Minimizing Writes in Parallel External-Memory Search

Nathan Sturtevant and Matthew Rutherford, University of Denver

Resarch Goals

External-memory algorithms have become more prevalent in heuristic search.

Research Question: How can we use parallel and external memory resources to solve or improve performance in large problems?

Paper Goal: Solid State Drives (SSDs) have a limited number of writes before failure. How would you design a parallel external-memory algorithm to mimimize writes?

Result summary: On certain classes of problems, minimizing writes results in significantly improved performance.

General Concepts/Definitions

A ranking/unranking function maps states to and from hash values. A perfect ranking function in a state space with *k* states will map states to the values 0...*k*-1.

An implicit representation represents a state by its address/offset in memory/disk, as opposed to an explicit representation.

We *expand* states to generate their *successors*.

Hard Disk Drives (HDDs) use spinning platters. Sequential access is required to achieve high throughput.

Solid State Drives (SDDs) use nonvolatile memory. SSDs have fast random access and limited writes. (Although limits are relatively large and may grow.)

Design Objectives

We want to save the results of large-scale breadth-first searches for later usage.

We assume the state space is too large to directly fit in memory.

We want to maximize the use of parallel resources.

Primary Test Hardware

16-core 2.6GHz AMD Opteron server
64GB of RAM
2 TB HDD, 500 GB HDD, 500 GB SDD

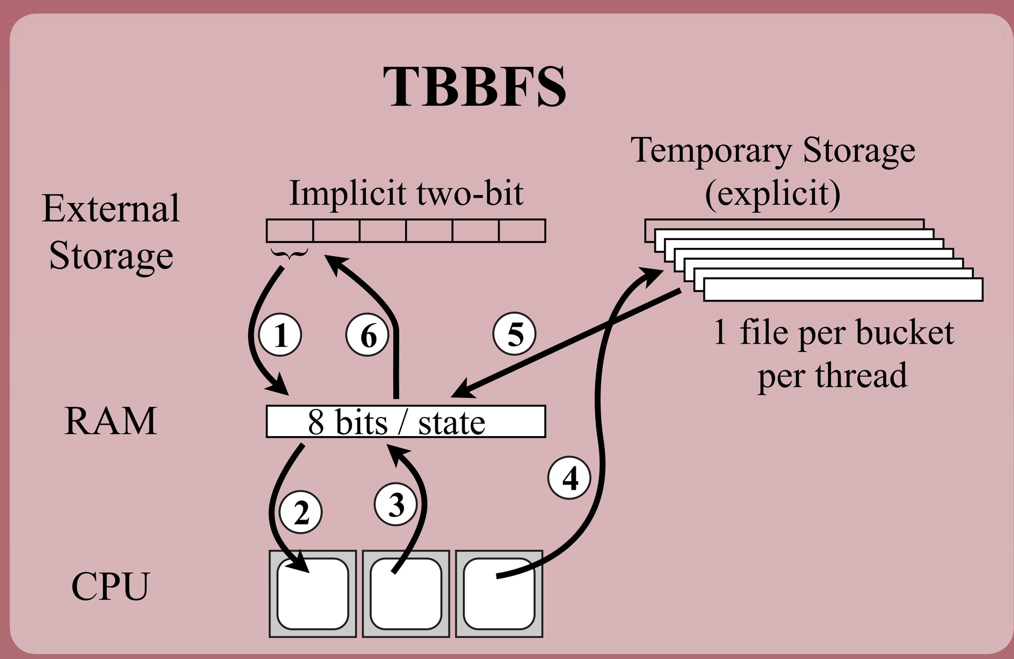
UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Example Algorithm: TBBFS

Two-Bit Breadth-First Search (TBBFS) (Korf, 2008) uses a 2-bit implicit representation to mark all states as open, closed, new, or unseen.

At each depth TBBFS expands and transitions states: open->closed, new->open, (some) unseen->new.

The state space is broken into buckets and (1) processed one bucket at a time.



When (2) successors of a state are generated, we must lookup their status. If they are in memory, (3) they can be processed immediately. Otherwise they are (4) written to external storage to handle later (5). The bucket is then (6) written back to disk.

In each level of the search, TBBFS expands a state just once (minimizing expansions).

TBBFS does not store the results of the BFS.

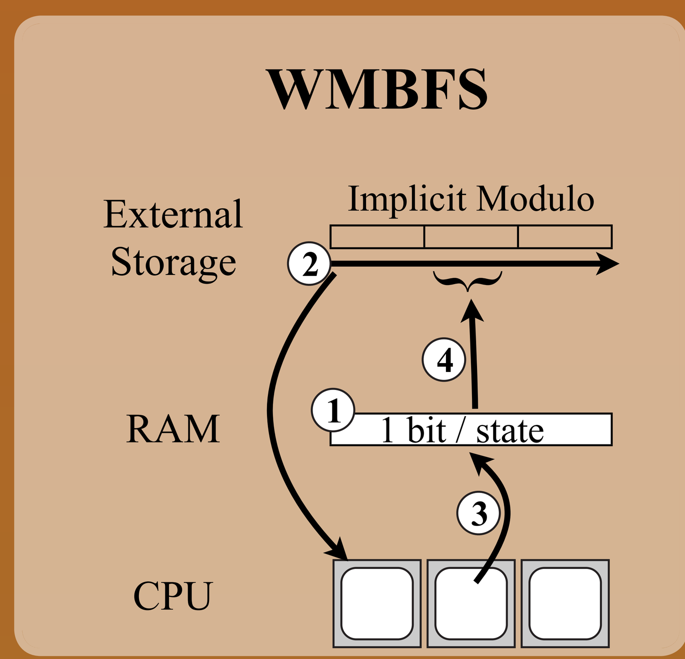
WMBFS - New Approach

We introduce a Write-Minimizing Breadth-First Search (WMBFS).

WMBFS uses a four-bit modulo representation to store the depth of a state. This implicitly determines open/closed/new. A reserved value marks unseen states.

WMBFS uses an implicit 1-bit representation (1) to mark potential successors at the next depth. The full 1-bit data is broken into buckets the size of RAM.

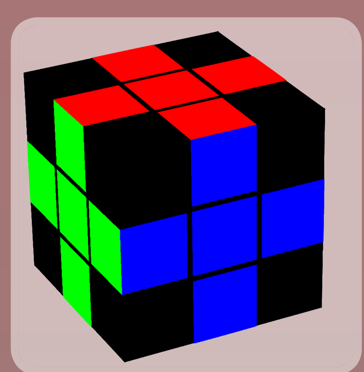
For each depth, WMBFS repeatedly (2) reads all states on disk, expanding those at the current depth. Successors in the current bucket are (3) marked in RAM and (4) written to disk. ***Successors not in RAM are discarded***.



At each BFS depth, steps (2-4) are repeated as many times as there are buckets.

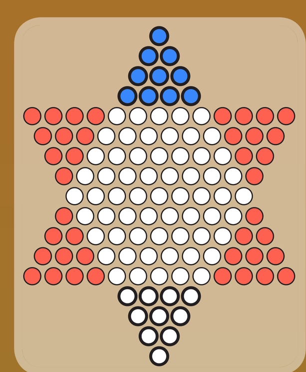
WMBFS trades off computation for less writing - no need for writing states to temporary storage.

Domain 1: Rubik's Cube Edges



\# of states: 980, 995, 276, 800
Expansion cost: small
\# successors: 18
Successor Locality: low
WMBFS Storage: 500 GB SSD
TBBFS Storage: 250 GB SSD + 2 TB HDD

Domain 2: Chinese Checkers (1P)



\# of states: 1,072,763,999,648 (w/symmetry)
Expansion cost: large
\# successors: 14-99
Successor Locality: high
WMBFS Storage: 500 GB SSD
TBBFS Storage: 250 GB SSD + 2 TB HDD

Rubik's Cube Results

	WMBFS	TBBFS
Total Time	586,433 sec 6.8 days	2,099,746 sec 24.3 days
Nodes Exp.	1,961,990,553,104	980,995,276,800
Total Writes	2.68 TB	>60.5 TB
Total Reads	6.85 TB	>60.5 TB

Chinese Checkers Results

	WMBFS	TBBFS
Total Time	2,632,266 sec 30.5 days	2,410,966 sec 27.9 days
Nodes Exp.	1,837,185,821,822	1,072,763,999,648
Total Writes	4.85 TB	>88 TB
Total Reads	10.80 TB	>88 TB