Comments on *"Parallelization of Matlab codes under Windows platform for Bayesian estimation: a Dynare application"* (I. Azzin, R. Girardi and M. Ratto)

Sébastien Villemot

PSE-CEPREMAP

Third Dynare Conference
September 10, 2007

Summary

- In the foreseeable future, technical improvement of CPUs will be in terms of parallel computing units, not clock speed
  $\Rightarrow$ Need to introduce parallelization in algorithms and software to take advantage of it

- Parallelization can be introduced in Dynare at a relatively low programming cost (e.g. in Metropolis-Hastings)

Processes versus Threads (1/2)

- Parallelization can be implemented by two means: processes or threads
- Processes have the following characteristics:
  - Each process is a completely independant execution unit
  - Each process has its own memory space and ressources (opened files, network connexions, ...)
  - Processes communicate with each other via inter-process communication mechanisms (IPC) (for example through network connexions)
  - Example: one process for web browser, another for word processing, yet another for music player...
- Processes are the “heavy” way of doing parallelization

Processes versus Threads (2/2)

- On the contrary, threads have the following characteristics:
  - Several threads can be spawned by a single process
  - The threads execute in parallel the same program code (that of the parent process)
  - Threads share the same memory space, and the same ressources
  - Each thread maintains minimum state information: line of code being run, register values, stack
  - Threads can communicate directly via memory space
- Threads are also called “light-weight processes”
- Threads have (almost) no memory cost, and consume less ressources during their creation, but must run on the same host

The implementation of Azzin, Girardi & Ratto (1/2)

- Metropolis-Hastings chains are completely independant from each other $\Rightarrow$ easy to parallelize
- Implementation uses processes rather than threads: an independant Matlab session is run in parallel for each chain
- Processes are launched with a batch command file and a Microsoft command-line tool (`psexec`)
- Inter-process communication implemented through files:
  - One input file shared by all processes (contains posterior kernel function and some other shared data)
  - One output file per process
  - Time synchronization done through dummy files

The implementation of Azzin, Girardi & Ratto (1/2)

- Advantages:
  - simple source code, only few modifications to official Dynare code
  - easy to adapt to a cluster of computers (through a network-shared filesystem)
- Inconvenients:
  - Windows only, uses third party (although free) software
  - suboptimal in memory consumption (and possibly speed), at least in single-host environment
- Easy improvement: create a standalone DLL module in C/C++ which directly creates the other Matlab sessions (through process "forking"). Would be more portable accross platforms (notably Linux)
- But implementing everything in C/C++ would mean rewriting Kalman filter and posterior kernel function in C/C++...

Alternative solution: using threads (1/2)

- Thread support non-existent up to Matlab version 7.3
- Since Matlab 7.4 (R2007a):
  - Some Matlab primitives are implemented with multi-threading (e.g. matrix multiplication)
  - Needs to be activated in Preferences→General→Multithreading
  - Leads to a speed-up on machines with a multi-core CPU (or even a Pentium 4 with Hyper-threading)
- But custom multi-threading cannot be implemented at the level of a Matlab M-file
  $\Rightarrow$ need to create a DLL module in C/C++

Alternative solution: using threads (2/2)

- Threads are easy to create from C/C++ code: native support under Windows and Linux (POSIX threads)
- Each thread in the DLL would call the posterior kernel function (written in Matlab code)
- Problem: current Dynare implementation of posterior kernel function is not thread-safe: it modifies the value of global variables
  $\Rightarrow$ would lead to write conflicts in multi-threading
  $\Rightarrow$ we first have to remove global variables from Dynare code
- Advantages over current implementation would be:
  - less memory consumption: no need to replicate input information (since the memory space is shared), no creation of processes
  - possible (though limited) speed-up: creation of threads is very fast
- Inconvenient: doesn't work for a cluster of machines

What can be parallelized in Dynare ?

- Besides Metropolis chains, authors mention the graphics
- Straight-forward parallelizations could be added in Monte-Carlo methods, as in:
  - IRFs in `stoch_simul`,
  - `forecast`,
  - and also in BVAR and BVAR-DSGE routines.
- More challenging: in the posterior mode computation (through optimization)
  - The directions along which are computed the numerical derivatives could be divided accross threads (derivation with respect to a given direction is independant of other directions)
  - Same problem than before: current code is not thread-safe

A note on Hyper-Threading

- Some Pentium 4 (P4) incoporate the *Hyper-Threading* (HT) technology
- These CPUs are single core, but are optimized to take advantage of idle parts of the core
- Two threads of execution are handled by the CPU in parallel: when the first thread doesn't use some resource of the core, this resource is given to the second thread for parallel execution
- The operating system treats a P4 with HT as two "logical" processors (and some dual-processor machines appear having 4 processors)
- Intel claims a 20% to 40% speed improvement over comparable non-HT processor $\Rightarrow$ result replicated by the paper
- This speed-up would not have been present with a non-HT P4