cuRnet

an R package for graph traversing on GPU.

Vincenzo Bonnici, Federico Busato, Stefano Aldegheri, Luciano Cascione, Alberto Arribas Carmena, Muodzhon Akhmedov, Francesco Bertoni, Nicola Bombieri, Ivo Kwee and Rosalba Giugno.

November 11, 2017

Introduction

Plenty of tools have been already developed as packages that extend the R functionality. Some of them regard the analysis of biological networks, which include collections of the most used basic algorithms as well as advanced solutions for specific tasks based on basic algorithms. In many cases, algorithms for biological network analysis are meaningful examples of applications with high-performance computing requirements, for which classical sequential implementations become inappropriate because of the data set complexity. Alternative solutions are given by parallel approaches, and in particular by those based on GPU architectures, which allow reducing the algorithm execution time through the massive parallelism paradigm. Even though some R packages based on GPU kernels have been proposed (e.g., gpuR for algebraic operations), none of them provide algorithm implementations for network analysis. This work presents cuRnet, an R package for analysing biological networks on GPUs. It currently includes an efficient parallel implementation of BFS (Breadth-First Search), SCC (Strongly Connected Components) and SSSP (Single-Source Shortest Paths) customized for biological network analysis (Busato and Bombieri 2016). It has been structured to modularly include, as current and future work, a wide collection of algorithm for biological network analysis.

cuRnet

cuRnet is a package that provides a wrap of parallel graph algorithms developed in CUDA for the R environment. It makes available GPU solutions to R end-users in a transparent way, by including basic data structures for representing graphs, automatic front-end and back-end data converters and, as a starting development point, paralle implementation of BFs, SCC and SSSP algorithms on top of GPU architectures (Busato and Bombieri 2016). cuRnet has been tested on different biological networks provided by the STRINGdb R package (Franceschini et al. 2012). cuRnet outperformed the standard sequential counterpart provided by the iGraph R package.

Package Dependencies and Installation

The `cuRnet` package depends on the following R-packages:

- Rcpp

The `cuRnet` package and its dependencies can be installed on Linux by running the following commands in the R console.

```r
install.packages("devtools", dep = TRUE)
library(devtools)
devtools::install_bitbucket("cuRnet/cuRnet")
```

the `cuRnet` R package rewuires GP-GPU hardware and the Nvidia CUDA toolkit installed on your machine. Please visit the Nvidia CUDA website and the documentation on how to install it on your machine.

`cuRnet` version 0.3 has been tested on Ubuntu 16.04 with CUDA v8.0. The 'cuRnet' software require a GP-GPU hardware with compute capabilities ≥ 3.0.

Visit also the task of CRAN related to High-Performance and Parallel Computing.

Example: creating a cuRnet graph object

cuRnet algorithms run over a cuRnet graph object that can be created from a 3-column `DataFrame`. The dataframe represents edges in the form (source vertex, destination vertex, weight). The first two colums are of type `charatcter`, instead weights are of type `numeric`. Weight column is optional, however it has to be specified for algorithms that require the information, such as SSSP. The following example shows how to create a cuRnet graph object starting from an `iGraph` one.

```
library(igraph)
library(cuRnet)
rg <- sample_fitness_pl(100, 1000, 2.2, 2.3)
cdf <- data.frame( ends(rg, E(rg))[,1], ends(rg, E(rg))[,2] )
colnames(cdf) <- c("from", "to")
my_graph <- cuRnet_graph(cdf)
```

Example: reading networks from a CSV file

The `DataFrame` object to be used as input network for `cuRnet` can be built from values stored in a textual CSV file. The following example shows how to do it. The input file `my_file.csv` is a CSV file having 3 columns named `from`, `to` and `score`. The first line of the CSV file is an header reporting column names. Identifires of vertices are represented as characters, and score must be positive values.

```
library(igraph)
x <- read.table("my_file.csv", header=TRUE)
```

The input CSV file has an header row specifing column names as “from”, “to” and “score”.

```
x$from <- as.character(x$from)
x$to <- as.character(x$to)
x$score <- abs(x$score)
```

The cuRnet graph can be created from the loaded dataframe.

```
my_graph <- cuRnet_graph(x)
```

Example: Breadth-First Search (BFS).

The BFS function traverses the graph via a breadth-first search from a given set of source vertices, and returns depth of visited nodes. The reutrn value is a `NumericMatrix` having a number of rows equal to the number of source vertices, and a number of columns equal to the total number of vertices of th input graph. Every row corresponds to a specific source vertex, and row cells report the depth from the given source to the correspondig graph vertex.

The following example shows how to perfrom BFS. The graph is created randomly by using the `iGraph` Power-Law random generation model.

```
library(igraph)
rg <- sample_fitness_pl(100, 1000, 2.2, 2.3)
cdf <- data.frame( ends(rg, E(rg))[,1], ends(rg, E(rg))[,2] )
colnames(cdf) <- c("from", "to")
library(cuRnet)
cg <- cuRnet_graph(cdf)
```

The example performs BFS by choosing the first 20 vertices of the graph as soruce vertices.

```
sources <- union(cdf$from, cdf$to)[1:20]
bfs <- cuRnet_bfs(cg, sources)
```

Each returned row correspond to a given source vertex.

```r
bfs[1,]
bfs[2,]
bfs[20,]
```

Example: Strongly Connected Components (SCC).

This function computes strongly connected components membership for every vertex of the input graph. The example if performed over a randomly generated graph.

```r
library(igraph)
rg <- sample_fitness_pl(10000, 30000, 2.2, 2.3)
cdf <- data.frame( ends(rg, E(rg))[,1], ends(rg, E(rg))[,2] )
colnames(cdf) <- c("from", "to")
library(cuRnet)
cg <- cuRnet_graph(cdf)
```

The cuRnet SCC call only requires the graph object in order to be performed.

```r
cc <- cuRnet_scc(cg)
```

The returned value is a `NumericMatrix` of 1 row and a number of columns equal to the number of graph vertices. Each cell reports the identifier of the connected component associated with the corresponding vertex. The following line calculates the number of distinct connected components.

```r
length(unique(cc[1,]))
```

Example: Single-Source Shortest Paths (SSSP) from a set of vertices.

The following example shows how to calculate distances and predecessors from a set of source vertices.

The examples uses the human (code 9606) PPI (Protein-Protein Interaction) network downloaded via the STRINGdb R package (version 10). STRING networks have scores ranging from 0 to 1000. A threshold of 900 is used.

```r
library(STRINGdb)
ss <- STRINGdb$new( version="10", species=9606, score_threshold=900)
```

The network data is available as an 'iGraph' object.

```r
library(igraph)
g <- ss$get_graph()
```

For the example, the first 10 vertices of the PPI are choosen as source vertex.

```r
from <- unique(ends(g,E(g))[,1])[1:10]
```

`cuRnet` takes in input a `DataFrame` having 3 columns. The data frame stores the edges of the network as a list of rows. Each row reports the source and destination vertices of the edges and the weight/costs of it. Weights can be eigther integer or decimal values. For this example, the `combined_score` of the `iGraph` object is firstly normalized and the is is used as weight.

```r
x <- data.frame("from" = ends(g,E(g))[,1],
                "to" = ends(g,E(g))[,2],
                "score" = E(g)$combined_score/1000)
```

`cuRnet` functions are available by loading the `cuRnet` library. The `cuRnet_sssp` function takes in input the data frame of the edges and the list of source vertices.

```
library(cuRnet)
ret <- cuRnet_sssp(x, from)
```

The `cuRnet_sssp` function returns a list of two `NumericMatrix` indexed as `"distances"` and `"predecessors"`. Matrix rows are source vertices and colums are network vertices. A entry fo the `distance` matrix is the distance along the shortest path from the source to the destination vertex. A entry fo the `predecessors` matrix is a minimal predecessor of the vertex and along the shortest path.

```
dists = ret[["distances"]]
preds = ret[["predecessors"]]
```

The fowlling lines list distances and predecessors from the first source vertex to every other vertex of the input PPI.

```
dists[1,]
preds[1,]
```

Example: computing only distances of shortest paths

`cuRnet` provides two different funcitons for SSSP computation, `cuRnet_sssp` and `cuRnet_sssp_dists`. Given a network and a set of source vertices, the `cuRnet_sssp_dists` function computes only distance values and ignore predecessors. The function is intended to be faster than the complete one (`cuRnet_sssp`) and to be used for such applications where predecessor information is not needed.

```
library(STRINGdb)
ss <- STRINGdb$new( version="10", species=9606, score_threshold=900)
library(igraph)
g <- ss$get_graph()
from <- unique(ends(g,E(g))[,1])[1:10]
x <- data.frame("from" = ends(g,E(g))[,1],
                "to" = ends(g,E(g))[,2],
                "score" = E(g)$combined_score/1000)
library(cuRnet)
ret <- cuRnet_sssp_dists(x, from)
```

The `cuRnet_sssp_dists` function takes the same arguments of `cuRnet_sssp` but returns just one `NumericMatrix`. the returned matrix corresponds to the one returned by `cuRnet_sssp` and indexed as `"distances`.

The following line of code prints out distances along shortest paths from the first source vertex to every other vertex in the input network

```
ret[1,]
```

REFERENCES

Busato, Federico, and Nicola Bombieri. 2016. "'An Efficient Implementation of the Bellman-Ford Algorithm for Kepler Gpu Architectures'." *IEEE Transactions on Parallel and Distributed System* 27 (8). IEEE: 2222–3.

Franceschini, Andrea, Damian Szklarczyk, Sune Frankild, Michael Kuhn, Milan Simonovic, Alexander Roth, Jianyi Lin, et al. 2012. "STRING V9. 1: Protein-Protein Interaction Networks, with Increased Coverage and Integration." *Nucleic Acids Research* 41 (D1). Oxford University Press: D808–D815.