The Stata Journal (*yyyy*) *vv*, Number *ii*, pp.“ 1“endash7

PARALLEL: Stata Module for Parallel Computing

George G. Vega Yon
g.vegayon@gmail.com

Abstract. With a multicore CPU computer, Stata users can achieve a substantial performance improvement using parallel: a module for parallel computing by means of shell scripting. Parallel can launch up as many Stata executables in batch mode as cores the CPU has to accelerate computations. By splitting the dataset, or repeated loading, into cluster parallel will repeat a task simultaneously over each core, which results in an efficiency improvement proportional to the number of CPU cores. Parallel delivers “kitchen sink” parallel computing to the data scientist. In this article we present its main features and show empirical applications using the command.

Keywords: st0001, parallel computing, simulations, high performance computing

Parallel computing is becoming reachable each day. Most modern computers processors consists in at least two cores. Frequently associated with *multitasking* users, many single applications can, and do, take advantage of having more than one processor.

A very close example, StataMP version. Making use of multi-processor computers, StataMP has managed to deliver out-of-the-box parallel computing capabilities to Stata without the user having to hold technical knowledge on parallel computing. That is how with StataMP users can do faster linear algebra, faster OLS estimation, and faster data management in general. Nevertheless, while quite a big advance, there is still space to improve its parallel computing capabilities.

For a start, the Multiprocessor version of Stata will not improve neither simulations or bootstrapping estimators *per se*, since the parallelization required to increase its performance relies on task and data-parallelization, this is, a higher (as in more simple, but specific) sort of parallelization. This is how *parallel* is originated.

1 Features

The latest version of parallel includes several built-in commands to take advantage of multicore processors, in particular, appending several datasets, running simulations, bootstrapping estimation, and embarrassingly parallel computing. These are described as follows:

1.1 appending datasets

```
parallel append [file(s)] , do(cmd|dofile) [in(in) if(if) expression(expand expression (see det
```



```
randtype(current|datetime|random.org) timeout(#)]
```

1.2 simulations

1.3 bootstrapping estimation

1.4 embarrassingly parallel computing

2 Extended example: Multiple imputation

3 Introduction

Multicore CPUs are standard in todays personal computer industry, and potentially move productivity upward for multi-tasking users. Several mathematical and statistical packages make good use of multicore CPUs, for example, MATLAB provides its own Parallel Computing Toolbox[1] which makes it possible to implement parallel computing methods through a multicore computer, GPUs and computer clusters. Alternatively, the GNU open-source R software does offer several packages to implement parallel computing algorithms such as "parallel" and "snow"[2]. With its multiprocessor edition, Stata/MP, Stata Corp. offers bit level parallellization that makes it possible to achieve up to (or greater than) constant scale speed improvements (**?**). However, besides of the fact that users must acquire special editions, some times bit level parellelization is not enough.

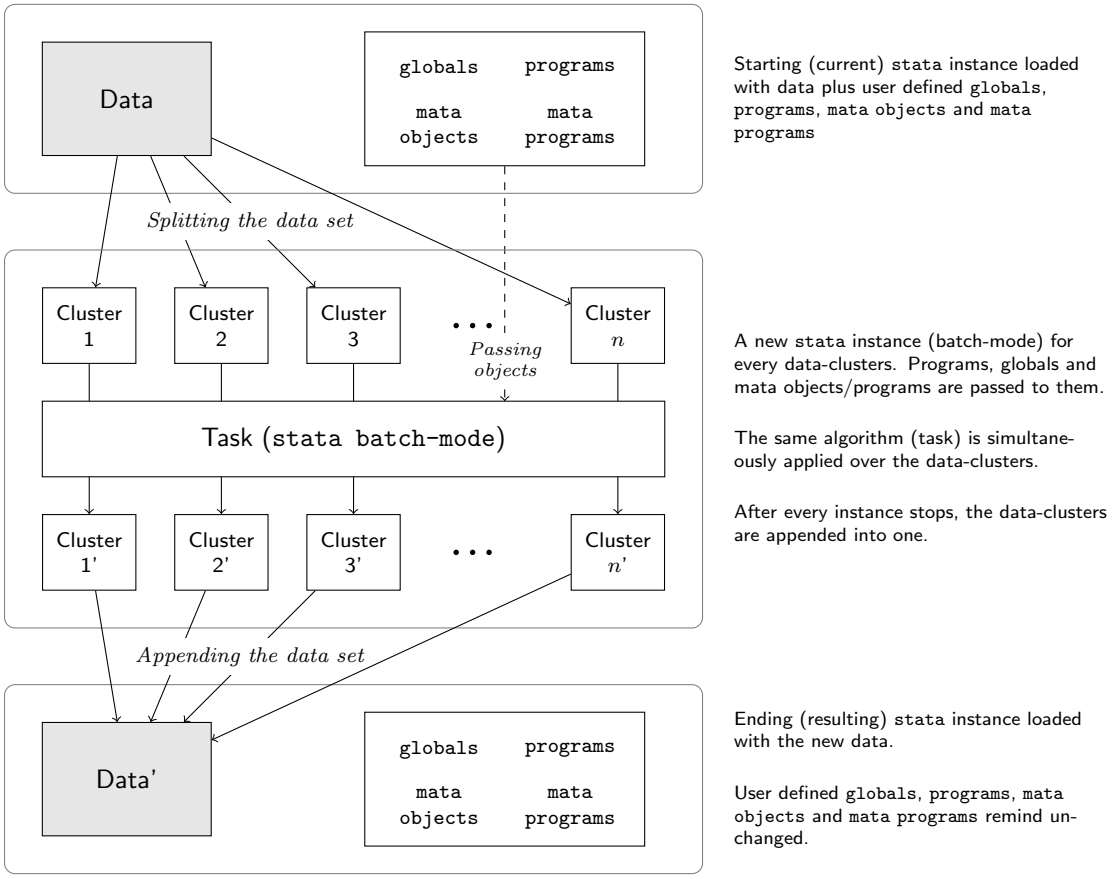
By using data or task parallelism, the module parallel allows the Stata user to drastically decrease the time required to complete repetitive computational problems. Parallel makes it possible to implement algorithms characterized by a large number of calculations or, in the case of Stata, code interpretation such as control-flow statements, like loops or simulation models.

4 Parallelization of Stata

To speedup computations, a dataset can be split and distributed over a given number of clusters and in so doing implement a data parallelism algorithm. Figure 1 illustrates this below the first block when parallel splits the dataset into n clusters.[3] Parallel starts n new independent Stata instances in batch mode by which the same task is simultaneously executed. By default, all loaded instances globals and programs are passed through. Optionally, mata programs (as mata libs) and objects are passed through as well. After

1. http://www.mathworks.com/products/parallel-computing/
2. For more details see CRAN Task View on High-Performance and Parallel Computing With R http://cran.r-project.org/web/views/HighPerformanceComputing.html
3. Or not, considering that the command also allows to work without data as it the user can also take advantage from just calling n Stata batch mode instances.

Figure 1: How parallel works



every cluster stops, the result datasets are appended and returned to the current Stata instance without modifying other elements.

Parallel is especially helpful in performance a series of computations on the same data sets but with different random samples. Example 5.3 shows how to assess possible sample bias within a cohort using parallel.

5 When to use parallel

Problems that are particularly suitable for data parallelization are:

- **Control flow statement** Mainly loops. Most loops can be parallelized as these do not require all observations to computer with. Like serial replacing (from observation to observation), loops are a suitable task to parallel computing. In the case that a loop needs to compute with clustered data (loops within individuals in a panel dataset), Stata users can ask parallel to avoid splitting clusters. Example 5.1 shows this.
- **Clustered computations** Such as an byable egen. Clustered egen can be parallelized without any problem as parallel can avoid splitting clusters. Stata users only have to declare it using the option by().
- **Bootstrapping** Since a bootstrapping process does not requires to work with all the generated data until the end of it, it is perfect to be computed in parallel process. Parallel allows to run simultaneously random data generation processes using different seeds for each independent computational procedure with reproducible results.
- **Monte Carlo Simulations** Example 5.2
- **Reshaping large datasets** Better than Stata/MP, parallel has shown to efficiently reshape large administrative datasets faster. Even though speed gains are not that of the examples showed in this article, parallel reaches speed improvements where Stata/MP does not.
- **Cohort sampling tasks like bias testing** Example 5.3

6 Syntax

Setting the number of clusters (data blocks)

```
parallel setclusters # [ , force statadir(string) ]
```

Parallelizing a do-file

```
parallel do filename [ , by(varlist) force keep keeplast programs mata
   noglobal seeds(numlist) nodata randtype(datetime|random.org)
   timeout(integer) processors(integer) ]
```

Parallelizing a Stata command

```
parallel [ , by(varlist) force keep keeplast programs mata noglobal
```

`seeds(`*numlist*`)` `nodata` `randtype(`*datetime*|*random.org*`)` `timeout(`*integer*`)` `processors(`*integer*`)`]: *stata_cmd*

Removing auxiliary files

`parallel clean` [`, event(`*pll_id*`)` `all`]

6.1 Options

Setting the number of clusters (data blocks)

`force` In order to protect the users' computers, setting more than 8 clusters it is not permitted (see the WARNING in description). With this option the user can skip this restriction.

`statadir` File path. parallel tries to automatically identify Stata's exe path. By using this option you will override this and force parallel to use a specific path of stata.exe.

Byable parallelization

`by` Varlist. Tells the command through which observations the current dataset can be splitted, avoiding stories splitting over two or more clusters (for example).

`force` When using by, parallel checks whether if the dataset is properly sorted. By using force the command skips this check.

Keeping auxiliary files

`keep` Keeps auxiliary files generated by parallel.

`keeplast` Keeps auxiliary files and remove those last time saved during the current session.

Passing Stata/Mata objects

`programs` In the case of having temporal programs loaded in the session, using this option parallel passes them to the clusters.

`mata` If the algorithm needs to use mata objects, this option allows to pass to each cluster every mata object loaded in the current session (including functions).

`noglobal` By default parallel takes into account every global macro loaded in the session. If the user needs to not include globals in clusters, he should use this option.

Simulation options

`seeds` Numlist. With this option the user can pass an specific seed to be used within each cluster.

`nodata` Tells parallel that there is no data loaded and thus should not try to split or append anything.

`randtype` String. Tells parallel whether to use the current seed to generate the seeds for each clusters (default option) or use the current datetime -datetime- or random.org API -random.org- (please read the Description section).

Removing auxiliary files

`event` Integer. Specifies which executed (and stored) event's files should be removed.

`all` Tells parallel to remove every remanent auxiliary files generated by it in the current directory.

Other options

`timeout` Integer.If a cluster hasn't start, how much time in seconds does parallel has to wait until assume that there was a connection error and thus the child process (cluster) won't start. Default value is 60.

`processors` Integer. If running on StataMP, sets the number of processors each cluster should use. Default value is 0 (do nothing).

7 Examples

Others examples are included in the Stata Help file of the command.

7.1 Serial replacing

This first test consists on, after a generation of N pseudo-random values, using Stata's `rnormal()` function, replacing each and every one of the observations in a serial way (loop) starting from 1 to N. The observation's variable was replaced using the PDF of the normal distribution

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

The code to be parallelized is

(INSERT CODE HERE)

which is contained inside a do-file named "myloop.do", and can be executed through four clusters with parallel as it follows

(INSERT CODE HERE)

This algorithm was repeated over $N \in \{10000; 100000; 1000000; 10000000\}$

7.2 Extended Example 1: Montecarlo Simulation

In **?** a simple Monte Carlo experiment is perform which simulates the performance of a estimator of sample mean, bar x, in a context of heteroskedasticity.

Table 1: Serial replacing using a loop on a Linux Server (16 clusters)

	100.000	1.000.000	10.000.000
CPU	1.43	16.94	144.68
Total	0.34	3.20	12.49
Setup	0.00	0.00	0.00
Compute	0.32	3.07	11.54
Finish	0.02	0.12	0.95
Ratio (compute)	4.50	5.51	12.53
Ratio (total)	4.22 (26%)	5.30 (30%)	11.58 (72%)

Tested on a Intel Xeon X470 (hexadeca-core) machine

The model to be

$$y_i = \mu + \epsilon_i \sim N(0, \sigma^2) \tag{1}$$

Let ϵ be a $N(0,1)$ variable multiplied by a factor cz_i, where z_i varies over i. We will vary parameter c between 0.1 and 1.0 and determine its effect on the point and interval estimates of μ; as a comparison, we will compute a second random variable which is homoskedastic, with the scale factor equalling $c\bar{z}$.

From web dataset `census2` the variables `age`, `region` (which can be 1, 2, 3 or 4) and the mean of `region` are used as μ, z_i and $\bar{z}$ respectively.

The simulation program, stored in "mcsimul1.ado", is defined as

(INSERT CODE HERE)

In what is next, the do-file that runs the simulation, stored as "montecarlo.do", it is compound of two parts: (a) setting the iteration range by which c is going to vary, and (b) looping over the selected range. For the first part the do-file uses the local macro `pll_instance` which is the numer of the parallel stata instance running, thus there are as many as clusters have been declared, number available with the global macro `PLL_CLUSTERS`. This way, if the macro `PLL_CLUSTERS` equals two and the macro `pll_instance` equals une, then the range will be defined from one to five[4].

(INSERT CODE HERE)

This do-file will be executed from stata using `parallel do` syntax. As there is no need of splitting any dataset (these are loaded every time that the main `loop_simul.do`'s loop runs), we add the option `nodata`. This way the main do-file will look like this Finally, the do-file from which parallel runs the simulation

4. Note that if the local macro `pll_id`, which contains a special random number that identifies an specific `parallel` run (more details in its help file), length is zero it means that the do-file is not running in parallel mode, thus it is been executed in a serial way where the loop range starts from one to ten.

(INSERT CODE HERE)

By this `parallel` will start, in this case, five new independent stata instances, each one looping over the ranges 1/2, 3/4, 5/6, 7/8 and 9/10 respectively.

Most of the code has been exactly copied with the exception of the addition of a new code line `set seed`. In order to be able to compare both serial and parallel implementations of the algorithm it was necessary to set a particular seed for each loop, inside "montecarlo.do" right before `simulate` command.

Table 2: Monte Carlo Experiment on a Windows Machine (4 clusters)

	2	4
CPU	111.49	114.13
Total	58.02	37.48
Setup	0.00	0.00
Compute	58.02	37.48
Finish	0.00	0.00
Ratio (compute)	1.92	3.04
Ratio (total)	1.92 (96%)	3.04 (76%)

Tested on a Intel i3 2120 (dual-core) machine

Table 3: Monte Carlo Experiment on a Linux Server (16 clusters)

	2	4	8	16
CPU	164.79	164.04	162.84	163.89
Total	69.85	34.28	19.00	10.78
Setup	0.00	0.00	0.00	0.00
Compute	69.85	34.28	19.00	10.78
Finish	0.00	0.00	0.00	0.00
Ratio (compute)	2.36	4.78	8.57	15.21
Ratio (total)	2.36 (118%)	4.78 (120%)	8.57 (107%)	15.21 (95%)

Tested on a Intel Xeon X470 (hexadeca-core) machine