Leveraging GPU libraries for efficient computation of Gaussian process models in *R*

Colin Rundel

Duke University, Department of Statistical Science

July 10, 2013

Background

Using intrinsic markers (genetic and isotopic signals) for the purpose of inferring migratory connectivity.

- Existing methods are too coarse for most applications
- Large amounts of data are available (>150,000 feather samples from >500 species)
- Genetic assignment methods are based on Wasser, et al. (2004)
- Isotopic assignment methods are based on Wunder, et al. (2005)

Background

Using intrinsic markers (genetic and isotopic signals) for the purpose of inferring migratory connectivity.

- Existing methods are too coarse for most applications
- Large amounts of data are available (>150,000 feather samples from >500 species)
- Genetic assignment methods are based on Wasser, et al. (2004)
- Isotopic assignment methods are based on Wunder, et al. (2005)

Paper - Rundel, C.W., *et al.* (2013) Novel statistical methods for integrating genetic and stable isotopic data to infer individual-level migratory connectivity. *Molecular Ecology*. (*in press*)

Data - DNA microsatellites and $\delta^2\mathrm{H}$

Hermit Thrush (*Catharus guttatus*)

- 138 individuals
- 14 locations
- 6 loci
- 9-27 alleles / locus

Wilson's Warbler (*Wilsonia pusilla*)

- 163 individuals
- 8 locations
- 9 loci
- 15-31 alleles / locus

Allele Frequency Model

For the allele i, from locus l, at location k

$$\boldsymbol{y_{l\cdot k}} \sim \text{Multinomial}\left(n_{lk},\ \boldsymbol{f}_{l\cdot k}\right) \qquad n_{lk} = \sum_i y_{lik}$$

$$f_{lik} = \frac{\exp(\Theta_{lik})}{\sum_i \exp(\Theta_{lik})} \qquad \boldsymbol{\Theta}_{li} \sim \mathcal{N}(\boldsymbol{M}_{li}, \boldsymbol{\Sigma})$$

Allele Frequency Model

For the allele i, from locus l, at location k

$$\boldsymbol{y_{l\cdot k}} \sim \text{Multinomial}\left(n_{lk},\ \boldsymbol{f}_{l\cdot k}\right) \quad n_{lk} = \textstyle\sum_i y_{lik}$$

$$f_{lik} = \frac{\exp(\Theta_{lik})}{\sum_i \exp(\Theta_{lik})} \quad \boldsymbol{\Theta}_{li} \sim \mathcal{N}(\boldsymbol{M}_{li}, \boldsymbol{\Sigma})$$

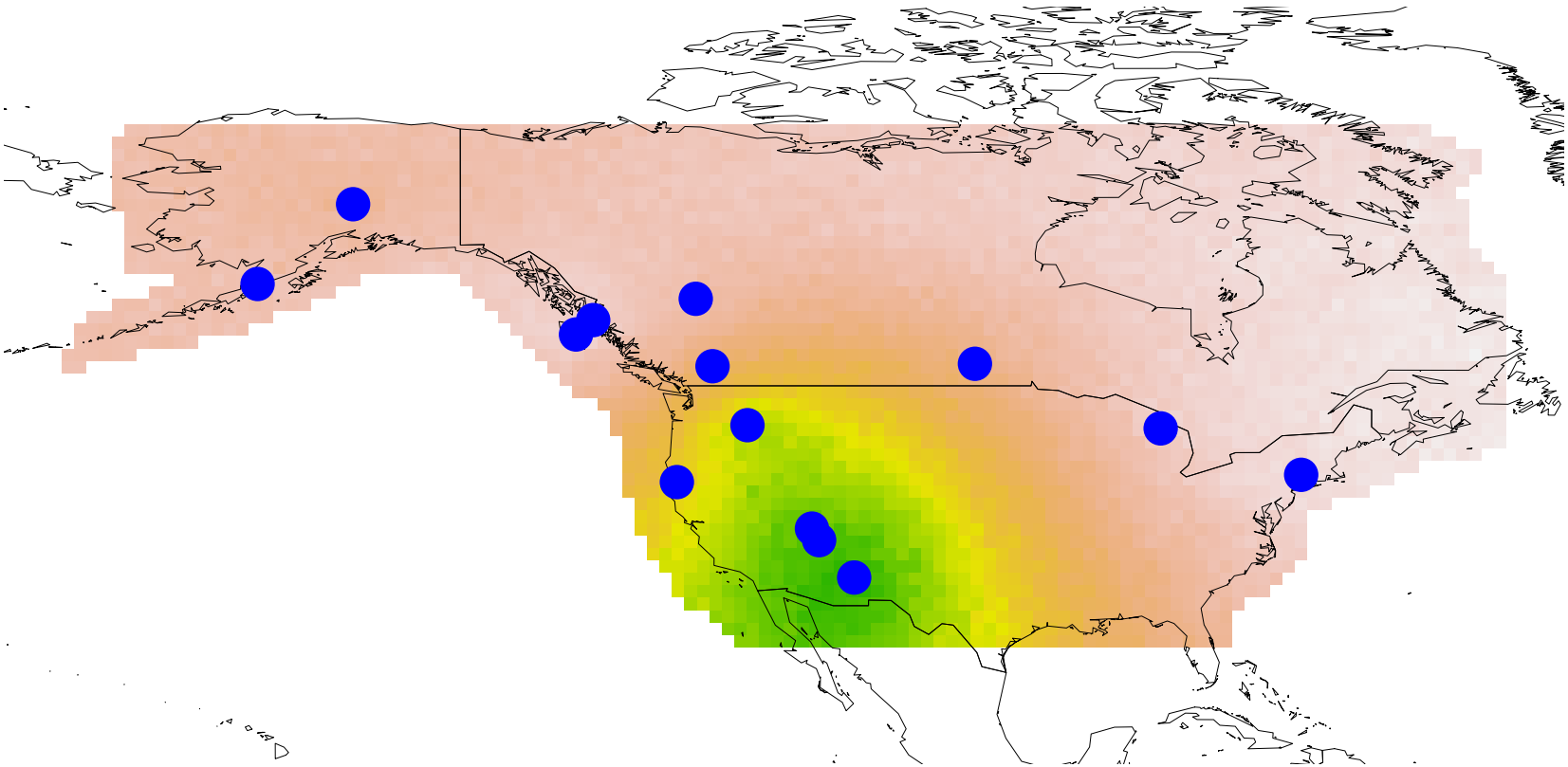
Posterior:

$$\prod_l \prod_k \frac{n_{lk}!}{\prod_i y_{lik}!} \prod_i \left(\frac{\exp(\Theta_{lik})}{\sum_i \exp(\Theta_{lik})} \right)^{y_{lik}}$$

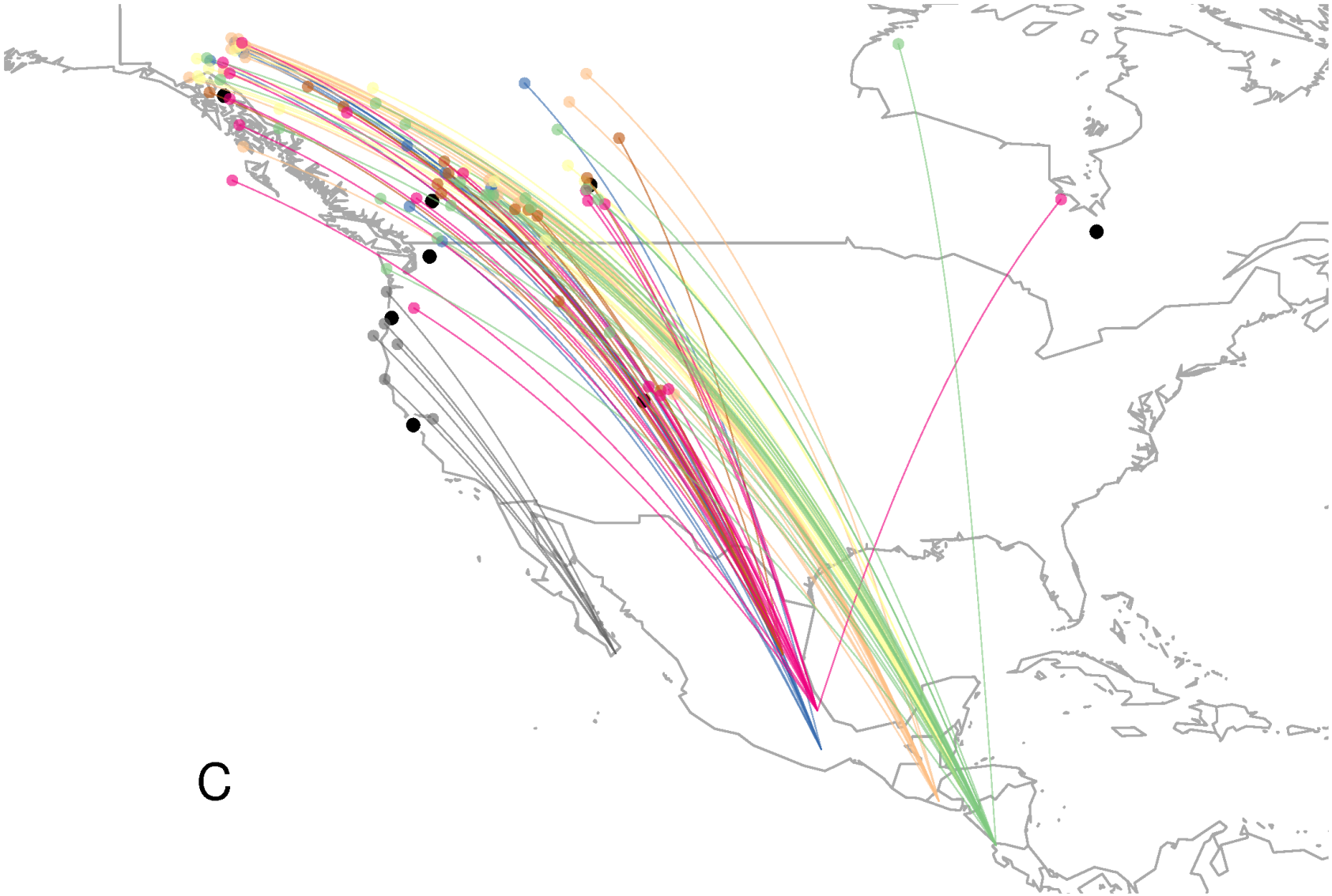
$$\times \prod_l \prod_i 2\pi^{-r/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left[-\frac{1}{2} (\boldsymbol{\Theta}_{li} - \boldsymbol{M}_{li})' \boldsymbol{\Sigma}^{-1} (\boldsymbol{\Theta}_{li} - \boldsymbol{M}_{li}) \right]$$

$$\times \pi(\boldsymbol{M}, \boldsymbol{\Sigma})$$

Model Output



Model Output



Implementation

Model fitting and prediction is done via MCMC

- Original implementation in pure C++ with minimal dependencies (Wasser, et al. (2004))
- Rewritten using R / C++ via Rcpp(Armadillo)
  - Code closer to matrix notation (and R)
  - Transparent use of high performance LAPACK implementations (ATLAS, OpenBLAS, Intel MKL)
- GPU based optimizations were added using CUDA, CUBLAS, and MAGMA libraries
- R package isoscatR (available on Github now, CRAN soon)

Performance

System specs - 4 core Intel i5-2500K, GeForce GTX 460

Software specs - Ubuntu 13.04, OpenBlas 0.2.6, CUDA 5.5 RC, Magma 1.4 beta1

Performance

System specs - 4 core Intel i5-2500K, GeForce GTX 460

Software specs - Ubuntu 13.04, OpenBlas 0.2.6, CUDA 5.5 RC, Magma 1.4 beta1

Performance during model fitting is quite good ...

300,000 iterations in $\sim$ 5.5 minutes

Performance

System specs - 4 core Intel i5-2500K, GeForce GTX 460
Software specs - Ubuntu 13.04, OpenBlas 0.2.6, CUDA 5.5 RC, Magma 1.4 beta1

Performance during model fitting is quite good ...

300,000 iterations in $\sim$ 5.5 minutes

Performance during prediction is much slower ...

1,000 prediction iterations in $\sim$ 30 mins
(predictions calculated every 100 iterations)

Performance

System specs - 4 core Intel i5-2500K, GeForce GTX 460
Software specs - Ubuntu 13.04, OpenBlas 0.2.6, CUDA 5.5 RC, Magma 1.4 beta1

Performance during model fitting is quite good ...

$$\text{300,000 iterations in} \sim \text{5.5 minutes}$$

Performance during prediction is much slower ...

$$\text{1,000 prediction iterations in} \sim \text{30 mins}$$
$$\text{(predictions calculated every 100 iterations)}$$

Not too bad in the greater scheme of things, but we need to perform cross validation ($150 - 200$ runs per species) ...

Prediction details

Why is the prediction slow?

Prediction details

Why is the prediction slow? Predicting allele frequencies for Hermit thrush at 3318 novel locations.

To do so we sample from:

$$\boldsymbol{\Theta}_p|\boldsymbol{\Theta}_m \sim \mathcal{N}(\boldsymbol{\mu}_p + \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}(\boldsymbol{\Theta}_m - \boldsymbol{\mu}_m),\ \boldsymbol{\Sigma}_p - \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}\boldsymbol{\Sigma}_{mp})$$

Prediction details

Why is the prediction slow? Predicting allele frequencies for Hermit thrush at 3318 novel locations.

To do so we sample from:

$$\boldsymbol{\Theta}_p|\boldsymbol{\Theta}_m \sim \mathcal{N}(\boldsymbol{\mu}_p + \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}(\boldsymbol{\Theta}_m - \boldsymbol{\mu}_m),\ \boldsymbol{\Sigma}_p - \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}\boldsymbol{\Sigma}_{mp})$$

Algorithm steps

1. Calculate $\boldsymbol{\Sigma}_{pm}$ and $\boldsymbol{\Sigma}_p$
2. Calculate Chol$(\boldsymbol{\Sigma}_p - \boldsymbol{\Sigma}_{pm}\boldsymbol{\Sigma}_m^{-1}\boldsymbol{\Sigma}_{mp})$
3. Sample from MVN
4. Calculate allele frequencies
5. Output results

Prediction algorithm timings

	Step	CPU (secs)	CPU+GPU (secs)	Rel. Improvement
1.	Covariances	1.080		
2.	Cholesky	0.467		
3.	Sample MVN	0.049		
4.	Allele Freq	0.129		
5.	Write	0.007		
	Total	1.732		

Prediction algorithm timings

	Step	CPU (secs)	CPU+GPU (secs)	Rel. Improvement
1.	Covariances	1.080	0.046	23
2.	Cholesky	0.467	0.208	2.3
3.	Sample MVN	0.049	0.052	0.9
4.	Allele Freq	0.129	0.127	1.0
5.	Write	0.007	0.032	0.2
	Total	1.732	0.465	3.7

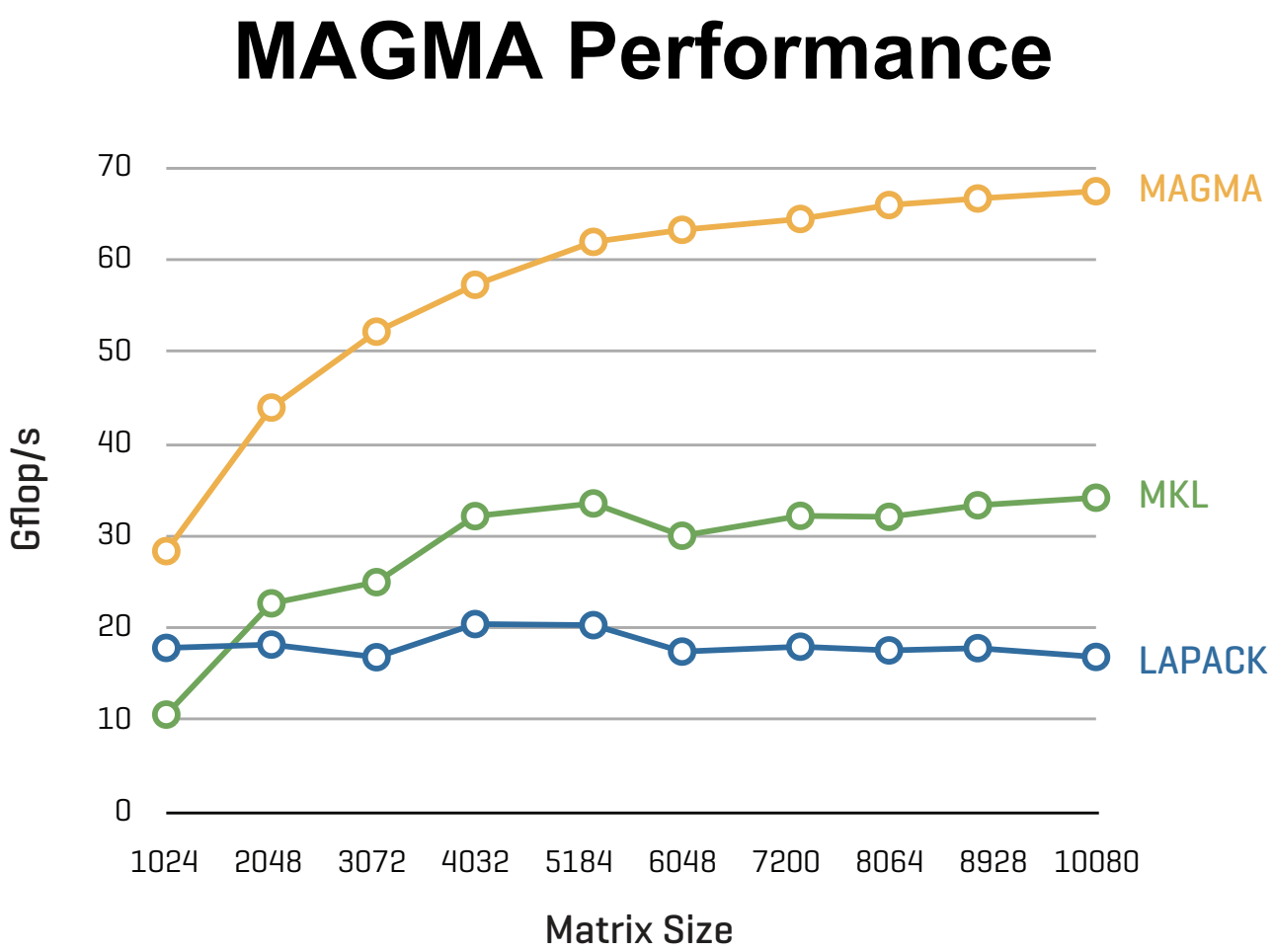
Improving the Cholesky step

Not surprising given Cholesky factorization is $\mathcal{O}(n^3)$ and $n = 3318$.

There isn't a magical solution to this, so we just want to use the fastest possible implementation of the Cholesky decomposition.

- **Intel MKL / ATLAS / Eigen** - all (multicore) CPU based with very marginal improvement
- **CUBLAS** - part of NVidia's CUDA toolkit, implements core BLAS functions (but not Cholesky)
- **CULA** - proprietary / closed source (dense and sparse) GPU linear algebra library with an expensive license
- **MAGMA** - open source Multicore+GPU dense hybrid linear algebra library (CUDA and OpenCL implementations)

Cholesky GPU



Ltaief, H. "A Scalable High Performant Cholesky Factorization for Multicore with GPU Accelerators"

VECTAR'10 Presentation, Berkeley, CA, June 22-25, 2010.

Additional Considerations

- There are costs for moving data on to and off of the GPU
- Once the data is there, may as well do as many calculations as possible
  - Drawing sample from the GP is sped up by performing the matrix multiplication on the GPU
- GPU code is much more verbose / dense (calculating $\Sigma_{pm}\Sigma_m^{-1}$)

Additional Considerations

- There are costs for moving data on to and off of the GPU
- Once the data is there, may as well do as many calculations as possible
  - Drawing sample from the GP is sped up by performing the matrix multiplication on the GPU
- GPU code is much more verbose / dense (calculating $\Sigma_{pm}\Sigma_m^{-1}$)

Armadillo

```
arma::mat tmp = cov12.t() * p.Sinv
```

GPU (CUBLAS)

```
cublasDgemm_v2(
    p.handle, CUBLAS_OP_T, CUBLAS_OP_N,
    n_pred, n_known, n_known,
   &one,
    p.d_cov12, n_known,
    p.d_invcov11, n_known,
   &zero,
    p.d_tmp, n_pred
)
```

Improving Covariance calculations

Covariance in our model is assumed to be stationary and isotropic (depends only on distance between locations)

- Elements of the covariance matrix can be calculated independently
- Small scale "embarrassingly parallel" $\Rightarrow$ good candidate for the GPU.
- Implementation is straight forward (if we ignore things like symmetry)

Improving Covariance calculations

Covariance in our model is assumed to be stationary and isotropic (depends only on distance between locations)

- Elements of the covariance matrix can be calculated independently
- Small scale "embarrassingly parallel" $\Rightarrow$ good candidate for the GPU.
- Implementation is straight forward (if we ignore things like symmetry)

```
__global__ void powered_exponential_kernel(double* dist, double* cov,
                                           const int n, const int nm,
                                           const double sigma2, const double phi,
                                           const double kappa, const double nugget)
{
    int n_threads = gridDim.x * blockDim.x;
    int pos = blockDim.x * blockIdx.x + threadIdx.x;

    for (int i = pos; i < nm; i += n_threads)
        cov[i] = sigma2 * exp(-pow(dist[i] / phi, kappa)) + nugget*(i%n == i/n);
}
```

Summary

Relatively small changes in one function resulted in $3 - 4$x improvement

- Cross validation results in a day or two and not a week
- Started with trying to find an improved Cholesky decomposition, other optimizations followed
- GPU implementation was relatively painless
- Libraries are under active development (read: things can and will break)
- External libraries make package development more complicated

What's next?

Much of these details are specific to this particular problem, but the approach suggests common tasks/bottlenecks that turn up when working with Gaussian Processes.

What's next?

Much of these details are specific to this particular problem, but the approach suggests common tasks/bottlenecks that turn up when working with Gaussian Processes.

To that end ...

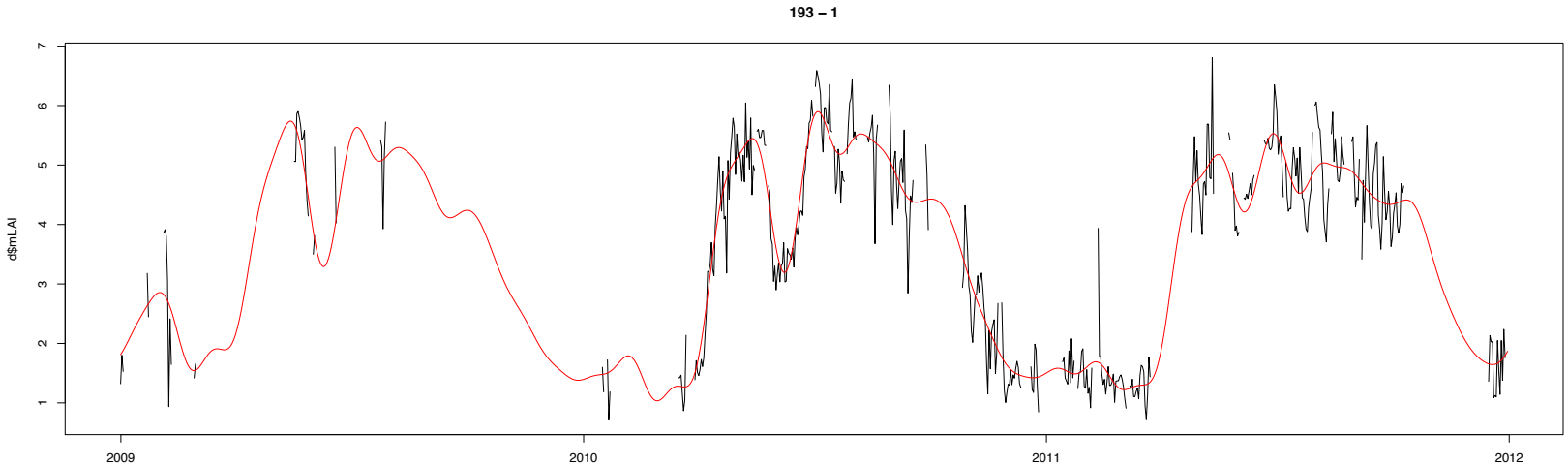
- Later this summer - hopefully releasing RcppGP package
  - Talk at JSM 2013 in Montreal
- Low level CPU and GPU R and C++ utility functions
- Methodology agnostic
- Support for composite covariance functions

What's next?

Much of these details are specific to this particular problem, but the approach suggests common tasks/bottlenecks that turn up when working with Gaussian Processes.

To that end ...

- Later this summer - hopefully releasing RcppGP package
  - Talk at JSM 2013 in Montreal
- Low level CPU and GPU R and C++ utility functions
- Methodology agnostic
- Support for composite covariance functions



Questions, Comments?

email : rundel@gmail.com

github : *http://github.com/rundel/*

presentation : *http://github.com/rundel/Presentations/*