Master Project Presentation

Given by

Yandong Wang

To committee of

Rochester Institute of technology

Project Committee:

Chair      : Prof Alan Kaminsky

Reader    : Prof Stanislaw Radziszowski

Observer: Prof James Heliotis
NVIDIA CUDA Architecture-based Parallel Incomplete SAT Solver

Agenda:

- Introduction of this project
- Introduction of satisfiability problem and SAT solver
- Introduction of CUDA GPU programming
- CUDA-based Parallel Incomplete SAT Solver Design
- Measurement and Observation of the new CUDA-based SAT Solver
- Related research and future work
Introduction of Project

Stochastic local search and Genetic Algorithm

Massive parallel Computing capability of CUDA GPU

combine

CUDA SAT Solver
Satisfiability (SAT) Problem

Problem description

- “Given a boolean expression, determine if there exists an assignment of true or false to all boolean variables that make entire expression to be true?”
Satisfiability (SAT) Problem

Terminology

- NP-complete problem
- Literal
- Clauses
- Conjunctive Normal Form (CNF)
- k-SAT (2-SAT, 3-SAT, max-SAT)
- Phase Transition Phenomenon
Satisfiability (SAT) Problem

Phase Transition Phenomenon

Graph showing the transition from easy satisfiable to hard region to easy unsatisfiable with time and C/V axes. The graph indicates a peak at a point marked as 'τ'.
SAT Problem Solver

- Complete SAT solver
- Incomplete SAT solver
SAT Problem Solver

complete SAT solver

- Based on **DPLL** algorithm whose principle is backtracking and divide-and-conquer
- Unit Propagation
- Pure Literal Elimination
SAT Problem Solver

Incomplete SAT solver

- Stochastic local Search.
  *Random walk strategy*

- Genetic Algorithm
  *Cellular genetic algorithm*
Random Walk Strategy

- Involving process

Pure (unbiased) random walk selection strategy

Biased heuristic search strategy

Input 1: Randomly generated boolean assignment P
Input 2: Maximum Try Times M
Input 3: Probability ρ

while (P doesn’t satisfies all clauses) AND (M is not reached) do
    C ← Select a random unsatisfied clause.
    if There exists a variables V₁ with break value^a = 0 then
        Flip V₁
    else
        if created Probability ≤ ρ then
            V ← Select a random variable from C
            Flip the value of V
        else
            Flip the variable with the smallest break value
        end if
    end if
end while
if P satisfies all clauses then
    Return P
else
    No Solution is Found
end if
Cellular Genetic Algorithm

- Inherit the properties of regular genetic algorithm
- Diffusion model
Open Issues

- Keep steady diversity of the search space.
- Population homogeneity
- Premature convergence
CUDA GPU Programming

- Designed specifically for computing high parallel intensive-computation.

- Concentrate on similar data processing rather than data caching, flow control.

- NVIDIA's CUDA SDK and high-level programming language C.
CUDA GPU Programming Model

- Threads block
- Blocks Grid
- k-threads unit call warps
- Maximum number of blocks: $65535 \times 65535$
- Number of threads is limited
- Transparent scalability
- Single instruction multiple thread
  - \_synthreads() & ThreadSynchronize()
CUDA GPU Memory Model

- Device memory and host memory
- Global memory
- Register
- Shared memory
- Local memory
- Constant memory
CUDA-based Parallel Incomplete SAT Solver Design

Initialization variables

Optimize clauses

Device configuration

Update random number generator

Population initialization

Generate random masks

Do{
  1: initialize necessary variables
     _synchronization()
  2: neighbor selection
     _synchronization()
  3: crossover and mutation
     evaluation() and _synchronization()
  4: random walk strategy
}

while(evaluation fail)

Print result

Time out

No solution found
Data Allocation in Device Memory

- Data need to be transferred to device memory.
- Put as much data as possible into the shared memory or constant memory.
- Truth assignment matrix in global memory.
- Random generated masks in global memory.
• Truth assignment matrix
Data in Constant memory and shared memory

- Put clause information into constant memory.
  - Limited by the size of the problem.

- Using shared memory as truth assignment cache.
  - Limited by the number of threads in each block.
Random Number Generator

- Keep diversity of the search space.
- Hash function
- Parallelize random number generator.
  - Multiple random number generators?
  - Using one random number generator?
Sequence Splitting

- Approach to parallelize a sequential random number generator.
- A tradeoff needs to be made. (speed or perfect random number sequence)
Generate initial random population

- Minimize the probability of different threads generating the same truth assignment.
- Each truth assignment is a `char array`.

\[
\text{jump\_length}_1 = \text{get\_thread\_id()} \times \left\{ \frac{\text{number of variables} N}{\text{size of (char)}} + (1 \text{ or } 0) \right\}
\]
Generate crossover and mutation masks

- Minimize the probability of different threads generating the same truth assignment.
- The probability of 1 in the mask is equal to the P.

\[ \text{jump\_length}_2 = \text{get\_thread\_id()} \times N \times 2 \]
Evaluation

Char array = word + Bit array
Neighbor Selection
Crossover and Mutation

original design

[Diagram showing crossover and mutation process with binary strings and indices i and j.]
Crossover and Mutation

\[ \text{child}_1 = (p_1 \land \text{mask}_\text{crossover}) \oplus (p_2 \land \neg \text{mask}_\text{crossover}) \]
\[ \text{child}_2 = (p_1 \land \neg \text{mask}_\text{crossover}) \oplus (p_2 \land \text{mask}_\text{crossover}) \]

\[ \text{child}_1 = \text{child}_1 \oplus \text{mask}_\text{mutation} \]
\[ \text{child}_2 = \text{child}_2 \oplus \text{mask}_\text{mutation} \]

Evaluation of \( \text{child}_1 \)
Evaluation of \( \text{child}_2 \)

Return The One with Better Fitness
Crossover and Mutation

modified design 2

\[ \text{child} = (p_1 \land \text{mask\_crossover}) \oplus (p_2 \land \neg \text{mask\_crossover}) \]

\[ \text{child} = \text{child} \oplus \text{mask\_mutation} \]
Random walk strategy and Evolution

- Random walk strategy consumes most of the running time.
- Greedy strategy.
- Back and forth M times.
- Always replace the old generation.
Testing result and Observation

Testbed

- Sun Microsystem Ultra 40 workstation with 1 GHz dual-core AMD Opteron 2218 CPU, 8GB main memory.
- NVIDIA Tesla C870, 16 multiprocessors, each has 8 cores. 500 MHz clock
- *Uniform Random 3-SAT* problem set.
  (size from 20 variables / 91 clauses to 250 variables / 1065 clauses)
Testing result and Observation

- Used all of the 16 multiprocessors.
- Running time depends on the initial value of seed at a great extent.
- Hardness of different instances at the same size varies greatly.
## Testing result and Observation

### Running time measurement

<table>
<thead>
<tr>
<th>Problem Size (variable, clauses)</th>
<th>Minimum Running Time</th>
<th>Maximum Running Time</th>
<th>Running Time Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>(20/91)</td>
<td>0.03</td>
<td>0.06</td>
<td>0.03</td>
</tr>
<tr>
<td>(50/218)</td>
<td>0.19</td>
<td>0.2</td>
<td>0.01</td>
</tr>
<tr>
<td>(75/325)</td>
<td>0.38</td>
<td>0.4</td>
<td>0.02</td>
</tr>
<tr>
<td>(100/430)</td>
<td>0.67</td>
<td>4.55</td>
<td>3.88</td>
</tr>
<tr>
<td>(125/538)</td>
<td>1.03</td>
<td>6.06</td>
<td>5.03</td>
</tr>
<tr>
<td>(150/645)</td>
<td>1.48</td>
<td>68.5</td>
<td>67.02</td>
</tr>
<tr>
<td>(175/753)</td>
<td>1.99</td>
<td>140.31</td>
<td>138.32</td>
</tr>
<tr>
<td>(200/860)</td>
<td>2.61</td>
<td>925.41</td>
<td>922.8</td>
</tr>
<tr>
<td>(250/1065)</td>
<td>4.04</td>
<td>546.53</td>
<td>542.49</td>
</tr>
</tbody>
</table>

### Graphical Representation

- **Minimum Running Time**
- **Maximum Running Time**
- **Running Time Difference**

**SAT problem size**

**Y-axis (second)**

- 1000
- 900
- 800
- 700
- 600
- 500
- 400
- 300
- 200
- 100
- 0

**Legend**
- Blue: Minimum Running Time
- Red: Maximum Running Time
- Green: Running Time Difference
Testing result and Observation

Running time measurement

Average Running Time for different Size Random 3 SAT, Population Size 1600

<table>
<thead>
<tr>
<th>Problem Size (variable, clauses) \ threads</th>
<th>Grid.x = 16, Grid.y = 1, block.x = 10, block.y = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(20/91)</td>
<td>0.05</td>
</tr>
<tr>
<td>(50/218)</td>
<td>0.19</td>
</tr>
<tr>
<td>(75/325)</td>
<td>0.39</td>
</tr>
<tr>
<td>(100/430)</td>
<td>1.02</td>
</tr>
<tr>
<td>(125/538)</td>
<td>1.68</td>
</tr>
<tr>
<td>(150/645)</td>
<td>12</td>
</tr>
<tr>
<td>(175/753)</td>
<td>16.1</td>
</tr>
<tr>
<td>(200/860)</td>
<td>90.54</td>
</tr>
<tr>
<td>(250/1065)</td>
<td>111.51</td>
</tr>
</tbody>
</table>

![Graph showing running time for different problem sizes](image_url)
Testing result and Observation

Running time measurement

![Graph showing running time measurement vs problem size.](image)
Testing result and Observation

Scalability measurement
Testing result and Observation

Scalability measurement
Testing result and Observation

Scalability measurement

Average Scalability of different SAT problems at same size

<table>
<thead>
<tr>
<th>Threads (block, threads)</th>
<th>SAT Problem</th>
<th>250 variables, 1065 clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>[(8,1), (20,10)]</td>
<td></td>
<td>209.26</td>
</tr>
<tr>
<td>[(10,1), (16,10)]</td>
<td></td>
<td>138.18</td>
</tr>
<tr>
<td>[(16,1), (10,10)]</td>
<td></td>
<td>101.266</td>
</tr>
</tbody>
</table>

Graph showing average scalability over different thread configurations.
Testing result and Observation

Scalability measurement

Scalability of uf250-01

Scalability of uf250-04

Scalability of uf250-044

Scalability of uf250-074

Scalability of uf250-098

Case 1

Case 2

Case 3

Case 4

Case 5
Future work

- Decide the right seed.
- Conditional statement in neighbor selection.
- Use of constant memory and shared memory.
- Unit Propagation and Pure Literal Elimination.
- Test on structured SAT problems.
Related work

- “Parallel resolution of the satisfiability problem: a survey”
- “Implementing Survey Propagation on Graphics Processing Units”
- “NVIDIA CUDA for research”
References


References


References


Questions ?