Παράλληλα και Διανεμημμένα Συστήματα

Εργασία 3 - All Pair Shortest Path, CUDA Implementation

ΠΕΤΡΟΣ ΜΗΤΣΕΑΣ 7925 - 7Ο ΕΞΑΜΗΝΟ, ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Παρουσίαση του αλγορίθμου

Το πρόβλημα αφορά την εύρεση, για κάθε ζευγάρι κόμβων σε ένα γράφο, της συντομότερης διαδρομής μεταξύ τους.

Η επίλυση του προβλήματος γίνεται σειριακά σε χρόνο $\Theta(n^3)$ με χρήση του αλγορίθμου Floyd-Warshall. Ο αλγόριθμος, ελέγχει κάθε ζευγάρι (i,j). Για κάθε k κόμβο που ανήκει στο γράφο, ελέγχει αν η διαδρομή i -> k και k-> j είναι μικρότερη από την ήδη αποθηκευμένη διαδρομή i -> j. Άν είναι αντικαθιστά το αποτέλεσμα. Για να υλοποιηθεί ο αλγόριθμος, χρειάζεται δύο εντολές for, για να ελεχθεί κάθε διαδρομή, και μια ακόμα για κάθε ενδιάμεσο κόμβο. Έτσι προκύπτει η παραπάνω πολυπλοκότητα.

Στην εργασία παρουσιάζεται η παραλληλοποίηση του αλγορίθμου αυτού σε περιβάλλον CUDA. Για το σκοπό αυτό υλοποιήθηκαν τρία kernel.

Ο πίνακας που δίνεται ως είσοδος στο πρόγραμμα έχει διαστάσεις n X n και κάθε στοιχείο (i,j) του πίνακα περιέχει το βάρος της διαδρομής i -> j. Ο πίνακας δεν είναι απαραίτητα συμμετρικός.

Ο πυρήνας του σειριακού αλγορίθμου φαίνεται παρακάτω:

```
for (int k=0; k<n; k++)
    {
        // Pick all vertices as source one by one
        for (int i=0; i<n; i++)
        {
            // Pick all vertices as destination for the
            // above picked source
            for (int j=0; j<n; j++)
            {

                // If vertex k is on the shortest path from
                // i to j, then update the value of Distance-Matrix[i][j]
                if (result[k+i*n] + result[j+k*n] < result[j+i*n])
                    result[j+i*n] = result[k+i*n]+result[j+k*n];
            }
        }
    }
```

Στο πρώτο kernel, έχουμε χρήση της global memory. Κάθε block στο device, περιέχει 256 threads και σύνολο υπάρχουν n*n/256 blocks. Κάθε thread αναλαμβάνει απο 1 ζευγάρι (i,j) του πίνακα και ελέγχει τη διαδρομή i -> k , k -> j για k απο 1 έως n. Το τμήμα του κώδικα που υλοποιεί το παραπάνω φαίνεται εδώ:

```
int i = blockIdx.x * blockDim.x + threadIdx.x;
int j = blockIdx.y * blockDim.y + threadIdx.y;
int index = i*n + j;

if (i<n && j<n){
  if (dist[k+i*n] + dist[j+k*n] < dist[index]){
    dist[index] = dist[k+i*n]+dist[j+k*n];
  }
}
__syncthreads();
```

Η συνάρτηση τρέχει n φορές για όλες τις τιμές του k. Η πολυπλοκότητα επομένως, είναι O(n)

Το δεύτερο kernel περιλαμβάνει και χρήση της Shared Memory μεταξύ των threads ίδιου block. Κάθε γραμμή του πίνακα ανατίθεται στα νήματα ενός block. Επομένως για μια δεδομένη γραμμή i, οι διαδρομές (i->1, i->2 , … , i- > n) αναλαμβάνονται απο τα thread1, thread2, …, threadn του block αντίστοιχα. Κάθε thread ελέγχει τη διαδρομη (i -> k, k -> …). Επομένως το στοιχείο i -> k είναι κοινό για κάθε thread οπότε υπολογίζεται απο το 1ο νήμα και τοποθετείται σε κοινή μνήμη.

```
int j=blockIdx.y*blockDim.y + threadIdx.y;
        if(j>=n) return;
        int idx=n*blockIdx.x+j;

        __shared__ float best;

        if(threadIdx.y==0) best=dist[n*blockIdx.x+k];
        __syncthreads();

        if(dist[k*n+j]+best<dist[idx]){
                dist[idx]=dist[k*n+j]+best;
        }
```

Τέλος στο τρίτο kernel, τα threads ενός block αναλαμβάνουν πάνω απο μία γραμμή, σε αντίθεση με το δεύτερο, όπου ενα block = μία γραμμή.

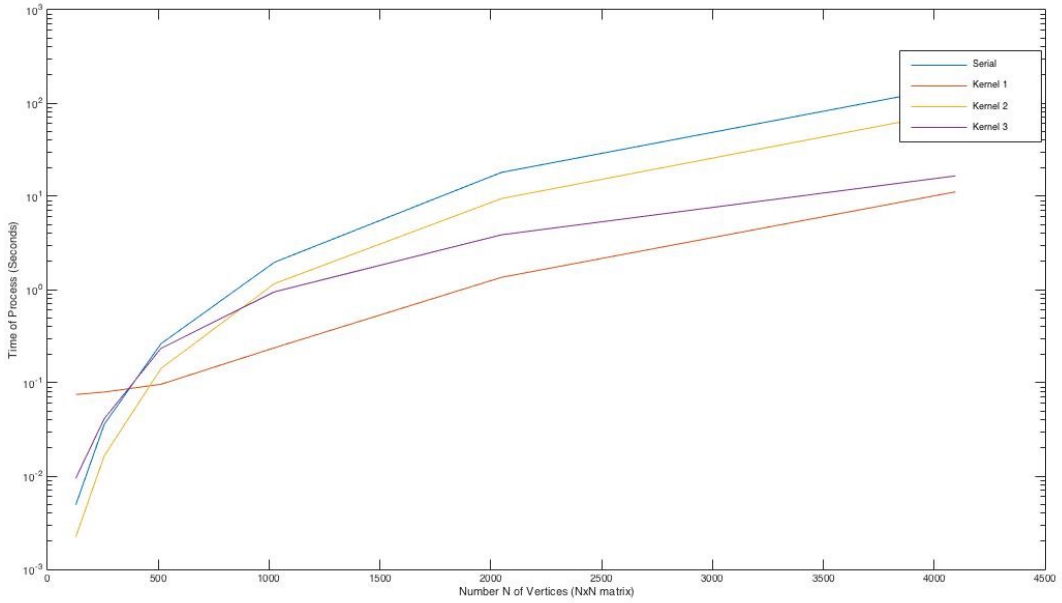
```
int j= blockDim.x*blockIdx.x+threadIdx.x;
   if (j>=n) return;

   __shared__ float best;

   for (int i=0; i<n; i++){
    if (threadIdx.x==0) best=dist[n*i+k];
    __syncthreads();

    if(best+dist[k*n+j]<dist[n*i+j]){
      dist[n*i+j]=best+dist[k*n+j];
    }
   }
```

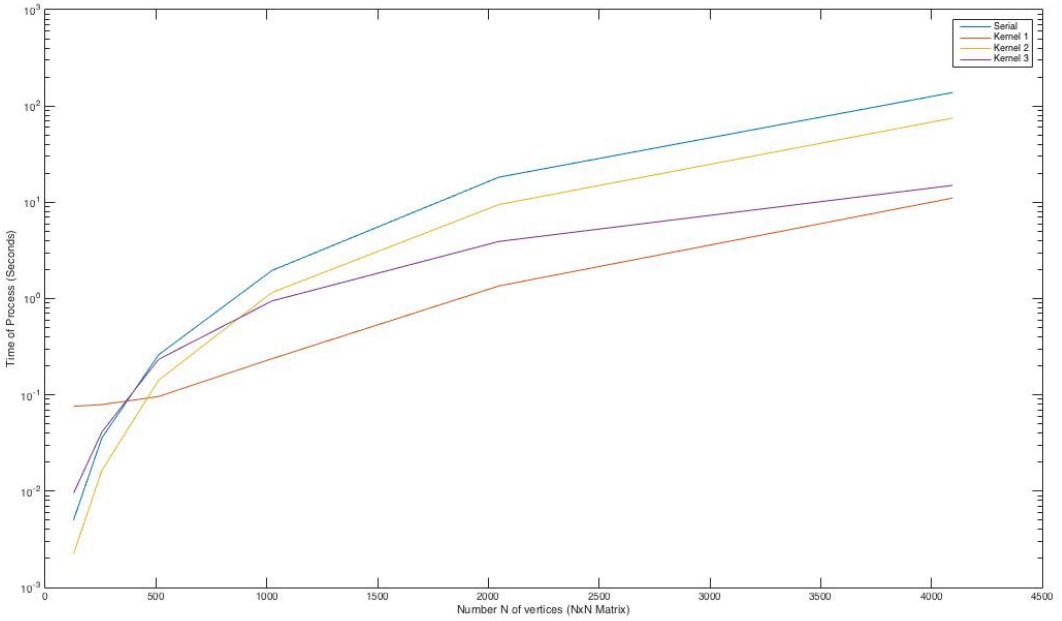
Μετρήσεις

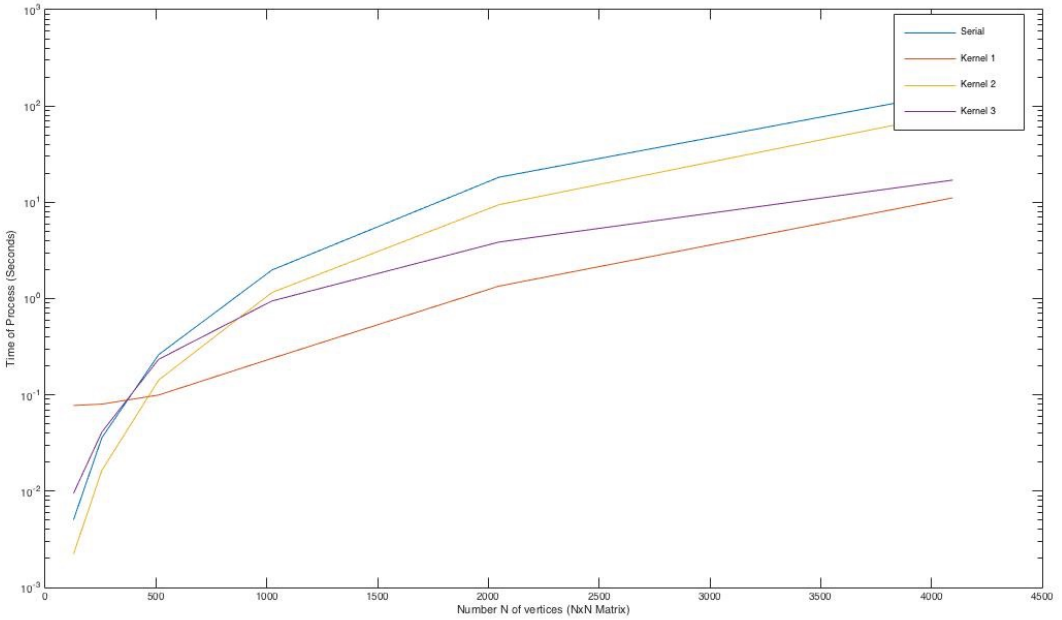
Ο κώδικας δοκιμάστηκε για $n=2^{[7,\ 12]}$, p=[0.33 0.45 0.66] και w=10, στο σύστημα Διάδης. Παρατηρήθηκε οτι όταν το πρόγραμμα εκτελείται παράλληλα έχουμε βελτίωση του χρόνου σε Θ(n).

Ο έλεγχος ορθότητας έγινε ελέγχοντας κάθε στοιχείο του τελικού πίνακα που φτιάχνει κάθε kernel με το αντίστοιχο σειριακό, ώστε να επιβεβαιωθεί οτι έχουν ίδιο αποτέλεσμα.

Παρακάτω φαίνονται τα διαγράμματα απο τις προσομοιώσεις για p=0.33, 0.45 και 0.66:


Serial
Kernel 1
Kernel 2
Kernel 3
Time of Process (Seconds)
Number N of Vertices (NxN matrix)


Serial
Kernel 1
Kernel 2
Kernel 3
Time of Process (Seconds)
Number N of vertices (NxN Matrix)


Serial
Kernel 1
Kernel 2
Kernel 3
Time of Process (Seconds)
Number N of vertices (NxN Matrix)