We Just Ran Two Million Regressions Faster

Damian C. Clarke George Vega Yon

August 23, 2012

Abstract

1 Introduction

The multilinear regression is the workhorse tool used in a large majority of applied economic work. Regression analysis is both a powerful and—as the title of Sala-i-Martin (1997) suggests—ubiquitous method in economics. However, despite great increases in the efficiency of modern computation and considerable advances in the complexity and availability of data, the basic way economists *physically* run regressions has changed little over the past decade. Fundamentally economists rely on the inbuilt ability of statistical programs such as Stata, MATLAB, R and so forth, to project vectors of observations onto each other, and hence estimate the coefficients of interest from their underlying equation.

This methodology, whilst completely effective in doing its job, is relatively inefficient when considering both computational and human resources. The delay in calculation time experienced when using large datasets is often considerable, meaning that researchers or their research assistants are forced to "run regressions" and then wait some non-trivial amount of time to study regression outputs. Such a process may imply interruption to the economist's ability to undertake research, and at the very least implies tying up the program or computer (or terminal's) analytical capacity whilst the regression "runs".

In this paper we provide a methodology by which the efficiency of multilinear regressions – and indeed a large range of economic calculations – can be increased by the order of up to XX. Whilst this methodology is not novel in the physical sciences, corporate enterprises, or even in

some iterative economic processes (see for example Aldrich et al., 2011), we believe that this paper represents its first analysis in terms of linear regression in which its successful(??) application is demonstrated in potentially the most widely used statistical program in applied economics; Stata.

By taking advantage of parallel computing and ...

2 Why Parallel computing?

- Huge data available (specially administrativa data) allow researchers and governments to replicate and inquire deeply in reality. Big data-ware houses are been built every where motive my open-data and transparency movements.
- Impressive advances in GPU technologies taking computation into a whole new level through parallel computation. In less than ten years, the computational capabilities of a home computer have multiplied thousands of times[1]
- Thus, the necessity of making assumptions such as ceteris-paribus is no longer necessary. Researchers can now simulate complete societies from the comfort of their desks just using mid capacity computers.

3 The Multilinear Regression and Parallel Computing

The nature of the matrix algebra employed in multilinear regression means that efficiency gains associated with parallel computing are particularly large relative to traditional single-core computes. As each variable is expressed as a vector and each of these vectors must be projected onto itself and each other vector (variable), this implies that for each pair of variables (p, q) a stepwise sum must be performed of each observation's coproduct $x_{pi}x_{qi}$, for $i \in \{1, \ldots, n\}$. In this way, as the number of observations n in a set of data increases, the quantity of individual multiplications must increase in line with this for each pair of variables.

Hence, rather than working stepwise through a summation of n units,[2] parallel computing allows us to split each vector and resulting summation into c sub-vectors of $\frac{n}{c}$ observations, where

[1] The new NVIDIA video card for laptos, GeForce GTX 680M, has 1344 cores.

[2] This is to say first multiplying x_1 by x_1, then x_2 by x_2 and so forth up to x_n by x_n, whereby each x is a scalar and the subindex represents observation number.

c represents the number of cores which the computer contains in its CPU or GPU:

$$\sum_{c=1}^{C} \left(\sum_{i \forall i \in c}^{\frac{n}{c}} x_i^2 \right)_c . \tag{1}$$

Furthermore, given that the process of multiplication and summation suffers from virtually no increasing returns to scale where subsequent additions to the vector size imply decreasing marginal additions to calculation time, as c increases in number we expect that the amount of time that the regression takes to "run" should decrease c-fold.

Formally, each multilinear regression, expressed in terms of a matrix $\mathbf{X}$ of independent variables and a vector $\mathbf{y}$ representing the dependent variable, requires two computationally expensive matricial calculations: $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}'\mathbf{y}$, and the subsequent (faster) combination of the two resulting matrices to produce the vector of estimands. Given that each individual variable in $\mathbf{X}$ must be multiplied by itself and each other variable in $\mathbf{X}$ (and likewise for the variable y), the summation (1) is performed for each variable pair as per (2b):

$$(\mathbf{X}'\mathbf{X}) = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k1} & x_{k2} & \dots & x_{kn} \end{bmatrix} \begin{bmatrix} x_{11} & x_{21} & \dots & x_{k1} \\ x_{12} & x_{22} & \dots & x_{k2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1n} & x_{2n} & \dots & x_{kn} \end{bmatrix} \tag{2a}$$

$$\equiv \begin{bmatrix} \sum_{i=1}^{n} x_{1i}^2 & \sum_{i=1}^{n} x_{1i}x_{2i} & \dots & \sum_{i=1}^{n} x_{1i}x_{ki} \\ \sum_{i=1}^{n} x_{2i}x_{1i} & \sum_{i=1}^{n} x_{2i}^2 & \dots & \sum_{i=1}^{n} x_{2i}x_{ki} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_{ki}x_{1i} & \sum_{i=1}^{n} x_{ki}x_{2i} & \dots & \sum_{i=1}^{n} x_{ki}^2 \end{bmatrix}. \tag{2b}$$

GEORGE

In this way, even though economists run this model on a daily basis, because of the lack of advanced computational knowledge (they are not computer scientists), tend to visualize this computational problem as a one step computational problem, this is, when most of economists think in a matrix multiplication bit-level operations aren't seen.

In spite of the existence of matrix operation capable programing languages, the manner as these perform calculations stills a serial fashion. So, the speed gains come directly from the bit level compilations and not from "matrix operations". At the end these still are simple arithmetic operations. This is where parallel computing enters.

Using bit code compiled functions with parallel computing techniques ...

References

E. M. Aldrich, J. Fernández-Villaverde, A. Ronald Gallant, and J. F. Rubio-Ramírez. Tapping the supercomputer under your desk: Solving dynamic equilibrium models with graphics processors. *Journal of Economic Dynamics and Control*, 35(3):386–393, March 2011. URL `http://ideas.repec.org/a/eee/dyncon/v35y2011i3p386-393.html`.

X. Sala-i-Martin. I just ran two million regressions. *American Economic Review*, 87(2):178–83, May 1997.

A. Vespignani. Modelling dynamical processes in complex socio-technical systems. *Nature Physics*, 8(1):32–39, 2011.