Parallelizing Gaussian Process Calculations in R

Christopher Paciorek    UC Berkeley Statistics

Joint work with:

Benjamin Lipshitz    UC Berkeley EECS (formerly)
Wei Zhuo    IBM
Prabhat    Lawrence Berkeley National Laboratory
Cari Kaufman    UC Berkeley Statistics (formerly)
Rollin Thomas    Lawrence Berkeley National Laboratory

www.jstatsoft.org/v63/i10
https://github.com/paciorek/bigGP

JSM, August 2015

Funded by DOE Office of Advanced Scientific Computing Research (AC02-05CH22131) and supported by NERSC
A basic Gaussian process (GP) model

\[
\begin{align*}
Y|g, \theta & \sim N(g, C_y(\theta)) \\
g|\theta & \sim N(\mu(\theta), C_g(\theta))
\end{align*}
\]

\[
f(y) \propto \left| C_y(\theta) + C_g(\theta) \right|^{-\frac{1}{2}} \cdot \exp \left( -\frac{1}{2} (y - \mu(\theta))^T (C_y(\theta) + C_g(\theta))^{-1} (y - \mu(\theta)) \right)
\]

Computational goals for GP calculations:
• Likelihood optimization
• Prediction
• Prediction uncertainty
• Simulation (unconditional and conditional on data)

Computational patterns for GP calculations:
• Construct covariance matrices (training and prediction points)
• Cholesky decomposition
• Forward/backsolve
• Matrix multiplication (various forms: matrices, vectors, crossproducts, diagonal matrices, etc.)
Responses to computational considerations

Change the model
• Sparsify the covariance/precision matrix
• Reduce dimensionality (e.g., basis functions)
• Predict based on subset of data / local models
• Approximate the likelihood

Use a lot of processing power
• Shared memory (multicore architecture)
• GPUs
• Distributed memory (clusters, supercomputers)

Our approach
• Use many cores to distribute computation and memory
• Hybrid parallelization = distributed processing (MPI) + threaded computation (OpenMP)
• Original plan to use ScaLapack from R (pbdR now does this)
• bigGP: tailor parallel linear algebra algorithms to Cholesky decomposition (rate-limiting step) and interface to R
Blockwise Cholesky (Crout’s algorithm)

Covariance Factorization

Forwardsolve

bigGP: Parallelizing Gaussian Process Calculations in R
Computational considerations

Questions
• How many submatrices / how big
• How many submatrices (and which ones) per node or per process
• How many processes per multi-core node / should individual submatrix calculations be threaded

Basic tradeoffs
• Efficient local computation (increase block size up to point that submatrix doesn’t fit in cache)
• Load-balancing (want all processors active)
• Reduce communication (pass less information between processes)
Computational approach

Suppose we have P=10 processes and D=4 blocks. We could have one submatrix per process (left with blocking factor B=4 and h=1) or multiple submatrices per process (right with replication factor h=3 and blocking factor of B=hD=12)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

bigGP: Parallelizing Gaussian Process Calculations in R
Computational approach

Answers:

• How many blocks (submatrices) / how big:
  Your choice but 1000x1000 may be good; submatrix size is ~ n/hD where P=D(D+1)/2

• How many submatrices (and which ones) per node or per process:
  Your choice of h. We allocate submatrices to processes for you.

• How many processes per node / should individual block calculations be threaded:
  One process per node with threading, but might define a “node” as a subset of processors on cluster machine – e.g. divide 24 cores into 4 6-core “nodes”

Comments

• Our allocation of which submatrices are grouped on a process improves load-balancing
• Using multiple submatrices per process and one process per “node” reduces communication
• Flexible choice of #submatrices per process allows tailoring of submatrix size to cache
• Flexible number of cores per process (cores on a virtual node) allows hybrid parallelization that balances threading vs. additional virtual nodes
Scaling results

![Graph showing scaling results for Cholesky decomposition with varying matrix dimensions and number of cores.](image)

- **Execution time, seconds (log scale)**
- **Number of cores (log scale)**

**Cholesky decomposition, n=32768**

- Optimal h
- ScaLAPACK

**Matrix dimension, n (log scale)**

- 2048
- 8192
- 32768
- 131072

**Execution time, seconds (log scale)**

- 0.01
- 0.1
- 1
- 10
- 100
- 1000
- 10000

**Number of cores (log scale)**

- 2048
- 8192
- 32768
- 131072

**Number of cores per process**

- 1 core per process
- 6 cores per process
- 24 cores per process
Software design of the bigGP package

- Initialization (from R) initializes multiple processes and sets up how submatrices are assigned to processes
- All computations done in a distributed fashion and sequential operations (e.g., Cholesky, then forwardsolve then matrix multiplication) carried out as pipeline
- Core distributed linear algebra operations (Cholesky, forwardsolve, matrix multiplication, etc.) are done via MPI from C
- bigGP API controls operations from R
  - Functions for managing objects on worker processes and moving general objects between master and workers
  - Functions for distributing and collecting distributed matrices/vectors between master and workers, hiding details of what is stored where and in what format
  - Wrapper functions that carry out the distributed linear algebra by calling the core C/MPI code
- Specific kriging (likelihood optimization) implementation via krigProblem
  ReferenceClass and member functions that carry out:
  - Construction of mean and covariance (in distributed fashion) using user-defined functions
  - Calculation of log density
  - Prediction with uncertainty
  - Simulation either conditional or unconditional on data
Astrophysics example

- Analysis by C. Kaufman in collaboration with R. Thomas
- Data are flux from the Type Ia supernova SN2011fe, as a function of wavelength and day
  - $n = 67,275$
- Interest lies in smoothing the data and estimating the wavelength of minimum flux
Statistical model and code

\[ Y_i = Z(t_i, w_i) + \alpha_{t_i} + \epsilon_i \]
\[ Z \sim GP(\mu(\cdot; \kappa, \lambda), \sigma^2 K(\cdot, \cdot; \rho_{\epsilon})K(\cdot, \cdot; \rho_{\omega})) \]

- Random effects, alpha, for each day
- Known, heteroscedastic error variances based on instrumentation
- Covariance of product form in the two dimensions
- Mean function (of time only) based on empirical pattern of variation with wavelength

- 465 processes, 6 cores per process, 117 nodes on NERSC’s Hopper supercomputer
- R code:

```r
R> prob$optimizeLogDens(method = 'L-BFGS-B', lower = rep(1e-15, nParams))
R> pred <- prob$predict(ret = TRUE, se.fit = TRUE)
R> realiz <- prob$simulateRealizations(r = 1000, post = TRUE)
```

bigGP: Parallelizing Gaussian Process Calculations in R
Results

- Predictions on subdomain on a grid in time and wavelength for 55,379 points
- Numerical issues arise in bigGP (no pivoting) for finer grids (numerically non-positive def.)
- Credible intervals based on 1000 posterior draws, using MLE parameter estimates

For methodological work on inferring minima, see Lee, Kaufman and Thomas:
Next steps

• Will try to continue to support bigGP at basic level in spare time
• Current kriging code allows flexible mean and covariance specification, but for a basic model
• Low-level API allows flexible use of linear algebra calls, so others could build more general and flexible models using the API
• pbdR is another option and more general