A Parallel Floyd-Warshall Algorithm on GPU

Roussian R. A. Gaioso, Wellington S. Martins, Edson N. Cáceres (UFMS/Facom), Walid A. R. Jradi and Hugo A. D. Nascimento

Universidade Federal de Goiás/Instituto de Informática

Abstract

The Floyd-Warshall (FW) algorithm is a simple and widely used way to compute shortest paths between all pairs of vertices in an edge weighted directed graph. Despite its simplicity, FW is an efficient algorithm, with very low hidden constants. Moreover, it has a predictable performance regardless of the underlying graph structure. In this work, we present a new parallel algorithm based on sequential FW algorithm. The proposed algorithm uses a combination of data redundancy and persistent threads to reduce CPU-GPU communication. Experiments on general digraphs with arbitrary degree showed impressive reduction of execution time, achieving gains of up to $150\times$ when compared to Floyd-Warshall's sequential implementation, and outperforming state of art FW parallel GPU implementations of Harish *et al.* and Katz and Kider by a factor of $4\times$ and $1.8\times$ respectively.

Motivation

The FW algorithm is considered efficient on storage space usage, with $O(|V|^2)$ complexity, where V represents the set of vertices of the graph. However, its cubic time complexity ($O(|V|^3)$) inspired the development of parallel extensions of FW in order to obtain better performance.

Sequential FW Algorithm

Require: Connected, weighted graph G, represented by the $A^{|V|\times|V|}$ distance matrix.
Ensure: Shortest paths between all pairs of vertices.

1: $n \leftarrow |\ V\ |$;
2: **for** $k = 0$ **to** $n-1$ **do**
3: **for** $i = 0$ **to** $n-1$ **do**
4: **for** $j = 0$ **to** $n-1$ **do**
5: **if** $a_{ik} + a_{kj} < a_{ij}$ **then**
6: $a_{ij} \leftarrow a_{ik} + a_{kj}$
7: **end if**
8: **end for**
9: **end for**
10: **end for**

Division of Data

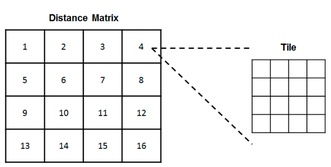

Figure 1: Division of a distance matrix in tiles. Each tile may contain many cells

References

GAIOSO, R. R. et al. Paralelização do algoritmo Floyd-Warshall usando GPU. Simpósio em Sistemas Computacionais. XIV, Brazil, 2013.

GPU Parallelization of FW

The algorithm is divided into **two phases (kernels)** according to the processing order of the data tiles. The **first phase** processes the primary tile (diagonal tile) of the corresponding iteration and respective axes (x and y) and is divided in two steps according to the processing: 1 - storing and executing the primary tile in the shared memory; 2 - blocks of threads execute the algorithm in the row and column of the current primary tile of data. An unconventional use of the CUDA programming model is utilized to keep the properties of the first phase of the algorithm: extension the threads lifetime, making them persistent, and to improve the SMs occupation, that is, making an **intensive occupation** of the SMs. It also prevents that any SM stays idle. Figure 2 and 3 exemplifies the steps of the first phase. In the **second phase** the rest of the tiles (doubly dependent) are processed using the traditional CUDA programming model.

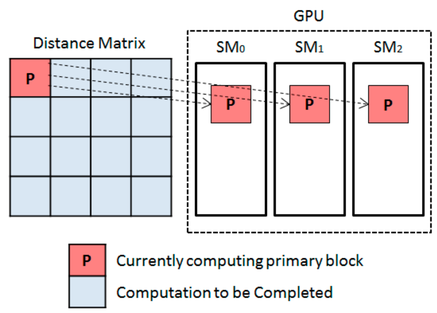

Figure 2: First phase – Step 1.

Figure 3: First phase – Step 2.

Platform Used

Ubuntu 12.04 x86, Intel Core i5 2.80GHz, Tesla C2075, CUDA toolkit-5.5

Conclusion

The speedup obtained via GPU parallelization of FW is due to algorithmic optimizations and the use of the **intensive occupation** technique that allowed the reduction of kernel invocations and consequently communications between CPU and GPU. The implementation showed competitive performance in relation to parallel GPU state of art implementations as the ones proposed by Harish *et al.* and Katz and Kider, with average speedup $4\times$ and $1.8\times$, respectively.

Results

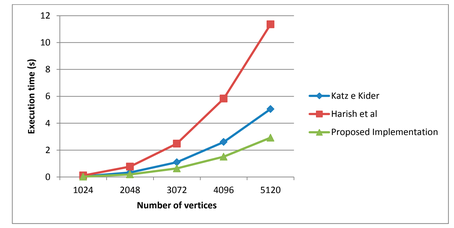

Figure 4: Running time of algorithms for graphs with 1024, 2048, 3072, 4096 and 5120 vertices.

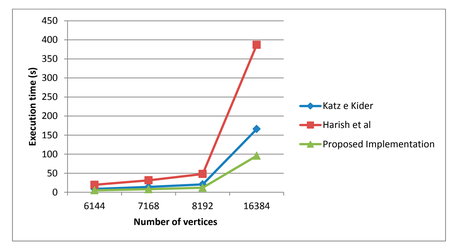

Figure 5: Running time of algorithms for graphs with 6144, 7168, 8192 and 16384 vertices.

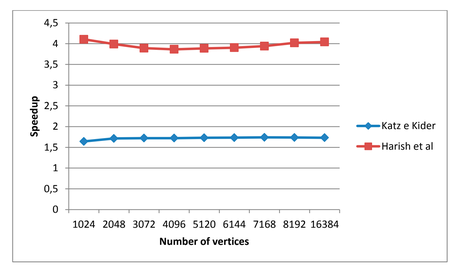

Figure 6: *Speedup* gains compared to the implementations of Harish *et al.* and Katz and Kider.

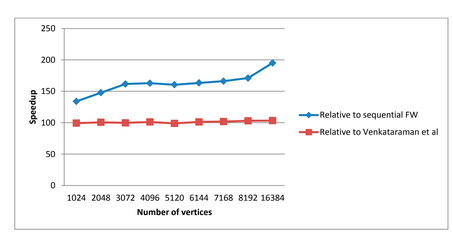

Figure 7: *Speedup* gains compared to the sequential implementations of Floyd-Warshall algorithm