GPU-accelerated Normal-To-Anything (gpu-NORTA)

Specifications for GPU course project

AG Schissler

February 12, 2018

Abstract

I'm writing an R package called `gpusim` that aims to accelerate Monte Carlo studies (simulations) using GPU-acceleration for large data sets. The first functionality of the package will be to simulate multivariate constructions with arbitrary correlation structure and marginal distributions. I've done this using the publicly available R package `gpuR` that interfaces with `openCL`, but it is not optimized or user-friendly yet. I like the idea of using `openCL` to increase accessibility for non-NVIDIA users, but I'm find with `CUDA` if it is much faster or it is the core instructional objective of the class.

Bullet points of software specifications

- The core user base will be statisticians and data scientists and so whatever program is written must interface well with R.
- The function will take in d marginal distribution specifications. This means a distribution name, e.g. "binomial", and their distributional parameters (a named vector). E.g., $size = 10, p = 0.3$ for a binomial marginal distribution.
- The function takes in an arbitrary correlation matrix, Σ and a desired number of simulation replicates, n (usually n is at least 2000). Σ can be assumed to be semi-positive definite by construction. (But sometimes numerical errors can give small negative eigenvalues and this should be handled by the program - I can supply an algorithm to fix this).
- A typical use case for the function will have $d \approx 20,000$. So Σ will have $\approx 4e8$ entries. The algorithm should scale to larger sets within reason/limitations of the computing reasons.
- The algorithm uses NORTA strategy:
  - Compute the matrix X that is $n \times d$ with each entry a realization from the standard normal distribution (i.e., $x_{ij} \sim N(0,1)$).
  - Compute $Q = PD^{1/2}$ from Σ, where P are the normalized eigenvectors of Σ and D is a diagonal matrix with eigenvalues of Σ as entries.
  - Obtain a the correlated multivariate normal Y by multiplying $Y = QX$.
  - Then use the probability integral transform get uniform marginals (`pnorm` in R) for each column $j = 1, 2, ...d$.
  - Finally apply the marginal distributions the user specified for each column, e.g. `qbinom`.
- We could also start directly with uniform marginally (like a copula idea), but I just did NORTA on my first try.
- The `eigen` step is expensive. I wasn't able to parallelize it and so I run it on a CPU with enough RAM as a preprocessing step.
- My algorithm basically involved finding the largest partitions of the matrices P and D that I could fit on a contingent block of RAM on the GPU device and performing the matrix multiplication then reassembling on my CPU.
- I'm flexible on the algorithm - whatever is fast, easy to use, and minimizes error propagation is great with me.