Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών
Α.Π.Θ.

27/11/2016

Παράλληλα και Κατανεμημένα Συστήματα Υπολογιστών

Εργασία 1

Διαμαντή Μαρία 8133

mfdiamanti@ece.auth.gr

Ντζιώνη Δήμητρα 8209

dntzioni@ece.auth.gr

Πίνακας περιεχομένων

1. Εισαγωγή

Κατά την πρώτη εργασία, στο μάθημα των Παράλληλων και Κατανεμημένων Συστημάτων Υπολογιστών, δίνεται η υλοποίηση ενός προβλήματος σε σειριακό κώδικα. Το πρόβλημα αυτό αφορά την ιεραρχική ομαδοποίηση N σωματιδίων στον τρισδιάστατο χώρο. Κάθε τμήμα του χώρου που δεν είναι άδειο, υποδιαιρείται σε οκτώ υποτμήματα και κάθε σωματίδιο κατατάσσεται στο υποτμήμα του οποίου τα γεωμετρικά όρια το περικλείουν. Η υποδιαίρεση του χώρου σταματά όταν ο πληθυσμός των σωματιδίων που ανήκουν σε ένα υποτμήμα μειωθεί κάτω από ένα συγκεκριμένο όριο S ή ο αριθμός των επιπέδων ξεπεράσει κάποιο πάνω όριο L. Η υλοποίηση του προβλήματος αυτού γίνεται με τη χρήση του κώδικα Morton, μια δημοφιλής τεχνική για την κατασκευή ενός οκταδικού δέντρου. Στόχος, λοιπόν, της εργασίας αυτής είναι ο εντοπισμός των σημείων όπου ο δοσμένος σειριακός κώδικας μπορεί να παραλληλοποιηθεί και έπειτα, η παραλληλοποίηση των σημείων αυτών με Cilk, OpenMP και Pthreads. Η σύγκριση των ταχυτήτων των υπολογισμών τόσο του σειριακού κώδικα συγκριτικά με τις τρεις μεθόδους παραλληλοποίησης, όσο και η σύγκριση μεταξύ των μεθόδων αυτών, είναι ο απώτερος στόχος της εργασίας.

Συνεπώς, η αναφορά αυτή περιλαμβάνει ανάλυση για τον τρόπο επιλογής των σημείων του σειριακού κώδικα που θα παραλληλοποιηθούν, τον τρόπο παραλληλοποίησης των σημείων αυτών με κάθε μια από τις τρεις προαναφερθείσες μεθόδους και τέλος, τα συμπεράσματα που προέκυψαν σχετικά με τη σύγκριση των ταχυτήτων όλων των μεθόδων μέσω διαγραμμάτων.

2. Σειριακός κώδικας προς παραλληλοποίηση.

Έπειτα από τη μελέτη του σειριακού κώδικα που δόθηκε, καταλήξαμε στο συμπέρασμα πως τα σημεία του κώδικα που πρέπει να αλλαχθούν είναι κυρίως τα αρχεία hash_codes.c, morton_codes.c, radix_sort.c, data_rearrangement.c, ενώ μικρές αλλαγές χρειάστηκαν και στο αρχείο test_octree.c αλλά και στο utils.h.

hash_codes.c:

Όσον αφορά το αρχείο hash_codes.c, επιλέξαμε να παραλληλοποιήσουμε μόνο την παρακάτω συνάρτηση.

```
void quantize(unsigned int *codes, float *X, float *low, float step, int N){

 for(int i=0; i<N; i++){

    for(int j=0; j<DIM; j++){

      codes[i*DIM + j] = compute_code(X[i*DIM + j], low[j], step);

    }

  }

}
```

Αυτό διότι οι υπόλοιπες συναρτήσεις του αρχείου είτε εκτελούσαν μικρό αριθμό επαναλήψεων είτε απαιτούσαν προσπέλαση και εγγραφή στο ίδιο σημείο μνήμης από όλα τα νήματα, προκαλώντας πιθανό σφάλμα στο τελικό περιεχόμενο της μεταβλητής. Για την αποφυγή του τελευταίου θα μπορούσαμε να εισάγουμε κάποιο κλειδί προκειμένου η διεργασία να εκτελείται κάθε φορά ατομικά, αλλά αυτό την καθιστούσε πολύ χρονοβόρα συγκριτικά με το σειριακό κώδικα. Σε κάθε περίπτωση, λοιπόν, οποιαδήποτε άλλη μετατροπή καθιστούσε τη διεργασία πιο αργή.

morton_encoding.c:

Στο αρχείο morton_encoding.c, το σημείο προς παραλληλοποίηση είναι το εξής:

```
void morton_encoding(unsigned long int *mcodes, unsigned int *codes, int N, int max_level){

  for(int i=0; i<N; i++){

  mcodes[i]=mortonEncode_magicbits(codes[i*DIM],codes[i*DIM+1],codes[i*DIM + 2]);

  }

}
```

Όπως και προηγουμένως, αυτό είναι το μοναδικό σημείο του κώδικα που μας συμφέρει να τροποποιήσουμε. Μια σκέψη ήταν να παραλληλοποιήσουμε και τη συνάρτηση mortonEncode_magicbits(), ωστόσο πάλι θα έπρεπε να περιμένουμε να συγχρονιστούν τα νήματα, γεγονός που μείωνε την ταχύτητα εκτέλεσης της συνάρτησης.

radix_sort.c:

Στο αρχείο radix_sort.c, το σημείο προς παραλληλοποίηση είναι το εξής:

```
   for(int i=0; i<MAXBINS; i++){ // Call the function recursively to split the lower levels

     int offset = (i>0) ? BinSizes[i-1] : 0;

     int size = BinSizes[i] - offset;

     truncated_radix_sort(&morton_codes[offset], &sorted_morton_codes[offset], &permutation_vector[offset], &index[offset], &level_record[offset], size, population_threshold, sft-3, lv+1);

   }
```

Στους υπόλοιπους βρόχους επαναλήψεων απαιτείται η χρήση κλειδιών προκειμένου να γίνεται σωστή εγγραφή στη μνήμη των μεταβλητών κατά την εκτέλεση των παρακάτω εντολών:

```
BinSizes[ii]++;

offset += ss;

BinCursor[ii]++;
```

Η χρήση των κλειδιών αυτών προκαλεί καθυστέρηση στο χρόνο εκτέλεσης των βρόχων και δεν παρατηρείται καμία βελτίωση, για αυτό και την αποφεύγουμε.

data_rearrangement:

Η μοναδική αλλαγή που μπορεί να γίνει στο συγκεκριμένο αρχείο είναι ο βρόχος επανάληψης που έχει, το οποίο παραλληλοποιείται απλά ως εξής:

```
void data_rearrangement(float *Y, float *X, unsigned int *permutation_vector, int N){
  cilk_for(int i=0; i<N; i++){
    memcpy(&Y[i*DIM], &X[permutation_vector[i]*DIM], DIM*sizeof(float));
  }
}
```

3. Μέθοδοι παραλληλοποίησης

3.1. Cilk

Για την υλοποίηση της παραλληλοποίησης με τη μέθοδο Cilk, εισήγαμε τα απαραίτητα headers: “cilk/cilk.h", "cilk/cilk_api.h". Σύμφωνα με την εκφώνηση της εργασίας είναι απαραίτητο να μεταβάλλεται ο αριθμός των νημάτων τα οποία θα χρησιμοποιηθούν. Για το λόγο αυτό, εισήγαμε ένα επιπλέον όρισμα στη main(), το argv[6], στο οποίο δίνεται από το χρήστη ο αριθμός αυτός. Ωστόσο, για να ορισθεί σε ολόκληρο το πρόγραμμα το πλήθος των διαθέσιμων νημάτων, χρησιμοποιήσαμε τη συνάρτηση:

```
__cilkrts_set_param("nworkers","argv[6]")
```

Όσον αφορά τα αρχεία προς τροποποίηση, η διαφοροποίηση εντοπίζεται στην εισαγωγή της cilk_for αντί για for στην αρχή ορισμένων επαναλήψεων που ορίστηκαν παραπάνω. Η cilk_for χωρίζει μια επανάληψη σε τμήματα κάθε ένα από τα οποία περιέχει μια ή περισσότερες επαναλήψεις. Έπειτα, τα τμήματα αυτά εκτελούνται το καθένα από ένα thread.

3.2. Openmp

Για την υλοποίηση της παραλληλοποίησης με τη μέθοδο Openmp, εισήγαμε το απαραίτητο header: <omp.h>. Προκειμένου να μεταβάλλεται ο αριθμός των νημάτων όπως το επιθυμεί ο χρήστης, εισήγαμε ένα επιπλέον όρισμα στη main(), το argv[6]. Επιπρόσθετα, ορίσαμε δύο extern int μεταβλητές, εμφανείς από όλα τα αρχεία, εκ των οποίων η μία αφορά τον αριθμό των διαθέσιμων νημάτων που θα έχω κατά την εκτέλεση του προγράμματος (numThreads), ενώ η άλλη προσμετρά τα ενεργοποιημένα νήματα κάθε χρονική στιγμή (enabledThreads). Οι μεταβλητές αυτές δηλώνονται στο utils.h, καθώς επίσης και στην αρχή κάθε άλλου αρχείου, ενώ αρχικοποιούνται μία φορά στο test_octree.c.

Η υλοποίηση της openmp απαιτεί τη χρήση συγκεκριμένων εντολών, όπως η εντολή #pragma omp parallel num_threads(numThreads) private() shared(), που καθορίζει κάθε φορά το parallel region, το πλήθος των νημάτων που θα το εκτελέσουν, τις private και shared μεταβλητές που χρησιμοποιούν τα νήματα. Οι private μεταβλητές είναι ανεξάρτητες για κάθε thread, ενώ στις shared έχουν πρόσβαση όλα τα νήματα. Έπειτα, η εντολή #pragma omp for δηλώνει την παράλληλη εκτέλεση ενός βρόχου for. Σε περίπτωση που υπάρχει φωλιασμένη επανάληψη, εάν χρειάζεται να εκτελεστεί κι αυτή παράλληλα, τότε πρέπει να ενεργοποιηθεί η επιλογή omp_set_nested(1). Σε ορισμένες περιπτώσεις, δηλώνεται ακόμη και το πλήθος των επαναλήψεων που θα εκτελέσει κάθε νήμα μέσω του schedule.

3.2.1. Hash codes, Morton encoding, Data rearrangement

Όσον αφορά, την τροποποίηση της συνάρτησης quantize() του hash_codes.c, επιλέχθηκε να μην ορισθεί κάποιο schedule και αφήνεται στην κρίση του συστήματος ο τρόπος με τον οποίο θα χωριστούν οι επαναλήψεις στο κάθε thread. Αυτό επιλέχθηκε διότι λόγω της φωλιασμένης επανάληψης ήταν αρκετά περίπλοκο να ορισθεί ένα chunk για κάθε πιθανό αριθμό από νήματα που θα κληθούν να εκτελέσουν αυτό το κομμάτι κώδικα παράλληλα.

Για την παραλληλοποίηση, τόσο της συνάρτησης morton_encoding() του αρχείου morton_encoding.c, όσο και της data_rearrangement() του αρχείου data_rearrangement.c ορίζουμε τον τρόπο διαμοιρασμού ως static κι αυτό διότι γνωρίζουμε εξαρχής τον αριθμό των επαναλήψεων, αλλά και το μέγεθος της εργασίας που πρέπει να εκτελέσει το κάθε νήμα. Με αυτόν τον τρόπο, όλοι οι επεξεργαστές είναι απασχολημένοι για ίδιο χρονικό διάστημα κι έτσι, οι εργασίες μοιράζονται ισόποσα σε κάθε νήμα παρατηρώντας μάλιστα βελτίωση στο χρόνο εκτέλεσης της συνάρτησης.

3.2.2. Radix sort

Όπως αναφέρθηκε το σημείο που παραλληλοποιείται στο αρχείο radix_sort.c είναι ο βρόχος όπου πραγματοποιείται η αναδρομική κλήση της ίδιας της συνάρτησης truncated_radix_sort(). Στη μέθοδο αυτή, υπάρχει μια δυσκολία στην υλοποίηση της παραλληλοποίησης καθώς ο αριθμός των νημάτων μπορεί να ξεπεράσει το όριο που θέτει ο χρήστης λόγω της συνεχούς αναδρομής. Προκειμένου να αντιμετωπίσουμε το πρόβλημα αυτό, ορίζουμε τη μεταβλητή enabledThreads, την οποία αυξάνουμε κάθε φορά που δημιουργείται ένα νέο νήμα. Όταν η μεταβλητή αυτή γίνει μεγαλύτερη από τον αριθμό των νημάτων που θέτει ο χρήστης, ο κώδικας τότε παύει να εκτελείται παράλληλα και γίνεται σειριακά. Στην περίπτωση που ο κώδικας εκτελείται παράλληλα, ενεργοποιείται η φωλιασμένη επανάληψη με τον εξής τρόπο, omp_set_nested(enabledThreads<=numThreads) και ο αριθμός των νημάτων που θα την εκτελέσουν ως εξής omp_set_num_threads((numThreads>=enabledThreads)?numThreads-enabledThreads: 0).

Αξιοσημείωτο είναι το γεγονός πως η μεταβλητή enabledThreads, θα πρέπει κάθε χρονική στιγμή να αποδίδει το σωστό περιεχόμενο και για το λόγο αυτό, χρησιμοποιείται η εντολή #pragma omp flush() για την ανανέωση του περιεχομένου της.

Συνεχίζοντας, ορίζεται το parallel region, ο αριθμός των νημάτων που θα χρησιμοποιηθούν, οι private και shared μεταβλητές. Έπειτα η εντολή #pragma omp for schedule(static) nowait δηλώνει την παράλληλη εκτέλεση του βρόχου for με schedule static, καθώς η εργασία των νημάτων είναι ισομοιρασμένη. Ακόμη, το όρισμα nowait επιβάλει στα νήματα να τερματίζουν δίχως να περιμένουν τα υπόλοιπα.

Τέλος, λόγω της ανάγκης να αυξάνεται και να μειώνεται το περιεχόμενο της μεταβλητής enabledThreads ανάλογα με τα νήματα που εκτελούνται κάθε χρονική στιγμή, επιβάλλεται μέσω της εντολής #pragma omp atomic η ατομική μετατροπή του περιεχόμενου της, δηλαδή μόνο από ένα νήμα κάθε φορά.

3.3. Pthreads

Η παραλληλοποίηση με τη μέθοδο pthreads, επιβάλλει τη χρήση του header: <pthread.h>. Προκειμένου να μεταβάλλεται ο αριθμός των νημάτων όπως το επιθυμεί ο χρήστης, εισήγαμε ένα επιπλέον όρισμα στη main(), το argv[6]. Επιπρόσθετα, ορίσαμε δύο extern int μεταβλητές, εμφανείς από όλα τα αρχεία, εκ των οποίων η μία αφορά τον αριθμό των διαθέσιμων νημάτων που θα έχει κατά την εκτέλεση του προγράμματος (numThreads), ενώ η άλλη προσμετρά τα ενεργοποιημένα νήματα κάθε χρονική στιγμή (enabledThreads). Οι μεταβλητές αυτές δηλώνονται στο utils.h, καθώς επίσης και στην αρχή κάθε άλλου αρχείου, ενώ αρχικοποιούνται μία φορά στο test_octree.c.

Η υλοποίηση της pthreads απαιτεί τη χρήση συγκεκριμένων εντολών, μέσω των οποίων ορίζεται ο πίνακας των νημάτων που θα εκτελέσουν τον κώδικα προς παραλληλοποίηση. Έπειτα, δηλώνεται και καθορίζεται το attribute των νημάτων. Σημαντικό είναι πως σε όλη την έκταση του κώδικα, το attribute των νημάτων ορίζεται ως joinable προκειμένου να υπάρχει δυνατότητα να συγχρονιστούν.

Για να δημιουργηθεί ένα νήμα, είναι απαραίτητη η χρήση της συνάρτησης pthread_create(threads, attrib, function, arg). Συγκεκριμένα, στη δική μας περίπτωση που ο αριθμός των ορισμάτων της συνάρτησης function είναι μεγάλος, πρέπει να προχωρήσουμε στη δημιουργία δομής η οποία θα περιέχει τόσες μεταβλητές, όσες και τα ορίσματα που πρέπει να χρησιμοποιηθούν.

Όταν ένα νήμα ολοκληρώνει τη δουλειά που του έχει ανατεθεί, είναι σημαντικό να περιμένει τα υπόλοιπα νήματα προκειμένου να επιστρέψει στη main(). Για το λόγο αυτό καλείται η συνάρτηση pthread_join().

3.3.1. Hash codes, Morton encoding, Data rearrangement

Η λογική που ακολουθείται και στις τρεις συναρτήσεις κατά την υλοποίηση με pthreads είναι η ίδια. Συγκεκριμένα, επειδή παραλληλοποιείται ένας μόνο βρόχος for σε κάθε περίπτωση, δημιουργείται μία συνάρτηση που περιλαμβάνει τον κώδικα προς παραλληλοποίηση αλλά και η απαραίτητη δομή που θα περιέχει όπως αναφέρθηκε τα απαραίτητα ορίσματα.

Μέσα στη δομή αυτή, υπάρχει και μια ακόμη μεταβλητή που καθορίζει τον αριθμό των επαναλήψεων που θα εκτελέσει κάθε νήμα (chunk). Αυτό ορίζεται ως εξής: chunk=N/numThreads. Για να αποφευχθεί η περίπτωση όπου το chunk είναι μηδέν, υπάρχει ο αντίστοιχος έλεγχος που το θέτει ίσο με 1. Επιπλέον, λόγω του ότι η παραπάνω πράξη αποδίδει στο chunk μόνο το ακέραιο μέρος της διαίρεσης, το τελευταίο νήμα είναι πιθανό να έχει μεγαλύτερη δουλειά να εκτελέσει.

Τέλος, πριν περαστεί η δομή στην εκάστοτε συνάρτηση, απαιτείται η αρχικοποίηση των μεταβλητών της.

3.3.2. Radix sort

Όπως αναφέρθηκε και προηγουμένως, το σημείο που παραλληλοποίησης είναι ο βρόχος for, μέσα στον οποίο γίνεται η αναδρομική κλήση της συνάρτησης truncated_radix_sort(). Στη μέθοδο αυτή, υπάρχει μια δυσκολία στην υλοποίηση της παραλληλοποίησης καθώς ο αριθμός των νημάτων μπορεί να ξεπεράσει το όριο που θέτει ο χρήστης λόγω της συνεχούς αναδρομής. Ωστόσο, στη μέθοδο αυτή το πρόβλημα δε σταματά εκεί. Συγκεκριμένα, αντιμετωπίσαμε δυσκολία στην περίπτωση που ο χρήστης θέτει ως αριθμό νημάτων κάποιον αριθμό διαφορετικό του 8 και αυτό διότι τα ορίσματα που πρέπει να περαστούν για να εκτελεστεί σωστά η αναδρομή δεν αποθηκεύονται σωστά στη δομή των νημάτων που ορίσαμε. Για την επίλυση των παραπάνω προβλημάτων, αποφασίσαμε να παραλληλοποιούμε το σημείο αυτό μόνο στην περίπτωση που υπάρχουν διαθέσιμα νήματα περισσότερα από 8, προκειμένου να χρησιμοποιηθούν τα 8 από αυτά για την παραλληλοποίηση. Διαφορετικά, εάν τα διαθέσιμα νήματα είναι λιγότερα του 8 τότε ο κώδικας εκτελείται σειριακά. Ο τρόπος με τον οποίο προσμετράμε τα ενεργοποιημένα νήματα είναι ο εξής:

```
pthread_spin_lock(&lock);
       enabledThreads++;
pthread_spin_unlock(&lock);
```

Η μεταβλητή enabledThreads είναι δηλωμένη ως extern προκειμένου να μπορεί να χρησιμοποιηθεί από όλα τα νήματα ανά πάσα στιγμή.

Όσον αφορά τις εντολές `pthread_spin_lock(&lock);` και `pthread_spin_unlock(&lock);` επιλέξαμε να τις χρησιμοποιήσουμε έναντι του mutex, καθώς είναι προτιμότερες όταν το κλειδί χρησιμοποιείται για πολύ μικρό χρονικό διάστημα, όπως στην περίπτωση μας, που αυξομειώνεται ένας μετρητής. Επιπλέον, εάν κάποιο νήμα είναι σε αναμονή δε χρειάζεται να το ενεργοποιήσουμε προκειμένου να γίνει ξεκλείδωμα το κώδικα, αποφεύγοντας οποιαδήποτε χρονοκαθυστέρηση.

Ακόμη, αξίζει να ειπωθεί πως είναι αναγκαίο όπως και προηγουμένως να δημιουργηθεί πέρα από τη δομή όπου αποθηκεύονται οι τιμές που θα περαστούν ως ορίσματα κατά τη δημιουργία των νημάτων, και μια συνάρτηση η οποία θα εκτελεί την αναδρομή. Η συνάρτηση αυτή καλεί με τη σειρά την truncated_radix_sort() περνώντας έτσι τα σωστά ορίσματα κάθε φορά.

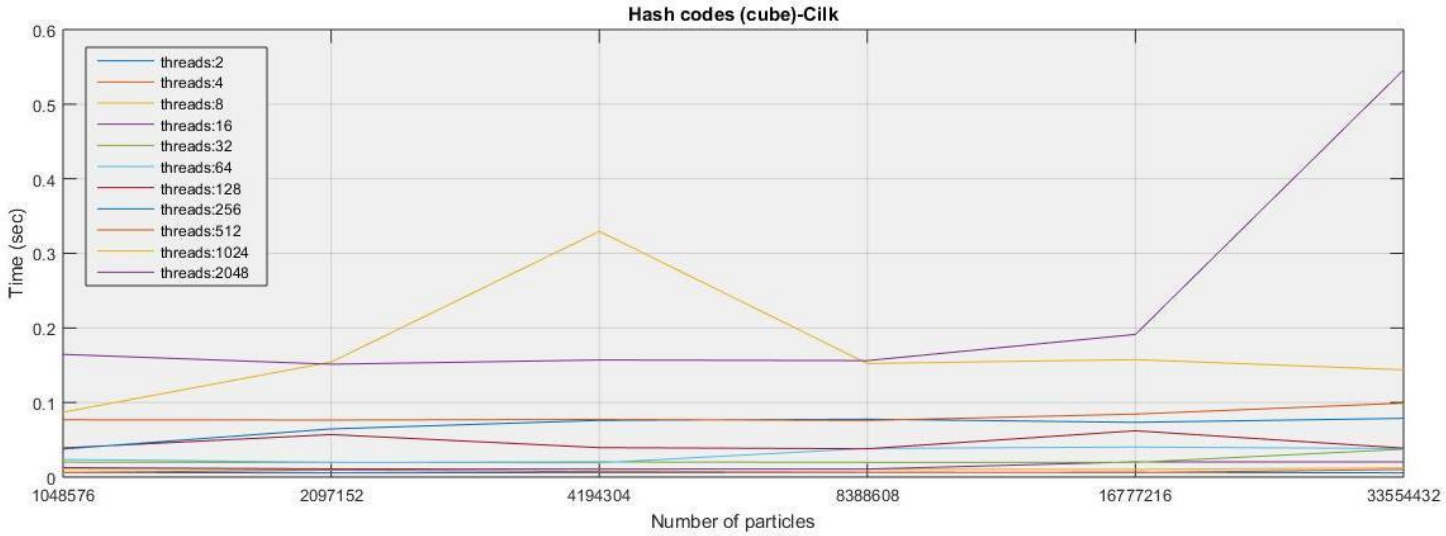
Τέλος, γνωρίζουμε πως εάν πραγματοποιούσαμε την παραλληλοποίηση αποκλειστικά για 8 νήματα θα είχαμε πιθανώς μια γρηγορότερη εκτέλεση της συνάρτησης αλλά δε θα ανταπεξέρχονταν στο ζητούμενο της εργασίας. Γι αυτό, επιλέξαμε να την παραλληλοποιήσουμε για όσο το δυνατόν περισσότερα νήματα δίνοντας μας τη δυνατότητα να τη συγκρίνουμε με τις άλλες υλοποιήσεις σε Cilk και OpenMP.

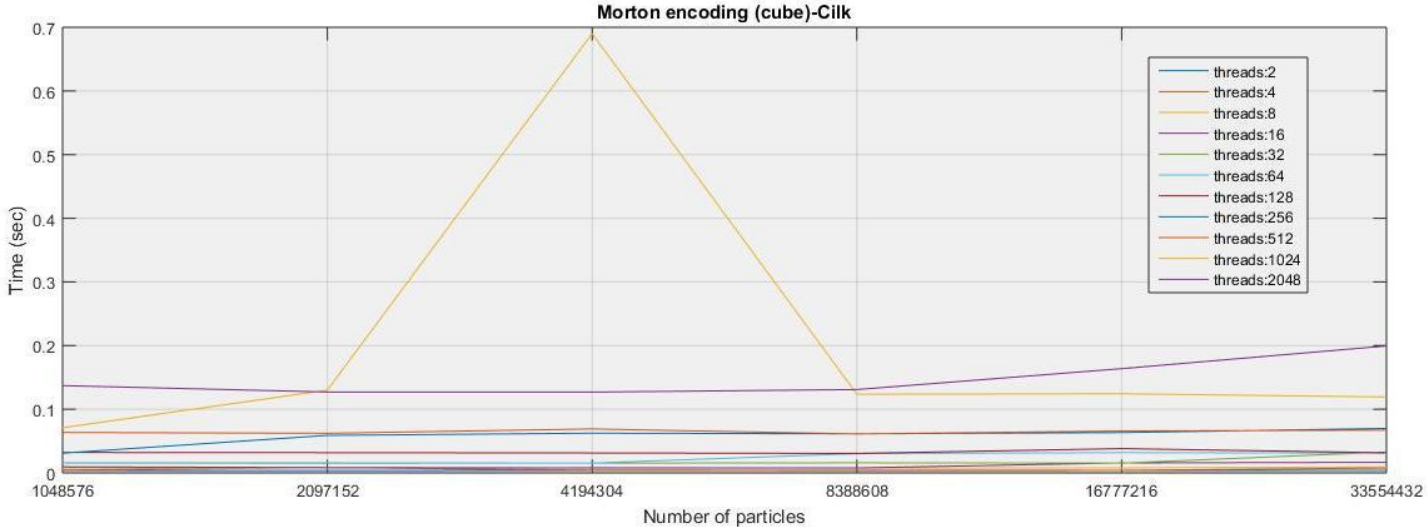
4. Σύγκριση Μεθόδων & Συμπεράσματα

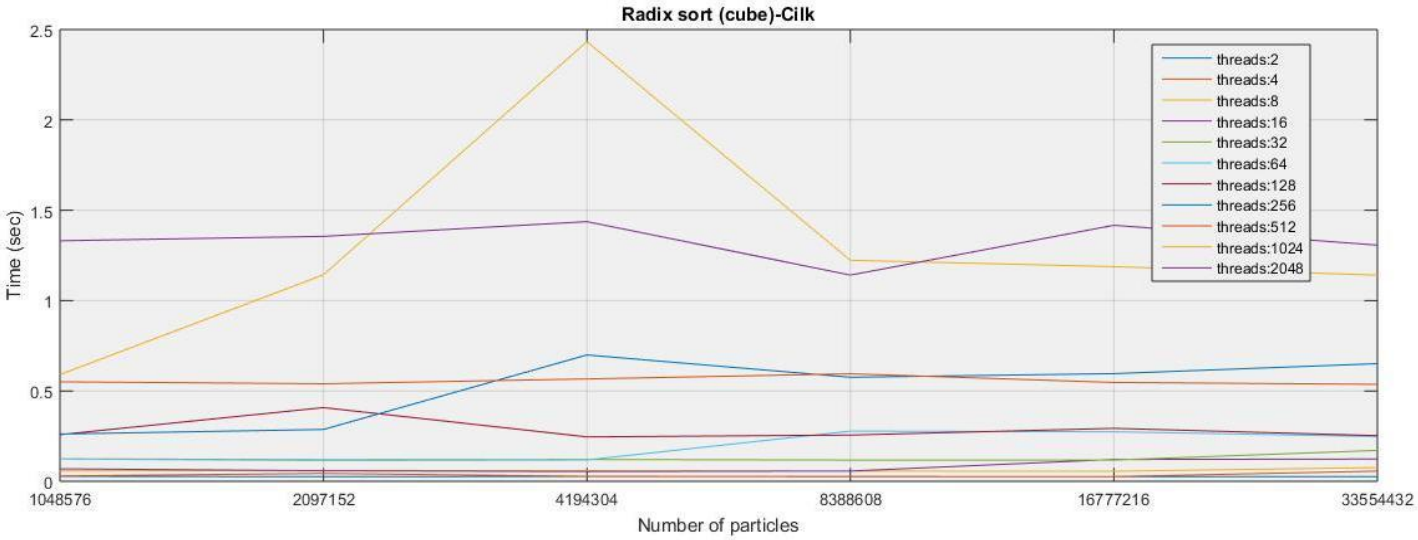
Έπειτα από την υλοποίηση της παραλληλοποίησης με κάθε μέθοδο, προχωρήσαμε στη σύγκριση των μεθόδων αυτών αναφορικά με την ταχύτητα εκτέλεσης των τεσσάρων προαναφερθέντων συναρτήσεων. Η ανάλυση μας πραγματοποιήθηκε λαμβάνοντας υπόψιν διαφορετικό αριθμό μεγίστου βάθους (L: 10, 14, 18) και κατώτατου πληθυσμού (S: 68, 128), συνολικού αριθμού σημείων (N: 1048576, 2097152, 4194304, 8388608, 16777216, 33554432) καθώς και νημάτων (T: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048). Ωστόσο, επιλέξαμε να παρουσιάσουμε σε μορφή διαγραμμάτων τους χρόνους για τις ακραίες τιμές του επιπέδου και του πληθυσμού, όπως ζητήθηκε στην άσκηση. Επίσης, επιλέξαμε τιμές των νημάτων στις οποίες η ταχύτητα της εκτέλεσης των συναρτήσεων παρουσιάζει μεγάλες αυξομειώσεις, προκειμένου να καλυφθεί το μεγαλύτερο μέρος της διαφορετικότητας των περιπτώσεων.

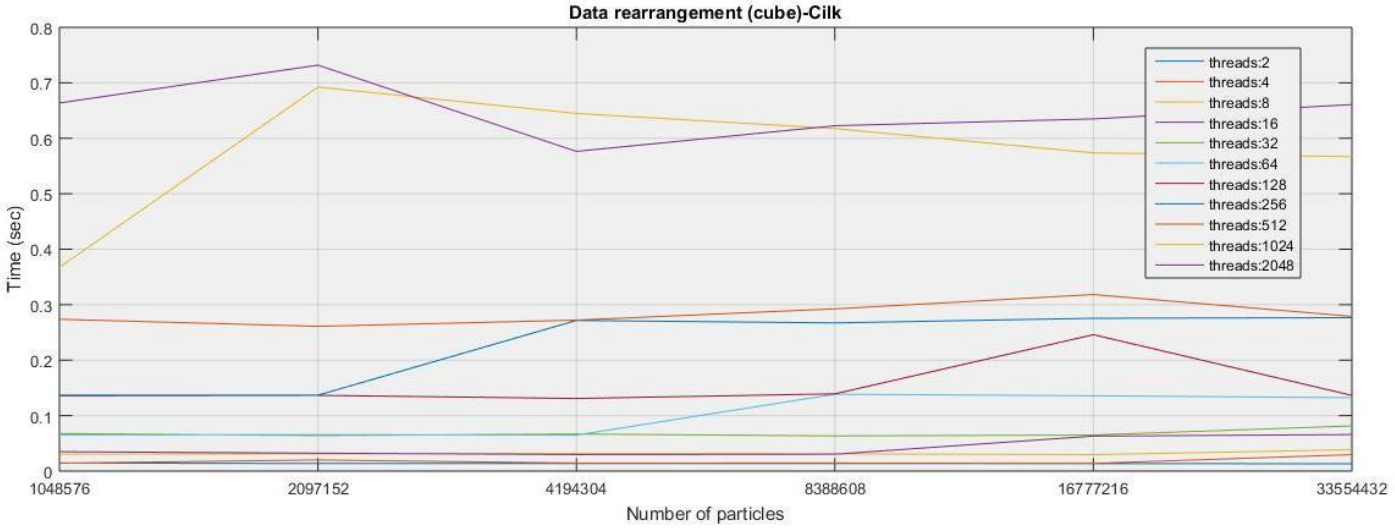
Η σύγκριση που πραγματοποιήσαμε χωρίζεται σε δύο επίπεδα. Το πρώτο αφορά την ταχύτητα εκτέλεσης των συναρτήσεων, όπου απεικονίζονται οι χρόνοι εκτέλεσης συναρτήσει του πλήθους των σημείων και των νημάτων, για κάθε μία μέθοδο. Σε δεύτερο επίπεδο, βασιζόμενοι στα αποτελέσματα που προέκυψαν από τα παραπάνω διαγράμματα, συγκρίναμε για κάθε συνάρτηση τους χρόνους κάθε υλοποίησης συμπεριλαμβανομένου και του σειριακού κώδικα, για συγκεκριμένο αριθμό μέγιστου επιπέδου (L: 18), κατώτατου πληθυσμού (S: 128), και ενδεικτικά τεσσάρων περιπτώσεων νημάτων (T: 2, 128, 512, 2048).

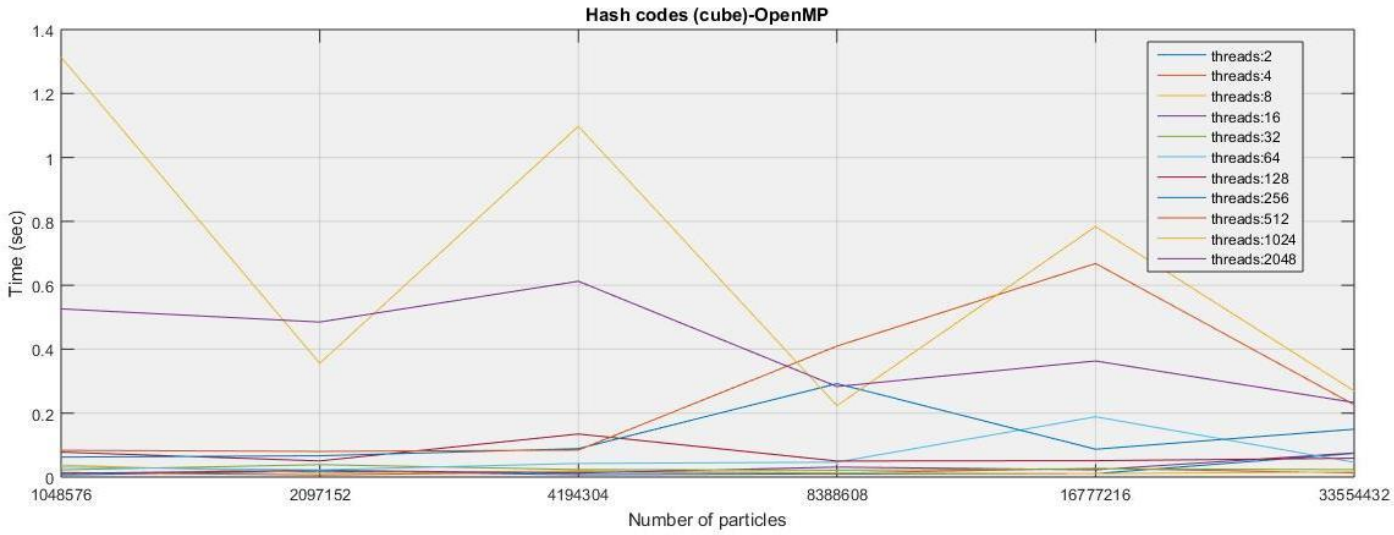
Παρακάτω παρουσιάζονται τα διαγράμματα. Εύκολα παρατηρείται πως η ανάλυση έγινε για κάθε μια συνάρτηση χωριστά προκειμένου να φανεί καθαρά το αποτέλεσμα της παραλληλοποίησης κάθε φορά. Αρχικά, παρουσιάζονται αυτά του κύβου, ενώ στη συνέχεια της σφαίρας.

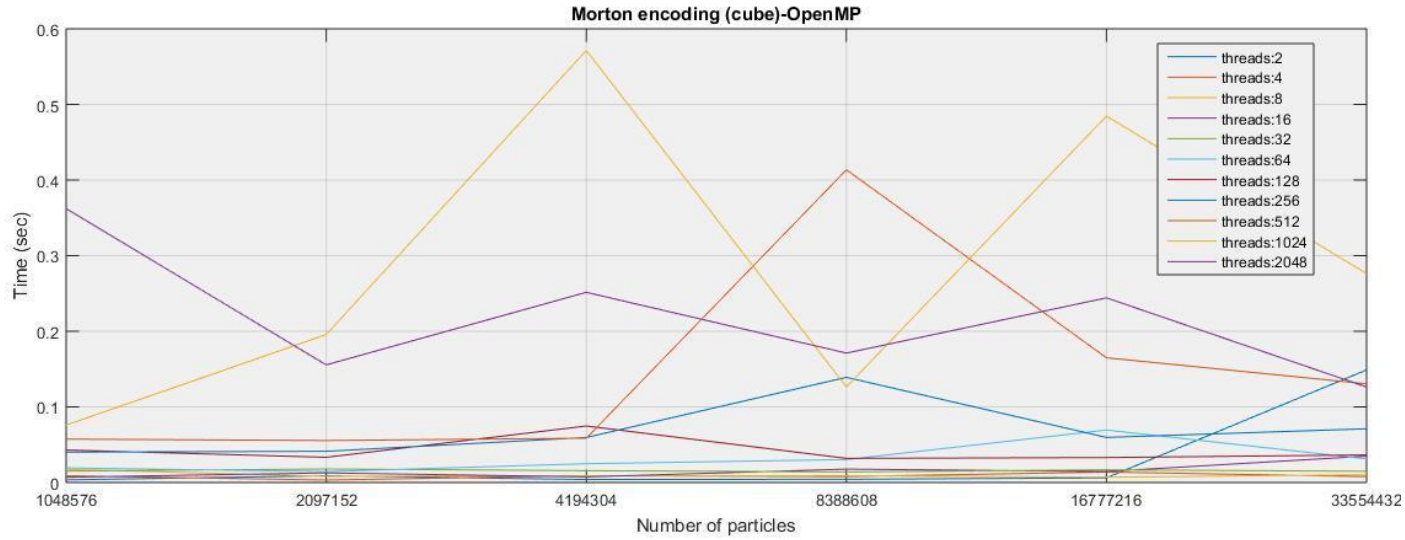


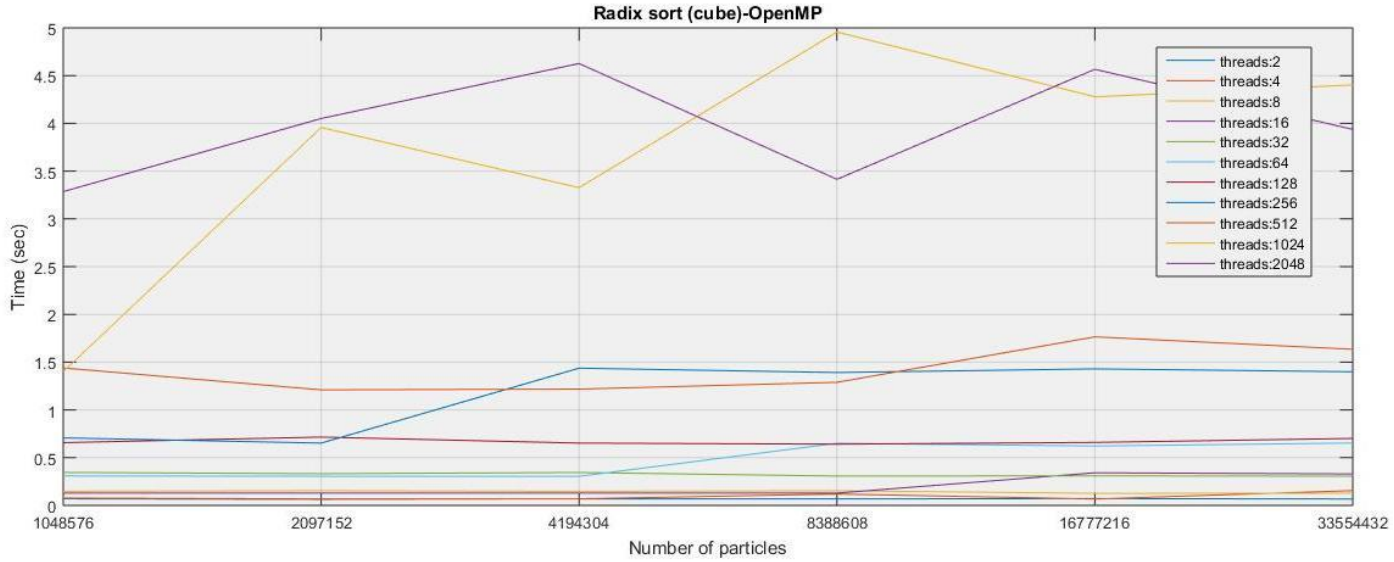

Morton encoding (cube)-Cilk
Time (sec)
Number of particles
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

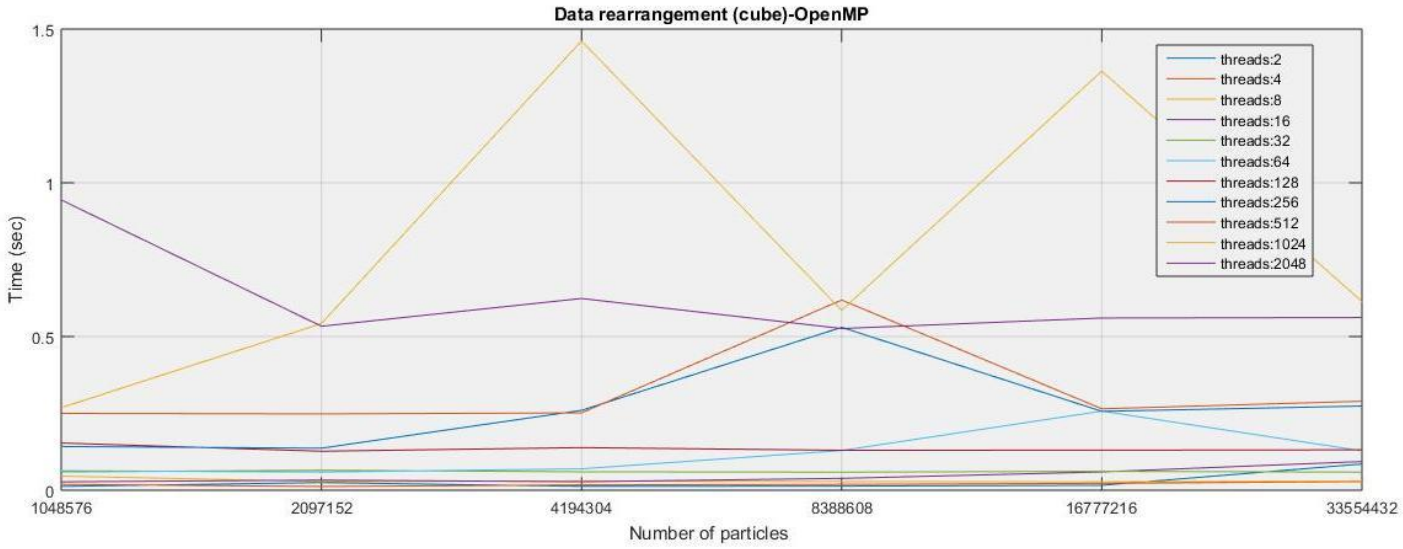

Radix sort (cube)-Cilk
Time (sec)
Number of particles
0
0.5
1
1.5
2
2.5
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

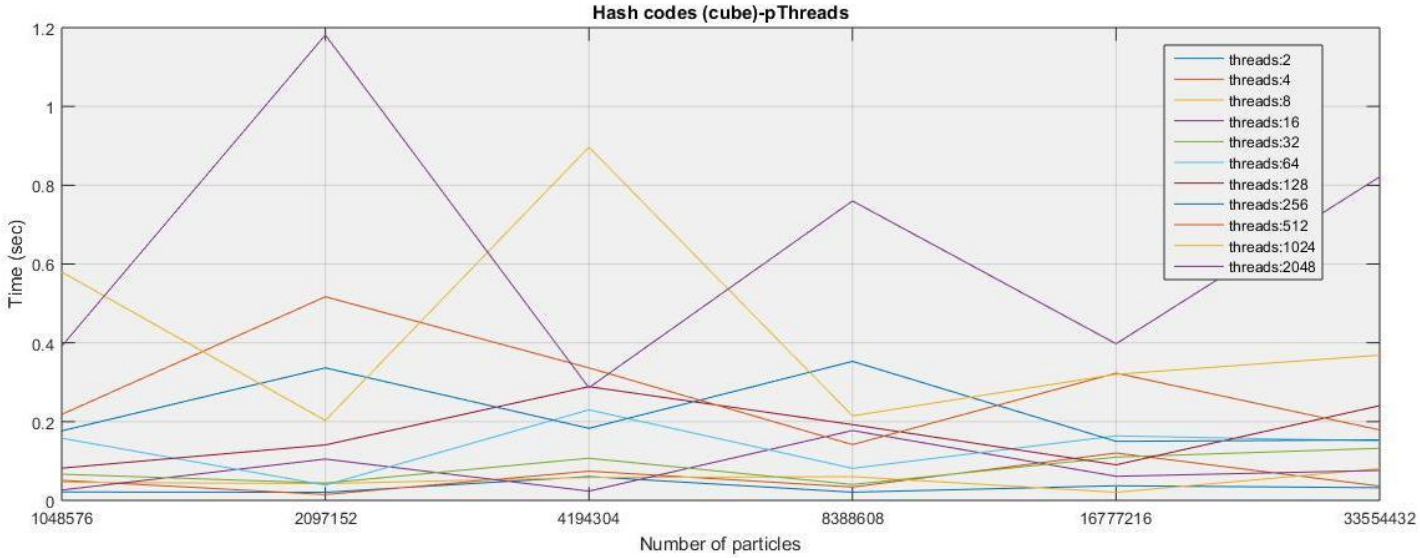

Data rearrangement (cube)-Cilk
Time (sec)
Number of particles
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

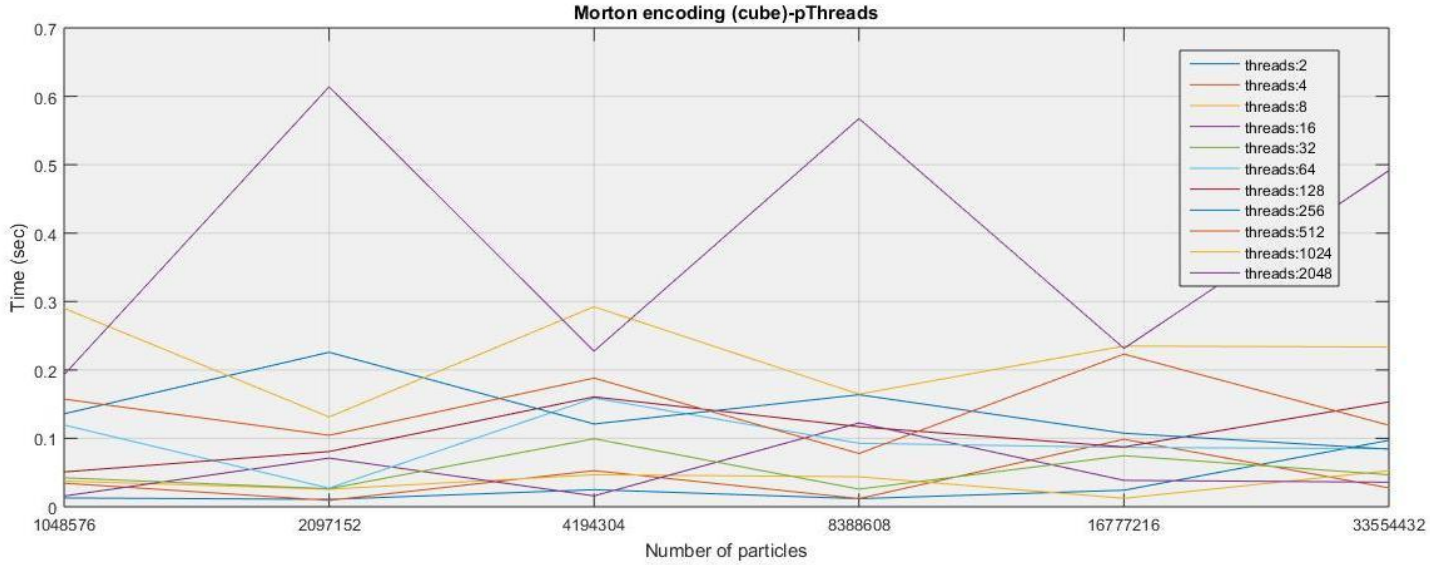

Hash codes (cube)-OpenMP
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

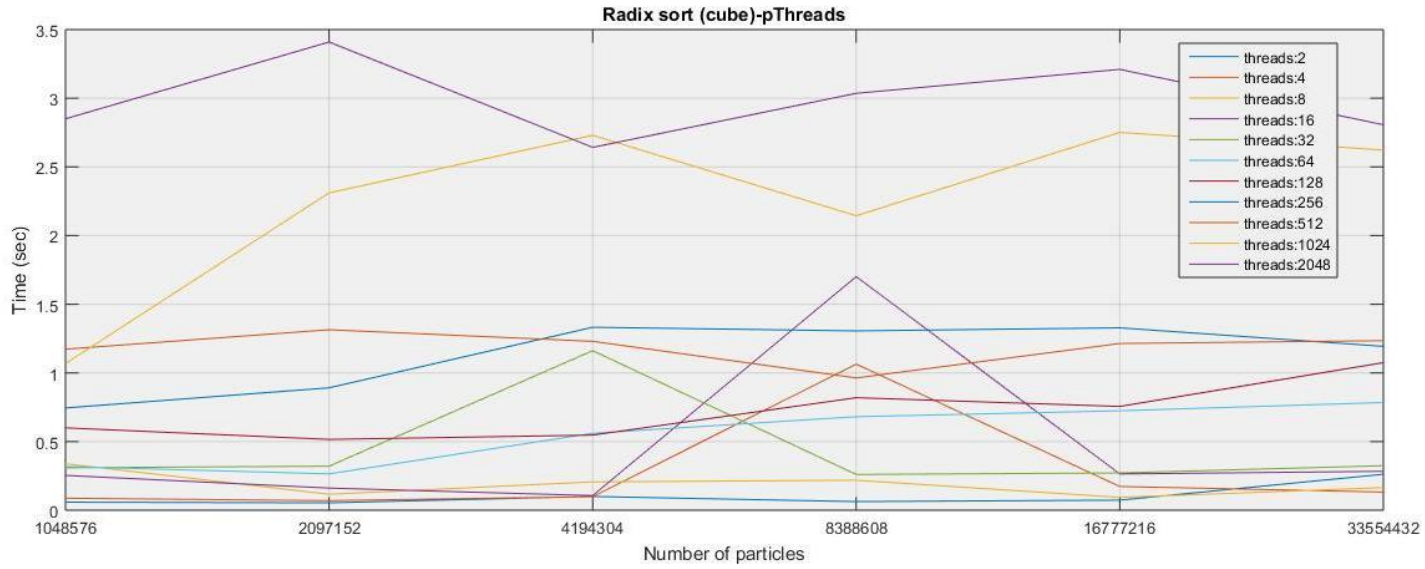

Morton encoding (cube)-OpenMP
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

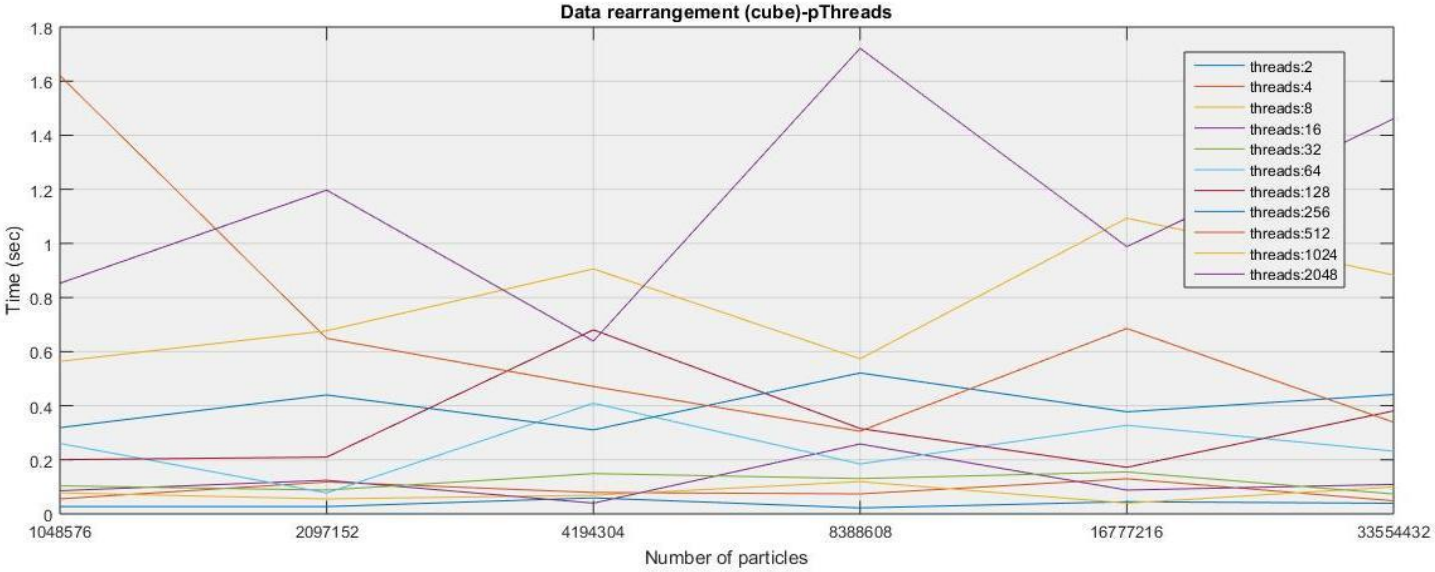

Radix sort (cube)-OpenMP
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

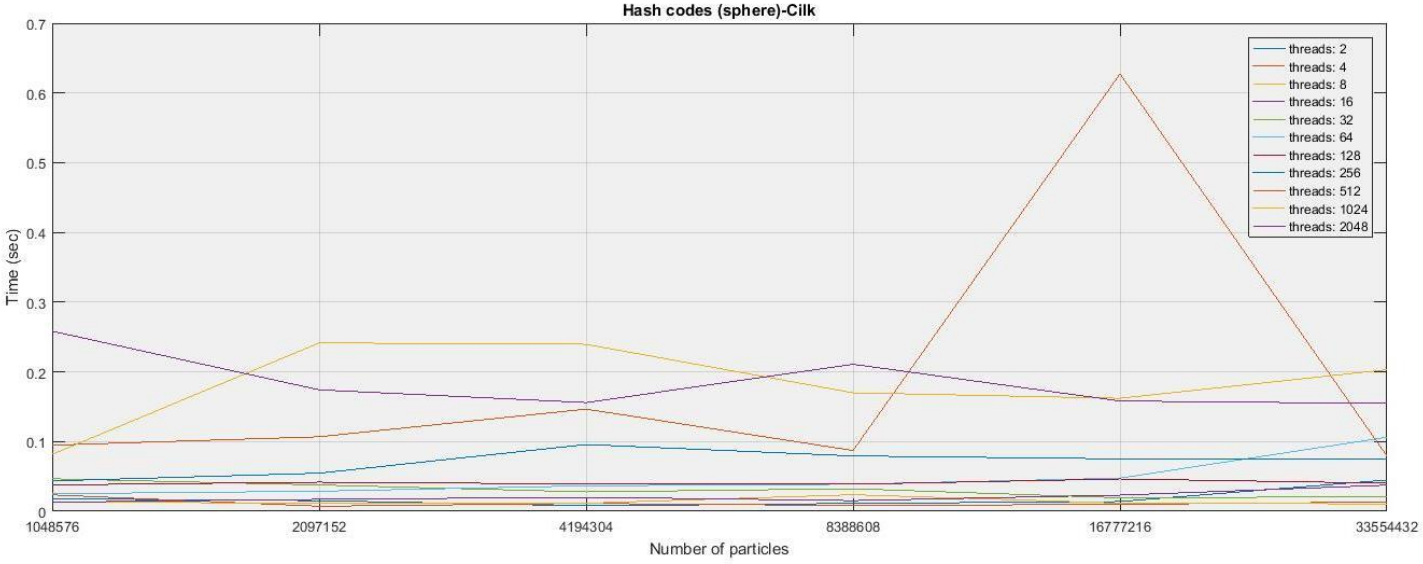

Data rearrangement (cube)-OpenMP
Time (sec)
Number of particles
1.5
1
0.5
0
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

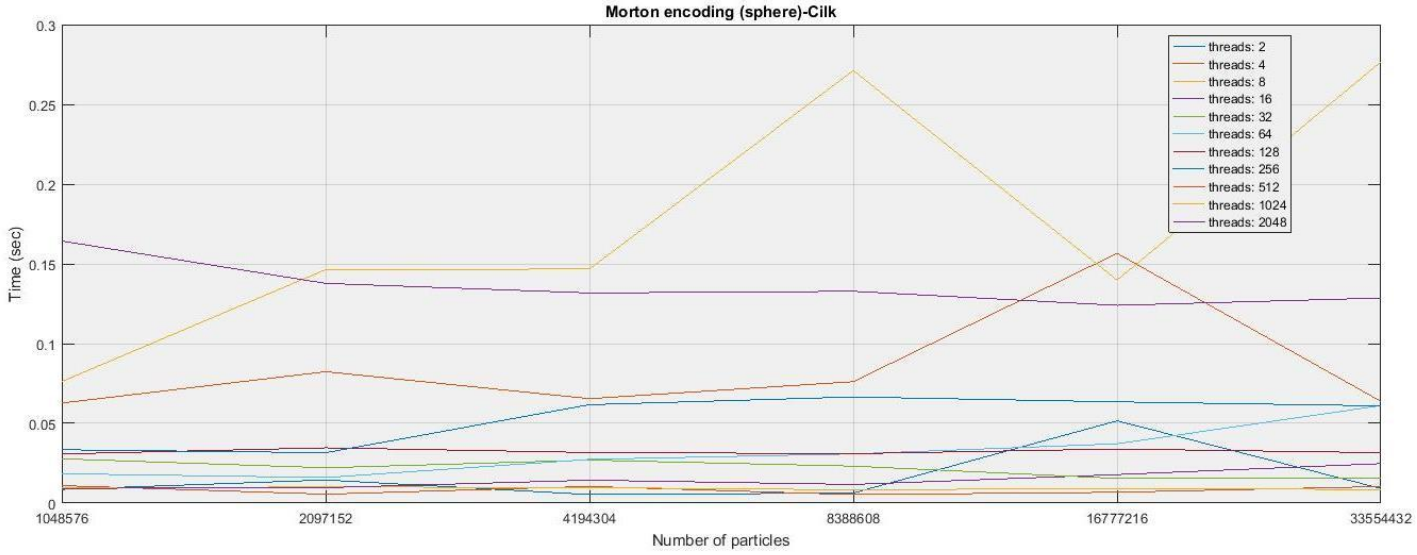

Hash codes (cube)-pThreads
Time (sec)
Number of particles
1.2
1
0.8
0.6
0.4
0.2
0
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

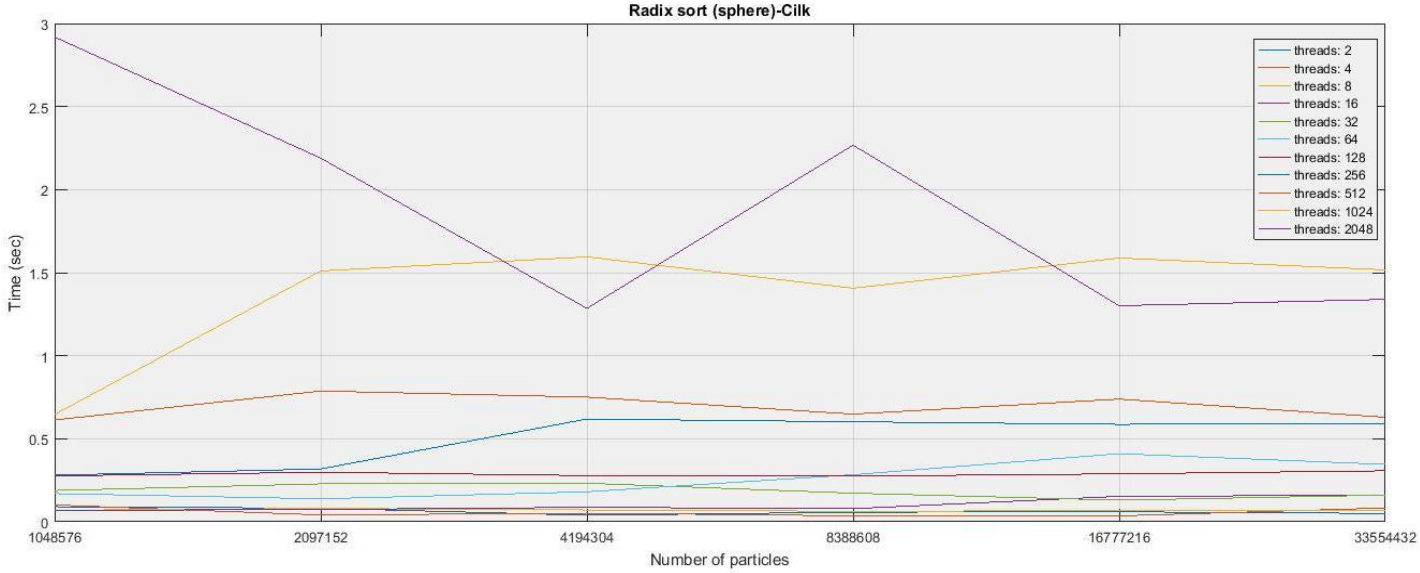

Morton encoding (cube)-pThreads
Time (sec)
Number of particles
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

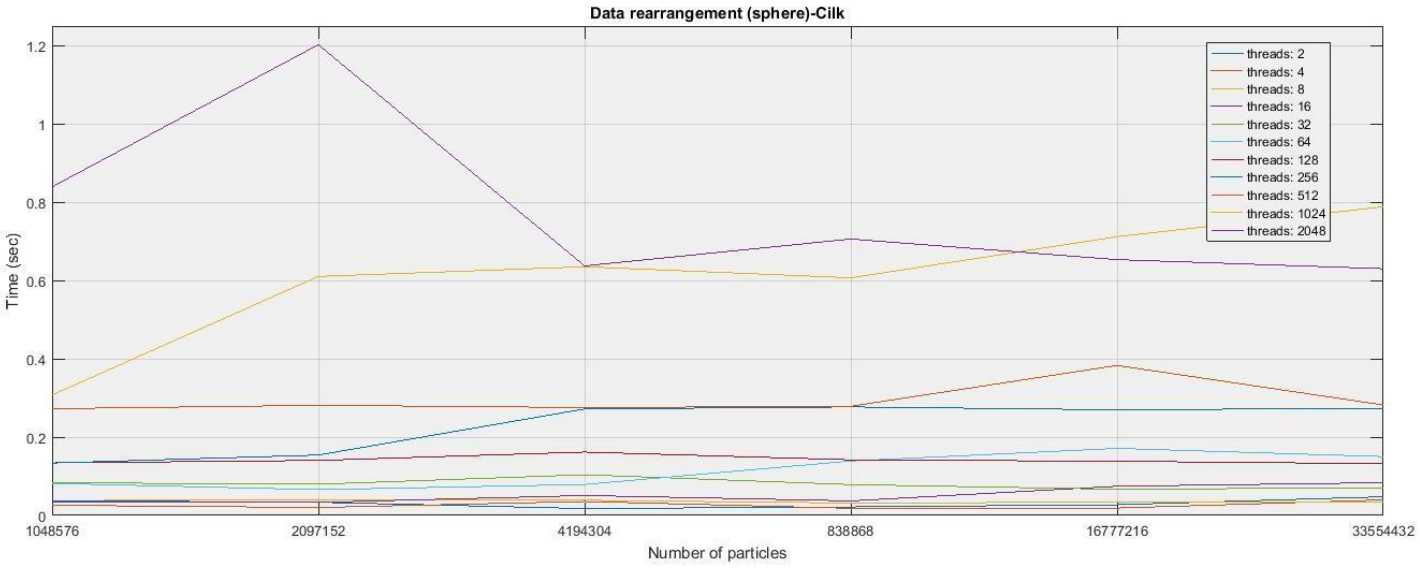

Radix sort (cube)-pThreads
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

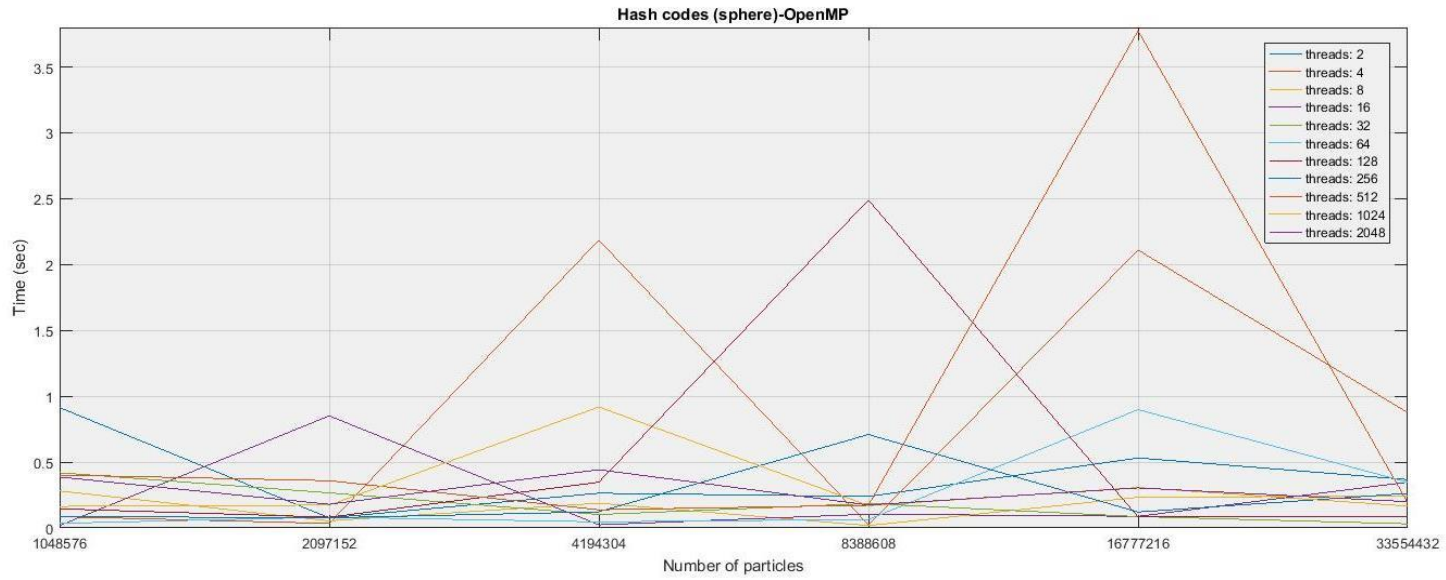

Data rearrangement (cube)-pThreads
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads:2
threads:4
threads:8
threads:16
threads:32
threads:64
threads:128
threads:256
threads:512
threads:1024
threads:2048

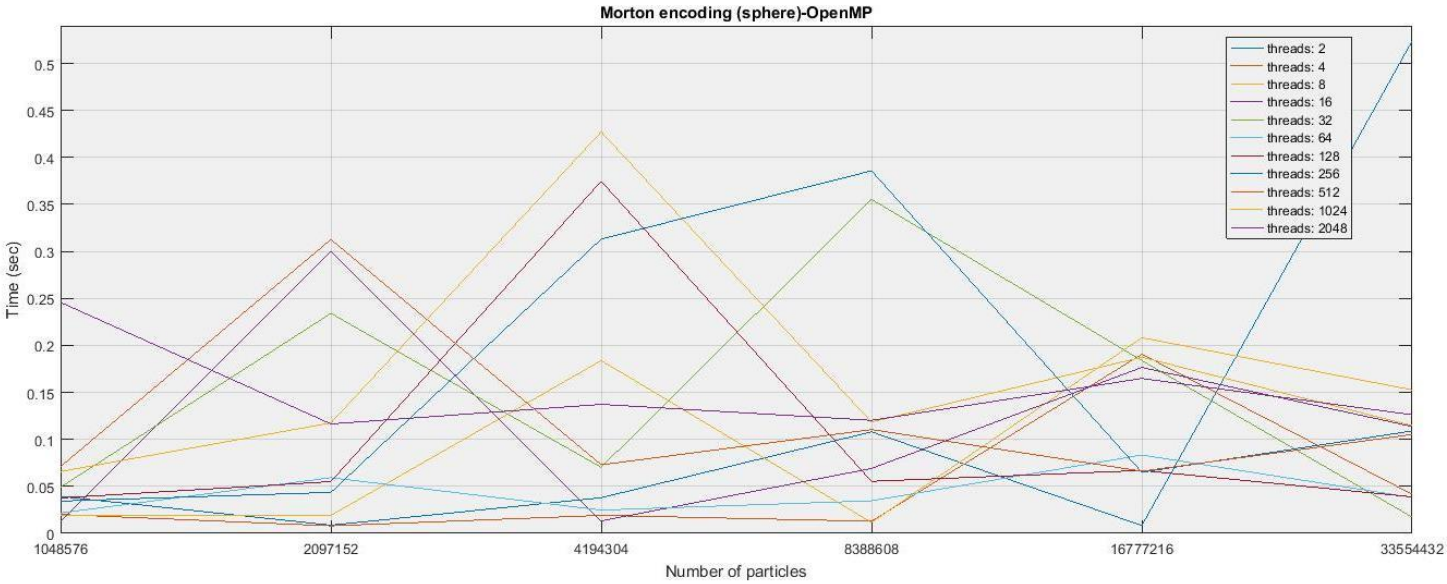

Hash codes (sphere)-Cilk
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

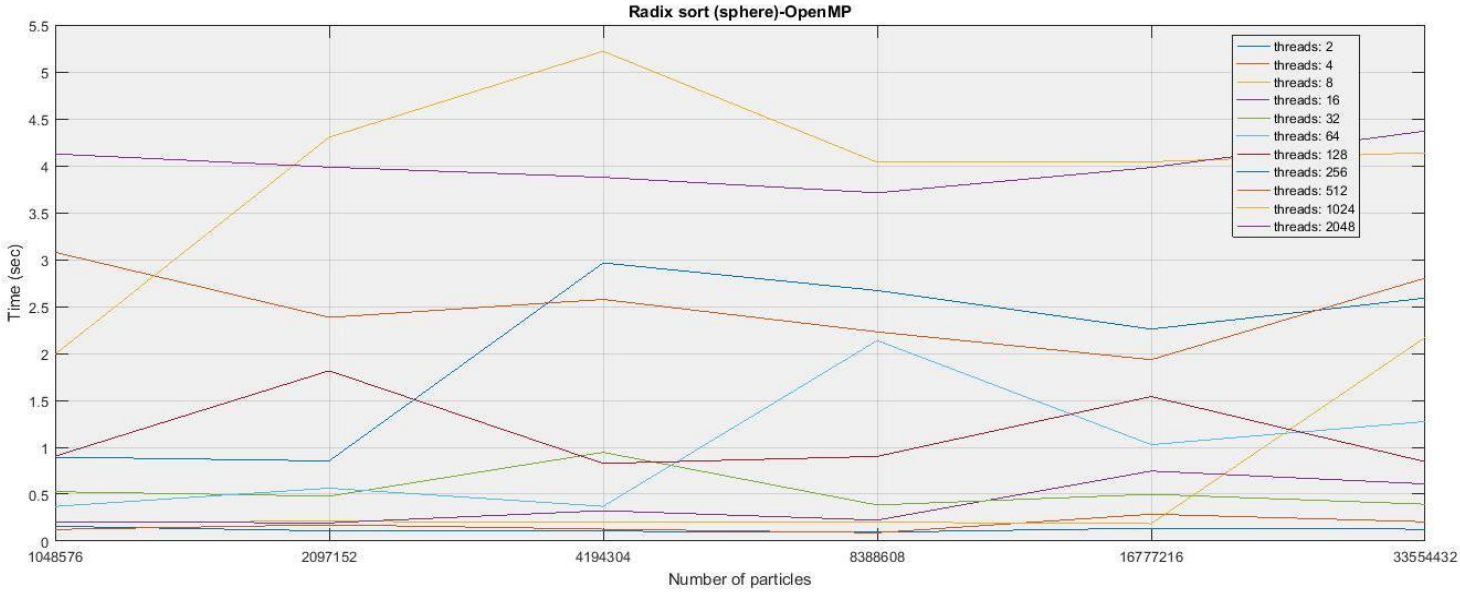

Morton encoding (sphere)-Cilk
Time (sec)
Number of particles
0.3
0.25
0.2
0.15
0.1
0.05
0
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

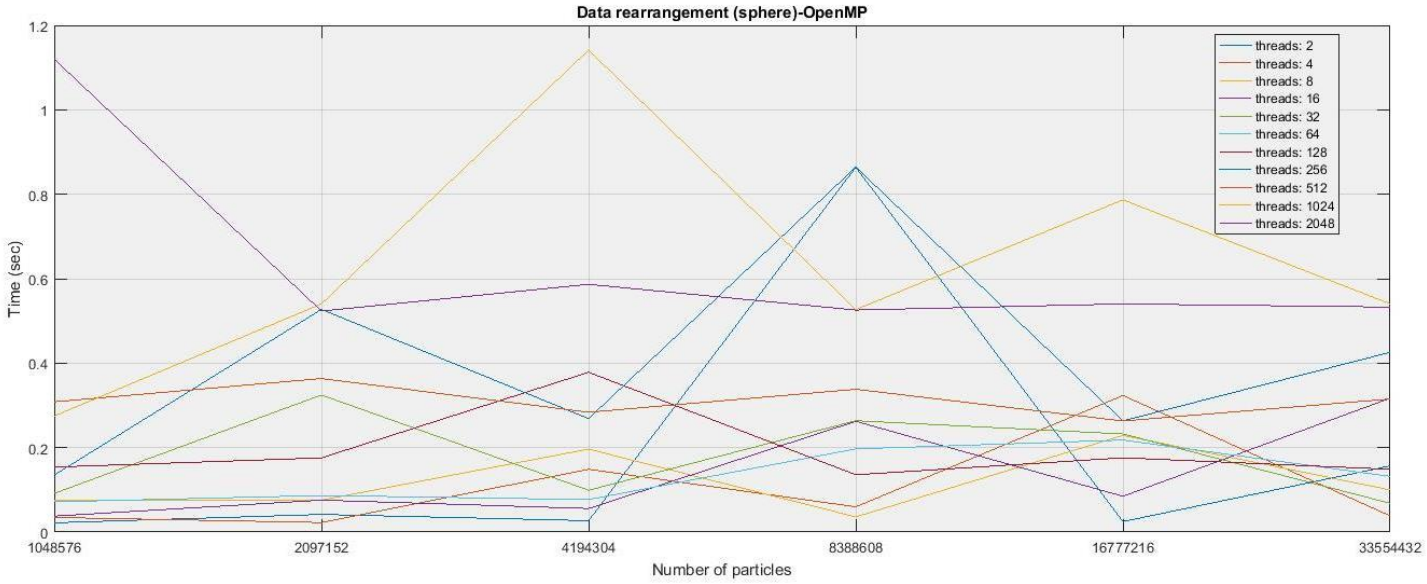

Radix sort (sphere)-Cilk
Time (sec)
Number of particles
3
2.5
2
1.5
1
0.5
0
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

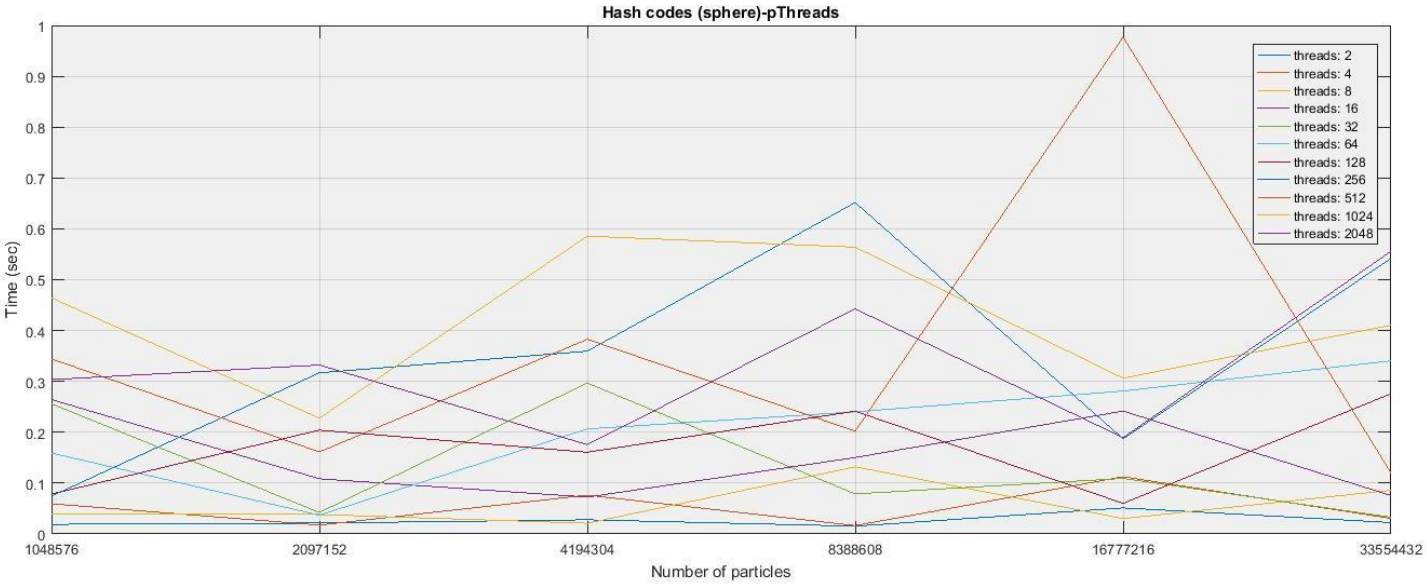

Data rearrangement (sphere)-Cilk
Time (sec)
Number of particles
1.2
1
0.8
0.6
0.4
0.2
0
1048576
2097152
4194304
838868
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

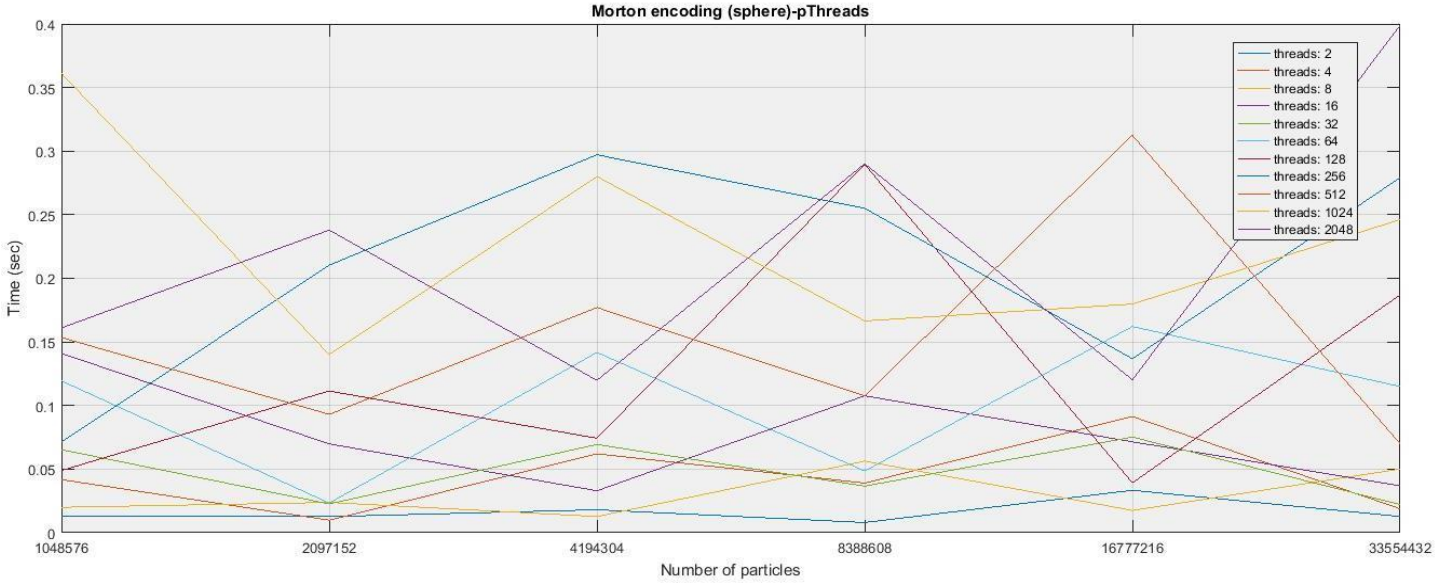

Hash codes (sphere)-OpenMP
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

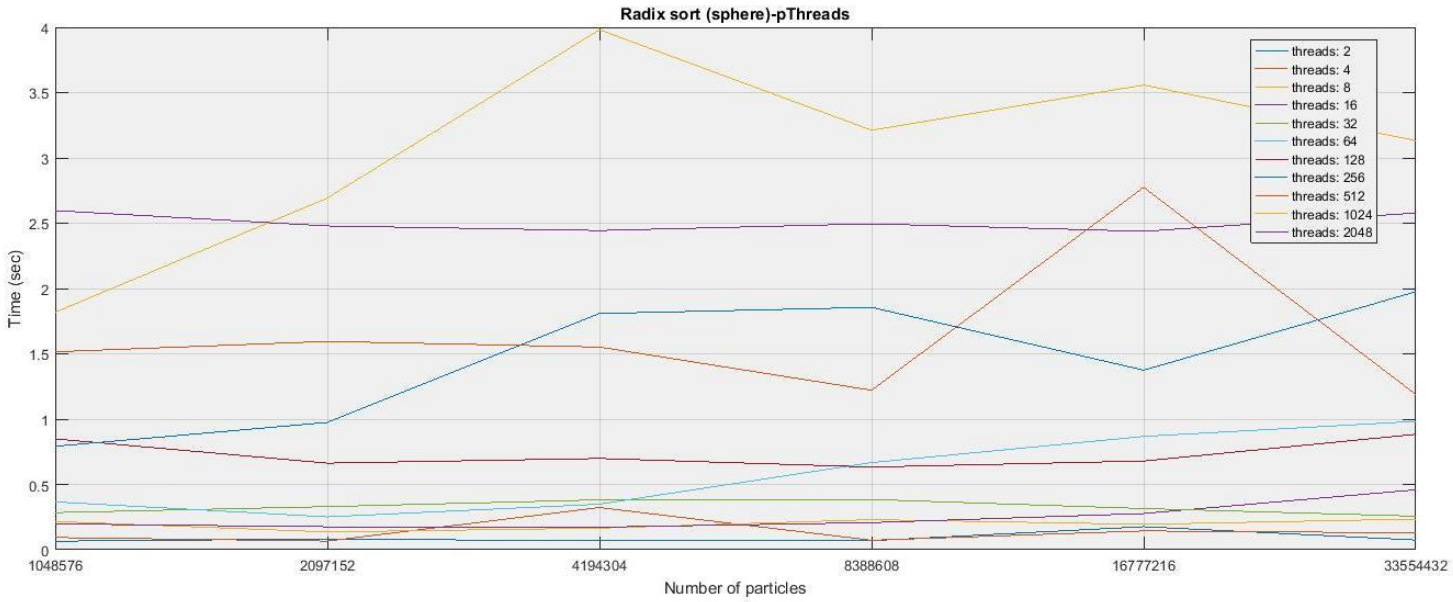

Morton encoding (sphere)-OpenMP
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

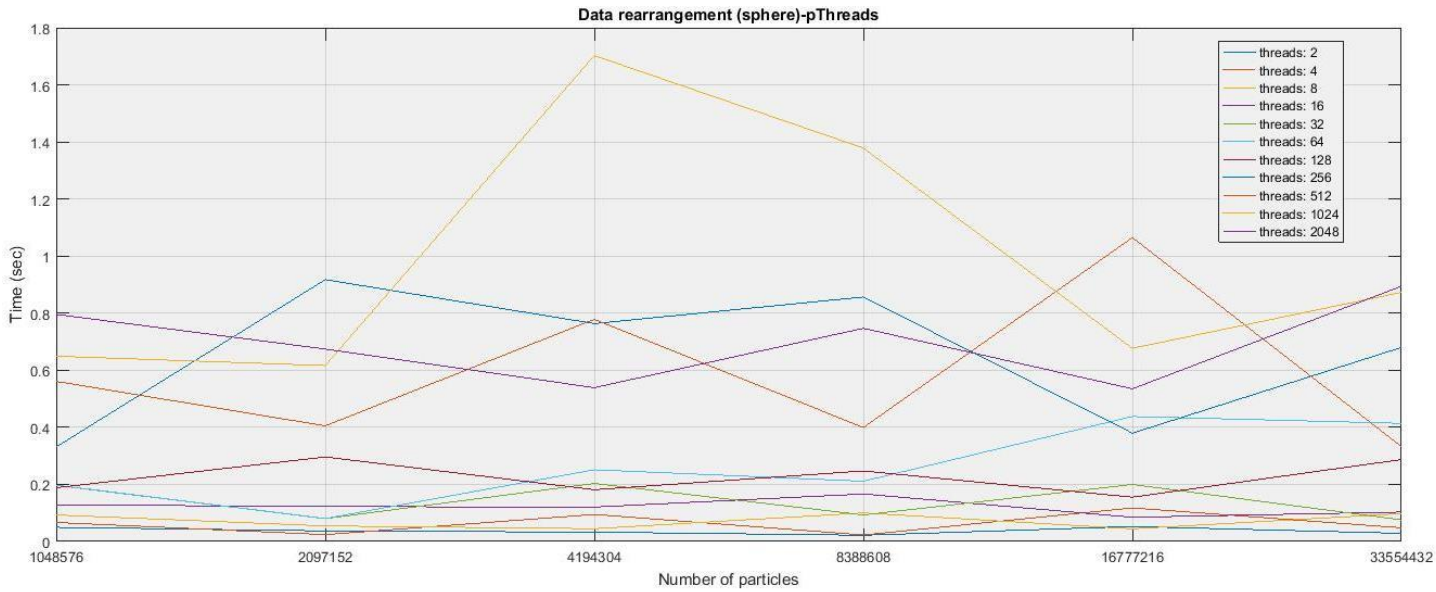

Radix sort (sphere)-OpenMP
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048


Data rearrangement (sphere)-OpenMP
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

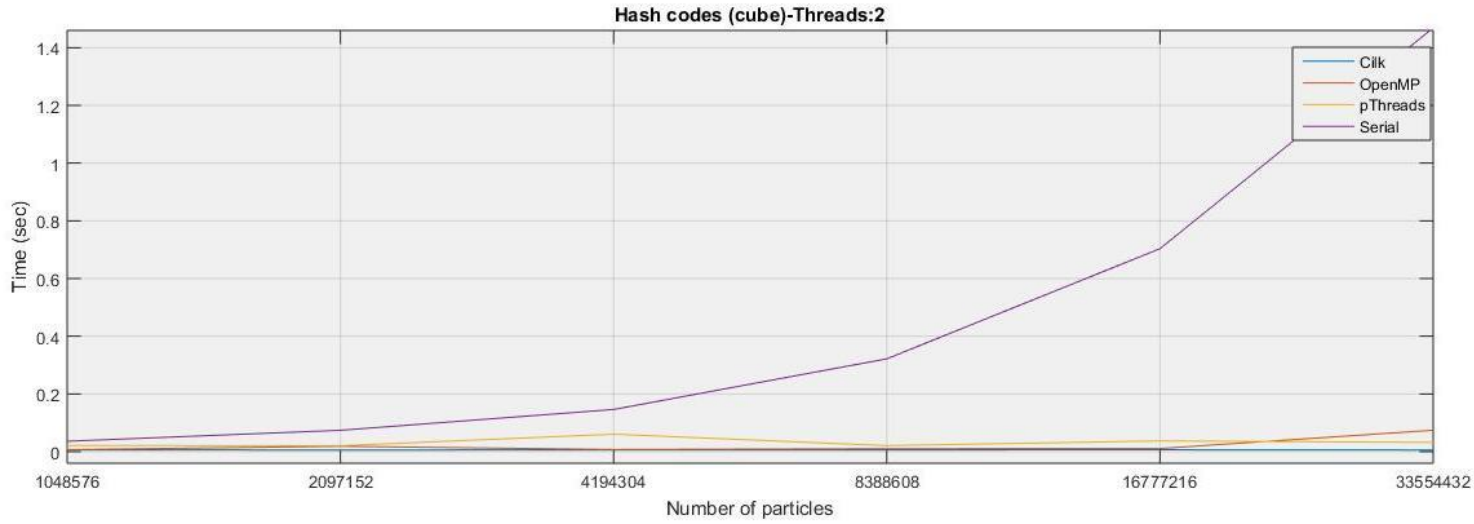

Hash codes (sphere)-pThreads
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

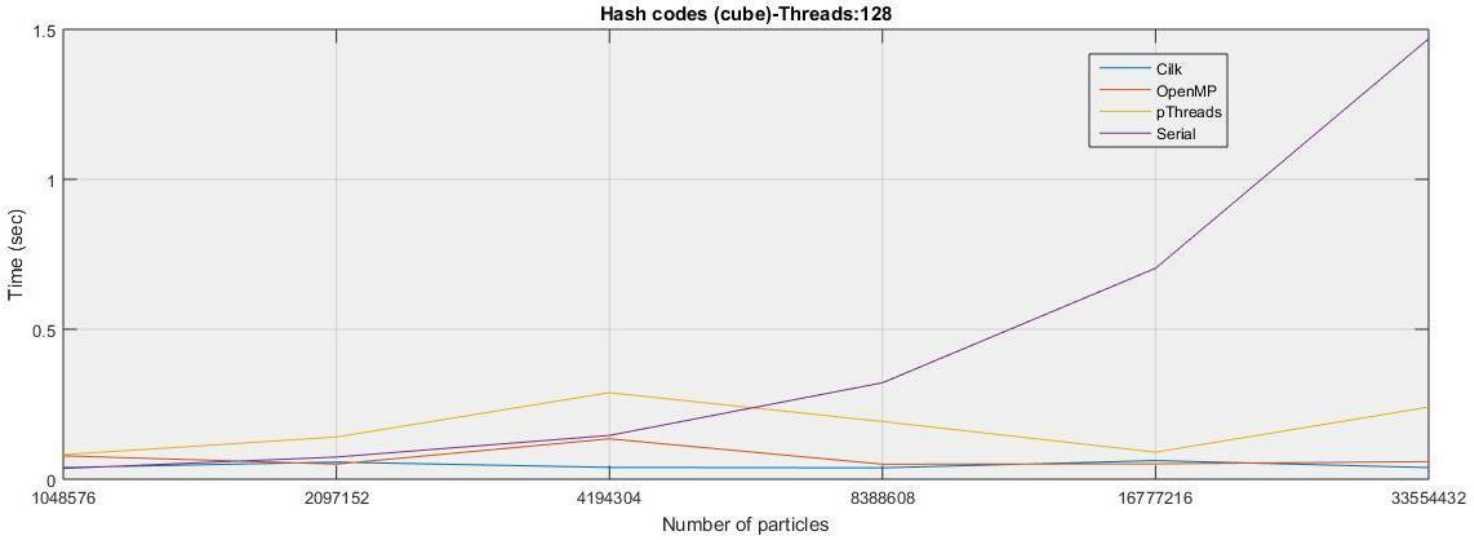

Morton encoding (sphere)-pThreads
Time (sec)
Number of particles
1048576
2097152
4194304
8388608
16777216
33554432
threads: 2
threads: 4
threads: 8
threads: 16
threads: 32
threads: 64
threads: 128
threads: 256
threads: 512
threads: 1024
threads: 2048

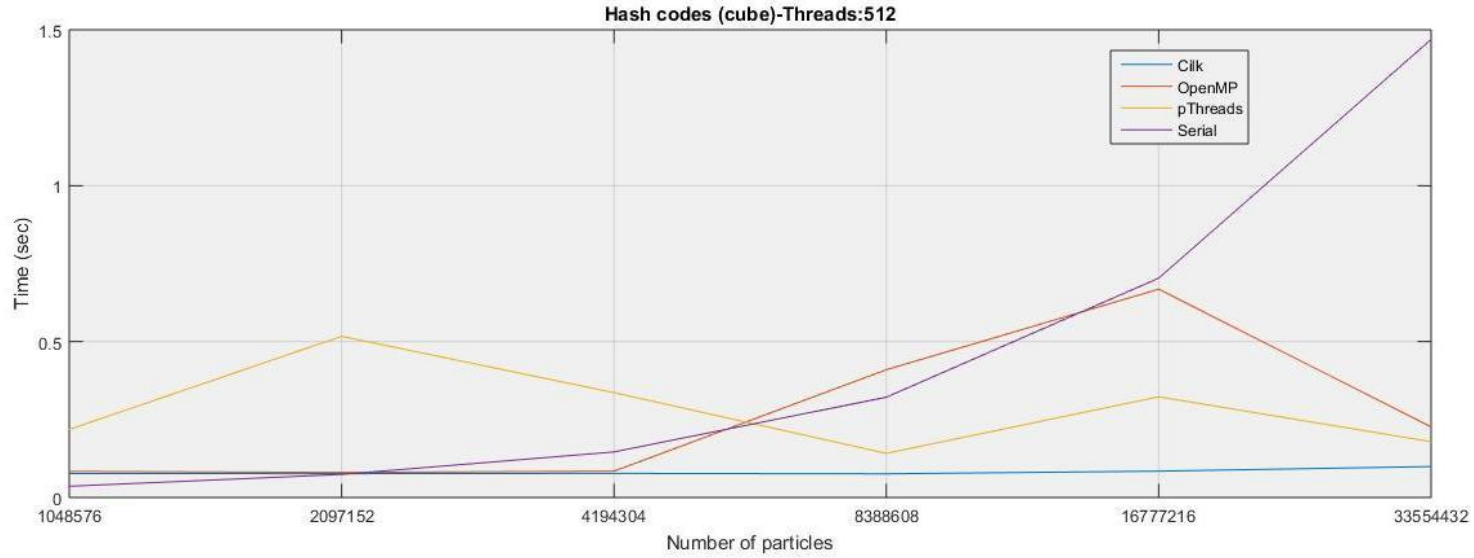


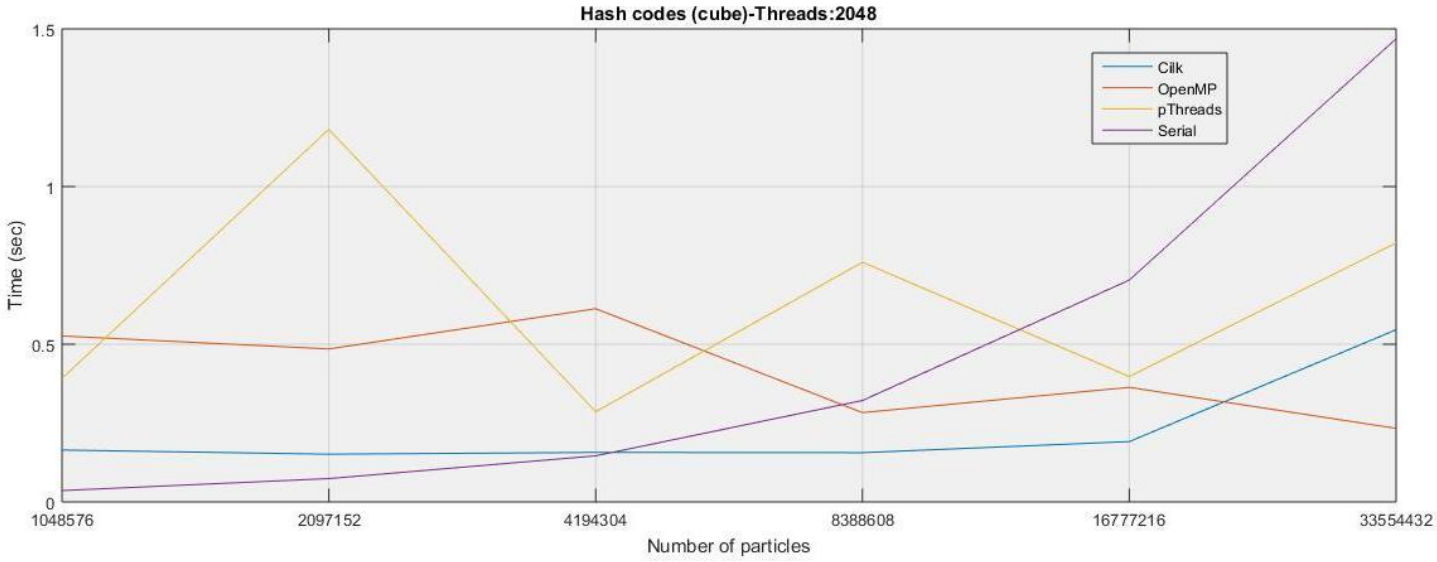


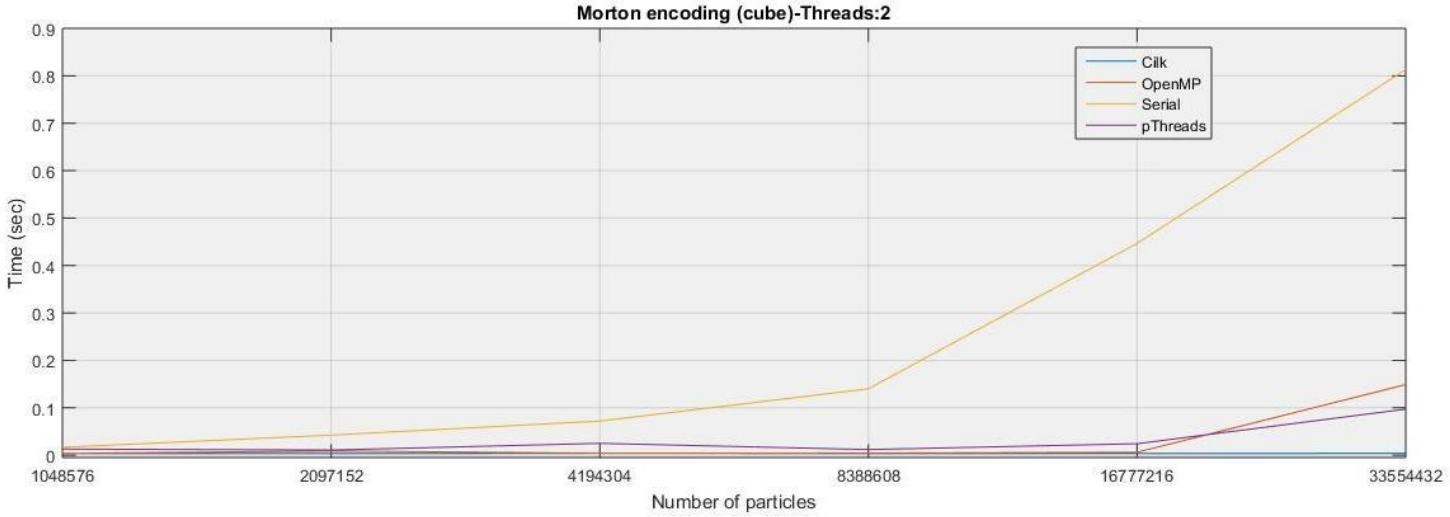
Από τα παραπάνω, γίνεται φανερό ότι καθώς αυξάνεται ο αριθμός των νημάτων σε κάθε περίπτωση ανεξαιρέτως, μειώνεται η απόδοση της παραλληλοποίησης. Σε αυτό το συμπέρασμα θα καταλήγαμε ούτως η άλλως, καθώς ο προτεινόμενος αριθμός νημάτων είναι N+1 σε περίπτωση που έχω N επεξεργαστές διαθέσιμους, έτσι ώστε να μη βρίσκεται κανένα νήμα σε αναμονή, αλλά ταυτόχρονα να έχω κι ένα διαθέσιμο σε περίπτωση που κάποιο δεσμεύεται για κατάσταση εισόδου/εξόδου από το χρήστη. Σε αντίθετη περίπτωση, που έχω δηλαδή περισσότερα από N+1 νήματα, θα υπάρξει και η αναμενόμενη καθυστέρηση επειδή αυτά θα προσπαθούν να «κλέψουν» τη δουλειά το ένα από το άλλο.

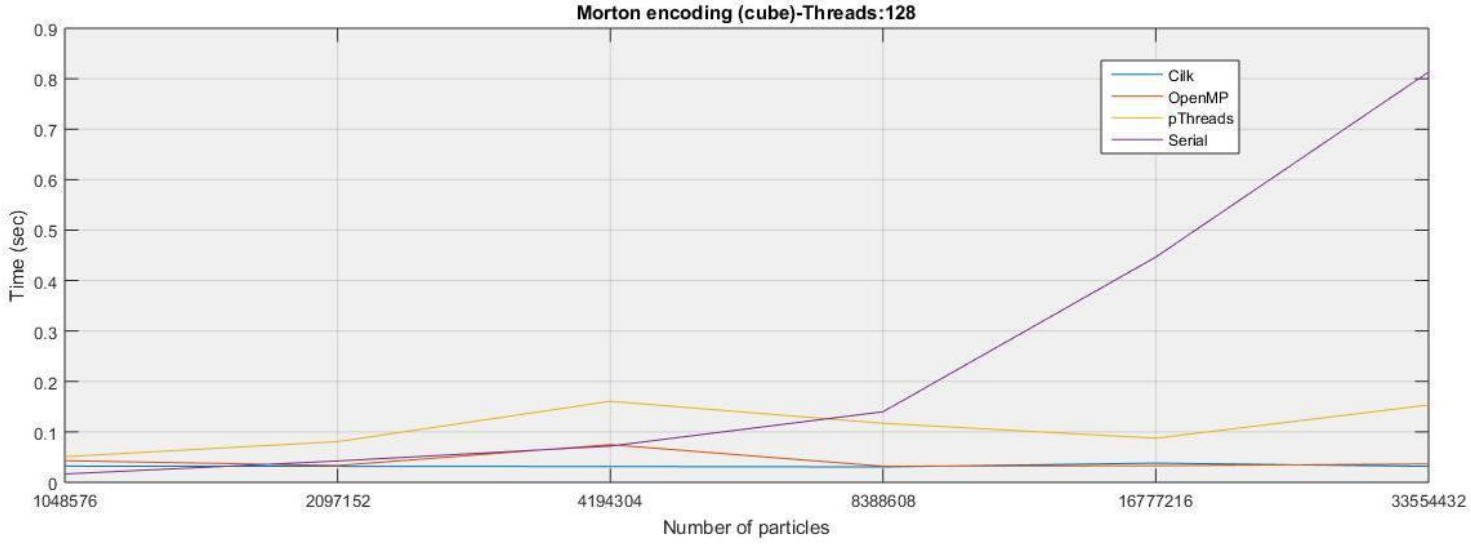

Hash codes (cube)-Threads:2
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

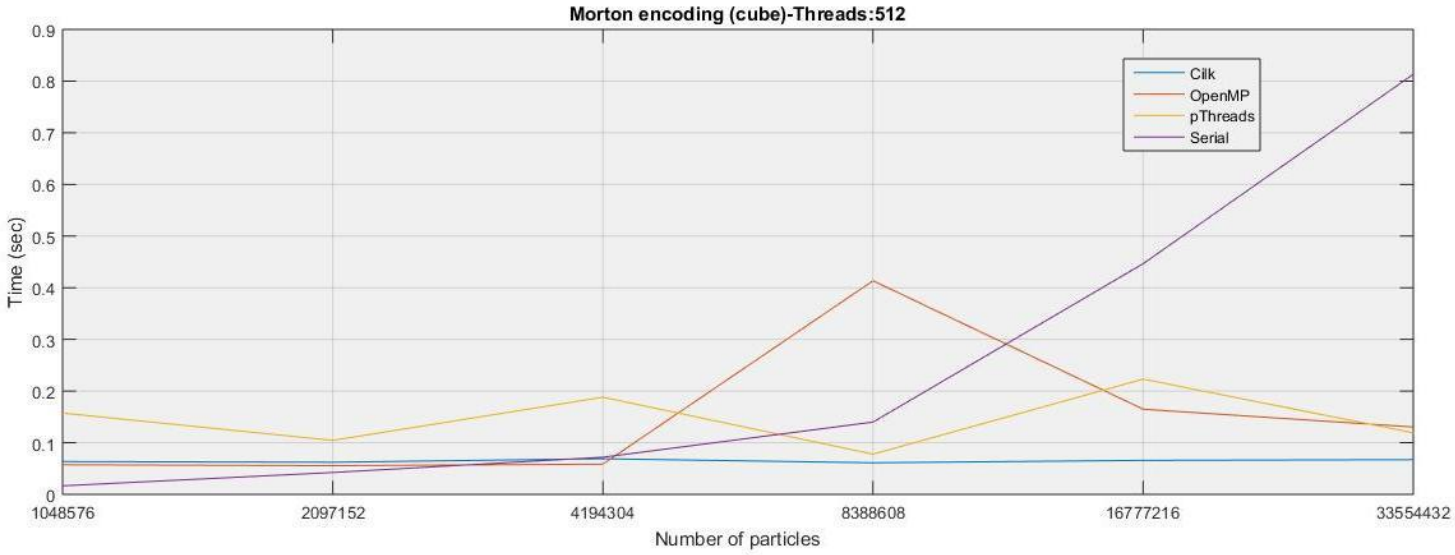

Hash codes (cube)-Threads:128
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

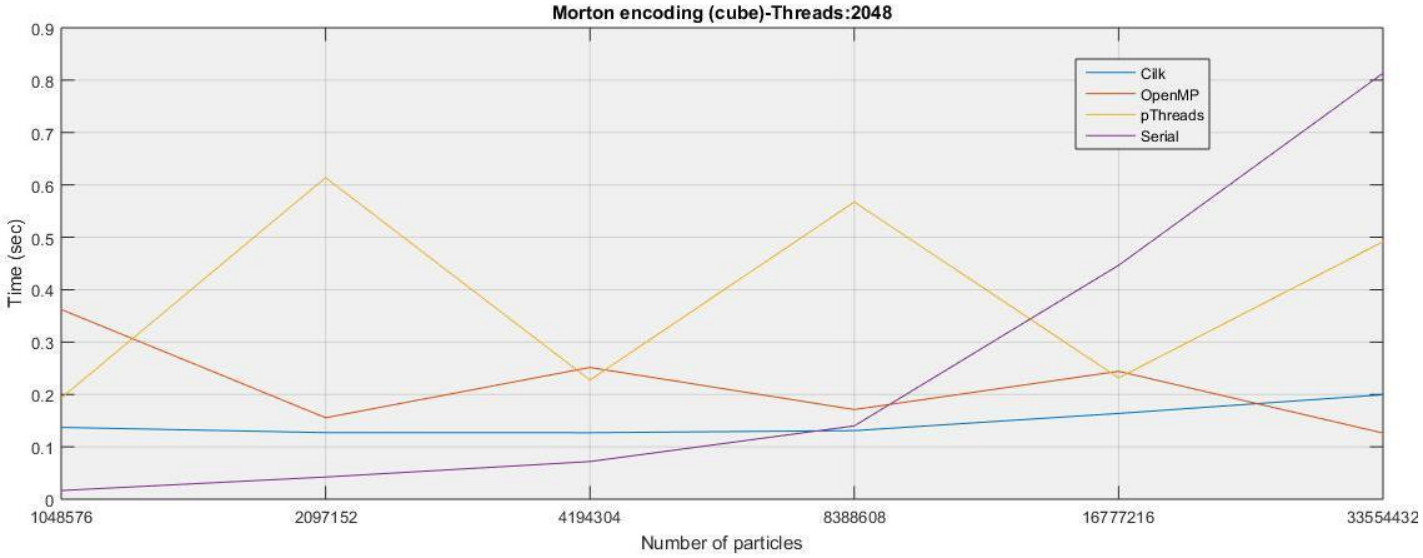

Hash codes (cube)-Threads:512
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

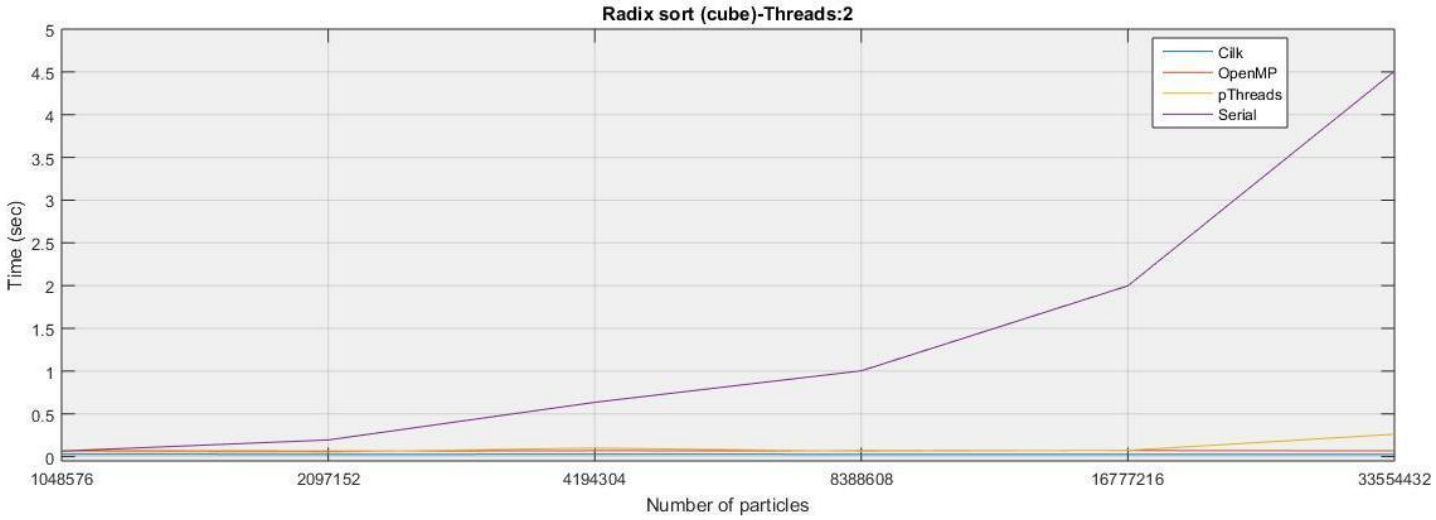

Hash codes (cube)-Threads:2048
Cilk
OpenMP
pThreads
Serial
Time (sec)
Number of particles
1.5
1
0.5
0
1048576
2097152
4194304
8388608
16777216
33554432

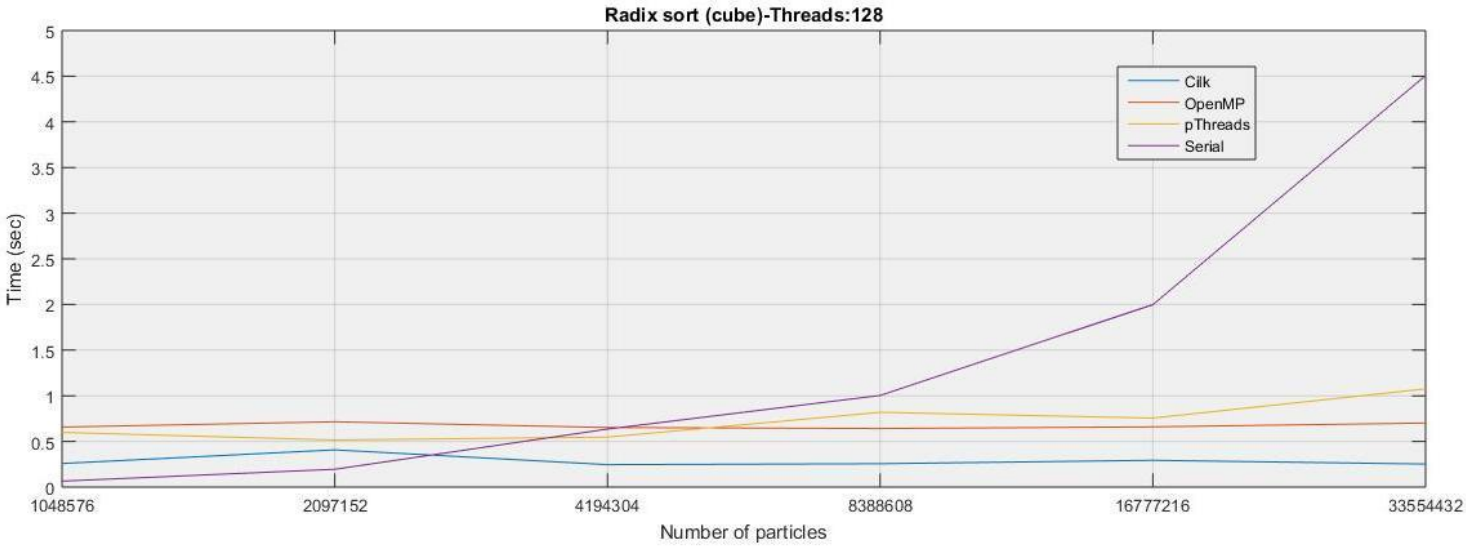

Morton encoding (cube)-Threads:2
Cilk
OpenMP
Serial
pThreads
Time (sec)
Number of particles
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1048576
2097152
4194304
8388608
16777216
33554432

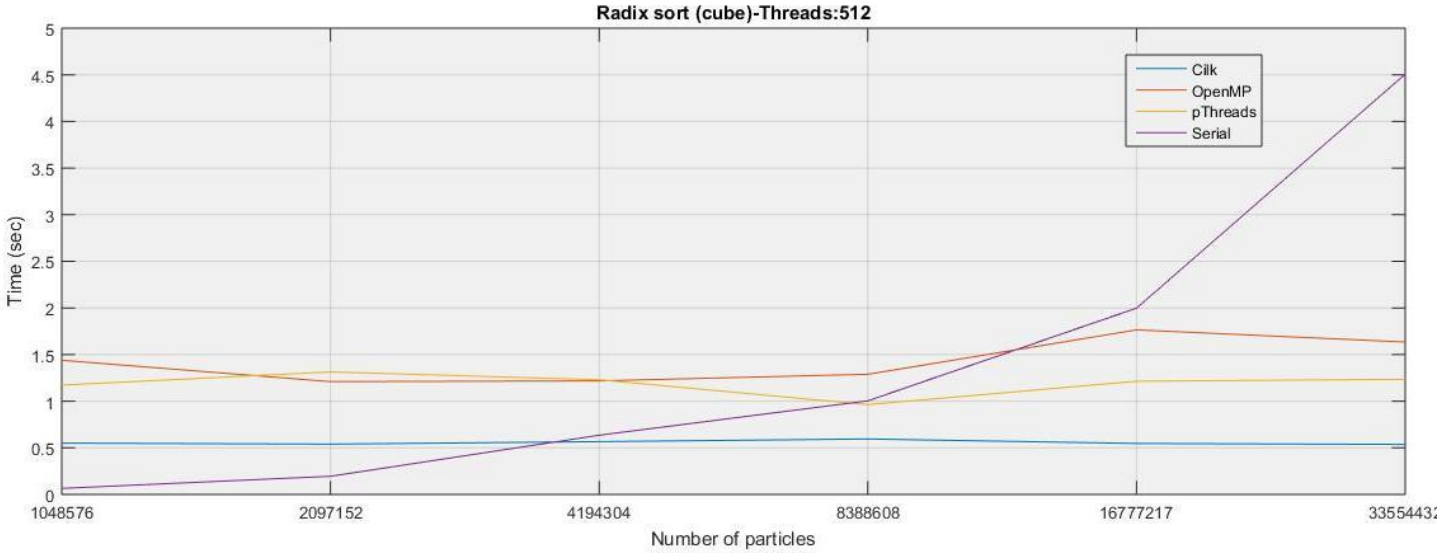

Morton encoding (cube)-Threads:128
Cilk
OpenMP
pThreads
Serial
Time (sec)
Number of particles
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1048576
2097152
4194304
8388608
16777216
33554432

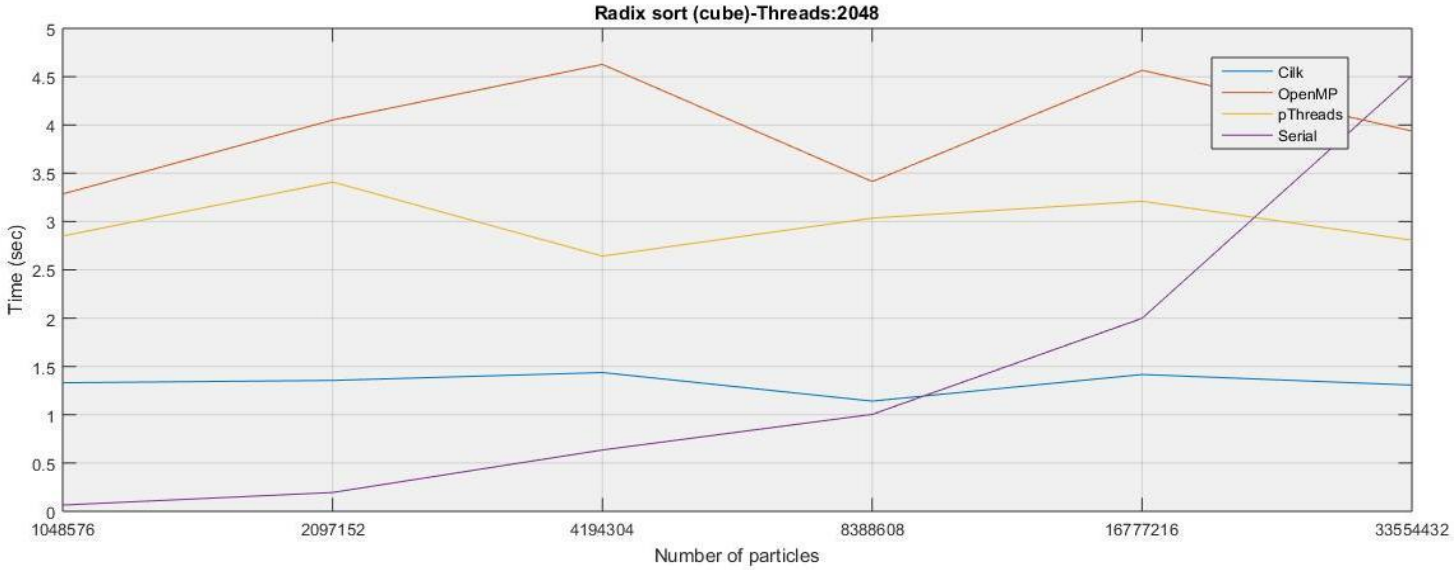

Morton encoding (cube)-Threads:512
Time (sec)
Number of particles
Cilk
OpenMP
pThreads
Serial
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1048576
2097152
4194304
8388608
16777216
33554432

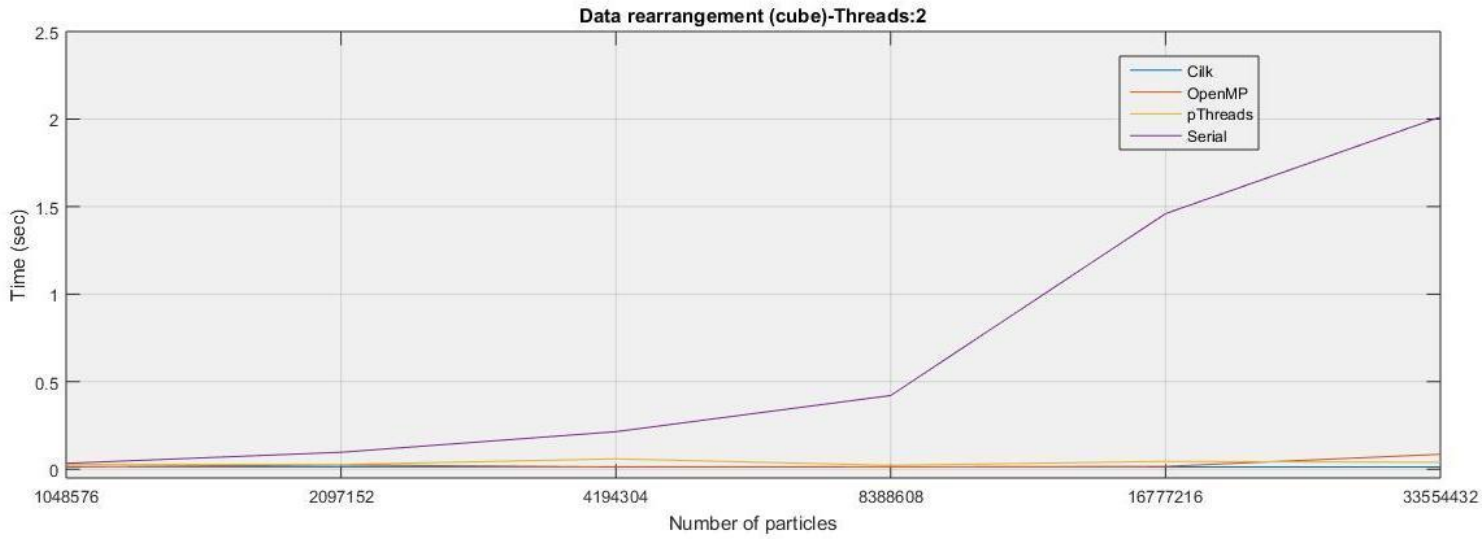

Morton encoding (cube)-Threads:2048
Time (sec)
Number of particles
Cilk
OpenMP
pThreads
Serial
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1048576
2097152
4194304
8388608
16777216
33554432

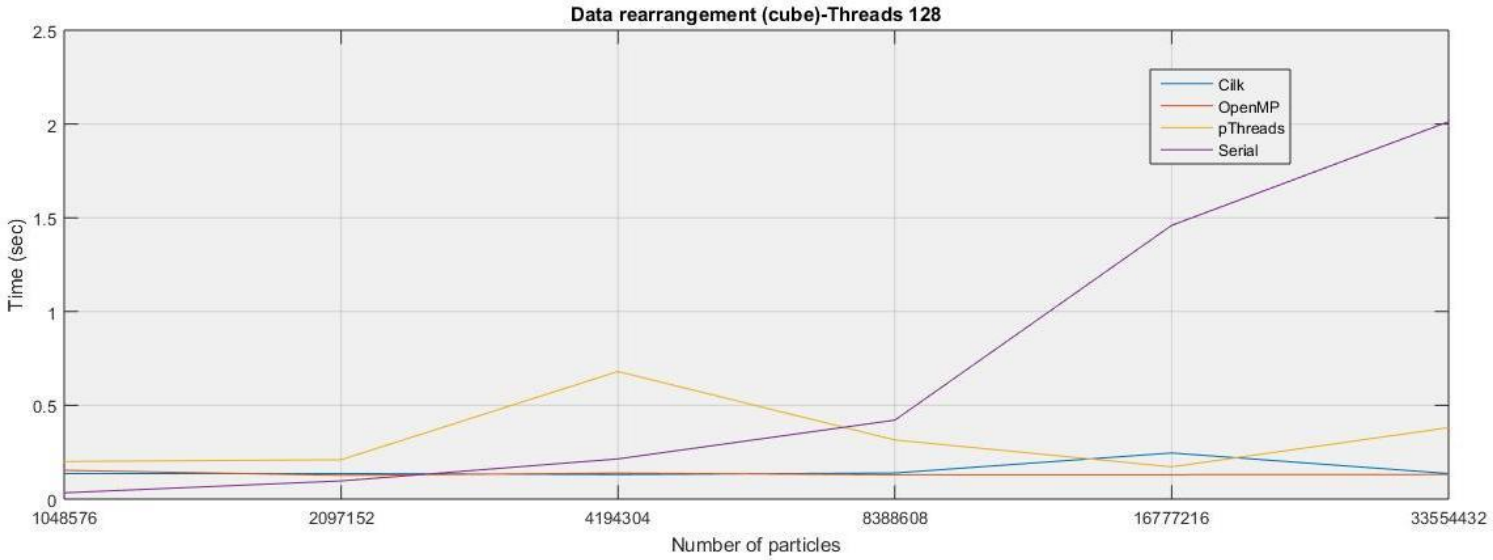

Radix sort (cube)-Threads:2
Time (sec)
Number of particles
Cilk
OpenMP
pThreads
Serial
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1048576
2097152
4194304
8388608
16777216
33554432

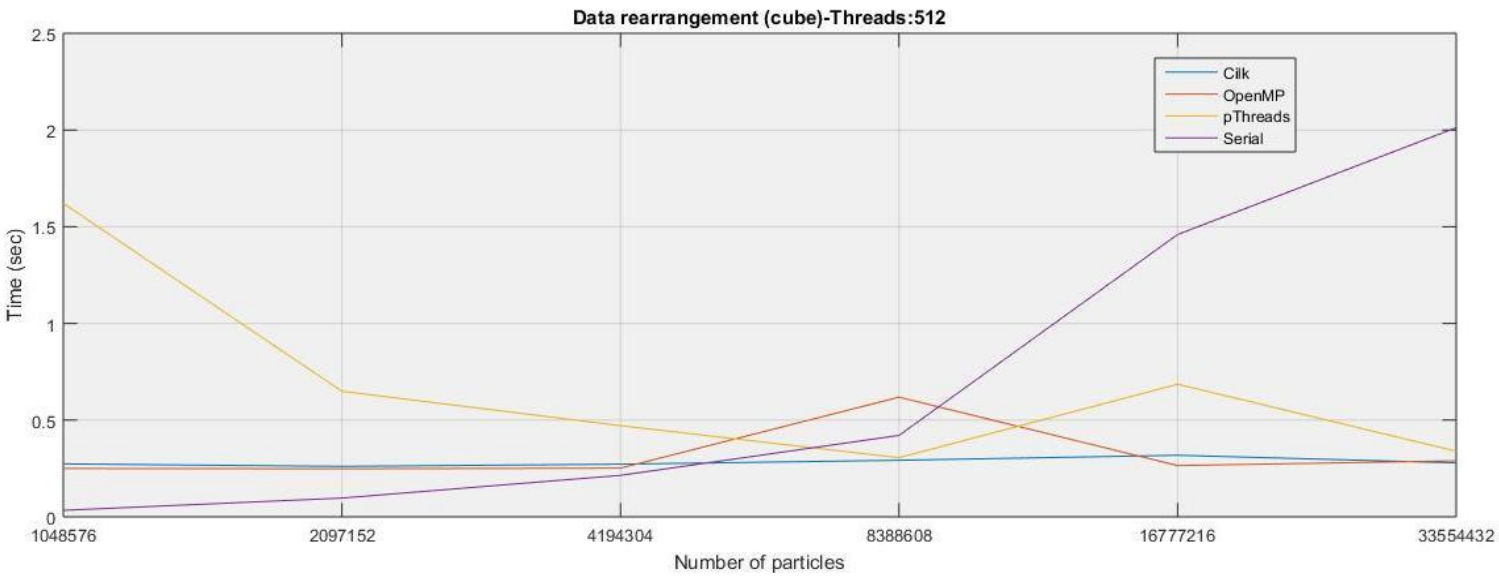

Radix sort (cube)-Threads:128
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

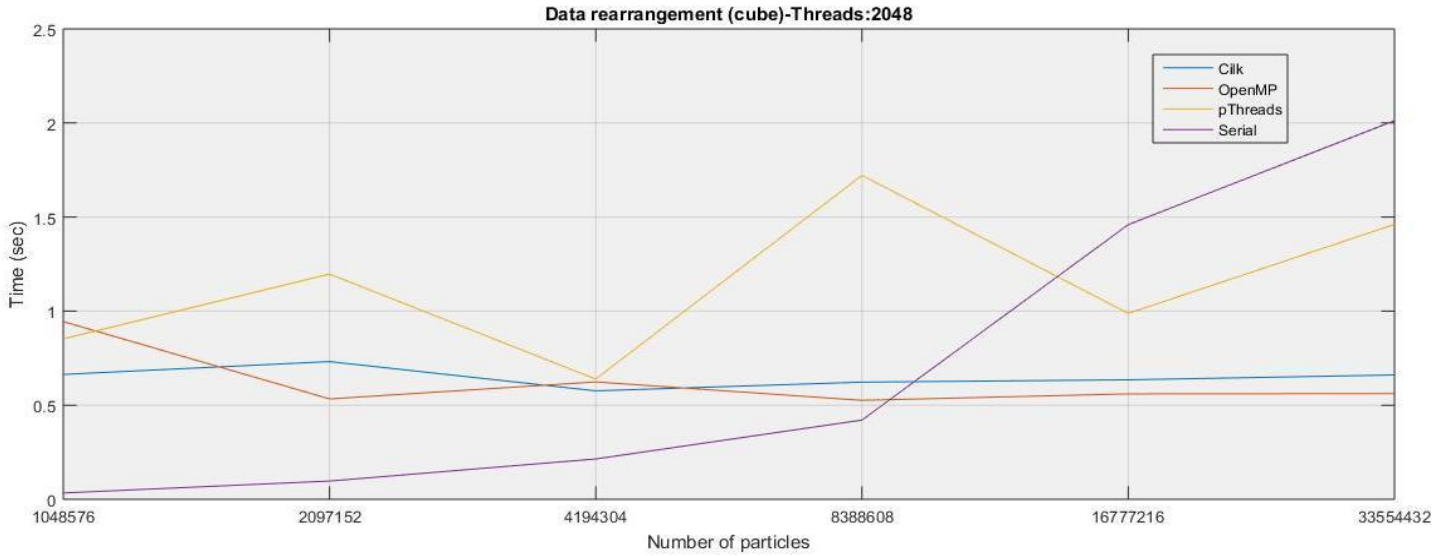

Radix sort (cube)-Threads:512
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1048576
2097152
4194304
8388608
16777217
Number of particles

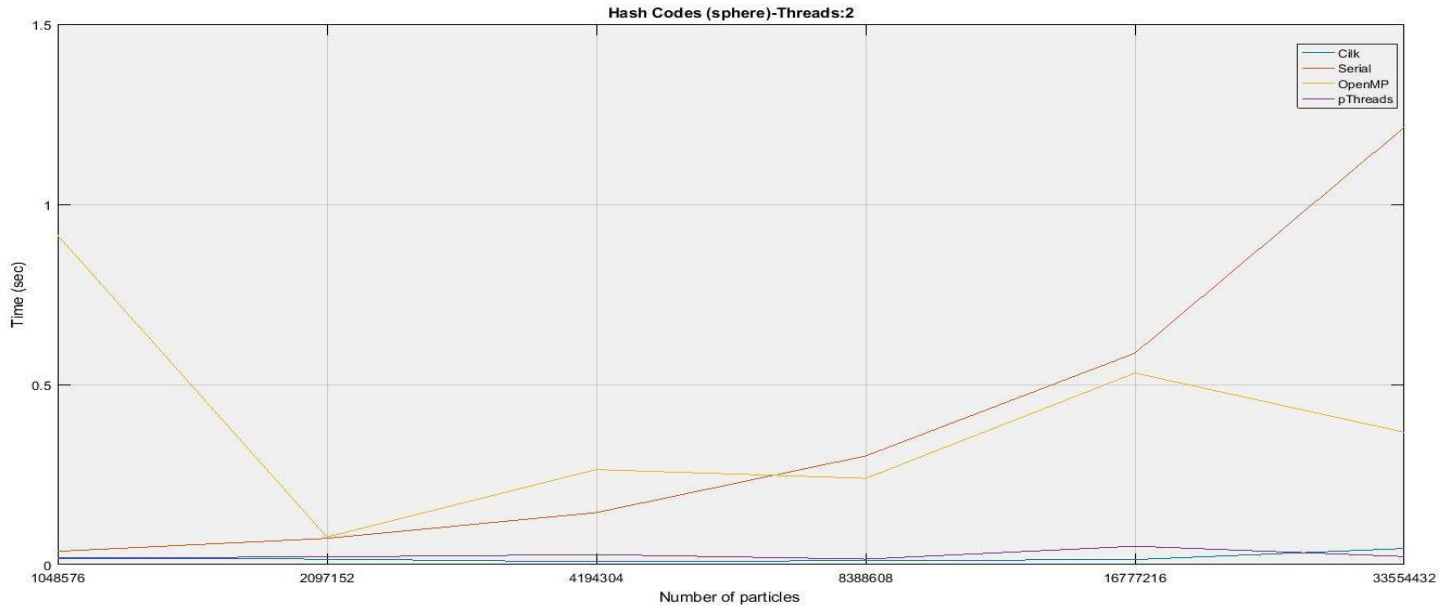

Radix sort (cube)-Threads:2048
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

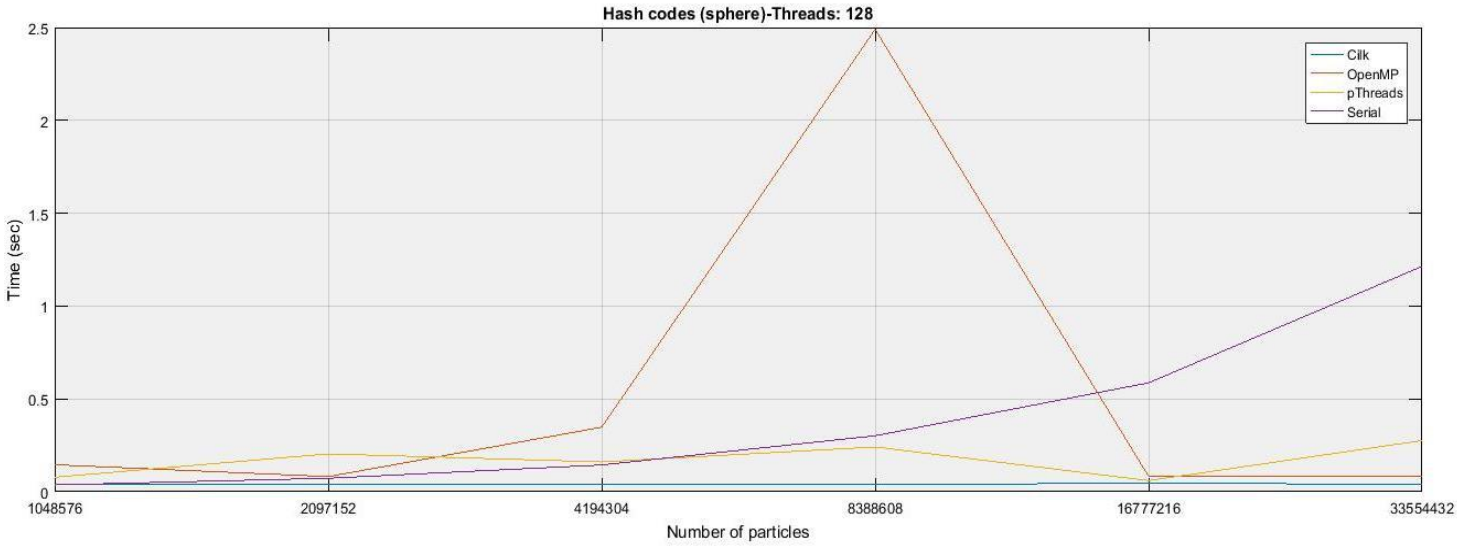

Data rearrangement (cube)-Threads:2
Cilk
OpenMP
pThreads
Serial
Time (sec)
2.5
2
1.5
1
0.5
0
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

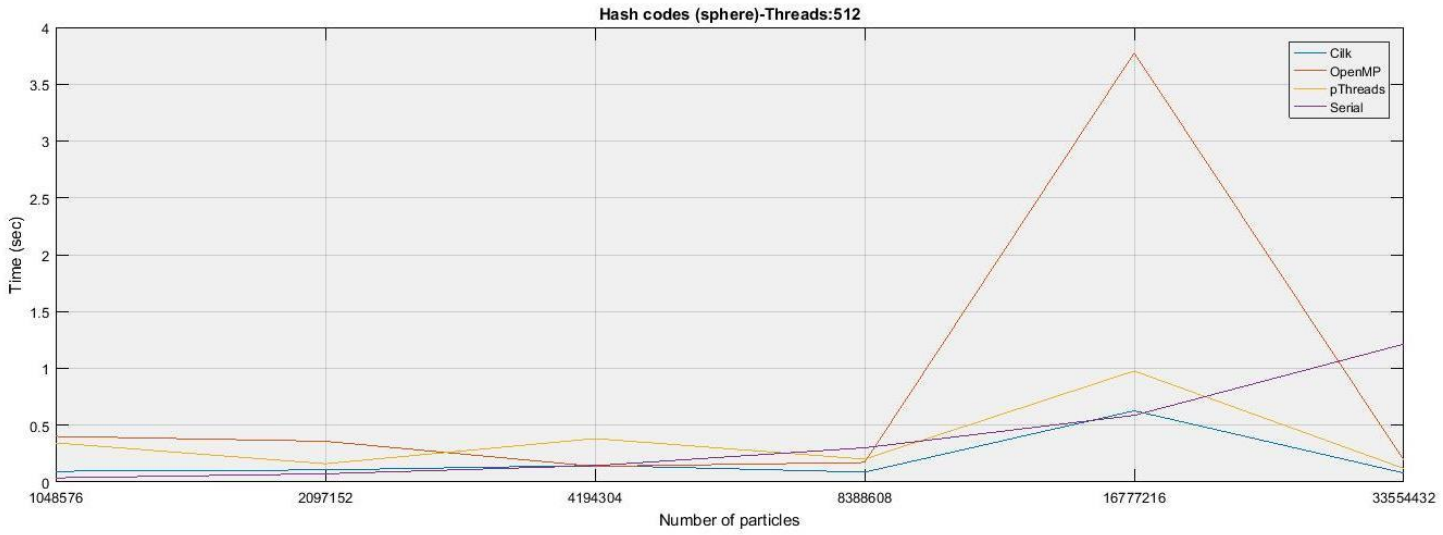

Data rearrangement (cube)-Threads 128
Cilk
OpenMP
pThreads
Serial
Time (sec)
2.5
2
1.5
1
0.5
0
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

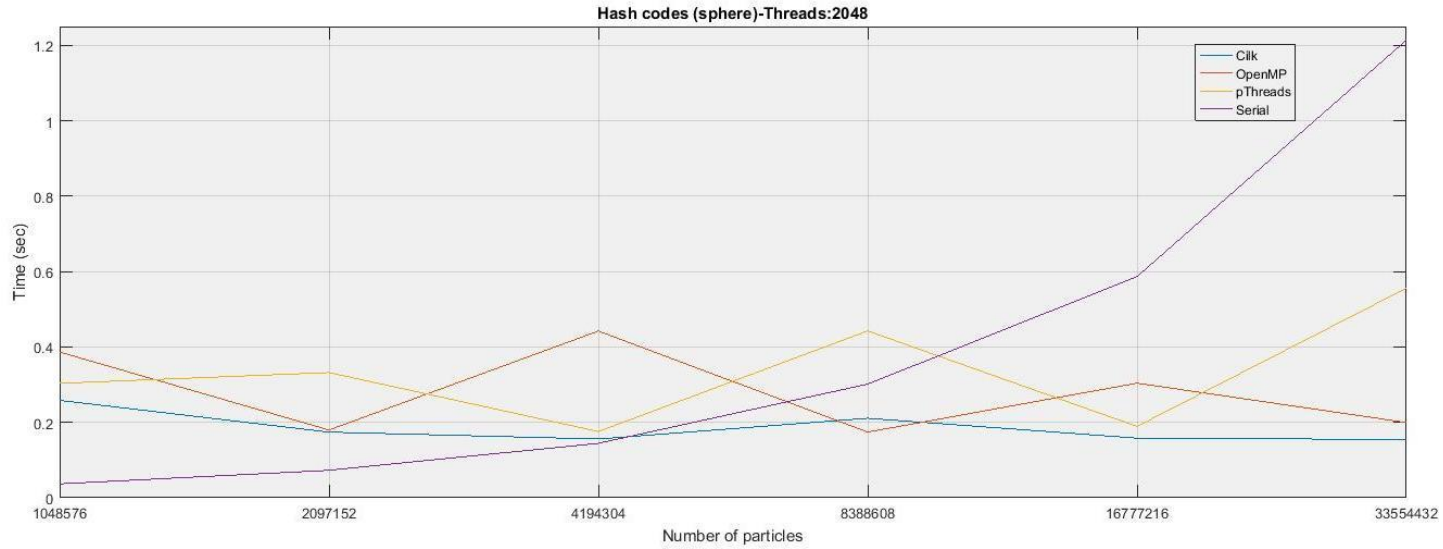

Data rearrangement (cube)-Threads:512
Cilk
OpenMP
pThreads
Serial
Time (sec)
2.5
2
1.5
1
0.5
0
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

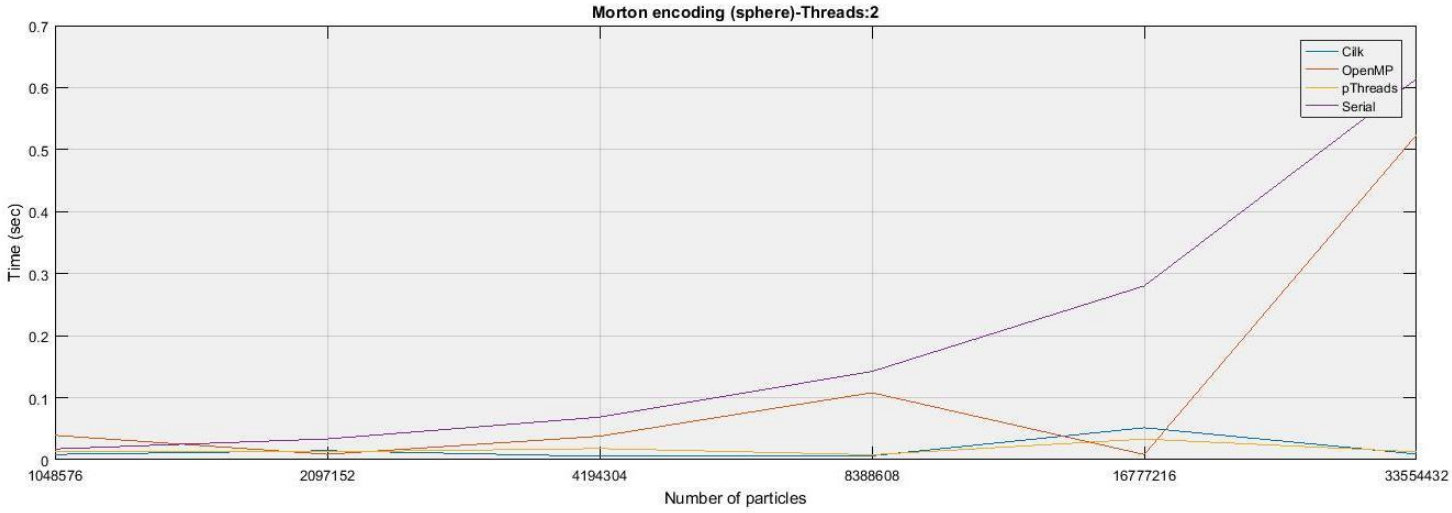

Data rearrangement (cube)-Threads:2048
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

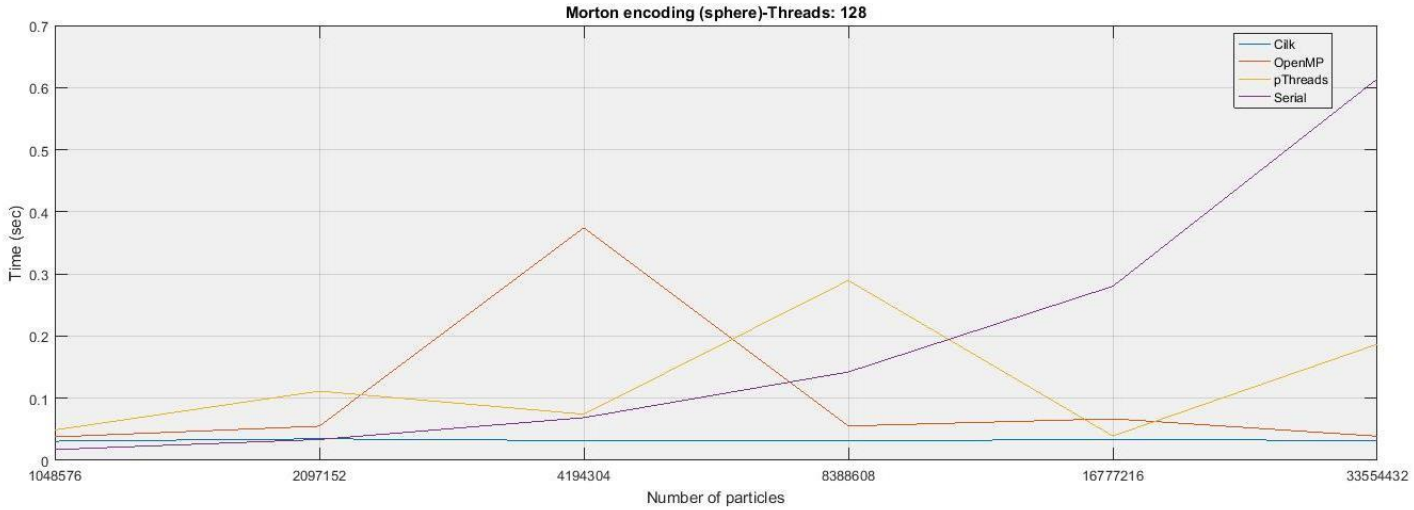

Hash Codes (sphere)-Threads:2
Cilk
Serial
OpenMP
pThreads
Time (sec)
0
0.5
1
1.5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

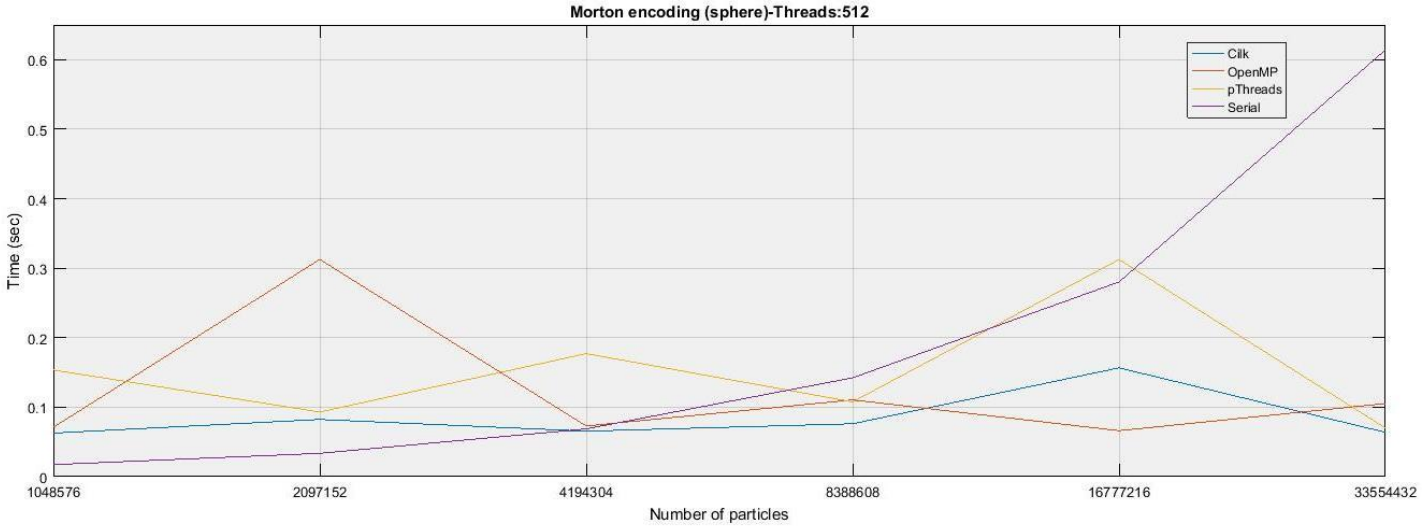

Hash codes (sphere)-Threads: 128
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

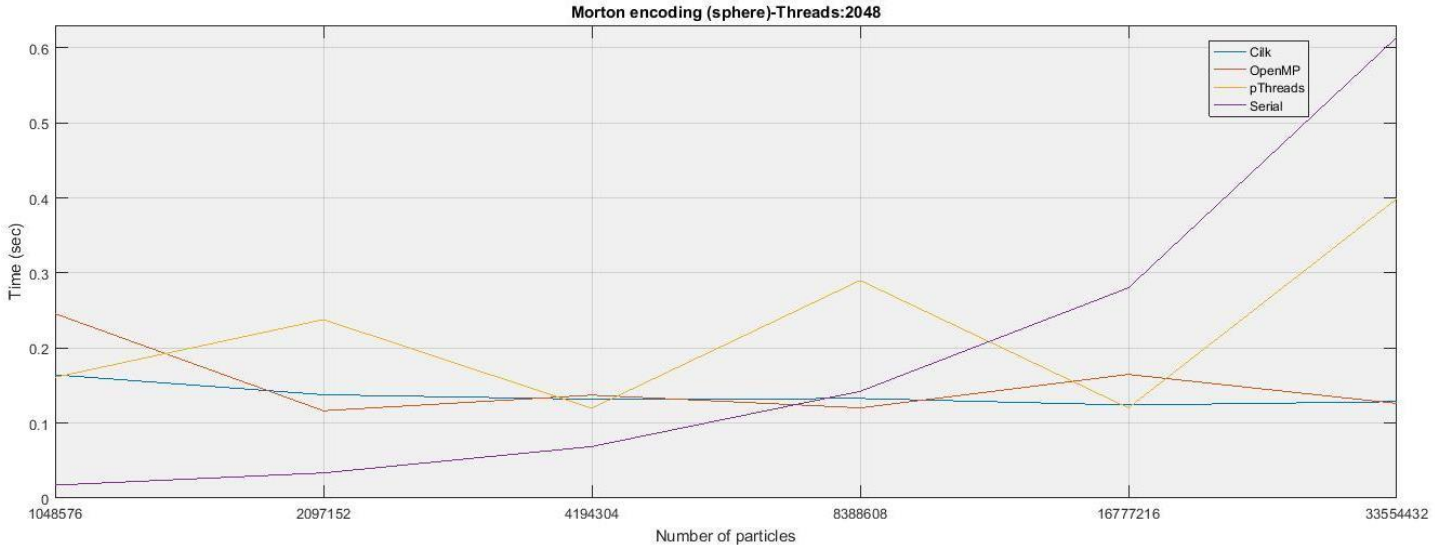

Hash codes (sphere)-Threads:512
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
3
3.5
4
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

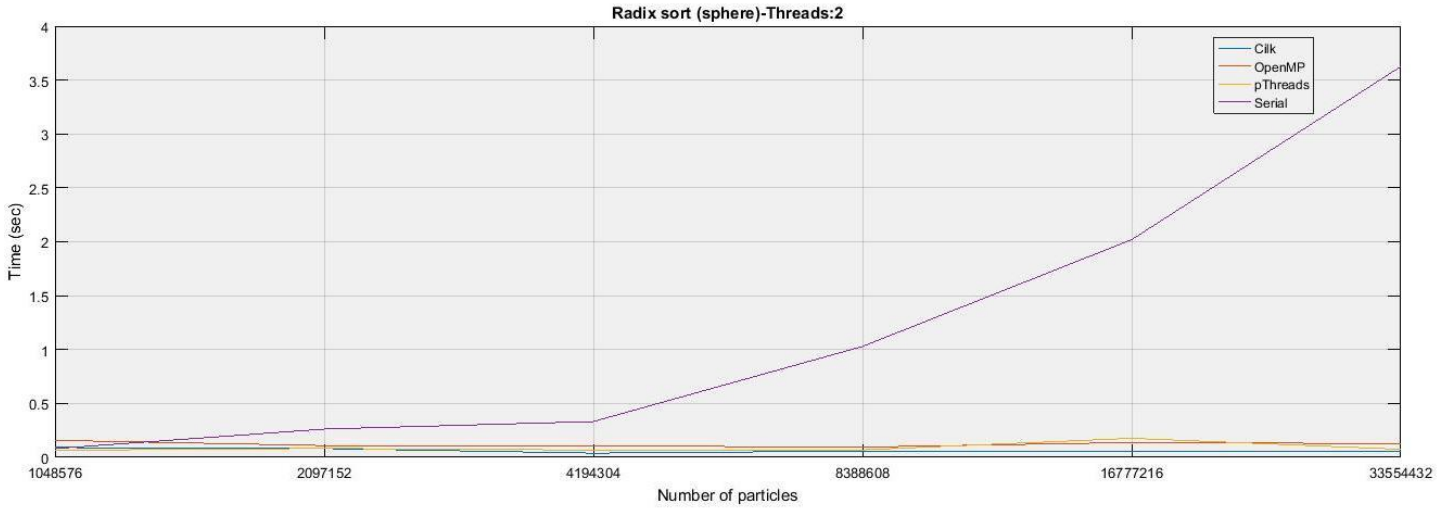

Hash codes (sphere)-Threads:2048
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.2
0.4
0.6
0.8
1
1.2
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

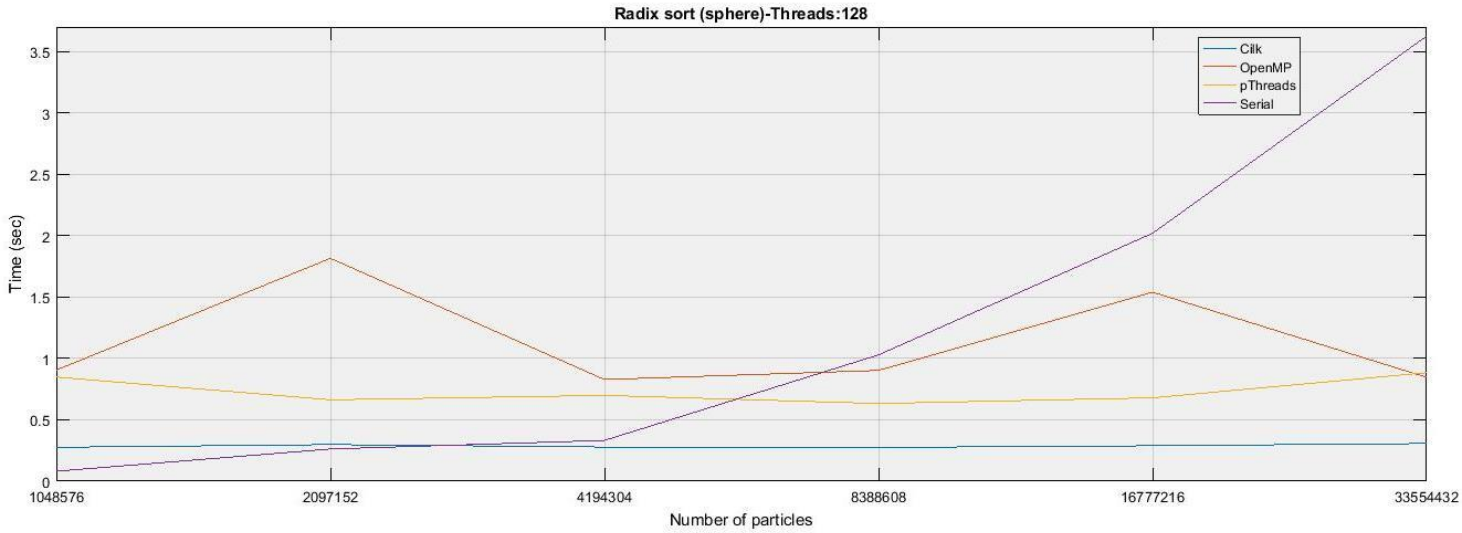

Morton encoding (sphere)-Threads:2
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

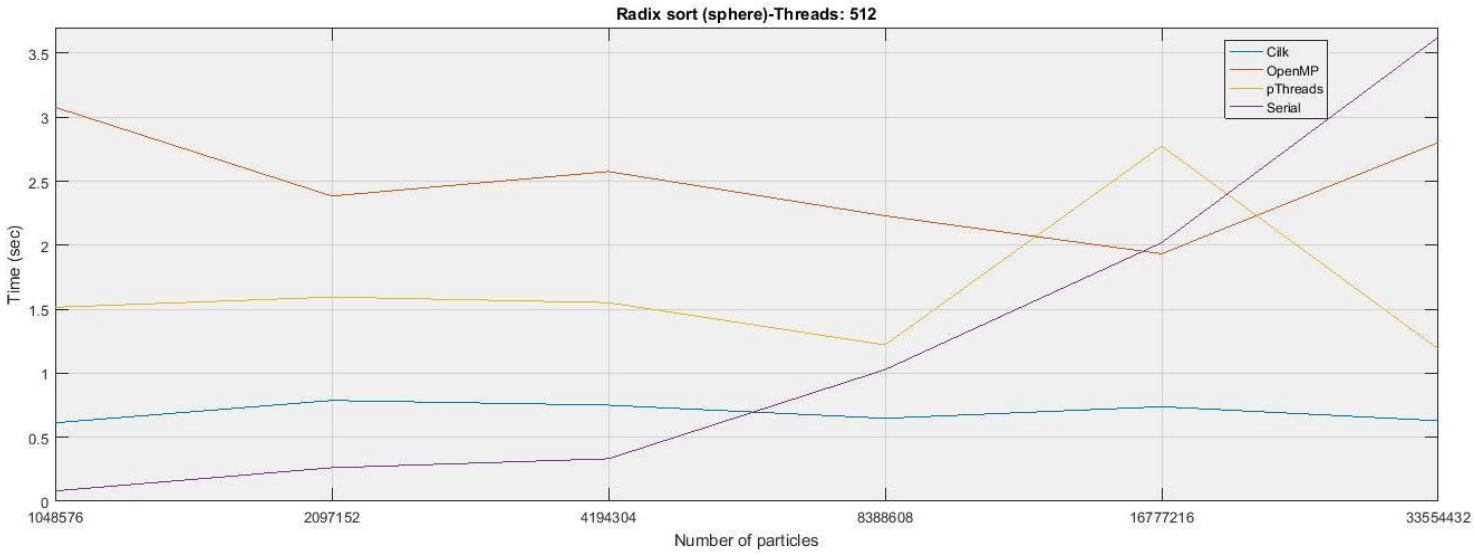

Morton encoding (sphere)-Threads: 128
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

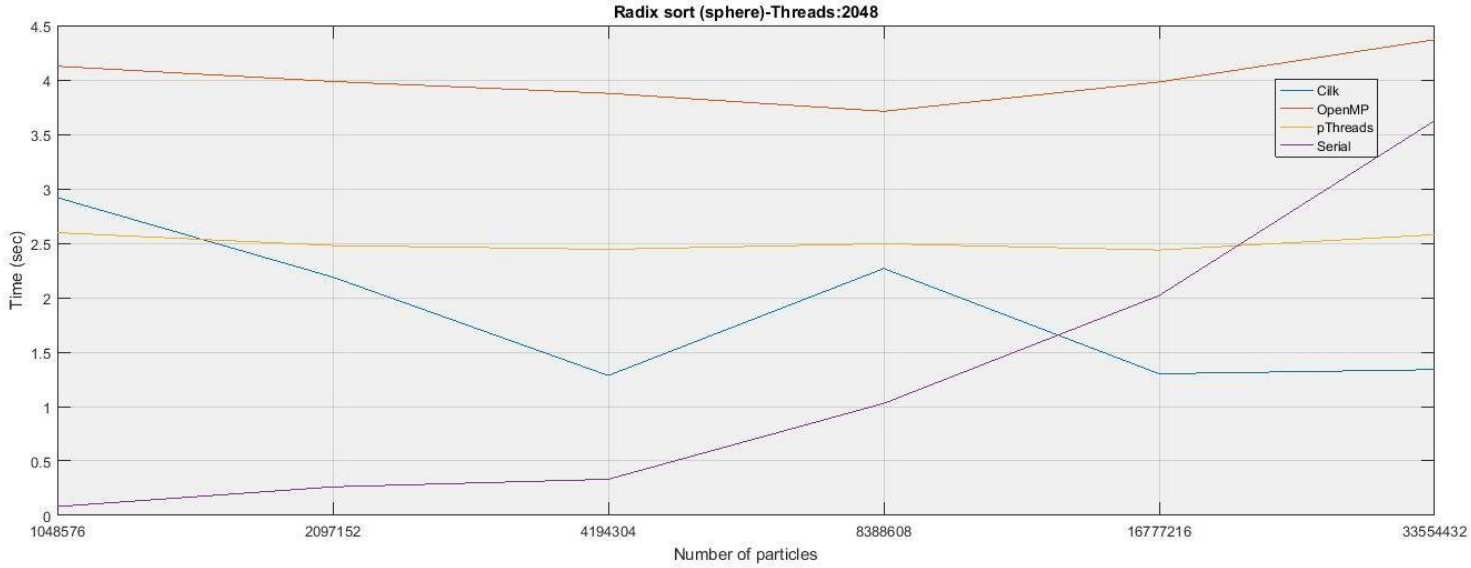

Morton encoding (sphere)-Threads:512
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.1
0.2
0.3
0.4
0.5
0.6
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

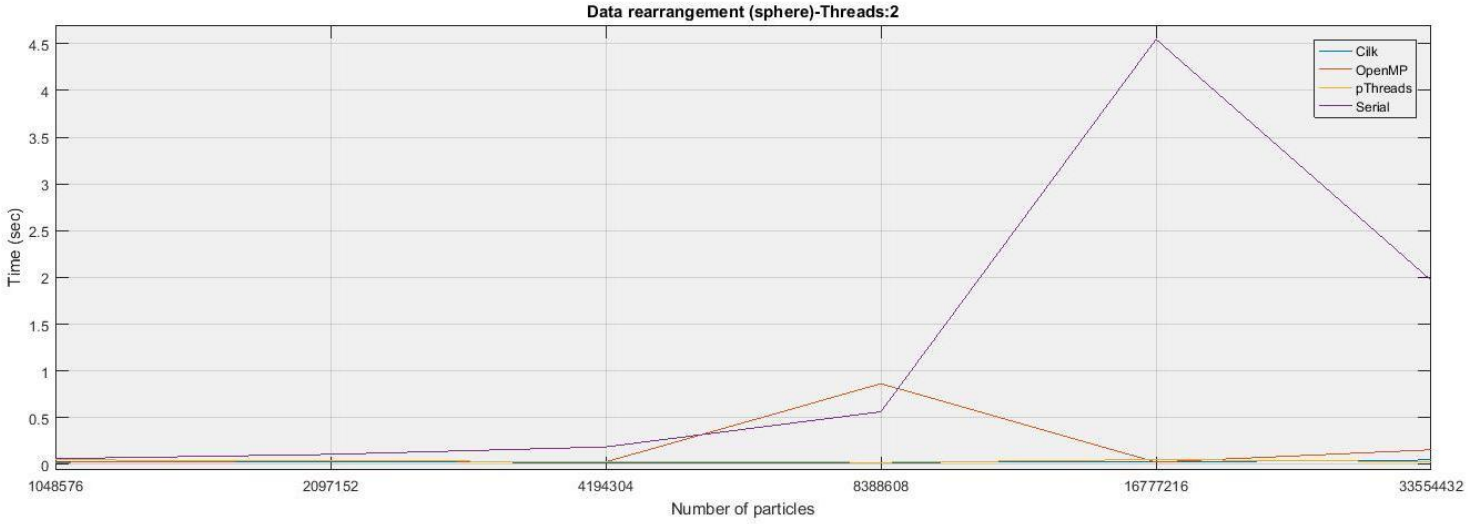

Morton encoding (sphere)-Threads:2048
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.1
0.2
0.3
0.4
0.5
0.6
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

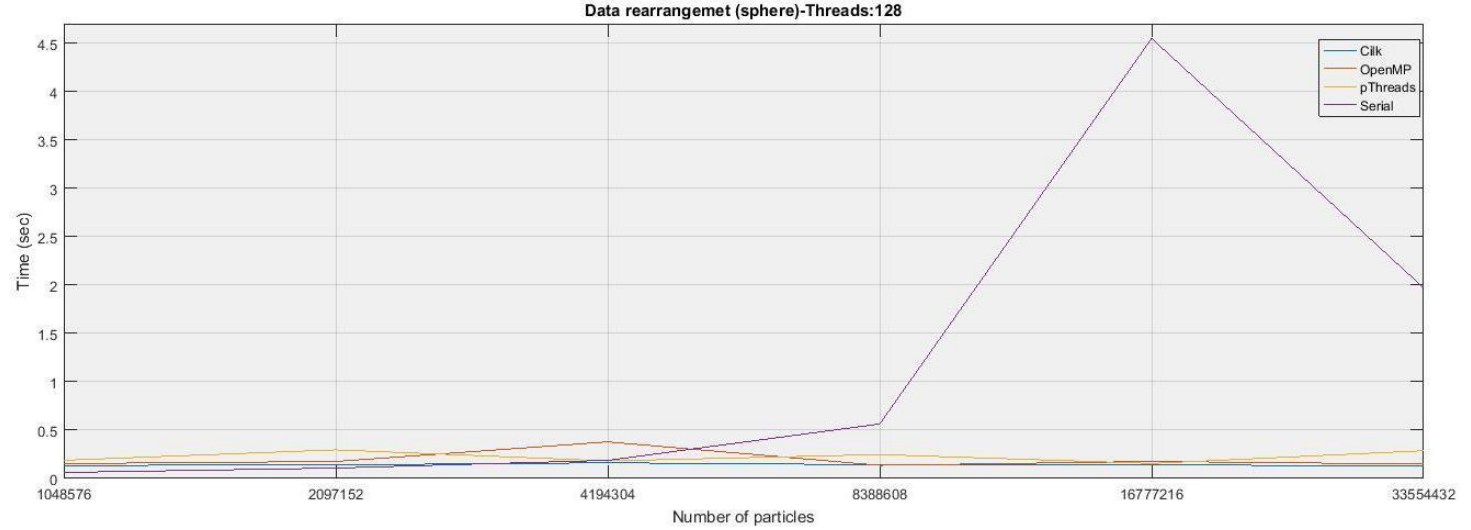

Radix sort (sphere)-Threads:2
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
3
3.5
4
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

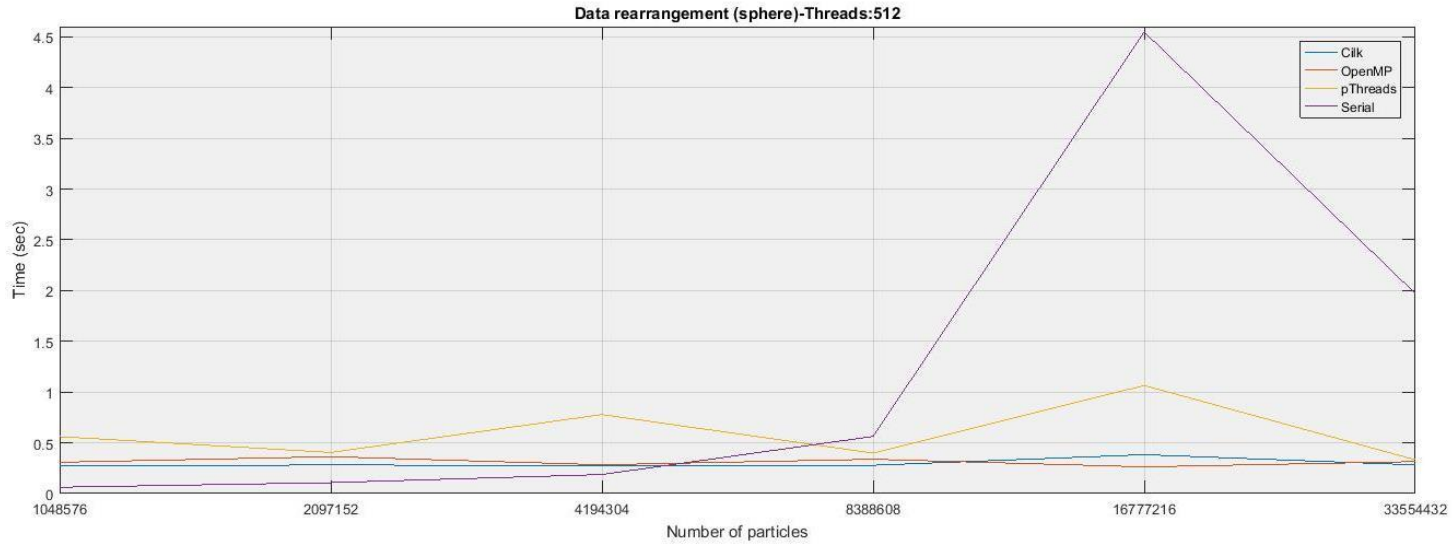

Radix sort (sphere)-Threads:128
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
3
3.5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles

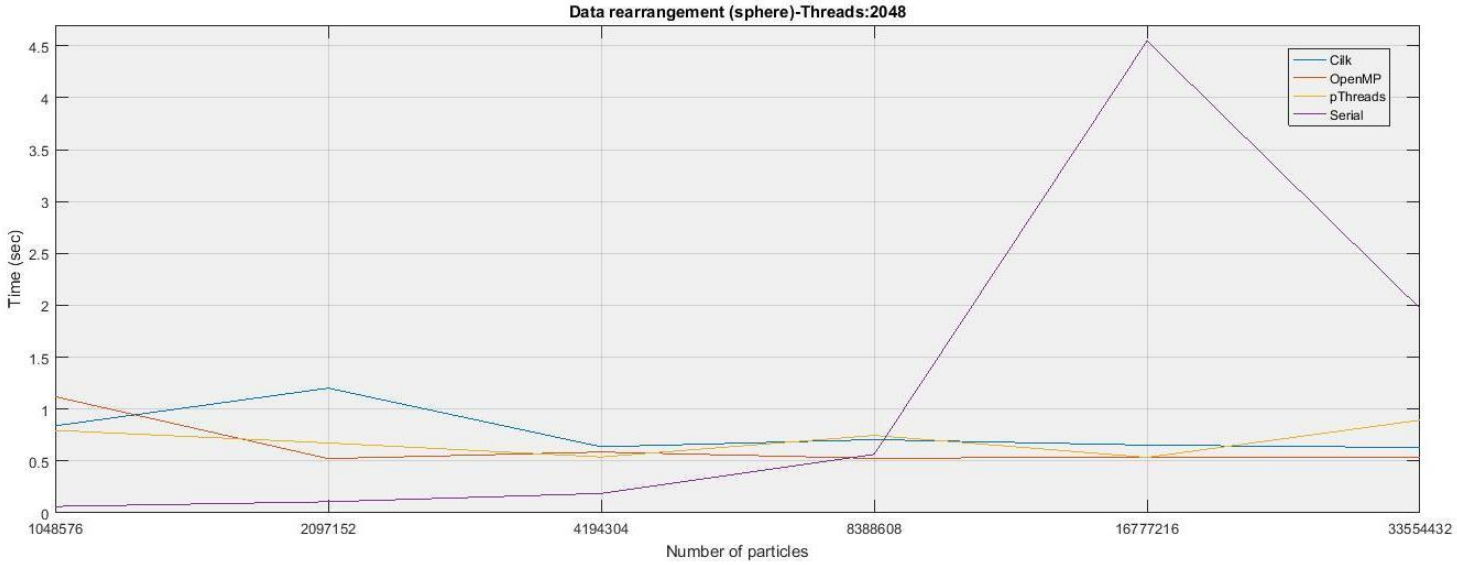

Radix sort (sphere)-Threads: 512
Cilk
OpenMP
pThreads
Serial
Time (sec)
0
0.5
1
1.5
2
2.5
3
3.5
1048576
2097152
4194304
8388608
16777216
33554432
Number of particles


Radix sort (sphere)-Threads:2048
Time (sec)
Number of particles
Cilk
OpenMP
pThreads
Serial
1048576
2097152
4194304
8388608
16777216
33554432


Data rearrangement (sphere)-Threads:2
Time (sec)
Number of particles
Cilk
OpenMP
pThreads
Serial
1048576
2097152
4194304
8388608
16777216
33554432


Data rearrangemet (sphere)-Threads:128
Time (sec)
Number of particles
Cilk
OpenMP
pThreads
Serial
1048576
2097152
4194304
8388608
16777216
33554432





Ομοίως με πριν, μπορούμε να παρατηρήσουμε από τα παραπάνω διαγράμματα πως όσο αυξάνεται ο αριθμός των νημάτων, συγκριτικά με τη σειριακή υλοποίηση που εκτελείται από ένα μόνο νήμα, ο χρόνος των παράλληλων εκδόσεων χειροτερεύει. Συγκεκριμένα, για αριθμό νημάτων από 128 και πάνω υπάρχει αυτή η τάση προς αύξηση του χρόνου εκτέλεσης η οποία τείνει να ενταθεί για 1024 ή 2048 νήματα. Επιπρόσθετα, αξιοσημείωτο είναι το γεγονός πως η υλοποίηση με τη Cilk παραμένει σχεδόν καθ' όλη την πορεία της εργασίας πρώτη στην ταχύτητα εκτέλεσης των συναρτήσεων και κατ' επέκταση του προγράμματος. Αυτό θα μπορούσε να εξηγηθεί λέγοντας πως η Cilk μοιράζει εξαρχής ισόποσα τις εργασίες στα υπάρχοντα νήματα εξετάζοντας κάθε φορά αυτόματα την περίπτωση, όπως αναφέραμε και νωρίτερα. Στο σημείο αυτό θα θέλαμε να αναφέρουμε πως ορισμένες κορυφές από τα παραπάνω διαγράμματα και συγκεκριμένα στην υλοποίηση της OpenMP, είναι πολύ μεγαλύτερες από ότι θα έπρεπε, γεγονός που μπορεί να οφείλεται και στο διάδη (server του πανεπιστημίου) λόγω των προβλημάτων που αντιμετώπισε, αλλά και το μεγάλο πλήθος των χρηστών εκείνη την περίοδο.

5. Αναφορές

Παρακάτω αναγράφονται οι αναφορές στις οποίες στηριχτήκαμε για την παρούσα εργασία.

http://stackoverflow.com/questions/6603404/when-is-pthread-spin-lock-the-right-thing-to-use-over-e-g-a-pthread-mutex
https://software.intel.com/en-us/articles/openmp-loop-scheduling
https://software.intel.com/en-us/forum
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.5002&rep=rep1&type=pdf
http://classes.engineering.wustl.edu/cse539/web/lectures/lec03.pdf
http://www.math.udel.edu/~plechac/M578/Intro.Par.Processing/section_8.html#PP_at_CTC
http://www.math.udel.edu/~plechac/M578/Intro.Par.Processing/section_7.html#Costs
https://computing.llnl.gov/tutorials/pthreads/
https://computing.llnl.gov/tutorials/openMP/
https://computing.llnl.gov/tutorials/parallel_comp/
http://forum.openmp.org/forum/viewtopic.php?f=3&t=83
http://www.forceflow.be/2013/10/07/morton-encodingdecoding-through-bit-interleaving-implementations/