

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

George G. Vega¹ Brian Quistorff²

¹University of Southern California
g.vegayon@gmail.com

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

Stata Conference Baltimore
July 27–28, 2017

Thanks to Damian C. Clarke, Félix Villatoro and Eduardo Fajnzylber, Tomás Rau, Eric Melse, Valentina Moscoso, the Research team of the Chilean Pension Supervisor and several 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, 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

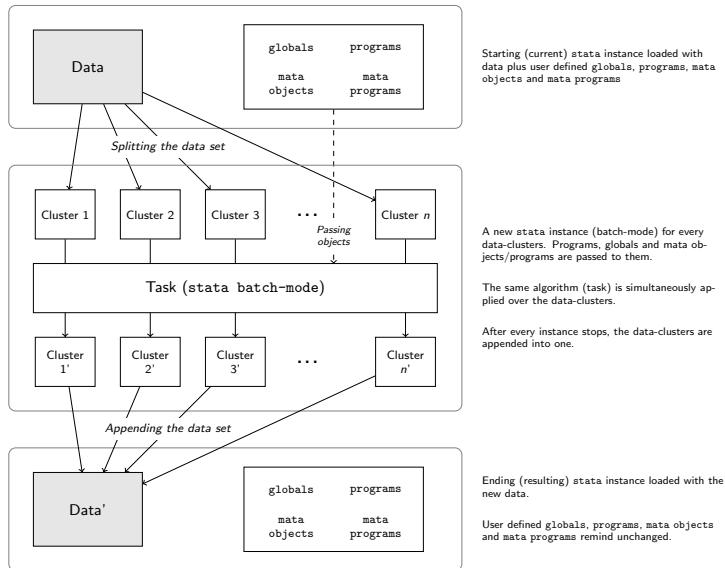
What is and how does it work

What is?

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



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 n = _N
```

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 n = _N
  }
  }
4 save __pllefcql2tsprdata1, 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 n = _N
  }
  }
4 save __pllefcql2tsprdata2, 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("`pl1.id'") == 0 {
    local start = 1
    local end = "num_of_intervals"
}
else {
    local ntot = floor("num_of_intervals"/`$PL1.CLUSTERS')
    local start = ("pl1.instance" - 1)*'ntot' + 1
    local end = ("pl1.instance")*'ntot'
    if 'pl1.instance' == '$PL1.CLUSTERS' local end = 10
}
local reps 10000
forval i="`start'/'`end'" {
    qui use census2, clear
    gen trunq = age
    gen x_factor = region
    sum x_factor, meanonly
    scalar ms = r(msum)
    qui {
        gen y1 = .
        gen y2 = .
        local c = 'i'
        set seed `c'
        simulate c=r(c) msi=r(ms1) se_msi = r(se_msi) ///
            ms2=r(ms2) se_ms2 = r(se_ms2), ///
            saving(cc'1', replace) nodata repse("reps"): ///
            montemsi, 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

Motivation

What is and how does it work

Benchmarks

Syntax and Usage

Concluding Remarks

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

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
- ▶ 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. Vega¹ Brian Quistorff²

¹University of Southern California
g.vegayon@gmail.com

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

Stata Conference Baltimore
July 27–28, 2017