Brazilian Journal of Probability and Statistics
2017, Vol. 31, No. 4, 686–691
https://doi.org/10.1214/17-BJPS365B
Main article: https://doi.org/10.1214/17-BJPS365

Comment: A brief survey of the current state of play for Bayesian computation in data science at big-data scale

David Draper[a] **and Alexander Terenin**[b]

[a]*University of California, Santa Cruz*
[b]*Imperial College London*

We wish to contribute to the discussion of this interesting paper by offering our views on the current best methods for Bayesian computation, both at Big-Data scale and with smaller data sets, as summarized in Table 1. This table is certainly an over-simplification of a highly complicated area of research in constant (present and likely future) flux, but we believe that constructing summaries of this type is worthwhile despite their drawbacks, if only to facilitate further discussion.

The entries in the table are tailored to problems—such as classification and regression—in which the data set is organized into n rows (representing subjects of study, which may or may not be organized hierarchically, for example, patients nested in hospitals) and p columns of variables measured on the subjects, but the table may be relevant for other data structures as well (for example, data sets in *topic modeling* (e.g., Blei, 2012) may be visualized as consisting of n words (nested inside documents) and p topics). The rows of Table 1 play out the four combinations of big and small n and p, and the columns identify the four major hardware configurations currently available: single CPU, single GPU, cluster of CPUs, and cluster of GPUs. A more detailed table would have broken both n and p out into (small, medium, large), and further detail could have been supplied in a more model-specific fashion, but in our view Table 1 already makes some interesting comparisons.

For us, "Big Data" refers to settings in which the available data set is too big to store and process in memory on a single machine, or *worker* (consisting of one single-threaded CPU, some RAM, and some disk storage). In our table this corresponds to big n, which in today's environment typically equates to a data set that takes tens of gigabytes (or more) to store. For our purposes, big p is in the hundreds of thousands, tiny p is a few hundred, and small p is in between.

Our comments on the individual cells in the table are as follows.

- (n, p) = (Small, Small) is the current Bayesian computational comfort zone: an example would be logistic regression using a regularization prior (e.g., the horseshoe (Carvalho, Polson and Scott, 2010)) with an n of a few thousand and p on the order of 100.

Received June 2017; accepted June 2017.

Table 1 *Our view of the current best methods for simulation-based Bayesian computation, as a function of number of observations* (n)*, number of predictors* (p)*, and available hardware. Big n specifies data sets that take tens of gigabytes* (*or more*) *to store*; *tiny p is a few hundred*; *big p is hundreds of thousands*

n	p	Single CPU	Single GPU	CPU Cluster	GPU Cluster
Small	Small	Standard MCMC	GPU-accelerated particle filters	Independent parallel chains	not needed
Small	Big	Hamiltonian Monte Carlo (HMC) or Gibbs sampling[(1)]	HMC or GPU-accelerated Gibbs sampling	Asynchronous Gibbs sampling[(2)]	No current Bayesian method
Big	Small	Continuous-time MCMC with subsampling and control variates[(3)]	Continuous-time MCMC or GPU-accelerated Gibbs sampling[(4)]	Asynchronous Gibbs sampling[(5)] or methods based on sharding	No current Bayesian method
Big	Big	Hopeless: point estimates only	Model-specific[(6)]	Model-specific[(7)]	No current Bayesian method

[(1)] HMC if gradient is available, else Gibbs.

[(2)] The exact algorithm if Metropolis–Hastings correction is cheap enough, otherwise the approximate algorithm if diagnostic indicates appropriate.

[(3)] If gradient is available and p is tiny.

[(4)] ZigZag (see text) or BPS with parallel gradient evaluations if p is tiny; GPU-accelerated Gibbs if gradient unavailable and/or p is medium-size.

[(5)] If data augmentation is possible with the model under study.

[(6)] May be possible for gradient-based methods in some models (current research).

[(7)] An example is topic modeling, using AD–LDA (approximate) or Pólya Urn LDA (essentially exact with big n).

- Standard MCMC on a single CPU is fine here, with home-grown code or using an environment such as `WinBUGS` (Lunn et al., 2000) or `RJAGS` (Plummer, 2003).
- If you have a single GPU and need answers extremely quickly, GPU-accelerated particle filters (Lee et al., 2010) are a good option with some models.
- If instead you have access to a cluster of workers, you can of course readily run independent parallel chains on each worker and merge the results. This includes computing on a single machine with multiple cores/threads, for example via `doParallel` or `snow` in `R`.
- A cluster of GPUs would be overkill in this situation.

- $(n, p) =$ (Small, Big) is already at the research frontier with some hardware configurations.
  - Using a single worker, you can do Hamiltonian Monte Carlo (HMC, Betancourt, 2017) if the gradient of the log posterior is available (e.g., with non-discrete likelihoods), writing the code yourself or fitting your model in `Stan` (Stan Development Team, 2016), or you can use Gibbs sampling in settings with or without a gradient.
  - With a single GPU, HMC is again a good option with a gradient (Beam, Ghosh and Doyle, 2016), or you can use GPU-accelerated Gibbs sampling (Terenin, Dong and Draper, 2016). The latter method can be better in models with heavy-tailed error distributions, where HMC sometimes performs poorly, but HMC within Gibbs would be even better than pure Gibbs in such problems.
  - If you have access to a cluster of CPUs, one good method is asynchronous Gibbs sampling (Terenin, Simpson and Draper, 2017), using either (a) the exact algorithm, if the Metropolis-Hastings correction built into the method is cheap enough or (b) the (much faster) approximate algorithm, if the diagnostic described in the paper indicates that this is safe.
  - It would appear that no one knows how to make efficient fully Bayesian use of a GPU cluster at present: the problem is that GPU computation is massively faster than available network speed, so it's typically not at all clear how to keep all of the GPUs busy at once.
- $(n, p) =$ (Big, Small) is an active area of recent and current research.
  - On a single CPU, in our view the most promising approach with big n and tiny p, when a gradient is available, involves methods based on creating continuous-time stochastic processes without discretization that sample correctly from the posterior of interest—these include ScaLE (Pollock et al., 2016), ZigZag sampling (Bierkens, Fearnhead and Roberts, 2016), and the Bouncy Particle Sampler (BPS, Bouchard-Côté, Vollmer and Doucet, 2015). The key advantage of these methods is the ability to run while evaluating one data point at a time.
  - With a single GPU, two good options are (a) BPS (with parallel gradient evaluations) if p is tiny and (b) GPU-accelerated Gibbs sampling if the gradient is unavailable and/or if p is medium in size (up to 10,000, say).
  - A cluster of workers at your disposal lands you in the cell into which Steve Scott's present paper fits, under *divide-and-conquer* methods based on sharding—other methods that partition the data set into small subsets of subjects, make standard calculations on each small data set, and then combine at the end include a variety of techniques developed by David Dunson and his collaborators (e.g., Srivastava, Li and Dunson, 2015). Many of the sharding papers concentrate on examples in which p is tiny; Asynchronous Gibbs sampling is a good alternative with larger p when data augmentation is possible

in the model at hand. One of the issues that sharding-based methods face with larger p is ensuring convergence on each of the individual shards.
  - With a cluster of GPUs, as noted above, the profession is currently at a loss to make fully effective use of the hardware for posterior exploration (as opposed to optimization: see below).
- $(n, p) = (\text{Big}, \text{Big})$ is where current methods start to reveal their (serious) limitations.
  - Using only a single CPU, fully Bayesian posterior exploration in this case is hopeless. The best you can hope for is employing optimization-based methods to obtain the maximum a posteriori (MAP) estimate. As a large body of work in machine learning shows, this is sometimes adequate to solve the problem at hand.
  - With a single GPU, success becomes model-specific. It may be possible to adapt gradient-based methods to drive a GPU efficiently in some models—this is a topic of current research.
  - Employing a cluster of workers, with n and p both big, good results are currently also available only on a model-specific basis—topic models provide an example, with Approximate Distributed LDA (AD–LDA, Newman et al., 2009) offering approximate results and Pólya Urn LDA (Terenin et al., 2017) producing answers that are exact up to Monte Carlo noise with big n. If it's sufficient to obtain MAP estimates in the problem you're working on, by far the best Big-Data optimization algorithm at present is *stochastic gradient descent* (SGD; see Recht et al., 2011 for an asynchronous parallellized version of the method). One of the great advantages of SGD is that it can be coded in such a way that the data set can be *streamed* through a processor, without any need to store it—see Tran, Toulis and Airoldi (2016) for an `R` implementation for some models in the `CRAN` package `sgd`.

    It's worth noting that ScaLE, ZigZag, and BPS can all handle streaming data, producing accurate full-Bayes posterior exploration (i.e., not just MAP estimates) in the (n big, p small) region of problem space. This is in contrast to other methods such as *stochastic gradient Langevin dynamics* (SGLD, Welling and Teh, 2011), which explores the posterior so slowly that it might as well just be an optimization method. SGLD provides a vivid example of the meta-theorem that if you have a simulation-based method that has an asymptotic convergence proof but that mixes extremely poorly, the convergence proof may be useless as a guide to the practical effectiveness of the method.
  - And finally, once again, nobody knows how to make efficient fully Bayesian use of a cluster of GPUs in the $(n, p) = (\text{Big}, \text{Big})$ setting, where such a cluster would be most needed if it could be effectively utilized. If it's sufficient in your problem to settle for MAP estimates, see Barkalov, Gergel and Lebedev (2016) for one approach to computing them on a GPU cluster.

Three more concluding comments:

- At several points we said things like "if you have access to [hardware configuration X]," but we wish to emphasize that *everybody* currently has access to the hardware in all four columns of Table 1—including people in academia, not just in industry—by renting this hardware from any of a number of cloud-computing companies, typically for less than US$1 per hour (and universities can often persuade such companies to donate many hours in the cloud to them for free). If you're currently using only the first column of Table 1 as your hardware resource and you want to begin exploring the Big-Data world, we would encourage you to venture into columns 2 and 3. If you do so, and you can handle the programming (see below), you'll find, as we have, that for many problems a single GPU is better than 100 CPUs.
- Having just issued an invitation to explore new hardware, it must be admitted that the availability of user-friendly software to make best use of GPUs and CPU clusters is currently nowhere near the level of ease of use of, for example, `CRAN` packages in `R`. One promising recent development to note, however, is that at least one hardware manufacturer has recently introduced an external GPU node that makes it possible to drive a GPU with `CUDA` programs residing on your laptop or desktop—this brings Bayes on GPUs one step further away from the research frontier and closer to day-to-day applications.
- One further note on streaming, GPUs, and the value of contemporary hardware: to work efficiently on a GPU, you either have to (1) hold your data set in the GPU's memory or (2) stream your data (if it's too big to fit in memory). Option (2) is how the machine-learning community has obtained its remarkable recent successes with the use of *deep learning* (e.g., Schmidhuber, 2015) via SGD on a GPU to solve the image classification problem (Krizhevsky, Sutskever and Hinton, 2012).

  This brings us to our final meta-theorem: statisticians run the serious risk of being marginalized in the field of data science, ceding the high ground to machine learning when, in our view, this field should involve an equal partnership between mathematical sciences (applied mathematics, statistics), computing sciences (machine learning, database organization and management, the hardware-software interface), and subject-matter expertise in the problem at hand.

We thank Steve for a stimulating paper, and we would be interested in his comments on how sharding methods in general, and his approach in particular, fit into the framework of our table, both currently and in the future.

References

Barkalov, K., Gergel, V. and Lebedev, I. (2016). Solving global optimization problems on a GPU cluster. *AIP Conference Proceedings* **1738**.

Beam, A. L., Ghosh, S. K. and Doyle, J. (2016). Fast Hamiltonian Monte Carlo using GPU computing. *Journal of Computational and Graphical Statistics* **25**, 536–548. MR3499693

Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. Available at arXiv:1701.02434.

Bierkens, J., Fearnhead, P. and Roberts, G. (2016). The zig-zag process and super-efficient sampling for Bayesian analysis of big data. Available at arXiv:1607.03188.

Blei, D. (2012). Probabilistic topic models. *Communications of the ACM* **55**, 77–84.

Bouchard-Côté, A., Vollmer, S. J. and Doucet, A. (2015). The bouncy particle sampler: A nonreversible rejection-free Markov chain Monte Carlo method. Available at arXiv:1510.02451.

Carvalho, C. M., Polson, N. G. and Scott, J. G. (2010). The horseshoe estimator for sparse signals. *Biometrika* **97**, 465–480. MR2650751

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1097–1105.

Lee, A., Yau, C., Giles, M. B., Doucet, A. and Holmes, C. C. (2010). On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *Journal of Computational and Graphical Statistics* **19**, 769–789.

Lunn, D. J., Thomas, A., Best, N. and Spiegelhalter, D. (2000). WinBUGS—A Bayesian modelling framework: Concepts, structure, and extensibility. *Statistics and Computing* **10**, 325–337.

Newman, D., Asuncion, A., Smyth, P. and Welling, M. (2009). Distributed algorithms for topic models. *Journal of Machine Learning Research* **10**, 1801–1828. MR2540777

Plummer, M. (2003). JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*.

Pollock, M., Fearnhead, P., Johansen, A. M. and Roberts, G. O. (2016). The scalable Langevin exact algorithm: Bayesian inference for big data. Available at arXiv:1609.03436.

Recht, B., Re, C., Wright, S. and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 693–701.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks* **61**, 85–117.

Srivastava, S., Li, C. and Dunson, D. B. (2015). Scalable Bayes via barycenter in Wasserstein space. Available at arXiv:1508.05880v2.

Stan Development Team (2016). Stan modeling language: Users' guide and reference manual, version 2.14.0. Available at http://mc-stan.org.

Terenin, A., Dong, S. and Draper, D. (2016). GPU-accelerated Gibbs sampling: A case study of the horseshoe probit model. Available at arXiv:1608.04329v2.

Terenin, A., Magnusson, M., Jonsson, L. and Draper, D. (2017). Pólya urn latent Dirichlet allocation: A sparse massively parallel sampler. Available at arXiv:1704.03581v1.

Terenin, A., Simpson, D. and Draper, D. (2017). Asynchronous Gibbs sampling. Available at arXiv:1509.08999v3.

Tran, D., Toulis, P. and Airoldi, E. (2016). Stochastic gradient descent methods for estimation with large data sets. Available at https://cran.r-project.org/web/packages/sgd/vignettes/sgd-jss.pdf.

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 681–688. MR3157685

Baskin School of Engineering
University of California
1156 High Street
Santa Cruz, California 95064
USA
E-mail: draper@ucsc.edu