# Software Alchemy:

# Turning the Complex into Embarrassingly Parallel

Norm Matloff

Department of Computer Science, University of California at Davis

Bay Area R Users Group, April 12, 2011

# On the Web

This PDF file contains my presentation at the R meeting. I've extended the document by including material summarizing the question-and-answer period of that talk, and will occasionally add some updates as well.

The most up-to-date version of these slides, and associated R code, will be available on the Web at `http://heather.cs.ucdavis.edu/barugApr11/`.

# The Setting

# The Setting

- **Problem:** Large data sets and complex statistical methods require large amounts of computation.

# The Setting

- **Problem:** Large data sets and complex statistical methods require large amounts of computation.
- **Solution:** Use a multicore machine or cluster.

# The Setting

- **Problem:** Large data sets and complex statistical methods require large amounts of computation.
- **Solution:** Use a multicore machine or cluster.
- **Problem:** The above solution usually works well only for *embarrassingly parallel* (EP) problems.

# The Setting

- **Problem:** Large data sets and complex statistical methods require large amounts of computation.
- **Solution:** Use a multicore machine or cluster.
- **Problem:** The above solution usually works well only for *embarrassingly parallel* (EP) problems. (Especially for R, given its functional programming approach.)

# The Setting

- **Problem:** Large data sets and complex statistical methods require large amounts of computation.
- **Solution:** Use a multicore machine or cluster.
- **Problem:** The above solution usually works well only for *embarrassingly parallel* (EP) problems. (Especially for R, given its functional programming approach.)
- **"Solution":** Run in parallel only if you have an embarrassingly parallel algorithm. :-)

# A (Rather) General Method

# A (Rather) General Method

- I will present a rather general solution here...

# A (Rather) General Method

- I will present a rather general solution here...I might even say a panacea.

# A (Rather) General Method

- I will present a rather general solution here...I might even say a panacea.
- Works for most *statistical* problems.

# A (Rather) General Method

- I will present a rather general solution here...I might even say a panacea.
- Works for most *statistical* problems.
- Our goal here: Turn highly NON-EP problems into EP ones!

# Overview

# Overview

- Old, old idea in parallel processing: Break data into chunks, work on each chunk, then combine results.

# Overview

- Old, old idea in parallel processing: Break data into chunks, work on each chunk, then combine results.
- But this requires EP.

# Overview

- Old, old idea in parallel processing: Break data into chunks, work on each chunk, then combine results.
- But this requires EP.
- New approach: Exploit the statistical properties.

# Overview

- Old, old idea in parallel processing: Break data into chunks, work on each chunk, then combine results.
- But this requires EP.
- New approach: Exploit the statistical properties.
- Key point:

# Overview

- Old, old idea in parallel processing: Break data into chunks, work on each chunk, then combine results.
- But this requires EP.
- New approach: Exploit the <u>statistical</u> properties.
- Key point: Calculate a **statistically equivalent** quantity that lends itself to EP computation.

# How It Works

## How It Works

- Suppose we wish to calculate an estimator $\widehat{\theta}$, say regression coefficients.

# How It Works

- Suppose we wish to calculate an estimator $\widehat{\theta}$, say regression coefficients.
- Have n data points, p processes (e.g. p $=$ 2 for dual core on a single machine).

# How It Works

- Suppose we wish to calculate an estimator $\widehat{\theta}$, say regression coefficients.
- Have n data points, p processes (e.g. $p = 2$ for dual core on a single machine).
- Break into r chunks of n/p data points each.

# How It Works

- Suppose we wish to calculate an estimator $\widehat{\theta}$, say regression coefficients.
- Have n data points, p processes (e.g. p = 2 for dual core on a single machine).
- Break into r chunks of n/p data points each.
- For i = 1,...,r calculate $\widehat{\theta}$ on chunk i, yielding $\tilde{\theta}_i$.
- Average all those chunked values:

$$\overline{\theta} = \frac{1}{r} \sum_{i=1}^{r} \tilde{\theta}_i$$

# What Does That Give You?

# What Does That Give You?

- The result, $\overline{\theta}$ can be proven to have the **same statistical accuracy** as the original $\widehat{\theta}$.

## What Does That Give You?

- The result, $\overline{\theta}$ can be proven to have the **same statistical accuracy** as the original $\widehat{\theta}$. (Manuscript in preparation.)

# What Does That Give You?

- The result, $\overline{\theta}$ can be proven to have the **same statistical accuracy** as the original $\widehat{\theta}$. (Manuscript in preparation.)
- But the computation of $\overline{\theta}$ is EP even if $\widehat{\theta}$ is non-EP.

# What Does That Give You?

- The result, $\overline{\theta}$ can be proven to have the **same statistical accuracy** as the original $\widehat{\theta}$. (Manuscript in preparation.)
- But the computation of $\overline{\theta}$ is EP even if $\widehat{\theta}$ is non-EP.
- Alchemy! Non-EP $\rightarrow$ EP.

# Example: Regression

# Example: Regression

What chunking does here:

# Example: Regression

What chunking does here:

- set up r R processes (via **snow**, **Rmpi**, **Rdsm** or whatever)

# Example: Regression

What chunking does here:

- set up r R processes (via **snow**, **Rmpi**, **Rdsm** or whatever)
- call **lm()** on each chunk

# Example: Regression

What chunking does here:

- set up r R processes (via **snow**, **Rmpi**, **Rdsm** or whatever)
- call **lm()** on each chunk (EP)

# Example: Regression

What chunking does here:

- set up r R processes (via **snow**, **Rmpi**, **Rdsm** or whatever)
- call **lm()** on each chunk (EP)
- average the regression coefficients over all chunks

# Example: Regression

What chunking does here:

- set up r R processes (via **snow**, **Rmpi**, **Rdsm** or whatever)
- call **lm()** on each chunk (EP)
- average the regression coefficients over all chunks
- use those values as your coefficients

## Example: Regression

What chunking does here:

- set up r R processes (via **snow**, **Rmpi**, **Rdsm** or whatever)
- call **lm()** on each chunk (EP)
- average the regression coefficients over all chunks
- use those values as your coefficients
- will have the **same statistical accuracy**, but will be faster

# Some Experiments with Regression

# Some Experiments with Regression

- compared ordinary sequential **lm()**,

## Some Experiments with Regression

- compared ordinary sequential **lm()**, my chunked method, and

# Some Experiments with Regression

- compared ordinary sequential **lm()**, my chunked method, and **gputools** (R package to interface GPU cards)

# Some Experiments with Regression

- compared ordinary sequential **lm()**, my chunked method, and **gputools** (R package to interface GPU cards)
- n = number of data points, q = number of predictors, p = number of processes (deg. of parallelism)

## Some Experiments with Regression

- compared ordinary sequential **lm()**, my chunked method, and **gputools** (R package to interface GPU cards)
- n = number of data points, q = number of predictors, p = number of processes (deg. of parallelism)
- used 3 dual-core PCs, so $p \leq 6$

# Some Experiments with Regression

- compared ordinary sequential **lm()**, my chunked method, and **gputools** (R package to interface GPU cards)
- n = number of data points, q = number of predictors, p = number of processes (deg. of parallelism)
- used 3 dual-core PCs, so $p \leq 6$
- regression is a non-EP problem

# Regression Experiments, cont'd.

Elapsed times in seconds (single runs):

# Regression Experiments, cont'd.

Elapsed times in seconds (single runs):

| n | q | p | ordinary | NM method | gputools |
|---|---|---|---|---|---|
| 500000 | 30 | 6 | 4.18 | 3.58 | 8.40 |
| 500000 | 50 | 6 | 9.41 | 6.61 | exceeded mem. |
| 100000 | 100 | 6 | 4.13 | 3.55 | 3.86 |
| 50000 | 150 | 6 | 4.14 | 3.36 | 2.92 |

# Regression Experiments, cont'd.

Elapsed times in seconds (single runs):

| n | q | p | ordinary | NM method | gputools |
|---|---|---|---|---|---|
| 500000 | 30 | 6 | 4.18 | 3.58 | 8.40 |
| 500000 | 50 | 6 | 9.41 | 6.61 | exceeded mem. |
| 100000 | 100 | 6 | 4.13 | 3.55 | 3.86 |
| 50000 | 150 | 6 | 4.14 | 3.36 | 2.92 |

NM method "handicapped": used **snow** (which uses **serialize()**), over a network.

# Second Example: Quantile Regression

## Second Example: Quantile Regression

- Model the population conditional quantiles, say medians, as a linear function.

## Second Example: Quantile Regression

- Model the population conditional quantiles, say medians, as a linear function.
- **VERY** non-EP.

# Second Example: Quantile Regression

- Model the population conditional quantiles, say medians, as a linear function.
- **VERY** non-EP.

Elapsed times in seconds (single runs):

| n | q | p | ordinary | NM method |
|---|---|---|---|---|
| 10000 | 50 | 2 | 2.39 | 1.50 |
| 10000 | 50 | 4 | 2.39 | 1.34 |
| 50000 | 50 | 4 | 36.10 | 13.43 |
| 50000 | 50 | 6 | 35.51 | 11.19 |

# Some Comments

## Some Comments

- How general is this method?

## Some Comments

- How general is this method?
    - My proof applies to i.i.d. random samples.

# Some Comments

- How general is this method?
  - My proof applies to i.i.d. random samples.
  - Proof could be extended to designed-experiment settings, e.g. clinical trials with pre-assigned sample sizes for treatment and control groups.

# Some Comments

- How general is this method?
  - My proof applies to i.i.d. random samples.
  - Proof could be extended to designed-experiment settings, e.g. clinical trials with pre-assigned sample sizes for treatment and control groups.
- I have R code available: General code to do the chunking in **snow** or **Rdsm**, and the specific code used for the simulations here.

## Some Comments

- How general is this method?
  - My proof applies to i.i.d. random samples.
  - Proof could be extended to designed-experiment settings, e.g. clinical trials with pre-assigned sample sizes for treatment and control groups.
- I have R code available: General code to do the chunking in **snow** or **Rdsm**, and the specific code used for the simulations here.
- Chunking has been used before for a different goal, that of larger-than-memory data sets: R's **biglm()**; Fan and Cheng (2007)

## Some Comments

- How general is this method?
  - My proof applies to i.i.d. random samples.
  - Proof could be extended to designed-experiment settings, e.g. clinical trials with pre-assigned sample sizes for treatment and control groups.
- I have R code available: General code to do the chunking in **snow** or **Rdsm**, and the specific code used for the simulations here.
- Chunking has been used before for a different goal, that of larger-than-memory data sets: R's **biglm()**; Fan and Cheng (2007)

# Q&A Period (slightly updated)

*Question: Does this only work on linear regression problems?*

- No, the math works on any function of i.i.d. data.
- I've tried it on logistic regression, principle components and estimation of hazard functions from censored data, getting modest to excellent speedups.
- Note that if $\widehat{\theta}$ is an unbiased estimator, then $\overline{\theta}$ is also unbiased.

*Question: Is there a convergence rate issue in your asymptotics?*

- In my experiments I've found only tiny differences between $\overline{\theta}$ and $\widehat{\theta}$.
- The only problems that are worth parallelizing have very large sample sizes, and thus the asymptotics have certainly taken effect by then.