15-418 F22: Parallel Computer Architecture and Programming Term Project: CPU and GPU Accelerated Parallel SAT Solver

Name: Jiarui Wang, Tejas Badgujar
Andrew ID: jiaruiwa, tbadguja

Project Github URL

1 Schedule

- **Week 1 (11/7-11/13)**
  - Finished writing project proposal and setting up development environment
- **Week 2 (11/14-11/20)**
  - Finished writing uniform 3-SAT problem generator python script for test case generation.
  - Found 3-SAT international benchmarks online for more standardized testing [3].
  - Finished file I/O structure of the code.
  - Designed and implemented two different boolean assignment representation
    - Array based representation with unlimited variable size
    - 64 bit integer based representation with limited size of 64 variables
- **Week 3 (11/21-11/27) (Thanksgiving week)**
  - Finished basic sequential naive brute force algorithm
  - Finished basic sequential DPLL algorithm
- **Week 4 (11/28-12/4)**
  - Implement basic sequential CDCL algorithm (Tejas)
  - Implement and benchmark vector based and thread based divide and conquer parallelism on naive brute force algorithm (Jerry)
  - Start on CUDA implementation of divide and conquer brute force algorithm (Jerry)
- **Week 5 (12/5-12/11)**
  - Combining DPLL and CDCL together to create the portfolio algorithm (Tejas)
  - Finish CUDA implementation of the naive brute force algorithm (Jerry)
  - Combining DPLL and divide and conquer together to create a divide and conquer DPLL algorithm (Jerry)
- **Week 6 (12/12-12/18)**
  - Prepare final presentation

2 Work Summary

We have designed and implemented two different representations of boolean variable assignment that supports unlimited amount of boolean variables and 64 boolean variables. The unlimited amount of boolean variables utilizes heap allocated arrays that is probably not necessary for most SAT problems. Since the search space of SAT problems increase exponentially in size, an implementation that supports up to 64 or 128 variables should be sufficient.

We have also implemented a python script that will generate random SAT problems uniformly in 3-CNF form to help us quickly benchmark and estimate the performance of our SAT solver implementations. We have also found a large amount of 3-SAT benchmarks that was used by researchers around the world[3]. This set of tests will give us a more consistent and robust benchmark on our algorithms.

As for SAT solvers, we have implemented a naive brute force implementation. It will search through the exponentially sized search space of all possible boolean value assignments and evaluate the CNF clauses with the assignment. If the assignment could satisfy all the CNF clauses, the program will write the assignment to each variable to an output file. If none of the assignment will satisfy the SAT problem, then the program will output unsatisfiable.

We have also implemented a sequential DPLL algorithm that could solve problems up to 150 variables and made significant progress on the CDCL algorithm. Overall we are almost done with our sequential implementation and we are ready to start parallelize them.

3 Progress on Goals

One of the goal for the term project is to have a divide and conquer style parallelism for CPU and GPU. Right now we already have the basic sequential algorithm ready and set up the proper testing framework. We made really good progress and we believe we are on track of finishing both divide and conquer algorithm for CPU and GPU by the poster session.

We have also implemented DPLL and made significant progress on CDCL so we are also on track of finishing the portfolio parallel algorithm by the poster session since portfolio algorithm is basically having each core running different literal choosing heuristics of DPLL and CDCL.

It would be nice if we could have the time to implement a divide and conquer DPLL with CPU. The implementation on CPU should be fairly straightforward and we believe we should be able to finish it by the poster session. If we have enough time we could also explore some other way that we could parallelize DPLL or even attempt to implement it on GPU. But all of these additional goals will depend on how fast we could finish our definitive goals.

4 Results We Will Show

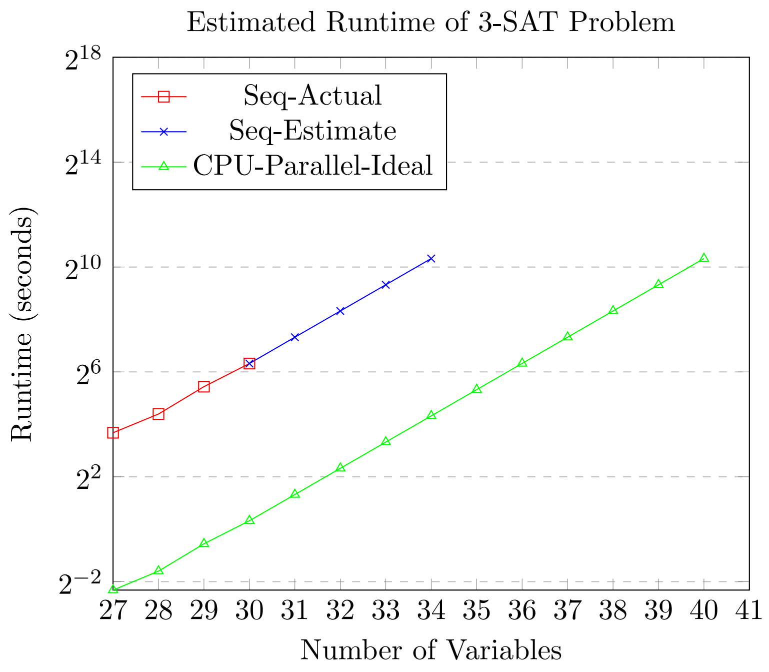
We will show case the plot of the time for each algorithm to run on various set of test cases for SAT problems [3]. This should give us a good indication about the speedup of our algorithms.

5 Preliminary Results

Due to the nature of SAT problems, it is hard to have an accurate benchmark of the algorithm. In the worst case scenario, the algorithm will have to go through every single possible boolean variable assignment in order to confirm that the assignment is indeed unsatisfiable. For a sequential brute force SAT solver, based on our randomly generated benchmarks, it is able to determine the satisfiability of unsatisfiable uniformly randomly generated 3-CNF problems with 125 clauses in the following time:

Variable amount	Program runtime
27	12813 ms
28	21046 ms
29	43347 ms
30	79820 ms

We could see that the program runtime increase exponentially as the number of possible boolean variables increases. This is expected since the search space of SAT problem increase exponentially with the number of variables. We can get a rough estimation in the following plot.



In the ideal scenario, we would fully utilize the 8 wide vector lane and all 8 cores of the CPU, achieving a 64x speedup. We can see that this will allow the naive brute force algorithm to push 6 more variables in the search space with the same amount of runtime. We should see even better speedup with GPU considering the throughput of the GPU is much higher than CPU. Getting close to the ideal speedup on CPU and a much faster speedup on GPU will be the focus of this week and next week.

We are also able to solve up to 150 variables with the DPLL algorithm. Combining different heuristics of DPLL and CDCL to form a portfolio algorithm will also be the focus of this week and next week.

6 Issues of Concern

The limited size of the 64 bit integer representation may prove problematic especially when GPU might allow the brute force method to go beyond 64 variables. So we would need to come up with a GPU friendly representation of the boolean assignment if we ever go beyond 64 variables. It is also challenging to vectorize the CPU code but with enough time and research we should be able to figure out how to vectorize the brute force code.

On the other hand, CDCL is notoriously hard to parallelize so we might have to just use the sequential version of it in the portfolio methods.

References

[1] "Boolean Satisfiability Problem." Wikipedia. Wikimedia Foundation, October 24, 2022. https://en.wikipedia.org/wiki/Boolean_satisfiability_problem.

[2] "SAT Solver." Wikipedia. Wikimedia Foundation, September 1, 2022. https://en.wikipedia.org/wiki/SAT_solver.

[3] SATLIB - Benchmark Problems. https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html