A survey study of parallel A* (in Rust)

Yimin Leng

April 28, 2023

Sequential A* algorithm

Algorithm 1 Sequential A*

```
Initialize OPEN to s_0;
while OPEN ≠ ∅ do                                  ▷ usually implemented as heaps
    Pop from OPEN a node n with a smallest f(n);
    Add n to CLOSED;
    if n ∈ end states then
        Return solution;
    for every successor n' in EXPAND(n) do
        g = g(n) + c(n, n');
        if n' ∈ CLOSED then
            if g < g(n') then                      ▷ won't happen with admissible heuristic
                Remove n' from CLOSED and add it to OPEN;
            else
                Continue;
        else
            if n' ∉ OPEN then
                Add n' to OPEN;
            else if g ≥ g(n') then
                Continue;
        Set g(n') = g;
        Set f(n') = g(n') + h(n');
        Set parent(n') = n;
Return failure (no path exists);
```

DPA

- Processors/threads locally maintain open and closed list.
- Individual processors/threads may visit nodes that are not optimal globally (slight altercation in algorithm needed to achieve optimality).
- Potential speedups occur when:
  - heuristic functions are not accurate,
  - the algorithm backtracks or encounters ties.
- Communication schemes are necessary to avoid duplicate work (search overhead):
  - Leaderboard strategy
  - Random communication

HDA

- Random communication potentially leads to poor load-balancing.
- Assign processors/threads as owners of states using hashing.
    - Additionally HDA solves duplication perfectly, as only owners of a state may open or close the state.
- Previous work has shown that performance depends heavily on the choice of hash functions.

Zobrist hashing

- Classic technique used in chess game programs to implement transposition table.
- Encode the state of board using the information of each piece and its position.

7	8	9
6	5	4
1	2	3

$\Longrightarrow$

1 at (0, 0)
2 at (0, 1)
$\vdots$
8 at (2, 1)
9 at (2, 2)

$\Longrightarrow$

0b11001001
0b11100111
$\vdots$
0b11010110
0b11011110

$\xrightarrow{\text{xor}}$ 0b11001100

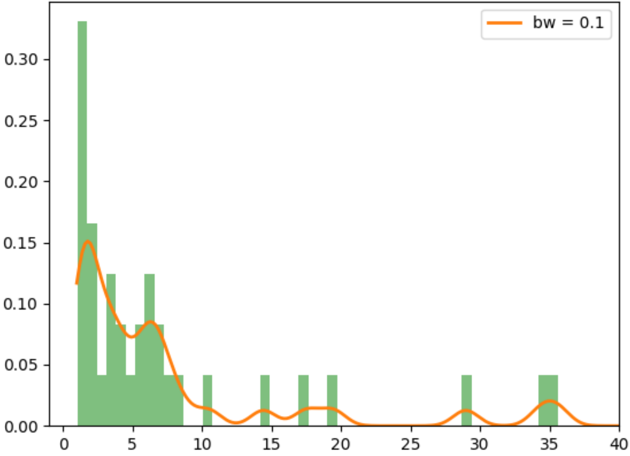
AHDA

- A disadvantage of HDA is that successors of a state are more likely to be sent to a different processor/thread, increasing communication cost.
- Map states to abstract states before hashing.
- A combination of abstract feature mapping and hashing may achieve both load-balancing and the reduction of communication overhead.

Setup

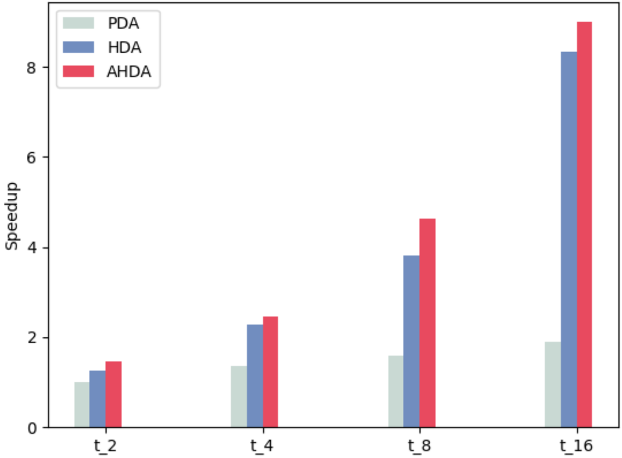
- All tests are performed on cycle3 machine.
- All parallel algorithms are implemented in shared memory context.
  - Using threading and channels for communication
- Test cases:
  - Randomly generated "15/24-sliding puzzles".
  - Minimal runtime of 1s using the sequential algorithm.

Runtime distribution of test cases



- 35 puzzles
- median: 4.10s, mean: 7.64s

Batch testing results



- PDA with a communication parameter of 0.6.
- AHDA uses a hand-crafted abstraction function.

Conclusions

- Use HDA and AHDA if possible, both scale very well (with the right hash function).
- Better performance can be achieved with DPA by fine-tuning communication frequency.

Future work

- Make the search framework generic.
- Explore inherent structure of the search problem that makes it suitable for parallellization.