A Multi-Threaded Economy

Ernesto Carrella
Computational Social Science,
George Mason University

May 14, 2012

Abstract

Multithreading allows us to model naturally simultaneous actions . I use the method to model two simple supply chain agent-based models with synchronized activation regimes. The model allocates CPU power as an economic resource through decentralized agents. Agents, individually trying to reduce their delays in the supply chain, obtain global efficiency. I discuss common problems of using multithreading models and possible solutions.

Keywords: Agent-Based Economics; supply chain

JEL Classification Numbers: Y90 (Miscellaneous Categories -Other -Other).

1 Economics as Computation

Can we use computation as a metaphor for social systems? Operating systems have to allocate CPU and memory to many competing applications. Allocation under scarcity is the defining problem of economics. Two related research topics are whether computer science can learn from economics (Huberman, 1998; Huberman & Hogg, 1995) and viceversa (Axtell, 2003).

I use multi-threading programming to provide one such metaphors. I study a simple supply chain that needs to synchronize its production and allocate workers efficiently. The system evolves completely in parallel.

At its core my model deals with synchronization and time-management. Firms' actions and incentives happen in real-time. Time and synchronizations in economics are modeled indirectly, as stag-hunt games, as discrete time periods and so on. Agent Based Modeling allows us to deal with time explicitly.

There are two ways of using multi-threading in agent based model. First, by implementing of a traditional model through multi-threading in order to make it run faster, simulate more agents, etc. Alternatively one can use simultaneous threads to simulate simultaneous agents. The model I present is of the second kind.

Multi-threaded modeling provides two advantages over more traditional ABM. It allows the model to have synchronized activation regimes. Agents act together rather than in turn. This is an alternative to the lock-step turn where each agent acts in vacuum and then the model aggregates results.

The second advantage is the ability to model "procedural" externalities. With that I mean the conflicts that arise by multiple players acting at the same time. If one agent is using a resource, say a place in queue, it stops others from doing the same. Having multiple threads running allows for this to be simulated naturally.

2 A simple supply-chain economy

2.1 The Model

Imagine an economy formed by functionally different firms. Each firm has its own blueprint requiring specific inputs to produce a specific output. This setup is a common one. In classical economics it's the input-output system (Leontief, 1966), in heterodox economics is the Sraffian commodity system (Sraffa, 1975) and we can trace its origin all the way to the *Tableau economique* (Quesnay, 1852).

A firm's input is another's output. Firms are mutually dependent from one another. Economic efficiency is achieved when profitable firms obtain needed inputs to sell their outputs.

But efficiency also requires that inputs are supplied in a timely fashion. A delay in production of one firm cascades throughout the supply chain. This is clear externality. To maintain efficiency there needs to be incentives that punish delays and allocate resources to key industries.

Our agents will be firms, our system is the supply chain. Each firm is heterogeneous regarding inputs, outputs and time needed to complete their work. Firms can cut their production time by hiring more workers. Firms hire workers when they are late in delivering their output; firms fire workers when their inputs are late.

The focus on time delays rather than prices and profits needs to be justified. In this model all agents act simultaneously and in parallel. Usual demand-supply assume the existence of a period, a market day, where what is currently available is exchanged. At the core this is the assumption that markets are perfectly synchronized so that all production is completed at the same instant.

In our case the synchronization of production is the *explanandum*. We can imagine firms contracting with one another for the delivery of goods at a specific time with fines for delays. Ideally we would like to model these contracts explicitly. Even so, I assumed delay

minimization to be a good place-holder.

2.2 The Implementation

2.2.1 Agents' Roles

Each firm is an independent CPU thread. This thread represents firm management. Its functions are to gather inputs, start production, sell output and hire workers (see figure 1). Transforming input into output requires work. Firms hire worker to do it.

Each worker in the firm is an independent thread. These threads work as long as there are inputs and jobs to be done.

The amount of work to do varies among firms. I assume work to be quantifiable and split into "units". Each work-unit is a processor instruction to be completed; the default being counting 20 milliseconds. The more workers are available the faster these tasks are accomplished.

Firms only care about delays. Output is placed in the market awaiting buyers. Input delay guide buyers' labor decisions. A firm hires a worker when its buyers have been waiting for more than $\bar{d}$ milliseconds. A firm fires a worker when it waited its inputs for more than $\underline{\mathrm{d}}$ milliseconds A firm also fires a worker when more than $\bar{o}$ of its output has not been bought.

2.2.2 Java Implementation

The model was simulated in pure Java 7. The code is freely available on GitHub. Firm is a class extending *Thread*. Each firm is initialized with its required inputs, its output (the Sraffian "blueprint") and the units of work required for completion.

Markets are *BlockingQueue*s . Producers fill queues, buyers empty them. Firms gathering inputs call the *acquire()* method potentially pausing until the input becomes available.

When all inputs are collected the firm activates its workers. This is accomplished by creating a *FixedThreadPool* of size equal to the

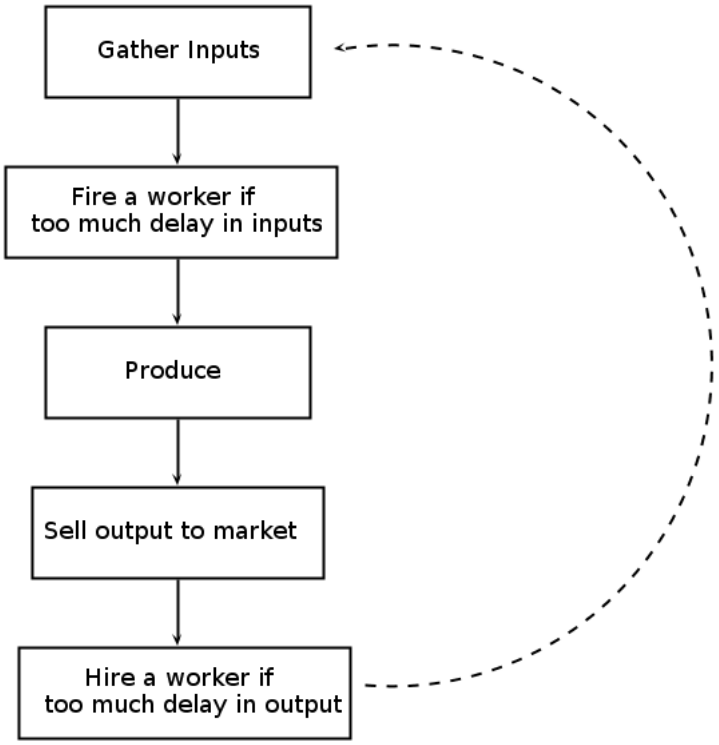

Figure 1: The flow of a firm agent operation. The "produce" step activates the worker threads

employees hired. A unit of work is a class extending *Runnable*. The firm feeds all the units of work to the workers and waits for the jobs to be completed. A *Semaphore* allows the firm to monitor how many jobs are left. Once all is done the firm puts its output in the appropriate BlockingQueue and starts over.

In some runs (see next section) workers are also consumers. If so each employed worker buy one unit of good every 3 seconds. The consumption routine runs, of course, on a parallel thread.

3 Sample Runs

3.1 Subsistence Sraffian Circle

Imagine a very simple economy as in figure 2. There are only four firms. You can imagine, as in Sraffa, that "firm" really represent a whole industry sector and inputs include workers' subsistence. Each firm requires 10 units of one input to produce 10 units of one output. Because the economy is in a circle as long as each firm is open the process can go on ad libitum.

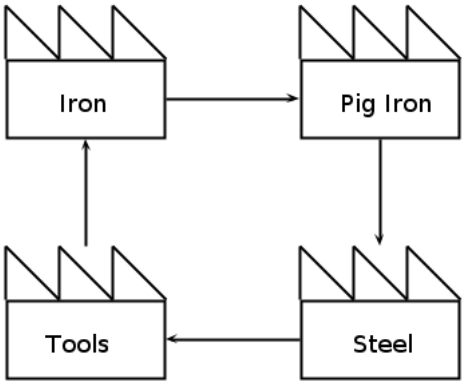

Figure 2: A simple subsistence economy. It takes 10 tools to produce 10 iron ores, 10 iron ores to produce 10 pig iron and so on.

Now assume some firms jobs are harder than others. In particular assume the Iron Mines require 10 tools and 300 work units to produce 10 iron. Iron Smelters require 10 iron and 600 work units to produce 10 pig iron. Steel mills require 10 pig iron and 900 work units to produce 10 steel. Equipment makers require 10 steel and 1200 work units to produce 10 tools.

Any distribution of workers where each firm hires at least 1 will allow the system to self-replicate, but not all such distributions are efficient. It is clear that the firms with higher job requirements will be the bottleneck of the system. For efficiency we need workers to

be allocated in larger amounts to the steel mill and the equipment makers.

The simple rules of our model are enough to achieve efficiency. The economy reaches an efficient proportion of workers after about 100 seconds of simulation (see figure 4). Each firm was responding to delays in their input and output markets depicted in figure 3. We can see from the graphs how delay in one market propagates through the supply chain.

The usual result of running this agent based model is that workers get allocated in proportion 3-6-9-12. This makes sense given our assumptions about work units. Speed of convergence varies from run to run but the multiple spikes we see in the delay charts are common in every run.

3.2 Worker driven growth

Model worker compensation explicitly. The new production structure of the economy is figure 5. The iron mines now produce iron with no input except labor. Each worker is paid with a coupon to buy one unit of tools. Workers that don't work don't eat.

Labor participation now is important for the whole system. Workers fill two functions: they work and they consume tools.

The main feedback loop in this model is between worker demand and production time. The firm supplying tools wants to limit the delay of workers consumption. To do so it must hire more workers which in turn force other firms in the supply chain to hire. But the more the firms hire the more the demand for tools.

The delay charts now are more jagged (see figure 6). The origin of the spikes is that all workers ask for tools at the same time (every 3 seconds). Since growth is consumption-driven is fair to assume that delays start in the tools market and percolates up. But at times tool

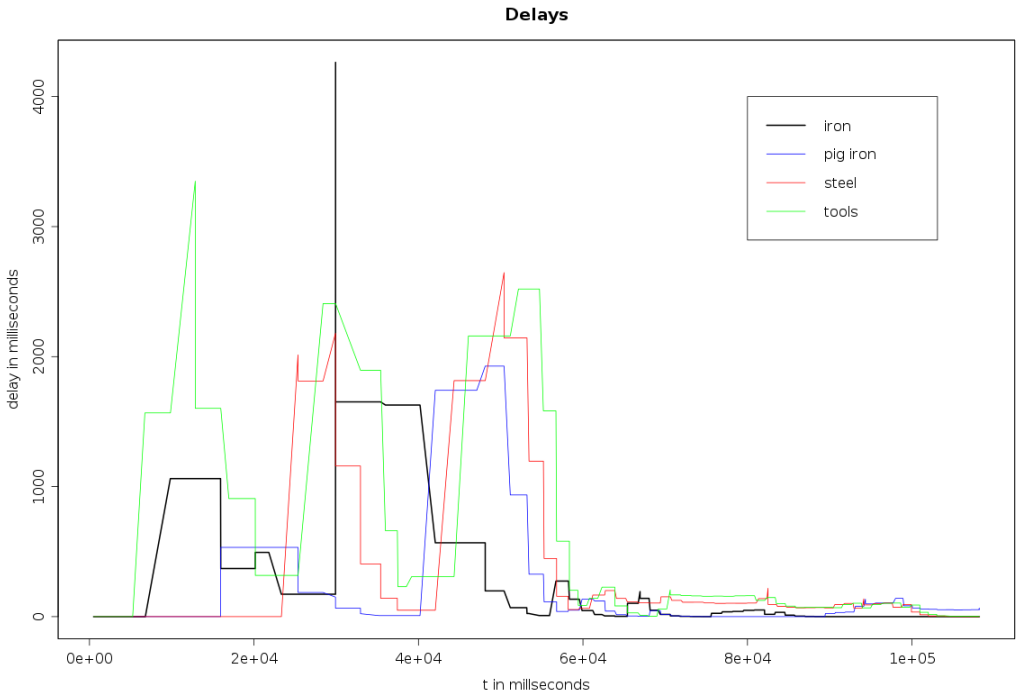

Figure 3: The delays in each market over a simulated run. Notice the cascading effects of delay of a good on the successive links in the supply chain

makers outstrip their suppliers, when that happens delays start at the source and cascade down.

The end result is reminiscent of the beer distribution game (Sterman, 1995). Both models deal with supply chains and generate spike in delays. The mechanisms that generates them are different though. Beer dynamics originate from wrong expectations of players,here agents are backward-looking. They react rather than predict.

Economic growth emerges in this run. I set the maximum number of workers at 500, and firms employ them all in about 10 minutes of simulation (see figure 7). In the end of this sample run there are 12 workers in the iron mines, 58 in the Smelters, 106 in the steel mills and 304 in the equipment makers. It seems that the employees triple

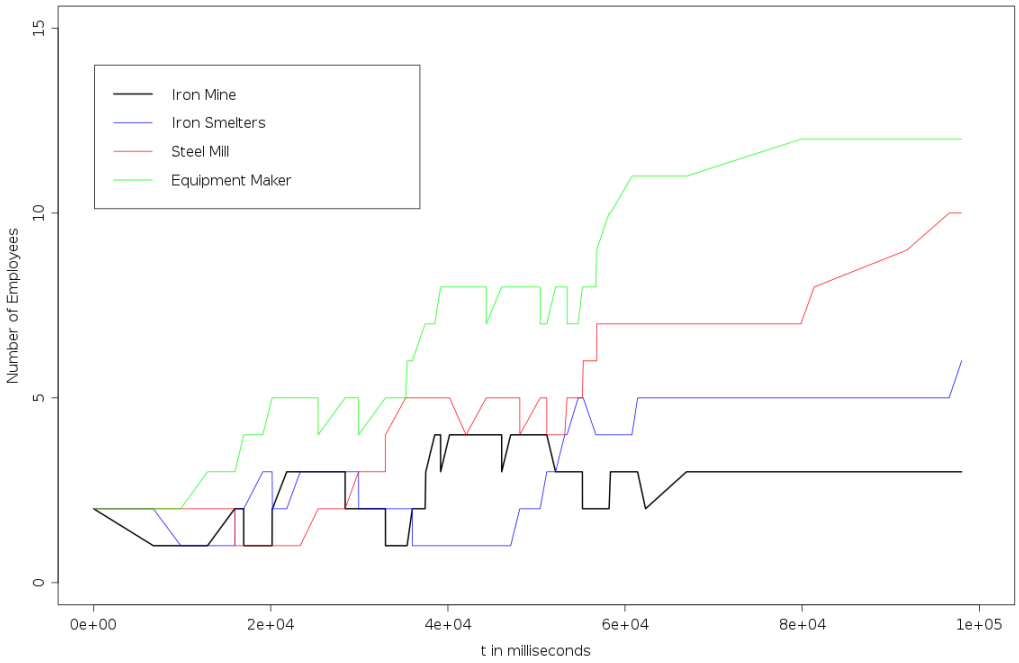

Figure 4: The number of workers in each firm over a simulated run

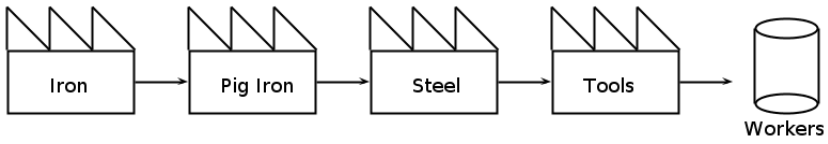

Figure 5: Now iron mines require no input and tools are consumed by workers

at each link in the supply chain. This is unwarranted by the work units assumptions alone.

The independence of labor allocation from production structure is puzzling. The results are clearer if we invert our distribution of work units, making iron mines the slowest product (1200 work units) and tool makers the faster (300 work units). The final workers distribution:

43 in the mines, 101 in the smelters, 117 in the steel mills and 239 producing tools. Possibly these results will disappear when I'll model explicit contracts and monetary decisions.

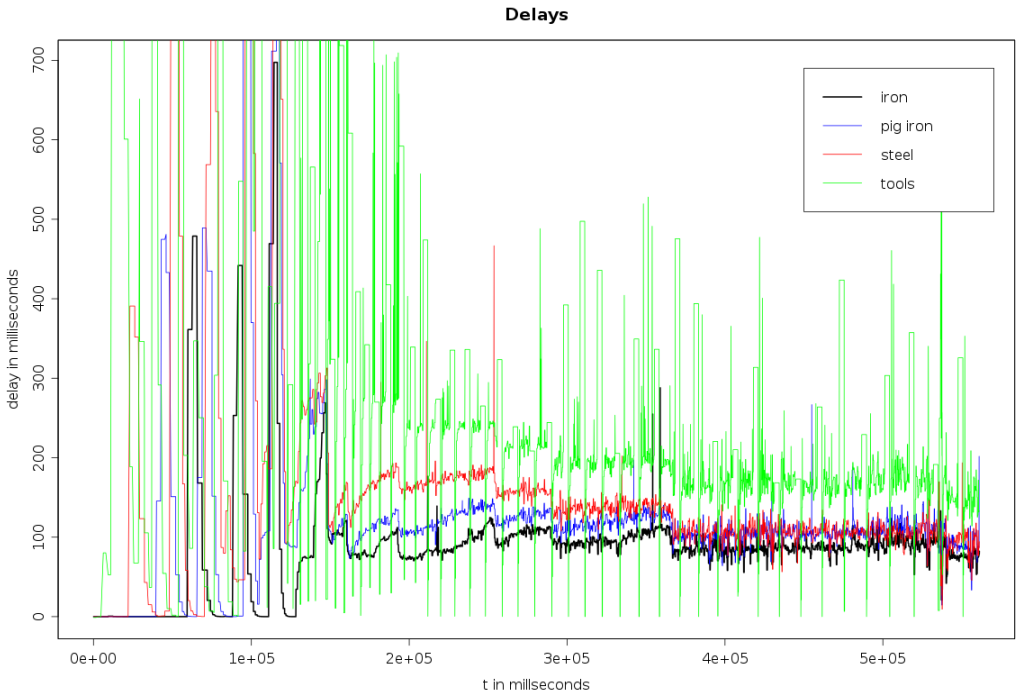

Figure 6: The delays in each market over a simulated run.

4 Multi-Threading Weaknesses

There are two major weaknesses highlighted by this exercise. The first is data gathering. Threads, once started, are very hard to stop. This means that data needs to be collected on the fly, as the model is running. There is the risk that the model will change before a full snapshots of its variables can be taken.

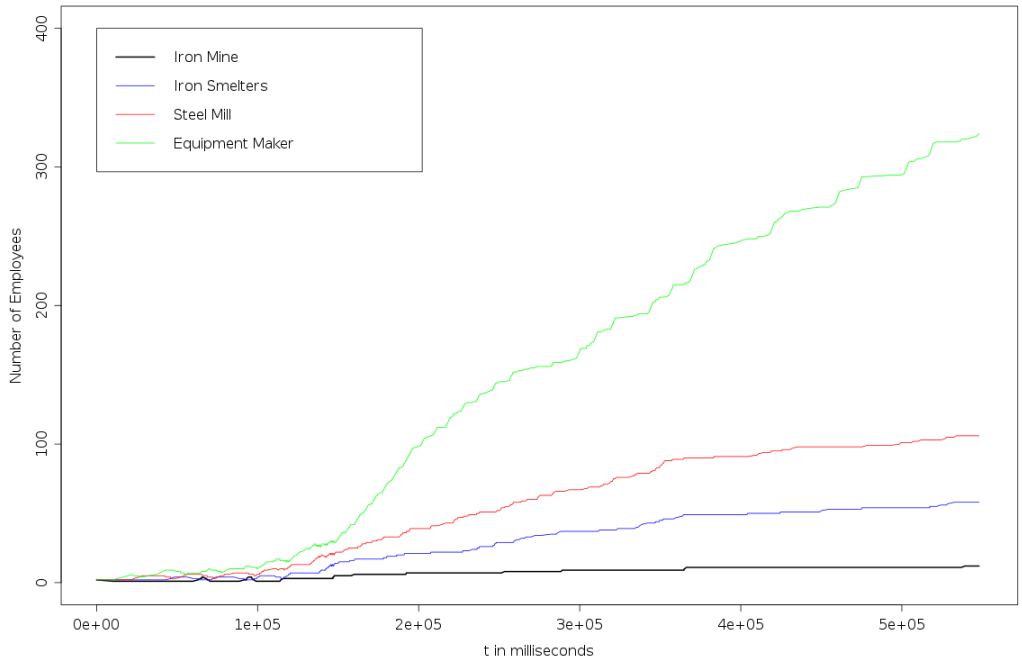

Figure 7: The number of workers in each firm over a simulated run

This was "solved" by having the model go more slowly. My model gathers data through a dedicated daemon thread. If the model is too fast the data thread would yield before it's done writing to file.

On one hand this is a problem of data changing rather than data gathering. Data becomes stale quicker than it can be observed. But as researchers we want to understand our models and taking snapshots is a clear way to do so. Multi-threading ability to do so is impaired.

The second issue is scalability. Threads and Runnables are Java's infrastructure to generate simultaneous programs. Such infrastructure has its costs. In section 5 toward the end there were 500 consumer threads plus 500 worker threads running. If we increased the number of workers then the CPU costs that Java had to bear to switch from one thread to the other would have fed into the firm delays. There is

the risk of artifacts creeping in the model.

To avoid this risk I let each thread do only very simple tasks. Make agents very simple is always a good strategy (Miller & Page, 2007) but for multi-threaded models it might be a necessity. How long will this stay a problem depends only on how accurate Moore's law is.

5 Future Extensions

5.1 Model Extensions

The first obvious expansion required by this model is to explicit contracts. That would require multiple firms per sector competing for contracts. I plan on having a specialized retail sector that acts as a "market maker", providing producing firms with contracts and waiting for consumers to sell to.

Adding contracts will allow me to couch delay minimization inside the more usual profit maximization. Prices will make more sense and I can study the effects on allocation of resources after taking into account production times.

I would also like to add different kind of jobs. One common problem of input-output models, both in the classical and heterodox alternatives, is the assumption that while goods are too different to be substitute one for the other there is only one kind of labor.

Eventually my goal is to create a full macro-economic model. That would require, after prices are in, savings to be modeled. This means also long-term inputs and depreciation.

5.2 Technique Extensions

I think multi-threading is a useful approach for any agent based modeler. I imagine it being especially useful in situations fraught with externalities when the number of agents is quite limited.

I think a perfect application field would be work-place simulation or industrial organization. The scope would increase as the cores in our computers multiply.

The data collection issue is going to be especially problematic when parameter sweeps. Perhaps Java inability to pause threads on command will prove fatal and we'll need to migrate to a more suitable programming language.

References

Axtell, R. L. 2003. Economics as distributed computation. *In:* Terano, Takao (ed), *Meeting the challenge of social problems via agent-based simulation*.

Huberman, B. A. 1998. Computation as economics. *Journal of Economic Dynamics and Control*, **22**(8-9), 1169–1186.

Huberman, B. A., & Hogg, T. 1995. Distributed computation as an economic system. *The Journal of Economic Perspectives*, **9**(1), 141–152.

Leontief, W. 1966. *Input-Output Economics*. 1st edn. Oxford University Press, USA.

Miller, John H., & Page, Scott E. 2007. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life (Princeton Studies in Complexity)*. Princeton University Press.

Quesnay, F. 1852. *Analyse du tableau conomique*.

Sraffa, P. 1975. *Production of commodities by means of commodities: Prelude to a critique of economic theory*. Cambridge University Press.

Sterman, JD. 1995. The beer distribution game. *Games and Exercises for Operations Management*, 101–112.