Parallel Computing in Matlab and Stata

And how to use the Linux computer cluster

Daniel Haanwinckel

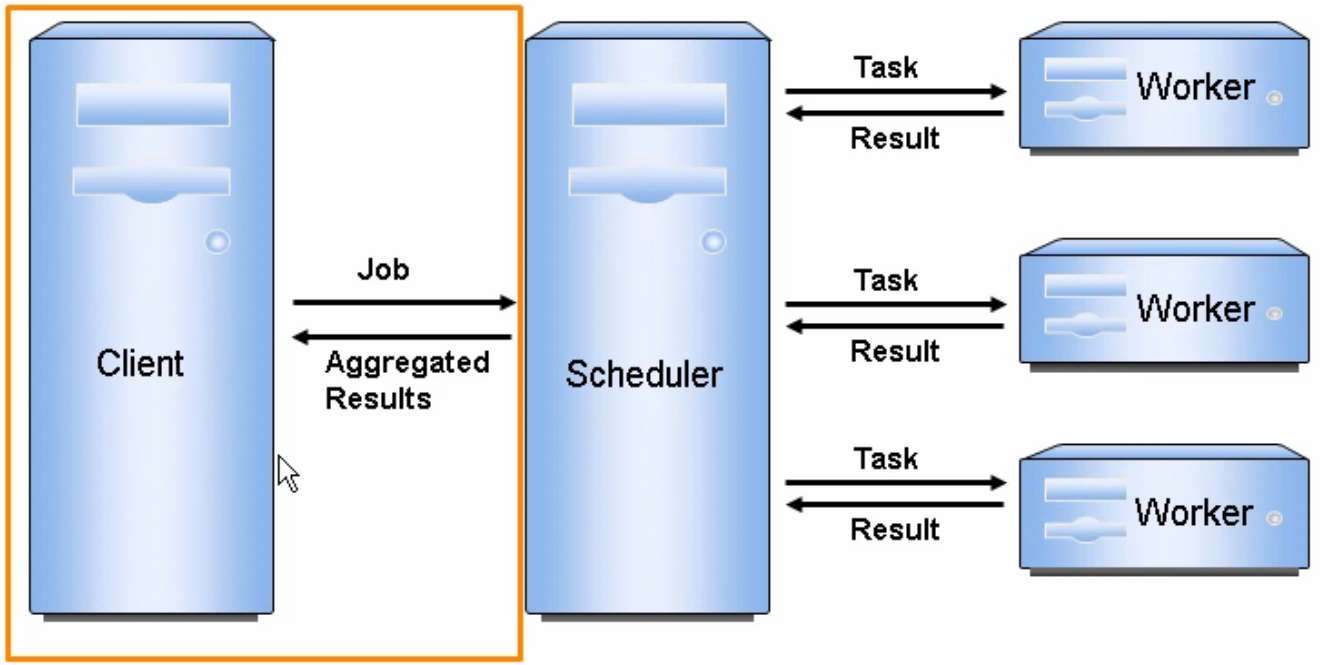
Productivity Seminar

April 2014

Introduction

Parallel Computing Concepts

- Client: Your computer
- Scheduler: The computer who sends the tasks, get the results and send back to you
- Workers: Each guy doing a piece of your calculation



Taken from Matlab's video tutorial

Parallel Computing Concepts

- When is parallel computing good?
  - You can divide your code into "tasks" that can be calculated independently
  - Ideally, no communication between tasks
  - Using CPU's: A "small" number of time-consuming tasks
  - Using GPU's: A large number (1,000,000) of very simple tasks (hardly the case in Econ)
- When is it not good?
  - Sending data to each worker and getting the results back takes time ("overhead")
  - If you have too many simple tasks or lots of data transfer between scheduler and workers, parallel computing can take more time than serial computing

Parallel Computing Concepts

- "Embarrassingly parallel" examples in Economics:
  - Monte Carlo simulations/bootstrap
  - Numerical derivatives or search methods in the minimization of multivariate problems
    - Good when calculating the objective function takes a long time – estimating/calibrating parameters of a complicated model

Parallel Computing in Matlab

- matlabpool and parfor: Matlab makes it easy for you
  - Assuming you're running it on a computer with many cores available and have configured it (simple to do)
  - You need the Parallel Computing Toolbox

```
%Setup:
slots = 4;
matlabpool(slots); %OR: matlabpool open 4

parameters = [10;20;50;100];
nParameters = size(parameters,1);
results = zeros(nParameters,1);
parfor i = 1:nParameters
   results(i) = calculateMyStuff(parameters(i));
end
display(results);

matlabpool close
```

Parallel Computing in Matlab

- For optimization programs such as fmincon: even easier.
  - Matlab automatically parallelizes the work when possible (e.g. calculating Jacobians; polling in pattern search)

```
%Setup
matlabpool open 4
x0 = 0.5;

opt = optimset('UseParallel','Always');
sol1 = fmincon(myObjectiveFun, x0, ...
   [],[],[],[],0,1,[],opt);

psopt = psoptimset('UseParallel','Always',...
         'CompletePoll', 'on',...
         'Vectorized', 'off');
sol2 = patternsearch(myObjectiveFun, x0, ...
   [],[],[],[],0,1,[],psopt);

matlabpool close
```

Using the Computer Cluster

- Prerequisite: you should be able to remotely access the department's computers
  - See the material sent after Waldo's Productivity Seminar presentation last year
- Steps:
  1. Setup your code so it knows how many processors to use
  2. Write a shell file with the job you want the cluster to do
  3. Put it in the folder where you want the program to run
  4. Log in to one of the Linux computers and go to that folder
  5. Use `qsub` to submit the job
  6. Use `qstat` and `qdel` to monitor or kill your job

Using the Computer Cluster

1. It's easier to work if your code can automatically detect the number of cores available

- Matlab: start your code with

```
slots=getenv('NSLOTS');
feature('numThreads', str2num(slots));
matlabpool(slots);
```

- Stata: start your code with

```
args ncores
set processors `ncores'
```

And then add $NSLOTS as an input when calling stata-mp in the shell file (see below)

Using the Computer Cluster

2. The shell file is usually a one-line file (say, job.sh) with the command line call to Matlab or Stata.

- Matlab:

  ```
  matlab -nodisplay -nodesktop < myMatlabCode.m > myOutput.out
  ```

- Stata:

  ```
  stata-mp -b do myStataCode.do $NSLOTS
  ```

Using the Computer Cluster

3. and **4.** See Waldo's stuff.

5. When logged to the Linux computer, and in the folder you want your program to run, type something like:

```
qsub -l h_rt=672:00:00 -pe smp 12 job.sh
```

- The -l option is the maximum time you allow the code to run. The default is 5 days (if it takes more than that, the cluster will kill it). The maximum is 28 days (672 hours).
- The -pe option defines how many cores you want – in the example above, 12. You can specify up to 32 cores, but Matlab won't use more than 12 unless you do some more advanced stuff.

Using the Computer Cluster

- If the cluster is crowded, it might be harder to get a big number of cores because -pe requires all cores to be in the same machine.
- You can also use MPI – that is, use cores scattered across many machines. See the documentation below.

6. Typing `qstat` shows you your current jobs and their status, as well as their ID. Typing `qdel <jobID>` will kill the job with the specified ID.

- Documentation:
  - https://eml.berkeley.edu/563: Basic guide on how to use the computer cluster.
  - http://emlab.berkeley.edu/scf/eml-parallel.pdf: Notes on parallel computing in the cluster, including more advanced stuff.
  - https://eml.berkeley.edu/cgi-bin/help.cgi: EML help page – lots of good information.

Parallel Computing in Stata

- Good news: Stata MP does everything for you
  - The usual Stata commands reprogrammed to make use of multiple cores.
  - Most commands display performance improvements when using multiple cores, though it's not linear.
  - According to Stata, gains are larger if you have lots of observations or panel data.
  - See http://www.stata.com/statamp/statamp.pdf for a thorough description of performance improvements.

Parallel Computing in Stata

- Bad news: It's not always pretty
  - Parallelization is always "fine-grained": Stata will send every simulation in the bootstrap to all workers sequentially, instead of sending one simulation to each worker.
  - Workaround: manually execute each simulation in a different instance of Stata
    - Send each simulation as a job, save file with the results, and have another program interpret the saved results

Some Comments

- Often, there are many ways to parallelize your code:
  - Should I parallelize the loop I have within the objective function and run the optimization procedure normally?
  - Or should I calculate the objective function normally and parallelize fmincon?
  - No general answer. You should experiment and see which way is more efficient (use tic/toc).
- You should also measure what you get when not using parallel: overhead costs can be high.

Some Comments

- Pay attention to Matlab complaints about your code (red underlines). It will help you organize your variables in a way that reduces overhead costs, by proper indexing/slicing.
- Do not let parallel computation be an excuse to write inefficient code.
  - Other programming techniques can have more sizable effects in reducing computational cost than parallel computing.
  - Examples: matrix calculations instead of for loops in Matlab; using an adequate minimization algorithm; setting the correct optimization parameters, such as stop conditions; better algorithms for your specific application.