Minimum Blocking Parallel Bidirectional Search

PDPTA'12 Session 5

Dale Parson and Dylan Schwesinger

Kutztown University of PA

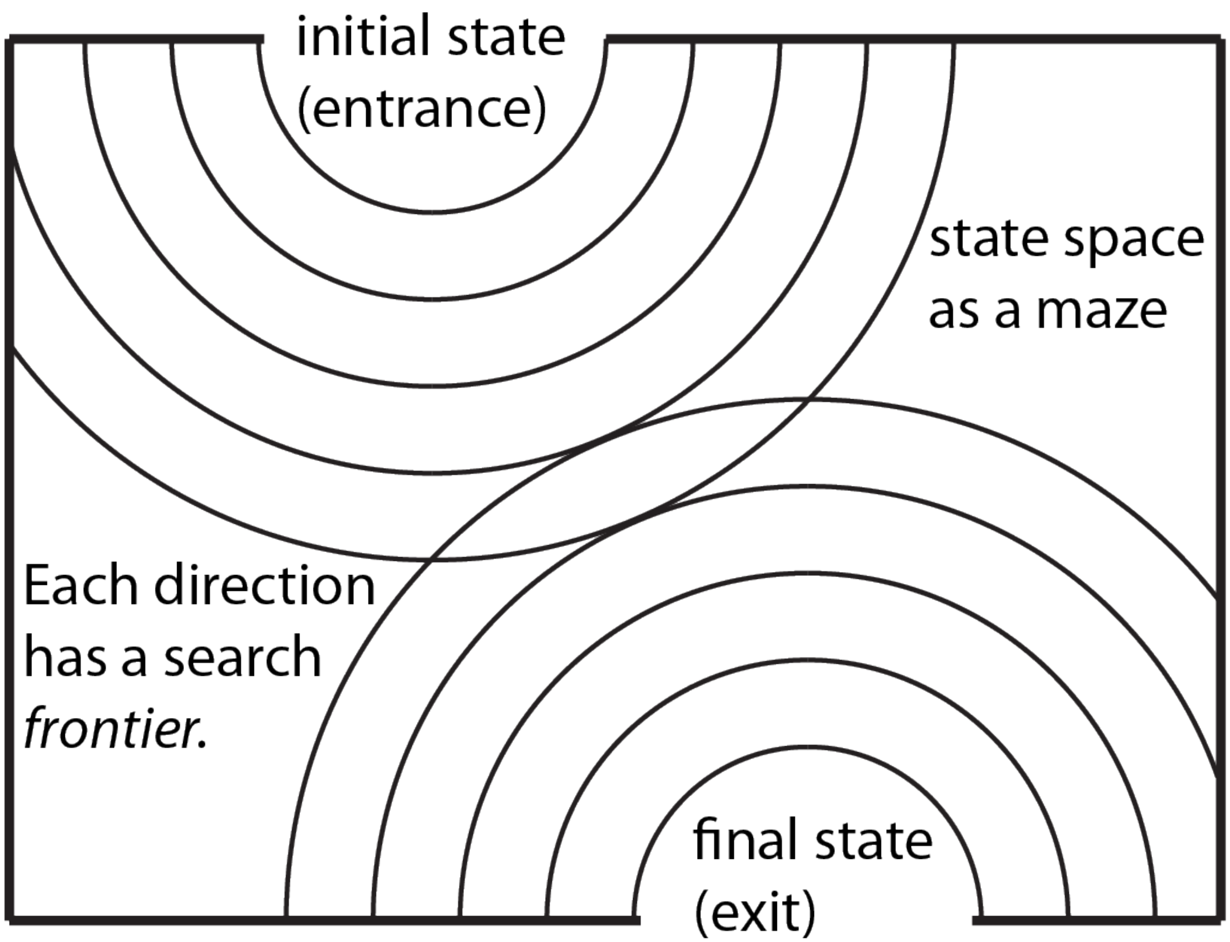
http://faculty.kutztown.edu/parson

Acknowledgements to Sun, NVIDIA, Intel, and PA State System of Higher Education

Bidirectional Search

- Initial state is known.
- Final state is known.
- Path connecting them is unknown.
- Worst case time, space for breadth-first unidirectional search is $O(b^d)$.
  - **b** is branching factor, **d** is depth.
- For bidirectional it is $O(b^{d/2})$.

Searches meet at the *frontier*



Data Structures Required

- Breadth-first maintains a queue of states to expand.
    - Each expansion step dequeues one state, computes up to **b** expansions, and enqueues them.
    - A state maintains a back-link to its predecessor.
    - It compares each expansion to the final state.
- Bidirectional adds two sets of states, those reached via forward search and those reached via backward search.

Related Work

- Related work on multiprocessing of bidirectional search focuses on parallel implementation of heuristic-based pruning of the search space.
- Our work is orthogonal -- heuristic-based pruning is supported, but it is not our focus. Focus is on finding most effective approach to algorithm / data structures for bidirectional search.

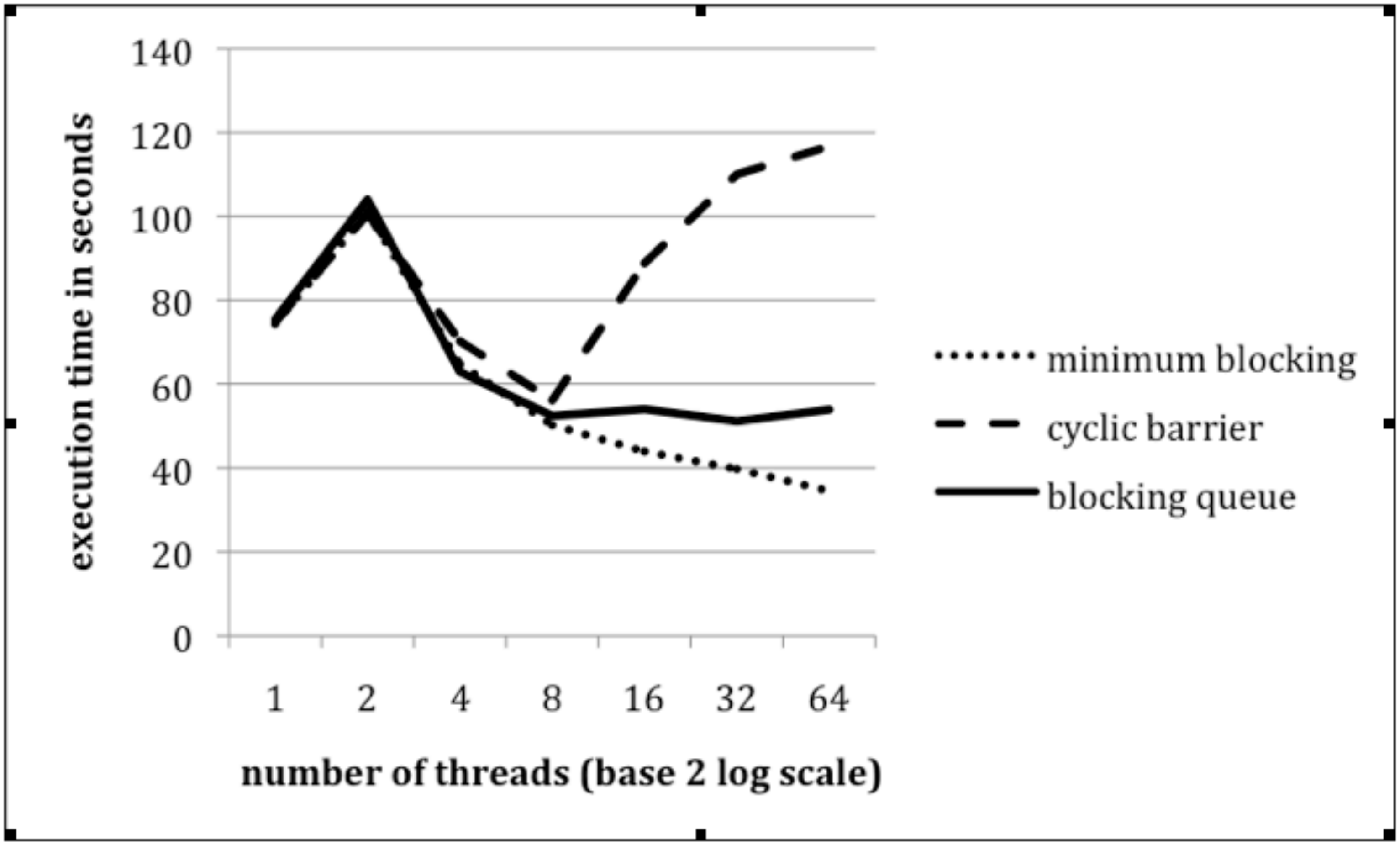
MIMD Multiprocessing

- Our initial attempt used a CyclicBarrier to restrict all threads to one side's frontier.
    - Two-phase state machine.
    - Dequeue a state, if it is a reversal of direction then wait in the CylicBarrier until all threads enter this barrier.
    - Otherwise, compute its expansions. If an expansion is in this side's set, discard it. If an expansion is in the other side's set, it is a solution. Otherwise, add it to this side's set and to the state queue.
- Java's ConcurrentLinkedQueue uses no locks. ConcurrentHashMap uses minimal write locks.

Non-blocking Version

- It eliminates the CyclicBarrier.
- Threads finish out a frontier and check a volatile *isdone* flag.
- Directions of search may overlap in time.
- Some threads may search a little too deeply after a solution is found.
    - This does not affect correctness nor increase time.
- We also tried a LinkedBlockQueue approach.

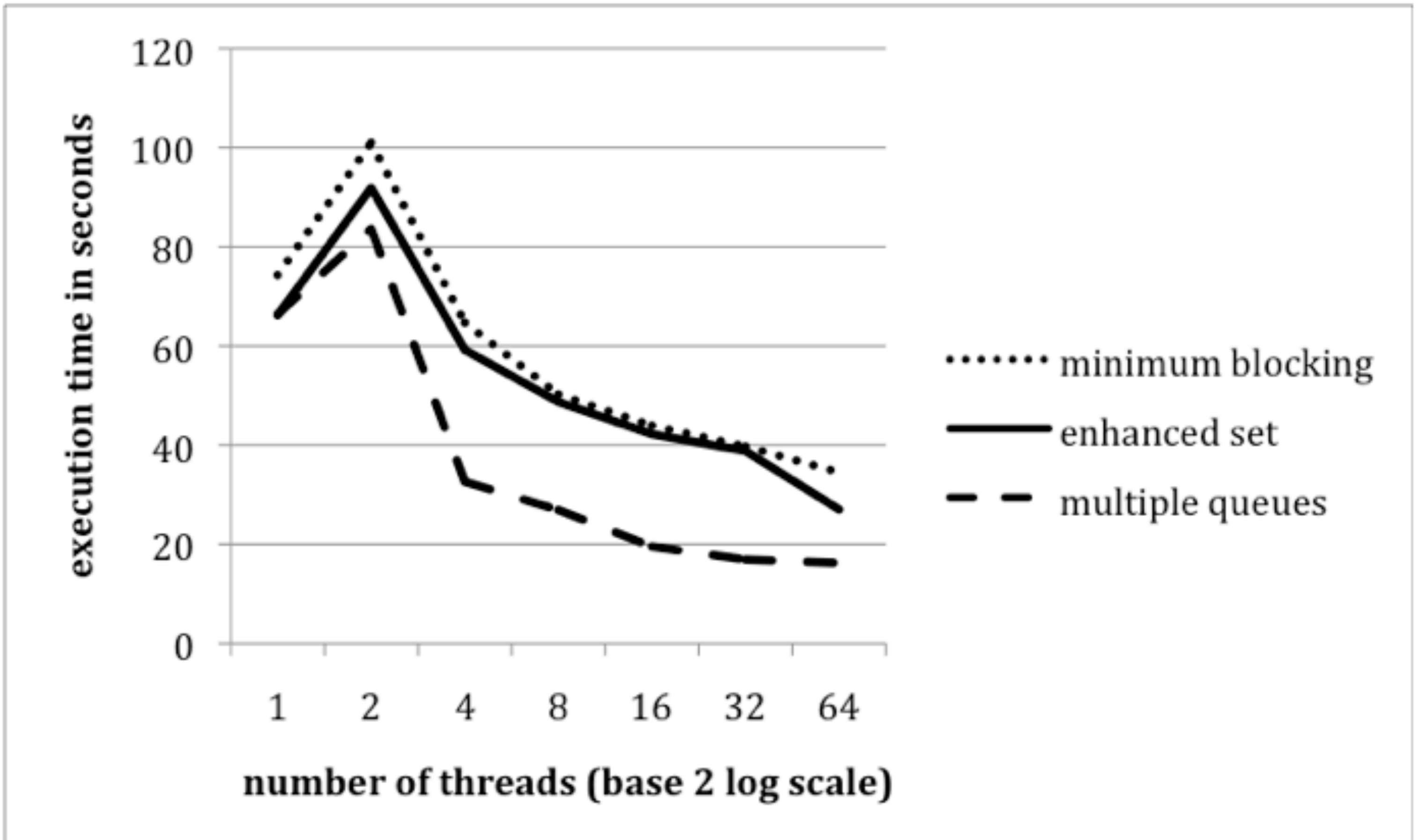
8 core x 8 thread Sparc server, basic algorithms



Algorithm Enhancements

- Initialize each ConcurrentHashMap to its maximum size of 4 million elements.
- Reduce load factor from .75 to .5.
- Increase lock stripes from 16 to 128.
- We also tried a variation with each thread getting its own state queue.
    - This eliminates polling read contention.
    - Write contention is minimized by using round-robin order given by an atomic index variable.

Enhanced Results



Subsequent work: C++11

- C++11 port in May / June 2012.
  - Support for atomic operations.
  - Addition of C++ wrappers for POSIX threads, mutexes and condition variables.
- Implemented *open address hash table* using lock free atomic pointers to state objects.
  - Entries are never deleted in bidirectional search.
- Implemented circular buffer queues using atomic "spin locks" on next spot to read (spin on NULL), next-to-write (non-NULL), and tail and head indices.

C++11 and Cuda/C2070 GPU

- C++11 results are slightly better than Java.
  - Static storage allocation of state object space makes another substantial improvement.
  - Deletion of “bad state objects” is avoided by reusing their storage. Allocation is from a first-out queue.
- Cuda did not do so well.
  - Poor spatial and temporal locality of the hash table does not work with Cuda memory & caches well.
  - Doing state expansion on GPU and everything else on MIMD Intel machine on par with MIMD-only approach.

Conclusions

- Use a dataflow architecture that streams states-to-expand to threads via a non-blocking, atomic-based, FIFO queue. Sync on a volatile *isdone* flag.
- Give each thread its own queue, eliminating read contention. Feed the queues in round-robin order to minimize write contention.
- Use minimal locking sets. An open address hash table using atomics needs no locks.
- Pre-allocate everything by determining size growth curves, often empirically.