GPU-Accelerated Approximate Model Counting (#SAT) via Parallel XOR-Hash Evaluation

Aryaman Chawla

Introduction

The Sharp Satisfiability Problem (#SAT) is the problem of counting the number of solutions for a given Boolean formula. The only known way to do this is to cover the entire domain of solutions (i.e. go through all possible values of all variables). However, for just 128 Boolean variables, searching across the entire set of values would take billions of years, so most #SAT solvers nowadays approximate the total number of solutions using a variety of heuristics. One such technique is XOR hashing, where random XOR constraints are added to the original Boolean formula to shrink the solution space. When the solution space is shrunk enough, the SAT solver counts the total number of solutions, and this value is scaled up based on the number of XOR hashes, which gives us a final value for the total number of solutions, with a high probability/confidence. This problem would clearly benefit from massive parallelization, so in this project, I will implement a #SAT solver that generates sparse XOR hashes, solves the resulting formula and produces an approximate count.

Description

The program accepts as input a file that describes the original Boolean formula. This will be in conjunctive normal form (CNF) as defined by the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS).

The algorithm works by:

- First building families of random sparse XOR constraints, and the hash function that generates these constraints comes from a pretrained machine learning model. I am unsure what this model will look like at the moment, however, I want to replace pure random sparse hashing to try and reduce the variance in outcomes across assignments
- These constraints are then added, and the domain of solutions is reduced (partial assignments) by performing Gaussian elimination in parallel on the GPU. This matrix row reduction will output reduced variable sets or detect contradictions early
- Next, these partial assignments are applied to the CNF, and finally the clauses are evaluated in parallel, also on the GPU
- Finally, the model count is approximated and returned over several randomized trials

Objective and Deliverables

My main objective is to build a #SAT solver that is comparable to the best performing approximate model counting solvers. The exact metrics I will use to determine performance will be detailed in the report (as shown below).

I will have three deliverables:

1. An end-to-end approximate #SAT solver. This will be a full pipeline prototype in CUDA that performs the entire loop (hash generation → XOR elimination → clause evaluation → pass/fail) on the GPU for thousands of independent hashes. The code will be on GitHub and will contain detailed instructions on how to replicate my findings.
2. A short report. This short paper will summarize the architecture, engineering choices, performance and statistical analysis. It will also contain a detailed analysis about the speedup/performance gains my model counter will make.
3. A slide deck or poster, depending on what the format of the presentation will be at the end of the quarter.

Background

I have a deep understanding of SAT algorithms, and this is the core knowledge I need to implement the project. I also have lots of experience with statistical analysis (for approximate counting) and machine learning, but this is only a small part of the project. I have very limited experience with field theory, but I have experience with linear algebra, and this is necessary for reducing the solution domain. However, the math is very separated from its implementation, so I will not need to learn too much. I was recommended the Encyclopedia of Mathematics and Its Applications to help me learn field theory, and specifically how it can be implemented.

Resources

I have already ordered the Encyclopedia of Mathematics and Its Applications, and it will be delivered over the weekend. I also have access to the Wilkinson lab machines, so I will not need any specialized computing resources. However, I also want to test my code on other GPUs, and I have access to the Quest HPC, which I will utilize.

I will provide a method of generating random CNFs to stress test my system. However, I will also require public benchmark CNFs, and these will come from public sources. For example, here is a link to CNFs from the 2024 SAT competition: https://zenodo.org/records/15095752. This is not the only source I will look towards; however, it is a good starting place. All sources I use will be detailed in my report.

There are some papers I will reference.

-Here is an approximate model counter, similar to the one I want to build: https://github.com/meelgroup/approxmc

-Here is a paper on sparse hashing, which I will reference to create my own hash generator: https://arxiv.org/abs/2004.14692

-Here is a #SAT solver that I will reference to offload work onto the GPU: https://github.com/daajoe/GPUSAT

Contact Information

Aryaman Chawla – aryamanchawla2025@u.northwestern.edu