk data
      threadData[i].array = vector + i*chunk;
      threadData[i].character = c;
//The next position of the array that is processed by thread.
      threadData[i].length = chunk;
       threadData[i].pointer_mutex = mtx;
//Creating and thread and calling the function of count with every struct. If struct
fails to be created then an error is returned.
 if (pthread_create(&threads[i],NULL,char_count_consecutive_AVX2, (void
*)&threadData[i])){
                  printf("ERROR!!!\n");
                  exit(-1);
            }
         }
//Make sure that every struct completed its job otherwise, an error is returned.

      for (i=0;i<numThreads;i++){
            void *pv;
            if(0 != pthread_join(threads[i], &pv)){
                  printf("ERROR2\n");
                  exit(-1);
            }
      }
//Getting the sum of the global count.
sum = global_count;
return sum;
}
```

9) Perf stat analysis (10%)
10) Load Balancing (5%)
11) Ιδανικός αριθμός threads (με αιτιολόγηση/μετρίσεις) (5%)

Παράρτημα:

1) **Πίνακας με όλες τις μετρήσεις (10+2 τιμές ανά ARRAY_SIZE)**

4) Αναμενόμενη επιτάχυνση (Speedup) των τριών συναρτήσεων. Επεξήγηση. (10%)

$$Speedup\_parallel = \frac{Serial\ execution}{pThreads\ execution}$$

- **Υπολογισμός speedup για pThreads Consecutive:**

Το $Fraction_{parallel}$ είναι ισο με 0.9. Το 0.9 το υπολογίζουμε από το πόσο παράλληλο είναι το πρόγραμμα μας. Αυτό προκύπτει από το ότι υπάρχουν overheads για τη δημιουργία του thread γι' αυτό και δεν είναι εντελώς παράλληλο.

Για 1 thread το speedup_parallel θα είναι:

$$Speedup\_parallel = \frac{1}{1}$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{1}} = 1$$

Οπότε αναμένουμε speedup 1 για 1 thread

Για 2 threads το speedup_parallel θα είναι:

$$Speedup\_parallel = \frac{1}{\frac{1}{2}} = 2$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{2}} = 1.81$$

Οπότε αναμένουμε speedup 1.81 για 2 threads

Για 4 threads το speedup_parallel θα είναι:

$$Speedup\_parallel = \frac{1}{\frac{1}{4}} = 4$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{4}} = 3.07$$

Οπότε αναμένουμε speedup 3.07 για 4 threads

- **Υπολογισμός speedup για pThreads Interleaved:**

Το $Fraction_{parallel}$ είναι ισο με 1 αφού μετράμε μόνο για τη συνάρτηση και όχι για ολόκληρο το πρόγραμμα

Για 1 thread το speedup_parallel θα είναι:

$$Speedup\_parallel = \frac{1}{1}$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{1}} = 0.909$$

Οπότε αναμένουμε speedup 1 για 1 thread

Για 2 threads το speedup_parallel θα είναι:

$$Speedup\_parallel = \frac{1}{\frac{1}{2}} = 2$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{2}} = 1.81$$

Οπότε αναμένουμε speedup 1.81 για 2 threads

Για 4 threads το speedup_parallel θα είναι:

$$Speedup\_parallel = \frac{1}{\frac{1}{4}} = 4$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{4}} = 3.07$$

Οπότε αναμένουμε speedup 3.07 για 4 threads

- **Υπολογισμός speedup για pThreads με AVX2:**

Το $Fraction_{parallel}$ είναι ισο με 1 αφού μετράμε μόνο για τις συναρτήσεις και όχι για ολόκληρο το πρόγραμμα

Για 1 thread το speedup_parallel θα είναι:

$$Speedup_{parallel} = \frac{1}{\frac{1}{32} + \frac{1}{128}} = 25.6$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{25.6}} = 7.39$$

Οπότε αναμένουμε speedup 7.39 για 1 thread

Για 2 threads το speedup_parallel θα είναι:

$$Speedup\_parallel = \frac{1}{(\frac{1}{64}+\frac{1}{128})/2} = 51.2$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{51.2}} = 8.50$$

Οπότε αναμένουμε speedup 8.50 για 2 threads

Για 4 threads το speedup_parallel θα είναι:

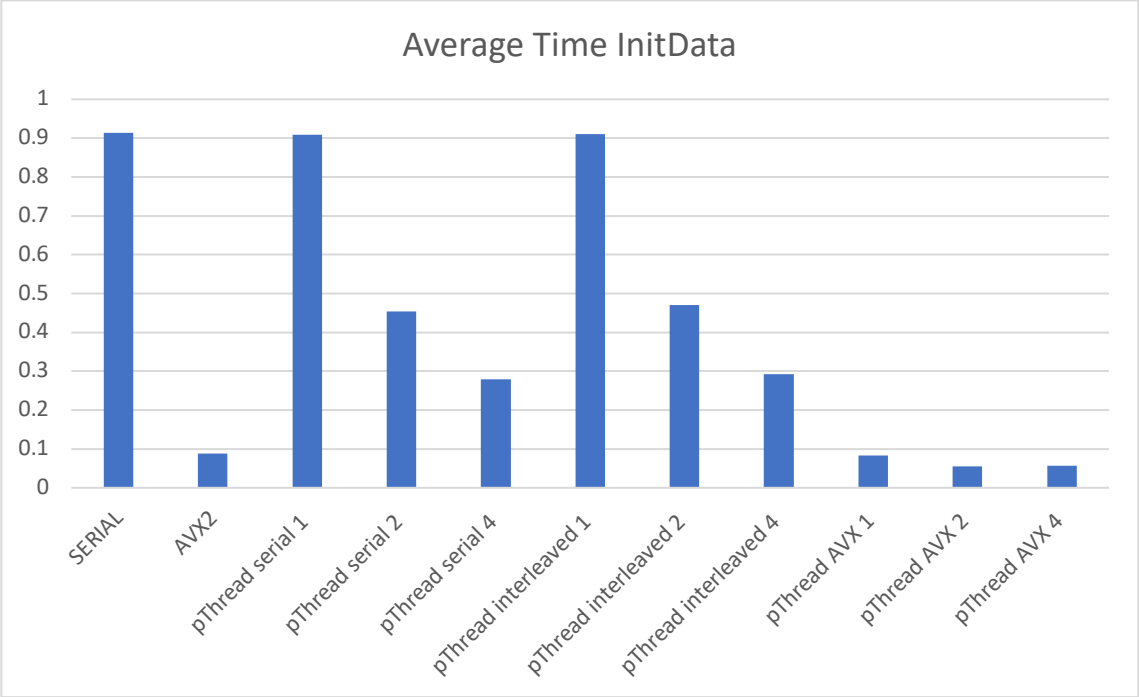
$$Speedup\_parallel = \frac{1}{(\frac{1}{128}+\frac{1}{128})/2} = 102.4$$

$$Speedup\_overall = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction\_parallel}{Speedup\_parallel}}$$

$$Speedup_{overall} = \frac{1}{(1-0.9)+\frac{0.9}{102.4}} = 9.19$$

Οπότε αναμένουμε speedup 9.11 για 4 threads

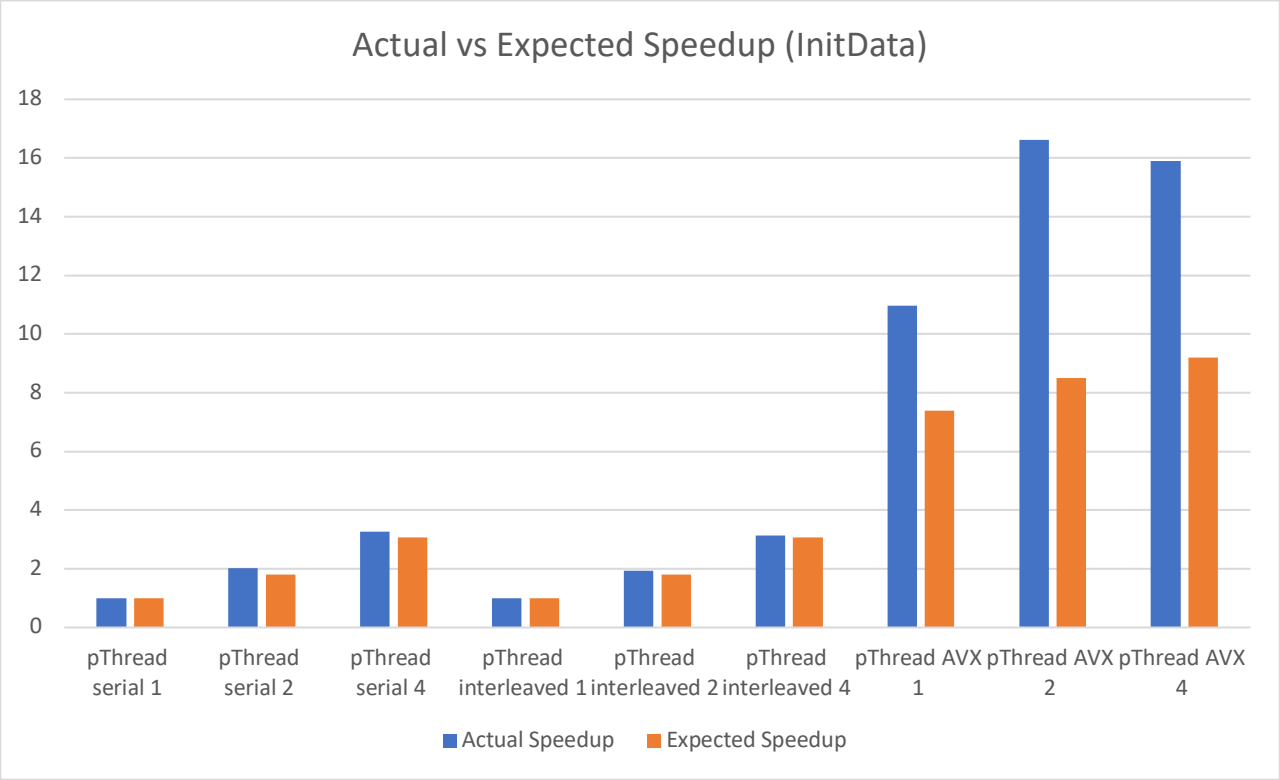
5) Χρόνο εκτέλεσης της κάθε συνάρτησης (5%)



	Average TIme
SERIAL	0.91416667
AVX2	0.08833333
pThread serial 1	0.90916667
pThread serial 2	0.45333333
pThread serial 4	0.27916667
pThread interleaved 1	0.91
pThread interleaved 2	0.47
pThread interleaved 4	0.29166667
pThread AVX 1	0.08333333
pThread AVX 2	0.055
pThread AVX 4	0.0575

Παρατηρούμε ότι υψηλότερο χρόνο έχουν οι συναρτήσεις SERIAL, pthreadConsecutive με 1 thread και pthreadInterleaved με 1 thread. Τους χαμηλότερους χρόνους παρουσιάζουν τα threads με AVX2 εντολές, ωστόσο ο αριθμός των threads σε αυτή την περίπτωση δεν παρουσιάζει και μεγάλη διαφορά. Αντίθετα, στα threads που είναι consecutive και interleaved όσο περισσότερα threads έχουμε τόσο μικρότερος είναι ο χρόνος που χρειάζονται για να διεκπεραιώσουν τη δουλειά που τους ανατέθηκε.

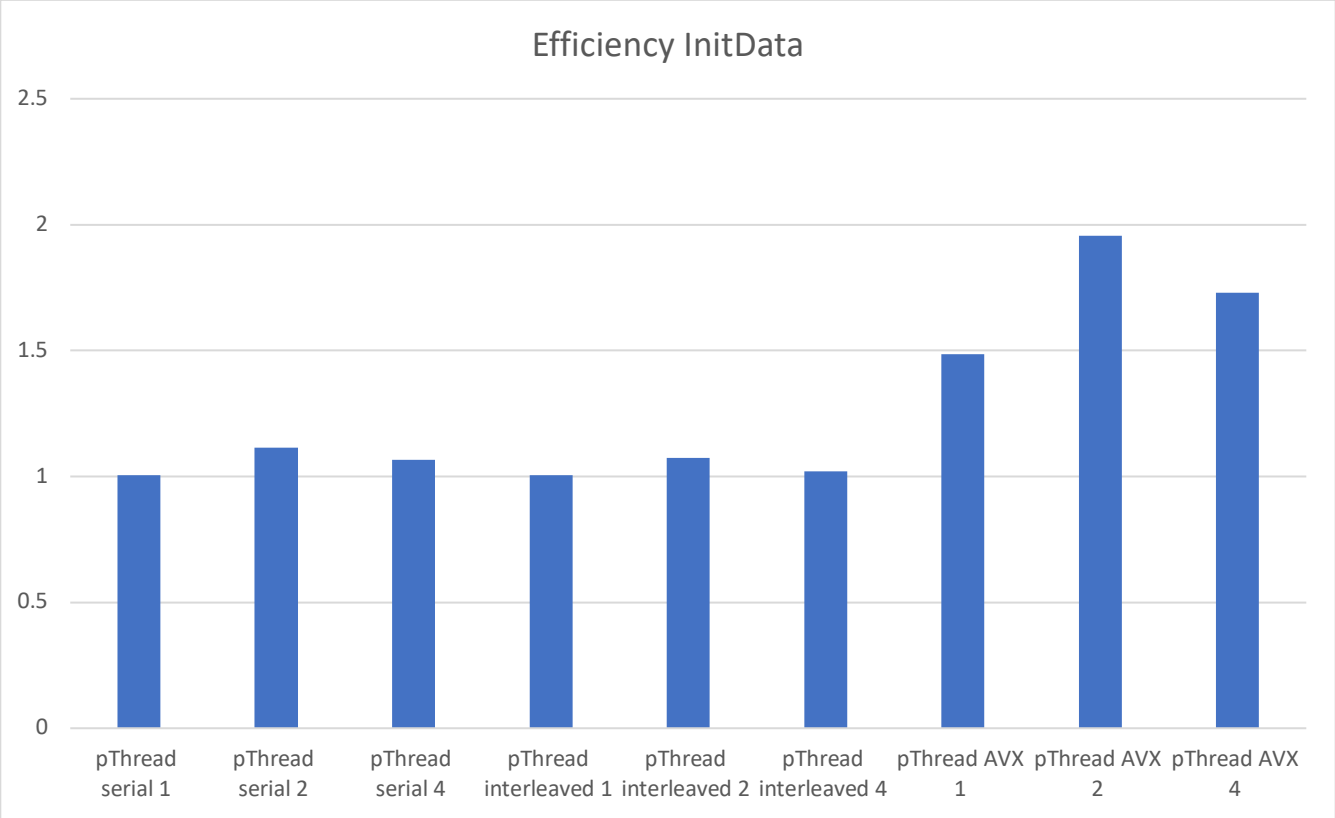
6) Αναμενόμενο speedup σε σχέση με το πραγματικό speedup(5%)



	Actual Speedup	Expected Speedup
pThread serial 1	1.00549954	1
pThread serial 2	2.01654412	1.81
pThread serial 4	3.27462687	3.07
pThread interleaved 1	1.00457875	1
pThread interleaved 2	1.94503546	1.81
pThread interleaved 4	3.13428571	3.07
pThread AVX 1	10.97	7.39
pThread AVX 2	16.621211	8.5
pThread AVX 4	15.8985	9.19

Παρατηρούμε ότι το actual και το expected speedup είναι παρόμοιο για όλες τις συναρτήσεις εκτός από τις συναρτήσεις που έχουν threads με AVX εντολές. Παρατηρούμε μεγάλη απόκλιση του actual από τον expected χρόνο για τα threads με AVX εντολές για το λόγο ότι το πόσοστο παραλληλοποίησης που έχουν οι συναρτήσεις πιθανόν να μην έχει υπολογιστεί σωστά. Για να υπολογίσουμε σωστά τον expected χρόνο, θα πρέπει να μελετήσουμε τις εντολές που βγάζει η assembly. Επίσης θα έπρεπε να συμπεριλάβουμε και άλλους παράγοντες που παίζουν ρόλο, όπως από πια μνήμη φορτώνονται τα δεδομένα (L1,L2,L3), cache misses. Το μεγαλύτερο speedup το δίνει η συνάρτηση 2 threads με AVX2 εντολές.

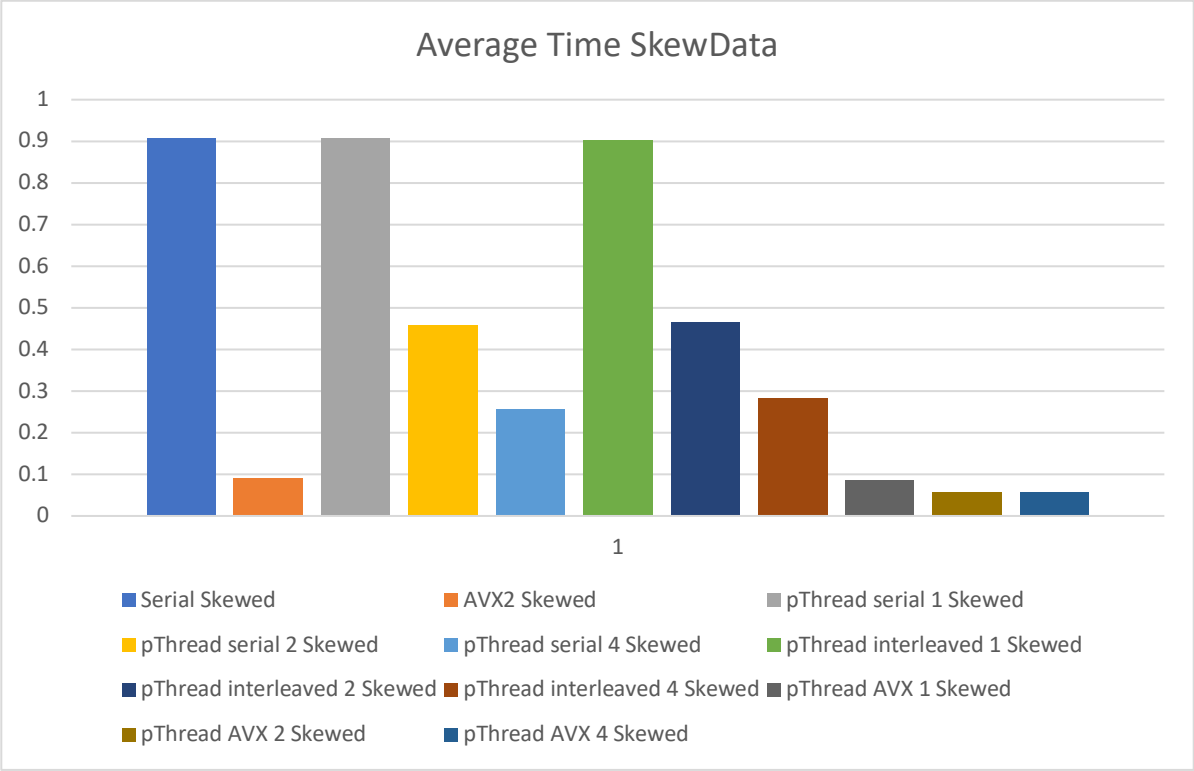
7) Efficiency. (5%)



	Efficiency
pThread serial 1	1.00549954
pThread serial 2	1.11411277
pThread serial 4	1.0666537
pThread interleaved 1	1.00457875
pThread interleaved 2	1.07460523
pThread interleaved 4	1.02093997
pThread AVX 1	1.48443843
pThread AVX 2	1.955437
pThread AVX 4	1.729984

Παρατηρούμε ξεκάθαρα ότι τα 2 threads με AVX2 εντολές δίνουν την μεγαλύτερη απόδοση. Όσο αφορά τα interleaved και τα consecutive threads, στα 2 threads παρατηρούμε μεγαλύτερη απόδοση, που όμως είναι χαμηλότερη από τα threads με AVX2 εντολές.

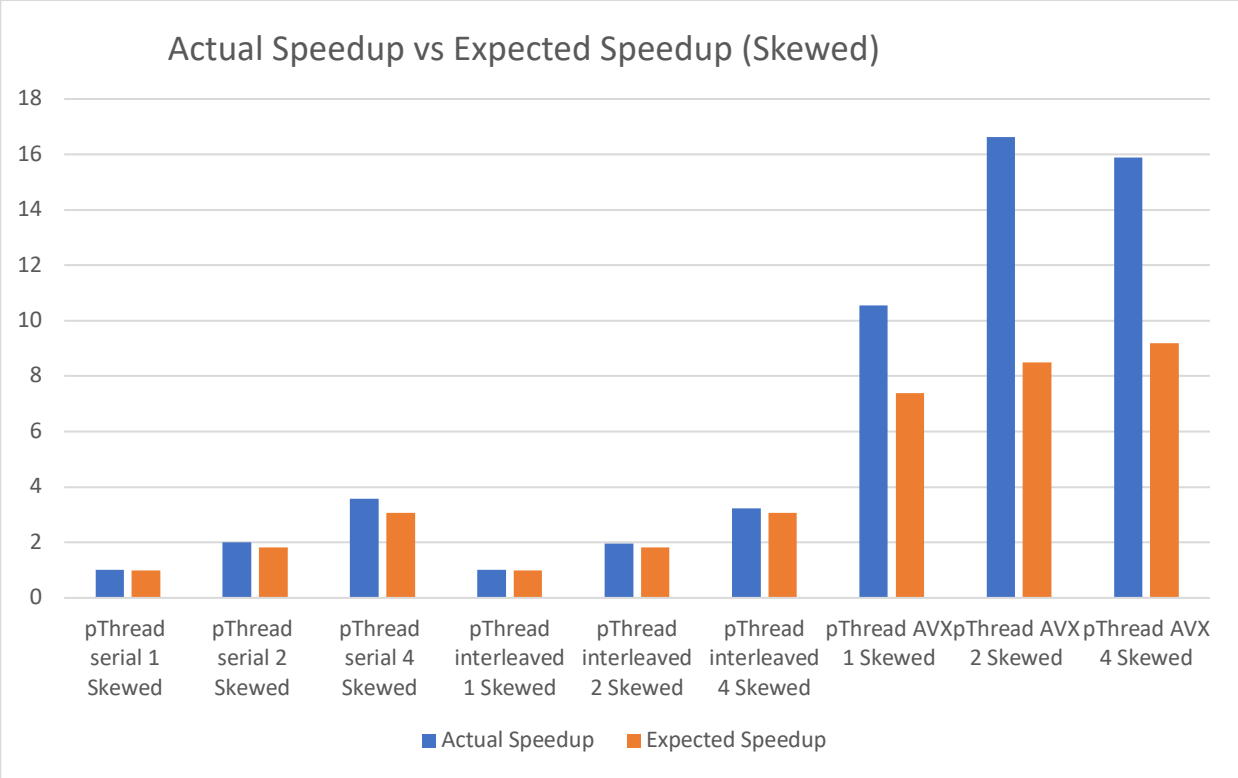
8) Skew Data for Q5,Q6,Q7



Παρατηρούμε ότι υψηλότερο χρόνο έχουν οι συναρτήσεις SERIAL, pthreadConsecutive με 1 thread και pthreadInterleaved με 1 thread. Τους χαμηλότερους χρόνους παρουσιάζουν τα threads με AVX2 εντολές, ωστόσο ο αριθμός των threads σε αυτή την περίπτωση δεν παρουσιάζει και μεγάλη διαφορά. Αντίθετα, στα threads που είναι consecutive και interleaved όσο περισσότερα threads έχουμε τόσο μικρότερος είναι ο χρόνος που χρειάζονται για να διεκπεραιώσουν τη δουλειά που τους ανατέθηκε. Παρατηρούμε ότι τα αποτελέσματα που πήραμε από τον Skew generator είναι όμοια με αυτά του InitData.

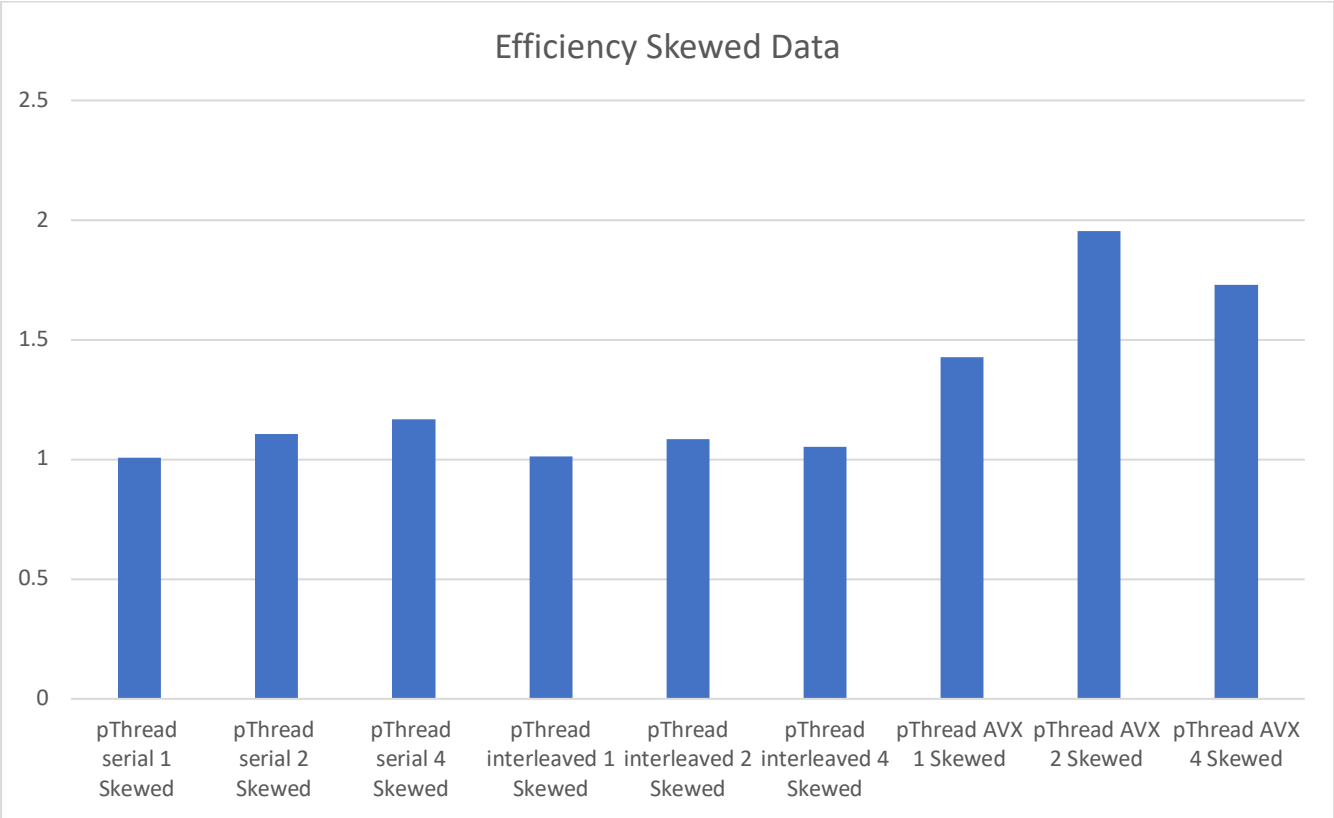
	Average Time
Serial Skewed	0.9075
AVX2 Skewed	**0.09**
pThread serial 1 Skewed	**0.90666667**
pThread serial 2 Skewed	**0.45666667**
pThread serial 4 Skewed	**0.255**
pThread interleaved 1 Skewed	**0.90166667**
pThread interleaved 2 Skewed	**0.465**
pThread interleaved 4 Skewed	**0.2825**
pThread AVX 1 Skewed	**0.08666667**
pThread AVX 2 Skewed	**0.055**

pThread AVX 4 Skewed	**0.0575**



	Actual Speedup	Expected Speedup
pThread serial 1 Skewed	1.00827206	1
pThread serial 2 Skewed	2.00182482	1.81
pThread serial 4 Skewed	3.58496732	3.07
pThread interleaved 1 Skewed	1.01386322	1
pThread interleaved 2 Skewed	1.96594982	1.81
pThread interleaved 4 Skewed	3.2359882	3.07
pThread AVX 1 Skewed	10.5480769	7.39
pThread AVX 2 Skewed	16.621211	8.5
pThread AVX 4 Skewed	17.898551	9.19

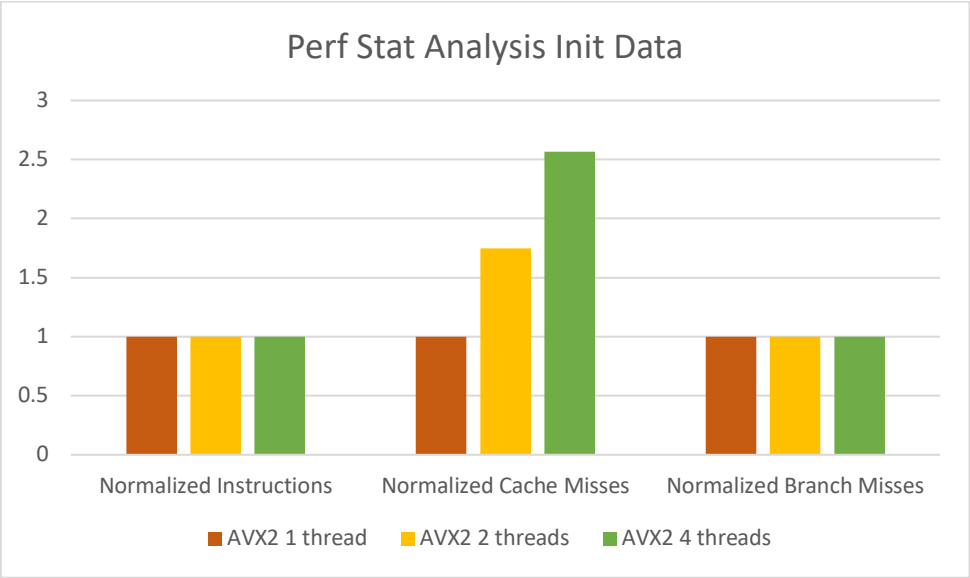
Παρατηρούμε ότι το actual και το expected speedup είναι παρόμοιο για όλες τις συναρτήσεις εκτός από τις συναρτήσεις που έχουν threads με AVX εντολές. Παρατηρούμε μεγάλη απόκλιση του actual από τον expected χρόνο για τα threads με AVX εντολές για το λόγο ότι το πόσοστο παραλληλοποίησης που έχουν οι συναρτήσεις πιθανόν να μην έχει υπολογιστεί σωστά. Για να υπολογίσουμε σωστά τον expected χρόνο, θα πρέπει να μελετήσουμε τις εντολές που βγάζει η assembly. Επίσης θα έπρεπε να συμπεριλάβουμε και άλλους παράγοντες που παίζουν ρόλο, όπως από πια μνήμη φορτώνονται τα δεδομένα (L1,L2,L3), cache misses. Το μεγαλύτερο speedup το δίνει η συνάρτηση 2 threads με AVX2 εντολές. Παρατηρούμε ότι τα αποτελέσματα που πήραμε από τον Skew generator είναι όμοια με αυτά του InitData.

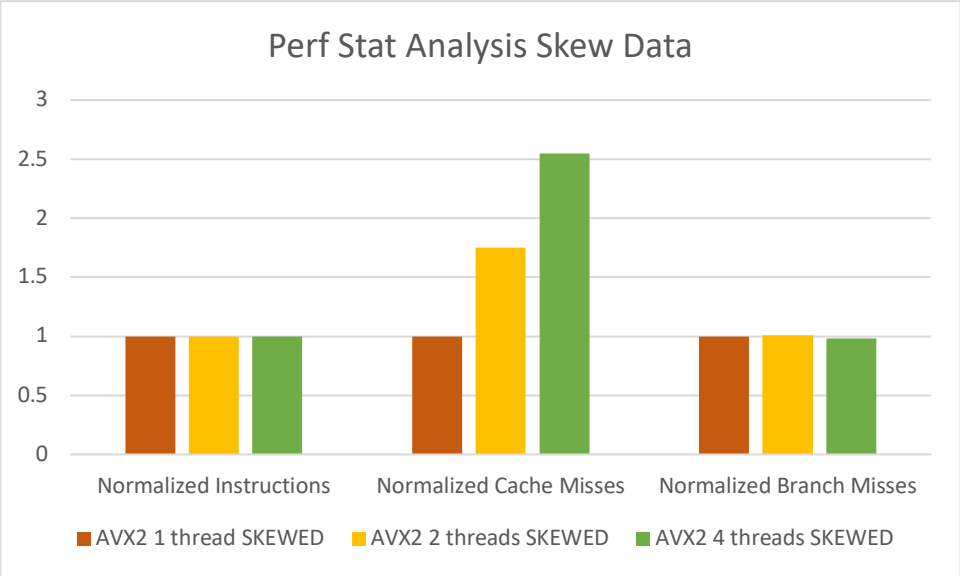


	Efficiency
pThread serial 1 Skewed	1.00827206
pThread serial 1 Skewed	1.10598056
pThread serial 1 Skewed	1.1677418
pThread interleaved 1 Skewed	1.01386322
pThread interleaved 2 Skewed	1.08616012
pThread interleaved 4 Skewed	1.05406782
pThread AVX 1 Skewed	1.42734464
pThread AVX 2 Skewed	1.955437
pThread AVX 4 Skewed	1.729984

Παρατηρούμε ξεκάθαρα ότι τα 2 threads με AVX2 εντολές δίνουν την μεγαλύτερη απόδοση. Όσο αφορά τα interleaved και τα consecutive threads, στα 2 threads παρατηρούμε μεγαλύτερη απόδοση, που όμως είναι χαμηλότερη από τα threads με AVX2 εντολές.

9)





Για τη δημιουργία των πιο πάνω γραφικών παραστάσεων έχουμε πάρει δεδομένα για Instructions,Cache Misses και Branch Misses. Έγινε στατιστική ανάλυση για την μέθοδο char_count_pthreads-AVX2 και για InitData και για Skewed Data. Για την άμεση σύγκριση των δεδομένων αφαιρέθηκε ο χρόνος που χρειάζεται το initData και ο χρόνος του skew έτσι ώστε να μπορούμε να δούμε τον πραγματικό χρόνο της κάθε συνάρτησης. Όπως βλέπουμε ο ρυθμός μεταβολής των Instructions και των Branch Misses παραμένει σταθερός. Ωστόσο ο ρυθμός μεταβολής των Cache Misses αυξάνεται σχεδόν ανάλογα με την αύξηση των threads κάθε μεθόδου.

10)

Από τον κώδικα βλέπουμε ότι το skewinit βάζει στο πρώτο μισό του πίνακα το στοιχείο που θέλουμε να προσμετρήσουμε και στο υπόλοιπο μισό βάζει random χαρακτήρες, ομοιόμορφα κατανεμημένους. Ενώ το initData αρχικοποιεί τον πίνακα με τυχαίους χαρακτήρες. Οπότε θεωρητικά τα threads στα skew δεδομένα θα έπρεπε να εκτελέσουν περισσότερα counts αφού πρέπει να εκτελέσουν μια πρόσθεση για κάθε στοιχείο τουλάχιστον μέχρι το μισό πίνακα. Ωστόσο, από τις πιο πάνω γραφικές παραστάσεις δεν παρατηρείται καμιά διαφορά μεταξύ skew και initData οπότε ο χρόνος εργασίας είναι ο ίδιος.

11) Από τις πιο πάνω γραφικές παραστάσεις του efficiency και του speedup παρατηρείται ότι σε κάθε συνάρτηση τα 2 threads δίνουν και καλύτερο speedup και καλύτερη απόδοση. Το ίδιο ισχύει είτε τα δεδομένα είναι skew είτε κατανεμημένα ομοιόμορφα με τυχαίους χαρακτήρες. Άρα καταλήγουμε στο συμπέρασμα ότι η καλύτερη επιλογή θα είναι τα 2 threads και σε speedup και σε efficiency γι'αυτό το πρόβλημα στις μηχανές των εργαστηρίων.

Οι μετρήσεις βρίσκονται σε ξεχωριστό αρχείο excel.