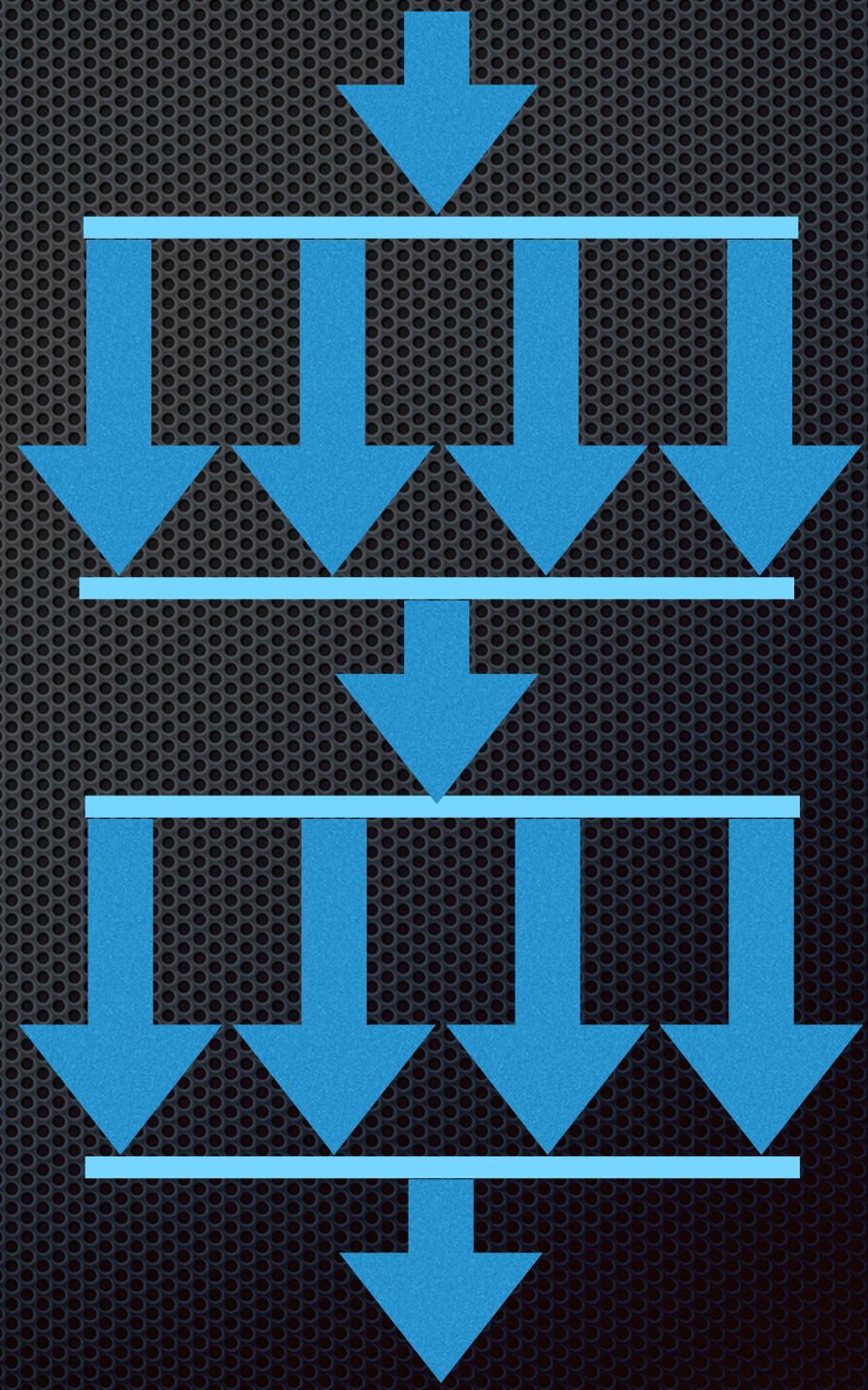
A Comparison of Software Frameworks for the Parallelization of Large-Scale ZI Trader Models

Rob Axtell
with
Dale Brearcliffe, Peter Froncek, Marta Hansen,
Vince Kane, and Stefan McCabe

George Mason University

Background/Opportunity

- Most (95+%?) of agent-based models are single-threaded
- When parallel (e.g., D-Mason, Repast HPC), usually spatial
- Multi-core hardware offers possibilities to partition agents into subpopulations



Amdahl's Law

T: execution time of single-threaded code

$$T = f_s T + f_p T$$

$$f_s + f_p = 1$$

$$T(\mathrm{n}) = f_s T + f_p T/\mathrm{n}$$

$$S(n) = T(1)/T(n)$$

$$S(\infty) = 1/f_s$$

Motivation

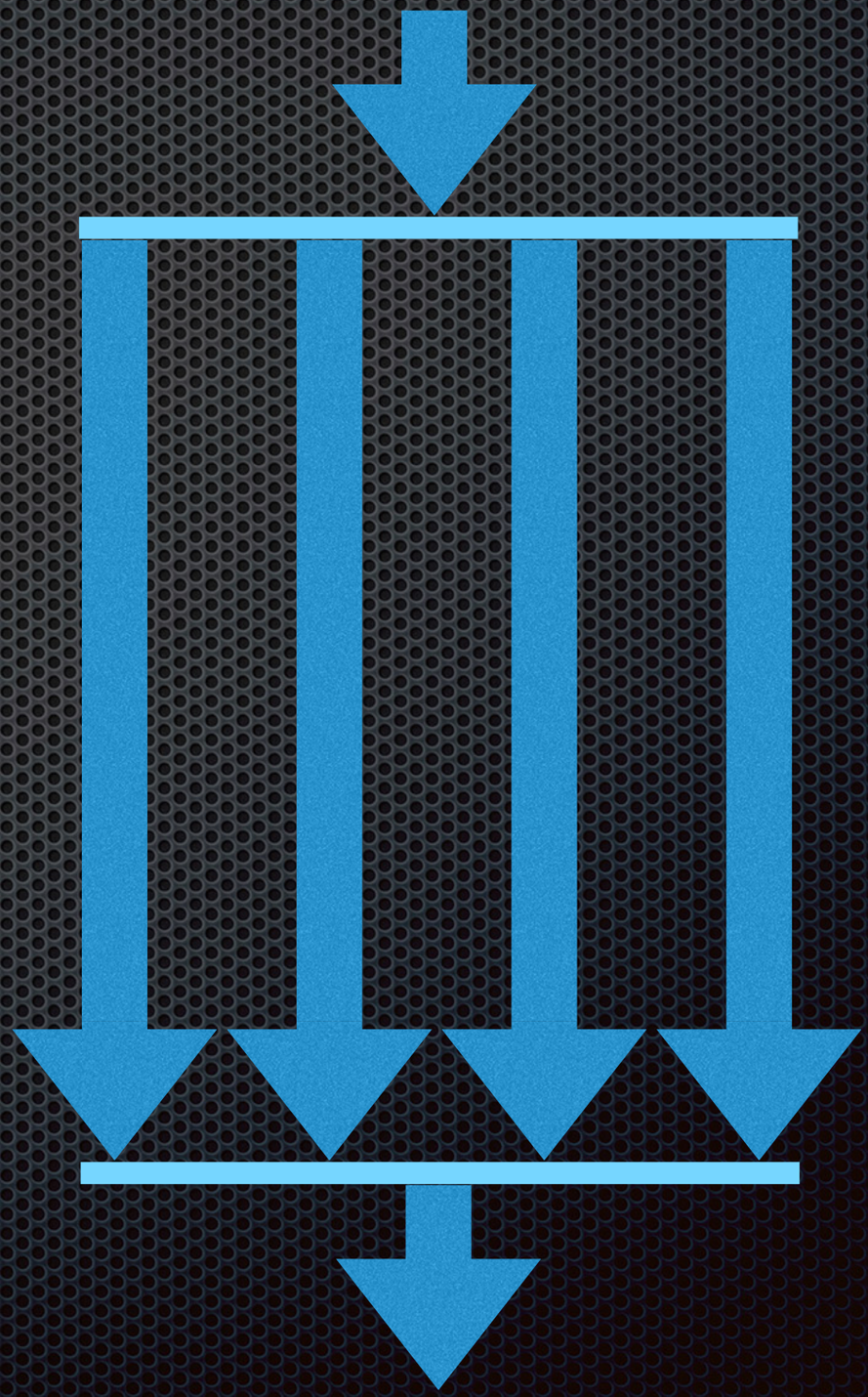
- Full-scale city models (millions of agents)
- Small country models (tens of millions of agents)
- Artificial economies (U.S. private sector has 120 million employees, 6 million firms)
- Global epidemic models (10^9 agents)
- Whole world simulations? ($O(10^{10})$ agents)
- Global population of mosquitos? 10^{11}-10^{12}?
- Nanoparticle medicine delivery… 10^9-10^{15}?

Big iron: Tianhe-2

- 31 million cores
- Intel Xeon Phi's on multiprocessor boards
- Imagine each agent on its own core....
- Approaching truly autonomous agents....
- Several agent projects planned...

Supply-Demand, ZI traders

- Initialize agents
- Partition the traders into subpopulations of buyers and sellers
- Let each ‘submarket’ run to completion
- Compute statistics
- $f_s < 0.05$



ZI Traders Code

- Pseudo-code:

```
• INSTANTIATE and INITIALIZE BUYER, SELLER, DATA and THREAD objects;
• Assign sub-populations of BUYERS and SELLERS to THREADS;
• FORK all THREADS;
• FOR each THREAD, REPEAT:
    ○ Randomly activate 1 BUYER agent + 1 SELLER agent:
        ▪ BUYER proposes a BID price;
        ▪ SELLER proposes an ASK price;
        ▪ IF (BID > ASK) THEN
            • Pick EXECUTION price between BID and ASK;
            • INCREMENT BUYER holdings;
            • DECREMENT SELLER holdings;
            • Collect DATA on the trade;
    ○ INCREMENT the attempted number of trades;
    ○ END when maximum trade attempts exceeded;
• JOIN all THREADS;
• Collect final DATA;
```

- <Run NetLogo version>

Implementation/execution

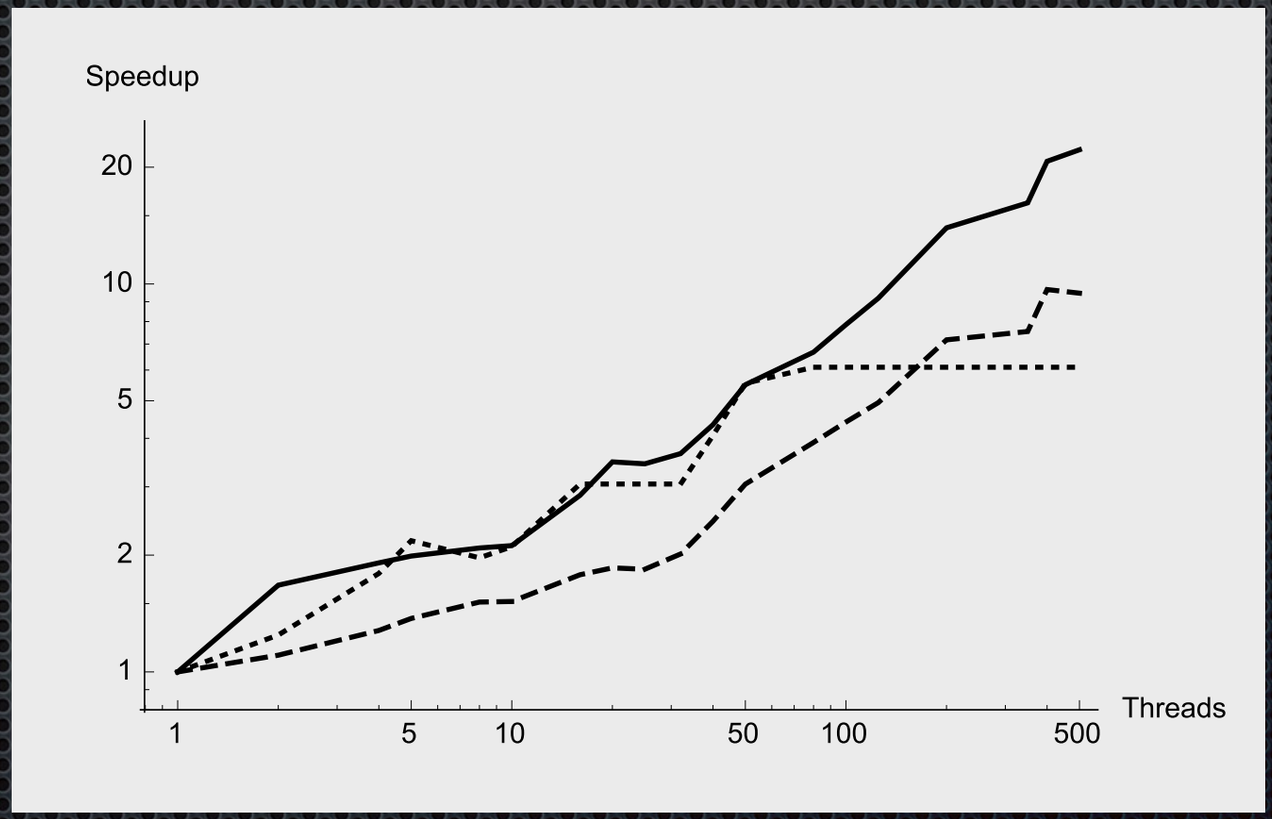
- C/C++: pthreads in C, C++11 threads, OpenMP <Rob>
- Java: native threads <various>
- Clojure <Dale>
- Erlang <Peter>
- Go <Stefan>
- Haskell <Vince>
- Scala <Marta>

3 Model Sizes

- 10K buyers and 10K sellers, 1 million attempted trades
- 100K buyers and 100K sellers, 10 million attempts
- 10^6 buyers and 10^6 sellers, 100 million trade attempts
- Microway workstation w/2 E5-2687W (8 cores/chip), 20.5 MB cache/core, 256 GB RAM, Linux (Fedora)
- NVIDIA GTX 980 on same workstation…

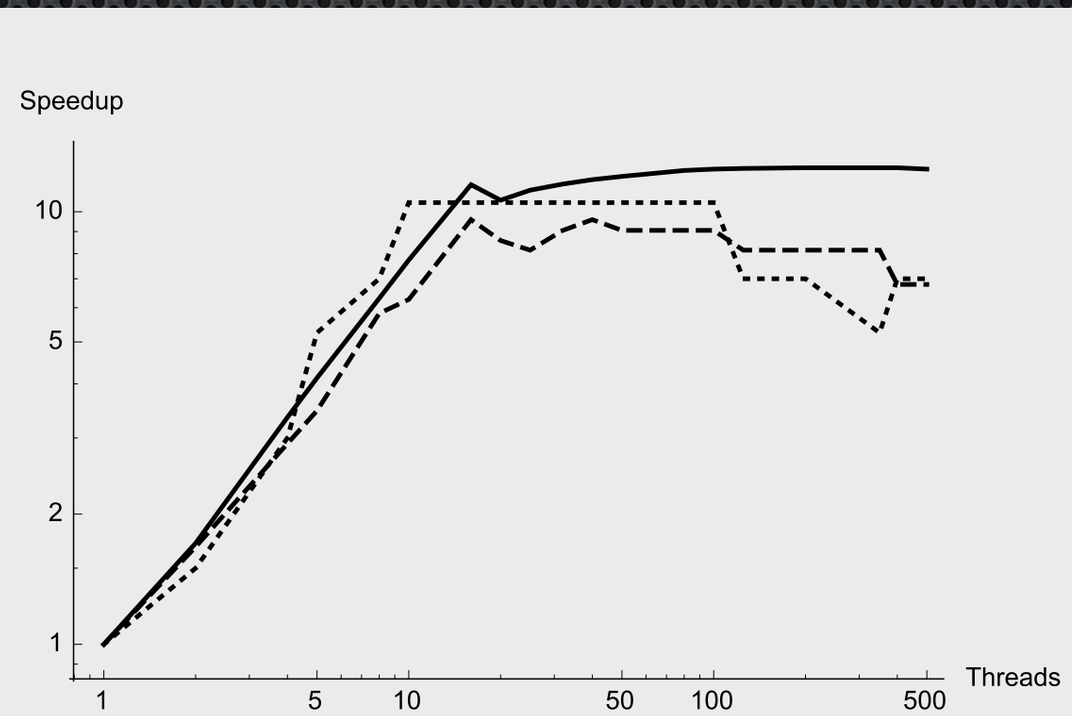
C and POSIX threads

- Several compilers used including Intel Parallel Processing Studio



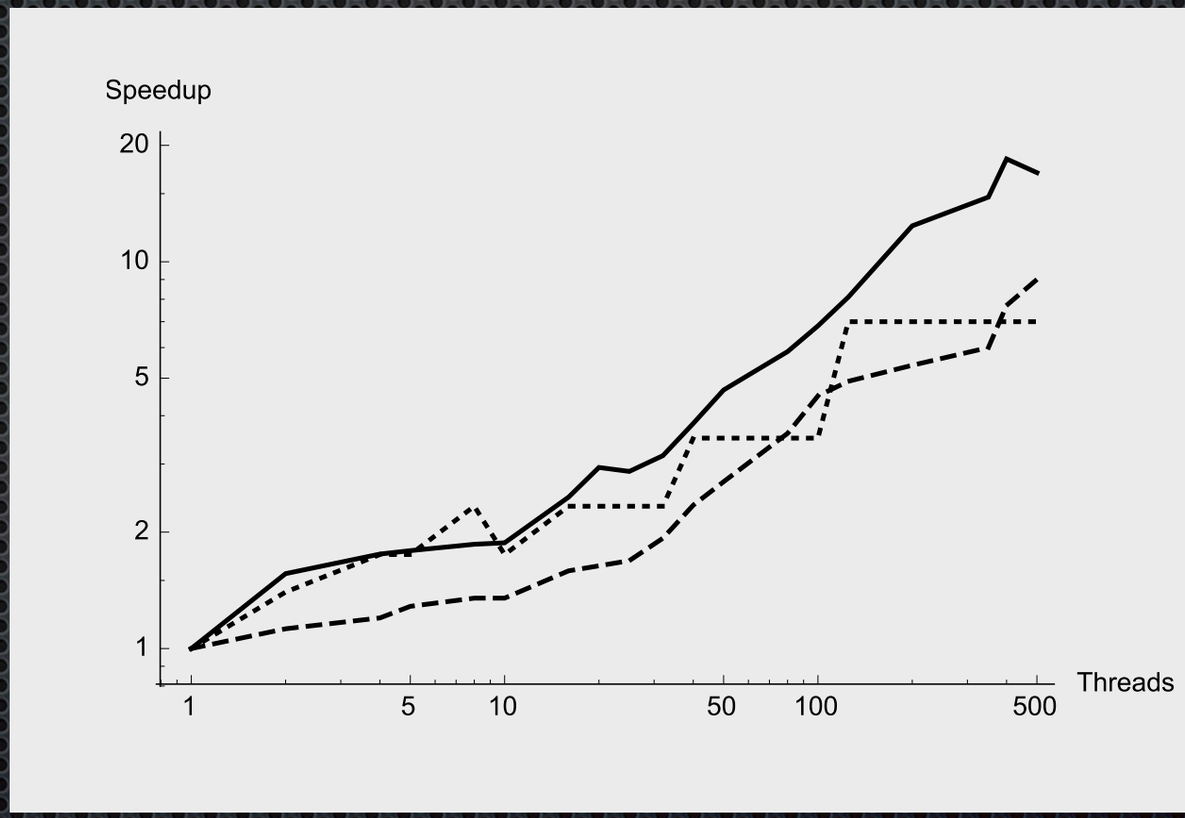
C++11 Threads

- Some compilers implement C11 standard using pthreads



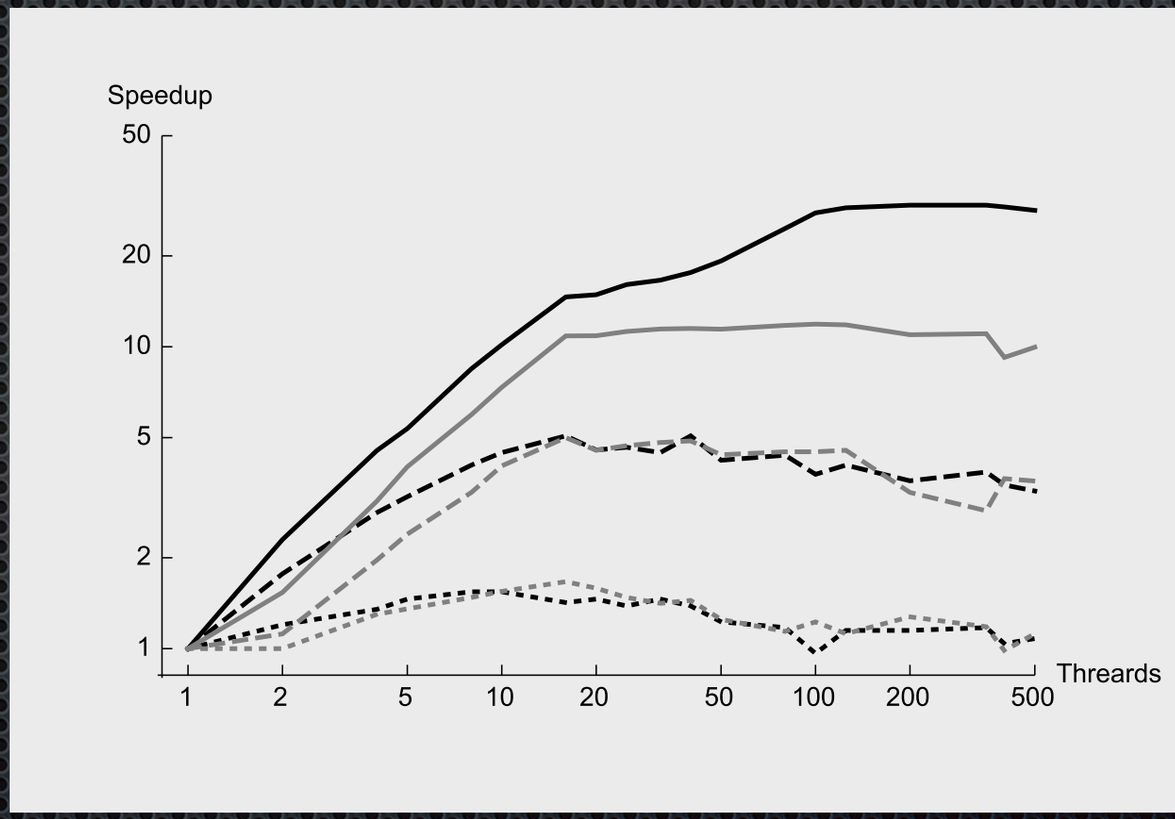
OpenMP under C

- Also runs under C++, FORTRAN, others…



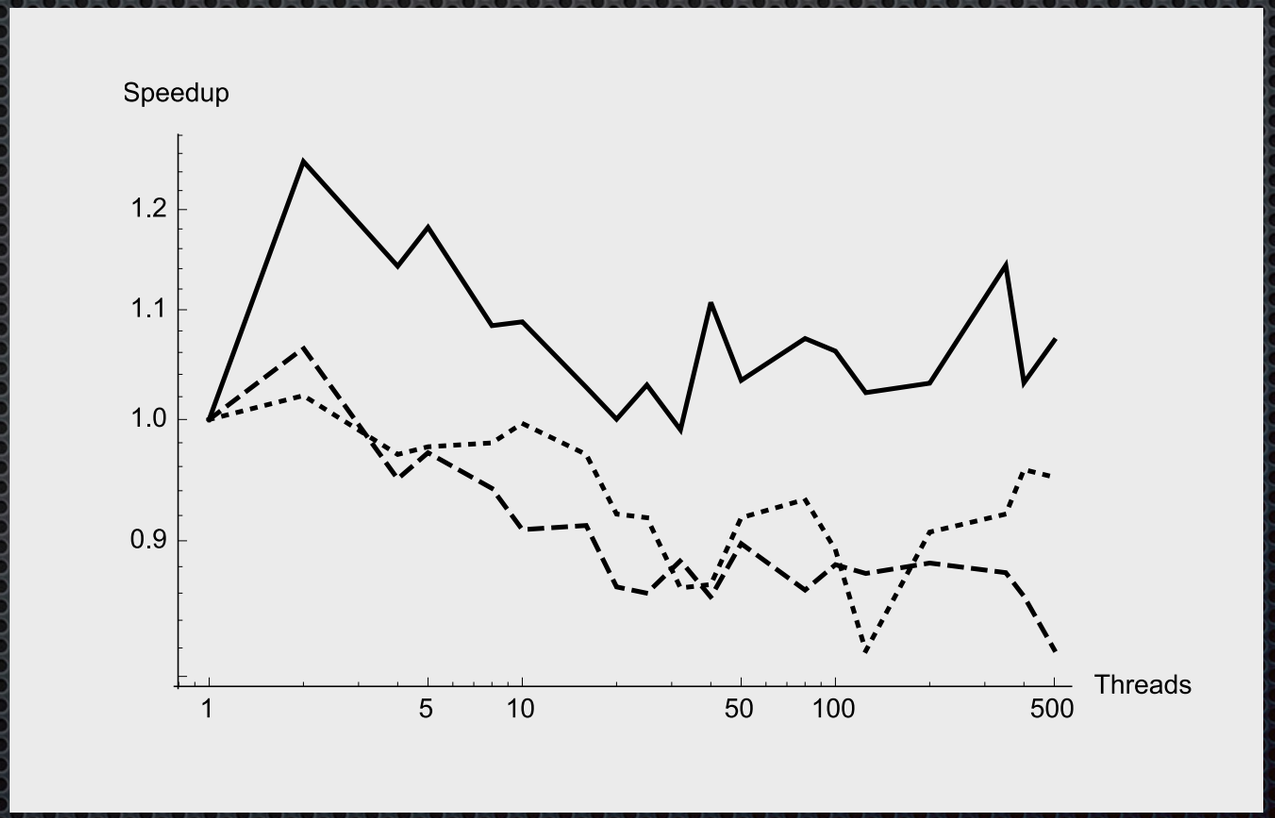
Java Threads

- 2 implementations: one naive, one sensitive to Java locks



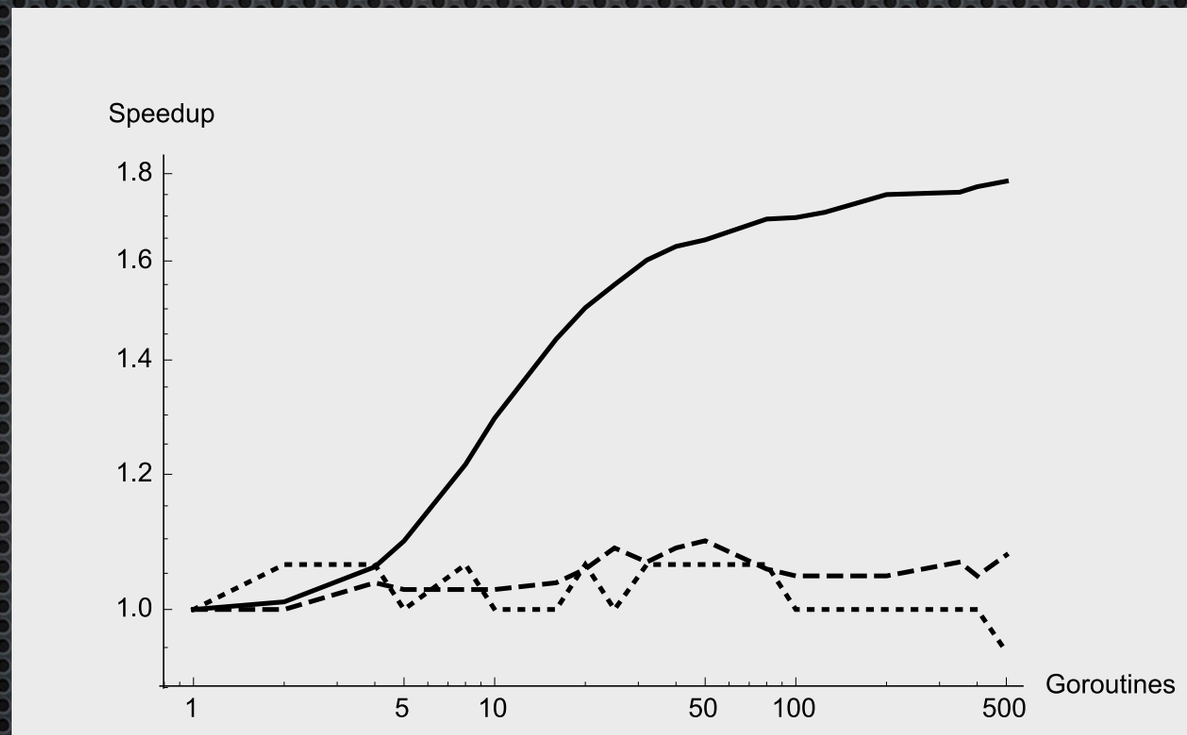
Clojure

- Functional programming running on JVM
- No need for code locking



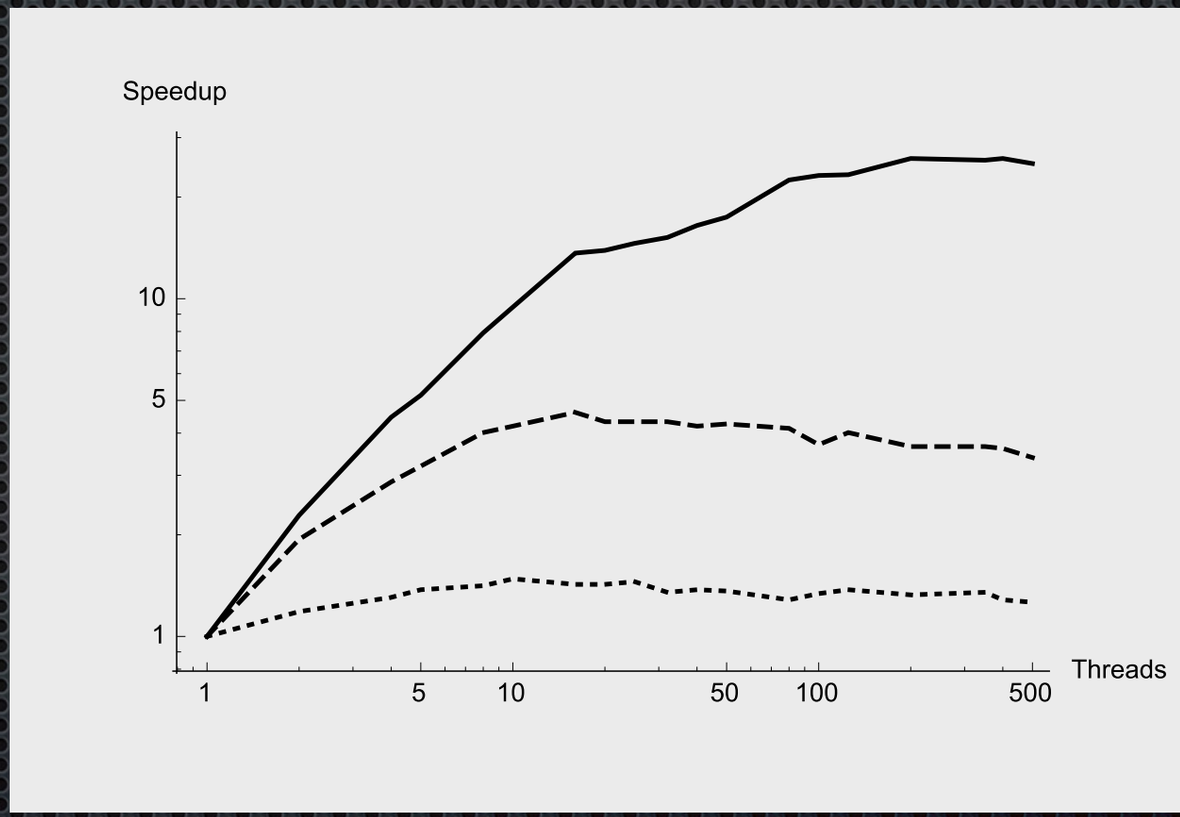
Go

- Language designed for concurrency from Google
- *goroutines* instead of threads



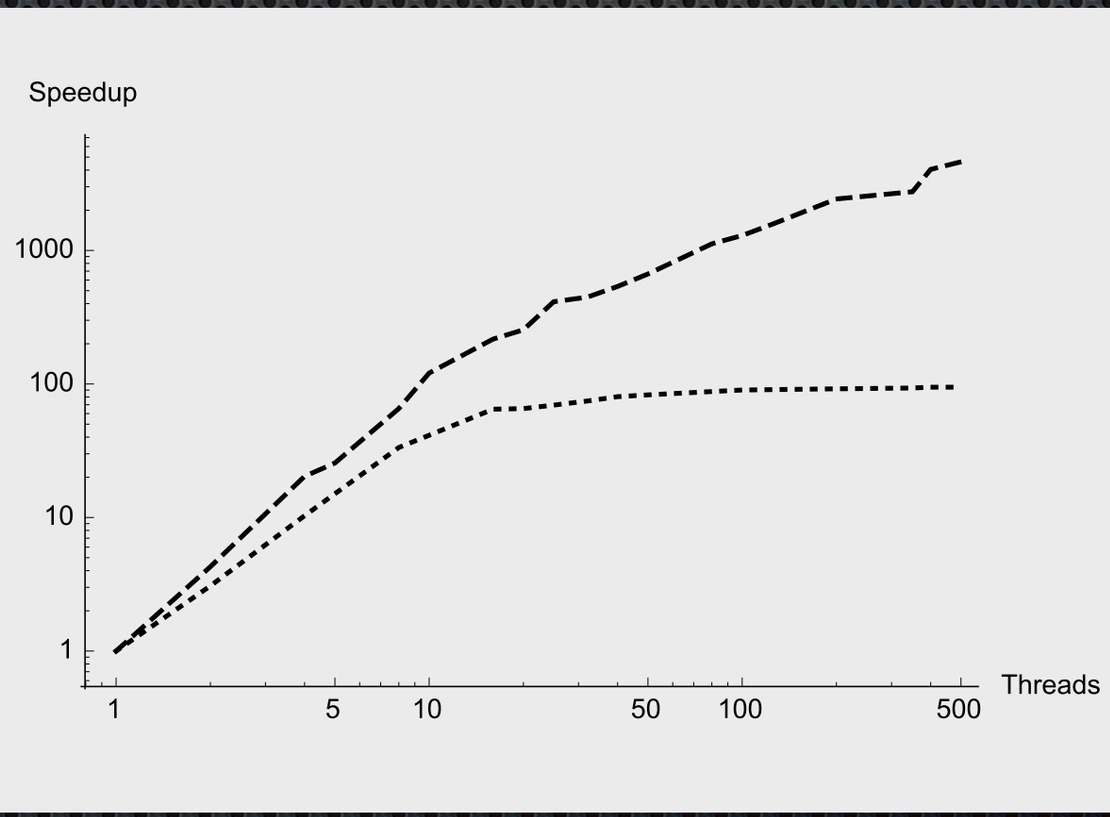
Scala

- Scalable, functional programming based on JVM
- Data structures can be mutable or immutable



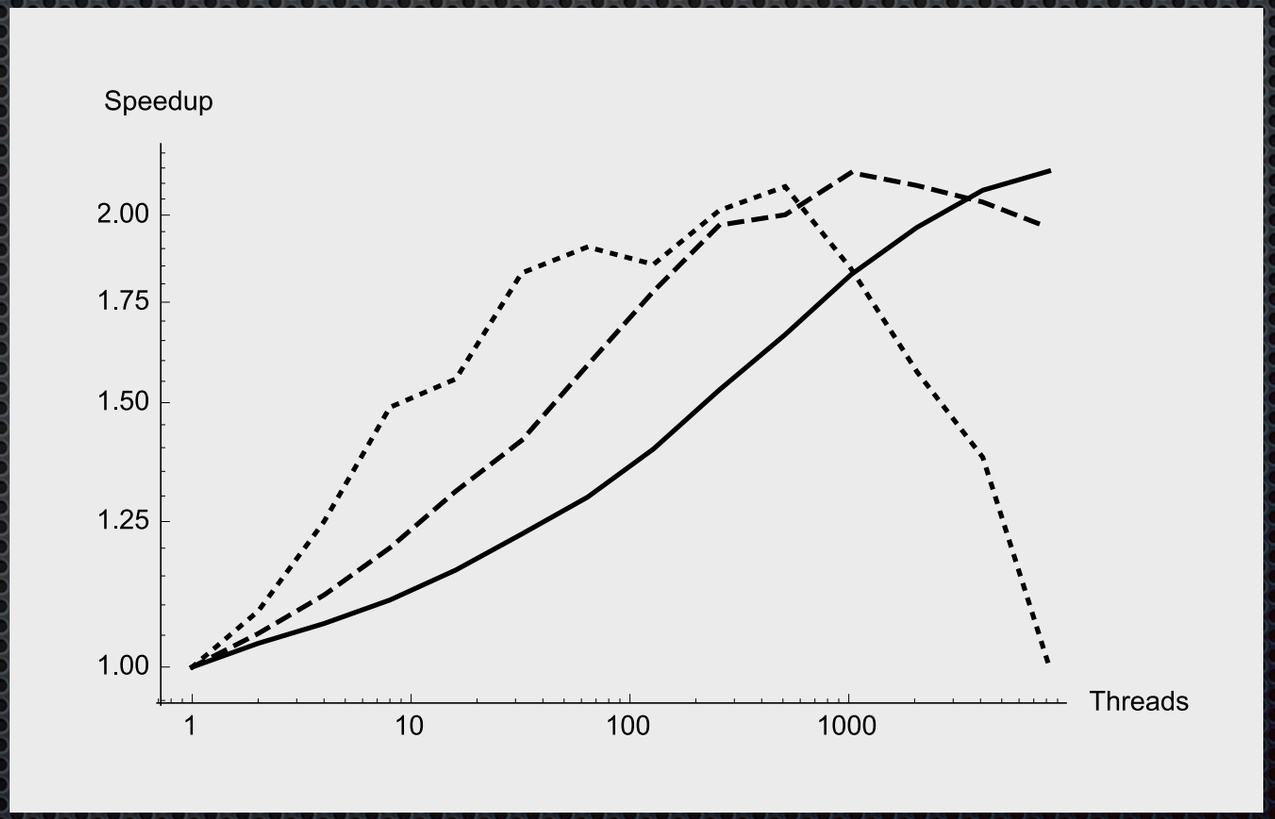
Erlang

- Older functional language originating in telecommunications
- Built for concurrency

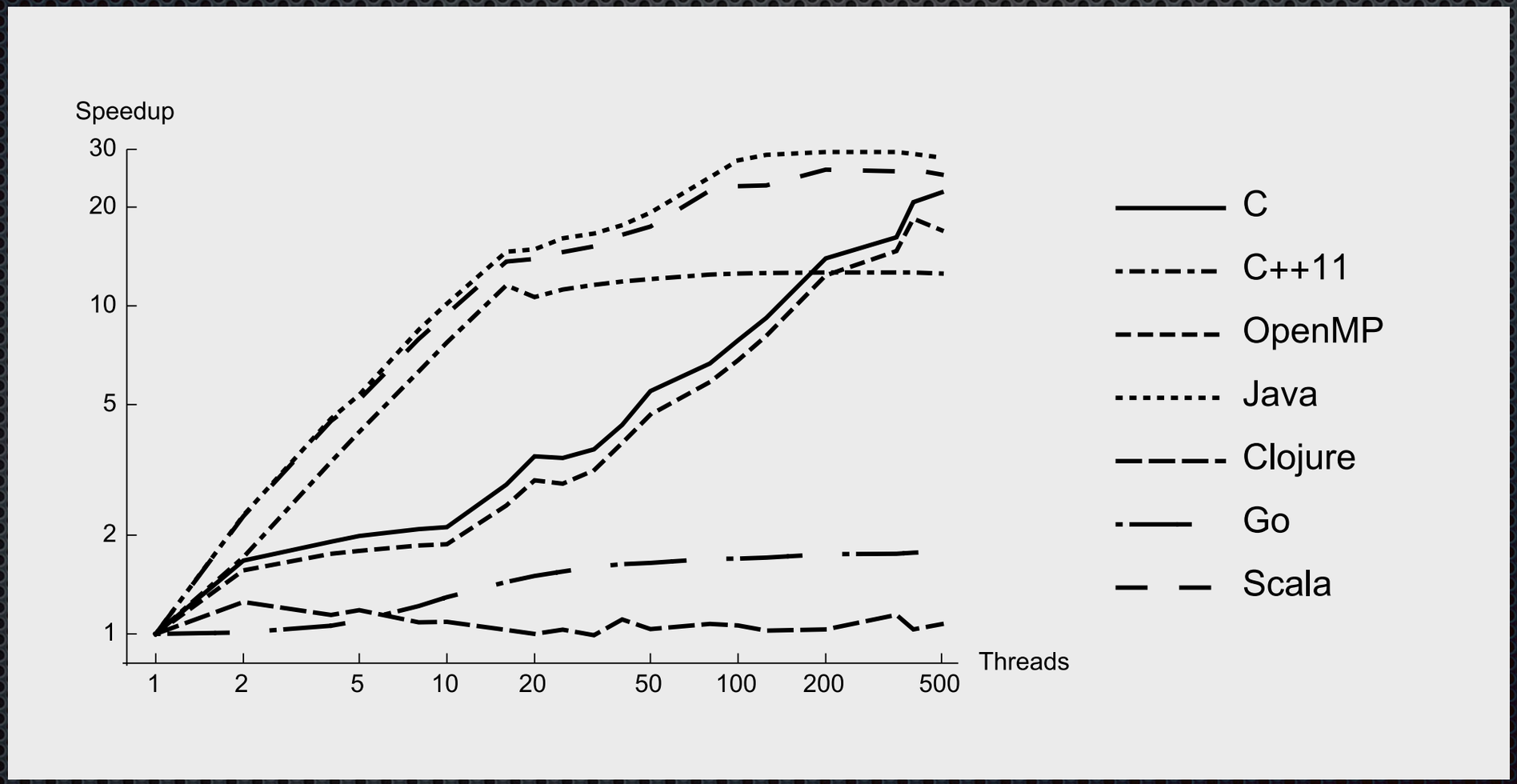


Haskell

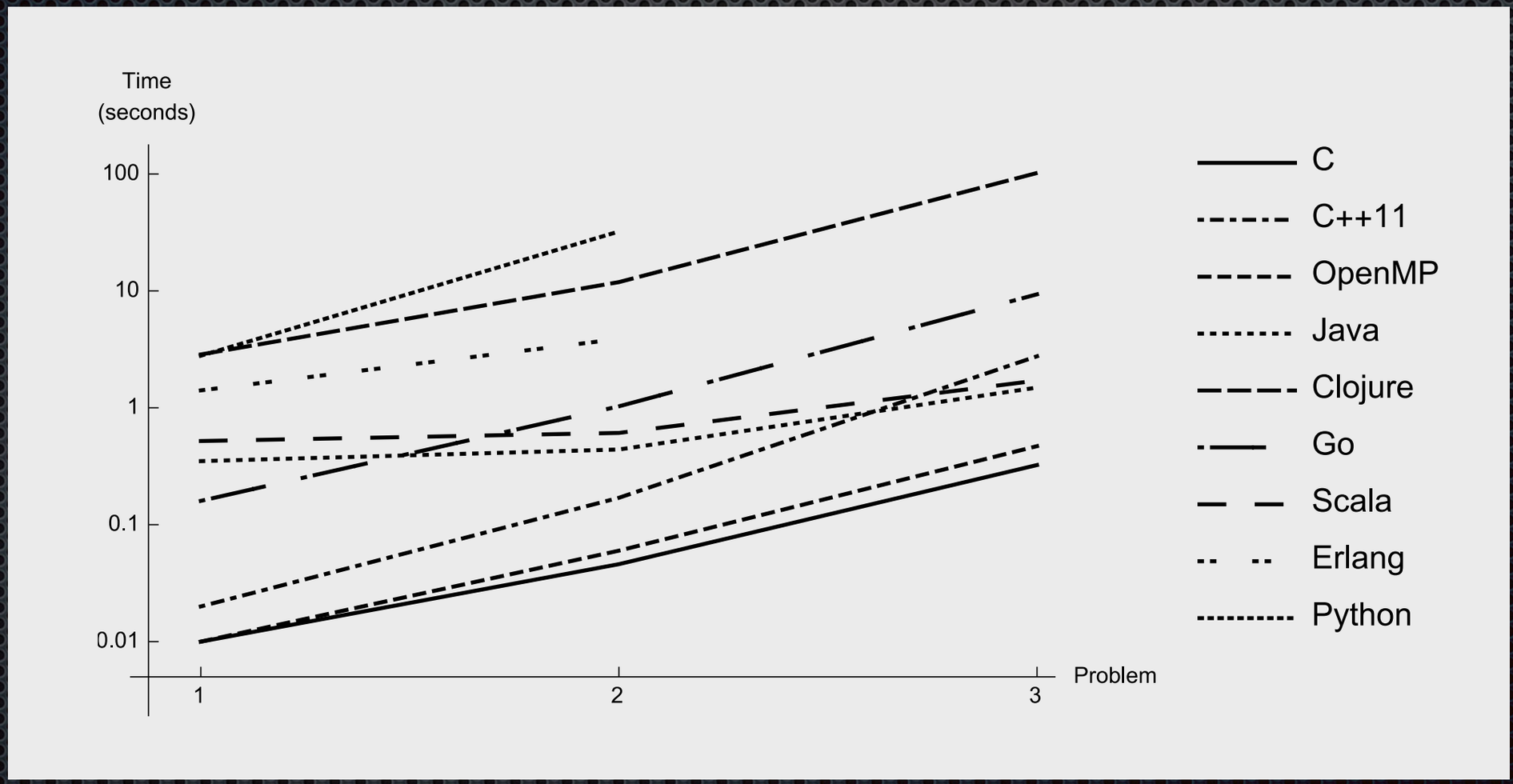
- Functional language based on the lambda calculus
- Concurrency and parallelism provided by extensions



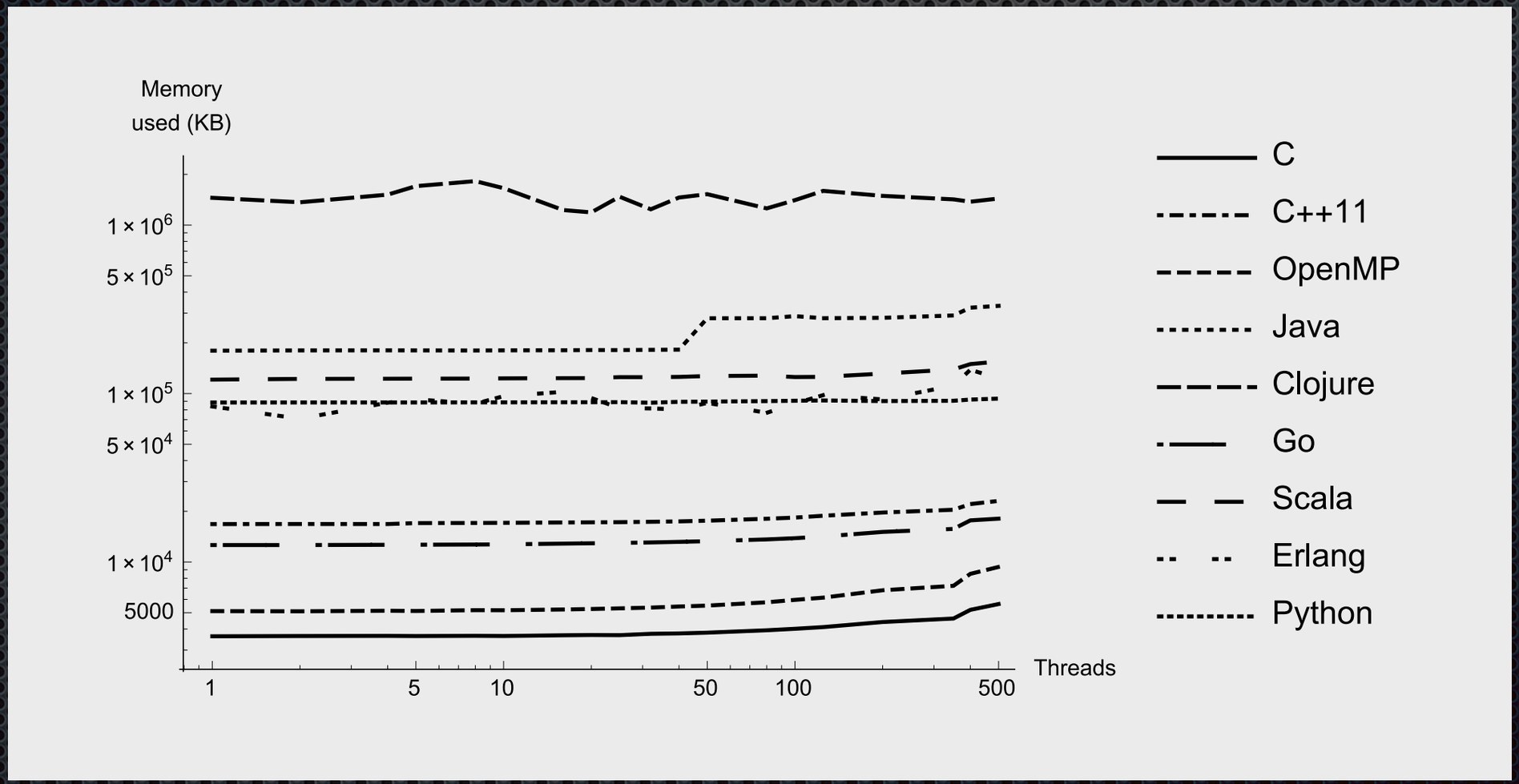
Comparison: Speedups



Comparison: Performance



Comparison: Memory



Summary and Conclusions

- Parallel execution is the future for ABMs
- Variety of agent parallelization software compared:
  - Factor of 100 in range of performance!
    - Partially due to language/compiler maturity
    - Partially due to design differences (e.g., persistence)
- Large speedups are possible!
- Today, coding for parallel execution is an art…