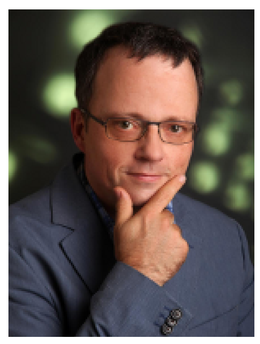
Implementing GPU computations in Octave and statistical applications



Albrecht Gebhardt[1] **& Gunter Spöck**[2]

Institute for Statistics, Alpen-Adria University Klagenfurt, Austria

[1]albrecht.gebhardt@aau.at, [2]gunter.spoeck@aau.at

Österreichische Statistische Gesellschaft
STATISTICS DAYS 2015
Wednesday, 21st – Friday, 23rd October
Statistics Austria, Guglgasse 13, A-1110 Wien

Abstract

GNU Octave[1] is a free software targeting scientific computing with focus on numerics and linear algebra. To some extent it shares its syntax with MATLAB, a commercial tool widely used in science and engineering. During the last years parallel computing has become more and more affordable, not only but to a large extent by the advent of end user programmable GPU hardware providing hundreds or even thousands of specialized computing cores. Such a parallel computing hardware is provided by Nvidia® with its CUDA architecture.

Soon after the availability of the CUDA SDK some parallel processing toolboxes for MATLAB became popular (GPUmat[2] and more recent the Parallel Computing Toolbox™). For Octave only a small proof-of-concept implementation exists (MMGPUOctave, see Teng (2008)), lacking all the comfortable features of its MATLAB counterparts. We present an extension of this work which tries to fill this gap, providing the end user with a special GPU matrix class where all relevant operations like multiplication and inversion are executed in parallel on the GPU. This works mainly by providing an interface to cuBLAS[3], a GPU accelerated implementation of the standard BLAS operations.

We will show how this new data type can help to speed up operations in statistical applications involving a large number of linear algebra operations. As example an application in monitoring network design for Trans-Gaussian Kriging is presented.

Parallel Matrix Operations

One basic idea behind parallel matrix operations is to partition the matrices into blocks, e.g. like in this simple example, and to split the matrix product AB into separate terms, see e.g. Chtchelkanova et al. (1997):

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} B_{11} & 0 \\ B_{21} & 0 \end{pmatrix} + \begin{pmatrix} A_{11} & A_{12} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & B_{12} \\ 0 & B_{22} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & 0 \\ B_{21} & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} 0 & B_{12} \\ 0 & B_{22} \end{pmatrix}$$

which can be split further by applying

$$\begin{pmatrix} A_{11} & A_{12} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} B_{11} & 0 \\ B_{21} & 0 \end{pmatrix} = \begin{pmatrix} A_{11} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} B_{11} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & A_{12} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ B_{21} & 0 \end{pmatrix}$$

and so on. Now each computing node only has to multiply two smaller sized matrices A_{ij} and B_{kl} and these results have to be added afterwards. This principle is in use in most enhanced BLAS implementations like in the Intel® Math Kernel Library[4], AMD® Compute Library[5] as well as in the Nvidia® cuBLAS Library.

Implementation in Octave

Octave enables developers to write functions not only as m-files in Matlab-like syntax but also in C++ to provide so called oct-files, dynamically loadable libraries (similar to MEX files in MATLAB). Through this oct-file API it is possible to interface octave with functions from the cuBLAS library. Using this approach we implemented the new data types GPUsingle, GPUdouble and GPUspdouble along with GPU enabled matrix multiplication operators. This work reused parts of the MMGPUoctave package which already provides basic access to cuBLAS functions.

A sample call with octave 4.0.0 under Linux on a 16GB AMD FX-8320E computer and a Nvidia GTX 980 Ti with 2816 CUDA cores and 6GB RAM on GPU gives the following timings:

```
Octave:1> A = rand(1024*4,1024*3);
octave:2> B = rand(1024*3,1024*5);
octave:3> A1 = GPUsingle(A);
octave:4> B1 = GPUsingle(B);
octave:5> A2 = GPUdouble(A);
octave:6> B2 = GPUdouble(B);
octave:7> tic, C = A*B; toc
Elapsed time is 14.3948 seconds.
octave:8> tic, C1 = A1*B1; toc
Elapsed time is 0.549073 seconds.
octave:9> tic, C2 = A2*B2; toc
Elapsed time is 0.956822 seconds.
```

It can be seen that single precision operations are even faster, partly due to the smaller amount of data to be copied from and to the graphics card. Note that the first timing (`A*B`) doesn't use CUDA but the ATLAS library[6], when used with standard non-optimized BLAS this timing increases to 90 seconds.

Application in Statistics

Many standard statistical applications are based on some sort of matrix algebra, e.g. solving systems of normal equations by QR decomposition for parameter estimation in linear models or spatial predictions, solving eigenvalue problems in multivariate statistics and so on. All these methods can benefit from accelerated low level matrix operations provided by a parallelized BLAS implementation.

Trans-Gaussian kriging

We consider a mean square continuous (m.s.c.) and isotropic random field $\{Z(x) : x \in \mathbf{X} \subseteq \mathbb{R}^2\}$ such that

$$Y(x) = \mathbf{f}(x)^T\beta + \varepsilon(x), \quad \mathrm{E}\varepsilon(x) = 0,$$

$$\mathrm{Cov}(Z(x), Z(y)) = C(||x - y||); \quad x, y \in \mathbf{X}.$$

which leads to the well known setup for universal kriging.

For Trans-Gaussian kriging apply a Box-Cox transformation

$$g_\lambda(z) = \begin{cases} \frac{z^\lambda - 1}{\lambda} & : \quad \lambda \neq 0 \\ \log(y) & : \quad \lambda = 0 \end{cases}.$$

to the positive valued data, see Spöck et al. (2009). This approach targets the deficiencies of classical universal kriging that arise from applying it to non-normal data, especially underestimation of the kriging variance.

[1]http://octave.org
[2]http://sourceforge.net/projects/gpumat/
[3]https://developer.nvidia.com/cuBLAS
[4]https://software.intel.com/en-us/intel-mkl
[5]http://developer.amd.com/tools-and-sdks/opencl-zone/acl-amd-compute-libraries/

Experimental Design

To incorporate uncertainty in the covariance function we apply spectral decomposition according to (Yaglom (2012)). This leads to a mixed linear model

$$Z(x) \approx \mathbf{f}(x)^T\beta + \mathbf{g}(x)^T\alpha + \varepsilon_0(x).$$

where $\mathbf{g}(\cdot)$ is made up of cosine-sine-Bessel surface harmonics. Now treating all parameters as random leads to a Bayesian spatial linear model

$$Z(x) = \mathbf{h}(x)^T\gamma + \varepsilon_0(x),$$

where

$$\mathbf{h}(x) = \begin{pmatrix} \mathbf{f}(x) \\ \mathbf{g}(x) \end{pmatrix}, \gamma = \begin{pmatrix} \beta \\ \alpha \end{pmatrix},$$

$$\mathrm{E}\gamma = \begin{pmatrix} \mu \\ 0 \end{pmatrix}, \mathrm{Cov}(\gamma) = \begin{pmatrix} \Phi & 0 \\ 0 & \mathbf{A} \end{pmatrix} =: \Gamma.$$

$\varepsilon_0(x)$ is white-noise with variance σ_0^2, $\mathbf{A} = \mathrm{Cov}(\alpha)$, $\mathbb{E}(\beta) = \mu \in \mathbb{R}^r$ and $\mathrm{Cov}(\beta) = \Phi$.

Continuous designs are just probability measures ξ on $\mathbf{X}$ and may be rounded to exact designs d_n. Introducing the continuous Bayesian information matrix

$$\mathbf{M}_B(\xi) = \int_{\mathbf{X}} \mathbf{h}(x)\mathbf{h}(x)^T\xi(dx) + \frac{\sigma_0^2}{n}\Gamma^{-1} \quad \text{and}$$

$$\mathbf{U} = \int_{\mathbf{X}} \mathbf{h}(x)\mathbf{h}(x)^T dx,$$

leads to the extended design functional

$$\Psi(\mathbf{M}_B(\xi)) = \mathrm{tr}(\mathbf{U}\mathbf{M}_B(\xi)^{-1}).$$

In Spöck and Pilz (2015) the Smith and Zhu (2004) 95% predictive interval criterion

$$\int_{\mathbf{X}} \mathrm{E}(\text{length of predictive interval at } x_0)dx_0.$$

is extended to this Bayesian setup, allowing to model uncertain covariance functions.

Exact design algorithm

The basic algorithm for calculating spatial sampling designs is an exchange algorithm from experimental design theory.

- Exchange algorithm
  Step 1. Use some initial design $d_{n,1} = \{x_{1,1}, \ldots, x_{n,1}\} \in \mathbf{X}^n$ of size n.
  Step 2. Beginning with $s = 1$ form the design $d_{n+1,s} = d_{n,s} + (x_{n+1,s})$ by adding the point

  $$x_{n+1,s} = \arg \min_{x \in \mathbf{X}} \Psi(\mathbf{M}_B(d_{n,s} + (x)))$$

  to $d_{n,s}$.
  Then form $d_{n,s}^j = d_{n+1,s} - (x_{j,s}), j = 1, 2, \ldots, n+1$ and delete that point $x_{j^*,s}$ from $d_{n+1,s}$ for which

  $$\Psi(\mathbf{M}_B(d_{n,s}^{j^*})) = \min_{j \in \{1,\ldots,n+1\}} \Psi(\mathbf{M}_B(d_{n,s}^j))).$$

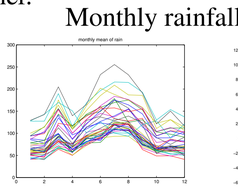
  Repeat Step 2 until the point to be deleted is equivalent to the point to be added.
- Generation of an initial design
  Step 1. Choose $x_1 \in \mathbf{X}$ such that $x_1 = \arg \min_{x \in \mathbf{X}} \Psi(\mathbf{M}_B((x)))$, and set $d_1 = (x_1)$.
  Step 2. Beginning with $i = 1$, find x_{i+1} such that $x_{i+1} = \arg \min_{x \in \mathbf{X}} \Psi(\mathbf{M}_B(d_i + (x)))$ and form $d_{i+1} = d_i + (x_{i+1})$.
  Continue with i replaced by $i + 1$ until $i + 1 = n$.
  Step 3. If $i + 1 = n$ then stop and take $d_{n,1} = \{x_1, \ldots, x_n\}$ as an initial design.
- Combination of the above two algorithms

Using the dataset below a single innermost step of the exchange algorithm (calculation of a single estimated interval length) takes around 10 seconds without GPU and around 5 seconds with GPU under the same settings as above, overall duration is several hours.

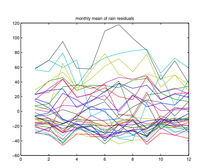
Upper Austria Rainfall Network

The data set considered is a rainfall data set from Upper Austria. The monitoring network comprises 36 locations. Average monthly rainfall has been measured at each location starting in January 1994 and ending in December 2009. Covariance functions are for each of the 12 months proportional to each other.

Monthly rainfall at 36 stations:

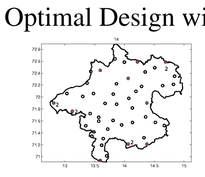


Average

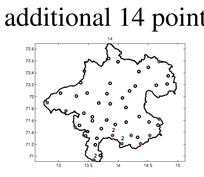


Resiudual

Optimal Design with additional 14 points:

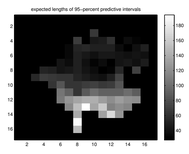


Gaussian kriging, space filling

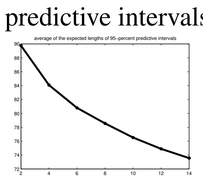


Trans-Gaussian kriging, more close locations, better cov. estimation

Smith & Zhu 95% predictive intervals.



Estimated lengths



Decrease in average

References

Chtchelkanova, A., J. Gunnels, G. Morrow, J. Overfelt, and R. A. van de Geijn (1997). Parallel implementation of blas: general techniques for level 3 blas. *Concurrency: Practice and Experience 9*(9), 837–857.

Smith, R. L. and Z. Zhu (2004, November). Asymptotic theory for kriging with estimated parameters and its application to network design.

Spöck, G., H. Kazianka, and J. Pilz (2009). Bayesian trans-gaussian kriging with log-log transformed skew data. In J. Pilz (Ed.), *Interfacing Geostatistics and GIS*, pp. 29–43. Springer Berlin Heidelberg.

Spöck, G. and J. Pilz (2015). Incorporating covariance estimation uncertainty in spatial sampling design for prediction with trans-gaussian random fields. *Frontiers in Environmental Science 3*(39).

Teng, G. C. (2008). Matrix multiplication on GPU in octave. Technical report, Advanced Computing Group.

Yaglom, A. (2012). *Correlation Theory of Stationary and Related Random Functions: Supplementary Notes and References*. Springer Series in Statistics. Springer New York.