NCAR | National Center for
UCAR | Atmospheric Research

Incorporating MAGMA into the `fields' spatial statistics package

John Paige
Isaac Lyngaas
Vinay Ramakrishnaiah
Dorit Hammerling
Raghu Kumar
Douglas Nychka

NCAR Technical Notes
NCAR/TN-519+STR

National Center for
Atmospheric Research
P. O. Box 3000
Boulder, Colorado
80307-3000
www.ucar.edu

NCAR/TN-519+STR
NCAR Technical Note

2015-08

Incorporating MAGMA into the `fields' spatial statistics package

John Paige
Climate Sciences Department,
Lawrence Berkeley Laboratory, Berkeley, CA
Isaac Lyngaas
Department of Scientific Computing,
Florida State University, Tallahassee, FL
Vinay Ramakrishnaiah
Department of Electrical and Computer Engineering,
University of Wyoming, Laramie, WY
Dorit Hammerling
Institute for Mathematics and Applied Geosciences,
National Center for Atmospheric Research, Boulder, CO
Raghu Kumar
Computational and Information Systems Laboratory,
National Center for Atmospheric Research, Boulder, CO
Douglas Nychka
Institute for Mathematics and Applied Geosciences,
National Center for Atmospheric Research, Boulder, CO

Computational and Information Systems Laboratory (CISL)
Institute for Mathematics Applied to the Geosciences (IMAGe)

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
P. O. Box 3000
BOULDER, COLORADO 80307-3000
ISSN Print Edition 2153-2397
ISSN Electronic Edition 2153-2400

Incorporating MAGMA into the ‘`fields`’ spatial statistics package

John Paige, Isaac Lyngaas, Vinay Ramakrishnaiah, Dorit Hammerling, Raghu Kumar, Douglas Nychka [*]

August 19, 2015

Abstract

In this report we describe how to incorporate the Cholesky decomposition from the Matrix Algebra on GPU and Multicore Architectures (MAGMA) library into some of the calculations of the ‘`fields`’ spatial statistics package in `R`. We provide MAGMA installation instructions as well as demonstrations of performance when applied to simulated datasets and the `CO2` dataset available in `fields`. While there are other spatial statistics packages in `R` using parallelism, such as `bigGP` and `parspatstat`, none to our knowledge directly incorporates GPUs or other coprocessors. Our code is timed on Caldera computational nodes in the National Center for Atmospheric Research’s Yellowstone supercomputing environment. We find that for 40,000 $\times$ 40,000 matrices the MAGMA-accelerated decomposition has a 30.7 and 46.2 times speedup for 1 and 2 GPU implementations respectively over `chol`, the standard Cholesky decomposition function in `R` (with settings allowing `R` programmers to use our accelerated function like they would `chol`). The speedups are greater when using in-place calculations where the original matrix is overwritten and not copied. In that case, the equivalent speedups are 41.8 and 54.4 times for in place decompositions on one and two GPUs respectively. We also time a simple spatial analysis workflow with maximum likelihood estimation with up to over 23,000 observations, where accelerated workflows achieved approximately 4.2 and 4.3 times speedup when using 1 and 2 GPUs respectively over a corresponding unaccelerated workflow. As problem size increases, speedups improve, and the 2 GPU decompositions perform increasingly well compared to their corresponding 1 GPU implementations. Performance for 2 GPU decompositions is slower than with 1 GPU in some cases due to additional communication overheads and data dependencies in the Cholesky decomposition algorithm, and will be explored further in Ramakrishnaiah et al. (2015).

Keywords: Spatial statistics, Kriging, High-Performance Computing, GPU, MAGMA

[*]John Paige, is Research Associate, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley CA 94720 (paigejo@uw.edu), Isaac Lyngaas, is PhD student, Florida State University, 600 W College Ave, Tallahassee FL 32306 (irl14@my.fsu.edu), Vinay Ramakrishnaiah, is PhD student, University of Wyoming, 1000 E University Ave, Laramie, WY 82071 (vramakr1@uwyo.edu), Dorit Hammerling, is Project Scientist II, National Center for Atmospheric Research, PO Box 3000, Boulder CO 30307-3000 (dorith@ucar.edu), Raghu Kumar, is Associate Scientist II, National Center for Atmospheric Research, PO Box 3000, Boulder CO 30307-3000 (raghuraj@ucar.edu), Douglas Nychka, is Senior Scientist, National Center for Atmospheric Research, PO Box 3000, Boulder CO 30307-3000 (nychka@ucar.edu).

Contents

List of Figures

1 Introduction

The top 23 supercomputers on the November 2014 Green500 list (gre 2014), a list of the 500 most energy efficient supercomputers on a GFLOP/W basis, use hardware accelerators in addition to traditional Central Processing Units (CPUs). Additionally, two of the top ten fastest supercomputers use Graphics Processing Units (GPUs), and over ten percent of all the fastest 500 supercomputers use GPU accelerators (top 2014). The prevalence of GPUs and multi-core CPUs even extends to most personal computers sold today. It is therefore important that computing software take advantage of these heterogeneous architectures, which utilize both CPUs and coprocessors such as GPUs, especially when working on computationally intensive problems.

The 'Kriging' algorithm, central to spatial data analysis, is $\mathcal{O}(n^3)$ in computation time for n spatial observations. The biggest computational problem lies in the required Cholesky decomposition, which is also $\mathcal{O}(n^3)$, and is associated with the evaluation of the data likelihood function given the covariance parameters (Katzfuss and Cressie 2012). Since high-resolution spatio-temporal datasets with millions or tens of millions of observations are readily available (Katzfuss and Cressie 2012), this computational bottleneck related to dataset size poses a major problem. The Matrix Algebra on GPU and Multicore Architectures (MAGMA) library is able to accelerate the Cholesky decomposition by using parallelized code optimized for heterogeneous computing systems (Tomov et al. 2009). By incorporating MAGMA into '`fields`', a widely used `R` package for spatial statistics (Nychka et al. 2015), the authors are able to accelerate the Cholesky decompositions required by likelihood calculations in `fields` and thereby reduce total spatial analysis computation time considerably.

While other spatial analysis `R` packages also use parallel computation, such as `parspatstat` and `bigGP`, none to our knowledge use coprocessors (GPUs or the Xeon Phi, for example), instead focusing only on CPUs (Lee 2013, Paciorek et al. 2013). Although reducing computation time on CPUs is useful as well, it is very important to be able to exploit the power of GPUs considering their prevalence in high-performance and personal computing (top 2014, gre 2014).

There are several packages for `R` meant to accelerate basic matrix algebra routines. Examples include the `magma` package (Smith 2015), `pbdDMAT` (Schmidt et al. 2015), and a set of `R` packages called High Performance Linear Algebra in `R` (HiPLAR) that accelerates `R` matrix algebra routines on heterogeneous architectures (Dongarra et al. 2012a, Dongarra et al.

2012b, Nash and Szeremi 2015).

The `magma` `R` package uses the shared memory computing capabilities of MAGMA for heterogeneous architectures and supports LU, QR, and other matrix decompositions. It does not accelerate the Cholesky decomposition, however, so it is less useful for the main statistical computations in `fields` (Smith 2015, Nychka et al. 2015).

The `pbdDMAT` package (Schmidt et al. 2013b) is one of many `R` packages offered by the *programming with big data in R* (pbdR) organization (Ostrouchov et al. 2012). It depends on Scalable Linear Algebra PACKage (ScaLAPACK), Message Passing Interface (MPI), `pbdMPI` (Chen et al. 2014a), `pbdSLAP` (Chen et al. 2014b), and `pbdBASE` (Schmidt et al. 2013a) and has massive scalability (Schmidt et al. 2015, Schmidt et al. 2012). `pbdR` also offers other useful tools such as its MPI profiler, called `pbdPROF` (Chen et al. 2014c, Ostrouchov et al. 2012). Unfortunately, the pbdR organization does not yet offer accelerated computation on coprocessors, providing only accelerated functionality for CPUs.

The set of `R` packages known as HiPLAR uses accelerated routines from MAGMA the Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA) library, hwloc (Broquedis et al. 2010), CUDA, and optimized Basic Linear Algebra Subprograms (BLAS) to achieve a high level of performance in `R` (Agullo et al. 2009, Dongarra et al. 2012a, Dongarra et al. 2012b, Nash and Szeremi 2015). At first glance HiPLAR may seem ideal for this study. Unfortunately we discovered through communications with Dr. Montana, HiPLAR's Principal Investigator, as well as our own experimentation that the project has not been maintained for several years, it is difficult to install due to its many dependencies, and there have been no published peer-reviewed results documenting its performance from its authors.

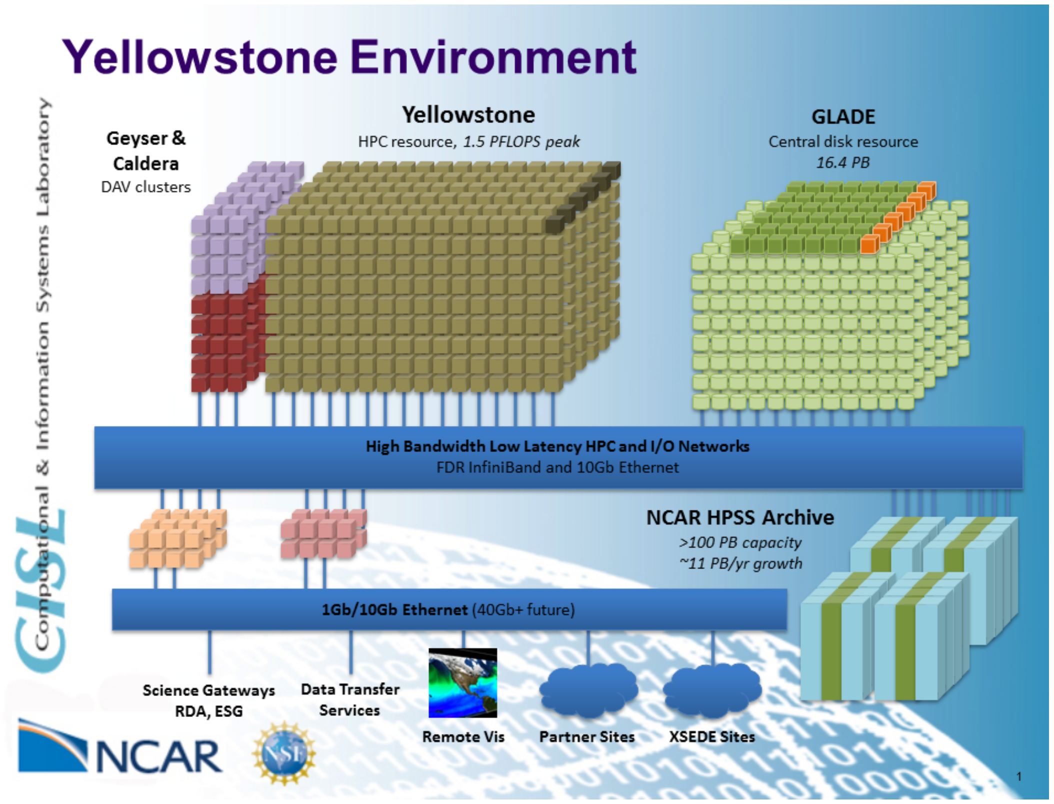


Figure 1: The Yellowstone computing environment as found in Yel (2015b). Caldera nodes are in red near the top-left. Data Analysis and Visualization (DAV) clusters have installed data visualization programs (Yel 2015a).

This technical report will focus on accelerating `fields` on *Caldera* nodes in the National Center for Atmospheric Research's (NCAR's) *Yellowstone* supercomputing environment, which is depicted in Figure 1. We use Caldera computation nodes because they have GPUs, unlike the main Yellowstone supercomputer nodes. Each Caldera node is dual-socket, including two 3.5 compute capability NVIDIA Tesla K20X GPUs and two Intel Xeon E5-2670 (Sandy Bridge) CPUs. All dependencies for the software used in this report aside from MAGMA are already installed on the Yellowstone computing environment, such as CUDA, MKL, and ICC. These are available as modules "cuda", "mkl", and "intel" respectively.

The main body of this report presents a simple way for Yellowstone `R` users to incorporate MAGMA's GPU-accelerated Cholesky decomposition into `R` code and `fields` as well as demonstrations of accelerated timings. Section 2 contains information on the MAGMA computational library, instructions for how to install it on the Yellowstone environment, and a brief example illustrating how to use the installed MAGMA functions in `R`. Section 3 describes how `fields` has been accelerated using MAGMA and how to use the accelerated functions in `R`. We supply installation instructions for using MAGMA with `R`, and we benchmark some spatial statistics computations with the '`CO2`' dataset from `fields`. Section 4 reports the timing and speedup of accelerated versus unaccelerated `fields`. Here the Cholesky decomposition itself, the `mKrig` function in `fields`, and a simple spatial analysis workflow are all timed separately. Lastly, Section 5 interprets the timing and speedup results on Caldera, and describes avenues for future work. Code for timing a basic accelerated spatial analysis (accelerated and default) is given in the appendix.

2 MAGMA

The MAGMA library uses coprocessors in conjunction with CPUs to maximize the performance of the computational system. It uses accelerated GPU functions from CUDA (CUD 2015), a parallel programming language for NVIDIA GPUs. MAGMA parallelizes the Cholesky decomposition by dividing the matrix into blocks and sending different blocks to different processors. Block sizes are automatically set based on the specific architecture of the system (Tomov et al. 2009, Agullo et al. 2009).

2.1 Installing MAGMA on Caldera

The following installation instructions assume the user runs all commands on the Yellowstone computing environment login nodes or on Caldera nodes (installation instructions for Mac computers will be provided in future work). It also assumes the use of a UNIX *bash* shell rather than tcsh, the default shell on Yellowstone. If you are using tcsh, substitute "`.tcshrc`" for "`.bashrc`", "`setenv`" for "`export`", and a space for the "`=`" sign in `export` commands. For instance, rather than

```
export CUDADIR=/ncar/opt/cuda/6.5
```

use:

```
setenv CUDADIR /ncar/opt/cuda/6.5
```

Also if you are using tcsh do not create a "`.bash_profile`" file.

1. Download `magma-1.6.1.tar` from
   `http://icl.cs.utk.edu/magma/software/index.html`
2. Transfer the file to your home directory
3. `tar -xvf magma-1.6.1.tar`

4. Create a `.bash_profile` file in your home directory, if you have not already done so. This will enable personalized settings for your terminal at login.

5. Make sure it contains the command:

   ```
   source ~/.bashrc
   ```

6. Create a `.bashrc` file in your home directory if you have not already done so

   NOTE: in the following commands, substitute the version of CUDA you are using for "6.5"

7.
   ```
   cat << EOF > .bashrc
   export MKLROOT=/ncar/opt/intel/psxe-2015_update3/composer_xe_2015.3.187/mkl/
   export CUDADIR=/ncar/opt/cuda/6.5
   export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/ncar/opt/intel/psxe-2015_update3/\
   composer_xe_2015.3.187/mkl/lib/intel64:/ncar/opt/cuda/6.5/lib64
   export PATH=${PATH}:/ncar/opt/cuda/6.5/bin
   EOF
   ```

8. Note: the following command is not necessary, but it will improve performance

   ```
   cat << EOF > .bashrc
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_avx.so
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_avx2.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_avx512.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_avx512_mic.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/l\
   ibmkl_blacs_intelmpi_ilp64.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_blacs_intelmpi_lp64.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_cdft_core.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_core.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_def.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_gf_ilp64.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_gf_lp64.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_gnu_thread.so:$LD_PRELOAD
   export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
   libmkl_intel_ilp64.so:$LD_PRELOAD
   ```

```
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_intel_lp64.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_intel_thread.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_mc.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_mc3.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_rt.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_scalapack_ilp64.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_scalapack_lp64.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_sequential.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_avx.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_avx2.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_avx512.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_avx512_mic.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_cmpt.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_def.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_mc.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_mc2.so:$LD_PRELOAD
export LD_PRELOAD=/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64/\
libmkl_vml_mc3.so:$LD_PRELOAD
EOF
```

9. You will now be able to use MAGMA from every new terminal window you open and at login.

10. Log out and log back in. The changes will now have come into effect.

11. `cd ~/magma-1.6.1`

12. Modify the "`make.inc.mkl-icc`" file to fit the specific needs of the system you will be running MAGMA on. Replace the lines:

```
CFLAGS = -O3 $(FPIC) -DADD_ -Wall -openmp -DMAGMA_SETAFFINITY -DMAGMA_WITH_MKL
```

with:

```
GPU_TARGET = Kepler sm35
CFLAGS = -O3 $(FPIC) -DADD_ -Wall -openmp -DMAGMA_SETAFFINITY -DMAGMA_WITH_MKL
-Xlinker -shared
```

13. `cp make.inc.mkl-icc make.inc`

14. Start an interactive session on Caldera with the command:

    `bsub -Is -q caldera -W 01:00 -n 1 -P XXXXXXXXX $BASH`

    where your project code should be substituted for "`XXXXXXXXX`". Note that "`-W 01:00`" submits an hour-long interactive job, and "`-n 1`" submits a job with a single core. A GPU is necessary from this point onward.

15.
    ```
    module load R/3.2.1
    module load mkl/11.2.3
    module load cuda
    module load intel/15.0.3
    module unload ncarcompilers/1.0
    ```

16. Still in the MAGMA root directory, build MAGMA with the `make` command (or, if this is not your first time running this command, run `make clean` first)

17. Once MAGMA is installed, `cd` into the `testing` directory and try:

    ```
    ./testing_dpotrf -c --ngpu 1
    ./testing_dpotrf_m -c --ngpu 2
    ```

    Note that the `-h` option gives help for the different options you can pass to the testing functions. If the commands are all successful, MAGMA should be correctly installed for use on Caldera nodes.

A similar installation process would likely work for using MAGMA on Geyser computational nodes on Yellowstone as well, since they also contain GPUs. Each Geyser node only has one GPU, however, with slightly different specifications than the GPUs on Caldera nodes, and we have not tested our installation instructions on Geyser.

3 Using MAGMA with `fields` and `R`

3.1 Incorporating MAGMA into `fields` and `R`

In order to use MAGMA within `R` and `fields`, there are a few more necessary installation steps. We have written `R` functions to use the MAGMA-accelerated Cholesky decomposition from a publicly available "Bitbucket" repository online. The repository also has MAGMA-accelerated versions of several functions from `fields` as will be discussed in detail later. The following instructions show how to download, compile, and use our code from `R` on Caldera nodes.

1. If either ∼`/.R` or ∼`/.R/Makevars` does not exist (Run "`ls -a`" in Terminal to show hidden files starting with a "."), create them and edit `Makevars`. Add the following lines to `Makevars`:

   ```
   PKG_CFLAGS= -I/glade/u/home/username/path/magma-1.6.1/include \
   -I/ncar/opt/intel/psxe-2015_update3/mkl/include \
   -I/ncar/opt/cuda/6.5/include -DHAVE_CUBLAS -DADD_ -openmp -O3
   PKG_LIBS=/glade/u/home/username/path/magma-1.6.1/lib/libmagma.a \
   -L/ncar/opt/intel/psxe-2015_update3/mkl/lib/intel64 \
   -lmkl_intel_lp64 -lmkl_core -lmkl_sequential -lmkl_vml_avx -lmkl_def \
   -L/ncar/opt/cuda/6.5/lib64 -lcublas -lcudart -openmp ${LIB_NCAR}
   ```

   The path to the MAGMA folder and the version of MAGMA will depend on the steps you have taken previous to this and your specific system.
   NOTE: be careful of the specific version of MAGMA and CUDA you are using and to adjust the above paths by changing any version numbers when appropriate.

2. If the git repository with our accelerated fields code is not yet cloned, clone it with the command:

   ```
   git clone https://bitbucket.org/jpaige/fieldsmagma_all.git
   ```

3. `cd` into the cloned repository, then `cd` to `fieldsMAGMA/magmaCholesky/`

4. Run:

   ```
   R CMD SHLIB *.c
   ```

5. `cd` to the repository's "`fieldsMAGMA`" directory in the main repository and change the line:

   ```
   thisDir = "~/git/fieldsmagma_all/fieldsMAGMA"
   ```

   in "`fieldsMAGMA.r`" as well as "`RwrapperMAGMA.r`" so that the directory is correct for your system and installation

6. Still in the `fieldsMAGMA` directory, begin an `R` session and run the following lines of code to check you have installed the accelerated code correctly:

```
#get magmaChol MAGMA-accelerated Cholesky decomposition
source("fieldsMAGMA.r")

#This function generates an n x n positive definite matrix
getMatForChol = function(n) {
  #get a random matrix
  A = matrix(rnorm(n^2), n, n)

  # A*A^T is positive definite
  return(A %*% t(A))
}

Sigma = getMatForChol(5)
U = magmaChol(Sigma)
SigmaEst = t(U)%*%U

# The following printed matrices should be the same:
print(Sigma)
print(SigmaEst)

# See how much faster MAGMA is:
Sigma = getMatForChol(2000)
system.time(U1 <- magmaChol(Sigma))[3]
system.time(U2 <- chol(Sigma))[3]

#check to make sure MAGMA Cholesky agrees with default
max(abs(U1 - U2))
```

In order to test using MAGMA to perform the Cholesky decomposition with n GPUs at once, substitute “`U = magmaChol(Sigma, nGPUs=n)`” for “`U = magmaChol(Sigma)`”. View `magmaChol.r` directly to see its various arguments that can be set to improve or customize its performance.

7. If both matrices with the above code are the same, then the code should be working correctly, and you will be able to use MAGMA to perform the Cholesky decomposition from `R`.

3.2 Accelerated `fields` Methods

MAGMA’s Cholesky decomposition can be loaded into `R` by sourcing ‘`fieldsMAGMA.r`’ in the `fieldsMAGMA` directory, and we have written accelerated versions of the following methods in `fields` version 8.2 (also available by sourcing ‘`fieldsMAGMA.r`’):

Replaced Method	$\Rightarrow$	**New Method**
`mKrig.MLE`	$\Rightarrow$	`mKrigMAGMA.MLE`
`mKrig`	$\Rightarrow$	`mKrigMAGMA`

Both new functions, `mKrigMAGMA.MLE` and `mKrigMAGMA`, operate exactly the same as the corresponding unaccelerated methods from `fields`, except the new functions have extra arguments specifying how many and which GPUs to use in MAGMA's Cholesky decomposition. The `nGPUs` argument specifies the number of GPUs to use in the calculations and defaults to zero. The `MAGMArank` argument sets the specific GPU ID to use and defaults to zero as well, but this option is currently only used when `nGPUs` is set to one. Otherwise, GPUs with ranks zero to `nGPUs-1` are used. In our experience, computers with one GPU should set `MAGMArank` to zero (the default value), and systems with n GPUS should set `MAGMArank` to any number between zero and $n-1$. In order to perform multiple Cholesky decompositions on GPUs at once, the user can set which GPU to use for each decomposition with this method in combination with a parallel programming language such as Message Passing Interface (MPI) or `Rmpi`, which is an `R` package implementing MPI (Yu 2014).

3.3 Spatial Analysis Workflow for `CO2` Dataset Example

The '`CO2`' dataset is available in the `fields` package and can be added to the `R` workspace with the simple command "`data("CO2")`" once `fields` is loaded. This section will use a portion of the full dataset, taking a subset of its 26,633 observations at latitude/longitude coordinates spread across the earth using an expanding window centered at zero degrees latitude and longitude. Code is available in `fieldsmagma_all/fieldsMAGMA/examples/mKrigWorkflowComparison1.r` and shown in the appendix to perform a comparison of accelerated and non-accelerated `fields` computation times for a basic spatial analysis workflow assuming an exponential covariance model. Under the assumption of exponential covariance, the correlation between two observations with locations x_i and x_j is:

$$k(x_i, x_j) = \begin{cases} e^{-|x_i - x_j|/\theta}, & i \neq j \\ e^{-|x_i - x_j|/\theta} + \lambda, & i = j \end{cases},$$

Where θ is the exponential scale parameter and λ is a spatial smoothing parameter that can be interpreted as a noise to signal ratio. The spatial analysis workflow has the following steps and will be used throughout this work for timing studies:

1. Source MAGMA-accelerated `fields` code and the `CO2` dataset
2. Take a subset of the data with approximately 10,000 observations
3. Fixing $\theta = 8$ (the exponential range parameter, in degrees), compute a maximum likelihood estimate for λ (the Kriging noise parameter) by evaluating the likelihood at 10 values of λ equally spaced on a log scale from 10^{-5} to 1 (10 Cholesky decompositions)
4. Estimate spatial process surface with the maximum likelihood estimate (MLE) of λ and with $\theta = 8$ (1 Cholesky decomposition)
5. Predict estimate on an 80×80 grid
6. Estimate theoretical prediction uncertainties at grid points by making five random draws from the conditional distribution of the spatial process given the observations. This uses the `sim.mKrig.approx` method in `fields`

On Caldera, the non-accelerated workflow took 7 minutes 23 seconds, whereas the accelerated workflow took 4 minutes 6 seconds. We therefore achieved a 1.8 times speedup using one GPU. This was based on an analysis with 9,943 observations, but as the number of observations increases, the Cholesky decomposition becomes a more significant part of the workflow computation time. Hence, the larger the problem, the bigger the speedup in general when using our accelerated code for `fields` (as we will see in the following sections).

4 Timing Results

All timing is performed in `R` 3.2.1 compiled with Intel compiler 15.0.3 and serial Intel Math Kernel Libraries (MKL) using CUDA 6.5 on Yellowstone's Caldera computational nodes. Each Caldera node is dual-socket, including two 3.5 compute capability NVIDIA Tesla K20X GPU cards each with 6 GB of 2.6 GHz GDDR5 memory and 2,688 732 MHz cores. Each node also has two 8-core 2.6-GHz Intel Xeon E5-2670 (Sandy Bridge) CPUs. `R`'s default Cholesky decomposition uses the `chol` function, which is a wrapper for the LAPACK Cholesky decomposition. On Caldera, `chol` was compiled with serial (single core) MKL and therefore already performs better than it would if `R` were installed with default settings. We tested the computation time and speedup of the accelerated version of the Cholesky factorization using exponential model covariance matrices with range parameter 0.2 for gridded data on a $[0, 1] \times [0, 1]$ square. Since the Cholesky decomposition computation time is independent of the specific covariance matrix (as long as its dimensions are fixed), our timings for the decomposition are independent of the specific covariance function. We test accelerated versions of `mKrig` and a complete example spatial analysis using the `CO2` spatial dataset in `fields` (shown in Section 3.3), also assuming an exponential covariance model. Just as in the example above, spatial workflows included calculating the maximum likelihood value of λ for a fixed exponential scale parameter, θ, via spline interpolation on log likelihoods. More specifically, spline interpolation is performed on ten exponentially spaced values of λ to calculate the MLE. `mKrig` is then run at the MLE λ and at $\theta = 8$, a prediction surface is calculated, and the spatial error process is estimated. Each call to `mKrig` requires a single Cholesky decomposition, and so 11 are required in total. For all functions timed, accelerated version are timed with 10 trials for each number of spatial observations (with an untimed warmup run to start the GPUs), and 5 trials for the unaccelerated versions. Plotted times are the averages of all trial times.

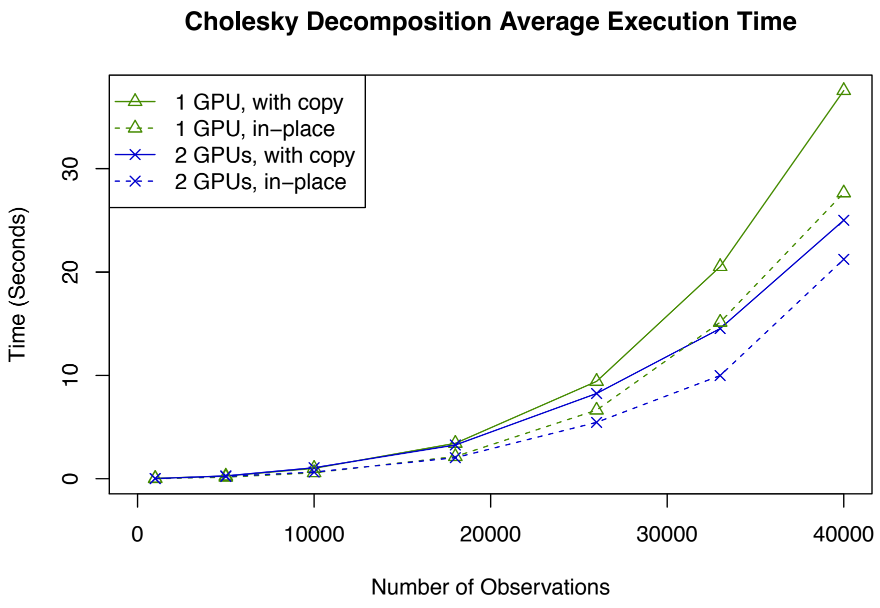
We determine the timings of *in-place* Cholesky decompositions as well as ones that perform a *deep copy* of the matrix to be factored. The in-place decompositions that we test perform all calculations without copying the input covariance matrix to be factored. Hence, the original covariance matrix is lost, but time is saved since no matrix copy is necessary. In the in-place timings the lower triangle of the factor matrix is also not set to zero to save time. These are the settings used in the accelerated `mKrigMAGMA` and `mKrigMAGMA.MLE` functions. The deep copy implementations first copy the matrix to be decomposed and then perform the Cholesky factorization on the copied matrix. Since each covariance matrix is not reused after being decomposed into its Cholesky factors in `mKrig` and throughout the spatial analysis workflow, we use in-place implementations of the MAGMA-accelerated Cholesky decomposition. In order for the `mKrig` timings to be more representative of the timings of the likelihood optimization process, we do not estimate the effective degrees of freedom of the spatial process since it is not necessary in likelihood maximization, and we therefore set `mKrig` argument “`find.trA`” to `FALSE`.

As shown in Figure 2, for problems with 10,000 observations, the accelerated Cholesky decomposition performs 18.6 times faster than the serial implementation (`chol`) when using one GPU with a deep-copy of the covariance matrix. When using two GPUs, the speedup is 17.5 times. When calculations are performed in-place, the one GPU decomposition achieves a 32.3 times speedup and the two GPU decomposition has a speedup of 29.0 times. The two GPU implementations quickly become faster than the corresponding 1 GPU implementations after 10,000 observations, however. For 40,000 observations, the 2 GPU deep copy and in-place calculations reached speedups of 46.2 and 54.4 times respectively, while the corresponding 1 GPU decompositions achieved speedups of 30.7 and 41.8 times. Rather than taking on the order of 20 minutes for 40,000 observations as in the case of `chol`, the accelerated decompositions took on the order of ∼30 seconds. In general, speedups improve with the number of observations.

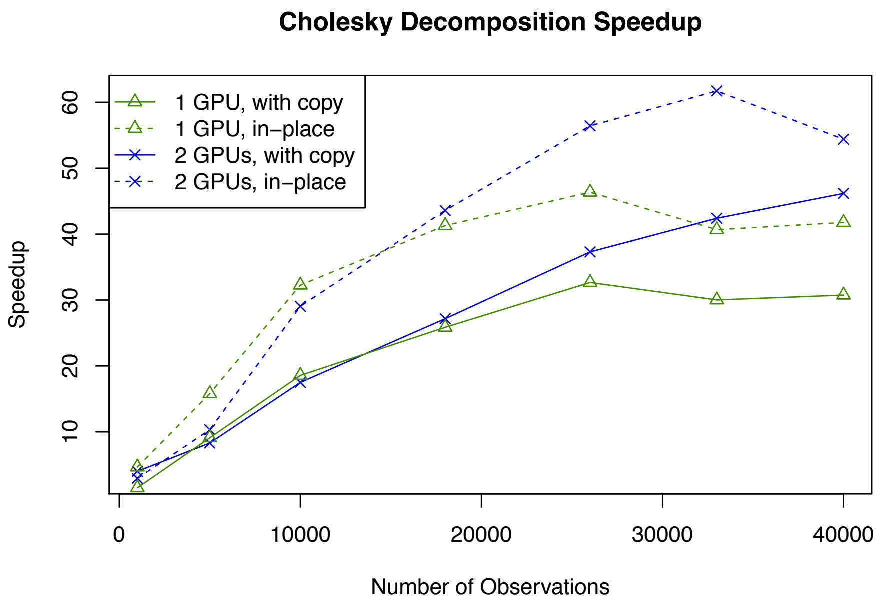
As the number of spatial observations is increased, the 2 GPU implementations of the Cholesky decomposition see improved speedups compared to the 1 GPU implementations. We believe much of this is due to communication overheads when using 2 GPUs. The Cholesky decomposition involves data dependency—namely the results of certain calculations are required for later calculations. This data dependency means that results calculated on one GPU may be required for calculations on the other GPU, which would require additional communication. This communication takes time and creates additional overheads, especially in multi-GPU Cholesky decompositions.

The accelerated `mKrig` function is also sped up considerably, although less so than the Cholesky decomposition itself due to `mKrig`'s unaccelerated subroutines. Computation times for `mKrig` and `mKrigMAGMA`, as visible in Figure 3, are reduced by at least half if there are at least approximately 7,500 observations, and appear to increase linearly throughout the domain of the plot to 5.2 and 5.3 times speedup for just over 26,600 observations in the one and two GPU implementations. The speedup improves as the number of observations increases, but using two GPUs does not provide more than one percent speedup over using one GPU until there are at least 23,000 observations. Using 1 GPU is at most 6 percent faster than using two GPUs, and that only occurs for approximately 1,000 spatial observations, and the 1 GPU decomposition is only significantly more than 1 percent faster than the two GPU implementation for fewer than 10,000 observations.

We see a similar pattern in Figure 4 for spatial analysis workflow computation times. Computation times are reduced by half above approximately 7,500 observations for both implementations, speedups increase with problem size, and timings between one and two GPUs are never different by more than 1 percent. Speedups are at least 4.2 and 4.3 times for both 1 and 2 GPU implementations when the number of spatial observations increases above 23,000 and increase approximately linearly over the range of observations tested.

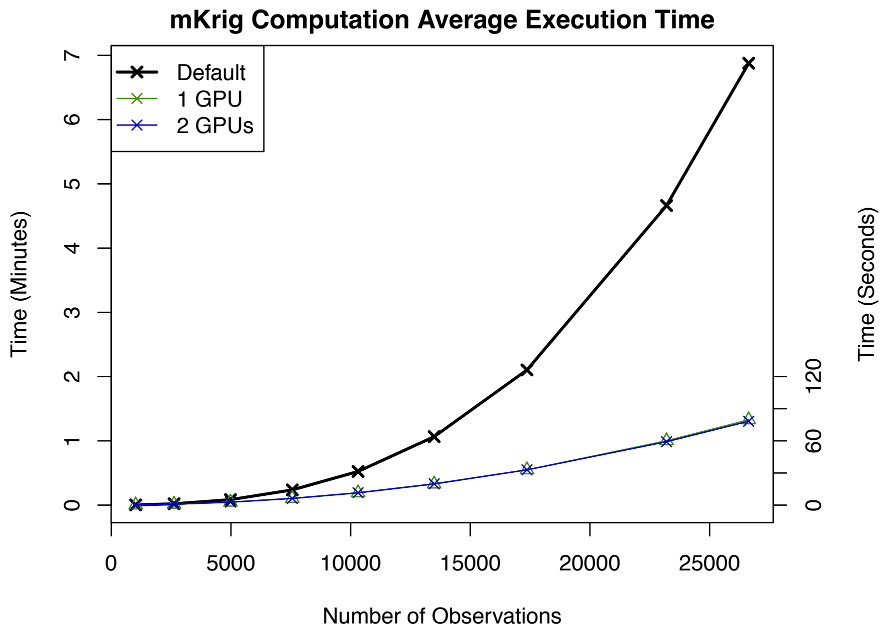


(a)

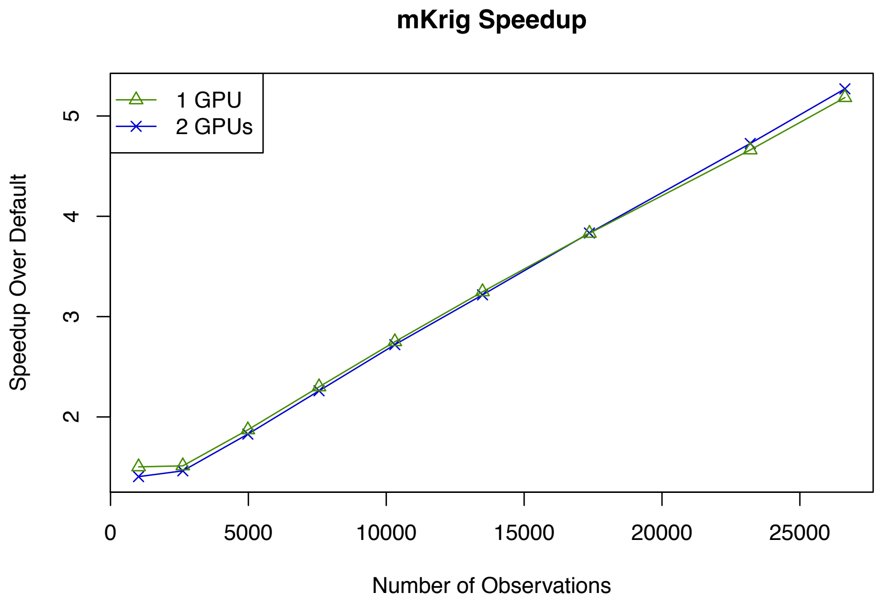


(b)

Figure 2: Average execution times of 1 and 2 GPU accelerated implementations of the Cholesky decomposition (green and blue respectively) (a) and speedup versus the default implementation (b) are plotted. Times for in-place calculations without setting the lower triangular portion of the matrix to zero are shown as dotted lines, while non-in-place (listed as "with copy") calculation times when setting the lower triangular portion of the decomposition to zero are shown as solid lines.

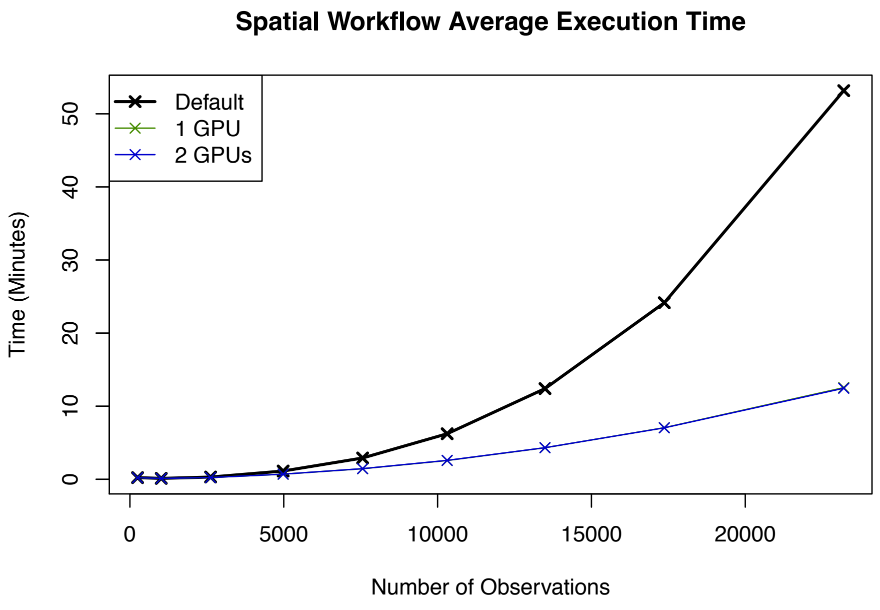


(a)

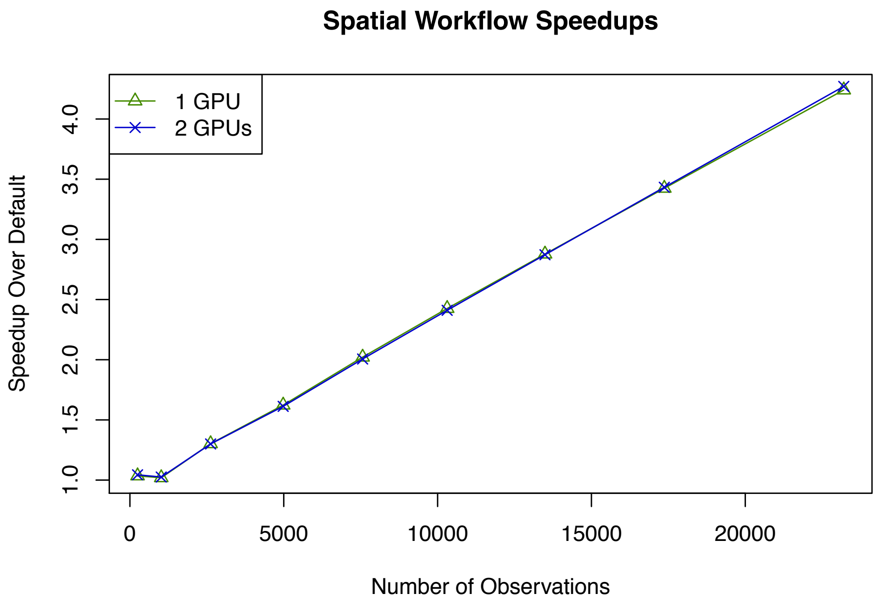


(b)

Figure 3: Average computation time of the `mKrig` function in `fields` for default (black) and accelerated implementations (green and blue) (a) and speedup versus the default implementation (b) are plotted. All calculations for accelerated Cholesky decompositions in `fields` are performed in-place while zeroing out the lower triangular portion of the matrix. The accelerated versions of `mKrig` use in-place Cholesky decompositions.



(a)



(b)

Figure 4: Average computation time for spatial analysis of the `CO2` dataset in `fields` for default (black) and accelerated workflows (green and blue) (a) and speedup over the default implementation (b). Spatial analysis includes 11 parameter samples for an exponential covariance function each including a Cholesky decomposition, generating a prediction of the spatial process, and estimating predictive error. The accelerated workflows use in-place Cholesky decompositions.

5 Discussion and Conclusions

Our results indicate the accelerated functions are considerably faster than the default, serial functions. The accelerated Cholesky decomposition reached a maximum speedup of 54.4 times over the range of observations tested for 40,000 observations with two GPUs using in-place calculations. While the simple spatial analysis workflow we timed was not accelerated as much as a percentage of the total unaccelerated execution time, we reduced its average execution time from 53 minutes and 10 seconds for just over 23,000 observations to 12 minutes 32 seconds and 12 minutes 26 seconds for 1 and 2 GPU accelerated workflows respectively corresponding to an 4.2 and 4.3 times speedup in each case. Although the Cholesky decomposition is by far the largest computational step in `mKrig` and the workflow, it is clear that other unaccelerated portions of the calculations take enough time to reduce the speedup of `mKrig` and the spatial analysis workflow. In spite of this, the reduction in computation time due to the accelerated Cholesky decomposition is significant. Speedup increases linearly over the range of observations tested as communication overhead becomes a smaller percentage of the computation time and the Cholesky decomposition increasingly dominates the spatial workflow execution time.

Using more than 1 GPU had little effect on the `mKrig` and spatial workflow computation times. For instance, the average workflow execution time when accelerated with 1 GPU was not more than 1 second different from the workflow accelerated with 2 GPUs until just over 23,000 spatial observations were used in the workflow. At that point, the 2 GPU workflow was only faster by 6 seconds out of approximately 12 and a half minutes. This can be explained by the unaccelerated subroutines used in accelerated `fields` taking a significant portion of the total computation time. Over the range of observations timed, the average in-place Cholesky decomposition computation times when using 1 GPU were different by at most a tenth of a second from the corresponding 2 GPU decomposition times. Out of the total accelerated workflow time, which was 12 minutes 32 seconds and 12 minutes 26 for 1 and 2 GPU accelerated workflow respectively, these fractions of a second, did not substantially change the accelerated workflow execution time on a percentage basis. In fact, for all numbers of observations tested the workflows were not different by more than 1 percent. Note that accelerated version of `mKrig` and the accelerated workflows use in-place decompositions rather than those involving a deep copy.

Part of the reason why there is little difference between using 1 and 2 GPUs in `mKrig` and the spatial workflow may also be due to additional memory usage in `mKrig` and the workflow compared to the Cholesky decomposition. This will be explored further in a future work (Ramakrishnaiah et al. 2015). Ramakrishnaiah et al. (2015) shows that MAGMA's CPU-GPU communication is serial in `magma_dpotrf_m`, the MAGMA Cholesky decomposition function we call when using 2 GPUs. The 2 GPU decompositions therefore do not use the full memory bandwidth between the CPUs and GPUs. As discussed in Section 4, data dependencies in the Cholesky decomposition causes the 2 GPU decomposition to require additional communication, which causes slowdowns. The communication speed decreases as the amount of memory available on the CPU decreases (Ramakrishnaiah et al. 2015), so the additional memory used by `mKrig` and the workflow compared to the decomposition itself creates overheads. The 2 GPU decomposition is especially affected due to the extra communications required by data dependencies.

Ultimately, using 2 GPUs has the advantage of being able to factor matrices using twice as much GPU memory (equating to about 40 percent more observations over a single GPU) and with twice the potential processing speed. While costs of communication between CPUs and GPUs are significant, they grow slower than computation costs, which grow as $\mathcal{O}(n^3)$ for n observations (Paciorek et al. 2013). Thus, asymptotically, communication costs become less significant compared to the calculations performed to compute the Cholesky factorization. When problem sizes increase to a point where limited memory on a single GPU becomes more of an issue, using 2 GPUs becomes more helpful. Thus, as problem sizes increase the 2 GPU decomposition becomes faster relative to the 1 GPU decomposition. This is evident in Figure 2 by the dip in performance of the single GPU Cholesky decomposition speedups from 26,000 to 33,000 observations. Since the 6GB memory on 1 Caldera GPU can hold at most a matrix of size just over 28k observations, additional communication overhead is incurred for the single GPU case when the number of observations increases above 28k. The extra GPU memory available when using 2 GPUs allows matrices of just under 40k $\times$ 40k to be stored in entirety on GPU memory. The final speedups in Figure 2 at 40,000 observations do not fit entirely on GPU memory, however, even with 2 GPUs. This may be the cause of the dip in the in-place 2 GPU decomposition speedup at 40,000 observations in Figure 2. In spite of this dip, the 2 GPU Cholesky decompositions are still substantially faster than the corresponding 1 GPU decompositions for 40,000 observations.

The main subroutines in `mKrig` and the spatial workflow aside from the Cholesky decomposition are those involved in computing the distance and covariance matrices, especially the covariance matrix. It is important to note that in this analysis we used the exponential covariance function, which is much easier to compute than certain other covariances such as the Matérn due to the Matérn's required computation of a modified Bessel function. We are currently exploring methods to accelerate the distance and covariance calculations by only evaluating the upper triangle of these matrices and saving the distance and covariance matrices between likelihood evaluations rather than recalculating it for each likelihood computation as is done currently. Also, the full distance and covariance matrices need not be computed to perform the Cholesky decomposition since `chol` and MAGMA only require the upper or lower triangle of the decomposed matrix.

The parameter optimization algorithm could also be restructured to use the `optim` function on the full, multidimensional parameter space. Currently `mKrig.MLE` (accelerated and non-accelerated versions) performs a basic grid search over the covariance parameters and calls `optim` to determine a maximum likelihood estimate for λ for each set of covariance parameters. Changing the distance and covariance matrix calculations and the parameter optimization algorithm could halve the computation time or better for the MAGMA-accelerated `mKrig` function and the spatial workflow.

It is also important to note that although MAGMA is very useful for shared memory systems, it does not yet fully support computations using distributed memory. Libraries such as DPLASMA, PaRSEC, StarPU-MPI, have all shown promise in distributed memory settings with heterogeneous architectures (Bosilca et al. 2011, Bosilca et al. 2013, Augonnet et al. 2014). Incorporating libraries such as these into `R` could be very useful for computations on multiple supercomputer nodes and on clusters, especially when dealing with problems too large to fit in memory on a single computer or computational node. As problem sizes increase there will be a need to use distributed systems to be able to store the entire problem on memory and speed up calculations. Making `R` packages with DPLASMA, PaRSEC, and StarPU-MPI functionality is therefore very important for spatial statistics as well many other subjects of study.

While speedups on Caldera computational nodes are considerable, it may be possible to achieve even higher speedups on personal computers that only have `R` installed with default settings from the CRAN distribution. `R` was built on the Yellowstone computing environment with specialized, accelerated computational libraries and compilation options many `R` users would not have access to or would not know how to use, and R would not be as fast when built with default settings. In addition, the Caldera GPUs have very high double precision performance in comparison to single precision performance. It may be worthwhile to use MAGMA's single precision Cholesky decomposition functions to further improve speedups, especially on less state-of-the-art GPUs that do not have such high double precision performance when compared to single precision. Assuming the accuracy of the calculated Kriging surface would not be significantly reduced, incorporating MAGMA's single-precision Cholesky calculations into `fields` would be warranted. We will release another technical report soon exploring single precision timing and accuracy on a mid-2014 MacBook Pro.

In order to make the installation process of our accelerated code easier, we are currently developing an R package with our accelerated functions as well as some modifications allowing the distance matrix to be reused across likelihood calculations. The package will be called `fieldsMAGMA` and we plan to release it soon.

References

(2014). URL `http://www.green500.org/news/green500-list-november-2014`.

(2014). URL `http://www.top500.org/lists/2014/11/`.

(2015a), "Resources overview." URL `https://www2.cisl.ucar.edu/resources`.

(2015), "What is cuda?" URL `http://www.nvidia.com/object/cuda_home_new.html`.

(2015b), "Yellowstone: Computational information systems laboratory." URL `https://www2.cisl.ucar.edu/resources/yellowstone`.

Agullo, Emmanuel, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Stanimire Tomov (2009), "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects." In *Journal of Physics: Conference Series*, volume 180, 012037, IOP Publishing.

Augonnet, Cédric, Olivier Aumage, Nathalie Furmento, Raymond Namyst, and Samuel Thibault (2014), "StarPU-MPI: Task programming over clusters of machines enhanced with accelerators." *Hal*, URL `https://hal.inria.fr/hal-00992208/PDF/RR-8538.pdf`.

Bosilca, George, Aurelien Bouteiller, Anthony Danalis, Mathieu Faverge, Azzam Haidar, Thomas Herault, Jakub Kurzak, Julien Langou, Pierre Lemarinier, Hatem Ltaief, et al. (2011), "Flexible development of dense linear algebra algorithms on massively parallel architectures with DPLASMA." In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, 1432–1441, IEEE.

Bosilca, George, Aurelien Bouteiller, Anthony Danalis, Mathieu Faverge, Thomas Hérault, and Jack J Dongarra (2013), "PaRSEC: Exploiting heterogeneity to enhance scalability." *Computing in Science &amp; Engineering*, 15, 36–45.

Broquedis, François, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst (2010), "hwloc: A generic framework for managing hardware affinities in hpc applications." In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, 180–186, IEEE.

Chen, Wei-Chen, George Ostrouchov, Drew Schmidt, Pragneshkumar Patel, Hao Yu, Christian Heckendorf, Brian Ripley, and R Core team (2014a), *pbdMPI: Programming with Big Data - Interface to MPI*. Version 0.2-5.

Chen, Wei-Chen, Drew Schmidt, George Ostrouchov, Pragneshkumar Patel, and Brian Ripley (2014b), *pbdSLAP: Programming with Big Data - Scalable Linear Algebra Package*. Version 0.2-0.

Chen, Wei-Chen, Drew Schmidt, Gaurav Sehrawat, Pragneshkumar Patel, and George Ostrouchov (2014c), *pbdPROF: Programming with Big Data - MPI Profiling Tools*. Version 0.2-3.

Dongarra, Jack, Yike Guo, Peter Nash, Vendel Szeremi, and Giovanni Montanna (2012a), *High Performance Linear Algebra in R*. HiPLAR, URL `http://www.hiplar.org/index.html`.

Dongarra, Jack, Yike Guo, Peter Nash, Vendel Szeremi, and Giovanni Montanna (2012b), *HiPLARb 0.1.0 Quick Start Guide*. HiPLAR, URL `http://www.hiplar.org/downloads/HiPLARb_0.1.0_quickstart-B.pdf`.

Katzfuss, Matthias and Noel Cressie (2012), "Bayesian hierarchical spatio-temporal smoothing for very large datasets." *Environmetrics*, 23, 94–107.

Lee, Jonathan (2013), *STOCHASTIC SIMULATION AND SPATIAL STATISTICS OF LARGE DATASETS USING PARALLEL COMPUTING*. Ph.D. thesis, The University of Western Ontario London.

Nash, Peter and Vendel Szeremi (2015), *Package 'HiPLARM'*. HiPLAR, URL `http://cran.r-project.org/web/packages/HiPLARM/HiPLARM.pdf`.

Nychka, Douglas, Reinhard Furrer, and Stephan Sain (2015), *Package 'fields'*. URL `http://cran.r-project.org/web/packages/fields/fields.pdf`.

Ostrouchov, G., W.-C. Chen, D. Schmidt, and P. Patel (2012), "Programming with big data in r." URL `http://r-pbd.org/`.

Paciorek, Christopher J, Benjamin Lipshitz, Wei Zhuo, Cari G Kaufman, Rollin C Thomas, et al. (2013), "Parallelizing gaussian process calculations in r." *arXiv preprint arXiv:1305.4886*.

Ramakrishnaiah, Vinay, Raghu Raj Prasanna Kumar, John Paige, and Dorit Hammerling (2015), "Accelerating 'fields' - spatial statistics package for r." Technical report, National Center for Atmospheric Research.

Schmidt, Drew, Wei-Chen Chen, George Ostrouchov, and Pragneshkumar Patel (2013a), *pbdBASE: Programming with Big Data - Base Wrappers for Distributed Matrices*. Version 0.2-3.

Schmidt, Drew, Wei-Chen Chen, George Ostrouchov, Pragneshkumar Patel, and R Core team (2012), *Guide to the pbdDMAT Package: Version 2.0*. pbdR, URL `http://cran.r-project.org/web/packages/pbdDMAT/vignettes/pbdDMAT-guide.pdf`.

Schmidt, Drew, Wei-Chen Chen, George Ostrouchov, Pragneshkumar Patel, and R Core team (2013b), *pbdDMAT: Programming with Big Data - Distributed Matrix Methods*. Version 0.2-3.

Schmidt, Drew, Wei-Chen Chen, George Ostrouchov, Pragneshkumar Patel, and R Core team (2015), *Package 'pbdDMAT'*. pbdR, URL `http://cran.r-project.org/web/packages/pbdDMAT/pbdDMAT.pdf`.

Smith, Brian J (2015), *Package 'magma'*. URL `http://cran.r-project.org/web/packages/magma/magma.pdf`.

Tomov, S, J Dongarra, V Volkov, and J Demmel (2009), "MAGMA library." *Univ. of Tennessee and Univ. of California, Knoxville, TN, and Berkeley, CA*.

Yu, Hao (2014), *Rmpi: Interface (Wrapper) to MPI (Message-Passing Interface)*. URL `http://cran.r-project.org/web/packages/Rmpi/Rmpi.pdf`. R package version 0.6-5.

Acknowledgements

This work was partially completed while Paige and Lyngaas were interns in Summer Internships in Parallel Computational Sciences (SIParCS). SIParCS is a program at the National Center for Atmospheric Research (NCAR), which is sponsored by the national science foundation. We would like to thank Srinath Vadlamani for his numerous contributions in the testing of several linear algebra libraries in the early stages of this project. We would also like to thank the Consulting Services Group at NCAR, especially Siddhartha Ghosh who answered several of our technical questions on using `R` in the Yellowstone supercomputing environment. In addition, Tim Hoar, an Associate Scientist at NCAR, assisted us with installing and linking MAGMA with `R`.

6 Appendix

6.1 `CO2` Dataset Example Code

The following code illustrates how to use the accelerated code for `fields` and compare its computation time to non-accelerated `fields` using 9,943 observations. It uses the `workflow` function, which performs all of the spatial calculations and is given in detail later.

```
#The following script computes how long mKrig takes for
   approximately 10000 observations
```

```
#using the CO2 dataset from fields.  Accelerated using 1 GPU.
   Uses approximate chordal distances.

#load accelerated fields
library(fields)
thisDir = "~/git/fieldsmagma_all/fieldsMAGMA/examples"
source(paste0(thisDir, "/../fieldsMAGMA.r"))
thisDir = "~/git/fieldsmagma_all/fieldsMAGMA/examples"

#now time the workflow for Krig or mKrig using CO2 dataset from
    fields:
data(CO2)

#filter dataset by lon/lat coords using "lim": only use a
   subset of
#the data that is close enough to(lon=0,lat=0)
lim <- 105
ind <- (-lim < CO2$lon.lat[,1]) & (CO2$lon.lat[,1] < lim)
ind <- (ind & -lim/2 < CO2$lon.lat[,2]) & (CO2$lon.lat[,2] <
   lim/2)
x = CO2$lon.lat[ind,]
y <- CO2$y[ind]
n <- length(y)

#record timings (note that there may be some GPU startup cost
   in the accelerated timing):
print(paste0('Using ', n, ' data points with lim: ', lim))
time = system.time(workflow(x, y, 0))[3]
print('finished non-accelerated workflow, beginning accelerated
    workflow')
accelTime = system.time(workflow(x, y, 1))[3]

print(paste0('non-accelerated time: ', time))
print(paste0('accelerated time: ', accelTime))

#plot prediction surface
png(height=5, width=7, units="in", res=300, file="
   CO2_prediction.png")
surface(out.p, xlab="Longitude", ylab='Latitude', main="CO2
   Concentration")
world(add=TRUE)
points(x, pch=20, cex=.1)
dev.off()

#plot estimated error surface
png(height=5, width=7, units="in", res=300, file="
   CO2_SE_est.png")
```

```
surSE<- apply( out.sim$Ensemble, 1, sd )
print(paste0("out.sim names: ", names(out.sim)))
image.plot(as.surface(out.sim$predictionGrid, surSE), xlab='
   Longitude', ylab='Latitude', main=sprintf('Kriging Estimated
    Standard Error for Lambda = %.1f, Theta = %.1f',
   lambda.MLE, theta.MLE))
world(add=TRUE)
points(x, pch=20, cex=.1)
dev.off()

#Smooth estimated error surface using thin plate spline
png(height=5, width=7, units='in', res=300, file="
   CO2_SE_smooth.png")
smoothSE = Tps(out.sim$predictionGrid, surSE)
surface(smoothSE, xlab='Longitude', ylab='Latitude', main=
   sprintf('Smoothed, Estimated Standard Error for Lambda = %
   .1f, Theta = %.0f', lambda.MLE, theta.MLE))
world(add=TRUE)
points(x, pch=20, cex=.1)
dev.off()

#save data:
save(list=c("n", "time", "accelTime", "lim"), file=paste0(
   thisDir, "/mKrigWorkflowComparison1_n", n, ".RData"))
```

This will produce plots of the results as well as save the timing data generated. The `workflow` function used in the above code is as follows:

```
#this function performs spatial analysis for given lon/lat
   coordinates (lonlat) and observations (y)
workflow = function(x, y, nGPU) {
  #start with spatial parameter guess of theta=8
  theta = 8

  print('optimizing over lambda')
  #calculate likelihood for different lambdas using mKrig.MLE
     and these lambda:
  lambda = 10^seq(-5, 0, length=10)
  out = mKrigMAGMA.MLE(x, y, theta=theta, lambda=lambda,
     cov.args= list(Covariance="Exponential", Distance="rdist")
     , lambda.profile=FALSE, nGPUs=nGPU)
  lnLikes = out$summary[,2]
  theta.MLE <<- theta
  lambda.MLE <<- out$lambda.MLE

  #use spline interpolator to find max likelihood:
```

```
  interpGrid = 10^(seq(-5, 0, length=150))
  interpLnLikes = splint(lambda, lnLikes, interpGrid)
  index = which.max(interpLnLikes)

  #compute MLE krig:
  out.mle <- mKrigMAGMA(x, y, theta=theta.MLE, lambda=
     lambda.MLE, cov.args= list(Covariance="Exponential",
     Distance="rdist"), nGPUs=nGPU)

  #Now predict surfaces and exact errors using Krig and mKrig
     objects:
  print('computing Kriging surfaces...')
  out.p <<- predictSurface(out.mle)

  #compute error:
  print('approximating Kriging error')
  minLat = min(x[,2])
  maxLat = max(x[,2])
  size = maxLat - minLat
  gridExpansion = max(c(1 + 1e-07, 10*theta/size))

  #make sure grid is at least 10 times as big as theta to avoid
      error
  set.seed(1)
  tmp <- try(sim.mKrig.approx(out.mle, M=5, gridExpansion=
     gridExpansion))
  if(class(tmp) == 'try-error') {
    print('Used extra large gridExpansion')
    out.sim <<- sim.mKrig.approx(out.mle, M=5, gridExpansion=2*
       gridExpansion)
    print('Finished using extra large grid expansion')
  } else {
    out.sim <<- tmp
  }

  invisible(NULL)
}
```

All the above code is available in `fieldsmagma_all/fieldsMAGMA/examples/mKrigWorkflowComparison1.r`, and requires X11 forwarding for the plotting.