# 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

Thanks to Stata users worldwide for their valuable contributions. The usual disclaimers applies.

## Agenda

Motivation

What is and how does it work

Benchmarks

Syntax and Usage

Concluding Remarks

## Motivation

- Both computation speeds and size of data are ever increasing
- Often our work is easily broken down into independent chunks
- But, implementing parallel computing even for these “embarrassingly parallel” problems is not easy,

## Motivation

- Both computation speeds and size of data are ever increasing
- Often our work is easily broken down into independent chunks
- But, implementing parallel computing even for these “embarrassingly parallel” problems is not easy, most of this due to lack of (user-friendly) statistical computing tools.

## Motivation

- Both computation speeds and size of data are ever increasing
- Often our work is easily broken down into independent chunks
- But, implementing parallel computing even for these "embarrassingly parallel" problems is not easy, most of this due to lack of (user-friendly) statistical computing tools.
- `parallel` aims to make a contribution to these issues.

Motivation

What is and how does it work

Benchmarks

Syntax and Usage

Concluding Remarks

## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.

## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.
- Inspired by the R package “snow” (several other examples exists: StataMP, HTCondor, Matlab’s Parallel Toolbox, etc.)

## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.
- Inspired by the R package “snow” (several other examples exists: StataMP, HTCondor, Matlab’s Parallel Toolbox, etc.)
- Is designed to be used in multiprocessor machines (e.g. simultaneous multi-threading, multiple cores, multiple sockets).

## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.
- Inspired by the R package "snow" (several other examples exists: StataMP, HTCondor, Matlab's Parallel Toolbox, etc.)
- Is designed to be used in multiprocessor machines (e.g. simultaneous multi-threading, multiple cores, multiple sockets).
- It implements parallel computing methods through an OS's shell scripting (using Stata in batch mode).

## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.
- Inspired by the R package “snow” (several other examples exists: StataMP, HTCondor, Matlab’s Parallel Toolbox, etc.)
- Is designed to be used in multiprocessor machines (e.g. simultaneous multi-threading, multiple cores, multiple sockets).
- It implements parallel computing methods through an OS’s shell scripting (using Stata in batch mode).
- On Windows we have a compiled plugging allowing

## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.
- Inspired by the R package "snow" (several other examples exists: StataMP, HTCondor, Matlab's Parallel Toolbox, etc.)
- Is designed to be used in multiprocessor machines (e.g. simultaneous multi-threading, multiple cores, multiple sockets).
- It implements parallel computing methods through an OS's shell scripting (using Stata in batch mode).
- On Windows we have a compiled plugging allowing
  - Functionality when the parent Stata is in batch-mode

## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.
- Inspired by the R package "snow" (several other examples exists: StataMP, HTCondor, Matlab's Parallel Toolbox, etc.)
- Is designed to be used in multiprocessor machines (e.g. simultaneous multi-threading, multiple cores, multiple sockets).
- It implements parallel computing methods through an OS's shell scripting (using Stata in batch mode).
- 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)

## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.
- Inspired by the R package "snow" (several other examples exists: StataMP, HTCondor, Matlab's Parallel Toolbox, etc.)
- Is designed to be used in multiprocessor machines (e.g. simultaneous multi-threading, multiple cores, multiple sockets).
- It implements parallel computing methods through an OS's shell scripting (using Stata in batch mode).
- 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)
- Depending on the task, can reach near to (or over) linear speedups proportional to the number of physical cores of the computer.

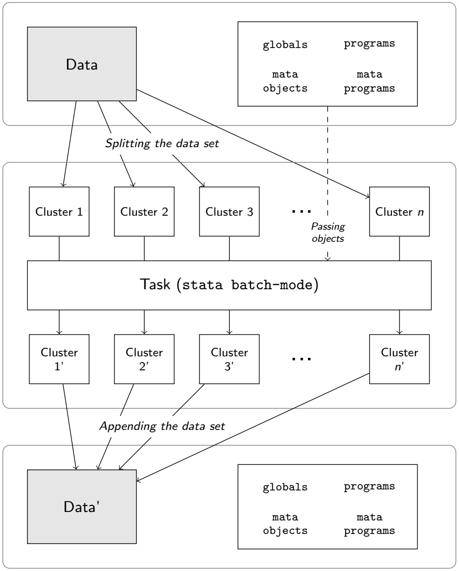
## What is and how does it work

### What is?

- NB: StataMP is available but only parallelizes some internal commands. Therefore a user-provided solution is needed.
- Inspired by the R package "snow" (several other examples exists: StataMP, HTCondor, Matlab's Parallel Toolbox, etc.)
- Is designed to be used in multiprocessor machines (e.g. simultaneous multi-threading, multiple cores, multiple sockets).
- It implements parallel computing methods through an OS's shell scripting (using Stata in batch mode).
- 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)
- Depending on the task, can reach near to (or over) linear speedups proportional to the number of physical cores of the computer.
- Thus having a quad-core computer can lead to a 400% speedup.

# What is 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`

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.

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

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

## What is and how does it work

### How does it work?

- Method is *split-apply-combine* like MapReduce

## What is and how does it work

### How does it work?

- Method is *split-apply-combine* like MapReduce
- Straightforward usage when there is observation or group-level work

## What is and how does it work

### How does it work?

- Method is *split-apply-combine* like MapReduce
- Straightforward usage when there is observation or group-level work
- If each iteration needs the entire dataset, then can be reformulated as

## What is and how does it work

### How does it work?

- Method is *split-apply-combine* like MapReduce
- Straightforward usage when there is observation or group-level work
- If each iteration needs the entire dataset, then can be reformulated as
    1. Save real dataset to file

## What is and how does it work

### How does it work?

- Method is *split-apply-combine* like MapReduce
- Straightforward usage when there is observation or group-level work
- If each iteration needs the entire dataset, then can be reformulated as
  1. Save real dataset to file
  2. Generate in-memory data as the list of tasks. For example,

## What is and how does it work

### How does it work?

- Method is *split-apply-combine* like MapReduce
- Straightforward usage when there is observation or group-level work
- If each iteration needs the entire dataset, then can be reformulated as
    1. Save real dataset to file
    2. Generate in-memory data as the list of tasks. For example,
        - List of seeds for each bootstrap resampling

## How does it work?

- Method is *split-apply-combine* like MapReduce
- Straightforward usage when there is observation or group-level work
- If each iteration needs the entire dataset, then can be reformulated as
  1. Save real dataset to file
  2. Generate in-memory data as the list of tasks. For example,
     - List of seeds for each bootstrap resampling
     - List of parameter values for simulations

## What is and how does it work

### How does it work?

- Method is *split-apply-combine* like MapReduce
- Straightforward usage when there is observation or group-level work
- If each iteration needs the entire dataset, then can be reformulated as
  1. Save real dataset to file
  2. Generate in-memory data as the list of tasks. For example,
     - List of seeds for each bootstrap resampling
     - List of parameter values for simulations
  3. Child process reads its list of tasks, loads the real dataset, and does it's own looping over tasks

## What is and how does it work

### How does it work?

- Method is *split-apply-combine* like MapReduce
- Straightforward usage when there is observation or group-level work
- If each iteration needs the entire dataset, then can be reformulated as
  1. Save real dataset to file
  2. Generate in-memory data as the list of tasks. For example,
     - List of seeds for each bootstrap resampling
     - List of parameter values for simulations
  3. Child process reads its list of tasks, loads the real dataset, and does it's own looping over tasks
- If the list of tasks is not initially definable then the "nodata" alternative mechanism allows for more flexibility.

## What is and how does it work

### Parallel's backend

When the user enters

```
parallel: gen x2 = x*x
```

`parallel` takes the command and writes something like this

## What is and how does it work

### Parallel's backend

When the user enters

```
parallel:  gen x2 = x*x
```

`parallel` takes the command and writes something like this

```
  cap clear all
  cd ~
1 set seed 34815
  set memory 16777216b
  cap set maxvar 5000
  cap set matsize 400
2 local pll_instance 1
  local pll_id efcql2tspr
  capture {
  noisily {
3 use __pllefcql2tsprdataset if _efcql2tsprcut == 1
  gen x2 = x*x
  }
  }
4 save __pllefcql2tsprdta1, replace
  local result = _rc
  cd ~
5 mata: write_diagnosis(st_local("result"),
  >"__pllefcql2tsprfinito1")
```

## What is and how does it work

### Parallel's backend

When the user enters

```
parallel:  gen x2 = x*x
```

`parallel` takes the command and writes something like this

```
  cap clear all
  cd ~
1 set seed 34815
  set memory 16777216b
  cap set maxvar 5000
  cap set matsize 400
2 local pll_instance 1
  local pll_id efcql2tspr
  capture {
  noisily {
3 use __pllefcql2tsprdataset if _efcql2tsprcut == 1
  gen x2 = x*x
  }
  }
4 save __pllefcql2tsprdta1, replace
  local result = _rc
  cd ~
5 mata: write_diagnosis(st_local("result"),
  >"__pllefcql2tsprfinito1")
```

```
  cap clear all
  cd ~
1 set seed 98327
  set memory 16777216b
  cap set maxvar 5000
  cap set matsize 400
2 local pll_instance 2
  local pll_id efcql2tspr
  capture {
  noisily {
3 use __pllefcql2tsprdataset if _efcql2tsprcut == 2
  gen x2 = x*x
  }
  }
4 save __pllefcql2tsprdta2, replace
  local result = _rc
  cd ~
5 mata: write_diagnosis(st_local("result"),
  >"__pllefcql2tsprfinito2")
```

Motivation

What is and how does it work

Benchmarks

Syntax and Usage

Concluding Remarks

## Benchmarks

### Simple example: Serial replace

**Serial fashion**

```
do mydofile.do
```

**Parallel fashion**

```
parallel do mydofile.do
```

Figure: mydofile.do

```
local size = _N
forval i=1/`size' {
        qui replace x = ///
            1/sqrt(2*`c(pi)')*exp(-(x^2/2)) in `i'
}
```

Table: 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

## Benchmarks

### Monte Carlo simulation (Windows Machine)

Serial fashion

```
do myexperiment.do
```

Parallel fashion

```
parallel do myexperiment.do, nodata
```

Figure: myexperiment.do

```
local num_of_intervals = 50
if length("‘pll_id’") == 0 {
    local start = 1
    local end = ‘num_of_intervals’
}
else {
    local ntot = floor(‘num_of_intervals’/$PLL_CLUSTERS)
    local start = (‘pll_instance’ - 1)*‘ntot’ + 1
    local end = (‘pll_instance’)*‘ntot’
    if ‘pll_instance’ == $PLL_CLUSTERS local end = 10
}
local reps 10000
forval i=‘start’/‘end’ {
    qui use census2, clear
    gen true_y = age
    gen z_factor = region
    sum z_factor, meanonly
    scalar zmu = r(mean)
    qui {
        gen y1 = .
        gen y2 = .
        local c = ‘i’
        set seed ‘c’
        simulate c=r(c) mu1=r(mu1) se_mu1 = r(se_mu1) ///
                mu2=r(mu2) se_mu2 = r(se_mu2), ///
                saving(cc‘i’, replace) nodots reps(‘reps’): ///
                mcsimul1, c(‘c’)
    }
}
```

Table: 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

## Benchmarks

### Monte Carlo simulation (Unix Machine)

#### Serial fashion

```
do myexperiment.do
```

#### Parallel fashion

```
parallel do myexperiment.do, nodata
```

Table: 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

## Benchmarks

Reshaping Administrative Data

### Serial fashion

```
reshape wide tipsolic rutemp opta derecho ngiros, ///
        i(id) j(time)
```

### Parallel fashion

```
parallel, by(id) :reshape wide tipsolic rutemp opta derecho ngiros, ///
        i(id) j(time)
```

Table: Reshaping wide a large database on a Linux Server (8 clusters)

| | 100,000 | 1,000,000 | 5,000,000 |
|---|---|---|---|
| CPU | 5.51 | 72.70 | 392.97 |
| Total | 2.33 | 17.46 | 86.44 |
| Setup | 0.00 | 0.00 | 0.00 |
| Compute | 1.83 | 12.42 | 57.93 |
| Finish | 0.50 | 5.04 | 28.51 |
| Ratio (compute) | 3.01 | 5.85 | 6.78 |
| Ratio (total) | 2.37 (29%) | 4.16 (52%) | 4.55 (57%) |

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

## 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 |
| | $\times$2.69 | $\times$1.48 | $\times$1.00 |
| 2,000 | 5.80s | 3.13s | 2.03s |
| | $\times$2.85 | $\times$1.54 | $\times$1.00 |
| 4,000 | 11.59s | 6.27s | 3.86s |
| | $\times$3.01 | $\times$1.62 | $\times$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
```

## Benchmarks

### Simulations with `parallel sim` (cont.)

| Problem size | Serial | 2 Clusters | 4 Clusters |
|---|---|---|---|
| 1000 | 2.19s | 1.18s | 0.73s |
| | $\times$3.01 | $\times$1.62 | $\times$1.00 |
| 2000 | 4.36s | 2.29s | 1.33s |
| | $\times$3.29 | $\times$1.73 | $\times$1.00 |
| 4000 | 8.69s | 4.53s | 2.55s |
| | $\times$3.40 | $\times$1.77 | $\times$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`

Motivation

What is and how does it work

Benchmarks

Syntax and Usage

Concluding Remarks

## Syntax and Usage

### Setup

**parallel setclusters** `# [, force]`

## Syntax and Usage

### Setup

```
parallel setclusters # [, force]
```

### By syntax

```
parallel [, by(varlist) programs mata seeds(string) randtype(random.org|datetime)
        processors(integer) nodata]: stata_cmd
```

## Syntax and Usage

### Setup

**parallel setclusters** # [, force]

### By syntax

**parallel** [, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]: *stata_cmd*

### Do syntax

**parallel do** *filename*
[, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]

## Syntax and Usage

### Setup

**parallel setclusters** `# [, force]`

### By syntax

**parallel** `[, by(`*varlist*`) programs mata seeds(`*string*`) randtype(`*random.org|datetime*`) processors(`*integer*`) nodata]:` *stata_cmd*

### Do syntax

**parallel do** *filename* `[, by(`*varlist*`) programs mata seeds(`*string*`) randtype(`*random.org|datetime*`) processors(`*integer*`) nodata]`

### Bootstrap syntax

**parallel bs** `[, by(`*varlist*`) programs mata seeds(`*string*`) randtype(`*random.org|datetime*`) processors(`*integer*`) nodata]:` *stata_cmd*

## Syntax and Usage

### Setup

**parallel setclusters** # [, force]

### By syntax

**parallel** [, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]: *stata_cmd*

### Do syntax

**parallel do** *filename*
[, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]

### Bootstrap syntax

**parallel bs** [, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]: *stata_cmd*

### Simulations syntax

**parallel sim** [, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]: *stata_cmd*

## Syntax and Usage

### Setup

**parallel setclusters** # [, force]

### By syntax

**parallel** [, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]: *stata_cmd*

### Do syntax

**parallel do** *filename*
[, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]

### Bootstrap syntax

**parallel bs** [, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]: *stata_cmd*

### Simulations syntax

**parallel sim** [, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]: *stata_cmd*

### Append syntax

**parallel append** [, by(*varlist*) programs mata seeds(*string*) randtype(*random.org*|*datetime*)
processors(*integer*) nodata]: *stata_cmd*

## Syntax and Usage

### Recommendations on its usage

```
parallel suits ...
```

- Monte-Carlo simulation.

## Syntax and Usage

### Recommendations on its usage

`parallel suits ...`

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).

## Syntax and Usage

### Recommendations on its usage

`parallel suits ...`

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).
- Bootstrapping/Jackknife.

## Syntax and Usage

### Recommendations on its usage

```
parallel suits ...
```

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).
- Bootstrapping/Jackknife.
- Simulations in general.

## Syntax and Usage

### Recommendations on its usage

`parallel suits ...`

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).
- Bootstrapping/Jackknife.
- Simulations in general.

`parallel doesn't suit ...`

- (already) fast commands.

## Syntax and Usage
### Recommendations on its usage

`parallel` suits ...

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).
- Bootstrapping/Jackknife.
- Simulations in general.

`parallel` doesn't suit ...

- (already) fast commands.
- Regressions, ARIMA, etc.

## Syntax and Usage

### Recommendations on its usage

`parallel suits ...`

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).
- Bootstrapping/Jackknife.
- Simulations in general.

`parallel doesn't suit ...`

- (already) fast commands.
- Regressions, ARIMA, etc.
- Linear Algebra.

## Syntax and Usage

### Recommendations on its usage

`parallel suits ...`

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).
- Bootstrapping/Jackknife.
- Simulations in general.

`parallel doesn't suit ...`

- (already) fast commands.
- Regressions, ARIMA, etc.
- Linear Algebra.
- Whatever StataMP does better.

## Syntax and Usage

### Recommendations on its usage

`parallel` suits ...

- Monte-Carlo simulation.
- Extensive nested control flow (loops, while, ifs, etc.).
- Bootstrapping/Jackknife.
- 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.

## Concluding Remarks

- Brings parallel computing to many more commands than StataMP

## 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.

## 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 de-serialized, it is possible to reach near to linear scale speedups.

## 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 de-serialized, it is possible to reach near to linear scale speedups.
- `parallel` can be incorporated into other user commands as an optional speedup.

## 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 de-serialized, it is possible to reach near to linear scale speedups.
- `parallel` can be incorporated into other user commands as an optional speedup.
- Caveat: Has not been tested on Stata 15.

## 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 de-serialized, it is possible to reach near to linear scale speedups.
- `parallel` can be incorporated into other user commands as an optional speedup.
- Caveat: Has not been tested on Stata 15.
- Contribute, find help, and report bugs at `http://github.com/gvegayon/parallel`

## 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 de-serialized, it is possible to reach near to linear scale speedups.
- `parallel` can be incorporated into other user commands as an optional speedup.
- Caveat: Has not been tested on Stata 15.
- 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