Accelerating Simulations in R using Automatically Generated GPGPU-Code

Frank Kramer[1*], Andreas Leha[1], Tim Beissbarth[1]

1. Department of Medical Statistics, University Medical Center Göttingen, Göttingen, Germany
*Contact author: frank.kramer@med.uni-goettingen.de

Keywords: GPGPU, CUDA, Simulation, simR2CUDA

Newly developed classifiers, estimators, and other algorithms are often tested in simulations to assess power, control of alpha-level, accuracy and related quality criteria.

Depending on the required number of simulation runs and the complexity of algorithms, especially the estimation of parameters and the drawing of random numbers under certain distributions, these simulations can run for several hours or even days. Such simulations are embarrassingly parallel and, therefore, benefit massively from parallelization. Not everyone has access to clusters or grids, though, but highly parallel graphics cards suitable for general purpose computing are installed in many computers. While there is a distinct number of maintained *R*-packages available (Eddelbuettel2011) that are able to interface with GPUs and allow the user to speed up computations, integrating these methods in simulation runs is often complicated and mechanisms are not easily understood.

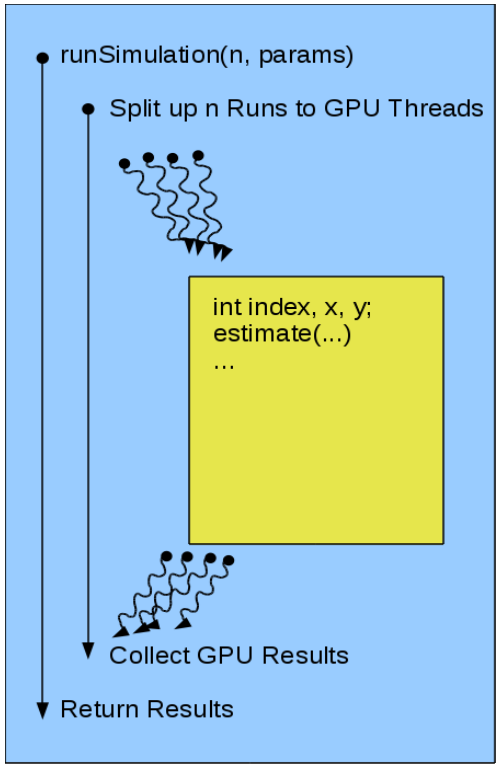

Figure 1: Illustration of generic process flow in generated code

We present a new package that acts as a wrapper for *CUDA*-implementations and offers a systematic way for a statistician to accelerate simulations written in *R*. After including the package **simR2CUDA** the user defines the algorithm for a single simulation run in an *R* function and passes the function, other required parameters and the number of simulation runs to a compiler function. The allowed syntax concerning flow-control for the function defining a single simulation run is restricted to comply with GPU and implementation limitations. Given that a *CUDA* environment is correctly installed and configured, a wrapper function, representing the whole simulation, and a *CUDA* function, representing an individual simulation step, are generated and integrated into *R* using `dyn.load`. Upon calling the generated code the simulation wrapper, a simple *C* function, splits up the independent simulation runs for parallel execution on GPU threads and collects and returns the results, after execution of all n runs on the GPU has finished (see Figure 1).

References

Eddelbuettel (2011). CRAN Task View: High-Performance and Parallel Computing with R, http://cran.r-project.org/web/views/HighPerformanceComputing.html.