Department of Electrical and Computer Engineering
Technical University of Crete
Parallel and Distributed Computing Architecture
Instructor: N.Alachiotis
Christos Ziskas - Anastasios Mpokalidis
June 2, 2019
"Second Report"

# Παραλληλισμός με χρήση SIMD εντολών, MPI & PTHREADS

# 1 ΕΙΣΑΓΩΓΗ

Η άσκηση αυτή στοχεύει σε προχωρημένη μελέτη μεθόδων παραλληλοποίησης μέσω *Streaming SIMD Extensions (SSE), MPI & Pthreads*. Τα *SIMD*(Single Instruction, Multiple Data) instructions, κατασκευάστηκαν με στόχό την διευθέτηση παράλληλων διεργασιών. Σε αντίθεση με τις απλές *SISD* (Single Instruction, Single Data) που εκμεταλεύονται το concurrency, εκμεταλλέουνται το data level parallelism. Στον ίδιο χρόνο γίνεται επεξεργασία πολλαπλών δεδομένων μέσω διαφόρετικών processing units.
Το *MPI* αποτελεί πρωτόκολλο επικοινωνίας για τον προγραμματισμό παράλληλων υπολογιστών. Υποστηρίζονται τόσο η επικοινωνία point-to-point όσο και η collective επικοινωνία .
Τα *Pthreads* αποτελούν μοντέλο παράλληλης εκτέλεσης διαχειρίζοντας πολλαπλες ροές στο πρόγραμμα.

# 2 ΠΕΡΙΓΡΑΦΗ ΚΩΔΙΚΑ

## 2.1 REFERENCE

Η μελέτη του κώδικα σχετίζεται με τον υπολογισμό μιας απλοποιημένης μορφής ω statistic, με εφαρμογή στην ανίχνευση θετικής επιλογής σε ακολουθίες DNA.

Το ω υπολογίζεται ως:

$$num = \frac{L + R}{\left(\frac{m*(m-1)}{2.0}\right) + \left(\frac{n*(n-1)}{2.0}\right)}$$

$$den = \frac{C - L - R}{m * n}$$

$$\omega = \frac{num}{den + 0.01}$$

Οι παραπάνω υπολογισμοί επαναλαμβάνονται επαναληπτικά για ένα σύνολο N $DNA$ θέσεων, για τις οποίες μας ενδιαφέρει ο εντοπισμός της μέγιστης ω τιμής, της ελάχιστης ω τιμής, καθώς και της μέσης ω τιμής. Το N είναι μεταβλητή για την οποία ισχύει $N \geq 1$ . Όλα τα δεδομένα εισόδου είναι τοποθετημένα σε $arrays$ μήκους N (μεταβλητή του χρήστη).
Ο κώδικας έχει δεχθεί τροποποίηση όσον αφορά τις αποδεκτές τιμές που λαμβάνουν οι μεταβλητές L,G,R ώστε να συμπεριλαμβάνουν και τις τιμές γύρω απο το μηδέν. Οι τιμές των μεταβλητών αναπαρίστούνται σε hexadecimal representation καθώς λαμβάνουν το ανάλογο αναγνωριστικό -f .

## 2.2 SSE

Στην υλοποίηση αυτή δεσμεύονται νέες μεταβλητές και pointers τύπου ($\_\_m128$) ώστε να συνάδουν με το format για παραλληλοποίηση. Η υλοποίηση χρησιμοποιεί μεταβλητές vectorized.
Συγκεκριμένα:

**Δέσμευση vectors** με τιμές [1, 2, 0.01f] για την εκτέλεση των πράξεων του ω statistic.
**Δέσμευση vectors** που χρησιμοποιούνται για τις συγκρίσεις των max,min,avg.
**Δέσμευση pointers** με τους οποίους αποφεύγεται η χρήση load & store, πάνω στους πίνακες των δεδομένων και γίνεται εκμετάλλευση παράλληλων κομματίων στα δεδομένα.
**Δέσμευση μεταβλητών** flags που χρησιμοποιούνται ως μάσκες ώστε να επικυρωθούν οι ορθές τιμές των max,min,avg.

Αναλύωντας περισσότερο τη διαδικασία, τα δεδομένα μήκους N διαχωρίζονται σε ίσα κομμάτια. Η for διαρθώνεται σε ίσα κομμάτια εκτελώντας έτσι το 25% των επαναλήψεων σε σχέση με το reference code. Τα directives σε κάθε πράξη με vectors επεξεργάζονται 4 float μεταβλητές ταυτόχρονα. Οι πράξεις είναι όμοιες,γίνονται με την ίδια σειρά με την αρχική υλοποίηση καθώς για πράξεις με αριθμούς κινητής υποδιαστολής δεν ισχύει η προσεταιριστική ιδιότητα. Αποθηκεύεται σε μεταβλητή το αποτέλεσμα της σύγκρισης του κάθε παράλληλου τμήματος από τα δεδομένα με την max τιμή. Αν το αποτέλεσμα του Fvec είναι μεγαλύτερο της μέγιστης τιμής, το flag διατηρεί τιμές μονάδας για ολές τις τιμές του αντίστοιχου πεδίου. Αλλιώς φορτώνεται με μηδενικά. Ουσιαστικά απαρτίζει μια μάσκα ώστε είτε να εκχωρηθεί η τιμή του Fvec στο

max ή το max να διατηρηθεί ως έχει. Τα 4 αποτελέσματα συγκρίνονται οριζοντίως με το max vector . Αυτά τα μέγιστα θεωρούνται τοπικά μέγιστα των γραμμών. Με το πέρας της for έχουν υπολογιστεί όλες οι τιμές και έχουν πραγματοποιηθεί η εύρεση της μέγιστης τιμής(κατακόρυφα). Το ολικό μέγιστο αναζητείται οριζοντίως πλέον από τις 4 τιμές που είναι αποθηκευμένες στον καταχωρητή. Η συγκεκριμένη διαδικασία είναι χρονοβόρα. Ο χρόνος αυτός ελαττώνεται, μειώνωντας τις προσβάσεις στη μνήμη. Στον αντίποδα, οι πράξεις γίνονται πιο σύνθετες. Η μέγιστη τιμή αντιγράφεται σε μεταβλητή float ώστε να συνεχιστεί η εκτέλεση για το υπολοιπόμενο κομμάτι της εισόδου για τις περιπτώσεις που δεν κατανέμονται ισαρίθμα τα δεδομένα εισόδου. Ομοία διαδικασία πραγματοποιείται για το min & avg vector. Το αποτέλεσμα της μέσης τιμής προκύπτει από την άθροιση των τιμών που βρίσκονται στο συγκεκριμένο καταχωρητή.Τα δεδομένα εισόδου δεν δημιουργούν edge cases διότι η διαίρεση με το 4 γίνονται τέλεια. Το πρόγραμμα λειτουργεί ορθά για όλες τις πιθανές εισόδους καθώς λαμβάνεται υπόψη ότι η διαίρεση των δεδομένων με το 4 μπορεί να μην είναι τέλεια. Συνεχίζει από το επόμενο του στοιχείου που σταμάτησε η προηγούμενη for(), υπολογίζει και συγκρίνει σειριακά όλα τα υπόλοιπα, τα οποία είναι το πολύ τρία. Προτιμήθηκε αυτή η λύση αντί τουpadding καθώς είναι μικρή η επιρροή στο performance για το υπολειπόμενο κομμάτι.

## 2.3 SSE ενδιάμεσες υλοποιήσεις

Η υλοποίηση του SSE έχει προκύψει διανύοντας 3 στάδια. Συγκεκριμένα:

- Loop Unroll
- Jam
- Μετατροπή σε SSE με προσθήκη ακραίων περιπτώσεων

### 2.3.1 Loop Unroll

Παραθέτεται τμήμα της υλοποίησης

Η βασική ιδέα είναι ότι η σειρά των εντολών τοποθετείται με τέτοιο τρόπο, ώστε να μεταβάλλονται οι αντίστοιχες εντολές από κάθε block και να πραγματοποιούνται η μία μετά την άλλη. Αρχικά η πρώτη εντολή , συνεχίζει η δεύτερη εντολή από κάθε block, και η εκτέλεση ακολουθεί το ίδιο μοτίβο. Ο παράγοντας του unroll καθιστά τις 4 ίδιες εντολές ως νέο μπλοκ. Η διαδικασία βελτιώνει το pipelining του επεξεργαστή και επιταχύνει το πρόγραμμα. Αν οι πράξεις σε κάθε νέο block είναι ανεξάρτητες μεταξύ τους, ο κώδικας δυνητικά μπορεί να εκτελεστεί παράλληλα. Ο χρόνος εκτέλεσης είναι βελτιωμένος.

```
    for(unsigned int i=0;i<N;i+=4){
        num_1[0] = mVec[i]*(mVec[i]-1.0f)/2.0f;
        num_1[1] = mVec[i+1]*(mVec[i+1]-1.0f)/2.0f;
        num_1[2] = mVec[i+2]*(mVec[i+2]-1.0f)/2.0f;
        num_1[3] = mVec[i+3]*(mVec[i+3]-1.0f)/2.0f;
        num_2[0] = nVec[i]*(nVec[i]-1.0f)/2.0f;
        num_2[1] = nVec[i+1]*(nVec[i+1]-1.0f)/2.0f;
        num_2[2] = nVec[i+2]*(nVec[i+2]-1.0f)/2.0f;
        num_2[3] = nVec[i+3]*(nVec[i+3]-1.0f)/2.0f;
        num_0[0] = LVec[i]+RVec[i];
        num_0[1] = LVec[i+1]+RVec[i+1];
        num_0[2] = LVec[i+2]+RVec[i+2];
        num_0[3] = LVec[i+3]+RVec[i+3];
        num[0] = num_0[0]/(num_1[0]+num_2[0]);
        num[1] = num_0[1]/(num_1[1]+num_2[1]);
        num[2] = num_0[2]/(num_1[2]+num_2[2]);
        num[3] = num_0[3]/(num_1[3]+num_2[3]);
        den_1[0] = mVec[i]*nVec[i];
        den_1[1] = mVec[i+1]*nVec[i+1];
        den_1[2] = mVec[i+2]*nVec[i+2];
        den_1[3] = mVec[i+3]*nVec[i+3];
        den_0[0] = CVec[i]-LVec[i]-RVec[i];
        den_0[1] = CVec[i+1]-LVec[i+1]-RVec[i+1];
        den_0[2] = CVec[i+2]-LVec[i+2]-RVec[i+2];
        den_0[3] = CVec[i+3]-LVec[i+3]-RVec[i+3];
        den[0] = den_0[0]/den_1[0];
        den[1] = den_0[1]/den_1[1];
        den[2] = den_0[2]/den_1[2];
        den[3] = den_0[3]/den_1[3];
        FVec[i] = num[0]/(den[0]+0.01f);
        FVec[i+1] = num[1]/(den[1]+0.01f);
        FVec[i+2] = num[2]/(den[2]+0.01f);
        FVec[i+3] = num[3]/(den[3]+0.01f);
        minF = FVec[i]<minF?FVec[i]:minF;
        minF = FVec[i+1]<minF?FVec[i+1]:minF;
        minF = FVec[i+2]<minF?FVec[i+2]:minF;
        minF = FVec[i+3]<minF?FVec[i+3]:minF;
        maxF = FVec[i]>maxF?FVec[i]:maxF;
        maxF = FVec[i+1]>maxF?FVec[i+1]:maxF;
        maxF = FVec[i+2]>maxF?FVec[i+2]:maxF;
        maxF = FVec[i+3]>maxF?FVec[i+3]:maxF;
        avgF += FVec[i];
        avgF += FVec[i+1];
        avgF += FVec[i+2];
        avgF += FVec[i+3];
    }
}
```

### 2.3.2 Jam

Παραθέτεται τμήμα της υλοποίησης

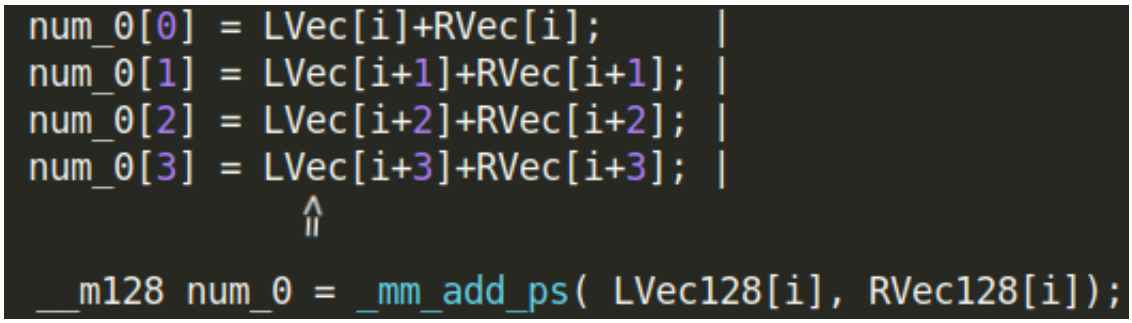
Οι νέες μεταβλητές έχουν μετατραπεί σε πίνακες με 4 floats. Η ροή του προγράμματος είναι παρόμοια με το σειριακό κώδικα για την εκτέλεση των υπολογισμών μέσω των νέων δεικτών. Εξαιτίας των εξαρτήσεων μεταξύ των πράξεων, η διαδικασία δεν επιδιώκει βελτίωση. Ο αριθμός των επαναλήψεων λιγοστεύει με όμοια αντιστοιχία με την οποία αντιπροσωπεύουν οι καταχωρητές στις νέες μεταβλητές. Δεν βελτιώνεται η ταχύτητα. Υπάρχει εκμετάλλευση των καταχωρητών του επεξεργαστή. Σε ακραίες περιπτώσεις όπου στις μεταβλητές υπάρχει μεγάλος αριθμός από καταχωρητές, ο επεξεργαστής μπορεί να είναι ανίκανος να προσφέρει του απαιτούμενους πόρους.

```
for(unsigned int i=0;i<N;i+=4){
    num_0[0] = LVec[i]+RVec[i];
    num_1[0] = mVec[i]*(mVec[i]-1.0)/2.0;
    num_2[0] = nVec[i]*(nVec[i]-1.0)/2.0;
    num[0] = num_0[0]/(num_1[0]+num_2[0]);
    den_0[0] = CVec[i]-LVec[i]-RVec[i];
    den_1[0] = mVec[i]*nVec[i];
    den[0] = den_0[0]/den_1[0];
    FVec[i] = num[0]/(den[0]+0.01f);
    maxF = FVec[i]>maxF?FVec[i]:maxF;
    minF = FVec[i]<minF?FVec[i]:minF;
    avgF+=FVec[i];
```

### 2.3.3 Τελική Έκδοση SSE

Η έκδοση του SSE έχει προκύψει ως ανάλυση των παραπάνω στοιχείων. Κάθε τετράδα εντολών μετατρέπεται σε μια εντολή όπως παρουσιάζεται στην εικόνα. Απαιτούνται καινούριες μεταβλητές vector όπου η κάθε μία περιέχει μια τετράδα floats. Η εντολές για την εκτέλεση των πράξεων ακολούθουν και αυτές vectorized πρότυπα. Οι εντολές λοιπόν, εκτελούνται παράλληλα και αυξάνεται το performance του προγράμματος. Ορισμένα ενδιάμεσα βήματα των υπολογισμών ( προσθεσεις,αφαιρέσεις στον αριθμητη/παρανομαστη) τοποθετούνται ως παράμετροι στις συναρτήσεις. Έτσι το πρόγραμμα χρησιμοποιεί λιγότερους πόρους στους υπολογισμούς, και αυτό αποτυπώνεται στην περαιτέρω βελτίωση

του χρόνου.




### 2.3.4 SSE & Pthreads

Η υλοποιήση με την χρήση Pthreads απαιτεί ως επιπρόσθετη είσοδο τον αριθμό των threads. Αναγκαία είναι η κατασκευή δυο struct με τις απαραίτητες πληροφορίες για κάθε thread.
Στο πρώτο struct εμπεριέχονται οι μεταβλητές

- Ταυτότητα του thread (ID)
- Συνολικός αριθμός threads
- Executed Operation
- Barrier
- Index των υπολογισμών που διαχειρίζονται

Στο δεύτερο struct εμπεριέχονται οι δείκτες του SSE μοντέλου που χρησιμοποιούνται στους υπολογισμούς, αλλά και οι μεταβλητές για τα avg,min,max. Εμπεριέχεται ως στοιχείο της πρώτης δομής.

Αρχικά το master thread κανονίζει τη δέσμευση μνήμης για τους πίνακες που θα χρησιμοποιηθούν για τους υπολογισμούς. Οι πίνακες αυτοί θα αποτελέσουν στοιχείο του κάθε thread. Γίνεται το initialization των μεταβλητών μέσα σε μία for() όπου το master thread δημιουργεί τα απαιτούμενα (τόσα όσα είναι και ο αριθμός των thread) structs. Τα νέα νήματα στέλνονται με την pthread_create() στη συνάρτηση threadFUNC που γίνονται

οι υπολογισμοί. Έπειτα τα threads βρίσκονται σε κατάσταση busywait και αναμένουν τα υπόλοιπα thread ($barrier = 1$). Ο master πριν ξεκινήσει πρέπει να έχει δημιουργήσει και να στείλει τα υπόλοιπα, οπότε δεν υπάρχει νόημα να δημιουργηθεί το struct του πρώτο. Για αυτόν τον λόγο η for() είναι ανάποδη, δηλαδή ξεκινάει απο το τελευταίο νήμα και τελειώνει με το πρώτο. Η ανάποδη υλοποίηση αποδείχτηκε ελαφρά πιο γρήγορη και επιλέχτηκε . Μέσα στα iterations, υπολογίζονται τα indexes που θα διαχειριστούν τα threads στους υπολογισμούς. Σύμφωνα με τα indexes αυτά, τα threads σε κάθε iteration θα διαχειριστούν διαφορετικά κομμάτια από τους πίνακες που βρίσκονται στο data struct και συνεπώς επιτυγχάνεται καλή διαχείριση τους. Πραγματοποιούνται οι υπολογισμοί απο τα worker threads σύμφωνα με το μοντέλο SSE που έχει χρησιμοποιηθεί. Μετα το πέρας του υπολογισμού τους αναμένουν τον συγχρονισμό τους. Οι ζητούμενες μεταβλητές max,min,avg υπολογίζονται ανατρέχοντας για κάθε thread , την εκάστοτε παράμετρο και ελέγχοντας την με τις αντίστοιχες τιμές των υπολοίπων thread. Για το average γίνεται πρόσθεση των τιμών. Αν υπάρχει υπόλοιπο στην κατάτμηση της εισόδου στα threads τότε ο υπολοιπόμενος υπολογισμός πραγματοποιείται από το master thread. Στο τέλος της συνάρτησης τα threads σκοτώνονται. Χρονομετρείται ολόκληρη η διαδικασία.

## 2.4 SSE & Pthreads & MPI

Το MPI παρέχει παραλληλισμό σε επίπεδο πράξεων. Μεταχειρίζεται το φόρτο στις διεργασίες. Στο περιβάλλον αυτό, αρχικά πραγματοποιούνται οι απαραίτητες αρχικοποιήσεις ώστε να επικοινωνούν οι διεργασίες. Σε MPI προγράμματα, χρησιμοποιείται η εντολή MPI_Init(). Η συνάρτηση αυτή επιτρέπει την επικοινωνία μεταξύ των processes και είναι η πρώτη συνάρτηση που εκτελείται. Κάθε διεργασία γνωρίζει το σύνολο των υπόλοιπων διεργασιών αλλα και την προτεραιότητα τους. Αυτό πραγματοποιείται με τις εντολές MPI_Comm_size & MPI_Comm_rank αντίστοιχα. Να σημειωθεί ότι στο struct των thread έχει συμπεριληφθεί η ταυτότητα της διεργασίας. Πλεον η κάθε διεργασία έχει τα δικά της threads και οι υπολογισμοί γίνονται παράλληλα. Ο αριθμός των iterations έχει ελαττωθεί στη μονάδα. Τα threads εκτελούν τους υπολογισμούς κάθε διεργασίας. Η μεθοδολογία που χρησιμοποιείται για τους υπολογισμούς μεσω thread δεν έχει υποστεί διαφοροποίηση.

Ορίζονται οι τύποι μεταβλητών με χρήση MPI_FLOAT_INT και το περι-

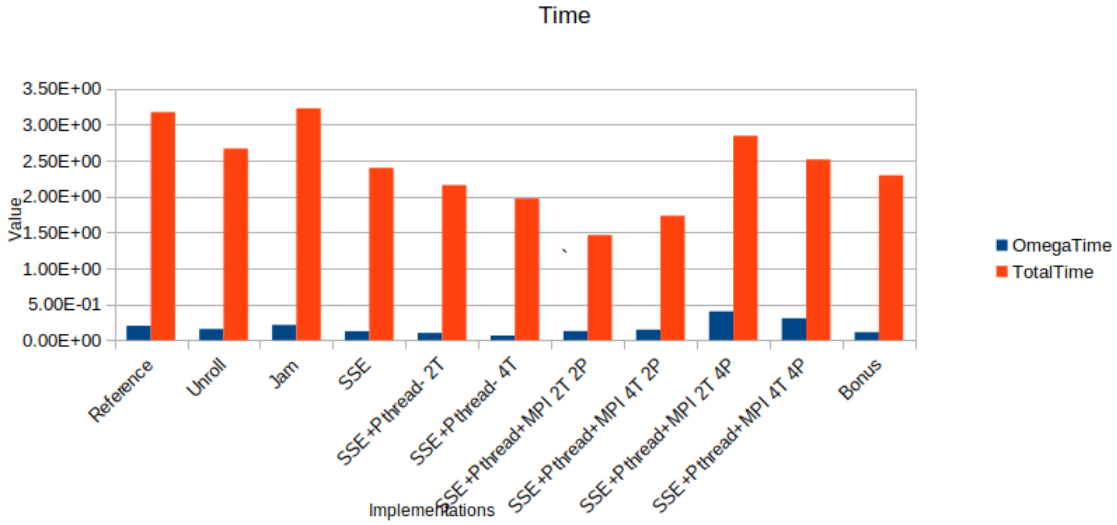
βάλλον των διεργασιών με χρήση MPI_COMM_WORLD. Η αποστολή των μυνημάτων γίνεται μια φορά πρίν την εκτύπωση του μέγιστου ώστε η επικοινωνία και η ανταλλαγή δεδομένων να είναι η ελάχιστη δυνατή. Πραγματοποιείται με την εντολή MPI_GATHER ώστε η κύρια διεργασία να επιτελέσει τις συγκρίσεις μεταξύ των αποτελεσμάτων των υπολοίπων διεργασιών καθώς και των δικών της. Τέλος η συνάρτηση MPI_Finalize() καλείται με σκοπό να καταστρέψει το περιβάλλον επικοινωνίας των διεργασιών. Δεν εγγυάται το τερματισμό τους παρα μόνο την συνέχεια της κύριας διεργασιας.

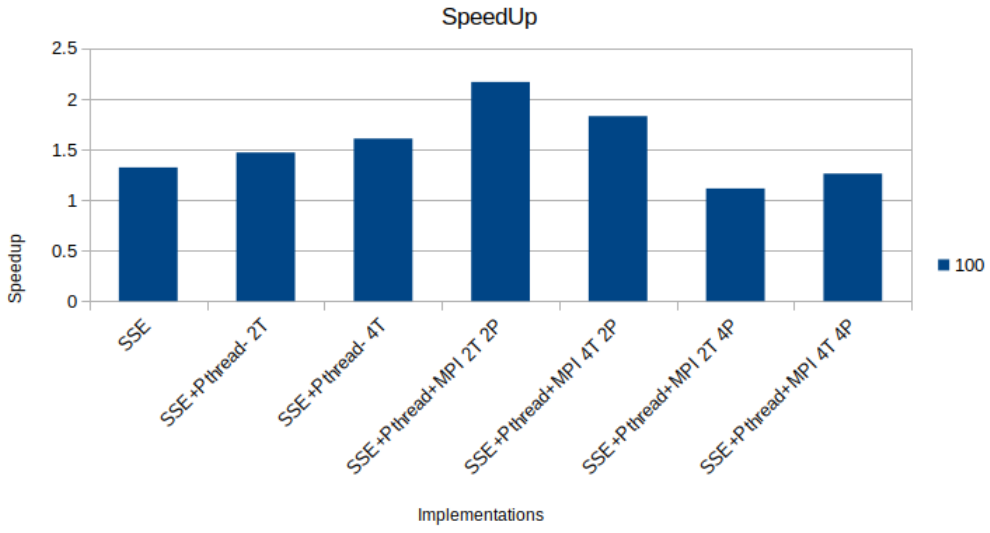
### 2.5 Bonus

Σε αυτό το σημείο παρουσιάζεται η εναλλακτική υλοποίηση με εκμετάλλευση διαφορετικού memory layout . Ο κώδικας για τις SSE εντολές έχει δεχθεί τροποποιήσεις. Διαπιστώθηκε ότι ο κώδικας θα μπορουσε να τροποποιηθεί εκτελώντας την μέθοδο του loop unroll ,που όπως διαπιστώθηκε από τον αρχικό κώδικα, προκαλεί βελτίωση της απόδοσης. Δεν υπάρχουν εξαρτήσεις με τους επόμενους υπολογισμούς οπότε η προτεινόμενη μέθοδος είναι εφικτή. Οι vectorized μεταβλητές έχουν τροποποιηθεί σε vectorized arrays 4 θέσεων. Τα δεδομένα εισόδου διαχωρίζονται σε πιο μικρά κομμάτια(αυτή τη φορά $\frac{N}{16}$) και σε κάθε iteration οι δείκτες επεξεργάζονται διαφορετικό κομμάτι δεδομένων( ο πρώτος δείκτης τα πρώτα $\frac{N}{16}$ , ο δεύτερος τα επόμενα κλπ). Οι ίδιες εντολές ομαδοποιούνται. Επομένως οι πίνακες πλέον διαχειρίζονται διαφορετικα κομμάτια υπολογισμών. Οι παράμετροι avg,min,max υπολογίζονται με όμοιο τρόπο μόνο.

## 3 Αποτελέσματα

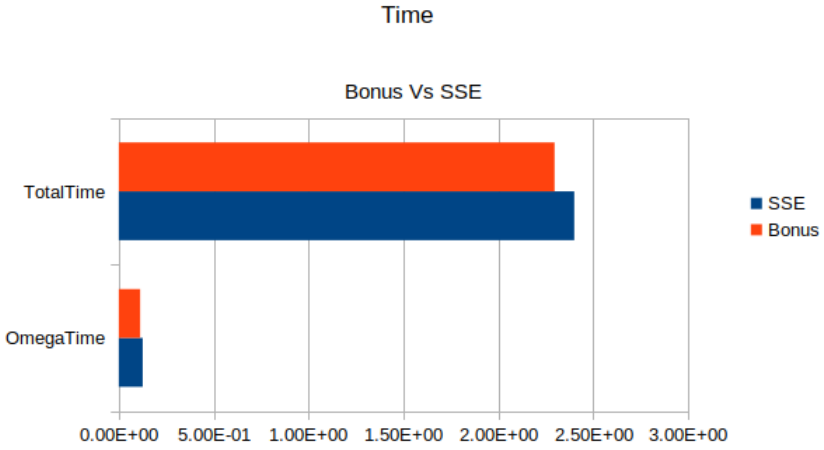
Έχουν πραγματοποιηθεί οι απαραίτητοι υπολογισμοί που αποδίδουν στην έκφραση του performance για τις διάφορες υλοποίησεις. Οι υπολογισμοί αυτοί αποτυπώνονται στα γραφήματα. Τα γραφήματα εστιάζουν στη διάρκεια των υπολογισμών καθώς και στη βελτίωση σε σχέση με την αρχική υλοποίηση. Η κάθε υλοποίηση δέχεται ως είσοδο τον ίδιο αριθμό δεδομένων N=10000000. Θεωρείται το πρώτο γράφημα, το γράφημα της χρονικής διάρκεια της κάθε υλοποίησης ενώ το δεύτερο διάγραμμα της έκθεσης του Speedup.

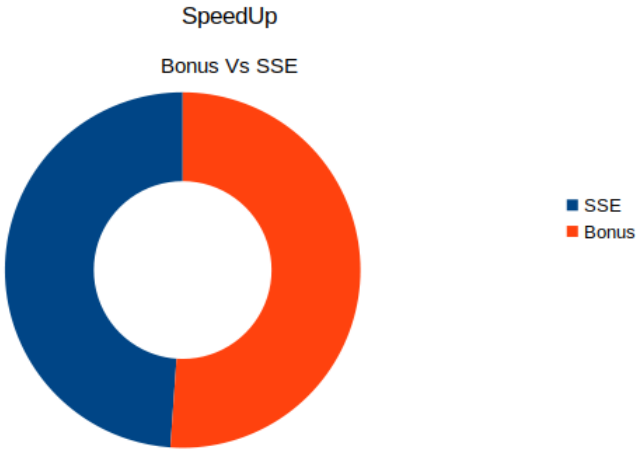



Επιπλέον, παρουσιάζεται ο πίνακας των αποτελεσμάτων από τα οποία εξήχθησαν τα παραπάνω γραφήματα.

| N = 10.000.000 | | |
|---|---|---|
| | Omega time | Total Time |
| Reference Code(RC) | 0.19963s | 3.174151s |
| RC with Jammingl | 0.21263s | 3.227579s |
| RC with Loop Unroll | 0.155849s | 2.668166s |
| SSE | 0.126033s | 2.398806s |
| SSE + 2 threads | 0.101284s | 2.158511s |
| SSE + 4 threads | 0.06446s | 1.973034s |
| SSE + 2 threads 2 procs | 0.127441s | 1.46363s |
| SSE + 2 threads 4 procs | 0.400856s | 2.845927s |
| SSE + 4 threads 2 procs | 0.146648s | 1.734404s |
| SSE + 4 threads 4 procs | 0.305505s | 2.515387s |
| Bonus | 0.112597 | 2.295190s |






Σύμφωνα με τα συμπεράσματα που εξάγονται από τα γραφήματα , σημειώνονται τα εξής:

1. Παρουσιάζεται βελτίωση από υλοποίηση σε υλοποίηση αναφορικά με την πρωτότυπη για κάθε τύπο παραλληλοποίησης.

2. Η χρήση καταχωρητών με το μοντέλο SSE είναι αποδοτικό, καθώς οι μεταβλητές είναι ανεξάρτητες, οπότε δίνει μια εύκολη λύση για την επιτάχυνση του προγράμματος.

3. Ο συνδιασμός των δυο μεθόδων παραλληλοποίησης (SSE & Pthread) είναι σαφώς καλύτερος από την παραλληλοποίηση με SSE intrinsics. Η βελτίωση μεγαλωνει με την αύξηση του αριθμού των threads. Ο φόρτος ελαττώνεται με την αύξηση τους και υπάρχει καλύτερη εκμετάλλευση της παραλληλοποίησης.

4. Ο συνδιασμός των υλοποιήσεων δεν αποδίδει στη μέγιστη βελτίωση όπως διαισθητικά θα έπρεπε. Για μικρό αριθμό διεργασιών ανεξάρτητα του αριθμού των νημάτων, υπάρχει άριστη βελτίωση. Για αύξηση του αριθμού των διεργασιών, αυξάνεται το κόστος overhead η δέσμευση πόρων , καθώς και η επικοινωνία μεταξύ τους. Αυτό αποδίδεται στους χρόνους εκτέλεσης και στην απόδοση.

5. Η προεραιτική υλοποίηση παρουσιάζει ελαφριά βελτίωση σε σχέση με την απλή υλοποίηση με εντολές SSE, λόγω καλύτερης εκμετάλλευσης των καταχωρητών καθώς και των εξαρτήσεων που παρουσιάζουν οι εντολές μεταξύ τους.

Ως γενικό συμπέρασμα, το speedup του SSE επηρεάζεται από την φύση των υπολογισμών. Προσθέτωντας νήματα στην υλοποίηση, η απόδοση επηρεάζεται με την υπερβολική εκμετάλλευση των πόρων και την σωστή κατανομή του φόρτου σε αυτά. Αντίθετα στο MPI είναι ανεξάρτητο από τις πράξεις, όμως επηρεάζεται από το μέγεθος των δεδομένων, την ανάγκη επικοινωνίας και τους πόρους που του διαθέτονται.

# 4 Συμπεράσματα & Παρατηρήσεις

Το SSE αποτελεί αξιόπιστο και αποδοτικό πρότυπο παραλληλοποίησης για ορθή και δομημένη υλοποίηση. Προσφέρει πληθόρα εντολών η οποία επεκτείνεται στους νέους επεξεργαστές. Λόγω του ακριβή και εκτεταμένου documentation, το API αποδίδει τα αναμενόμενα αποτελέσματα. Μειονέκτημά του είναι η αδυναμία εκτέλεσης αποδοτικών πράξεων μεταξύ των στοιχείων ενός vector, κάτι που αποφεύγεται ή μειώνεται με καλύτερη δόμη στον κώδικα. Τέλος το speedup έχει ανώτατο όριο πόσα δεδομένα χωράνε ταυτόχρονα στο pipeline του επεξεργαστή.

Το MPI προσφέρει μείωση του χρόνου λειτουργίας.Εξαρτάται όμως από πολλούς παράγοντες όπως αναφέρθηκε στα αποτελέσματα των speedups. Με τη βοήθεια λογισμικού όπως το LAM, το MPI έχει προοπτικές για βελτίωση τουspeedup, γιατί ενδέχεται να συγχρονίσει παραπάνω από έναν υπολογιστή και να δημιουργήσει ένα cluster επεξεργαστών. Υπάρχουν έτοιμες συναρτήσεις όπως η MPI_GATHER() που προσφέρουν μεγαλύτερη ταχύτητα και βελτιωμένη λειτουργικότητα. Τέλος ένα μειωνέκτημα του είναι η αβεβαιότητα για το

πως επιδρούν ορισμένες συναρτήσεις στον κώδικα. Λόγου χάρη, δε γίνεται να είναι γνωστό ακριβώς ποιες διεργασίες συνεχίζουν μετά το MPI_Finalize().

Τα Pthreads δεν έχουν αυτοματοποιημένη λειτουργικότητα. Χρονοβόρα στη δημιουργία της υλοποίησης και στο συγχρονισμό με ευθύνη του δημιουργού την διασφαλίση της επικοινωνίας μεταξύ των thread . Τα οφέλη τους είναι η δυνατότητα αλλαγής και βελτίωσης χαρακτηριστικών, καθώς και γνώση της ακριβούς λειτουργικότητας του κώδικα.

Η παραλληλοποίηση μέσω των Pthreads με εντολές SSE είναι αισθητή και αναμενόμενη στη βελτίωση του speedup. Αντίθετα ο συνδιασμός SSE , Pthreads & MPI καθώς διευσθητικά βελτίωνει ακομά περισσότερο σε σχέση με SSE & Pthread η βελτίωση δεν είναι η αναμενόμενη. Η περαιτέρω δέσμευση των πόρων είναι αυτή που εμποδίζει αυτην την βελτίωση.

Η διαισθητική προσέγγιση για την απόδοση της παραλληλοποίηση μέσω των Pthreads & MPI & SSE θα μπορουσε να εκφραστεί ως πολλαπλή βελτίωση καθώς η μεμονομένη υλοποίηση τους βελτιώνει την αρχική υλοποίηση. Θεματα συγχρονισμού και overhead αποφέρουν χειρότερα αποτελέσματα στην απόδοση τελικά. Στην κατάσταση της απόδοσης να προστεθεί βέβαια και το μέγεθος των δεδομενων (στη συγκεκριμένη περίπτωση είναι μεγάλος όγκος όπου χειροτερεύει το communication)

Εν κατακλείδι, το SSE, και γενικά τα SIMD πρότυπα, είναι ανεξάρτητα από το μεγέθος των δεδομένων, όμως θέτεται ανώτατο όριο στο speedup από τον επεξεργαστή και το πάχος του pipeline . Αντίθετα, το MPI αντιμετωπίζει δυσκολίες ανάλογα τον όγκο δεδομένων αλλά έχει υψηλό scalability. Το ανώτατο φράγμα του speedup ορίζεται από τους πόρους που του διαθέτονται.