Journal of Statistical Software

July 2016, Volume 71, Issue 4. *doi: 10.18637/jss.v071.i04*

Software Alchemy: Turning Complex Statistical Computations into Embarrassingly-Parallel Ones

Norman Matloff
University of California, Davis

Abstract

The growth in the use of computationally intensive statistical procedures, especially with big data, has necessitated the usage of parallel computation on diverse platforms such as multicore, GPUs, clusters and clouds. However, slowdown due to interprocess communication costs typically limits such methods to "embarrassingly parallel" (EP) algorithms, especially on non-shared memory platforms. This paper develops a broadly-applicable method for converting many non-EP algorithms into statistically equivalent EP ones. The method is shown to yield excellent levels of speedup for a variety of statistical computations. It also overcomes certain problems of memory limitations.

Keywords: parallelization, embarrassingly parallel algorithms, big data, cloud computing, GPU computing, memory limitations, i.i.d. estimators, asymptototically normal estimators.

1. Introduction

Many modern statistical methods involve computationally intensive statistical algorithms, and are often applied to data sets with large numbers of records and/or variables. Indeed, in this era of big data, it is common to have millions of records, but even with tens of thousands of cases computation may present a real challenge.

This has necessitated the usage of parallel computational methods on diverse platforms such as multicore, graphics processing units (GPUs), clusters and clouds. Interest in such methods is keen. Consider the R statistical language, for instance (R Core Team 2016). The central list of parallel processing tools for R, given by the task view on "High-Performance and Parallel Computing with R" (Eddelbuettel 2016), includes dozens of general and application-specific packages. Many of these packages did not exist at the time a broad survey paper on the topic was published in 2009 (Schmidberger, Morgan, Eddelbuettel, Yu, Tierney, and Mansmann 2009), so the growth in interest is remarkable.

However, slowdown due to interprocess communication costs (network delays, memory consistency actions and so on) often limits such methods to "embarrassingly parallel" (EP) algorithms. Technically, the term EP applies to algorithms that are so easy to parallelize that it is "embarrassing," in the sense of presenting no challenge to the programmer. However, the term as typically used imposes the additional condition that there is very little communication overhead among the processes. This is key in statistical contexts, in which algorithms are often not EP.

This paper provides a solution to the problem, applicable generally to statistical estimators based on i.i.d. samples. In essence, it converts a non-EP to an EP algorithm of the same statistical accuracy. The method itself is easy to explain. It begins with a traditional method of parallel processing, in which one partitions the data into chunks, applies an algorithm to each chunk, and then somehow combines the results of the chunks. That word *somehow* is the central issue in traditional approaches; it is in this results-combining stage that problems often arise, as the combining procedure typically involves slow, non-EP computation.

Consider *mergesort*, a method for sorting an array of numbers. Here the array is broken into chunks, and each process sorts one of the chunks (using a nonparallel algorithm). The sorted chunks are then merged, and it is there that the slowdown occurs: The merging process is not EP. With this example in mind, we might take as working definition of EP problems of *map-reduce* form (Lee, Choi, Chung, Lee, and Moon 2011), provided that the reduce portion involves little or no computation.

What makes the method developed here different is that it exploits *statistical* properties. The idea is to combine the chunks simply by averaging them, an operation involving essentially no non-EP work, and then recognize the statistical properties of this averaging. This last point will form the crux of this paper.

We will refer to this chunking and averaging approach as CA. Clearly, a core question must be answered: CA would be of little value unless it were verified that the proposed estimator has the same statistical accuracy as the original, nonchunked one. (We will refer to the latter as the *full* estimator, FE.) If the CA method were to not produce a consistent estimator, or if it were to produce larger standard errors than those of the full estimator, the speedups would not hold much value.

Thus the key to CA is showing that it produces the same asymptotic standard errors as FE. This property is indeed verified here, in Section 3, for asymptotically normally distributed FEs. In other words, if the FE is asymptotically normal, then CA has the same asymptotic standard errors as FE. Since most widely-used estimators in statistics are in this category, the CA method is broadly applicable.

CA is inherently EP, and thus achieves software alchemy, turning non-EP algorithms into EP ones. In other words, *CA provides a fairly general method for parallelizing statistical computation.* Another use of CA will be to circumvent memory limitations. Today's very large data sets can exceed common memory size constraints. Such constraints can be either physical (insufficient RAM and swap space), or software based, such as the R language's former maximum $2^{31}-1$ byte size for any single object. As of this writing, R is in a transition process under which *long* vectors can now have size up to $2^{53}-1$, but matrix row/column indices are still limited to 32-bit range. And even with further broadening of indexing capabilities, some big data applications' memory requirements may exceed the physical and virtual memory of one's machine. CA can remedy that problem.

2. Related work and contribution of this paper

The present work presents CA as a tool for performing *parallel computation* in general *statistical analyses, i.e., estimation and inference.* However, the CA idea has arisen in various previous contexts. In the following an overview of previous work and the contribution of the present paper is given.

Chunking was proposed for speedup purposes in Hegland, McIntosh, and Turlach (1999). Here only a specific estimator was considered, a certain additive nonparametric regression model.

Chunked estimation was then investigated in another special case, that of linear regression, in Fan, Lin, and Cheng (2007), motivated by memory requirement problems rather than speed. The authors allowed their chunk averaging to be weighted, and they derived the optimal weights. They also showed that certain statistics would be t-distributed under the assumption of normally-distributed populations, and studied the resulting chunked estimators via simulation. However, that work was restricted to linear regression, not general types of estimators, and was not aimed at parallel processing, i.e., speedup.

The question of parallel computation specifically for statistical quantities was addressed in the context of database hardware and software in Cohen, Dolan, Dunlap, Hellerstein, and Welton (2008). There the emphasis was on parallelization of sums computation, which could then be applied to some statistical operations involving sums.

A group led by W. Cleveland (Guha, Hafen, Kidwell, and Cleveland 2009) developed a variant of CA, essentially what will be called "C without the A" in Section 7. Here the emphasis was primarily on visualization.

The present author's work on CA as "software alchemy" began in early 2010 (Matloff 2010) as an effort to avoid the overhead of task queues in shared-memory computing. His emphasis was mainly that seen in the present work, i.e., statistical estimation and inference, asymptotics, time complexity analysis and so on.

During that time, the present author also became aware of work then in progress by the Cleveland group in statistical contexts, Guha (2010), which eventually appeared in Guha, Hafen, Xia, Rounds, Li, Xi, and Cleveland (2012). The group has continued the project in a largely data visualization and exploration context, and has developed an extensive, Hadoop-based open source package for it, **Tessera** (Cleveland 2016).

Kleiner, Talwalkar, Sarkar, and Jordan (2012), focusing on statistical estimation and inference, developed a bootstrap approach related to CA, based on applying the bootstrap method to a number of small random subsamples of the original data. Since any bootstrap method requires the user to choose the values of "hyperparameters" – size and numbers of subsamples – the authors proposed an adaptive method to choose these parameters. They derived some asymptotic results, but also noted that their method will fail in settings in which the ordinary bootstrap fails. On the other hand, they indicated how their method can be used for time series data, something other CA methods have not yet been extended to.

Though the names are similar, CA has little relation to *model averaging* (Claeskens and Hjort 2008). The latter method, and the related technique of bagging (Breiman 1996), are not intended as a mechanism for parallel computation.

The contributions of this paper are as follows:

- It is verified that CA works, i.e., is fully statistically efficient, in fairly general settings.
- Prescriptions are given for computing standard errors and estimating covariance matrices.
- A rough characterization is given of circumstances for which CA produces a speedup.
- It is shown that CA can yield *superlinear* speedup, to a degree rare in the parallel processing world.
- It is shown that CA can bring a speedup even in the nonparallel case.
- Timing experiments are presented for a variety of statistical methods.

3. Statistical properties of CA

Suppose we have i.i.d. data $V_i, \; i = 1, \ldots, n$ from some distribution F_V. Let θ denote some population value of interest, that is, θ is some function of F_V. The full estimator FE of θ based on d observations is some function b_d of those values. So, the FE based on the first n observations will be denoted by

$$\widehat{\theta} = b_n(V_1, \ldots, V_n). \tag{1}$$

Note that V_i, θ and so on are possibly vector-valued.

Partition the data into r chunks, where r is the number of parallel processes; r might be the number of cores in a multicore machine or the number of nodes in a cluster, for example. Assuming for now that r evenly divides n, the chunk size is $k = n/r$.[1] Write the jth observation in chunk m as W_{mj}, $m = 1, \ldots, r; j = 1, \ldots, k$.

Denote the estimator of θ on chunk m, i.e., FE applied to that chunk, by $\widetilde{\theta}_{mr}$ (retaining the r for clarity below). Then we have

$$\widetilde{\theta}_{mr} = b_k(W_{m1}, \ldots, W_{mk}). \tag{2}$$

Now define the CA estimator:

$$\overline{\theta} = \frac{1}{r} \sum_{m=1}^{r} \widetilde{\theta}_{mr}. \tag{3}$$

The key result will be that $\overline{\theta}$ is asymptotically normal, and most important, that it has the same asymptotic covariance matrix as $\widehat{\theta}$. In other words, the statistical accuracy of $\overline{\theta}$ is just as good as that of $\widehat{\theta}$. This is intuitively clear, but a careful derivation follows.

It is assumed that the V_i are independent and identically distributed. For notational convenience, the analysis here will treat the case of scalar θ. The vector-valued case follows the same derivation path.

It is assumed that $\widehat{\theta}$ has an asymptotic normal distribution, i.e., that there exists some σ for which

$$\lim_{n \to \infty} \mathsf{P}\left[\frac{\sqrt{n}(\widehat{\theta} - \theta)}{\sigma} \leq t \right] = \Phi(t) \tag{4}$$

[1] It would be natural to take the first chunk to consist of $V_1, \ldots, V_k$, the second one as $V_{k+1}, \ldots, V_{2k}$, and so on. Since the original data are assumed i.i.d., any non-data dependent assignment of chunks would work. However, note the comments in Section 8.

for all t, where Φ is the cdf of the $\mathrm{N}(0,1)$ distribution. It will be shown below that $\overline{\theta}$ does just as well as $\widehat{\theta}$, in the sense that for fixed r, $\sqrt{rk}(\overline{\theta}-\theta)/\sigma$ converges in distribution to $\mathrm{N}(0,1)$, as k goes to infinity.[2]

Let g_k and h_{rk} be the characteristic functions of $\sqrt{k}(\widetilde{\theta}_{mr}-\theta)/\sigma$ and $\sqrt{rk}(\overline{\theta}-\theta)/\sigma$, respectively; the former is the characteristic function for the standardized estimator based on one chunk, while the latter is the corresponding function for $\overline{\theta}$ as a whole. Then

$$\begin{aligned} h_{rk}(t) &= \mathsf{E}\left[e^{it\sqrt{rk}(\overline{\theta}-\theta)/\sigma}\right] && (5)\\ &= \mathsf{E}\left[e^{it\sqrt{rk}\cdot\frac{1}{r}\sum_{m=1}^{r}(\widetilde{\theta}_{mr}-\theta)/\sigma}\right] && (6)\\ &= \prod_{m=1}^{r}\mathsf{E}\left[e^{it\sqrt{k/r}(\widetilde{\theta}_{mr}-\theta)/\sigma}\right] && (7)\\ &= \prod_{m=1}^{r} g_k(t\sqrt{1/r}) && (8)\\ &= \left[g_k(t\sqrt{1/r})\right]^r. && (9)\end{aligned}$$

Under our normality assumption for $\widehat{\theta}$, which applies to the $\widetilde{\theta}_{mr}$ as well since they are "mini-versions" of $\widehat{\theta}$, we have that

$$\lim_{k\to\infty} g_k(t) = e^{-0.5t^2}, \tag{10}$$

which is the characteristic function of the standard normal distribution. Thus for fixed r,

$$\lim_{k\to\infty} h_{rk}(t) = [e^{-0.5(t/\sqrt{r})^2}]^r = e^{-0.5t^2}. \tag{11}$$

Thus $\overline{\theta}$ is asymptotically normal, and its asymptotic variance σ^2 does match that of $\widehat{\theta}$, as promised.

Since CA will be employed almost exclusively on large data sets anyway (small ones would not need the speed CA provides), the asymptotic results should hold well in practice. This is confirmed in the empirical results presented in Section 6 below.

4. Refinements

One can obtain standard errors for $\overline{\theta}$ empirically (similar to bootstrapping). The estimated covariance matrix of $\overline{\theta}$ is

$$\frac{1}{r}\sum_{m=1}^{r}(\widetilde{\theta}_{mr}-\overline{\theta})(\widetilde{\theta}_{mr}-\overline{\theta})^\top. \tag{12}$$

One can then obtain standard errors for the components of $\overline{\theta}$ (and for linear combinations of them) in the usual manner.

Often n will not be an exact multiple of r. In such cases, let $k=\lfloor n/r\rfloor$, and $k'=n-(r-1)k$. For $i=1,\ldots,r-1$, let chunk i consist of k observations as before, but define the rth chunk to consist of the last k' observations.

[2] It is clear that $k\to\infty$ is generally also a necessary condition, as otherwise $\overline{\theta}$ would fail to be even a consistent estimator of θ in many applications. The results here describe the situation in which larger and larger applications are run on the same r-core machine, r-node cluster, etc.

The weightings in Equation 3 will no longer be $1/r$. To determine the proper weightings, note first that due to symmetry, the weights for the first $r-1$ estimators will be the same, with a common value we will call λ. The weight for the rth estimator will then be $1-(r-1)\lambda$.

To determine λ, again think of the case of scalar θ for convenience. The asymptotic variance of the overall weighted estimator $\overline{\theta}$ will be proportional to

$$\lambda^2(r-1) \cdot \frac{1}{k} + [1-(r-1)\lambda]^2 \cdot \frac{1}{k'}. \tag{13}$$

Minimizing this with respect to λ, we find

$$\lambda = \frac{1}{r-1+\frac{k'}{k}}. \tag{14}$$

The definition of the CA estimator then becomes a modified version of Equation 3:

$$\overline{\theta} = \sum_{m=1}^{r-1} \lambda \widetilde{\theta}_{mk} + [1-(r-1)\lambda]\widetilde{\theta}_{rk}. \tag{15}$$

5. On what types of statistical methods will CA be faster?

The above findings indicate that CA will produce speedup in many types of statistical methods. But which types?

5.1. Algorithmic analysis

Let us consider speedup from an algorithmic time complexity point of view. As before, let n denote the number of observations in an i.i.d. sample, with r denoting the number of processes that work in parallel.

Consider statistical methods needing time $O(n^c)$. If we have r processes, then CA assigns n/r data points to each process. Speaking in rough, exploratory terms, CA would reduce the run time to $O((n/r)^c) = O(n^c/r^c)$ time under CA, a speedup of r^c. The larger the exponent c is, the greater the speedup.

Thus statistical applications with computational time complexity greater than or equal to $O(n)$ will be major beneficiaries of CA. Examples of such applications are quantile regression, estimating hazard functions with censored data, and so on.

The situation, though, is more subtle when there are two variables affecting run time, not just n. Consider, for example, linear regression with p predictor variables. The time needed is first $O(np^2)$ to form sums of squares and cross products, followed by $O(p^3)$ to derive the least-squares estimates from those sums. The latter, which is the time complexity for matrix inversion or equivalent operations,[3] is independent of n, so the benefit accruing from using CA may be more modest. Similar reasoning applies to principal components analysis. There the first phase consists of a computation of sums that benefits from CA, but the second phase involves eigenanalysis with time independent of n. In both of these examples, CA is helpful for the first stage of computation, so it is helpful overall, but often only to a modest degree.

[3] If QR factorization is used, the complexity may be $O(p^2)$, depending on exactly what is being computed.

5.2. Superlinear behavior

Remarkably, the CA method can enable performance increases on a *superlinear* scale. This term, from the parallel processing literature, refers to speedups of more than r from only r processes.

In the general parallel processing world, this phenomenon is quite rare. Moreover, when it does occur, it is small in size, and arises from ancillary cache effects and the like; see for instance Kosec and Depolli (2012). But in the statistical world, users of CA will find superlinear speedups to be commonplace and quite substantial, as follows.

As noted above, CA reduces the time complexity of an $O(n^c)$ problem to roughly $O(n^c/r^c)$ for a statistically equivalent problem, whereas a linear speedup would only reduce the time to $O(n^c/r)$. If $c > 1$, then the speedup obtained from CA is greater than linear in r, hence the term "superlinear."

Indeed, in the superlinear case, CA may yield a speedup even on a uniprocessor machine. Here is a back-of-the-envelope computation: The r chunks must now be computed serially rather than in parallel, and take time $r\ O(\frac{n^c}{r^c}) = O(\frac{n^c}{r^{c-1}})$. This suggests that for $c > 1$, CA may be faster than the FE even in uniprocessor settings. The same reasoning shows that it may pay to *oversubscribe* one's hardware, e.g., have more threads than cores on a multicore machine.

6. Empirical investigation

To illustrate the value of CA, the method was used on the following diverse set of statistical applications:

- Kendall's τ correlation ($O(n^2)$ and $O(n \log n)$ algorithms),
- quantile regression,
- hazard function estimation,
- log-concave density function estimation, and
- linear regression.

Simulation runs were conducted on an Intel Xeon 2.0 Gz 64-bit machine with 16 cores and a hyperthreading degree of 2. Thus as many as 32 threads may be in computation at once. Runs with 4, 8, 16, 24, 32 and 40 threads were conducted.

In most cases the problem size was chosen to be large enough for parallelization to be worthwhile. The criterion for the latter was arbitrarily set to having an FE run time of at least a few seconds.

The R language was used, using R's standard packages for each of the statistical methods listed above. Interprocess communication used R's **parallel** package, in the portion derived from the **snow** package (Tierney, Rossini, Li, and Sevcikova 2015).

Elapsed times were recorded in seconds. As will be seen, CA yielded speedups in all cases, but as predicted in Section 5, the magnitude of speedup varied widely. In some cases, the speedup

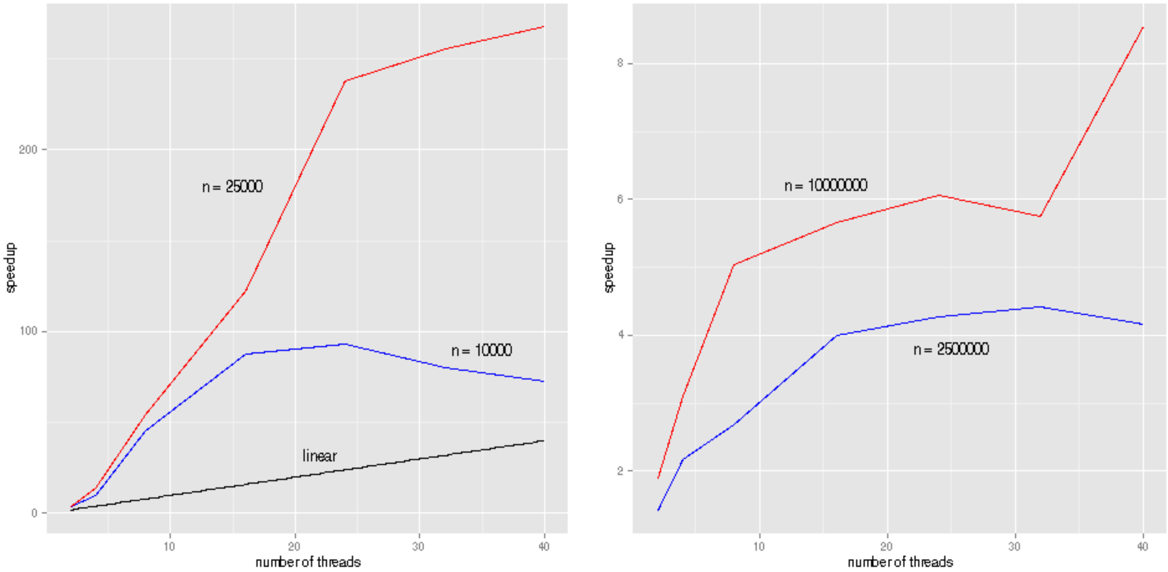

Figure 1: Kendall's τ (left); Kendall's τ, Knight's algorithm (right).

was superlinear, while in others it was much more modest. Where feasible, a 45-degree line was also plotted, to assess possible superlinearity.

In addition to measuring run times, the l_1 relative absolute differences between CA $(\overline{\theta})$ and the full estimator $(\widehat{\theta})$ were computed. For a p-component θ, this is

$$\frac{\sum_{i=1}^{p} |\overline{\theta}_i - \widehat{\theta}_i|}{\sum_{i=1}^{p} |\widehat{\theta}_i|}. \tag{16}$$

Values are reported to four decimal places. In all cases this relative difference was negligible, amplifying the point that the two estimators have the same asymptotic distribution.

6.1. Results

Kendall's τ

Here (X, Y) pairs were formed, as follows. Independent U$(0, 1)$ variates U_1 and U_2 were generated, and then (X, Y) was formed as

$$(X, Y) = (U_1, 0.2U_1 + U_2). \tag{17}$$

The R function `cor.test()` was used, with the argument `method = "kendall"`. The results are shown in Figure 1 on the left. Here the FE times were 2.98 and 17.13 seconds, and relative difference value ranges were 0.0010–0.0093 and 0.0000–0.0020, respectively for $n = 10000$ and 25000.

The "ordinary" Kendall algorithm has an $O(n^2)$ time complexity. Section 5 indicates that we should expect superlinear behavior, which is confirmed here, especially for the larger value of n. The timings here suggest similar performance for other methods of this type, such as any U-statistic.

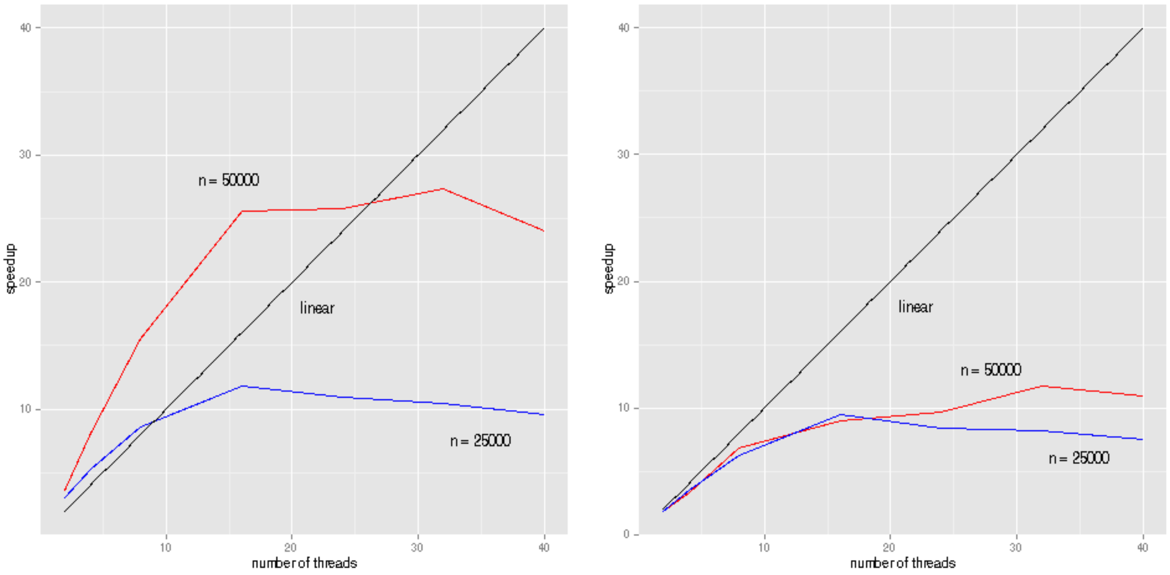

Figure 2: Quantile regression (left); Hazard function estimation (right).

But the clever algorithm due to Knight (Christensen 2005) has only $O(n \log n)$ time complexity. This is implemented in the function `cor.fk()` in the R package **pcaPP** (Filzmoser, Fritz, and Kalcher 2014) available from the Comprehensive R Archive Network (CRAN). CA is beneficial here too, though only in much larger problems and then modestly compared to the number of threads used; see Figure 1 on the right.

Here the FE times were 3.24 and 22.05 seconds, and relative difference values were 0, up to four decimal places, for both $n = 2500000$ and 10000000. No superlinear effect was observed at these values of n, but speedups of up to a factor of more than 8 occurred.

Quantile regression

Here p is the number of predictor variables, and each observation is of the form $(X_1, \ldots, X_p, Y)$, with the X_i being i.i.d. $\mathrm{U}(0,1)$ and with

$$Y = X_1 + \ldots + X_p + 0.2U, \tag{18}$$

where U had a $\mathrm{U}(0,1)$ distribution and was independent of the X_i. The function `rq()` from the package **quantreg** (Koenker 2016) was used, with $p = 75$. The results are displayed in Figure 2 on the left.

Here the FE times were 16.20 and 60.89 seconds, and relative difference value ranges were 0.0000–0.0010 and 0.0003–0.0009, respectively for $n = 25000$ and 50000. Strong speedups are obtained, sometimes superlinear.

Hazard function estimation

Here p is the proportion of censored observations. The data were sampled from $\mathrm{U}(0,1)$, and the parameter vector θ consisted of the values of the hazard function h(t) at $t = 0.2$, 0.4, 0.6, 0.8. The function `muhaz()` from the CRAN package of the same name (Hess and Gentleman 2014) was used, with the default settings. The results are shown in Figure 2 on the right.

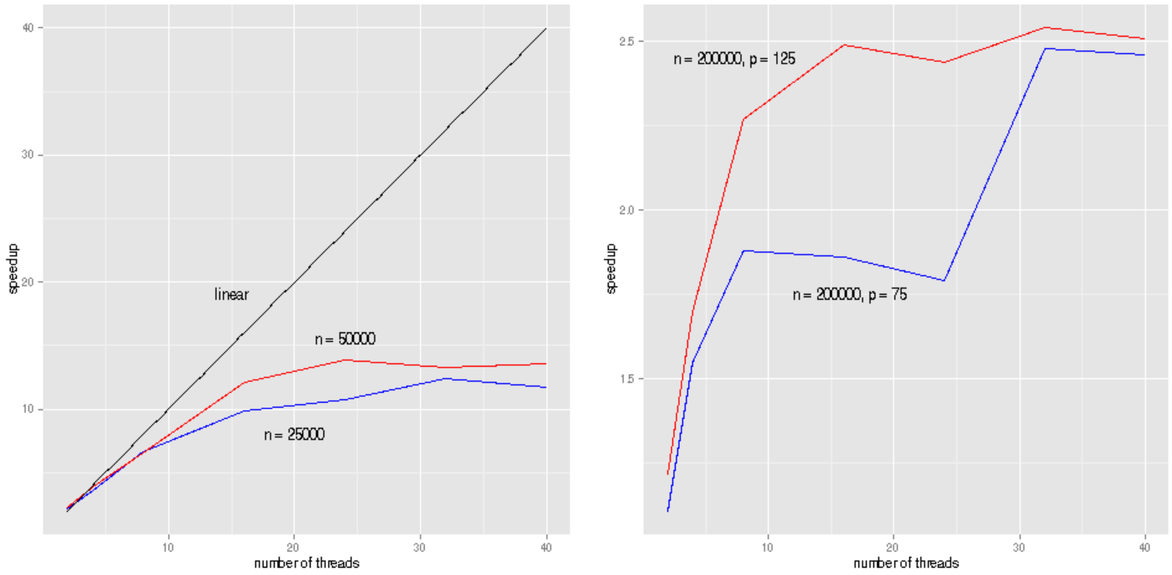

Figure 3: Log-concave density estimation (left); Linear regression (right).

Speedups here were well below linear (except for small numbers of threads), but were still strong, as large as 10 or more. FE times were 5.59 and 10.97 seconds, and relative difference value ranges were 0.0031–0.0106 and from 0.0002–0.0108, respectively for $n = 25000$ and 50000.

Log concave density estimation

This is a type of nonparametric density estimation (Dümbgen and Rufibach 2011). The data were generated from $\mathrm{N}(0, 1)$, and θ was taken to be the value of the density at 0. The function `logConDens()` from the CRAN package **logcondens** was used. See Figure 3 on the left for the results.

The pattern here is similar to that found for hazard function estimation, near-linear only for smaller number of threads. FE times were 28.90 and 67.92 seconds, and relative difference value ranges were 0.0031–0.0105 and 0, up to four decimal places, respectively for $n = 25000$ and 50000.

Linear regression

Here p is the number of predictor variables, and the distribution of the data was as in the quantile regression case above. The outcomes are shown in Figure 3 on the right. FE times were 5.22 and 11.06 seconds, and relative difference values were 0, up to four decimal places, for both $p = 75$ and 125. For the reasons explained in Section 5, prospects for speedup in the linear regression case are somewhat limited, but still CA yields approximately 50% speedup even with only 4 cores.

6.2. Discussion

In the above experiments, CA achieved good performance in a diverse set of statistical methodologies, with significant speedups even in the more modest cases. In addition, CA achieved this in a *convenient* manner: Off-the-shelf R packages were used on the chunks, and the CA method is quite easy to implement.

The CA method is based on asymptotics. As pointed out earlier, since the performance boost of parallel processing is generally needed only on large samples, the asymptotics should work well. This was confirmed in the simulations, which showed that the relative difference between CA and FE was negligible even for just moderately large data sets.

In some cases, speedup decreased slightly at the larger numbers of threads. Though this may be due to sampling variation, it should be noted that although each of the threads is doing the same amount of work,[4] there may be some variation from one thread to another; with more threads, the maximum interthread variation will increase, thus increasing overall run time.

If the basic estimator (FE) itself is unstable and of questionable use, CA may produce different results from FE. This may occur, for instance, in hazard function estimation when the proportion of censored observations is high. CA may actually have a stabilizing effect in such cases, say if CA is used with medians instead of means. This is a topic for future study, possibly in connection to the literature on model averaging.

7. “C without the A”

Guha *et al.* (2009) were interested in visualization of very large datasets. Their solution to the huge computation involved was to use a form of CA, which they termed *Divide and Recombine* (D&R). The ‘R’ here roughly corresponds to the ‘A’ in “CA” in the present paper. In that setting, though, there was no ‘A’ and even the ‘R’ meant only collecting the chunk results into a single database of graphs.

Actually, “C without the A,” is a very useful variant of CA. We will call it CWA, meaning that we divide into chunks and apply some estimator to each chunk, but then somehow use the resulting collection of estimators separately rather than averaging them.

The main example here will involve density estimation. Suppose we wish to compute a nonparametric density estimate at each observation in our sample. Instead of using the entire data set for this at any given point, we use only the data in the chunk to which the point belongs.

Why is CWA so computationally advantageous? The results of Section 5 provide an explanation. Consider a kernel-based density estimator, for instance. If we graph the estimated density at each data point, we need to do $O(n^2)$ work.

The significance of the results in Section 5 in the present context is as follows. The amount of computation needed is growing as $O(n^2)$, yet the amount of statistical value we get is growing only as $O(n)$, e.g., in variance of estimators. So, the extra work involved with doing density estimation on the full data set will not produce a commensurate improvement in statistical quality.

Moreover, in the case of graphics, in which small differences may not adversely affect our visual perception, the large n may not be very useful anyway. This consideration becomes even more important in light of the fact that the nonzero nature of pixel width on the screen

[4] If the number of processes evenly divides n.

makes the use of larger data sets meaningless at some point, due to overplotting and the "black screen effect." So CWA will not be harmful in large data, and will yield large savings in run time. In other words, CWA is a win.

The author also used CWA to great benefit in Matloff (2013), which presented several novel visualization techniques for large datasets. (Here "large" simply meant large enough for graphing of the full data to produce significant overplotting, which can occur even for moderately-large values of n.) For example, one of the methods proposed was a novel approach to the screen-clutter problem in plotting parallel coordinates. The author used k-nearest neighbor methods to estimate the multivariate density, as had been done with parallel coordinates previously, but with the new twist that only the lines with highest density are plotted. To compute the estimated density at the ith observation V_i, the author only used the data in chunk j, i.e., $V_{(j-1)k+1}, \ldots, V_{jk}$.

8. The case of non-i.i.d. data

The derivation in Section 3 assumed i.i.d. data. This of course is a standard condition in many statistical methods, and many packages both in base R and on CRAN assume it. In our context here of CA methods, Kleiner *et al.* (2012) also assumes i.i.d. throughout, except for the final section, which briefly discusses an extension to time series.

However, one seldom sees discussion of the implications of that assumption for day-to-day data analysis, and for CA there is an important reason to bring it up: For many data sets, the physical storage has been ordered in some way, rather than being "random."

For example, suppose in a certain data set one of the variables is gender, say consisting of 5000 men and 5000 women. The data file may have been arranged so that the first 5000 records consist of the men and the second 5000 contain the data for the women. Suppose $r = 2$. Then the distribution in the first chunk is different from that of the second, i.e., the chunks are not identically distributed, and the CA theory does not hold.

Thus, if the analyst knows or suspects that the arrangement of the data is ordered in some way, they should first apply a random permutation to the n records. If the matrix **x** contains the original data, then one might run, say

```
x <- x[sample(1:n, n, replace = FALSE), ]
```

But what about the truly non-identically distributed case? Our derivation here could be generalized to this case. This would involve a proper definition of the term *asymptotic*, along the lines of the classic analysis in Jennrich (1969), in which it is essentially assumed that the empirical cdf of the data converges to some distribution.

9. Conclusions and future work

The method developed here turns non-embarrassingly parallel calculations into statistically equivalent embarrassingly parallel ones. It was found here to produce excellent speedups in a diverse set of applications. It is also very easy and convenient to use, requiring no expertise in parallel algorithms or subsampling techniques.

As noted earlier, areas for future investigation are the study of possible stability-enhancing effects of CA for iterative algorithms, a formal derivation for the nonidentically distributed case, and further investigation into CWA. Extension to time series models would also be of interest.

General code for CA, automating much of the computation of estimators and overall standard errors, is available in the **partools** package (Matloff and Rumbaugh 2016). The algorithm can also be easily ported for R packages that run under Hadoop, such as **rmr2** (Revolution Analytics 2015).

Acknowledgements

The author wishes to thank the referees and an associate editor for valuable suggestions.

References

Breiman L (1996). "Bagging Predictors." *Machine Learning*, **24**(2), 123–140. doi:10.1007/bf00058655.

Christensen D (2005). "Fast Algorithms for the Calculation of Kendall's τ." *Computational Statistics*, **20**, 51–62. doi:10.1007/bf02736122.

Claeskens D, Hjort NL (2008). *Model Selection and Model Averaging*. Cambridge University Press.

Cleveland W (2016). "**Tessera**: Open Source Environment for Deep Analysis of Large Complex Data." URL http://tessera.io/.

Cohen J, Dolan B, Dunlap M, Hellerstein J, Welton C (2008). "MAD Skills: New Analysis Practices for Big Data." *Technical report*, University of California, Berkeley.

Dümbgen L, Rufibach K (2011). "**logcondens**: Computations Related to Univariate Log-Concave Density Estimation." *Journal of Statistical Software*, **39**(6), 1–28. doi:10.18637/jss.v039.i06.

Eddelbuettel D (2016). "CRAN Task View: High-Performance and Parallel Computing with R." Version 2016-06-27, URL https://CRAN.R-project.org/view=HighPerformanceComputing.

Fan TH, Lin D, Cheng KF (2007). "Regression Analysis for Massive Datasets." *Data & Knowledge Engineering*, **61**(3), 554–562. doi:10.1016/j.datak.2006.06.017.

Filzmoser P, Fritz H, Kalcher K (2014). ***pcaPP**: Robust PCA by Projection Pursuit*. R package version 1.9-60, URL https://CRAN.R-project.org/package=pcaPP.

Guha S (2010). "**RHIPE**: A Distributed Environment for the Analysis of Large and Complex Datasets." Seminar talk, Bay Area R Users Group, March 11, 2010, URL http://files.meetup.com/1225993/RHIPE%20-%20Saptarshi%20Guha.pdf.

Guha S, Hafen R, Kidwell P, Cleveland WS (2009). "Visualization Databases for the Analysis of Large Complex Datasets." In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS) 2009*, pp. 193–200. Clearwater Beach.

Guha S, Hafen R, Xia J, Rounds J, Li J, Xi B, Cleveland WS (2012). "Large Complex Data: Divide and Recombine (D&R) with **RHIPE**." *Stat*, **1**(1), 53–67. doi:10.1002/sta4.7.

Hegland M, McIntosh I, Turlach B (1999). "A Parallel Solver for Generalised Additive Models." *Computational Statistics & Data Analysis*, **31**(4), 377–396. doi:10.1016/s0167-9473(99)00038-9.

Hess K, Gentleman R (2014). ***muhaz**: Hazard Function Estimation in Survival Analysis.* R package version 1.2.6, URL https://CRAN.R-project.org/package=muhaz.

Jennrich RI (1969). "Asymptotic Properties of Non-Linear Least Squares Estimators." *The Annals of Mathematical Statistics*, **40**, 633–643. doi:10.1214/aoms/1177697731.

Kleiner A, Talwalkar A, Sarkar P, Jordan M (2012). "A Scalable Bootstrap for Massive Data." *Technical report*, University of California, Berkeley.

Koenker R (2016). ***quantreg**: Quantile Regression.* R package version 5.26, URL https://CRAN.R-project.org/package=quantreg.

Kosec G, Depolli M (2012). "Superlinear Speedup in OpenMP Parallelization of a Local PDE Solver." In *MIPRO, 2012 Proceedings of the 35th International Convention*, pp. 389–394.

Lee KH, Choi H, Chung Y, Lee YJ, Moon B (2011). "Parallel Data Processing with MapReduce: A Survey." *SIGMOD Record*, **40**(4), 11–20. doi:10.1145/2094114.2094118.

Matloff N (2010). "Future Directions for Parallel R in Data Analysis." Seminar talk, Fudan University, Shanghai, December 30, 2010, URL http://heather.cs.ucdavis.edu/Fudan.pdf.

Matloff N (2013). "Long Live (Big Data-Fied) Statistics!" In *JSM Proceedings*, pp. 98–108. Alexandria.

Matloff N, Rumbaugh A (2016). ***partools**: Tools for the **parallel** Package.* R package version 1.1.5, URL https://CRAN.R-project.org/package=partools.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Revolution Analytics (2015). ***rmr2**: R and Hadoop Streaming Connector.* R package version 3.3.1, URL https://github.com/RevolutionAnalytics/rmr2.

Schmidberger M, Morgan M, Eddelbuettel D, Yu H, Tierney L, Mansmann U (2009). "State of the Art in Parallel Computing with R." *Journal of Statistical Software*, **31**(1), 1–27. doi:10.18637/jss.v031.i01.

Tierney L, Rossini AJ, Li N, Sevcikova H (2015). ***snow**: Simple Network of Workstations.* R package version 0.4-1, URL https://CRAN.R-project.org/package=snow.

Affiliation:

Norman Matloff
Department of Computer Science
University of California, Davis
Davis, CA, 95616, United States of America
E-mail: matloff@cs.ucdavis.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/

July 2016, Volume 71, Issue 4 *Submitted:* 2012-09-05
doi:10.18637/jss.v071.i04 *Accepted:* 2015-04-12