ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Συστήματα Παράλληλης Επεξεργασίας

Άσκηση 3: Παραλληλοποίηση και βελτιστοποίηση αλγορίθμων σε επεξεργαστές γραφικών

Όνομα	Επώνυμο	Α.Μ.
Αλτάν	Αβτζή	03119241
Τζόναταν	Λουκάι	03119230
Σταύρος	Λαζάρου	03112642

Αλγόριθμος K-means

Υλοποίηση Naive version

1. Υλοποίηση κώδικα

- Υλοποίηση υπορουτίνας `get_tid()`:

```
__device__ int get_tid()
{
  return blockDim.x * blockIdx.x + threadIdx.x;
}
```

- Υλοποίηση υπορουτίνας `euclid_dist_2()`:

```
__host__ __device__ inline static
double euclid_dist_2()
{
  int i;
  double ans=0.0;

  for (i=0; i<numCoords; i++)
    ans += (objects[objectId*numCoords + i] - clusters[clusterId*numCoords
           (objects[objectId*numCoords + i] - clusters[clusterId*numCoords

  return(ans);
}
```

- Υλοποίηση του πυρήνα `find_nearest_cluster()` :

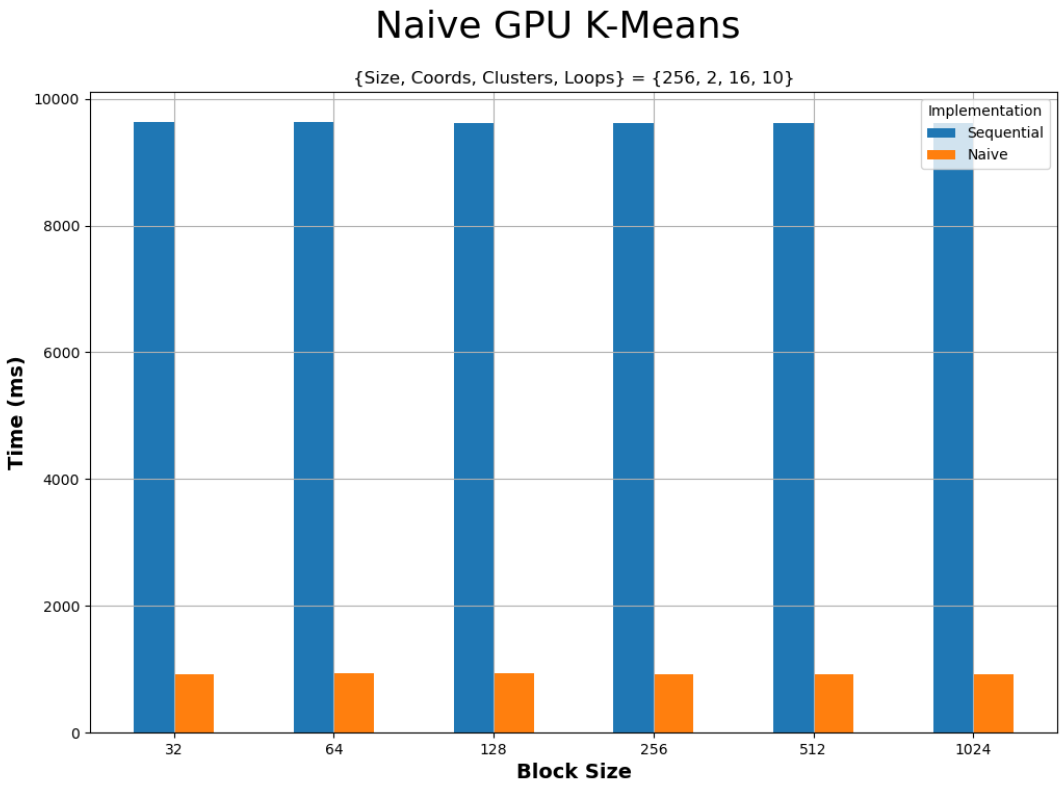
```
  __global__ static
void find_nearest_cluster()
{

        /* Get the global ID of the thread. */
    int tid = get_tid();

    if (tid < numObjs) {
        int   index, i;
        double dist, min_dist;

        /* find the cluster id that has min distance to object */
        index = 0;
        min_dist = euclid_dist_2(numCoords, numObjs, numClusters, objects,

        for (i=1; i<numClusters; i++) {
            dist = euclid_dist_2(numCoords, numObjs, numClusters, objects,

            /* no need square root */
            if (dist < min_dist) {
            /* find the min and its array index*/
                min_dist = dist;
                index    = i;
            }
        }

        if (deviceMembership[tid] != index) {
            atomicAdd(devdelta, 1.0);
        }

        /* assign the deviceMembership to object objectId */
        deviceMembership[tid] = index;
    }}
```

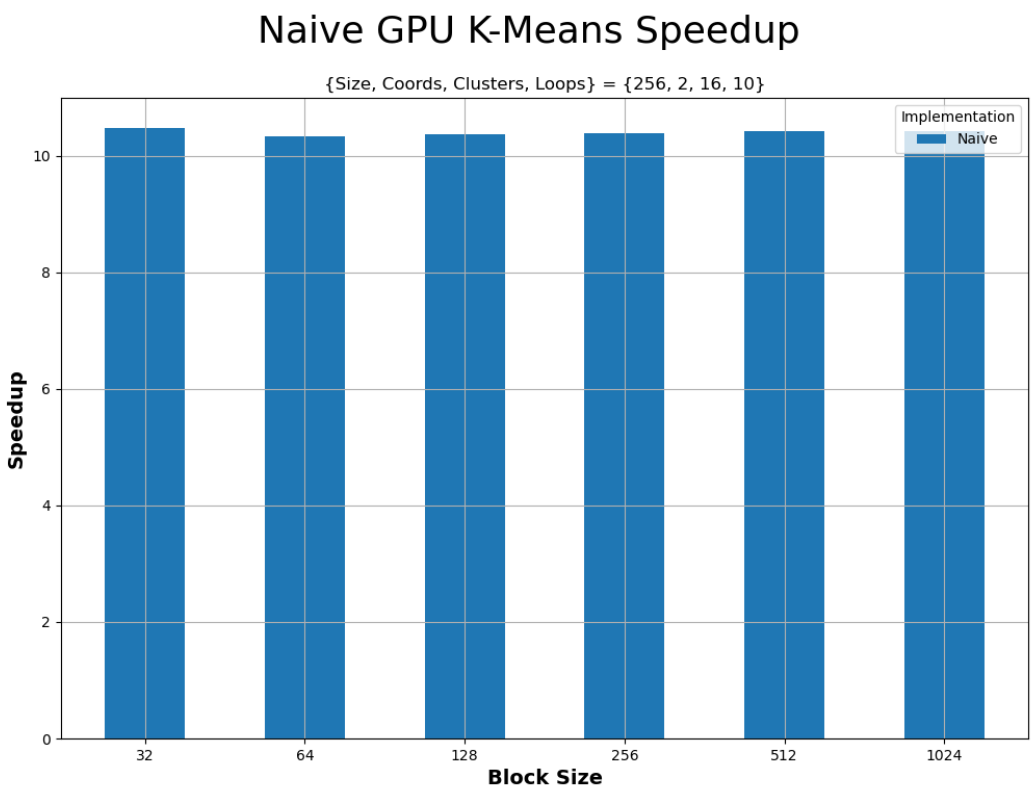
- Πραγματοποιούμε τις ζητούμενες μεταφορές δεδομένων σε κάθε iteration του αλγορίθμου:

```
checkCuda(cudaMemcpy(deviceClusters, clusters, numClusters*numCoords*sizeof

/* ... */

checkCuda(cudaMemcpy(membership, deviceMembership, numObjs*sizeof(int), cud

checkCuda(cudaMemcpy(&delta, dev_delta_ptr, sizeof(double), cudaMemcpyDevic
```

2. Αξιολόγηση επίδοσης:

- Γραφικές Παραστάσεις





Παρατηρούμε μια μεγάλη βελτίωση στους χρόνους εκτέλεσης της παράλλληλης έκδοσης σε σχέση με τη σειριακή. Συγκεκριμένα για όλα τα block sizes, το speedup είναι περισσότερο από 10. Η βελτίωση αυτή επιδεικνύει την αποτελεσματική χρήση των πόρων της GPU.

Για αυτό το configuration βλέπουμε ότι η διαφορά στους χρόνους για block sizes εύρους 32-1024 είναι αμελητέα. Αυτό μπορεί να οφείλεται σε μερικούς λόγους, όπως:

1. Οι σύγχρονες κάρτες γραφικών είναι σχεδιασμένες έτσι ώστε να είναι σε θέση να μπορούν να διαχειριστούν με ευκολία αυτό το εύρος.
2. Η naive έκδοση του αλγόριθμου είναι memory-bound, δηλαδή η επίδοση του περιορίζεται από τον ρυθμό με τον οποίον μεταφέρονται δεδομένα μεταξύ της κεντρικής μνήμης και των υπολογιστικών νημάτων, παρά από τον υπολογισμό τον ίδιο.
3. Ο αλγόριθμος σε ορισμένα σημεία χρησιμοποιεί ατομικές εντολές για την ενημέρωση δεδομένων με αποτέλεσμα να σειριοποιείται η πρόσβαση σε αυτά και έτσι να μειώνονται τα οφέλη τα οποία προσδίδει ο μεγαλύτερος αριθμός νημάτων για κάθε block.

Υλοποίηση Transpose version

1. Υλοποίηση κώδικα

- Υλοποίηση του πυρήνα `euclid_dist_2_transpose()` :

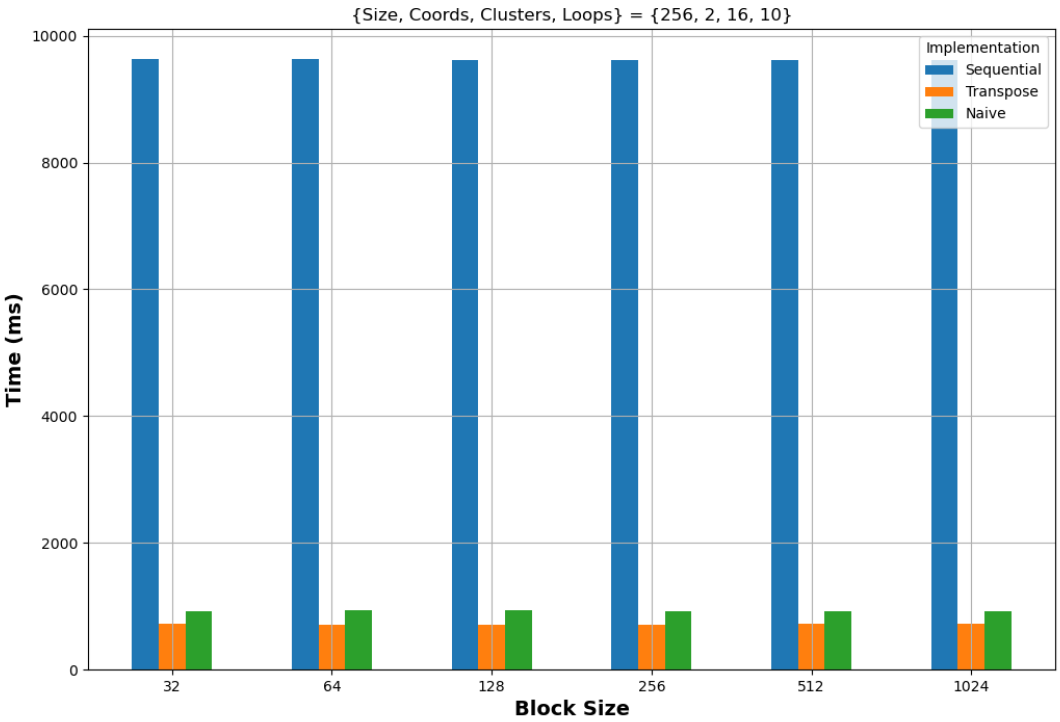
```
__host__ __device__ inline static
double euclid_dist_2_transpose()
{
  int i;
  double ans=0.0;

  for (i=0; i<numCoords; i++) {
    ans += (objects[i*numObjs + objectId] - clusters[i*numClusters + cluste
           (objects[i*numObjs + objectId] - clusters[i*numClusters + cluste
  }

    return(ans);
}
```

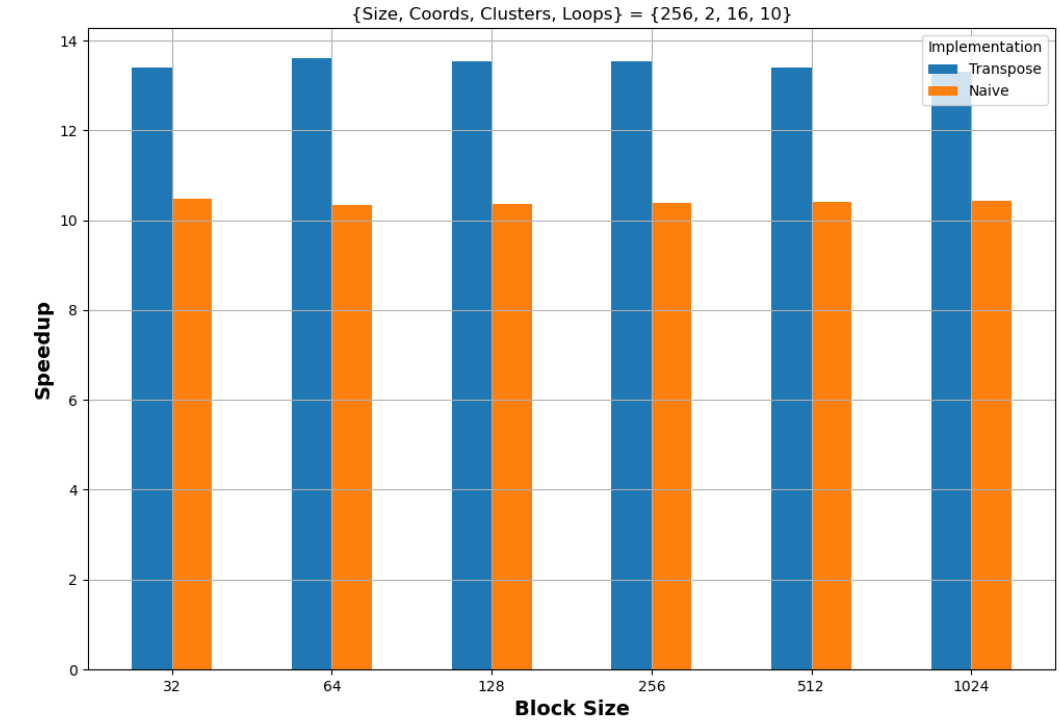
- Φροντίζουμε επίσης για την σωστή αρχικοποίηση και μετατροπή των δεδομένων:

```
double  **dimObjects = (double**) calloc_2d (numCoords, numObjs, sizeof(dou

double  **dimClusters = (double**) calloc_2d (numCoords, numClusters, sizeo

double  **newClusters = (double**) calloc_2d (numCoords, numClusters, sizeo

/* ... */

for(i=0; i<numObjs; i++) {
  for(j=0; j<numCoords; j++) {
    dimObjects[j][i] = objects[i*numCoords + j];
  }
}

/* ... */

const unsigned int numClusterBlocks = (numObjs + numThreadsPerClusterBlock

/* ... */

for(i=0; i<numClusters; i++) {
  for(j=0; j<numCoords; j++) {
    clusters[i*numCoords + j] = dimClusters[j][i];
  }
}
```

2. Αξιολόγηση επίδοσης:

- Γραφικές παραστάσεις





Παρατηρούμε ότι η transpose έκδοση του αλγορίθμου παρουσίαζει μια μικρή βελτίωση στην επίδοση σε σχέση με τη naive έκδοση. Παρόλα αυτά, βλέπουμε ξανά ότι το block size δεν παίζει κάποιο ιδιαίτερο ρόλο.

Η αύξηση της επίδοσης της transpose έκδοσης πιθανώς οφείλεται στη διαφορετική δομή δεδομένων και στο μοτίβο με τον οποίο ανακτώνται αυτά τα δεδομένα από τη μνήμη. Στην column-based διάταξη, οι γραμμές των πινάκων αντιπροσωπεύουν μια διάσταση, ενώ οι στύλες κάποιο cluster center. Αυτό σημαίνει ότι οι συντεταγμένες των clusters αποθηκεύονται με συνεχή τρόπο, και συνεπώς η πρόσβαση σε αυτές γίνεται παρομοίως. Με αυτόν τον τρόπο, μειώνουμε σε κάποιο βαθμό το memory latency που παρουσιάζεται με τη naive έκδοση, και ενισχύουμε την υπολογιστική αποδοτικότητα.

Υλοποίηση Shared version

1. Υλοποίηση κώδικα

- Ορίζουμε το μέγεθος της διαμοιραζόμενης μνήμης που χρειάζεται η συγκεκριμένη υλοποιήση:

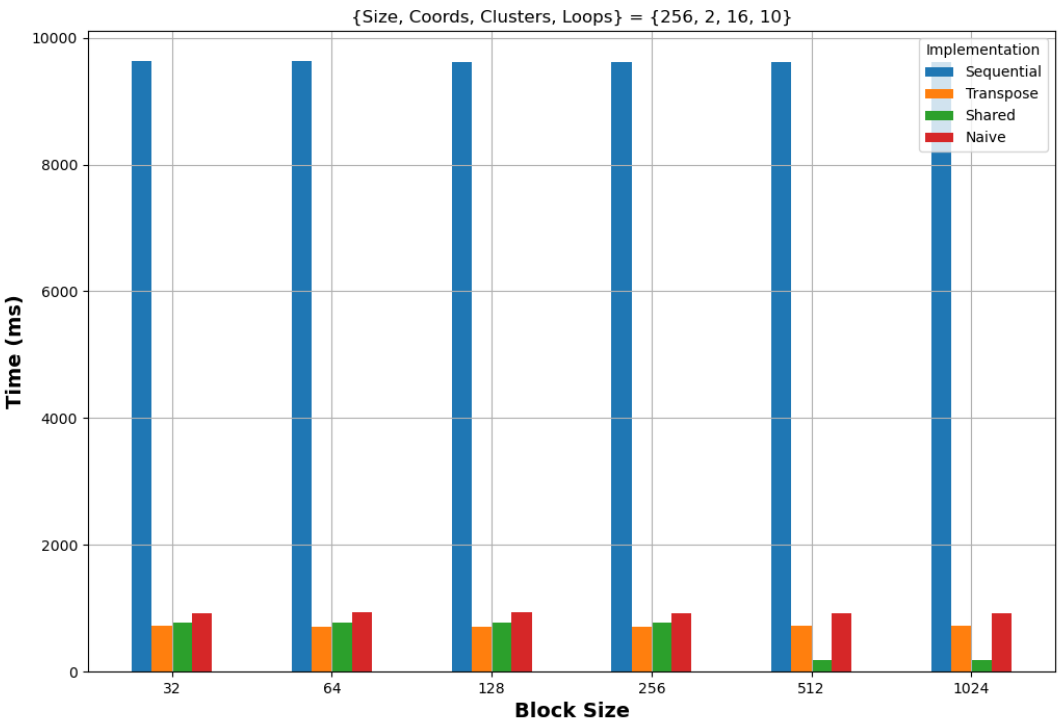
```
const unsigned int clusterBlockSharedDataSize = numClusters * numCoords * s
```

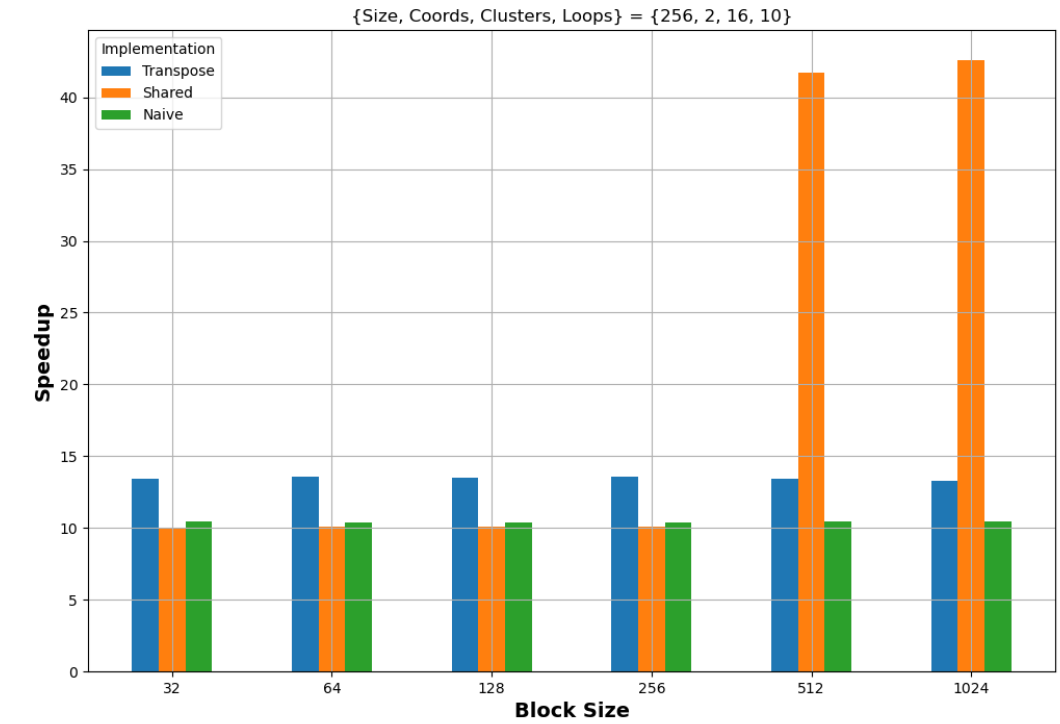
- Προσθέτουμε στον πυρήνα `find_nearest_cluster()` την μεταφορά των clusters στην διαμοιραζόμενη μνήμη:

```
if (tid < numClusters*numCoords) {
  shmemClusters[tid] = deviceClusters[tid];
}

__syncthreads();
```

2. Αξιολόγηση επίδοσης

- Γραφικες Παραστάσεις



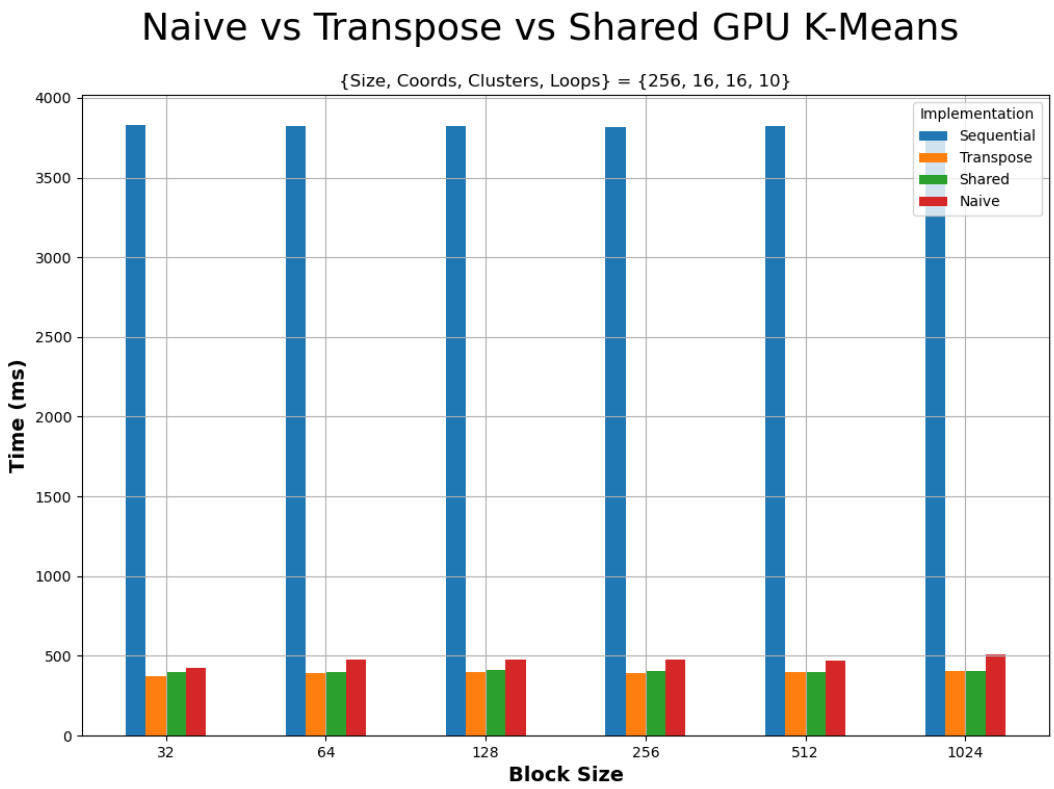


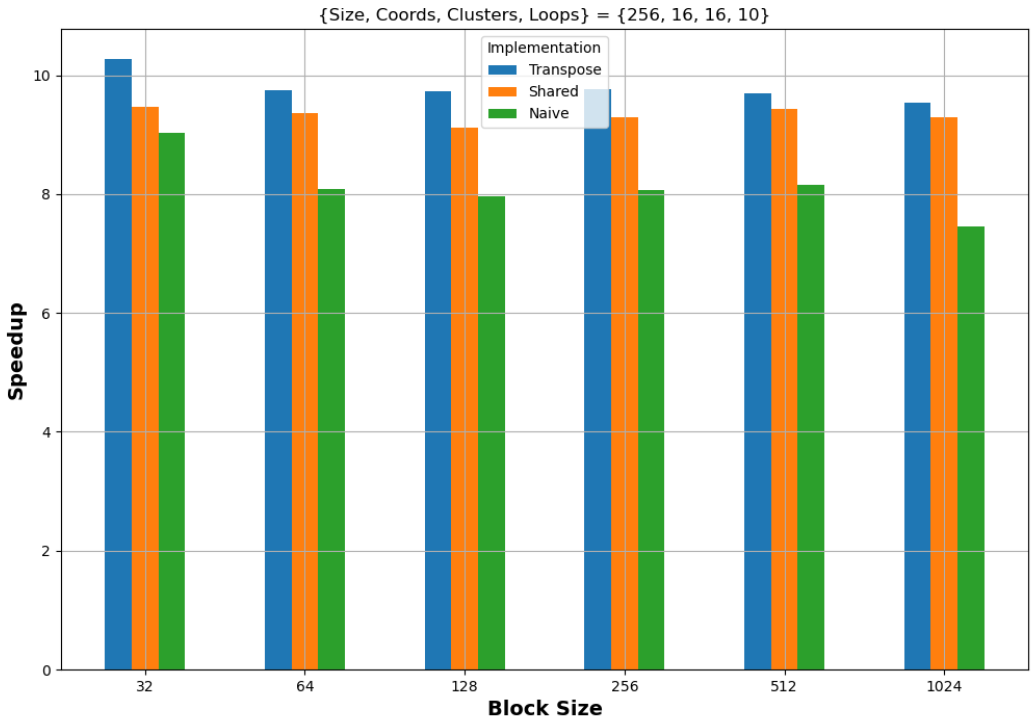
Παρατηρούμε ότι, σε σύγκριση με τις naive και transpose εκδόσεις του αλγορίθμου, η shared έκδοση έχει παρόμοια επίδοση με τις δυο προηγούμενες, κάνοντας όμως διαφορά στα μεγαλύτερα block sizes. Η ταχύτητά της για block sizes 512-1024 είναι αρκετά καλύτερη.

Σε αυτήν την περίπτωση, όταν το block size είναι τόσο μεγάλο, όλο και περισσότερα νήματα της κάρτας γραφικών μπορούν να έχουν πρόσβαση στα δεδομένα ενός block. Με αυτόν τον τρόπο αξιοποιείται καλύτερα η κοινή μνήμη, επειδή επιτυγχάνουμε λιγότερες συνολικά προσβάσεις στην κύρια μνήμη, αποφεύγουμε το κόστος που επιφέρουν, και έχουμε περισσότερο φόρτο εργασίας για τις υπολογιστικές μονάδες τις κάρτας γραφικών.

Σύγκριση υλοποίησεων / bottleneck Analysis

- Γραφικές παραστάσεις





Η shared υλοποίηση σε αυτήν την περίπτωση δεν επιτυγχάνει καλύτερη επίδοση από την transpose. Όποτε η επιλογή της για αυθαίρετα configurations δεν είναι πάντα ιδανική.