Parallel A* pathfinding algorithm

Giuseppe Accaputo
Pascal Iselin

December 16, 2013
Overview

- Goal
- Literature Review
- Approaches
- Results
Our Goals were

- Implement a correct parallel A* algorithm
- Make it faster than the serial version
Our Goals were

- Implement a correct parallel A* algorithm ✓
- Make it faster than the serial version
Our Goals were

- Implement a correct parallel A* algorithm ✓
- Make it faster than the serial version ❌
Best first heuristic search

\[ f(n) = g(n) + h(n) \]

- **exact cost**
- **estimated cost**
Parallel A* in the literature

- **Shared Priority Queue**: Threads produce and consume simultaneously. Does not perform well!
  
  [Cohen et al., 2010]

- **Bidirectional Search**: Run two searches. Does not scale!
  
  [Rios, Luis Henrique Oliveira, and Luiz Chaimowicz. PNBA*: A Parallel Bidirectional Heuristic Search Algorithm.]

- **Sacrifice path quality for speed**: Converge towards other algorithms
  

- **Clustering**: Too complicated
  
Ideas we implemented

- Concurrent Neighbor Expansion
- Shared Priority Queue
- Atomic ClosedFlags + Shared JobQueue

Same underlying datastructure for all the Implementations. SquareLattice filled with Objects of Type MapNode
**Concurrent Neighbor Expansion**

Concurrent calculation of the neighbors in each step. This does not scale but it’s easy!
Parallel A* pathfinding algorithm

Approaches

Shared Priority Queue

- concurrent_priority_queue (CPQ) from Intel’s Thread Building Blocks (TBB)
- CPQ does not allow rebalancing
- Recursive call of the search function with a counter
- One lock per node

```c
atomic <size_t> num_threads;
task_group t_group;

t_group.run([&]{ parallel_search(); });

void parallel_search() {
    //--------
    // A* Magic
    //--------
    size_t n_threads = ++num_threads;
    if(n_threads < max_threads) {
        t_group.run([&]{ parallel_search(); });
    } else {
        --num_threads;
    }
}
```
Parallel A* pathfinding algorithm

Approaches

Atomic ClosedFlags + Shared JobQueue

- Run serial A* until we have enough open nodes
- Run a new A* in parallel on each of the open nodes
- Each thread has its own priority queue
- Threads communicate through a shared grid of closed flags
- Use atomic CAS to set the flags
Parallel A* pathfinding algorithm

Approaches

Atomic ClosedFlags + Shared JobQueue

- Run serial A* until we have enough open nodes
- Run a new A* in parallel on each of the open nodes
- Each thread has its own priority queue
- Threads communicate through a shared grid of closed flags
- Use atomic CAS to set the flags

- How to make sure threads don’t just terminate? → Shared JobQueue
- How to guarantee the shortest path?
How to guarantee the shortest path?
How to guarantee the shortest path?
How to guarantee the shortest path?

Just take the green path... NO! That would be fringe search!
Test setup

- Tested on Kanifushi
  - (32 Cores Intel Xeon E7-4830 @ 2.13Ghz)
- Compiled with GCC 4.7 and TBB 3.0
- 10 runs per test case
- 90% of random map was walkable
Parallel A* pathfinding algorithm

Results

Serial A* vs. Boost A* run time (Random Map)

Execution time in seconds

Size of square lattice

Boost A*
Serial A*

N=100  N=200  N=300  N=400  N=500  N=600  N=700  N=800  N=900  N=1000

Box plots showing the execution time in seconds for Serial A* and Boost A* for different sizes of square lattices.
Parallel A* pathfinding algorithm

Results

Serial A* vs. Boost A* run time (Wall map)

Execution time in seconds

Size of square lattice

N=100 N=200 N=300 N=400 N=500 N=600 N=700 N=800 N=900 N=1000

Boost A*

Serial A*
Parallel A* pathfinding algorithm

Results

Serial A* vs. Parallel Neighbor Expanding A* run time (Random map)

Execution time in seconds

Size of square lattice

N=100 N=200 N=300 N=400 N=500 N=600 N=700 N=800 N=900 N=1000

Parallel Neighbor Expanding A*

Serial A*
Parallel A* pathfinding algorithm

Results

Size of square lattice

Serial A* vs. Parallel Neighbor Expanding A* run time (Wall map)

- Parallel Neighbor Expanding A*
- Serial A*

Execution time in seconds

N=100 N=200 N=300 N=400 N=500 N=600 N=700 N=800 N=900 N=1000

Size of square lattice
Parallel A* pathfinding algorithm

Results

Serial A* vs. Parallel A* run time (Random map)

Number of threads

Execution time in seconds

Parallel A*
Serial A*

Number of threads

Execution time in seconds
Parallel A* pathfinding algorithm

Results

Serial A* vs. Parallel A* run time (Wall map)

Execution time in seconds vs Number of threads

Serial A* vs Parallel A*
Parallel A* pathfinding algorithm

Results

Parallel Neighbor Expanding A* vs. Parallel A* run time (Wall map, N=1000)
Conclusions

- We comply with the literature!
- One must sacrifice path quality for speed
- There are much better alternatives out there:
  - Ripple Search \([Brand \ et \ al., \ 2012]\)
  - Fringe Search