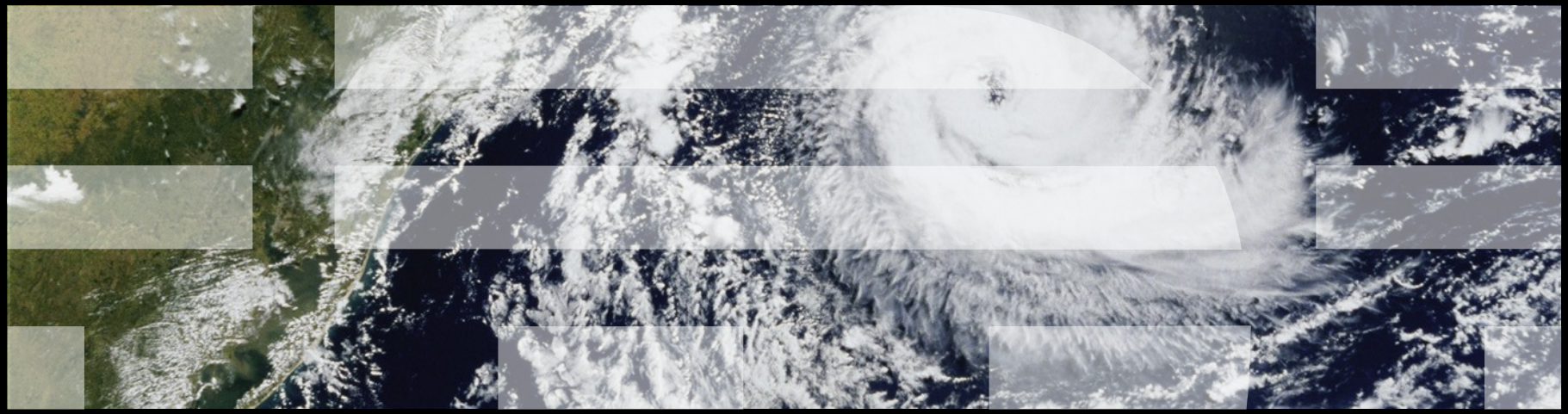
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center

June 8, 2012

Retrofitted Parallelism Considered Grossly Sub-Optimal

HotPar '12: 4th USENIX Workshop on Hot Topics in Parallelism



HotPar'12: Retrofitted Parallelism Considered Grossly Suboptimal

Labirinto do Outeiro do Cribo, A Armenteira, Meis, Pontevedra, Galicia, Spain. Possibly dating from as early as the Bronze Age (though rock carvings are notoriously difficult to date with certainty). *10 October 2006, Froaringus*

But Now We Use Computers To Solve Mazes

Goals

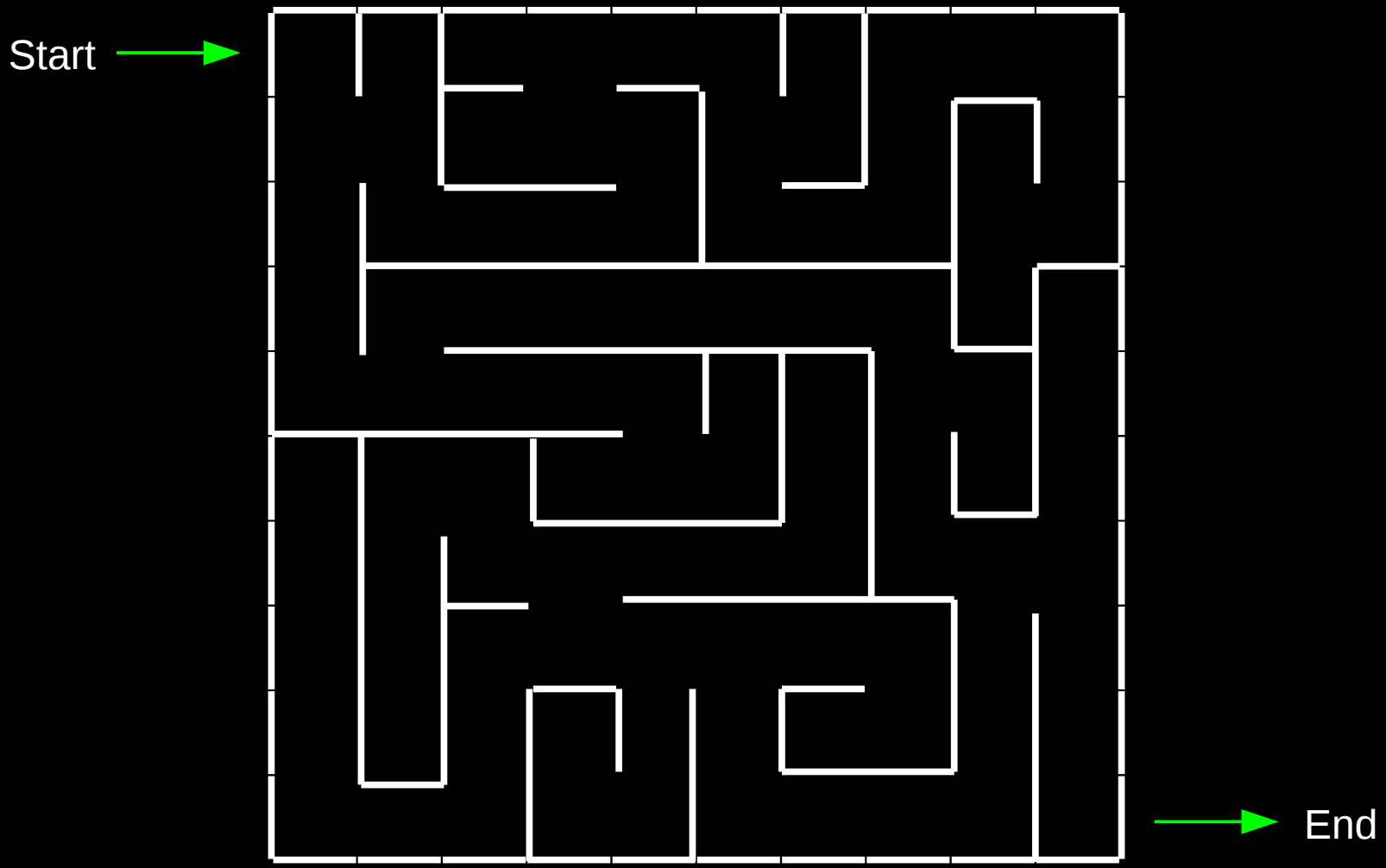
Goals (Why Was I Messing With Mazes???)

Goals (Why Was I Messing With Mazes???)

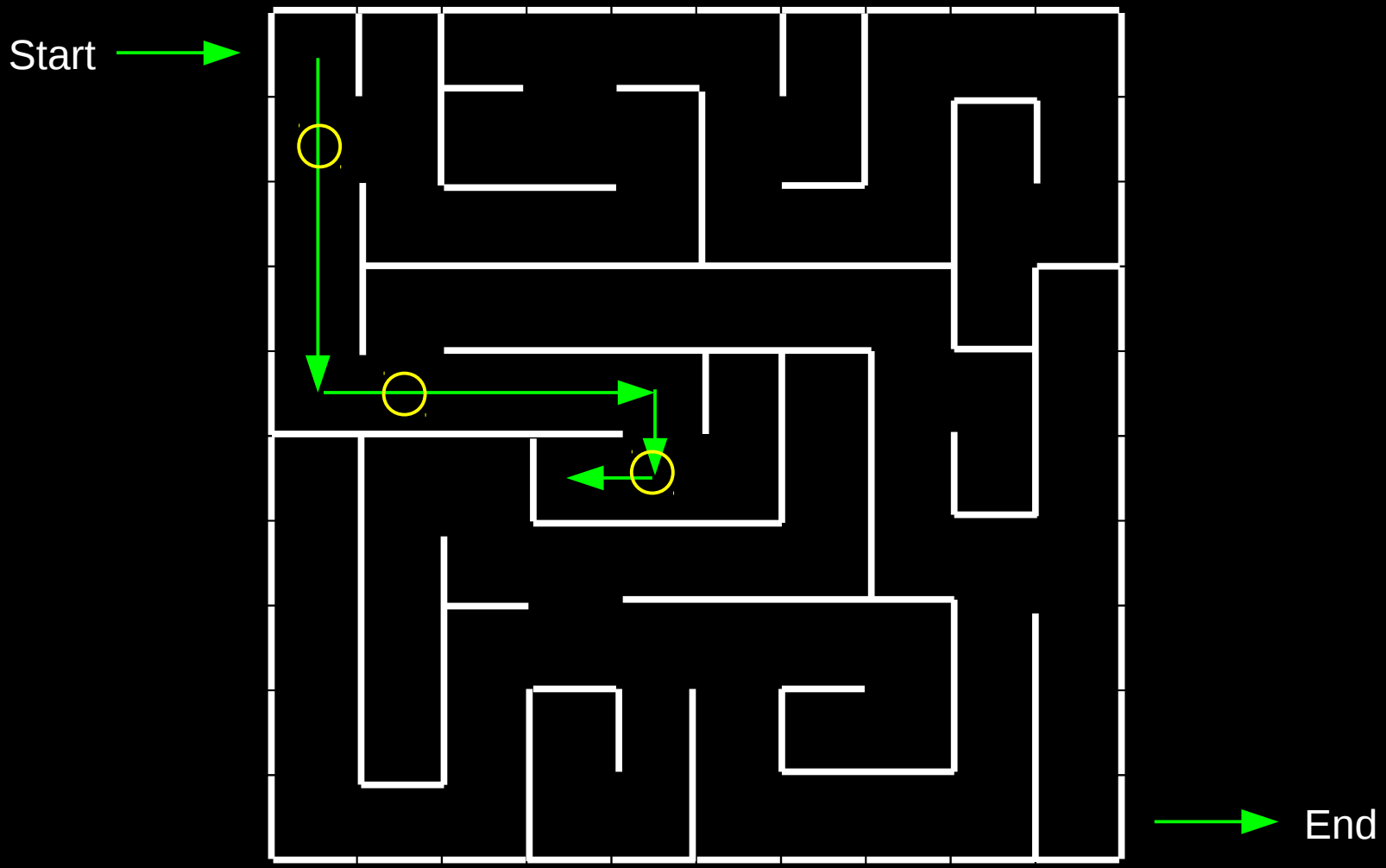
- An example of near-perfect partitioning for “Is Parallel Programming Hard, And If So, What Can You Do About It?”
- Use case for RCU-protected union-find data structure

But First, A Sequential Maze Solver

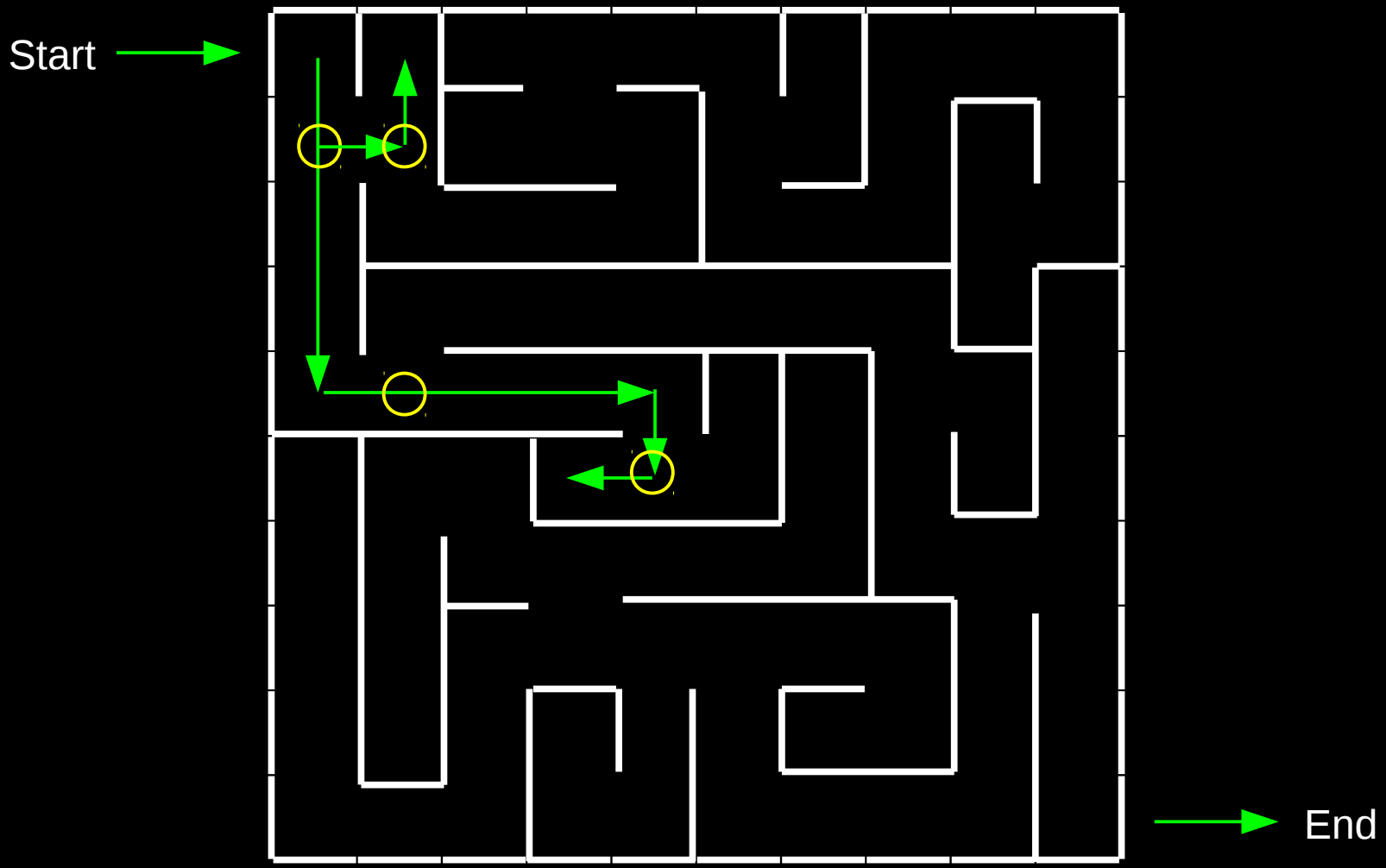
Sequential Maze Solving (SEQ)



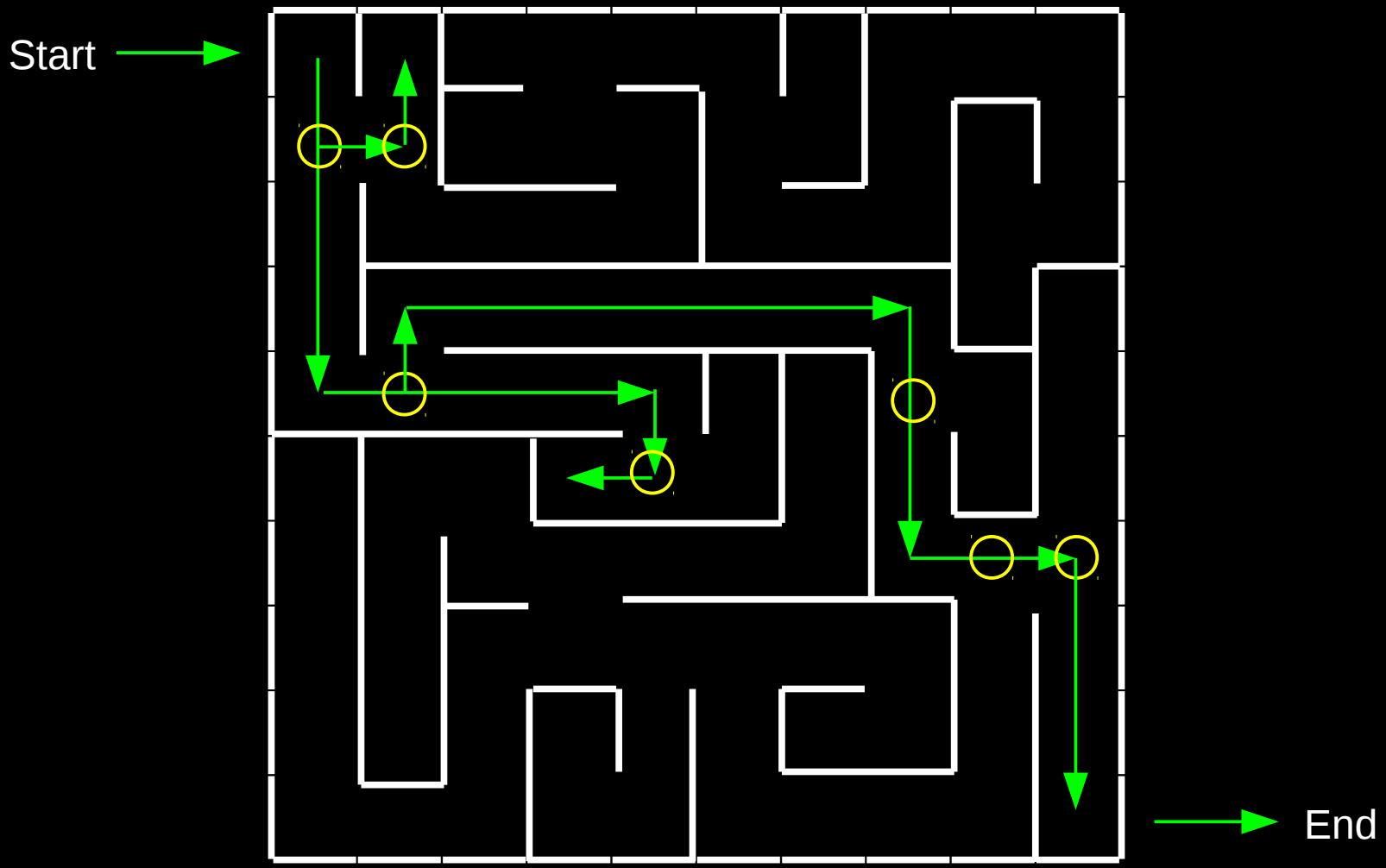
Sequential Maze Solving



Sequential Maze Solving

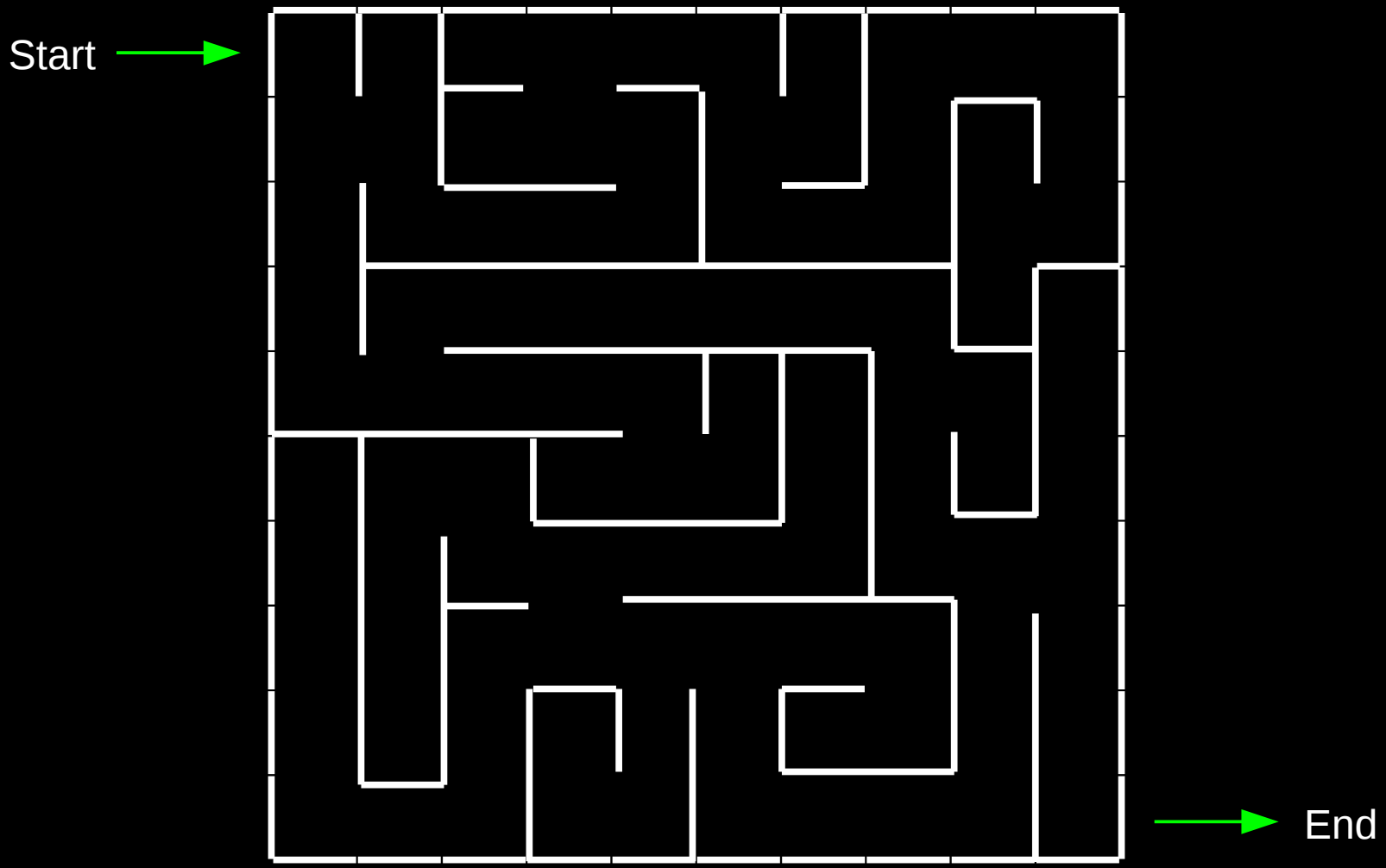


Sequential Maze Solving

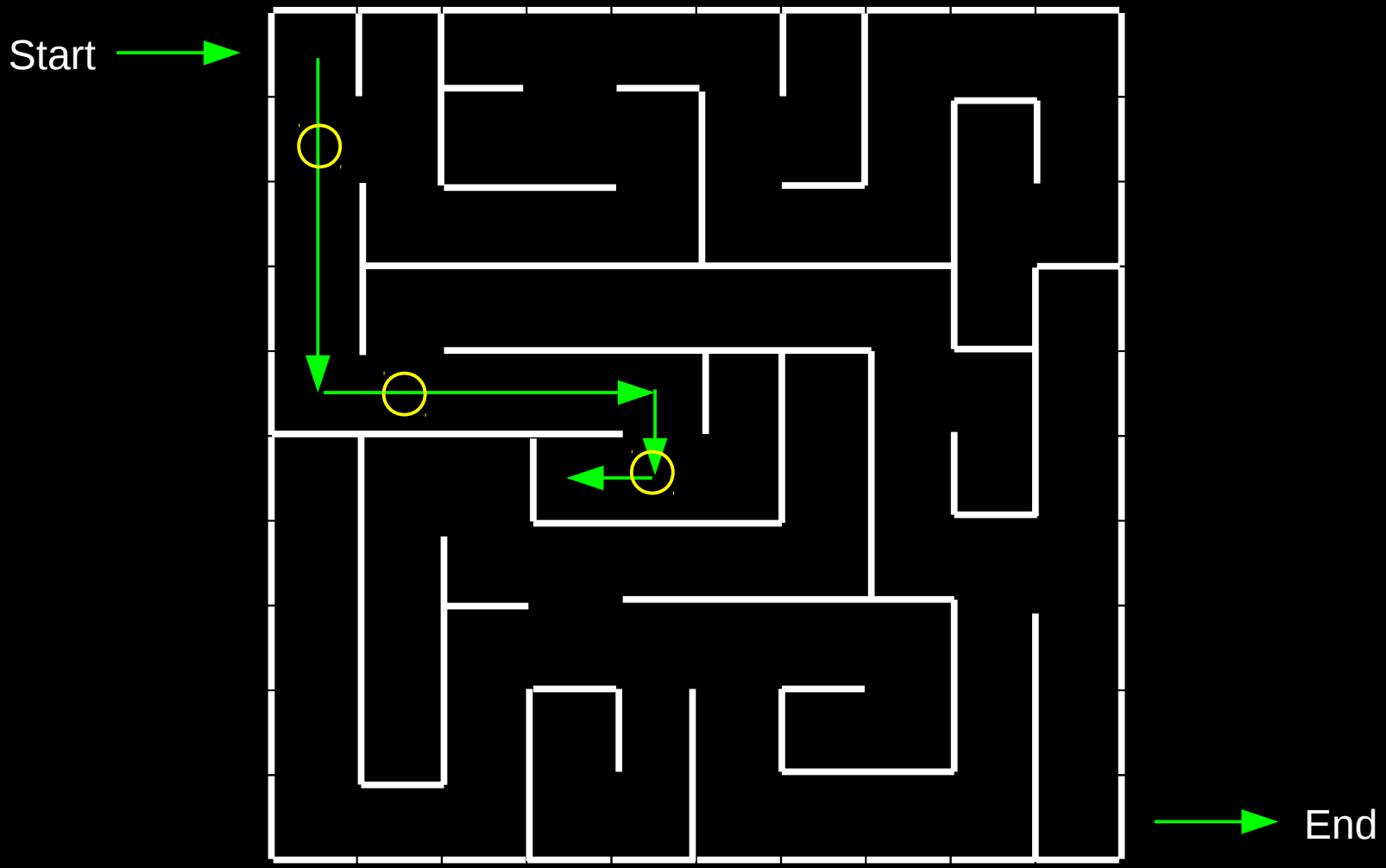


Parallel Maze Solving: Work-Queue Approach

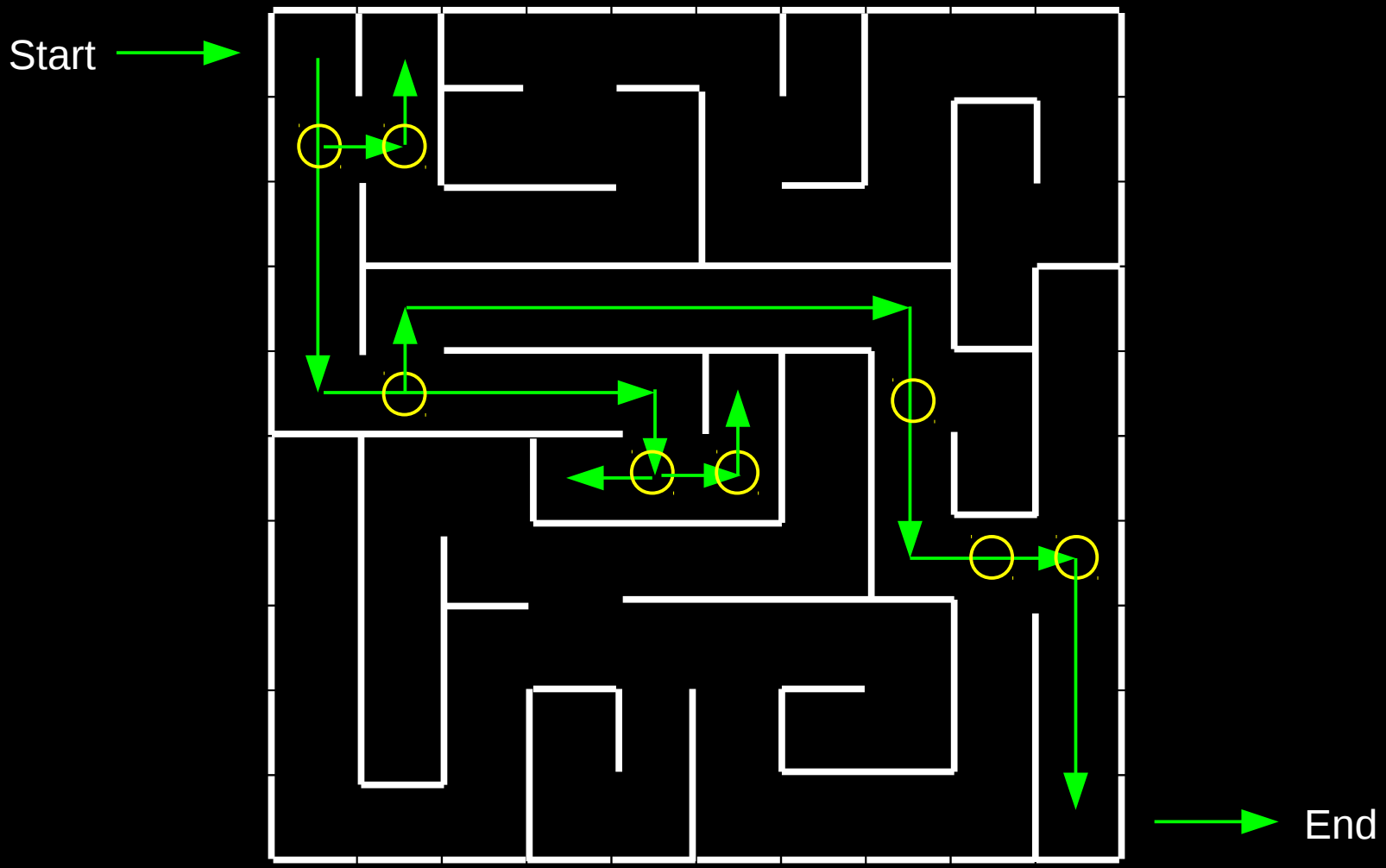
Parallel Work Queue (PWQ)



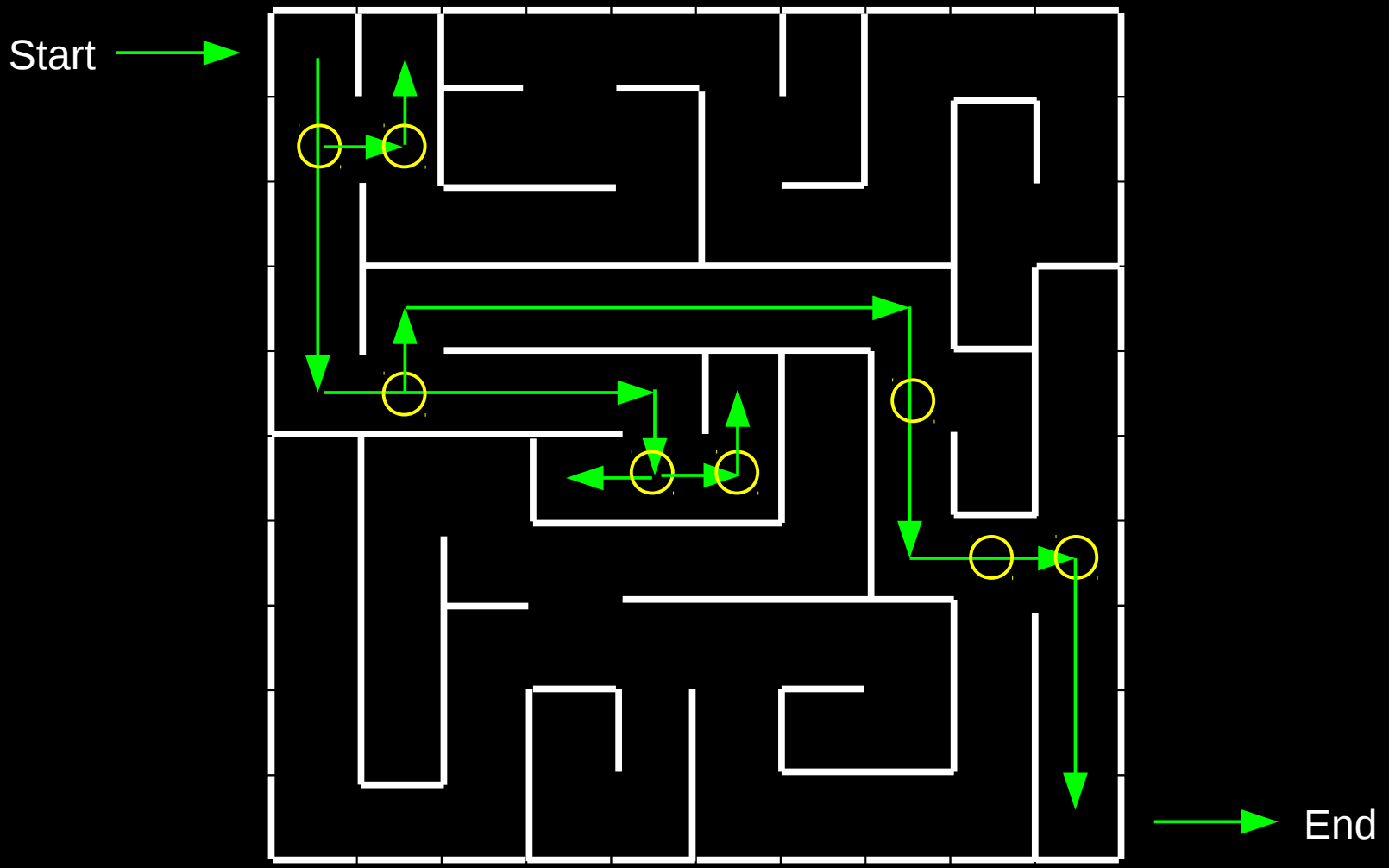
Parallel Work Queue



Parallel Work Queue



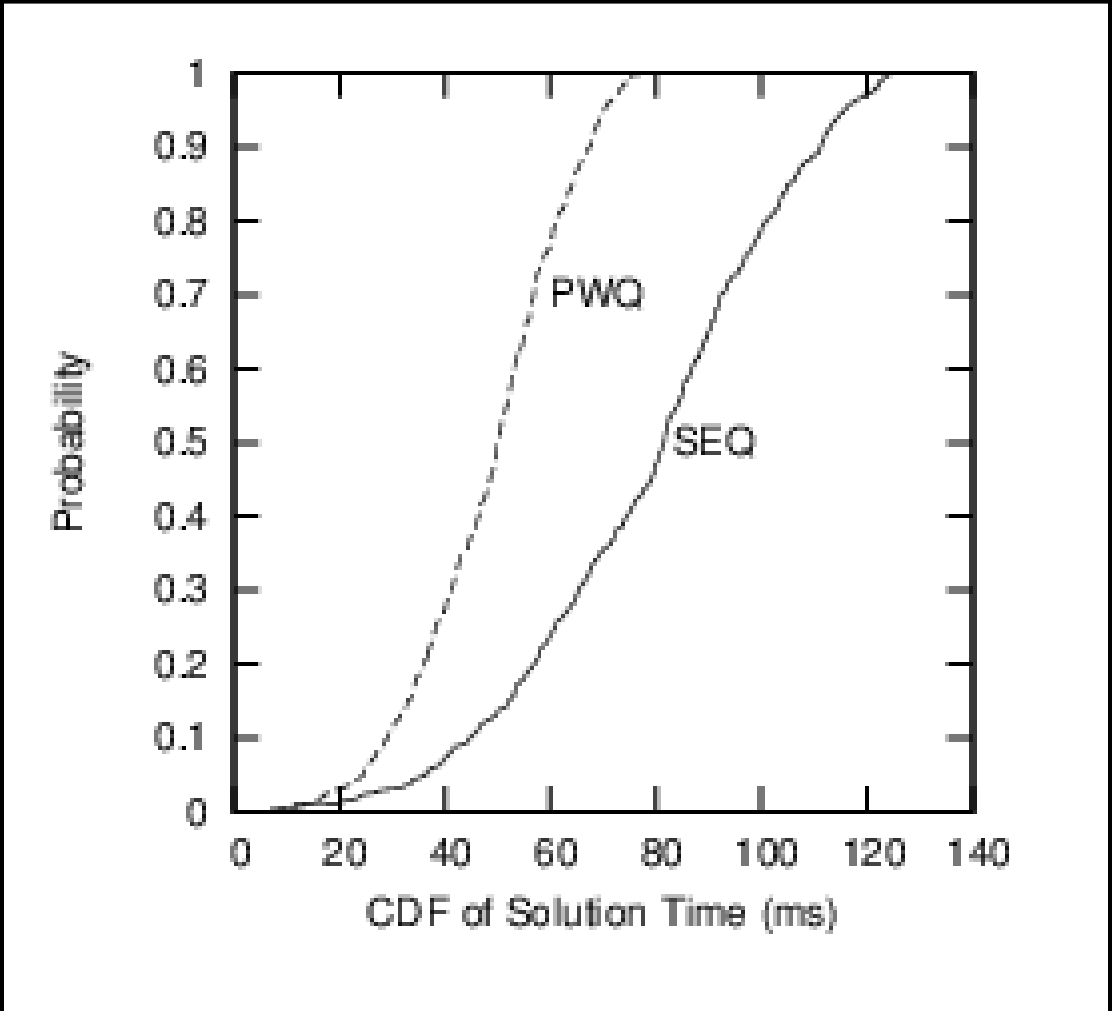
Parallel Work Queue: Saved An Iteration!!!



But can you see the weak point?

Performance Comparison: PWQ vs. SEQ

Performance Comparison: PWQ vs. SEQ (Two Threads)

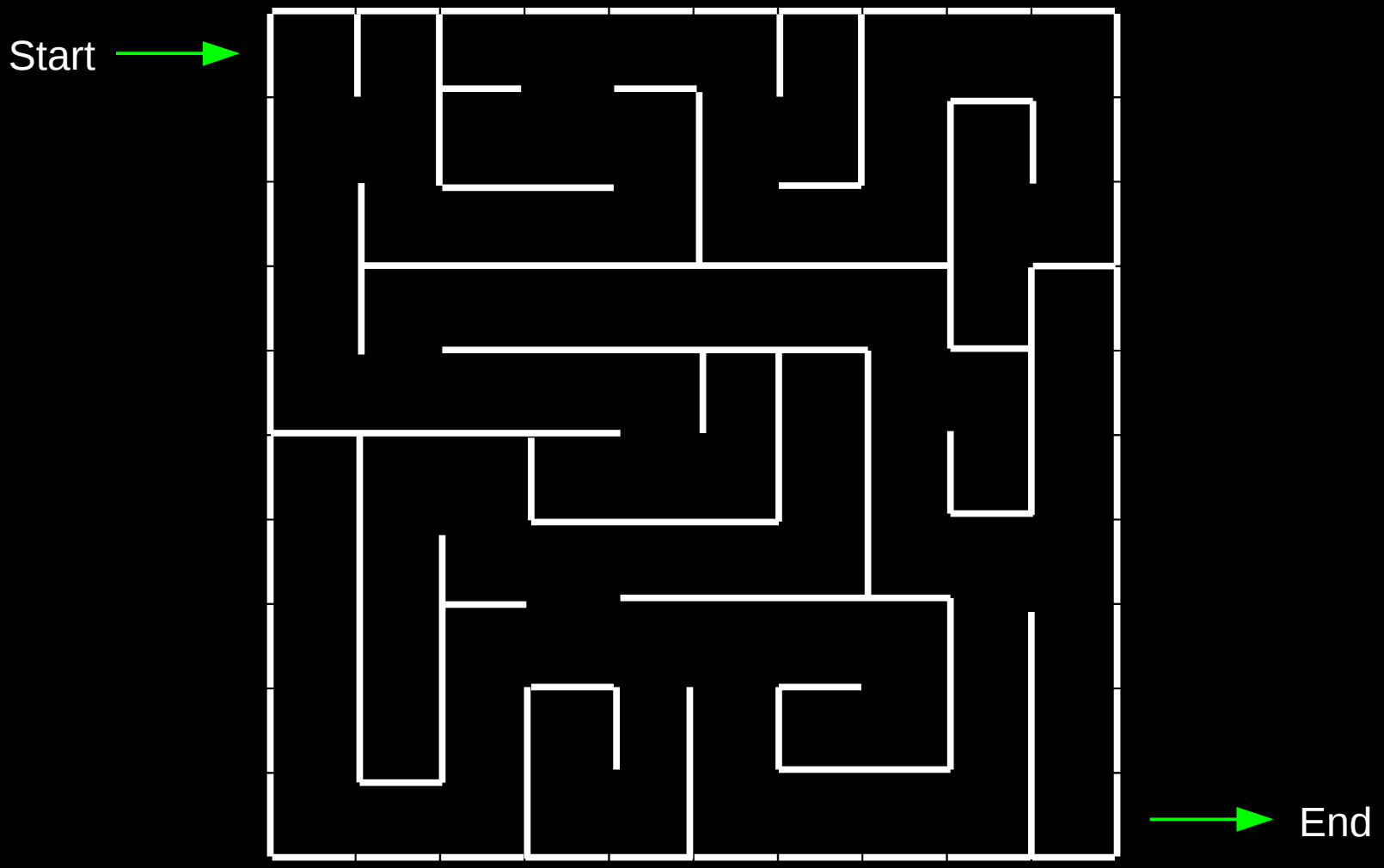


Everything I Need to Know, I Learned in Kindergarten

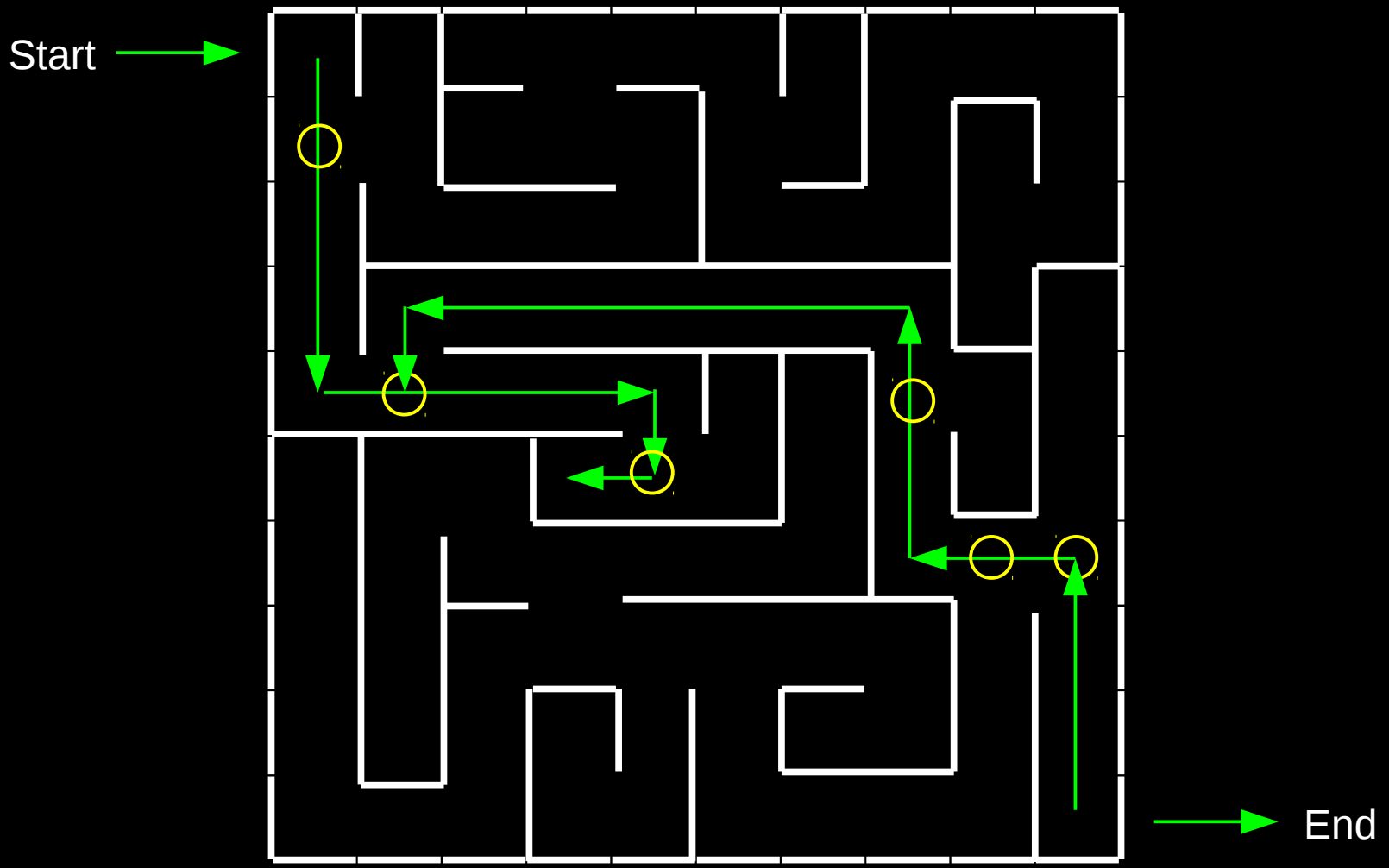
Everything I Need to Know, I Learned in Kindergarten

- In this case, when solving a maze, start at both ends!!!

Partitioned Parallel Solution (PART)

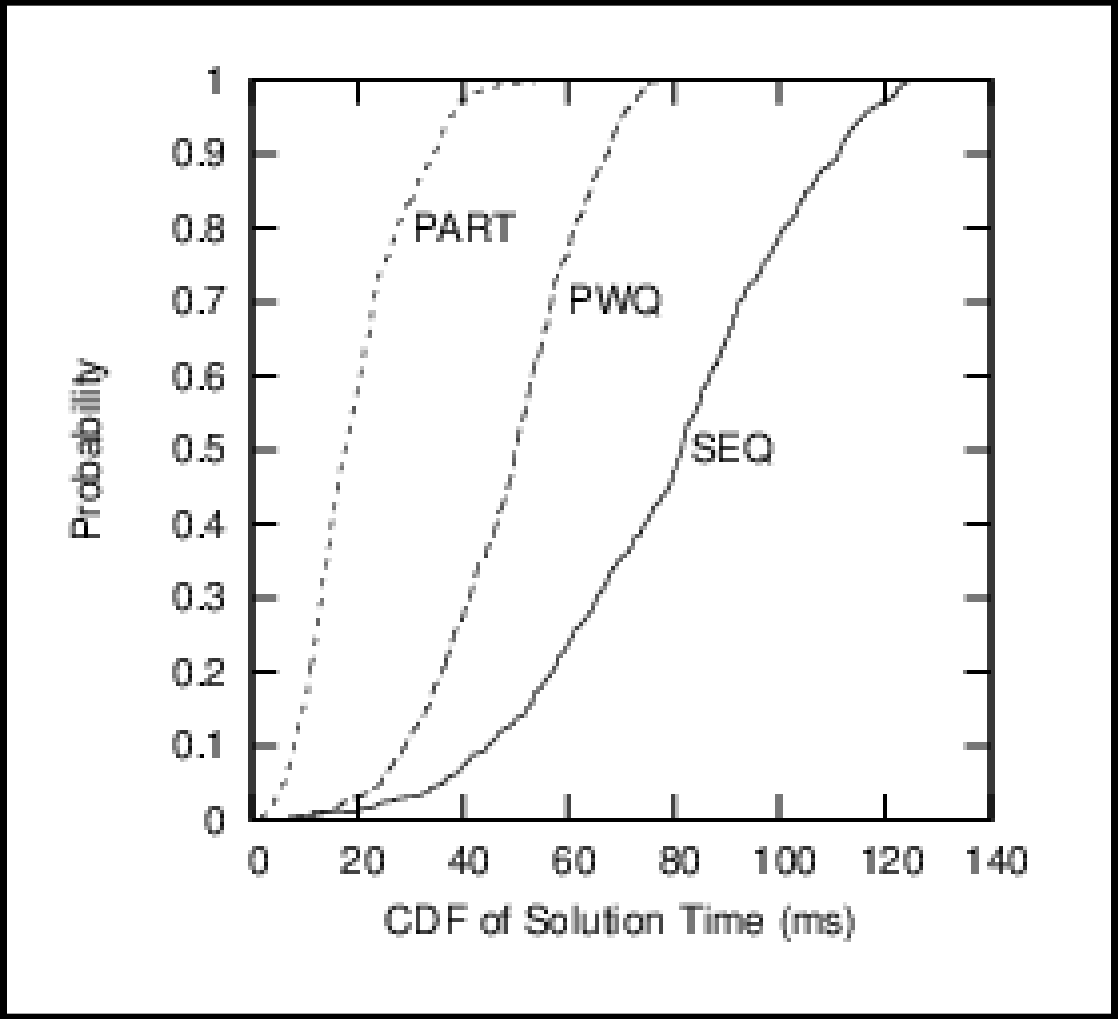


Partitioned Parallel Solution



Performance Comparison: SEQ vs. PWQ vs. PART

Performance Comparison: SEQ vs. PWQ vs. PART: Two Threads



Lots of overlap – are these really different???

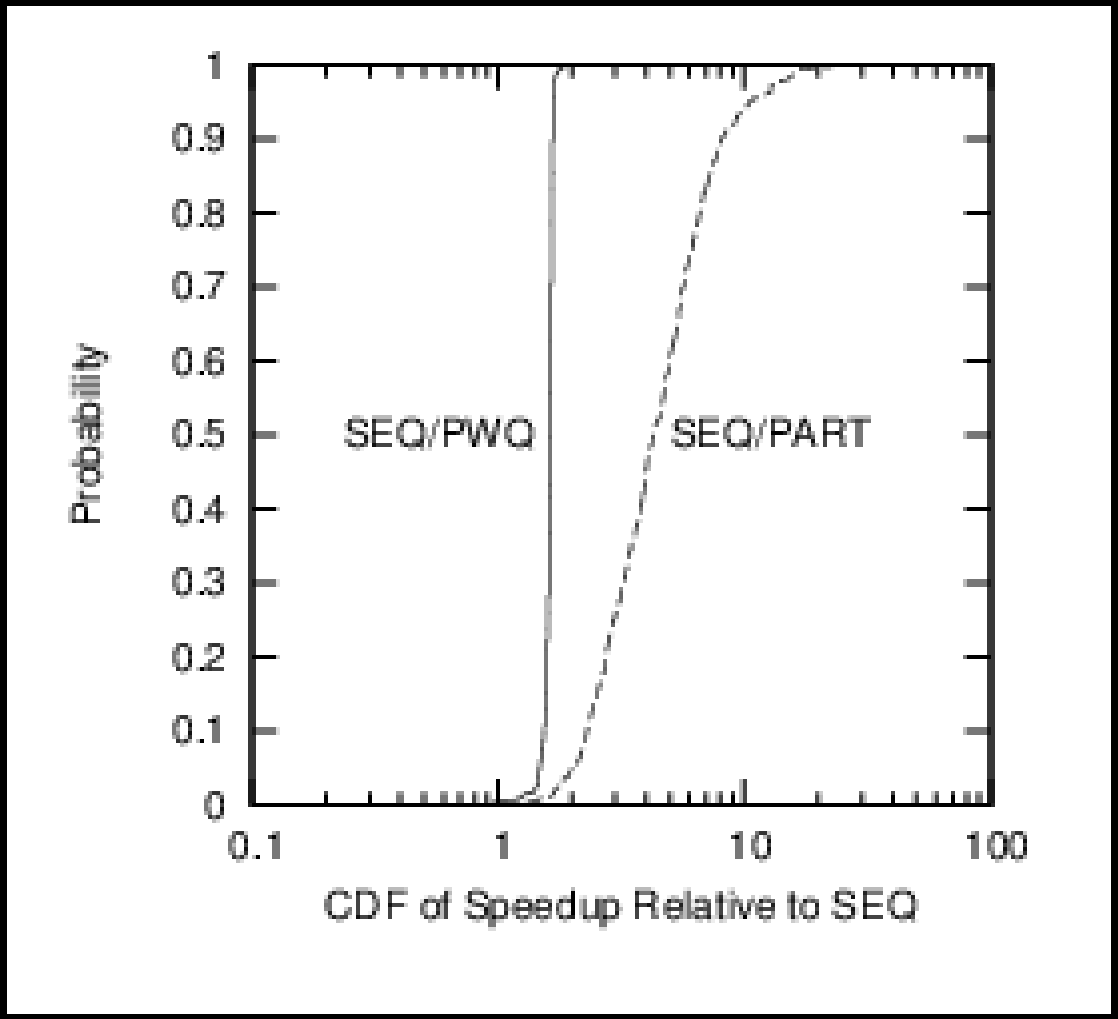
Performance Comparison: SEQ vs. PWQ vs. PART

- The CDFs assume independence
- This is not true: data is highly correlated
  - Test script generates a maze, then runs all solvers on that same maze
  - CDFs lose the relationship between those solutions

Performance Comparison: SEQ vs. PWQ vs. PART

- The CDFs assume independence
- This is not true: data is highly correlated
  - Test script generates a maze, then runs all solvers on that same maze
  - CDFs lose the relationship between those solutions
- Preserve this relationship by taking CDF of ratios
  - SEQ/PWQ and SEQ/PART

Performance Comparison: SEQ/PWQ vs. PWQ/PART: Two Threads



Anything odd about this graph?

What is Going on Here???

- Median speedup of 4x on only two threads!!!
- Individual data points show speedups of up to ***40x***!!!
- This is not merely embarrassingly parallel
  - **Embarrassingly parallel:** Adding threads does not significantly increase the aggregate amount of work, resulting in linear scaling

What is Going on Here???

- Median speedup of 4x on only two threads!!!
- Individual data points show speedups of up to ***40x***!!!
- This is not merely embarrassingly parallel
  - **Embarrassingly parallel:** Adding threads does not significantly increase the aggregate amount of work, resulting in linear scaling
- This is ***humiliatingly parallel***

What is Going on Here???

- Median speedup of 4x on only two threads!!!
- Individual data points show speedups of up to ***40x***!!!
- This is not merely embarrassingly parallel
  - **Embarrassingly parallel:** Adding threads does not significantly increase the aggregate amount of work, resulting in linear scaling
- This is ***humiliatingly parallel***
  - **Humiliatingly parallel:** Adding threads significantly *decreases* the aggregate amount of work, resulting in superlinear scaling

What is Going on Here???

- Median speedup of 4x on only two threads!!!
- Individual data points show speedups of up to ***40x***!!!
- This is not merely embarrassingly parallel
  - **Embarrassingly parallel:** Adding threads does not significantly increase the aggregate amount of work, resulting in linear scaling
- This is ***humiliatingly parallel***
  - **Humiliatingly parallel:** Adding threads significantly *decreases* the aggregate amount of work, resulting in superlinear scaling
- Yeah, yeah, it is great to have a definition, but how is this happening???

What is Going on Here???

- First assumption: there is a bug in either the solver or the data-reduction scripts
  - There probably still is, but the solutions and times checked out

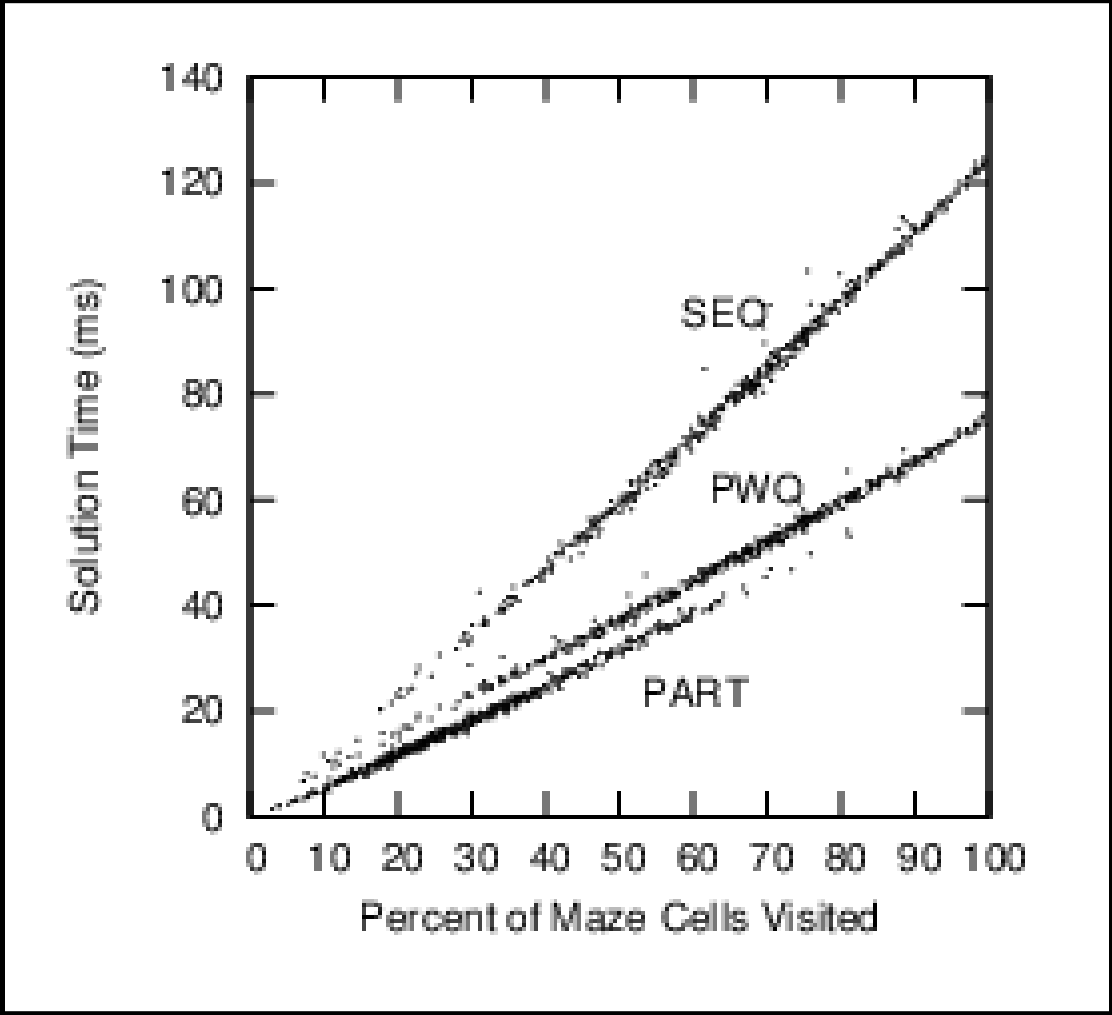
What is Going on Here???

- First assumption: there is a bug in either the solver or the data-reduction scripts
  - There probably still is, but the solutions and times checked out
- The solver also prints the fraction of cells visited
  - SEQ and PWQ never visited fewer than 9% for 500x500 maze

What is Going on Here???

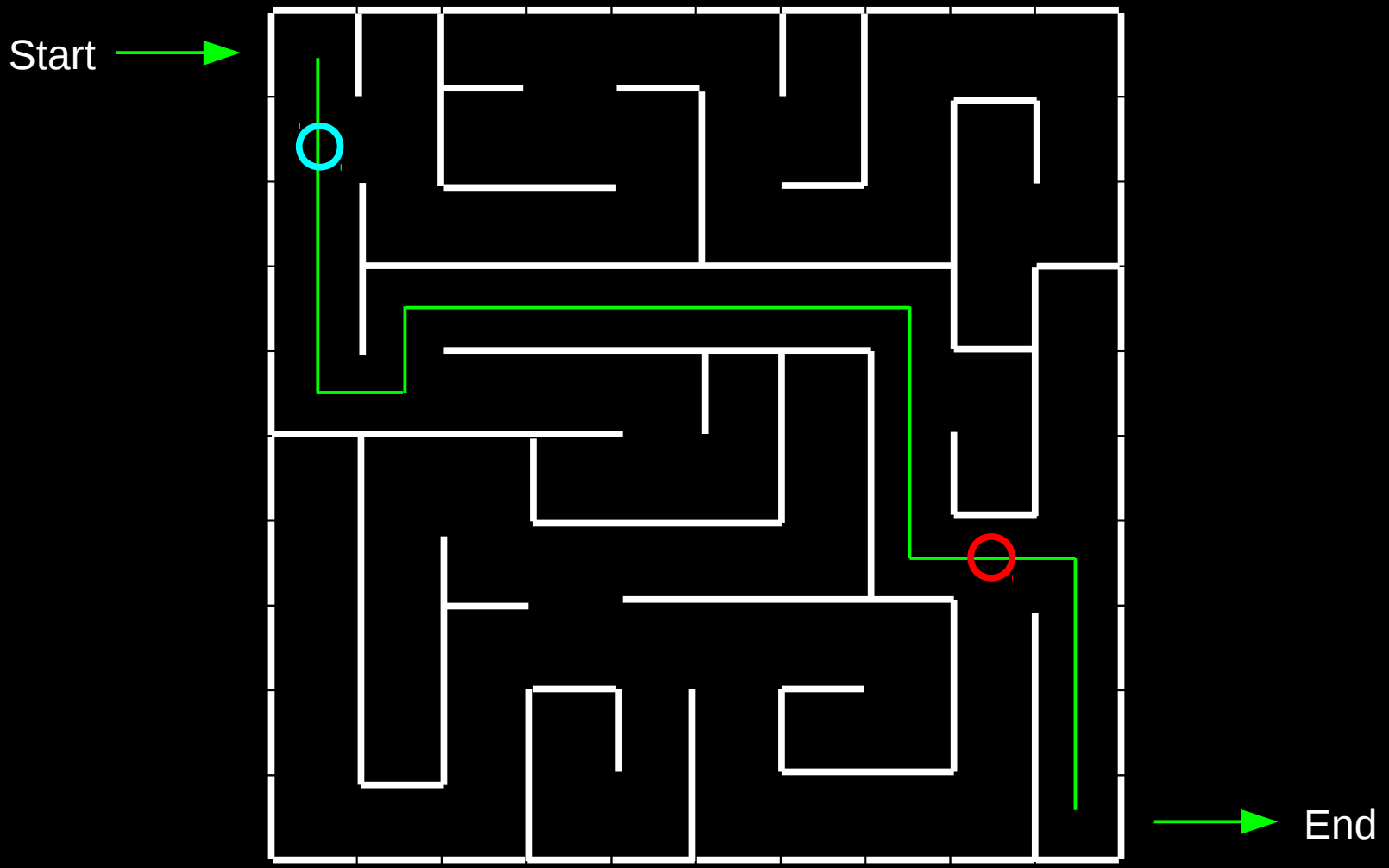
- First assumption: there is a bug in either the solver or the data-reduction scripts
  - There probably still is, but the solutions and times checked out
- The solver also prints the fraction of cells visited
  - SEQ and PWQ never visited fewer than 9% for 500x500 maze
  - But PART sometimes visited fewer than 2%!!!

Visit Fraction vs. Solution Time Correlation

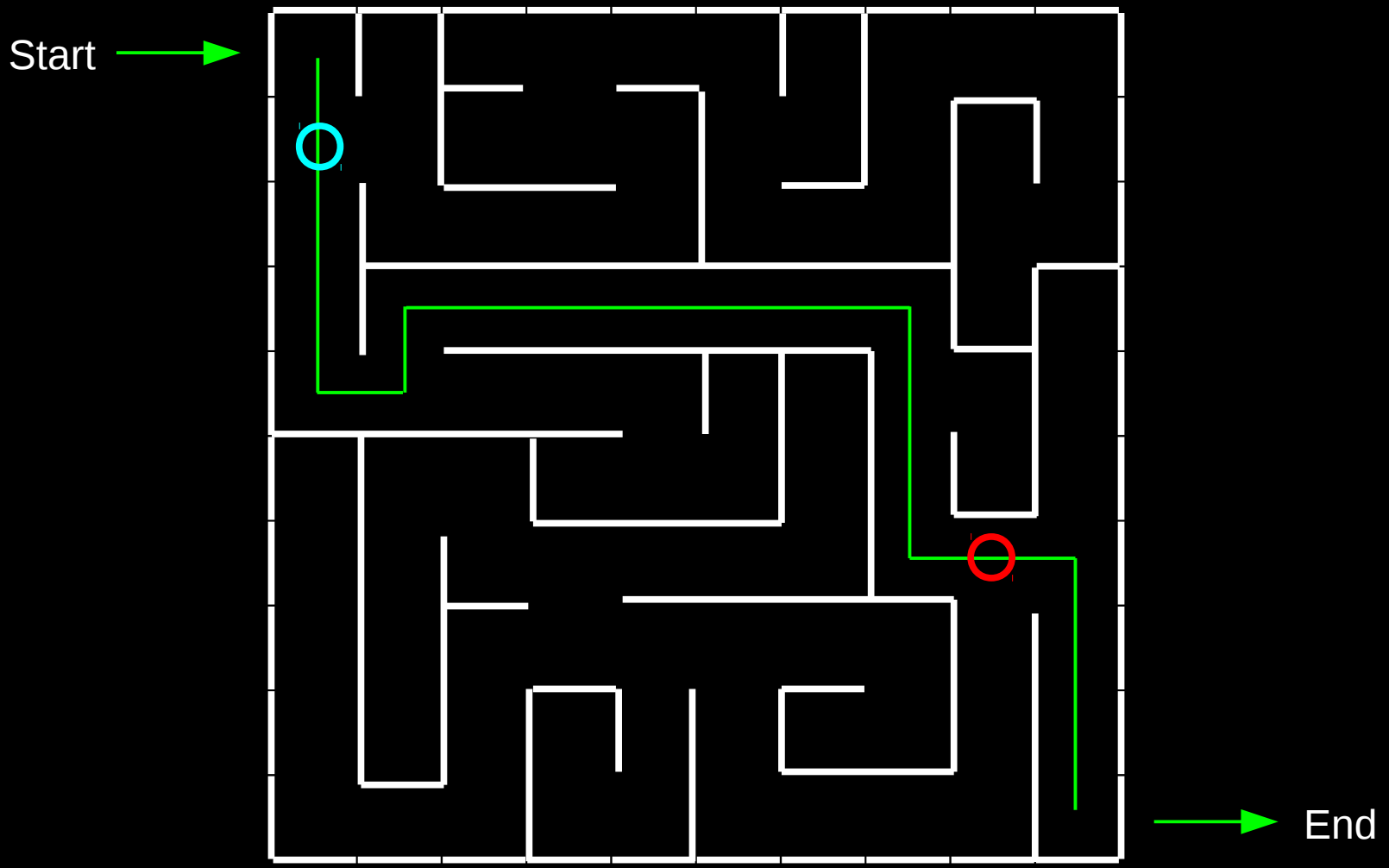


But correlation is not causation, nor is it “why”...

Partitioned Parallel Solution

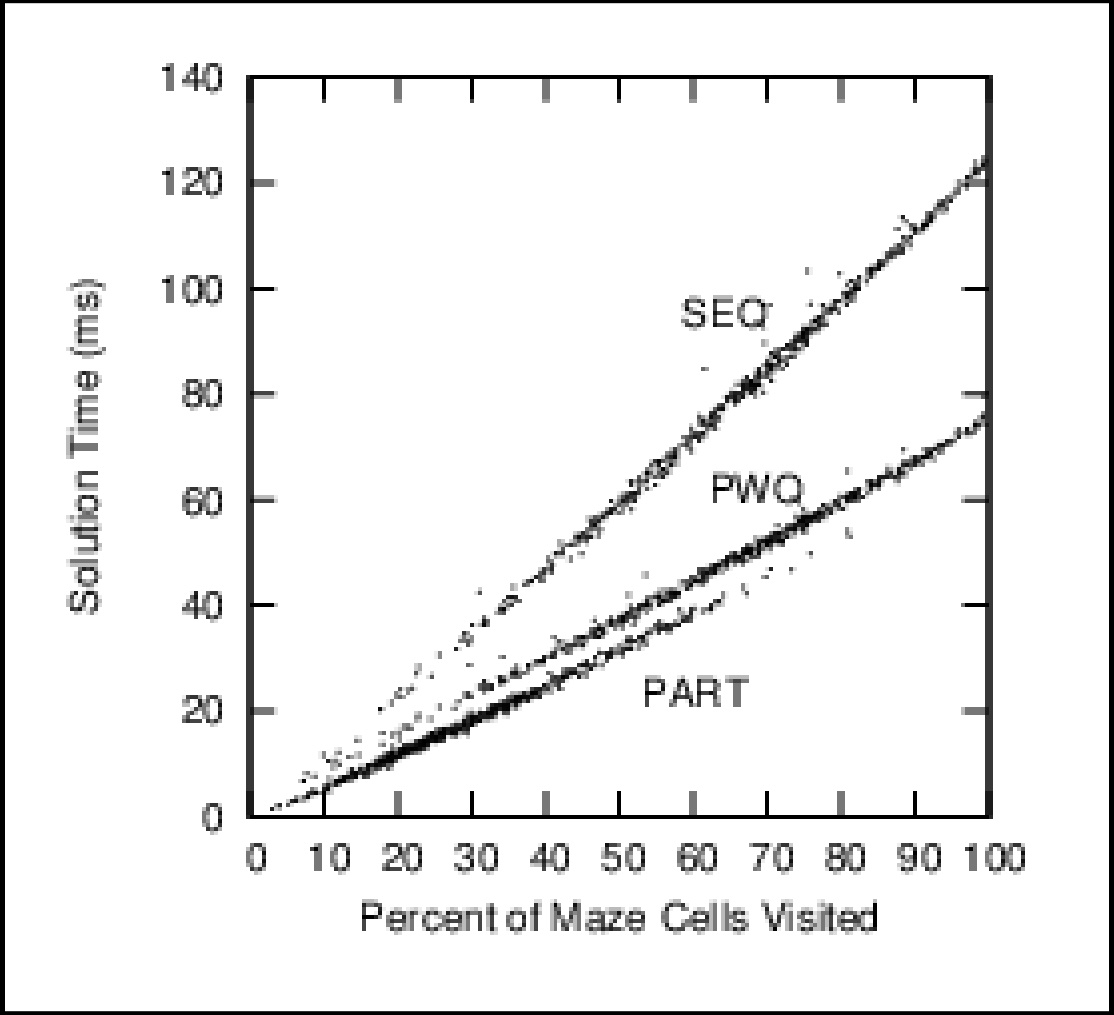


Partitioned Parallel Solution

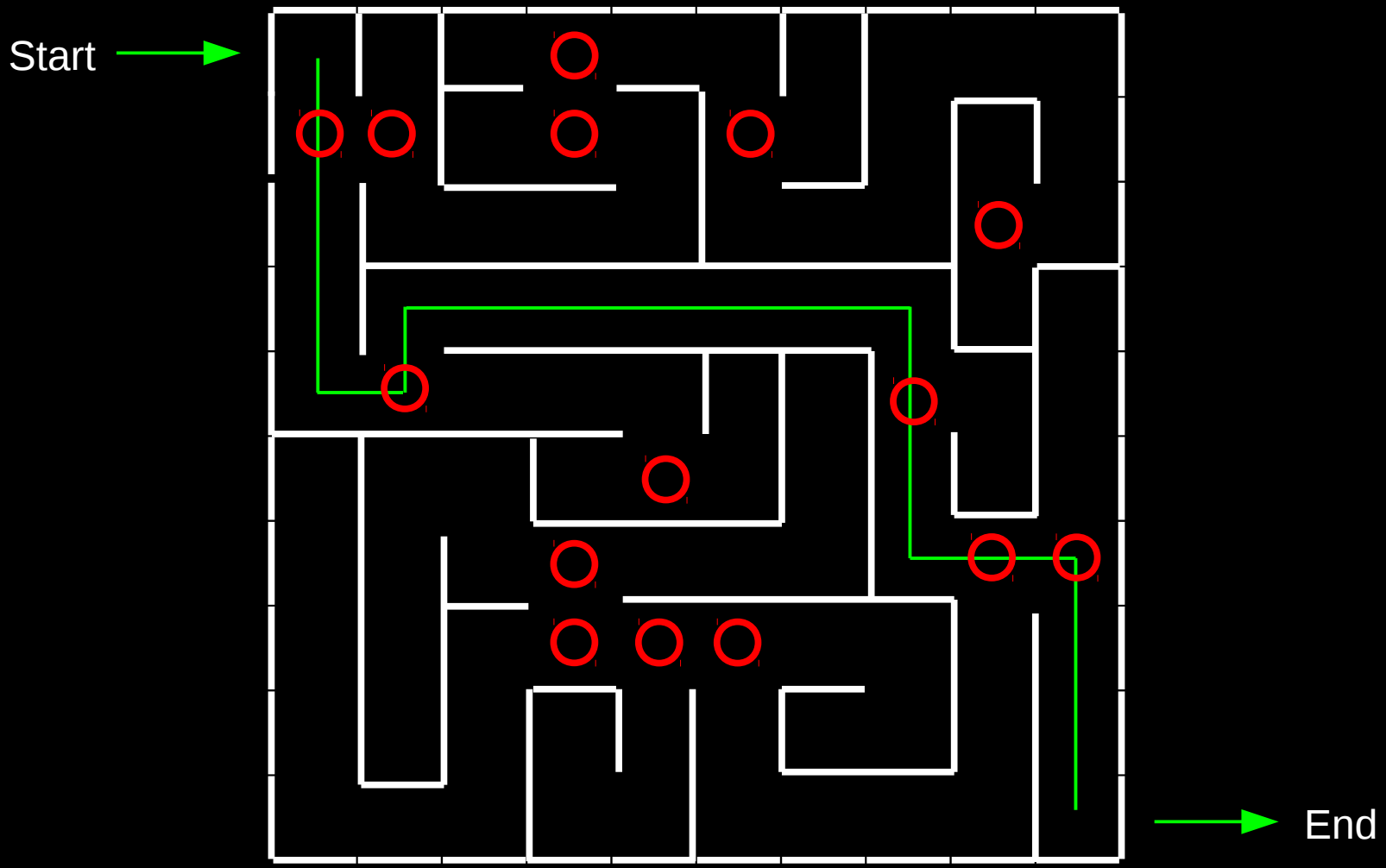


The threads get in each others' way!

But Why The Separation Between PWQ and PART?

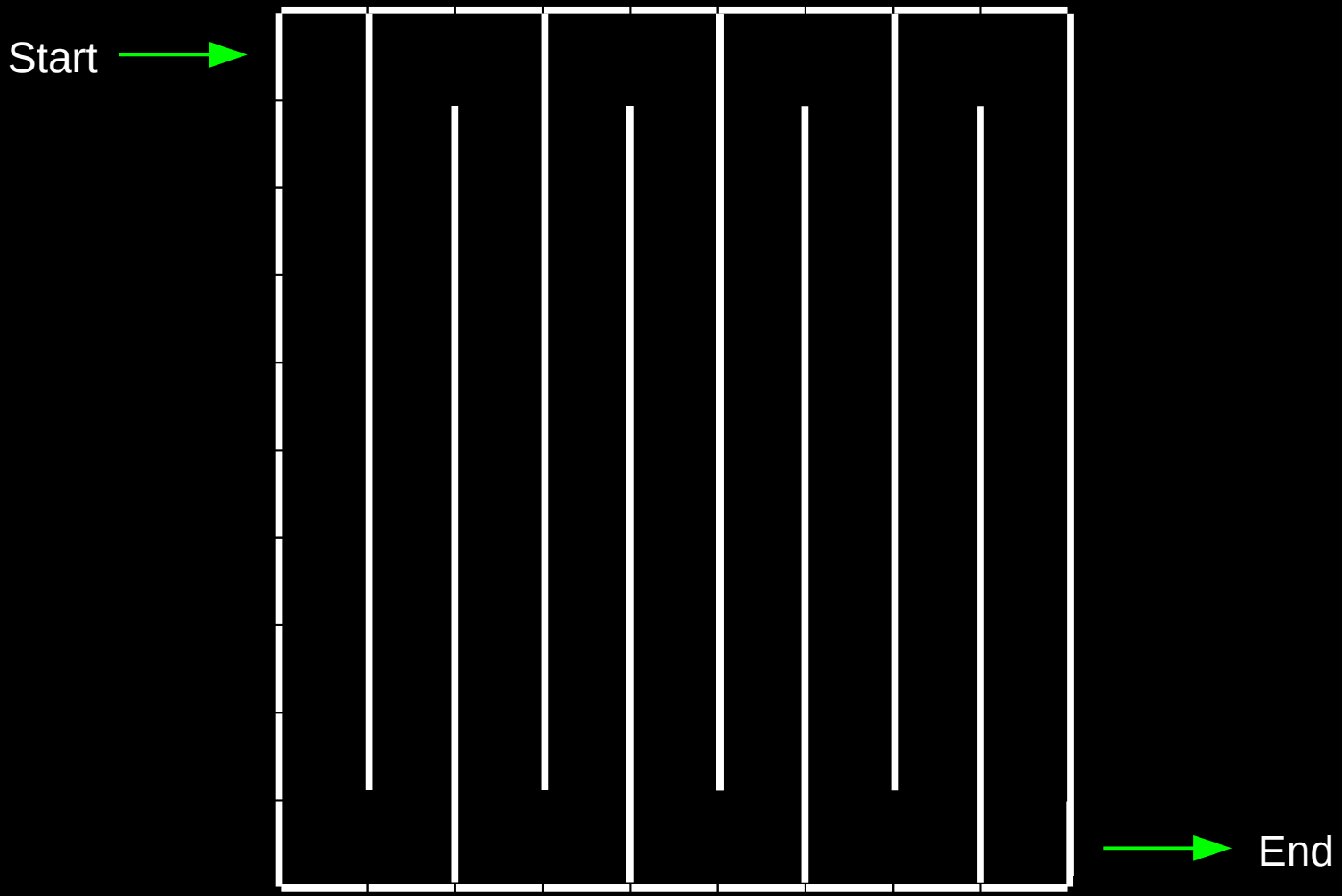


PWQ Has Many Potential Contention Points: Contention is Expensive



Does PART Always Achieve Humiliating Parallelism?

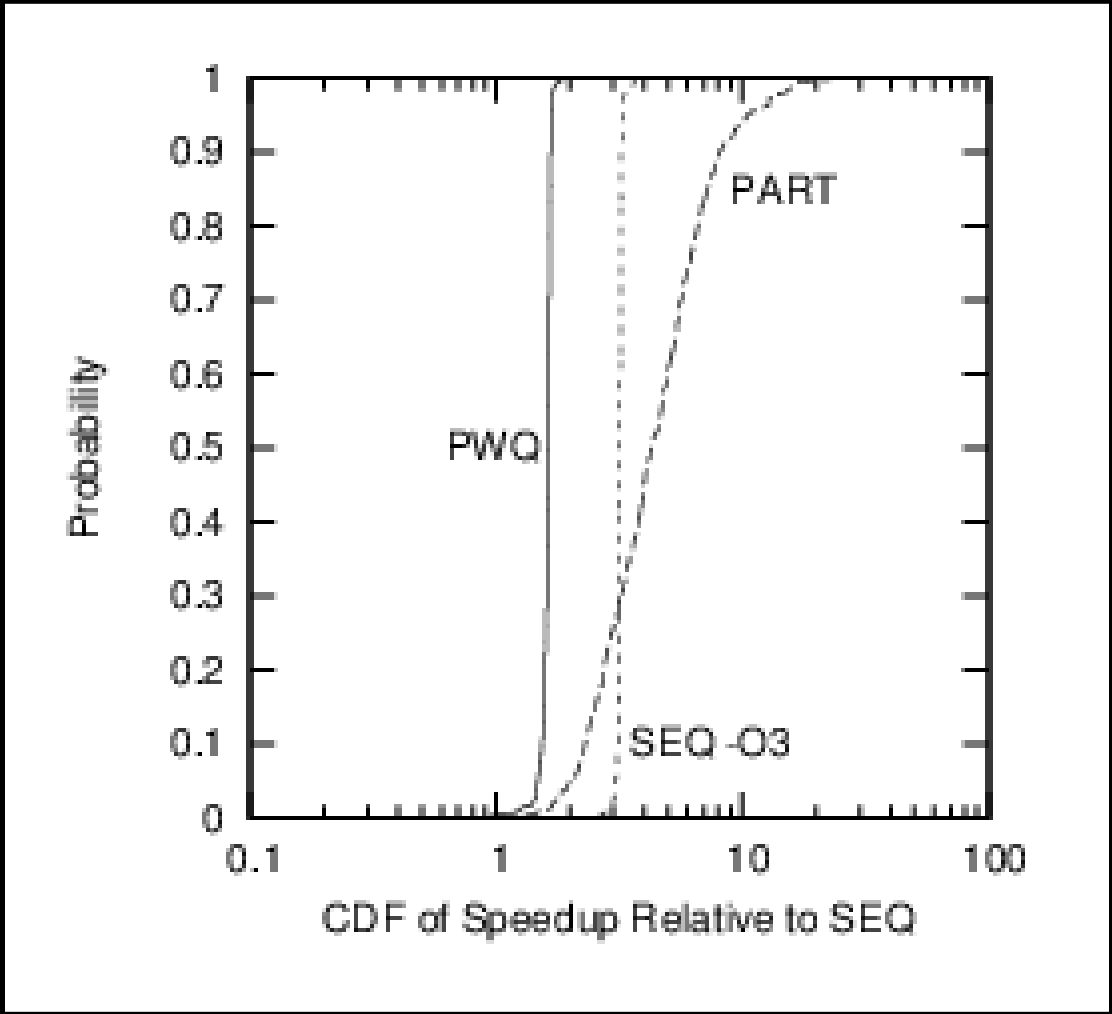
Does PART Always Achieve Humiliating Parallelism?



Partitioning is a Powerful Parallelization Tool

Partitioning is a Powerful Parallelization Tool
But Let's Not Forget Sequential Optimizations!!!

Partitioning is a Powerful Parallelization Tool
But Let's Not Forget Sequential Optimizations!!!



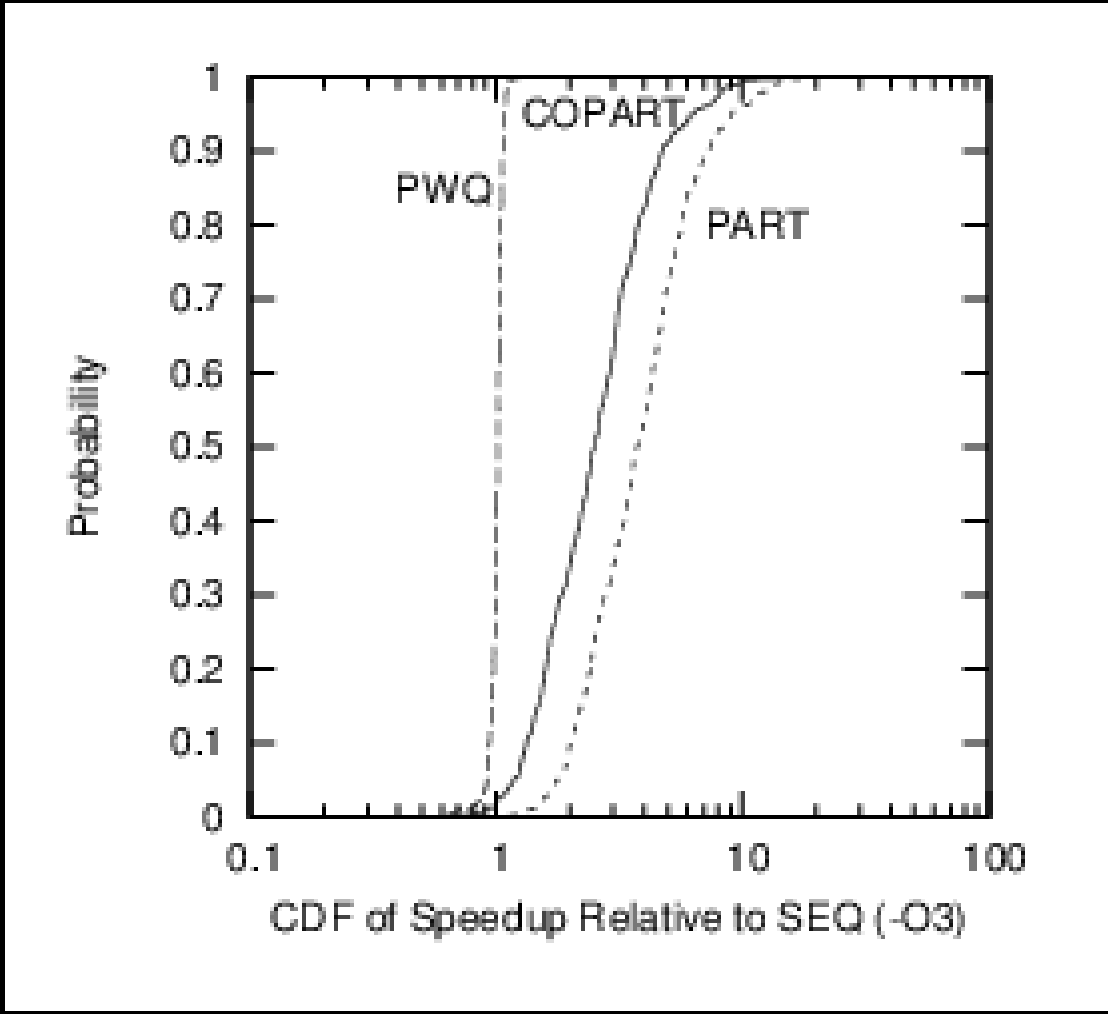
-O3 much better than PWQ, almost as good as PART!

Compiler Optimizations Beat PWQ!!!

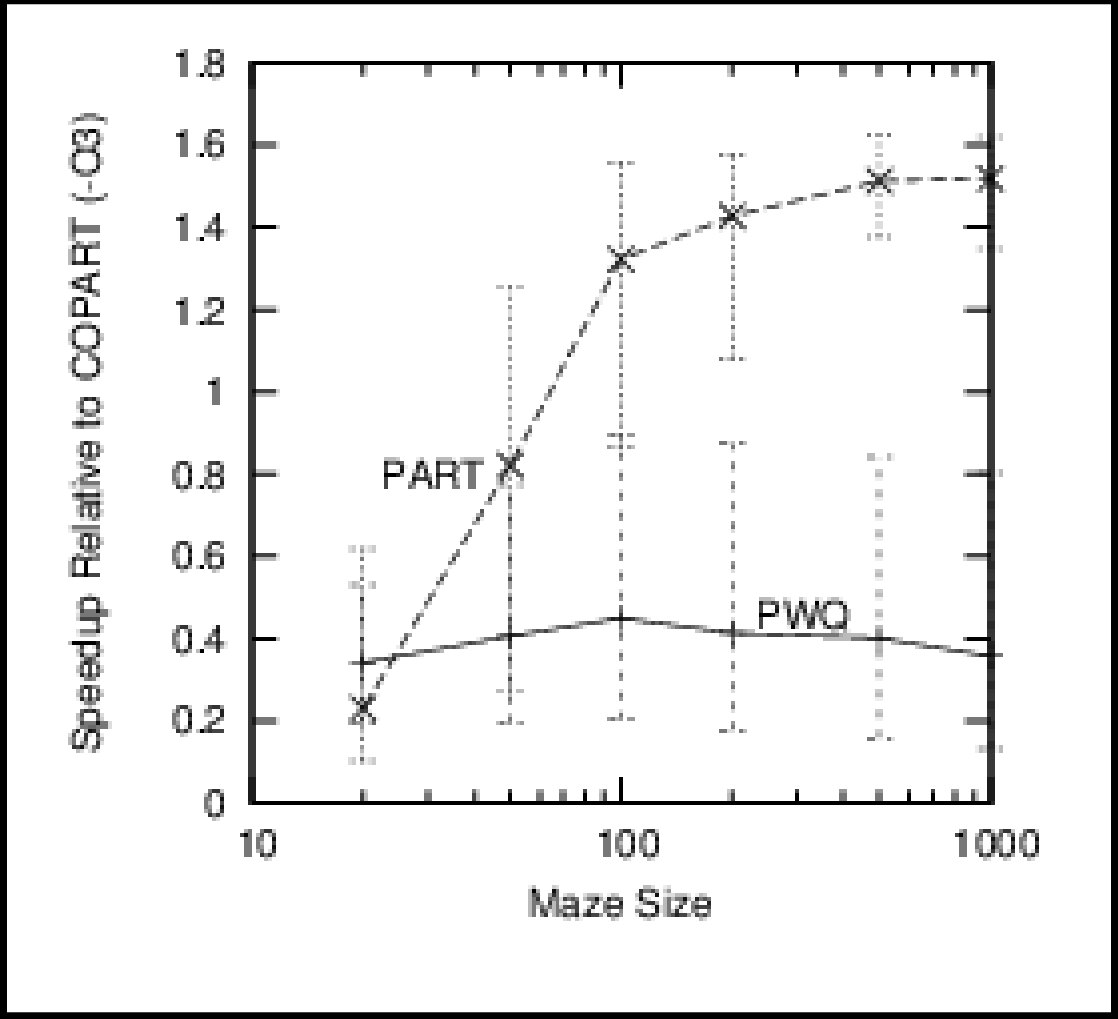
- Yes, PART is even better, but if all you need is a 2x improvement (rather than optimality), compiler optimization is an extremely attractive option
- These results indicate that parallel-programming research making use of high-level/overhead languages is vulnerable to invalidation given improvements in optimization

And The Threads Will Get In Each Other's Way Even If They Are Running on One CPU... (Coroutines!!!)

And The Threads Will Get In Each Other's Way Even If They Are Running on One CPU... (Coroutines!!!)

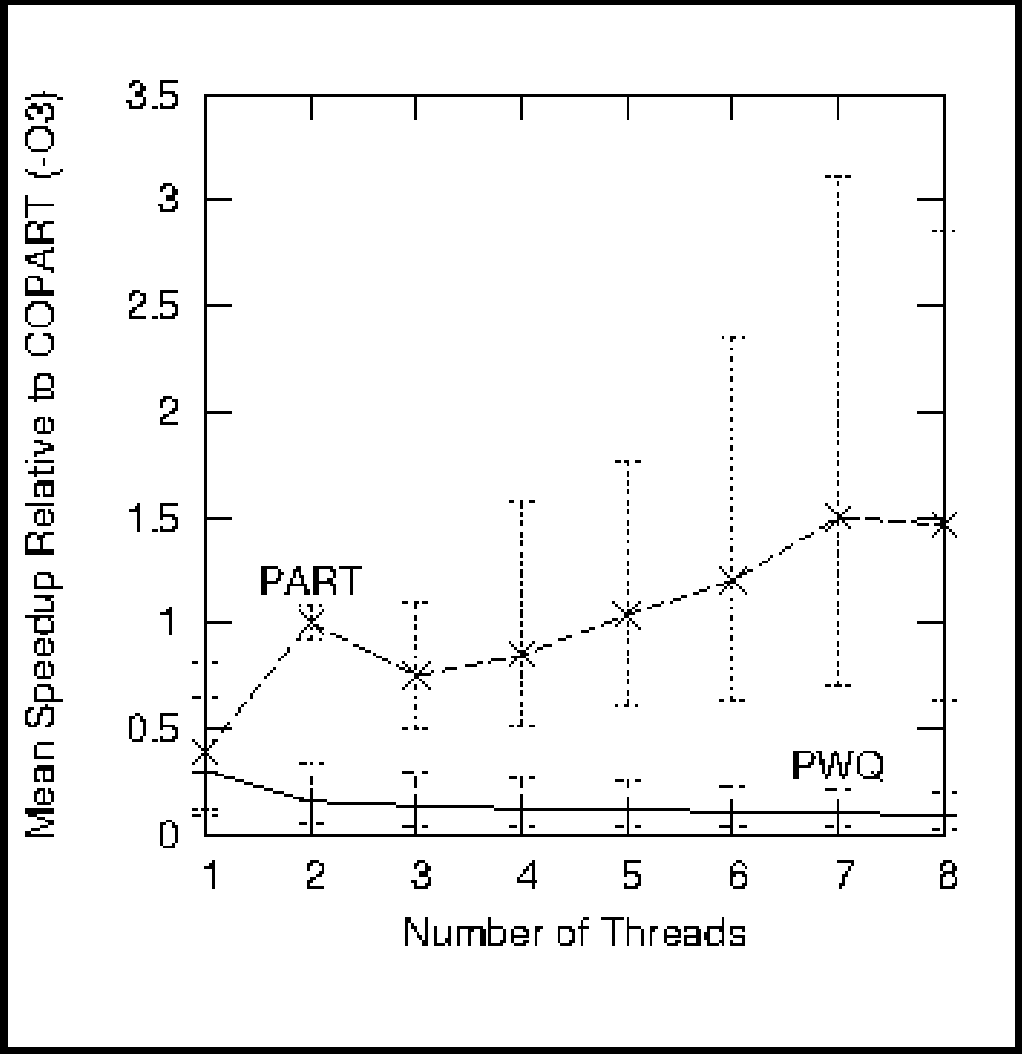


Effect Of Maze Size



Back to merely modest speedups!

Effect Of Increasing Numbers of Threads



Larger, older, less tightly integrated HW: Smaller speedups

Summary and Conclusions

How Did I Do Against My Goals?

How Did I Do Against My Goals?

- ~~An example of near-perfect partitioning for "Is Parallel Programming Hard, And If So, What Can You Do About It"~~
  - Not so good!
  - From modestly scalable to humiliatingly parallel and back again
- ~~Use case for RCU-protected union-find data structure~~
  - Not so good!
  - No need for RCU in this problem

How Did I Do Against My Goals?

- ~~An example of near-perfect partitioning for "Is Parallel Programming Hard, And If So, What Can You Do About It"~~
  - Not so good!
  - From modestly scalable to humiliatingly parallel and back again
- ~~Use case for RCU-protected union-find data structure~~
  - Not so good!
  - No need for RCU in this problem

- On the other hand, this problem turned out to be interesting in its own unexpected way!
  - And a nice change of pace from Linux kernel's RCU implementation

Open Questions

- Can other human-maze-solver techniques be applied?
  - Follow walls to exclude portions of maze
  - Choosing internal starting points based on traversal
- Do these results apply to unsolvable or cyclic mazes?
- Do other problems exhibit humiliating parallelism?
- Does humiliating parallelism always lead to a more-efficient sequential solution?
- How much current parallel-programming research can stand up to improved optimization?

Open Questions

- Can other human-maze-solver techniques be applied?
  - Follow walls to exclude portions of maze
  - Choosing internal starting points based on traversal
- Do these results apply to unsolvable or cyclic mazes?
- Do other problems exhibit humiliating parallelism?
- Does humiliating parallelism always lead to a more-efficient sequential solution? (No, it does not.)
- How much current parallel-programming research can stand up to improved optimization?

Conjecture

- Conjecture (Due to Jon Walpole):
  - Thinking from a parallel perspective leads to a much more efficient search strategy.
  - It is not the parallelism of the implementation that is important, but rather the parallelism of the strategy.

Parting Words of Advice

- Apply parallelism as a first-class optimization technique
  - Apply at as high a level as possible, to full application
  - Often simplifies solution
  - Usually reduces synchronization overhead, thereby improving both performance and scalability
- In contrast, retrofitted parallelism is likely to be grossly suboptimal
  - Especially when applied as a low-level after-the-fact optimization
  - Might be OK in some situations, but we can do much better

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.

Questions?