Εργασία 4

Ονοματεπώνυμο: Νικόλαος Θεοδώρου
Αριθμός Ταυτότητας: 1030496

Ερώτημα:

0) Προεργασία τροποποίηση memory alignment:

```
masses = (double *)aligned_alloc(64, bodies * sizeof(double)); // allocate aligned memory
positions = (vector *)aligned_alloc(64, bodies * sizeof(vector));
velocities = (vector *)aligned_alloc(64, bodies * sizeof(vector));
accelerations = (vector *)aligned_alloc(64, bodies * sizeof(vector));
```

Τροποποίησα το Alignment της μνήμης για να γίνεται σωστά το φόρτωμα των τιμών των πινάκων στα διανύσματα __mm512d.

***Επίσης το compilation έγινε με flags -O3 και -mavx512f, που δίνει τη δυνατότητα στο compiler να τρέξει το σειριακό κομμάτι του κώδικα με SIMD εντολές και έτσι να το κάνει πιο γρήγορο. Συνεπώς ο χρόνος εκτέλεσης του σειριακού κώδικα από 5 δευτερόλεπτα που ήταν στο OMP, τώρα μειώνεται στα 1.8 δευτερόλεπτα.**

1) Συνάρτηση `nBody_OMP_SIMD_static` (20%Κώδικας+5%Σχόλια) (`CourierNew10`):

```
void n_body_omp_static(int threads)
{
    SimulationTime++;
    omp_set_num_threads(threads);
    char *execution_type = "static";

    computeAccelerations(execution_type);
    computePositions(execution_type);
    computeVelocities(execution_type);
    resolveCollisions(execution_type);
}
```

```
void computeAccelerations(char *exec_type)
{
    int i, j;
    __m512d s_ieqj = _mm512_setzero_pd();
    __m512d GC = _mm512_set1_pd(GravConstant);

    if (strcmp(exec_type, "static") == 0)
    {
#pragma omp parallel private(i, j) shared(accelerations, positions, masses, GravConstant, threads, bodies, s_ieqj, GC) default(none)
        {
#pragma omp for schedule(static, bodies / threads)
            for (i = 0; i < bodies; i++)
            {
                __m512d i_indexes =_mm512_set1_pd((double)i);
                __m512d accxi = _mm512_setzero_pd();
```

```
            __m512d accyi = _mm512_setzero_pd();
            __m512d acczi = _mm512_setzero_pd();
            __m512d posxi = _mm512_set1_pd(positions[i].x);
            __m512d posyi = _mm512_set1_pd(positions[i].y);
            __m512d poszi = _mm512_set1_pd(positions[i].z);

            for (j = 0; j < bodies; j += 8)
            {

                __m512d mass = _mm512_load_pd(&masses[j]);

                __m512d posxj = _mm512_setr_pd(positions[j].x, positions[j + 1].x, positions[j + 2].x, positions[j + 3].x, positions[j + 4].x, positions[j + 5].x, positions[j + 6].x, positions[j + 7].x);
                __m512d posyj = _mm512_setr_pd(positions[j].y, positions[j + 1].y, positions[j + 2].y, positions[j + 3].y, positions[j + 4].y, positions[j + 5].y, positions[j + 6].y, positions[j + 7].y);
                __m512d poszj = _mm512_setr_pd(positions[j].z, positions[j + 1].z, positions[j + 2].z, positions[j + 3].z, positions[j + 4].z, positions[j + 5].z, positions[j + 6].z, positions[j + 7].z);

                __m512d sijx = _mm512_sub_pd(posxi, posxj);
                __m512d sijy = _mm512_sub_pd(posyi, posyj);
                __m512d sijz = _mm512_sub_pd(poszi, poszj);

                __m512d mod = _mm512_sqrt_pd(_mm512_fmadd_pd(sijx, sijx, _mm512_fmadd_pd(sijy, sijy, _mm512_mul_pd(sijz, sijz))));
                __m512d mod2 = _mm512_mul_pd(mod, mod);
                __m512d mod3 = _mm512_mul_pd(mod2, mod);



                __m512d j_indexes = _mm512_setr_pd((double)j,(double)j+1,(double)j+2,(double)j+3,(double)j+4,(double)j+5,(double)j+6,(double)j+7);
                __mmask8 mask = _mm512_cmp_pd_mask(i_indexes, j_indexes, _CMP_EQ_OQ);
                __m512d s_inotj = _mm512_div_pd(_mm512_mul_pd(GC, mass), mod3);
                __m512d s = _mm512_mask_blend_pd(mask, s_inotj, s_ieqj);

                __m512d Sx = _mm512_mul_pd(s, _mm512_mul_pd(sijx, _mm512_set1_pd(-1)));
                __m512d Sy = _mm512_mul_pd(s, _mm512_mul_pd(sijy, _mm512_set1_pd(-1)));
                __m512d Sz = _mm512_mul_pd(s, _mm512_mul_pd(sijz, _mm512_set1_pd(-1)));


                accxi = _mm512_add_pd(accxi, Sx);
                accyi = _mm512_add_pd(accyi, Sy);
                acczi = _mm512_add_pd(acczi, Sz);
              }

            accelerations[i].x = _mm512_reduce_add_pd(accxi);
            accelerations[i].y = _mm512_reduce_add_pd(accyi);
            accelerations[i].z = _mm512_reduce_add_pd(acczi);
        }
      }
    }
```

Η n_body_omp_static δέχεται ως argument των αριθμό των threads που δόθηκαν από το command line και θέτει τη ρύθμηση του openmp για το με πόσα threads να τρέξει. Στη συνέχει παραλληλοποιήσα τη Compute Accelerations, αφού είναι η πιο χρονοβόρα μέθοδος του προγράμματος. Η Compute Accelerations δέχεται ως argument τον τύπο παράλληλης εκτέλεσης σε αυτή τη περίπτωση είναι static, άρα θα εκτελέσουμε το branch του static scheduling. Το κάθε thread εκτελεί bodies/threads επαναλήψεις για να υπολογήσει τις επιταχύνσεις των σωματηδίων. Ο δεδομένος κώδικας χρησιμοποιεί εντολές Single Instruction Multiple Data (SIMD) για την εκτέλεση πολλαπλών υπολογισμών παράλληλα χρησιμοποιώντας διανυσματικούς καταχωρητές. Συγκεκριμένα, χρησιμοποιεί τις οδηγίες AVX-512, οι οποίες μπορούν να λειτουργήσουν σε διανυσματικούς καταχωρητές 512-bit που μπορούν να κρατήσουν 8 τιμές κινητής υποδιαστολής διπλής ακρίβειας. Αυτό επιτρέπει μια σημαντική αύξηση της απόδοσης σε σύγκριση με τις βαθμωτές λειτουργίες, καθώς πολλαπλοί υπολογισμοί μπορούν να εκτελεστούν παράλληλα σε μία μόνο εντολή. Ο κώδικας χρησιμοποιεί τις οδηγίες SIMD για να υπολογίσει την επιτάχυνση κάθε σώματος λόγω βαρυτικών δυνάμεων, η οποία περιλαμβάνει μεγάλο αριθμό υπολογισμών. Χρησιμοποιώντας οδηγίες SIMD, το πρόγραμμα μπορεί να μειώσει σημαντικά τον χρόνο που απαιτείται για την εκτέλεση αυτών των υπολογισμών, κάτι που μπορεί να είναι κρίσιμο για την επίτευξη υπολογιστών υψηλής απόδοσης. Συνοπτικά, η χρήση εντολών SIMD είναι ένας αποτελεσματικός τρόπος για να αυξηθεί η αποτελεσματικότητα του προγράμματος, ιδιαίτερα όταν έχουμε να κάνουμε με μεγάλα σύνολα δεδομένων, όπως η περίπτωση στον συγκεκριμένο κώδικα.

2) Συνάρτηση `nBody_OMP_SIMD_dynamic` (10%Κώδικας+5%Σχόλια) (CourierNew10):

```
void n_body_omp_dynamic(int threads)
{
   SimulationTime++;
   omp_set_num_threads(threads);
   char *execution_type = "dynamic";

   computeAccelerations(execution_type);
   computePositions(execution_type);
   computeVelocities(execution_type);
   resolveCollisions(execution_type);
}
void computeAccelerations(char *exec_type)
{
   int i, j;
   __m512d s_ieqj = _mm512_setzero_pd();
   __m512d GC = _mm512_set1_pd(GravConstant);
if (strcmp(exec_type, "dynamic") == 0)
   {
#pragma omp parallel private(i, j) default(none) shared(accelerations, positions, masses, GravConstant, bodies, s_ieqj, GC,  threads)
      {
         int chunksize = (threads > 1) ? (int)(bodies / threads * log(threads)) : bodies;
#pragma omp for schedule(dynamic, chunksize)
         for (i = 0; i < bodies; i++)
         {
            __m512d i_indexes =_mm512_set1_pd((double)i);
            __m512d accxi = _mm512_setzero_pd();
            __m512d accyi = _mm512_setzero_pd();
            __m512d acczi = _mm512_setzero_pd();
```

```
            __m512d posxi = _mm512_set1_pd(positions[i].x);
            __m512d posyi = _mm512_set1_pd(positions[i].y);
            __m512d poszi = _mm512_set1_pd(positions[i].z);

            for (j = 0; j < bodies; j += 8)
            {

                __m512d mass = _mm512_load_pd(&masses[j]);

                __m512d posxj = _mm512_setr_pd(positions[j].x, positions[j + 1].x, positions[j + 2].x, positions[j + 3].x, positions[j + 4].x, positions[j + 5].x, positions[j + 6].x, positions[j + 7].x);
                __m512d posyj = _mm512_setr_pd(positions[j].y, positions[j + 1].y, positions[j + 2].y, positions[j + 3].y, positions[j + 4].y, positions[j + 5].y, positions[j + 6].y, positions[j + 7].y);
                __m512d poszj = _mm512_setr_pd(positions[j].z, positions[j + 1].z, positions[j + 2].z, positions[j + 3].z, positions[j + 4].z, positions[j + 5].z, positions[j + 6].z, positions[j + 7].z);

                __m512d sijx = _mm512_sub_pd(posxi, posxj);
                __m512d sijy = _mm512_sub_pd(posyi, posyj);
                __m512d sijz = _mm512_sub_pd(poszi, poszj);

                __m512d mod = _mm512_sqrt_pd(_mm512_fmadd_pd(sijx, sijx, _mm512_fmadd_pd(sijy, sijy, _mm512_mul_pd(sijz, sijz))));
                __m512d mod2 = _mm512_mul_pd(mod, mod);
                __m512d mod3 = _mm512_mul_pd(mod2, mod);



                __m512d j_indexes = _mm512_setr_pd((double)j,(double)j+1,(double)j+2,(double)j+3,(double)j+4,(double)j+5,(double)j+6,(double)j+7);
                __mmask8 mask = _mm512_cmp_pd_mask(i_indexes, j_indexes, _CMP_EQ_OQ);
                __m512d s_inotj = _mm512_div_pd(_mm512_mul_pd(GC, mass), mod3);
                __m512d s = _mm512_mask_blend_pd(mask, s_inotj, s_ieqj);

                __m512d Sx = _mm512_mul_pd(s, _mm512_mul_pd(sijx, _mm512_set1_pd(-1)));
                __m512d Sy = _mm512_mul_pd(s, _mm512_mul_pd(sijy, _mm512_set1_pd(-1)));
                __m512d Sz = _mm512_mul_pd(s, _mm512_mul_pd(sijz, _mm512_set1_pd(-1)));


                accxi = _mm512_add_pd(accxi, Sx);
                accyi = _mm512_add_pd(accyi, Sy);
                acczi = _mm512_add_pd(acczi, Sz);
            }

            accelerations[i].x = _mm512_reduce_add_pd(accxi);
            accelerations[i].y = _mm512_reduce_add_pd(accyi);
            accelerations[i].z = _mm512_reduce_add_pd(acczi);
        }
    }
}
```

Η n_body_omp_dynamic δέχεται ως argument τον αριθμό των threads που δόθηκαν από το command line και θέτει τη ρύθμιση του openmp για το πόσα threads να τρέξει. Στη

συνέχεια παραλληλοποιούμε την Compute Accelerations, αυτή τη φορά με dynamic scheduling. Η Compute Accelerations δέχεται ως argument τον τύπο παράλληλης εκτέλεσης, σε αυτήν τη περίπτωση είναι dynamic. Εκτελούμε το branch του dynamic scheduling, το οποίο χρησιμοποιεί μια δυναμική ουρά για να διανείμει τις επαναλήψεις στα threads. Αντί για να διανέμονται ισόποσα στα threads, κάθε thread αναλαμβάνει ένα μικρότερο αριθμό επαναλήψεων. Αυτό μπορεί να είναι χρήσιμο όταν υπάρχει μεγάλος αριθμός επαναλήψεων και το σύστημα μπορεί να διαχειριστεί καλύτερα την κατανομή τους στα threads. Η κατανομή γίνεται με τύπο bodies/threads*log(threads) που είναι rule of thumb για dynamic scheduling workload distribution. Η παραλληλοποίηση γίνεται χρησιμοποιώντας το OpenMP και ο βρόχος χωρίζεται στα διαθέσιμα νήματα. Μέσα στον βρόχο, χρησιμοποιούνται οδηγίες SIMD για την παράλληλη εκτέλεση των υπολογισμών σε πολλαπλά στοιχεία δεδομένων ταυτόχρονα. Συγκεκριμένα, χρησιμοποιείται το σετ εντολών AVX-512, το οποίο επιτρέπει καταχωρητές 512 bit που μπορούν να χωρέσουν οκτώ τιμές κινητής υποδιαστολής διπλής ακρίβειας.

Ο βρόχος περιέχει δύο ένθετους βρόχους, όπου ο εσωτερικός βρόχος υπολογίζει τις δυνάμεις βαρύτητας μεταξύ ενός ουράνιου σώματος και μιας ομάδας οκτώ άλλων ουράνιων σωμάτων χρησιμοποιώντας οδηγίες SIMD. Κάθε ουράνιο σώμα έχει μια θέση και μια μάζα και οι δυνάμεις μεταξύ τους υπολογίζονται χρησιμοποιώντας το νόμο της βαρύτητας του Νεύτωνα. Οι δυνάμεις συσσωρεύονται σε τρία διανύσματα (ένα για κάθε χωρική διάσταση) χρησιμοποιώντας οδηγίες προσθήκης SIMD. Τέλος, η προκύπτουσα επιτάχυνση υπολογίζεται για κάθε ουράνιο σώμα διαιρώντας τη συνολική δύναμη με τη μάζα του και ενημερώνοντας ανάλογα τη θέση και την ταχύτητά του.

2.5) Συνάρτηση nBody_OMP_SIMD_guided:

```
void n_body_omp_guided(int threads)
{
   SimulationTime++;
   omp_set_num_threads(threads);
   char *execution_type = "guided";

   computeAccelerations(execution_type);
   computePositions(execution_type);
   computeVelocities(execution_type);
   resolveCollisions(execution_type);
}
void computeAccelerations(char *exec_type)
{
   int i, j;
   __m512d s_ieqj = _mm512_setzero_pd();
   __m512d GC = _mm512_set1_pd(GravConstant);

if (strcmp(exec_type, "guided") == 0)
   {
#pragma omp parallel private(i, j) default(none) shared(accelerations, positions, masses, GravConstant, bodies, s_ieqj, GC, threads)
      {
#pragma omp for schedule(guided, bodies/threads)
         for (i = 0; i < bodies; i++)
         {
            __m512d i_indexes =_mm512_set1_pd((double)i);
            __m512d accxi = _mm512_setzero_pd();
            __m512d accyi = _mm512_setzero_pd();
            __m512d acczi = _mm512_setzero_pd();
            __m512d posxi = _mm512_set1_pd(positions[i].x);
```

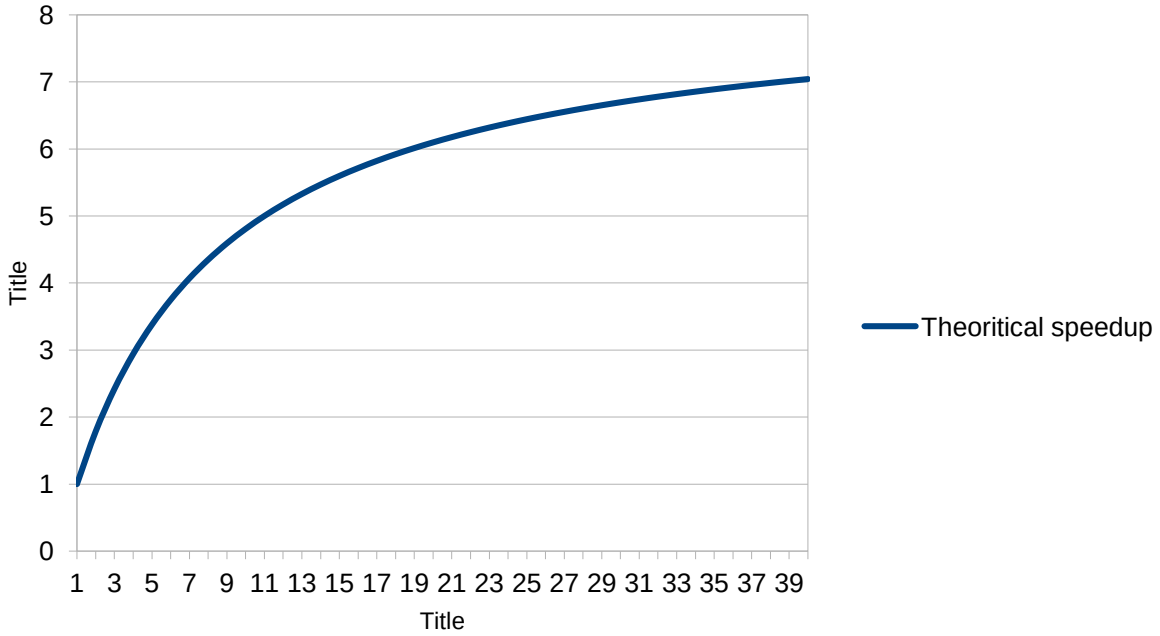
```
            __m512d posyi = _mm512_set1_pd(positions[i].y);
            __m512d poszi = _mm512_set1_pd(positions[i].z);

            for (j = 0; j < bodies; j += 8)
            {

                __m512d mass = _mm512_load_pd(&masses[j]);

                __m512d posxj = _mm512_setr_pd(positions[j].x, positions[j + 1].x, positions[j + 2].x, positions[j + 3].x, positions[j + 4].x, positions[j + 5].x, positions[j + 6].x, positions[j + 7].x);
                __m512d posyj = _mm512_setr_pd(positions[j].y, positions[j + 1].y, positions[j + 2].y, positions[j + 3].y, positions[j + 4].y, positions[j + 5].y, positions[j + 6].y, positions[j + 7].y);
                __m512d poszj = _mm512_setr_pd(positions[j].z, positions[j + 1].z, positions[j + 2].z, positions[j + 3].z, positions[j + 4].z, positions[j + 5].z, positions[j + 6].z, positions[j + 7].z);

                __m512d sijx = _mm512_sub_pd(posxi, posxj);
                __m512d sijy = _mm512_sub_pd(posyi, posyj);
                __m512d sijz = _mm512_sub_pd(poszi, poszj);

                __m512d mod = _mm512_sqrt_pd(_mm512_fmadd_pd(sijx, sijx, _mm512_fmadd_pd(sijy, sijy, _mm512_mul_pd(sijz, sijz))));
                __m512d mod2 = _mm512_mul_pd(mod, mod);
                __m512d mod3 = _mm512_mul_pd(mod2, mod);



                __m512d j_indexes = _mm512_setr_pd((double)j,(double)j+1,(double)j+2,(double)j+3,(double)j+4,(double)j+5,(double)j+6,(double)j+7);
                __mmask8 mask = _mm512_cmp_pd_mask(i_indexes, j_indexes, _CMP_EQ_OQ);
                __m512d s_inotj = _mm512_div_pd(_mm512_mul_pd(GC, mass), mod3);
                __m512d s = _mm512_mask_blend_pd(mask, s_inotj, s_ieqj);

                __m512d Sx = _mm512_mul_pd(s, _mm512_mul_pd(sijx, _mm512_set1_pd(-1)));
                __m512d Sy = _mm512_mul_pd(s, _mm512_mul_pd(sijy, _mm512_set1_pd(-1)));
                __m512d Sz = _mm512_mul_pd(s, _mm512_mul_pd(sijz, _mm512_set1_pd(-1)));


                accxi = _mm512_add_pd(accxi, Sx);
                accyi = _mm512_add_pd(accyi, Sy);
                acczi = _mm512_add_pd(acczi, Sz);
              }

            accelerations[i].x = _mm512_reduce_add_pd(accxi);
            accelerations[i].y = _mm512_reduce_add_pd(accyi);
            accelerations[i].z = _mm512_reduce_add_pd(acczi);
        }
      }
    }
```

Η n_body_omp_guided δέχεται ως argument τον αριθμό των threads που δόθηκαν από το command line και θέτει τη ρύθμιση του openmp για το πόσα threads να τρέξει. Στη συνέχεια παραλληλοποιούμε την Compute Accelerations, αυτή τη φορά με guided scheduling. Η Compute Accelerations δέχεται ως argument τον τύπο παράλληλης εκτέλεσης,

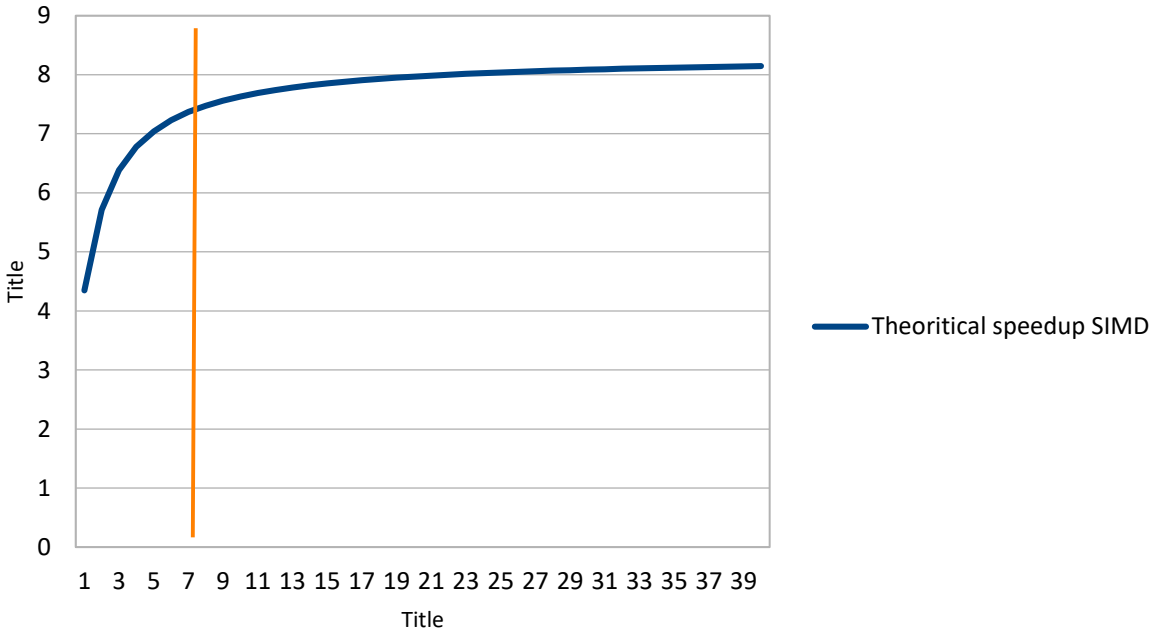
σε αυτήν τη περίπτωση είναι guided. Εκτελούμε το branch του guided scheduling, το οποίο χρησιμοποιεί μια δυναμική ουρά για να διανείμει τις επαναλήψεις στα threads. Αντί για να διανέμονται ισόποσα στα threads, κάθε thread αναλαμβάνει ένα μικρότερο αριθμό επαναλήψεων. Επίσης όσο προχωρά η εκτέλεση το guided scheduling εξυπηρετά μικρότερα chunk sizes. Ο κώδικας χρησιμοποιεί επίσης οδηγίες SIMD για να εκτελέσει τους διανυσματοποιημένους υπολογισμούς, οι οποίοι μπορούν να επιταχύνουν τους υπολογισμούς εκτελώντας πολλαπλές λειτουργίες παράλληλα στα ίδια δεδομένα. Ο τύπος δεδομένων __m512d αντιπροσωπεύει ένα διάνυσμα 512 bit αριθμών κινητής υποδιαστολής διπλής ακρίβειας, το οποίο μπορεί να αποθηκεύσει έως και οκτώ τιμές κάθε φορά. Οι συναρτήσεις _mm512_set1_pd και _mm512_setr_pd χρησιμοποιούνται για τη φόρτωση δεδομένων σε αυτά τα διανύσματα και οι υπόλοιπες συναρτήσεις εκτελούν λογικές πράξεις πάνω τους για να εκτελέσουν τη ίδια λογική με το σειριακό πρόγραμμα.
Ο κώδικας χρησιμοποιεί επίσης τη συνάρτηση _mm512_reduce_add_pd για να μειώσει τα διανύσματα σε μεμονωμένες τιμές αθροίζοντας όλα τα στοιχεία τους. Τέλος, ο κώδικας ενημερώνει τις θέσεις και τις επιταχύνσεις των σωματιδίων χρησιμοποιώντας τις υπολογισμένες τιμές.

3) Ο ιδανικός αριθμός threads. (5%)

Σύμφωνα με την 1η εργασία η συνάρτηση Compute Accelerations απασχολούσε το πρόγραμμα για 88% του χρόνου. Επίσης, σύμφωνα με το νόμο του Amdahl η θεωρητική μέγιστη επιτάχυνση που μπορεί να επιτευχθεί με την παραλληλοποίηση της συνάρτησης εξαρτάται από το ποσοστό της συνάρτησης που μπορεί να παραλληλοποιηθεί και τον αριθμό των επεξεργαστών που χρησιμοποιούνται. Για το συγκεκριμένο ποσοστό παραλληλοποίησης, αν χρησιμοποιηθούν 40 επεξεργαστές, η μέγιστη θεωρητική επιτάχυνση που μπορεί να επιτευχθεί βάσει του νόμου του Amdahl είναι 1 / ((1 - 0.88) + (0.88 / 40)) = 7.04.



Σε αυτή την εργασία χρησιμοποιούμε SIMD όπου το κάθε thread για κάθε κύκλο εκτελεί 8 εντολές αντί 1. Αυτό σημαίνει ότι κάθε thread συμπεριφέρεται σαν 8 διαφορετικά threads. Άρα μπορούμε να τροποποιήσουμε το νόμο του Αμνταλ και να θεωρήσουμε πως για κάθε thread έχουμε 8πλασια computing units. Συνεπώς για 40 threads, ο νόμος αλλάζει σε 1 / ((1 - 0.88) + (0.88 / 40 * 8)) = 8.15 και η γραφική παράσταση έχει ως εξής.



Μπορούμε να πούμε ότι θεωρητικά μετά τα 7-8 threads η γραμμή του γραφήματος να αρχίζει να ευθυγραμμίζεται.

Μετά από πιάσιμο μετρήσεων ο ιδανικός αριθμός threads που επέλεξα για μέγιστο speedup και λιγότερους πόρους για κάθε περίπτωση έχει ως εξής.

Schedule	Threads	Speedup
Static O0	10	5.22
Static O3	8	3.32
Dynamic O0	8	4.03
Dynamic O3	8	3.13
Guided O0	10	5.22
Guided O3	8	3.74

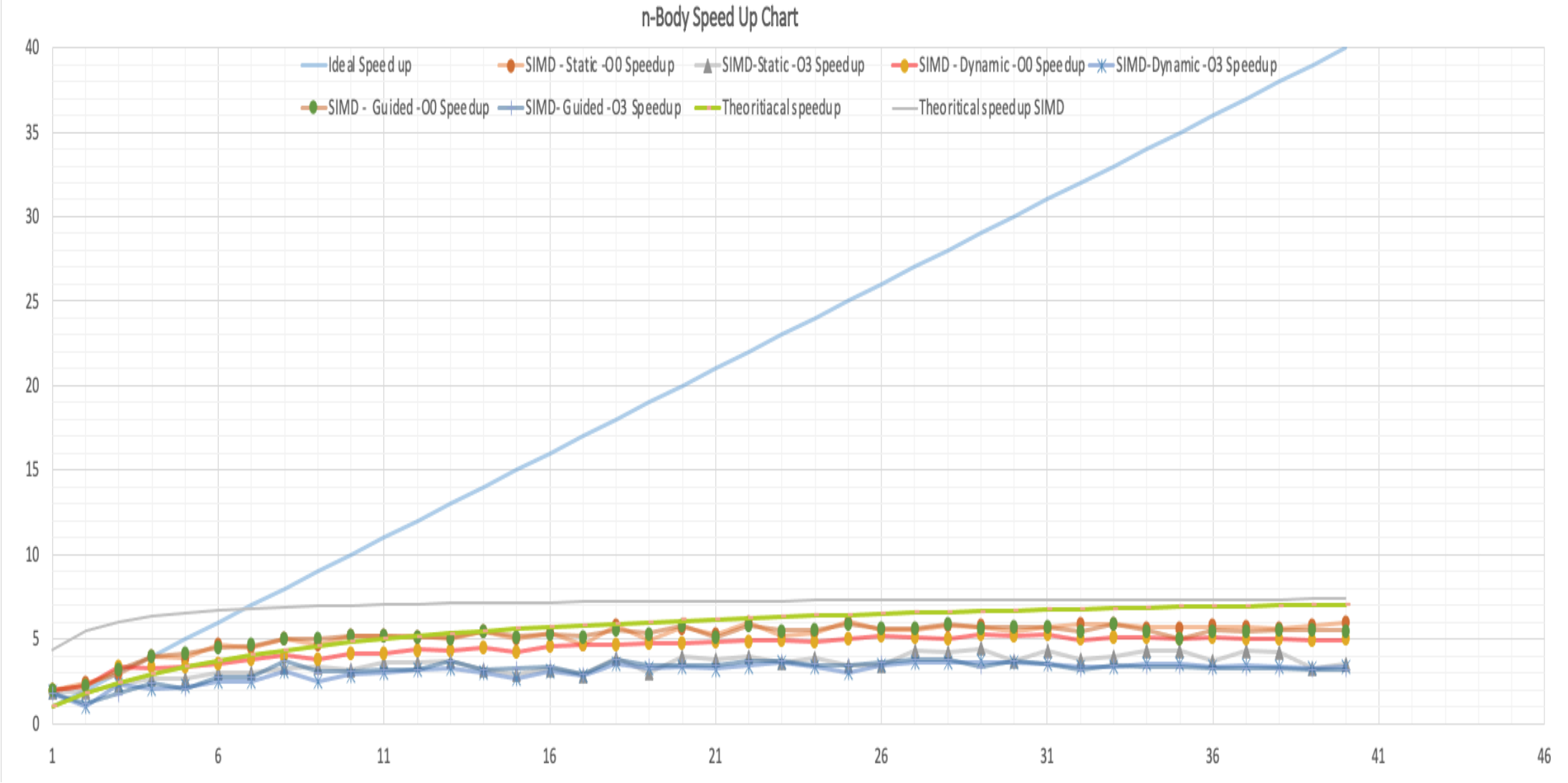
Σε σύγκριση με την προηγούμενη εργασία ο βέλτιστος αριθμός thread είναι μικρότερος. Σημαίνει πως μεγιστοποιούμε το όφελος παραλληλίας πιο σύντομα, αν βάλουμε και υπόψη το κόστος πόρων και το communication overhead.

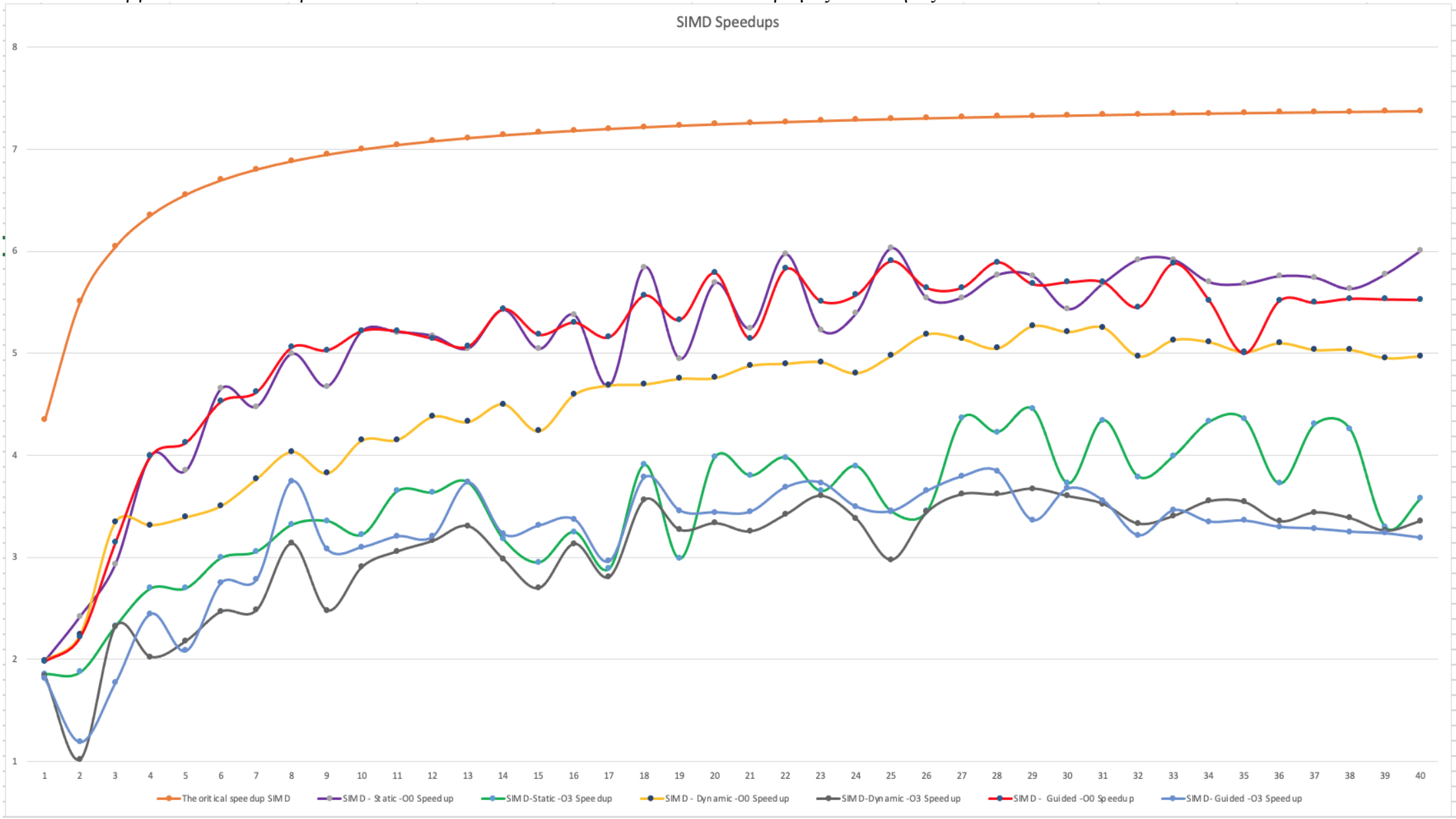
4) Αναμενόμενο και καταμετρημένο (Speedup). Efficiency. (5%)

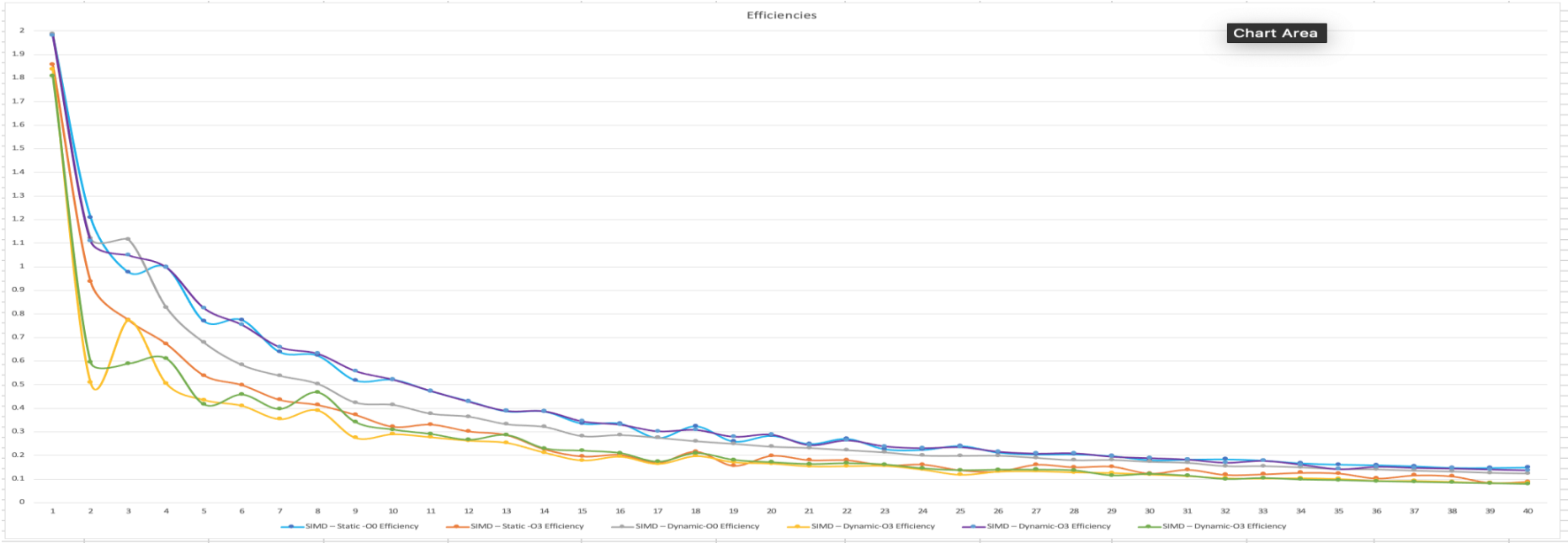
Threads	Theoritical speedup SIMD	SIMD - Static -O0 Speedup	SIMD – Static -O0 Efficiency	SIMD-Static -O3 Speedup	SIMD – Static -O3 Efficiency	SIMD - Dynamic -O0 Speedup	SIMD – Dynamic-O0 Efficiency	SIMD-Dynamic -O3 Speedup	SIMD – Dynamic-O3 Efficiency	SIMD - Guided -O0 Speedup	SIMD – Dynamic-O3 Efficiency	SIMD- Guided -O3 Speedup	SIMD – Dynamic-O3 Efficiency
1	4.347826087	1.983899465	1.983899465	1.857190071	1.857190071	1.987698531	1.987698531	1.836890086	1.836890086	1.981161531	1.981161531	1.809048159	1.809048159
2	5.50660793	2.419654173	1.209827087	1.874608827	0.937304413	2.242905298	1.121452649	1.01870306	0.50935153	2.219744001	1.109872001	1.190415999	0.595208
3	6.043513296	2.933977009	0.977992336	2.324918628	0.774972876	3.347804777	1.115934926	2.318681593	0.772893864	3.147546267	1.049182089	1.770629259	0.590209753
4	6.353240152	3.995290352	0.998822588	2.695523258	0.673880814	3.313242933	0.828310733	2.020757306	0.505189326	3.992360219	0.998090055	2.444794953	0.611198738
5	6.554798112	3.848085289	0.769617058	2.697294005	0.539458801	3.393269789	0.678653958	2.178565028	0.435713006	4.125618482	0.825123696	2.081659239	0.416331848
6	6.696428571	4.651541405	0.775256901	2.994735063	0.49912251	3.504804737	0.584134123	2.467664965	0.411277494	4.528784501	0.754797417	2.75082007	0.458470012
7	6.801399145	4.475254083	0.639322012	3.055757721	0.436536817	3.768708339	0.538386906	2.481459098	0.354494157	4.619885094	0.659983585	2.780576626	0.397225232
8	6.882312457	4.99351592	0.62418949	3.31811638	0.414764548	4.034322401	0.5042903	3.134167651	0.391770956	5.058975001	0.632371875	3.741724452	0.467715557
9	6.946588453	4.672401572	0.51915573	3.354802869	0.372755874	3.823071318	0.424785702	2.4780506	0.275338956	5.029735798	0.558859533	3.075930841	0.341770093
10	6.998880179	5.219596243	0.521959624	3.223039534	0.322303953	4.148602958	0.414860296	2.908449515	0.290844951	5.220988793	0.522098879	3.098455936	0.309845594
11	7.042253521	5.204169997	0.473106363	3.649885696	0.331807791	4.152208331	0.377473485	3.057872707	0.277988428	5.216345314	0.47421321	3.204189564	0.29128996
12	7.07881076	5.167996048	0.430666337	3.63391261	0.302826051	4.378805182	0.364900432	3.160142784	0.263345232	5.143556667	0.428629722	3.204515265	0.267042939
13	7.110041566	5.044477155	0.388036704	3.731949703	0.287073054	4.327222862	0.332863297	3.304393244	0.254184096	5.067676283	0.389821253	3.731186096	0.287014315
14	7.137030995	5.422928794	0.387352057	3.175893263	0.226849519	4.500085502	0.321434679	2.97809658	0.212721184	5.433049348	0.388074953	3.225974296	0.230426735
15	7.160588123	5.039334526	0.335955635	2.949932041	0.196662136	4.236071163	0.282404744	2.696527568	0.179768505	5.182885903	0.345525727	3.308931446	0.22059543
16	7.181328546	5.373345413	0.335834088	3.248789127	0.20304932	4.592565922	0.28703537	3.131787645	0.195736728	5.301684398	0.331355275	3.37119585	0.210699741
17	7.199728951	4.679181293	0.275245958	2.889365272	0.169962663	4.684161071	0.275538887	2.806788807	0.165105224	5.159676509	0.303510383	2.961590087	0.174211182
18	7.216164208	5.841657208	0.324536512	3.905167418	0.216953745	4.694822456	0.26082347	3.56396259	0.197997922	5.565700591	0.309205588	3.78160999	0.210089444
19	7.230933171	4.942137455	0.260112498	2.986335109	0.157175532	4.749752182	0.249986957	3.26729647	0.171962972	5.328757707	0.280460932	3.454834539	0.181833397
20	7.244277021	5.685536393	0.28427682	3.98554063	0.199277032	4.757625165	0.237881258	3.332974949	0.166648747	5.786467766	0.289323388	3.439196485	0.171959824
21	7.256392536	5.241004994	0.249571666	3.799935442	0.180949307	4.877394496	0.232256881	3.252407831	0.154876563	5.144578078	0.244979908	3.445522765	0.164072513
22	7.26744186	5.967580522	0.27125366	3.977273942	0.180785179	4.897574574	0.222617026	3.418721729	0.155396442	5.826456658	0.264838939	3.685313826	0.167514265
23	7.277559803	5.226181287	0.227225273	3.648539701	0.158632161	4.912481683	0.21358616	3.601879949	0.156603476	5.509745086	0.239554134	3.729876714	0.162168553
24	7.286859364	5.390013276	0.224583886	3.892997001	0.162208208	4.803417728	0.200142405	3.375055797	0.140627325	5.570146244	0.232089427	3.492774089	0.145532254
25	7.295435975	6.026783024	0.241071321	3.447994602	0.137919784	4.976154269	0.199046171	2.974372423	0.118974897	5.902717316	0.236108693	3.451443204	0.138057728
26	7.303370787	5.536733737	0.212951298	3.448940841	0.132651571	5.185812861	0.199454341	3.440233344	0.132316667	5.639588558	0.216907252	3.648317851	0.140319917
27	7.310733239	5.541354296	0.205235344	4.360720691	0.161508174	5.138495294	0.190314641	3.618282833	0.134010475	5.64347462	0.209017579	3.791847935	0.140438812
28	7.317583107	5.764941431	0.205890765	4.226283393	0.150938693	5.047907163	0.180282399	3.61555388	0.129126924	5.892116525	0.210432733	3.840761214	0.137170043
29	7.323972118	5.753001772	0.198379371	4.451731029	0.153507967	5.26634542	0.181598118	3.671528481	0.12660443	5.677131323	0.195763149	3.361441733	0.115911784
30	7.32994527	5.428778498	0.180959283	3.725252606	0.124175087	5.208542218	0.173618074	3.59827747	0.119942582	5.695737139	0.189857905	3.672789958	0.122426332
31	7.335541884	5.682354131	0.183301746	4.342344866	0.140075641	5.251091439	0.169390046	3.52202687	0.11361377	5.695571693	0.183728119	3.554393689	0.114657861
32	7.340796476	5.91396832	0.18481151	3.787215501	0.118350484	4.968840402	0.155276263	3.327507214	0.1039846	5.450708156	0.17033463	3.213728495	0.100429015
33	7.345739471	5.912352915	0.17916221	3.992521535	0.120985501	5.129909837	0.155451813	3.405227574	0.103188714	5.880194564	0.178187714	3.459666979	0.104838393
34	7.350397786	5.694002052	0.167470649	4.326537258	0.127251096	5.109157142	0.150269328	3.548961356	0.104381216	5.517898961	0.162291146	3.348533756	0.098486287
35	7.354795327	5.6772957	0.162208449	4.351844269	0.124338408	5.011172409	0.143176355	3.540422681	0.101154934	4.998847766	0.142824222	3.358231187	0.095949462
36	7.358953393	5.751830376	0.159773066	3.722017333	0.10338937	5.09781951	0.141606097	3.351363428	0.093093429	5.515403419	0.153205651	3.296417007	0.091567139
37	7.362891029	5.736918678	0.155051856	4.30199048	0.116270013	5.032874738	0.136023642	3.438071279	0.092920845	5.495837178	0.14853614	3.280029908	0.088649457
38	7.36662531	5.627629862	0.148095523	4.253022847	0.111921654	5.033571404	0.132462405	3.382131798	0.089003468	5.535330075	0.145666581	3.249237477	0.085506249
39	7.370171593	5.770496949	0.14796146	3.294303512	0.084469321	4.9527851	0.12699449	3.259121159	0.083567209	5.528041266	0.141744648	3.233303724	0.082905224
40	7.373543725	6.00024361	0.15000609	3.572889834	0.089322246	4.971219127	0.124280478	3.354397507	0.083859938	5.52359847	0.138089962	3.190629841	0.079765746

Φαίνεται το static scheduling να έχει το μεγαλύτερο speedup, μετά το guided και τέλος το dynamic. Έχει να κάνει με την εκμετάλλευση της χωρικής τοπικότητας.
Όσο αφορά τις εκτελέσεις O0, το καταμετρημένο speedup είναι λίγο πιο χαμηλό από το αναμενόμενο, διότι έχουμε communication overhead μεταξύ των threads.
Όσο αφορά τις εκτελέσεις με O3, το καταμετρημένο speedup είναι κατά πολύ πιο χαμηλό από το αναμενόμενο, ίσως επειδή το O3 μειώνει το μερίδιο του χρόνου εκτέλεσης της compute accelerations. Έτσι το παραλληλοποιήσιμο μερίδιο του προγράμματος μειώνεται σε λιγότερο από το αναμενόμενο 88%. Ακόμη μπορεί να γίνονται βελτιστοποιήσεις της σειριακής εκτέλεσης, χωρίς να αφήνει περιθώριο βελτίωσης του χρόνου εκτέλεσης με πολλά threads.
Η αποδοτικότητα είναι καλύτερη στο Static O0 για τους πιο πάνω λόγους.

5) Γραφική Παράσταση και Σχολιασμός (~100 λέξεις) για τις λύσεις πιο πάνω. (10%-10%)



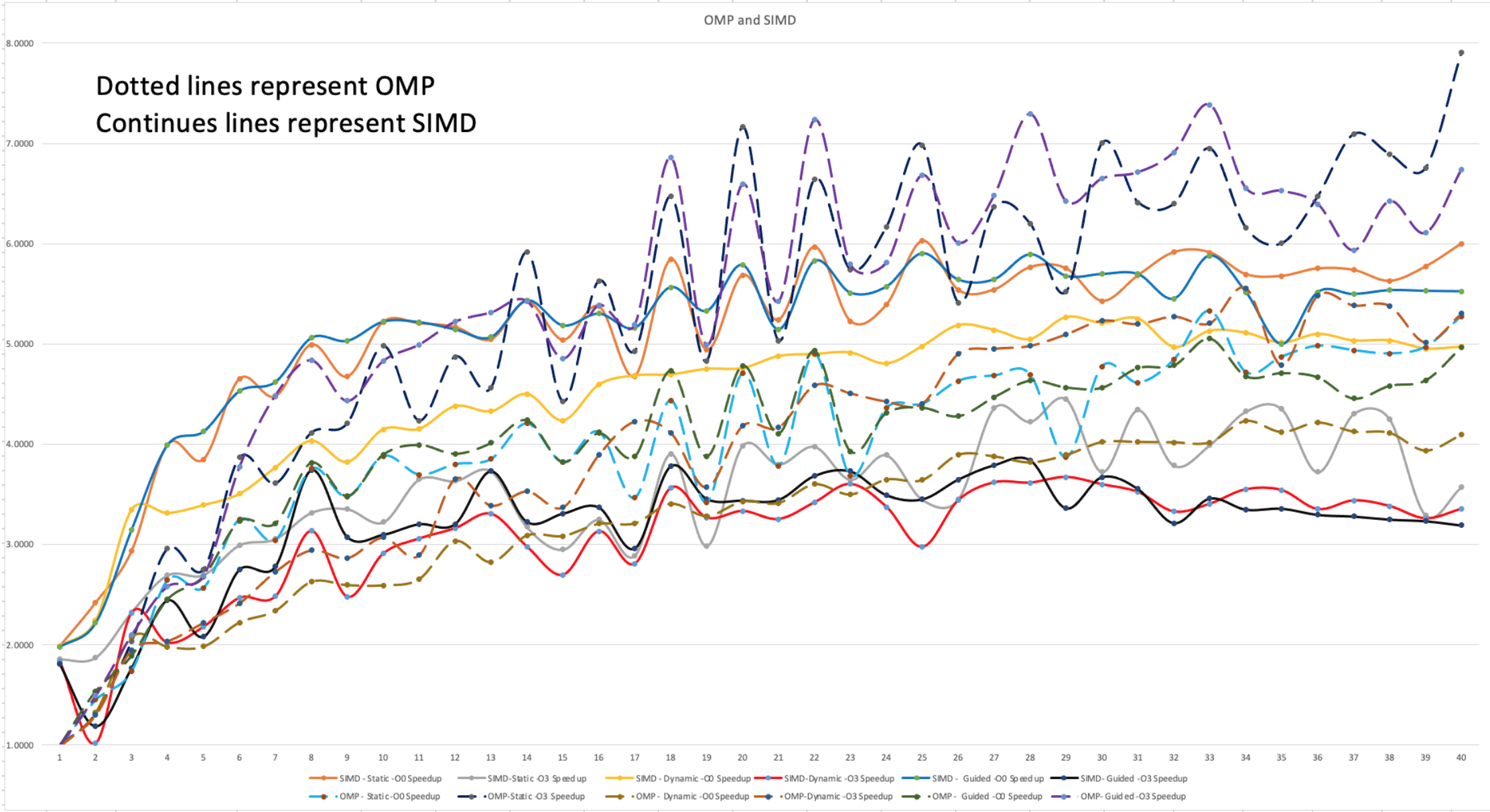

SIMD Speedups
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
The oritical spee dup SIMD
SIMD - Static -O0 Speed up
SIMD-Static -O3 Spee dup
SIMD - Dynamic -O0 Speed up
SIMD-Dynamic -O3 Speed up
SIMD - Guided -O0 Speedup
SIMD- Guided -O3 Speed up



Από τις γραφικές παραστάσεις μπορούμε να συμπεράνουμε ότι τα καταμετρημένα και θεωρητικά speedups δεν είναι καθόλου κοντά στο ιδεατό speedup. Αυτό γίνεται επειδή δεν είναι 100% του κώδικα παραλληλοποιημένος. Επίσης παρατηρούμε πως είμαστε αρκετά πιο κάτω από το θεωρητικό speedup που υπολογίστηκε με το νόμο του Amdahl. Αυτό πιθανότατα γίνεται επειδή έχουμε communication overhead μεταξύ των threads. Επίσης ίσως έχουμε καθυστερήσεις κατά το διάβασμα των τιμών από τη μνήμη, αφού ανάλογα με ποια στρατηγική χρησιμοποιούμε για το κατακερματισμό εργασίας, μειώνεται ή αυξάνεται η εκμετάλλευση της τοπικής χωρικότητας. Μπορούμε να παρατηρήσουμε ότι ο κατακερματισμός εργασίας με static scheduling για αυτό το πρόβλημα, λειτουργεί πιο αποδοτικά από τις άλλες 2 μεθόδους, αφού εκμεταλλεύεται καλύτερα την τοπική χωρικότητα. Το dynamic scheduling ακόμη και αν συγχρωνίζει καλύτερα τα threads, δεν λειτουργεί καλά επειδή δεν εκμεταλλεύεται τη χωρική τοπικότητα. Το guided λειτουργεί καλύτερα όταν έχουμε λίγα threads στη διάθεσή μας, αφού τα threads συγχρονίζονται καλύτερα από το static και δεν έχουμε καθυστέρηση όταν περιμένουμε το αργότερο thread να τελειώσει. Όμως κατά τα αρχικά στάδια εκτέλεσης εκμεταλλεύεται τη χωρική τοπικότητα. Συνεπώς το guided είναι μια λύση που παίρνει τα πλεονεκτήματα του static και dynamic scheduling, όμως στο συγκεκριμένο πρόγραμμα λειτουργεί καλύτερα μόνο για λίγα threads.

Όσο αφορά τις εκτελέσεις μεταξύ O3 και O0 οι εκτελέσεις με O3 εκτελούνται σε πολύ πιο μικρό χρονικό διάστημα σε σύγκριση με το O0. Αυτό σημαίνει πως δεν έχουμε την πιο βέλτιστη λύση του προγράμματος και ο compiler μπορεί να δημιουργήσει πιο βελτιωμένη λύση. Όσο αφορά τα speedups μεταξύ των 2 λύσεων και των efficiencies, ισχύουν τα ίδια που προ-ανάφερα στο πιο πάνω ερώτημα.

6) Γραφική Παράσταση μαζί με <u>OpenMP</u> και Σχολιασμός (~200 λέξεις) για τις λύσεις πιο πάνω. (30%)



Παρατηρούμε ότι με τα SIMD έχουμε καλύτερα speedups όταν το compilation γίνεται με O0, ενώ στην OMP έχουμε καλύτερα speedups όταν το compilation γίνεται με O3. Αυτό δείχνει πως στο OMP υπάρχουν αρκετές περεταίρω βελτιστοποιήσεις που μπορούν να γίνουν στο παραλληλοποιημένο κομμάτι του κώδικα. Αντίθετα με την υλοποίηση του SIMD στο παραλληλοποιημένο κομμάτι του κώδικα δεν αφήνουμε αρκετό περιθώριο βελτιστοποίησης και έτσι βελτιστοποιείται μόνο το σειριακό κομμάτι κατά το compile με O3. Άρα σύμφωνα και με τη σημείωση που έγραψα στο 0) ερώτημα, έχουμε πιο γρήγορο σειριακό O3 που ίσως χρησιμοποιεί SIMD, δεν μας αφήνει πολύ περιθώριο για speedup. Ακόμη παρατηρούμε ότι χρησιμοποιώντας SIMD έχουμε βέλτιστη αναλογία πόρων (threads) ως προς το speedup όταν χρησιμποιούμε 7-9 threads, αν αυξήσουμε των αριθμό των threads δεν έχουμε σημαντική άνοδο στα speedups. Από την άλλη η λύση που χρησιμοποιεί μόνο το OMP έχει βέλτιστη αναλογία πόρων ως προς το speedup, όταν χρησιμοποιούμε 16-18 threads. Αυτό σημαίνει πως πετυχαίνουμε το ίδιο επίπεδο παραλληλίας με λιγότερα threads όταν χρησιμοποιούμε τα SIMD instructions.