Just tired of endless loops!
or `parallel`: Stata module for parallel computing

George G. Vega Yon[1]   Brian Quistorff[2]

[1]University of Southern California
vegayon@usc.edu

[2]Microsoft AI and Research
Brian.Quistorff@microsoft.com

Stata Conference Baltimore
July 27–28, 2017

Agenda

- Both computation power and size of data are ever increasing
- Often our work is easily broken down into independent chunks
- Implementing parallel computing even for these "embarrassingly parallel" problems, however, is not easy.
- StataMP exists, but only parallelizes a limited set of internal commands, not user commands.
- `parallel` aims to make this more convenient.

- Inspired by the R package "snow" (several other examples exists: HTCondor, Matlab's Parallel Toolbox, etc.)
- Launches child batch-mode Stata processes across multiple processors (e.g. simultaneous multi-threading, multiple cores, sockets, cluster nodes).
- Depending on the task, can reach near linear speedups proportional to the number of processors.
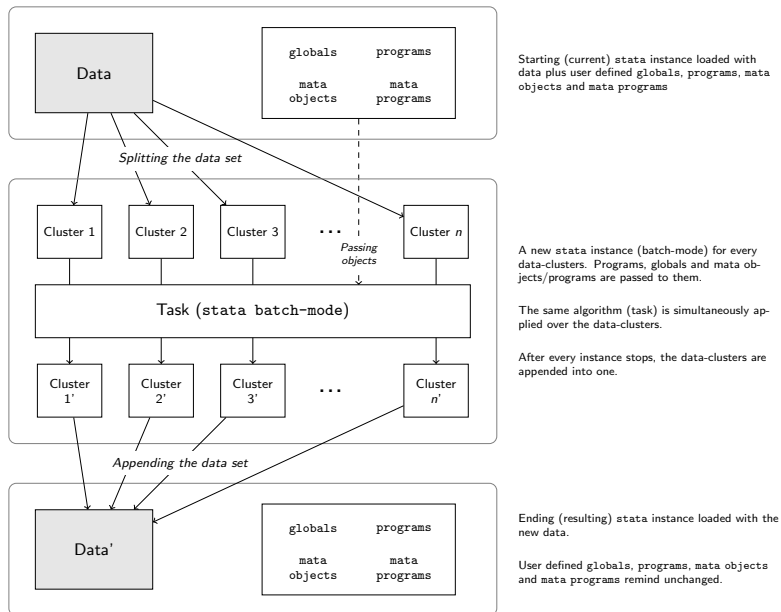 - Thus having a quad-core computer can lead to a 400% speedup.

Simple usage

Serial:

- gen v2 = v*v
- do byobs_calc.do
- bs, reps(5000): reg price foreign rep

Parallel:

- parallel: gen v2 = v*v
- parallel bs, reps(5000): reg price foreign rep
- parallel do byobs_calc.do

What is it and how does it work
How does it work?

Starting (current) stata instance loaded with data plus user defined globals, programs, mata objects and mata programs

Splitting the data set

Passing objects

A new stata instance (batch-mode) for every data-clusters. Programs, globals and mata objects/programs are passed to them.

The same algorithm (task) is simultaneously applied over the data-clusters.

After every instance stops, the data-clusters are appended into one.

Appending the data set

Ending (resulting) stata instance loaded with the new data.

User defined globals, programs, mata objects and mata programs remind unchanged.

- Method is *split-apply-combine* like MapReduce. Very flexible!
- Straightforward usage when there is observation- or group-level work
- If each iteration needs the entire dataset, then use procedure to split the tasks and load the data separately. Examples
  - Table of seeds for each bootstrap resampling
  - Table of parameter values for simulations
- If the list of tasks is data-dependent then the "nodata" alternative mechanism allows for more flexibility.

Implementation

- Uses shell on Linux/MacOS. On Windows we have a compiled plugging allowing
    - Functionality when the parent Stata is in batch-mode
    - Seamless user experience by launching the child programs in a hidden desktop (otherwise GUI for each steals focus)
- For a computer cluster with a shared filesystem (e.g. NFS) can distribute across nodes.
    - New feature so we'd appreciate help from the community to extend to other cluster settings (e.g. PBS)

Benchmarks
Bootstrap with `parallel bs`

```
sysuse auto, clear expand 10

// Serial fashion
bs, rep($size) nodots: regress mpg weight gear foreign

// Parallel fashion parallel setclusters $number_of_clusters
parallel bs, rep($size) nodots: regress mpg weight gear foreign
```

Problem size	Serial	2 Clusters	4 Clusters
1,000	2.93s	1.62s	1.09s
	×2.69	×1.48	×1.00
2,000	5.80s	3.13s	2.03s
	×2.85	×1.54	×1.00
4,000	11.59s	6.27s	3.86s
	×3.01	×1.62	×1.00

Table: Absolute and relative computing times for each run of a basic bootstrap problem. For each given problem size, the first row shows the time in seconds, and the second row shows the relative time each method took to complete the task relative to using parallel with four clusters. Each cell represents a 1,000 runs.

Benchmarks
Simulations with `parallel sim`

```
prog def mysim, rclass
    // Data generating process
    drop _all
    set obs 1000
    gen eps = rnormal()
    gen X = rnormal()
    gen Y = X*2 + eps

    // Estimation
    reg Y X
    mat def ans = e(b)
    return scalar beta = ans[1,1]
end

// Serial fashion
simulate beta=r(beta), reps($size) nodots: mysim

// Parallel fashion
parallel setclusters $number_of_clusters
parallel sim, reps($size) expr(beta=r(beta)) nodots: mysim
```

Problem size	Serial	2 Clusters	4 Clusters
1000	2.19s	1.18s	0.73s
	×3.01	×1.62	×1.00
2000	4.36s	2.29s	1.33s
	×3.29	×1.73	×1.00
4000	8.69s	4.53s	2.55s
	×3.40	×1.77	×1.00

Table: Absolute and relative computing times for each run of a simple Monte Carlo exercise. For each given problem size, the first row shows the time in seconds, and the second row shows the relative time each method took to complete the task relative to using parallel with four clusters. Each cell represents a 1,000 runs.

Code for replicating this is available at
https://github.com/gvegayon/parallel

Syntax and Usage

Setup

parallel setclusters #|default [, <u>f</u>orce <u>h</u>ostnames(namelist)]

Main command types

parallel [, by(*varlist*) programs(*namelist*) <u>m</u>ata seeds(*string*) <u>r</u>andtype(*random.org|datetime*)
        <u>no</u>data]: *stata_cmd*

parallel do *filename* [, by(*varlist*) programs(*namelist*) <u>m</u>ata seeds(*string*)
        <u>r</u>andtype(*random.org|datetime*) <u>no</u>data]

Helper commands

parallel bs [, <u>ex</u>pression(*exp_list*) programs(*namelist*) <u>m</u>ata seeds(*string*)
        <u>r</u>andtype(*random.org|datetime*) *bs_options*]: *stata_cmd*

parallel sim [, <u>ex</u>pression(*exp_list*) programs(*namelist*) <u>m</u>ata seeds(*string*)
        <u>r</u>andtype(*random.org|datetime*) *sim_options*)]: *stata_cmd*

parallel append [*files*], do(*command/dofile*) [in(*in*) if(*if*) <u>ex</u>pression(*expand_exp*)
        programs(*namelist*) <u>m</u>ata <u>s</u>eeds(*string*) <u>r</u>andtype(*random.org|datetime*)]

Additional Utilities

parallel version/clean/printlog/viewlog/numprocessors

Syntax and Usage
Recommendations on its usage

`parallel suits ...`

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).
- Bootstrapping/Jackknife.
- Multiple MCMC chains to test for convergence (Gelman-Rubin test).
- Simulations in general.

`parallel doesn't suit ...`

- (already) fast commands.
- Regressions, ARIMA, etc.
- Linear Algebra.
- Whatever StataMP does better.
- (Currently) Tasks that already take up all of RAM.

Use in other Stata modules

- EVENTSTUDY2: Perform event studies with complex test statistics
- MIPARALLEL: Perform parallel estimation for multiple imputed datasets
- Synth_Runner: Performs multiple Synthetic Control estimations for permutation testing

Concluding Remarks

- Brings parallel computing to many more commands than StataMP
- Its major strengths/advantages are in simulation models and non-vectorized operations such as control-flow statements.
- Depending on the proportion of the algorithm that can be parallelized, it is possible to reach near to linear scale speedups.
- We welcome other user commands optionally including `parallel` for speedup.
- Contribute, find help, and report bugs at http://github.com/gvegayon/parallel

Thank you very much!

George G. Vega Yon[1]   Brian Quistorff[2]

[1]University of Southern California
vegayon@usc.edu

[2]Microsoft AI and Research
Brian.Quistorff@microsoft.com

Stata Conference Baltimore
July 27–28, 2017