Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών

Παράλληλη Επεξεργασία

Ακαδημαϊκό Έτος 2016 – 2017

ΚΑΡΑΓΚΟΥΝΗΣ ΑΓΓΕΛΟΣ – 5532

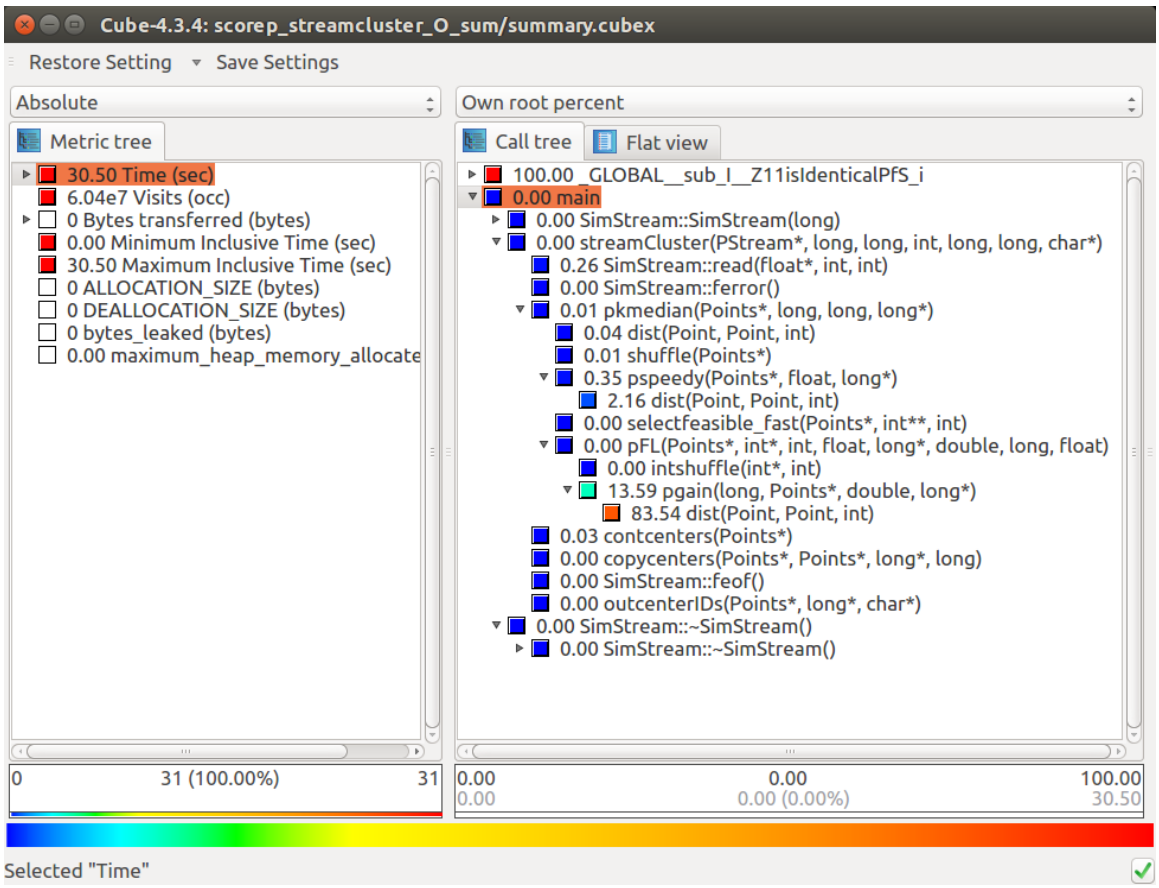
ΛΑΝΤΖΟΣ ΧΑΡΑΛΑΜΠΟΣ – 5570

ΜΑΡΓΑΡΙΤΗΣ ΣΕΡΑΦΕΙΜ – 5581

Ανάλυση σειριακής έκδοσης

Για την ανάλυση του σειριακού κώδικα, χρησιμοποιήθηκε το εργαλείο Scalasca. Όμως λόγω προβλήματος της συνεργασίας του sclasca με το papi, δεν ήταν δυνατή η χρήση των performance counters του papi, συνεπώς δεν μπορούσαμε να εντοπίσουμε τυχόν cache misses και tlb misses.

Πριν την παραλληλοποίηση του προγράμματος, αναλύοντας την σειριακή έκδοση του προγράμματος παίρνουμε τα εξής αποτελέσματα:



Παρατηρούμε ότι οι συναρτήσεις με τον μεγαλύτερο χρόνο εκτέλεσης είναι η pspeedy και η pgain. Συγκεκριμένα, ο μεγαλύτερος χρόνος εκτέλεσης είναι κατά τις κλήσεις τους στην συνάρτηση dist.

Παραλληλοποίηση με OpenMP

Τα σημεία που θα παραλληλοποιήσουμε είναι οι κλήσεις της dist μέσα στις συναρτήσεις pgain και pspeedy.

Δύο είναι τα σημεία στην pgain οπού καλείται σε βρόχο for η dist.

```
#pragma omp parallel for \
shared(switch_membership,points,lower,center_table,x) private(i) \
reduction(+:cost_of_opening_x)
  for ( i = 0; i < points->num; i++ ) {
      float x_cost = dist ( points->p[i], points->p[x], points->dim
) * points->p[i].weight;
      float current_cost = points->p[i].cost;
```

Εισάγοντας εκεί εντολές OpenMP μειώνεται αρκετά ο χρόνος εκτέλεσης. Εδώ απαιτείται reduction για την μεταβλητή cost_of opening_x αφού είναι κοινή σε όλα τα νήματα

```
int assign = points->p[i].assign;
#pragma omp atomic
lower[center_table[assign]] += current_cost – x_cost;
```

Είναι απαραίτητη η εντολή atomic ώστε να αποκλειστούν ενημερώσεις στην ίδια θέση του πίνακα lower από διαφορετικά νήματα.

```
#pragma omp parallel for \
shared(points,gl_lower,center_table,switch_membership,x) \
schedule(static)
for ( int i = 0; i < points->num; i++ ) {
    bool close_center = gl_lower[center_table[points->p[i].assign]]
> 0 ;
    if ( switch_membership[i] || close_center ) {
```

Εισάγουμε παραλληλοποίηση με OpenMP αφού καλείται στην συνάρτηση pspeedy σε ένθετο βρόγχο η dist.

```
#pragma omp parallel for \
shared(points) \
schedule(static)
for ( int k = 0; k < points->num; k++ )  {
    float distance = dist( points->p[i],points->p[k],points->dim );
```

Στις παραπάνω περιπτώσεις επιλέχθηκε στατικός διαμοιρασμός των επαναλήψεων στα νήματα αφού καλείται η συνάρτηση dist που έχει σταθερό πλήθος σημείων, άρα ο φόρτος στο πρόγραμμα είναι ίδιος.

Όλα τα σημεία που έχουν παραλληλοποιηθεί στο πρόγραμμα επιλέχθηκαν έτσι ώστε να επιτευχθεί μέγιστη χρονοβελτίωση, αφού είναι σημεία που δημιουργούν τις μεγαλύτερες καθυστερήσεις.

Παραλληλοποίηση με εντολές διανυσματοποίησης

Οι συναρτήσεις σπαταλούν τον περισσότερο χρόνο στην συνάρτηση dist, η οποία υπολογίζει την απόσταση μεταξύ δύο σημείων.

Για να μειωθεί ο χρόνος αυτός, χρησιμοποιούμε εντολές SIMD, οι οποίες χρησιμοποιούνται για γρήγορη εκτέλεση αριθμητικών πράξεων.

Για να γίνει παραλληλοποίηση με διανυσματοποιηση θα χρησιμοποιήσουμε την δομή v4sf η οποία ορίζει ένα vector που έχει χωρητικότητα 4 αριθμούς κινητής υποδιαστολής μεγέθους 32 bit ο καθένας. Η παραπάνω δομή δεν υποστηρίζει τους κλασσικούς τελεστές αριθμητικών πράξεων, και αντί για αυτούς θα χρησιμοποιηθούν συναρτήσεις που χρησιμοποιούν τις δυνατότητες των σύγχρονων επεξεργαστών να επεξεργάζονται 128 Bit την φορά. Ο βρόγχος μας θα εκτελεστεί N φορές, όπου N ο αρχικός αριθμός επαναλήψεων. Σε κάθε επανάληψη αντιγράφουμε τα δεδομένα που θέλουμε να επεξεργαστούμε στις παραπάνω δομές, και τις χρησιμοποιούμε για να εκτελεστούν πολύ πιο γρήγορα αριθμητικές πράξεις σε τετράδες. Έτσι θα παραχθούν τέσσερα αποτελέσματα, τα οποία θα αθροίσουμε μεταξύ τους. Τέλος, αν το N δεν είναι πολλαπλάσιο του 4, ελέγχουμε αν έχουν μείνει δεδομένα εκτός βρόγχου, και μεταφέρουμε το αποτέλεσμα σε μία μεταβλητή float.

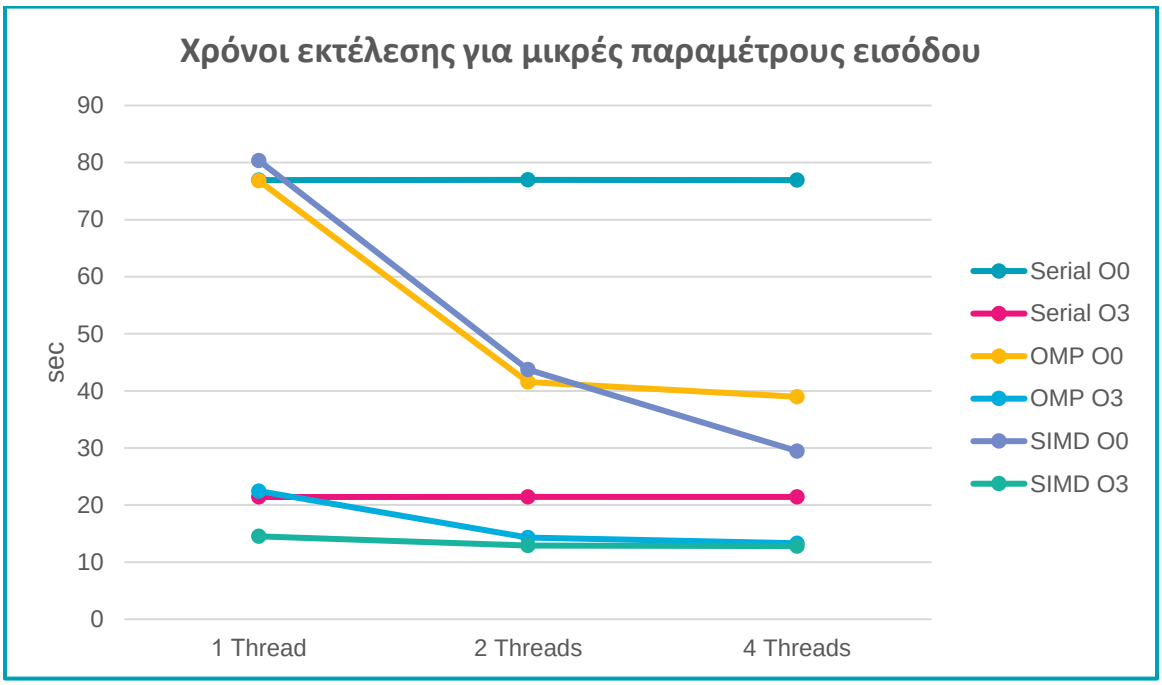
Σύγκριση σειριακής και παράλληλων εκδόσεων

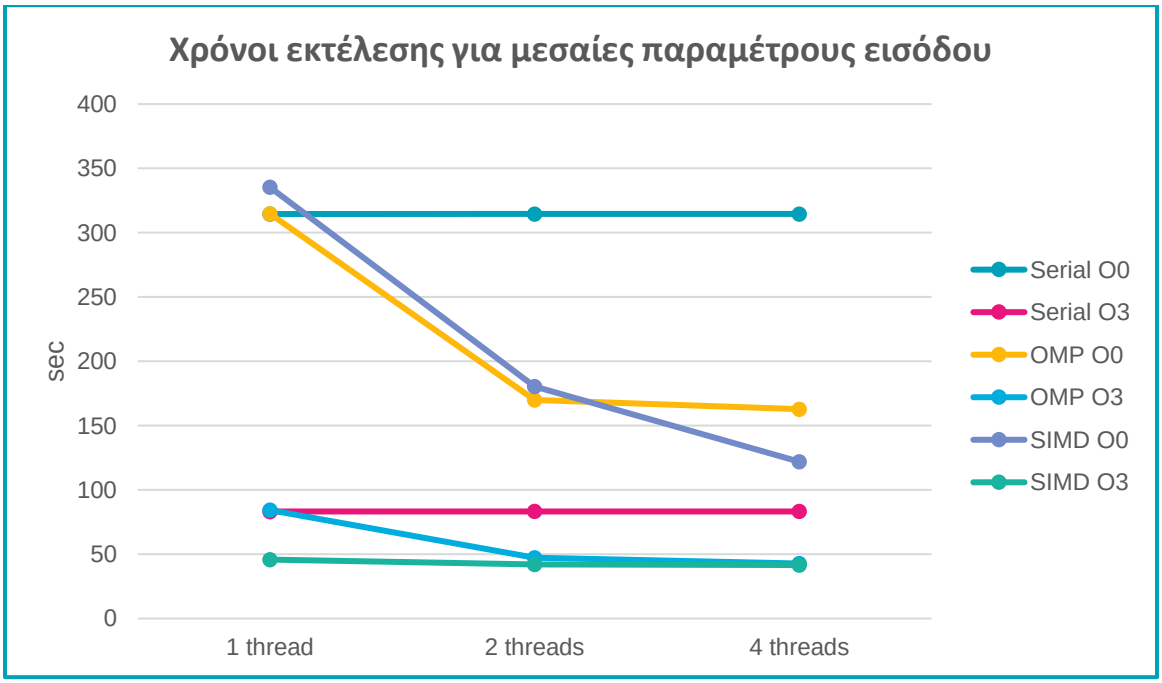
Χρόνοι Εκτέλεσης για είσοδο small						
	Χρόνος εκτέλεσης με O0			Χρόνος εκτέλεσης με O3		
Νήματα	Serial	OpenMP	SIMD	Serial	OpenMP	SIMD
1	76,94	76,82	80,35	21,51	22,44	14,51
2	76,97	41,57	43,72	21,53	14,31	12,92
4	76,89	38,96	29,46	21,64	13,30	12,81

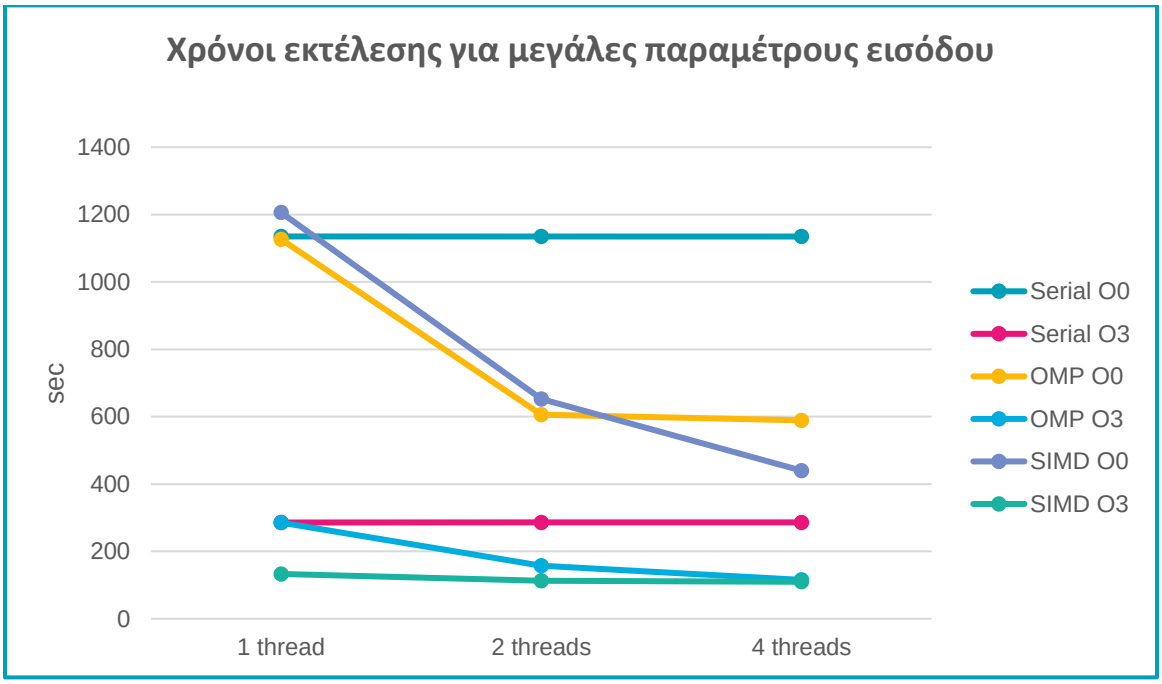
Χρόνοι Εκτέλεσης για είσοδο medium						
	Χρόνος εκτέλεσης με O0			Χρόνος εκτέλεσης με O3		
Νήματα	Serial	OpenMP	SIMD	Serial	OpenMP	SIMD
1	314,61	314,73	335,30	83,25	84,30	45,83
2	314,98	169,85	180,46	83,54	47,27	42,06
4	314,25	162,81	121,97	83,12	42,79	41,63

Χρόνοι Εκτέλεσης για είσοδο large						
	Χρόνος εκτέλεσης με O0			Χρόνος εκτέλεσης με O3		
Νήματα	Serial	OpenMP	SIMD	Serial	OpenMP	SIMD
1	1134,96	1126,60	1205,95	286,23	285,53	133,07
2	1135,21	606,20	652,15	285,93	157,08	112,93
4	1134,23	589,15	439,84	286,89	115,48	109,81

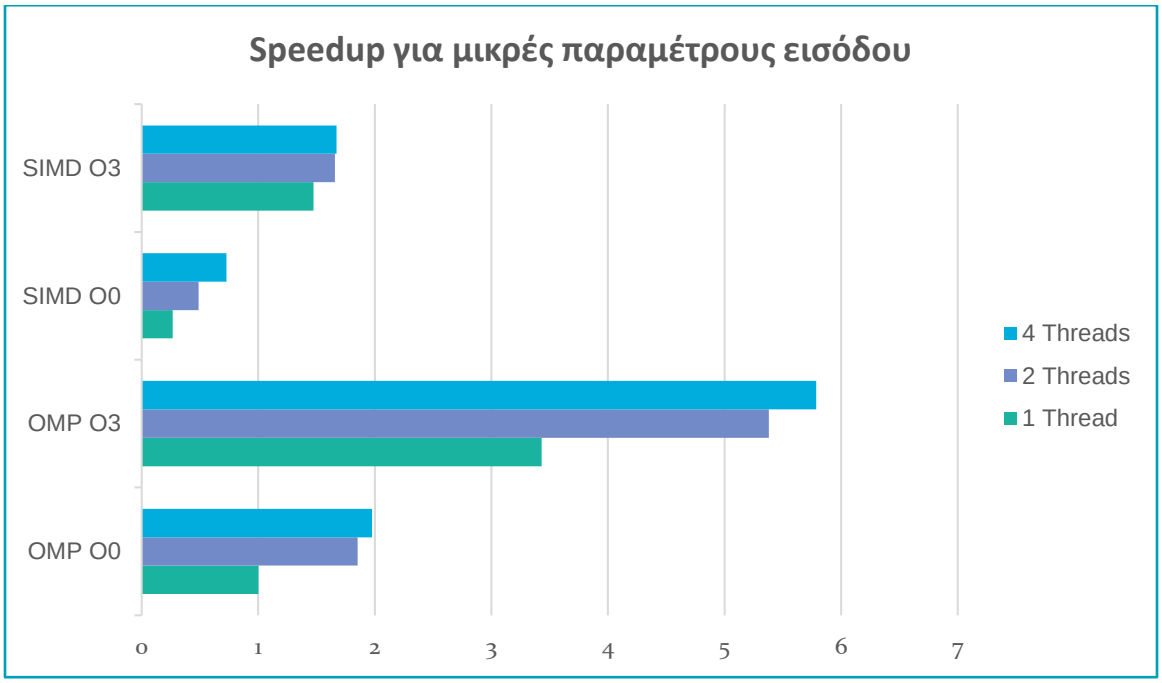
Στα παρακάτω διαγράμματα απεικονίζεται συνοπτικά ο χρόνος εκτέλεσης του προγράμματος σε δευτερόλεπτα για όλες τις διαφορετικές υλοποιήσεις και μεγέθη εισόδου.

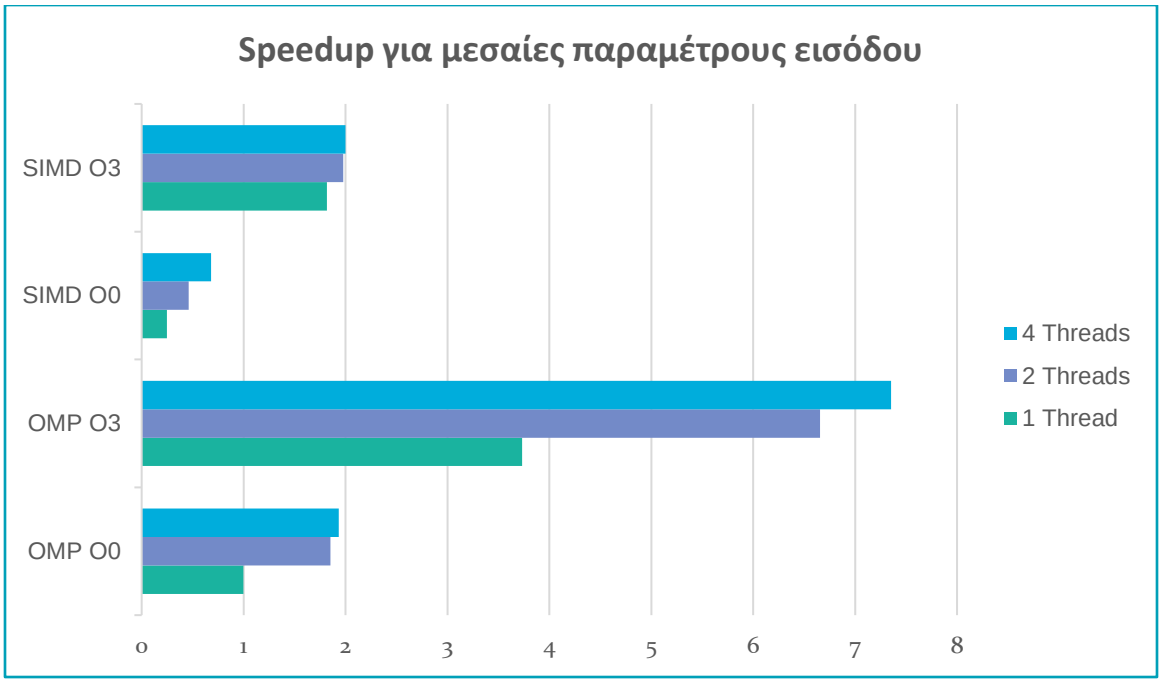


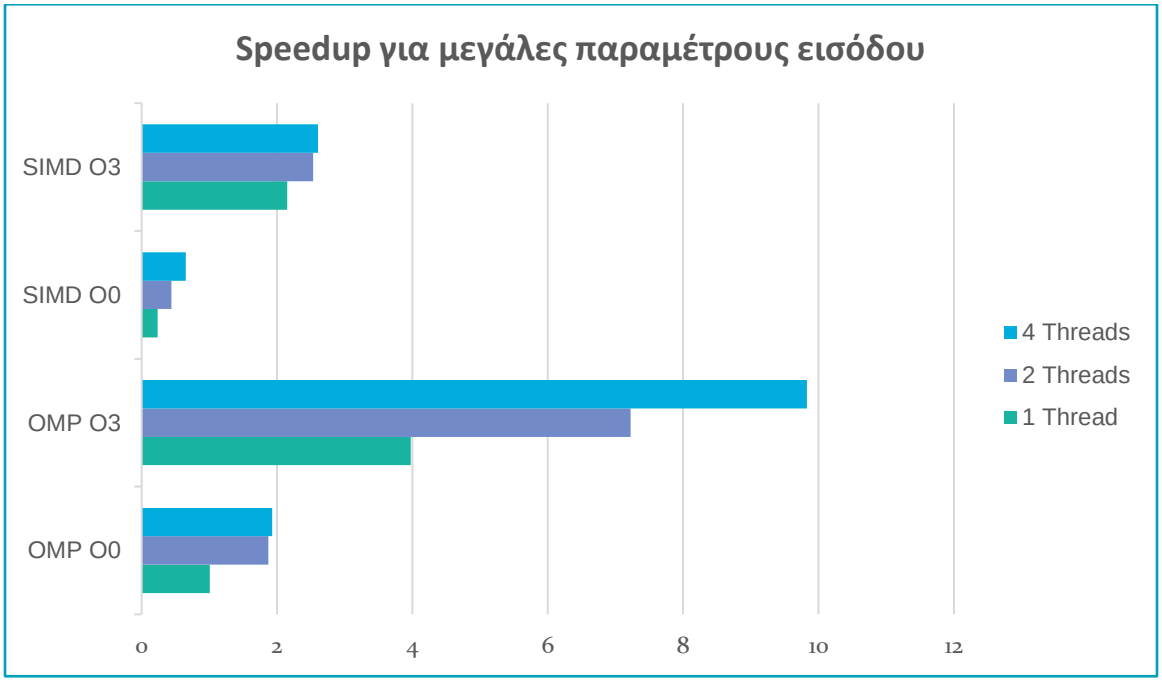




Στα παρακάτω διαγράμματα απεικονίζονται οι χρονοβελτιώσεις (speedup) της κάθε παράλληλης έκδοσης του προγράμματος σε σχέση με το σειριακό:




Speedup για μεσαίες παραμέτρους εισόδου
SIMD O3
SIMD O0
OMP O3
OMP O0
0
1
2
3
4
5
6
7
8
4 Threads
2 Threads
1 Thread


Speedup για μεγάλες παραμέτρους εισόδου
SIMD O3
SIMD O0
OMP O3
OMP O0
0
2
4
6
8
10
12
4 Threads
2 Threads
1 Thread

Τα **speedups(χρονοβελτιώσεις)** υπολογίζονται για κάθε περίπτωση ως εξής:

για τις υλοποιήσεις με O0 : serial_O0 / omp _O0
serial_O0 / simd_O0
για τις υλοποιήσεις με O3: serial_O3 / omp _O3
serial_O3 / simd_O3

Οι χρονοβελτίωση για κάθε περίπτωση βρίσκεται στον παρακάτω πίνακα:

		Small	Medium	Large
OMP O0	1 Thread	1.0024	0.9996	1.0074
	2 Threads	1.8522	1.8522	1.8720
	4 Threads	1.9762	1.9324	1.9264
OMP O3	1 Thread	3.4309	3.7321	3.9749
	2 Threads	5.3802	6.6545	7.2253
	4 Threads	5.7872	7.3512	9.8277
SIMD O0	1 Thread	0.2664	0.2483	0.2373
	2 Threads	0.4897	0.4613	0.4389
	4 Threads	0.7267	0.6825	0.6507
SIMD O3	1 Thread	1.4749	1.8116	2.1508
	2 Threads	1.6570	1.9792	2.5343
	4 Threads	1.6712	1.9999	2.6064

Συμπεράσματα

Είναι φανερό ότι στις περιπτώσεις που δεν χρησιμοποιούμε την βελτιστοποίηση του compiler, η έκδοση με εντολές διανυσματοποίησης είναι πιο αργή σε σχέση με την έκδοση με εντολές OpenMP.
Όταν οι βελτιστοποιήσεις του compiler είναι ενεργές, παρατηρούμε μια μείωση χρόνου όταν προσθέσουμε εντολές διανυσματοποίησης στην έκδοση με OpenMP.

Αυτό είναι φανερό στα παραπάνω διαγράμματα με τα Speedups, στα οποία παρατηρούμε ότι η χρονοβελτίωση για O0 είναι μεγαλύτερες από αυτές του O3.