Instructions on Running the Code on Linux

Xin Tang
International Monetary Fund

August 7, 2019

This note provides step-by-step instruction on how to run the code on Stony Brook Seawulf and Ubuntu. We start with Seawulf. In what follows, everything starts with a $ is what you type, while those preceded by > are what the terminal prints.

1. Log in the login node of the server.

2. Load first the `shared` and then the `git` modules.

```
$ module load shared
$ module load git
```

3. Clone the GitHub repository that hosts the source code.

```
$ git clone https://github.com/zjutangxin/afv_replication
> Cloning into 'afv_replication'...
 remote: Enumerating objects: 104, done.
 remote: Counting objects: 100% (104/104), done.
 remote: Compressing objects: 100% (67/67), done.
 remote: Total 104 (delta 58), reused 74 (delta 34), pack-reused 0
 Receiving objects: 100% (104/104), 611.89 KiB | 0 bytes/s, done.
 Resolving deltas: 100% (58/58), done.
 Checking out files: 100% (17/17), done.
```

Notice that this clones the `master` branch of the repository, which solves the model in serial. The paralleled version is hosted in another branch called `mpi_debt`. To clone that branch, first create a local git branch `mpi_debt` (could be any name).

```
$ git branch mpi_debt
```

Then switch to that branch.

```
$ git checkout mpi_debt
> Switched to branch 'mpi_debt'
```

Once on the local `mpi_debt` branch, clone the remote branch.

```
[xintang@login1 afv_replication]$ git pull origin mpi_debt
> From https://github.com/zjutangxin/afv_replication
  * branch            mpi_debt   -> FETCH_HEAD
 Updating 3f369e4..55c3b62
 Fast-forward
  compile.sh          |  19 +++++++++++++++----
  src/debt_main.f90   | 204 +++++++++++++++++++++++++++++++++++
  src/mod_global.f90  |   1 +
  src/mod_param.f90   |   5 ++++-
  src/solve_system.f90 | 116 +++++++++++++++++++++++++++++++++++
  5 files changed, 300 insertions(+), 45 deletions(-)
```

Now you have all the files ready to go. I find this branching feature extremely helpful in developing code, which allows you to switch back and forth between different versions of the code without messing things up.

4. Submit an interactive job to `Slurm`, requesting 40 cores.

```
$ srun -n 40 -p short-40core --pty bash
```

5. Once on the debugging node, load the modules for `gcc` (the compiler), `git` (version control), `LAPACK` and `BLAS` (linear algebra), and `OpenMPI` (parallel computing).

```
$ module load lapack/gcc/64/3.7.0 blas/gcc/64/3.7.0 git gcc openmpi
```

6. A couple of working directories need to be created initially. Specifically, the `./obj/` for intermediate files and `./results/` for saving results.

```
$ mkdir obj results
```

7. To avoid having to compile `MINPACK` everytime, I first compile it to an relocatable object file, and later just link it.

```
$ gfortran -c ./lib_src/minpack.f90 -Ofast -march=native -o ./obj/minpack.o
```

8. Now compile the program in parallel. The slashes are used to break the line for a very long command.

```
[xintang@dn025 afv_replication]$ mpif90 ./src/mod_param.f90 ./src/mod_global.f90 \
    ./src/mod_routines.f90 ./src/solve_system.f90 ./src/debt_main.f90 \
    ./obj/minpack.o /gpfs/software/lapack/lapack-3.7.0/liblapack.a \
    /gpfs/software/blas/BLAS-3.7.0/libblas.a \
    -o debt_main_par.out -Ofast -march=native -Wall
> /usr/bin/ld: warning: libgfortran.so.3, needed by \
    /usr/mpi/gcc/openmpi-3.1.1rc1/lib64/libmpi_usempi.so, \
    may conflict with libgfortran.so.5
```

The warning message is because the linker (`ld`) complains about the possible mismatch between the versions of dynamically linked libraries used by `gfortran` and `OpenMPI`. We are not using any fancy feature here, so just ignore it and code will run just fine.

9. Finally, let us launch the program with 40 cores!

```
[xintang@dn025 afv_replication]$ mpirun -np 40 ./debt_main_par.out
```

You will see some warning messages about the system environmental variables initially. Again, they will do no harm, just ignore them.

```
> ibv_exp_query_device: invalid comp_mask !!! \
    (comp_mask = 0xffffffffffffffff valid_mask = 0x1)
[dn025][[12049,1],26][btl_openib_component.c:1670:init_one_device] \
    error obtaining device attributes for mlx4_0 errno says Invalid argument
--------------------------------------------------------------------------
WARNING: There was an error initializing an OpenFabrics device.

  Local host:   dn025
  Local device: mlx4_0
--------------------------------------------------------------------------
ibv_exp_query_device: invalid comp_mask !!! \
    (comp_mask = 0xffffffffffffffff valid_mask = 0x1)
[dn025][[12049,1],13][btl_openib_component.c:1670:init_one_device] \
    error obtaining device attributes for mlx4_0 errno says Invalid argument
[1565186835.756078] [dn025:81591:0] mxm.c:196  MXM  WARN  \
    The 'ulimit -s' on the system is set to 'unlimited'.
[1565186835.756551] [dn025:81599:0] mxm.c:196  MXM  WARN  \
    The 'ulimit -s' on the system is set to 'unlimited'.
[1565186835.757112] [dn025:81569:0] mxm.c:196  MXM  WARN  \
    The 'ulimit -s' on the system is set to 'unlimited'.
...
```

After these warning messages, you should start seeing the normal printout from the program. For now, the code is set to replicate Figures 9 and 10 in the paper. So we are solving the government problem with 200 periods. Then using Autarky as initial starting point, simulate forward 100 periods to reach the steady state of open economy. The messages on the terminal are organized as follows. At first, the backward induction part when solving the government problem.

```
 POLICY ITER     1   68.09246174870   68.09246174870    0.00000000000
 t =   1   SS Debt 1 =    0.007846   SS Debt 2 =    0.007846   Error =     0.000000
  Land Price =       0.189198      0.189198

 POLICY ITER     2   26.38190931408   26.38190931408    0.00000000000
 t =   2   SS Debt 1 =    0.014613   SS Debt 2 =    0.014613   Error =     0.000000
  Land Price =       0.368034      0.368034
  ...
```

```
POLICY ITER   198    0.00370471053    0.00404022301    0.00000000000
t = 198   SS Debt 1 =   0.461559   SS Debt 2 =   0.461593  Error =     0.000000
 Land Price =       3.506217      3.506217

POLICY ITER   199    0.00454127798    0.00486093482    0.00000000000
t = 199   SS Debt 1 =   0.461562   SS Debt 2 =   0.461564  Error =     0.000000
 Land Price =       3.506211      3.506211
```

Then it shows the convergence of the distribution of entrepreneurs for the initial steady state.

```
  Dist Iter =   250  8.201552E-05
  Dist Iter =   500  1.261716E-07
  Dist Iter =   750  1.982667E-10
  Dist Iter =  1000  3.348815E-13
Infinite Horizon
Share top 1%,    Share top 10%,    Gini
     5.830364     24.616527     26.915247
```

And finally, the transition path from autarky to open economy

```
    1 0.2678 0.2678 0.0346 0.0346 3.2642 3.2642   5.6687  24.1195  25.6054
    2 0.2890 0.2890 0.0339 0.0339 3.2872 3.2872   5.6944  24.2236  25.7124
...
    99 0.4616 0.4616 0.0238 0.0238 3.5062 3.5062   5.7730  24.8920  26.7379
    100 0.4616 0.4616 0.0238 0.0238 3.5062 3.5062   5.7728  24.8919  26.7377

    Debt-Output Ratio =   0.46156168761281585       0.46156368575488221

    Time =     450.5362
```

In total, it takes only 7.5 minutes to run!

10. The results are exported as text files under `./results/`. The policy functions of the government are saved in `DebPol1.txt` and `DebPol2.txt`. Element i, j there reports the for government 1 (or 2), if $(B_1, B_2) = (b_i, b_j)$, what would be the debt choice by government 1 (or 2), where b_i is the bond grid, which is set to 20 points between 0 and 0.8. These results can be used to produce Figure N2 in the Appendix.

    Files `Debt1_seqa.txt` shows the evolution of debt when the economy opens up. The first ith elements of the ith row shows if the transition path if the time horizon is $t = i$. So the blue dotted curves in Figure 9 are plotted respectively using the first 200, 100, and 50 elements of the 200th, 100th, and 50th rows.

    The blue line in Figure 10 can be produced using the second column printed on the terminal:

```
    1 0.2678 0.2678 0.0346 0.0346 3.2642 3.2642   5.6687  24.1195  25.6054
    2 0.2890 0.2890 0.0339 0.0339 3.2872 3.2872   5.6944  24.2236  25.7124
...
    99 0.4616 0.4616 0.0238 0.0238 3.5062 3.5062   5.7730  24.8920  26.7379
    100 0.4616 0.4616 0.0238 0.0238 3.5062 3.5062   5.7728  24.8919  26.7377
```

11. The code is paralleled where the Nash Equilibrium is solved. When solving for the Nash Equilbrium, for each combination of current debt $(B_{1,t}, B_{2,t})$, the government calculates the potential welfare of each possible combination of debt choice $(B_{1,t+1}, B_{2,t+1})$, where $(B_{1,t+1}, B_{2,t+1})$ belongs to a denser grids. At this moment, the number of grids is set to `maxgrid = 200` declared in `mod_param.f90`. Each core computes a fraction of the 200 points. For instance, in the case of 4 cores, cores 1 to 4 computes $B_{1,t+1} = [b_1, \cdots, b_{50}], B_{2,t+1} = [b_1, \cdots, b_{200}]$ to $B_{1,t+1} = [b_{151}, \cdots, b_{200}], B_{2,t+1} = [b_1, \cdots, b_{200}]$. So when setting the number of nodes `np` used, please make sure `np` can always be divided by `maxgrid`.

12. To run the code on Ubuntu is much simpler. First make sure that all the libraries are correctly installed.

```
$ sudo apt-get install build-essential
  sudo apt-get install gfortran
  sudo apt-get install git
  sudo apt-get install libblas-dev
  sudo apt-get install libatlas-base-dev
  sudo apt-get install liblapack-dev
  sudo apt-get install openmpi-bin
  sudo apt-get install openmpi-doc
  sudo apt-get install libopenmpi-dev
```

Similarly cloning the `mpi_debt` branch of the GitHub repository, create the working directories, compile `MINPACK` to relocatable binary format, and then compile the program using

```
$ mpif90 ./src/mod_param.f90 ./src/mod_global.f90 ./src/mod_routines.f90 \
     ./src/solve_system.f90 ./src/debt_main.f90 \
     ./obj/minpack.o -llapack -latlas -lblas \
     -o debt_main_par.out -Ofast -march=native -Wall
```

Notice that in the above command, `LAPACK`, `ATLAS` and `BLAS` are linked dynamically. On a personal computer, it is likely that you have the authorization to run bash script. So the above command is conveniently stored in the bash script `compile.sh`. Each time when the program needs to be compiled, simply invoke the bash script

```
$ ./compile.sh
```

and the system will do the work. The same command is used to run the program.

```
$ mpirun -np 4 ./debt_main_par.out
```

On my desktop with 4 cores, it takes about 20 minutes to run the code.