GPU-Accelerated Dense Covariance Matrix Generation for Spatial Statistics Applications

Zipei Geng[1], Sameh Abdulah[2], Hatem Ltaief[2], Ying Sun[1,2], Marc G. Genton[1,2], and David E. Keyes[2]
[1] Statistics Program, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
[2] Extreme Computing Research Center (ECRC), King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

Introduction

- Spatial statistics applications aim to study patterns, distributions, and relationships of variables across different locations in a given spatial region.
- In the present era, the abundance of massive spatial data volumes makes High-Performance Computing (HPC) indispensable for effectively handling the processing of high-resolution maps.
- Software such as *ExaGeoStat* offers a high-performance computing framework tailored for large-scale spatial data challenges, providing effective solutions for processing vast amounts of data.
- *ExaGeoStat* relies on the Maximum Likelihood Estimation (MLE) algorithm to model spatial data through statistical inference. This statistical framework assumes that the spatial measurements are realizations of Gaussian spatial random field, i.e., $Z(\boldsymbol{s}_1), ..., Z(\boldsymbol{s}_N) \sim \mathcal{F}(\boldsymbol{s})$, where $\boldsymbol{s}$ refers to spatial locations. In the context of our research, we denote mean function $m(\boldsymbol{s})$ and covariance/kernel matrix
$$\boldsymbol{\Sigma}(\boldsymbol{\theta}) := \big(C(\boldsymbol{s}_i - \boldsymbol{s}_j; \boldsymbol{\theta})\big)_{ij}, i, j \in 1, ..., N,$$
where N is the number of spatial locations, $\boldsymbol{s}_i - \boldsymbol{s}_j$ is the spatial lag vector of location $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$, and $C(\cdot\,; \boldsymbol{\theta})$ is a stationary parametric covariance/kernel function w.r.t. spatial lag vectors.
- The problems involve an extensive number of measurements distributed regularly or irregularly across a geographical region, requiring the generation of a dense covariance matrix $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ with a size of $N \times N$ in each step of the MLE process:
$$l(\boldsymbol{\theta}) = -\frac{N}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}(\boldsymbol{\theta})| - \frac{1}{2}\boldsymbol{Z}^\top \boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}\boldsymbol{Z}.$$
- *ExaGeoStat* primarily focuses on efficiently computing the underlying linear algebra operations on the dense covariance matrix. However, in this study, our objective is to accelerate the matrix generation process on GPU, which becomes time-consuming on the CPU when dealing with a large number of locations. Previous approaches have relied solely on CPU-based matrix generation.

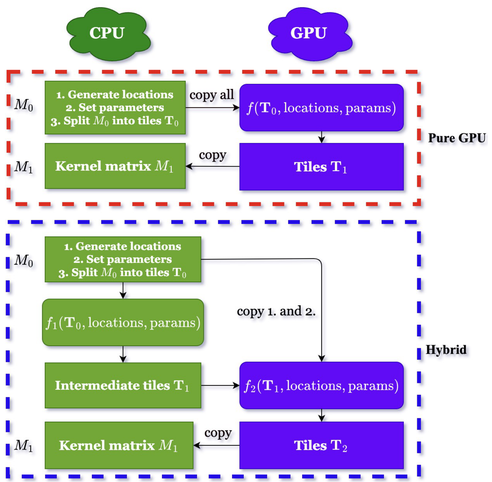
Contributions

- We introduce two matrix generation schemes, each accommodating different kernel functions:
  1. **Pure GPU**: the matrix kernel function relies on mathematical operators fully supported on the GPU.
  2. **Hybrid**: offload unsupported mathematical operators (e.g., the modified Bessel function of the second kind) on the CPU.
- We used the CUDA programming model for kernel matrix generation. This study highlights the two schemes, each with a specific example
- The red dashed line follows the **Pure GPU** approach. Matrix $\mathbf{M}_0$ is first divided into tiles $\boldsymbol{T}_0 = (T_{01}, ..., T_{0n_t})$. CPU initializes variables, sets parameters, and generates locations. These variables are transferred to GPU. Kernel function $f(\cdot)$ is applied to tiles $\boldsymbol{T}_0$ to generate the tiles $\boldsymbol{T}_1$ and thus the kernel matrix $\mathbf{M}_1$.
- The blue dashed line follows a **Hybrid** approach. It starts by initializing the matrix $\mathbf{M}_0$, dividing it into $\boldsymbol{T}_0$, generating locations, and setting parameters. Intermediate tiles $\boldsymbol{T}_1$ are then created, evaluating CPU-only math functions $f_1(\cdot)$ to fill matrix entries. On the GPU, function $f_2(\cdot)$ is applied, multiplying with $f_1(\cdot)$ to set tile entries as $f_1(\cdot) \times f_2(\cdot)$, resulting in tiles $\boldsymbol{T}_2$ and kernel matrix $\mathbf{M}_1$.

Algorithm 1 Algorithm for Matérn kernel matrix generation

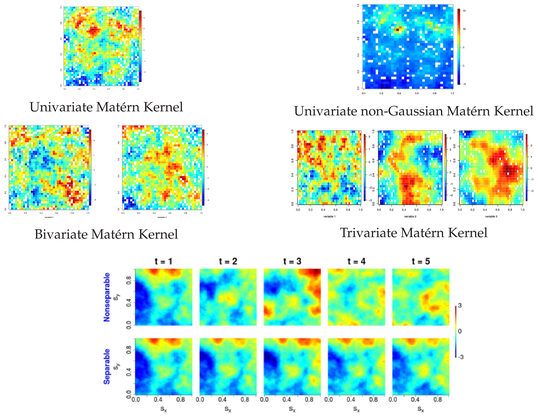
1: Initialize matrix M_0 with entries zero and divided into tiles $T_1, ..., T_{n_t}$, where n_t is the number of tiles
Require: locations $s_1, ..., s_N$, parameters $\boldsymbol{\theta} = (\sigma, \beta, \nu)$
2: **for** $i = 0$ to n_t - 1 **do**
3: Calculate the distance $\|\boldsymbol{h}\|_2$ of the $j_i k_i$-th entry of T_i using $\|\boldsymbol{s}_{j_i} - \boldsymbol{s}_{k_i}\|_2$
4: Fill the $j_i k_i$-th tile entry $T_{ij_ik_i}$ with $f_1(\|\boldsymbol{h}\|_2) := \mathcal{K}_\nu(\|\boldsymbol{h}\|_2/\beta)$ on CPU
5: Repeat until all entries in tile T_i are filled with $f_1(\|\boldsymbol{h}\|_2)$
6: Let $f_2(\|\boldsymbol{h}\|_2) := \sigma^2/(2^{\nu-1}\Gamma(\nu))\,(\|\boldsymbol{h}\|_2/\beta)^\nu$
7: $T_{ij_ik_i} \leftarrow T_{ij_ik_i} \times f_2(\|\boldsymbol{h}\|_2)$ on GPU
8: Repeat until all entries in tile T_i are filled with $f_1(\|\boldsymbol{h}\|_2) f_2(\|\boldsymbol{h}\|_2)$
9: **end for** ▷ Note that this for loop is parallelizable using multiple cores

Proposed Schemes



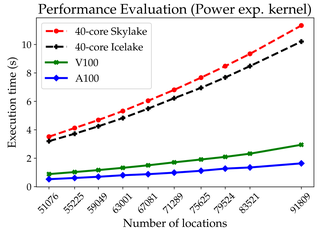
Examples of Generated Geospatial Data

- Given N locations uniformly randomly distributed, the covariance matrix $\boldsymbol{\Sigma}$ can be built using the Matérn kernel function. This covariance matrix can be used to generate observations $\tilde{\mathbf{Z}}$ represented at the generated N locations as follows:
$\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$ Cholesky factorization;
$\tilde{\mathbf{Z}} = \mathbf{L}\boldsymbol{e}$ where $e_i \sim \mathcal{N}(0, 1), i = 1, ..., N$.

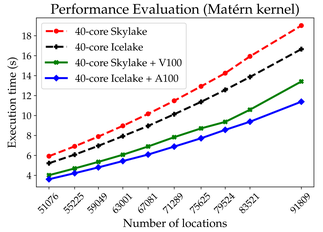


Performance Results

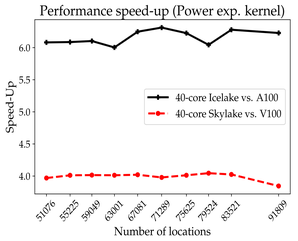
- We evaluated our proposed implementations by conducting a comparison with the CPU-based matrix generation on two distinct systems. The first machine consists of a dual-socket 20-core Intel Skylake Xeon Platinum 8260 CPU running at 2.40 GHz and an NVIDIA Tesla V100 GPU SXM2 with 32GB memory. The second machine comprises a dual-socket 28-core Intel Icelake Xeon Gold 6330 running at 2.00 GHz and one NVIDIA A100 Tensor Core GPU with 40GB memory.
- We have chosen two representative examples to illustrate the two matrix generation schemes. For the **Pure GPU** approach, we utilize the power exponential kernel function, which does not require CPU-only special functions,
$$C_1(\|\boldsymbol{h}\|_2) = \sigma^2 \exp(-\|\boldsymbol{h}\|_2^{\beta_2}/\beta_1)$$
For the **Hybrid** implementation, we use Matérn kernel function which has modified Bessel function of the second kind ($\mathcal{K}_\nu$) that cannot be computed on GPU,
$$C_2(\|\boldsymbol{h}\|_2) = \frac{\sigma^2}{2^{\nu-1}\Gamma(\nu)}\left(\frac{\|\boldsymbol{h}\|_2}{\beta}\right)^\nu \mathcal{K}_\nu\left(\frac{\|\boldsymbol{h}\|_2}{\beta}\right)$$
- To conduct the performance comparison, we repeated each experiment five times and calculated the average elapsed time from the results.

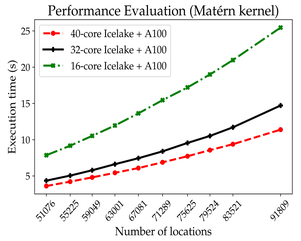


The pure GPU approach demonstrates a significant speed-up, generating the power exponential kernel matrix approximately 4X to 6X faster than the CPU. Moreover, as the number of locations increases, the advantages of utilizing GPU become even more evident. Notably, the performance of the NVIDIA A100 GPU surpasses that of the NVIDIA V100 GPU.



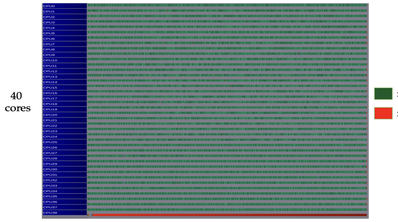
In the case of the Matérn kernel, the hybrid GPU/CPU approach demonstrated superior performance compared to the CPU. However, since some computations were still performed on the CPU, the advantages of the hybrid approach were diminished when compared to the pure GPU approach. We also showed the scalability of our hybrid implementation across various core counts, extending up to 40 cores on a shared-memory system architecture.



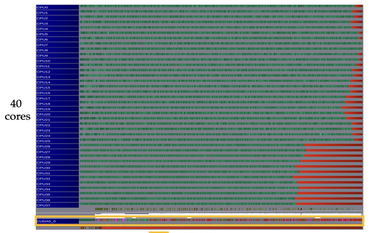


Performance Profiling

- We used StarPU FxT and ViTE software to profile creating a Matérn kernel matrix on a dual-socket 20-core Intel Skylake machine with an NVIDIA V100 GPU. The graph shows the trace while running the generation function exclusively on CPUs. It reveals that without cross-unit synchronization needs, all CPUs stay fully active (shown in green).



- The graph below displays execution traces on a dual-socket 20-core Intel Skylake with an NVIDIA V100 GPU. The profiling graph indicates that each tile is promptly sent for $f_2(\cdot)$ processing on the GPU once $f_1(\cdot)$ processing on the CPU is done, demonstrating CPU-GPU synchronization. Asynchronous execution is evident across all CPUs. Similar patterns emerge in results when using an A100 GPU.



Conclusion and Future Work

- The utilization of GPU-based matrix generation significantly reduces the elapsed time compared to CPU-based approaches. Host-device and device-device communications can be further optimized in the future.
- It can be further applied in zero-gradient optimization with high-dimensional matrices, including geospatial studies and other studies that use Gaussian regression models.
- The proposed two schemes allow for the implementation of additional kernels, such as the bivariate Matérn, non-Gaussian Matérn, and others.
- The method of evaluating special mathematical functions, like the Bessel function, demonstrated in tools like the GNU Scientific Library, can be re-implemented using CUDA to enable direct execution on the GPU.

References

[1] Abdulah, Sameh, et al. "ExaGeoStat: A high performance unified software for geostatistics on manycore systems." IEEE Transactions on Parallel and Distributed Systems 29.12 (2018): 2771-2784.
[2] Powell, Michael JD. "The BOBYQA algorithm for bound constrained optimization without derivatives." Cambridge NA Report NA2009/06, University of Cambridge, Cambridge 26 (2009).
[3] Augonnet, Cédric, et al. "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures." Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference, Delft, The Netherlands, August 25-28, 2009. Proceedings 15. Springer Berlin Heidelberg, 2009.
[4] ViTE. V1.2, https://solverstack.gitlabpages.inria.fr/vite/, INRIA, last accessed: August 3, 2023.