6.172 Project 4 Beta Write-up
agrebe, crinard, jaytlang, adriano

Table of Contents:

Executive Summary

We were given an engine for playing Leiserchess, a variant of laser chess. The original implementation contained many algorithmic strengths, including alpha-beta pruning and caching board states, but many of the low-level routines, especially for evaluating heuristics of board state, were unoptimized. We were tasked with optimizing the engine enough to be competitive with other staff implementations, including one designed to run five times faster.

Our high-level algorithmic design is fundamentally the same as the given implementation. Our main change was to increase the speed of the laser-coverage heuristic called during board evaluation. We also implemented an opening book that pre-computed common opening sequences and the best responses to them.

We also attempted various changes which were less successful. These include a bit-board, a “branching” laser coverage function, and more. We also had some ideas we did not implement, including a graph data structure and a short-circuited version of `eval()` which only computed laser coverage if the other heuristics did not already conclude as to whether the board state was low or high value.

Remaining Bottlenecks

From perf, about half the runtime in the beta submission is in computing laser_coverage and the add_laser_path subroutine that it calls. About 15% of runtime is spent in move generation for the laser coverage, about 10% of runtime spent in scout_search (a significant fraction of which is move sorting), and the other 25% of runtime is in miscellaneous function calls. Optimizing add_laser_path would be a significant goal for the final, but given that we have already studied this code, there is unlikely to be low-hanging fruit remaining and further optimizations will require more effort. A diagram of current bottlenecks according to pprof is pictured in Figure 1:

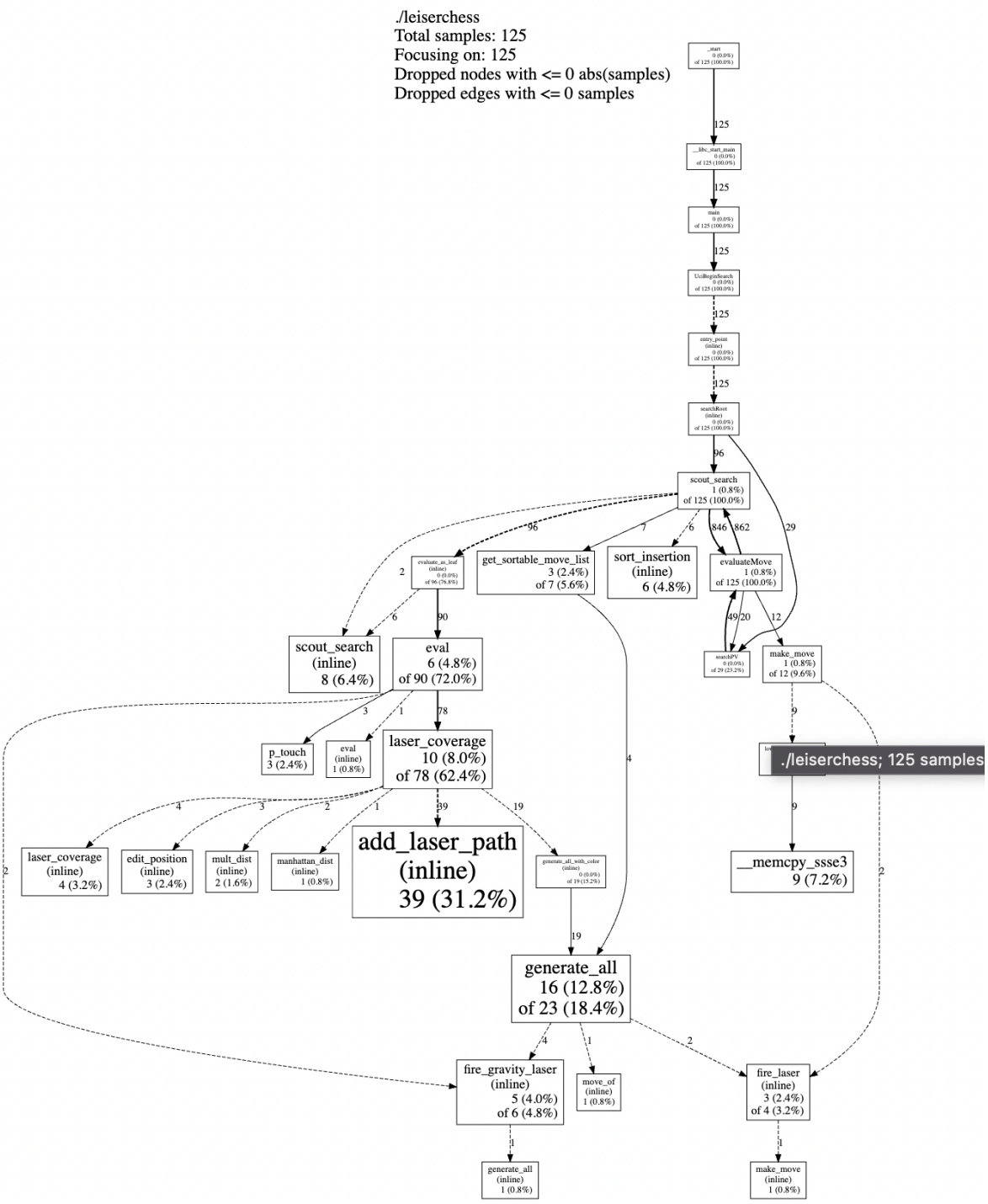

Figure 1: Diagram of current bottlenecks according to Google's pprof

Performance optimizations attempted and their impact (With Data)

A significant bottleneck in the original code, taking roughly a quarter of the runtime, was copying the board at each move, and in particular for each ply during laser coverage computation. This could be avoided by making many of the moves in place (without copying the board) and then undoing them afterwards, saving about 25% of runtime. We also realized that about half the moves considered during laser coverage had no effect on the laser path, so adding the laser path on these moves could be avoided, saving close to half of total runtime. Enabling link-time optimization gained us about 10% of runtime. Overall, this gained about a factor of four in performance relative to the original implementation.

Opening book

From a limited sample of watching games in the GUI, it appeared that the bots spend about a quarter of their total allocated time in the first half-dozen moves by each side, and in this time, they only managed to reach depth 5 or 6 and therefore made sub-optimal moves. (In particular, the best opening move at depth 9 is b4c3, but this is not found at the depth 5-6 searches typically run.) Since the initial position is fixed, common opening sequences can be hard-coded into the bot in an opening book. Our beta submission had an opening book of common openings out to about 10 ply, with the initial moves computed to depth 8 and subsequent ones computed to depth 6. For most opening lines, this makes our bot both better and faster. Empirical playtesting found that the version with an opening book outperformed the one without by about 150-200 Elo, which was a significant factor in our ability to beat the reference bot.

For the final submission, we hope to search to larger depth at each move and also to more plies. There were also some missing lines in the opening book that should be included in the final submission.

Failed/Not Tried Approaches

Smaller boards

We spent a lot of time and effort on trying to find faster algorithms for laser coverage, and board representation. We built a shim layer to abstract away the board representation from its behavior and replaced calls in the entirety of the codebase to use this shim layer. This made it easy to try different board sizes as well as a bit-board

representation of the board using three uint128s. It also enabled logic that iterates through the number of pawns for the laser path, after calculating a bitmask for where mirrors exist. Unfortunately, this took 1.2x the time of the original code to run. The distribution of the runtime of this was evenly distributed across instructions, indicating that the algorithm was just slower. Of the runtime in the new add_laser_path, around 38% came from computing the bitmap, which could be gotten rid of using some of the architecture from the bitboard branch, but in the end it would only speed up by 10-15%. This may be worth revisiting if we decide that the parallelization of eval is something worth working on.

Graph Board Representation

We also searched for a graph representation of the board that would allow for easier laser coverage, in the end we were unable to come up with a conceptualization that was effective for the operations that we needed. Our best idea was to store, for each pawn and monarch, the closest pawn or monarch (or wall) in each of the four directions. This would make it easier for us to do certain aspects of laser coverage since we would not need to iterate through empty cells. However, it complicated other aspects, such as updating the graph, calculating the shortest distance to each non-pawn, non-monarch square hit by the laser, etcetera.

Short Circuiting Eval

We also tried to short-circuit eval to avoid calling laser coverage. The idea was that if the other heuristics were highly predictive of laser coverage in certain cases, there was no need to calculate it. For example, if the other heuristics gave the board a very, very low value, it is unlikely that laser coverage was necessary, and the same is true for a very, very high value. Before we implemented it we ran a simple analysis by calculating the correlation between laser coverage and the other heuristics' summed value. Both were scaled by their weights. There was a bug in the correlation code which we did not resolve in time, so we never finished the short circuit code.

Branchless Laser Traversal

There is a switch statement in laser coverage which checks the type of each cell it traverses to decide what to do. We had an idea to remove that by storing "accelerations" which you can add to your vertical and horizontal "velocity," thereby making the code branchless. However, we never tried it since it was complicated and the switch did not appear to be a significant bottleneck.

Branching within Laser Path Calculation

Finally, the current implementation first generates a list of moves and then calls add_laser_path on each of them. One idea was to walk along the laser path and execute

moves as they are relevant to the path of the laser (i.e. moving pieces in or out of the path), since this would reuse the first half of the path rather than recomputing the entire path for each move. However, the attempted implementation of this includes the original path in the set of paths considered in the laser coverage (rather than just modifications to the original path), a small change but one that meant that existing correctness tests were insufficient to evaluate this change. Playtesting against the previous implementation was inconclusive, so this change was not merged into the main branch.

Team Dynamics

Chris and Adriano did a lot of peer programming. Jay did tests kind of. Anthony made eval fast asynchronously. He too peer programmed with Chris. Many changes were also made synchronously by various teammates. We also met frequently to discuss improvements made and steps moving forward, most people knew the areas that needed to be improved and were kept updated on areas of potential improvement.

Individual Contributions

Anthony: Searched for bottlenecks with perf and found and eliminated several significant bottlenecks in laser coverage. Also generated a simple opening book and created scripts that can be used to extend the opening book. Tried unsuccessfully to perform branching within laser path computation to eliminate the call to move generation. Ran and analyzed play-testing of various ideas to determine which ideas were net improvements.

Chris: Pair-programmed with Anthony early on in the project to get easy gains (editing rather than copying board). Worked on for() loop implementation of laser coverage, while(1) {head->wall} on clever laser coverage algorithm, first pass on short-circuiting, bitboard and different board implementations.

Adriano: Pair programmed with Chris to try and implement bit-board. Tried to make changes to laser coverage to use integers instead of floats. Wrote documentation and lots of pseudo-code for different algorithms.

Jay: Design document, unit testing infrastructure, consistency testing infrastructure. Wrote considerable documentation for various different dev-ops tasks integral to the rest of the team's capabilities. Opened up a 64-core machine for testing during the final project phase, started transition into programming for the parallel project.

Project Logs

Jay:

- 11/10: team contract, 1 hr
- 11/13: meet with folks to do overview, play leiserchess, wrote unit tester - 6 hr
- 11/15: Implement CHECK_REP and DUMP_DS subsystems, write design doc - 7.5 hr
- 11/20: Implement consistency checking infrastructure + script ASAN/MSAN - 6 hr
- 11/21: Salvaged + opened up 64-core machine for testing use by other group members - 3 hr

Adriano:

- 11/10: Team contract (1 hr)
- 11/13: Meet with team to do overview, play leiserchess (1.75 hrs)
- 11/14 & 11/15: Unclear; did not log what I did, maybe something hopefully not nothing (4.25 hours)
- 11/18: Reading the code, System diagram on paper, documentation (2.25 hrs)
- 11/19: Documentation centralization, OH, float to int for laser coverage, designing data structure with Chris (7.5 hrs)
- 11/20: Pseudocode for bit-boards and branchless laser traversal, call with Chris, created a log in Github wiki (4 hrs)
- 11/21: Reviewed Shim, planned w/ Anthony (1.25 hrs)
- 11/23: Working on 8x8 board, modified shim but was unsuccessful (6 hrs)
- 11/24: Working on short circuiting correlation and bitboard with Chris (7 hrs)
- 11/29: Writing this with my team (1 hr)

Chris:

- 11/9: Check out codebase (0.75 hr)
- 11/10: Team contract, pair working w/ Anthony (2 hr)
- 11/13: Meet with team to do overview, play leiserchess, start messing around with board representations (4 hr)
- 11/15: Meet with team to go over design submission, do final edits and formatting, submit (4.5 hr)
- 11/17: Pack board into 10x10, troubleshoot correctness and performance measurement issues (6 hr)
- 11/18: Rewrite 10x10 board rep, try to figure out how to validate (5 hr)
- 11/20: Figure out how to validate 10x10 rep, discover it has no performance impact, discuss branchless laser traversal and boards to support this (5 hr)
- 11/21: 11/20: Head->wall on closing book concepts (3 hr), build shim layer for boards (6 hr)

- 11/22: MITPOSSE meeting (1 hr), finish shim layer, start server testing and comparing branches with correctness-breaking optimizations (3 hr)
- 11/23: Work with Adriano on short-circuiting and bitboard (6 hr)
- 11/24: Individual work on bitboard, then work with Adriano & Anthony on bitboard, then short-circuiting, discuss final ideas (8 hr)

Anthony:
- 11/8: Becoming oriented with project and codebase (1.5 hr)
- 11/9: Continuing to become oriented with codebase (1.5 hr)
- 11/10: Initial group meeting, preliminary search for low-hanging fruit for optimizations (4 hr)
- 11/11: Generation of rudimentary testing suite, optimizing codebase (4.5 hr)
- 11/13: Group meeting, individual programming (4.5 hr)
- 11/14: Worked on initial optimizations and merged these into master branch (6.5 hr)
- 11/15: Worked on design document, some programming (3.5 hr)
- 11/17: Individual programming (3.5 hr)
- 11/19: Started working on opening book (3 hr)
- 11/20: More work on opening book, attempt to find more performance optimizations (5 hr)
- 11/21: More work on opening book, attempt to find more performance optimizations (6.5 hr)
- 11/22: More work on opening book, playtesting of various branches (3.5 hr)
- 11/23: Analyzing results of playtesting, continuing to work on opening book, attempt to branch within add_laser_path (4.5 hr)
- 11/24: Minor optimizations, meetings to finalize beta submission (6 hr)

Total time for beta submission: 58 hr

Response to MITPOSSEs

Jay: My MITPOSSE didn't get back to me at all despite multiple follow-ups, (see private piazza post @1391), so as recommended by the course staff, I read through and discussed my partners' changes (see below)

Chris: My MITPOSSE told me similar things to Adriano's. He emphasized that there was solid design and good principles that needed a bit more visual documentation. He also heavily emphasized the need for a table of contents and more broadly more centralized documentation. To address these comments, we've included more diagrams in our various performance discussions, as well as more comprehensive git documentation in issues and a wiki. In this document, we included a table of contents.

Adriano: My MITPOSSE told me that our code was solid and did not absolutely need any changes. However, we encouraged us to draw a program flow diagram, do smoke tests, and document our unit tests. We did not add additional smoke tests since we ran out of time and the existing ones were sufficient for most of our needs. We did create unit tests only for the bit board branch, and they were documented. We did not have time to make a full flow diagram, but we have ample documentation of that kind in our github wiki.

Anthony: My MITPOSSE was impressed by the plan for testing in the design document and said that a properly implemented testing suite (and especially adversarial testing) would help catch bugs and accelerate things later on. He also commented that the design document probably could have been better organized (e.g. reserving top-level headers for the most important sections), which is good advice for writing going forward (even if it's too late to implement for the design document).

Release and Parallelization Plan (With Data)

Serially speeding up our bit-board's operations (as defined by the abstraction methods) may be helpful to decrease memory copies prior to attempting parallelization. This could be useful, since we could more quickly copy the bitboard to different threads and use them in parallel.

We think that a threaded search algorithm will yield immediate results and will provide higher parallelism than would, say, parallelizing laser coverage. Note that it is not obvious to us how to make laser coverage, for example, very parallel since finding the minimum laser path for each square probably would involve a non-trivial "reduce" step (we estimate O(log(n)) complexity by merging pairs). Hence we will prioritize parallel search first, even if we attempt both of these optimizations.

In addition, we could run a subset of heuristics in parallel if we determine that thread spawning overhead does not defeat potential benefits, inserting new heuristics as needed, and finding other areas in the codebase where parallelization via `cilk_for` would benefit. An initial analysis with perf reveals that many items we aren't yet sure how to parallelize effectively (namely laser coverage, brute-force move generation, and scout search's insertion sort) take up roughly 56% of our serial implementation's runtime. This implies that higher-level parallelization, mostly in the alpha/beta search, will be critical in accelerating our performance rather than parallelizing algorithms themselves.

Acknowledgements

We would like to thank the course TAs for helping us come up with ideas on Friday November 19th, 2021. Specifically, Sameer, and Obada helped Adriano explore different theoretical options for speeding up laser coverage by changing the board rep. The lectures by course staff were also extremely helpful both in understanding the engine algorithms and in developing ideas for parallelism.