gpusat2 – An Improved GPU Model Counter

Johannes K. Fichte\textsuperscript{1}  Markus Hecher\textsuperscript{2,3}  Markus Zisser\textsuperscript{2}

\textsuperscript{1} TU Dresden, Germany
\textsuperscript{2} TU Wien, Austria
\textsuperscript{3} University of Potsdam, Germany

Pragmatics of SAT (POS) Workshop 2019, Lisbon, Portugal

July 8, 2019
Motivation

Model Counting (#SAT)

- Generalizes Boolean satisfiability problem (SAT)
- #SAT: output the number of satisfying assignments
- WMC: output the weighted model count
- Various applications in AI and reasoning, e.g.,
  - Bayesian reasoning [Sang et al.’05]
  - Learning preference distributions [Choi et al.’15]
  - Infrastructure reliability [Meel et al.17]
- Computational complexity: #P-hard [Roth’96]
Motivation: A somewhat different approach.

#SAT/WMC Solving
- There are already plenty solvers based on various techniques:
  - approximate (Meel) / CDCL (Baccus/Thurley) /
  - knowledge compilation based (Darwiche et al.)

Parameterized Algorithms
- Lots of theoretical work over last 20 years and various algorithms for #SAT

Research Question
Are (theoretical) algorithms from parameterized complexity even useful for implementations in #SAT/WMC solving?
# Motivation: A somewhat different approach.

## #SAT/WMC Solving
- There are already plenty solvers based on various techniques: approximate (Meel) / CDCL (Baccus/Thurley) / knowledge compilation based (Darwiche et al.)

## Parameterized Algorithms
- Lots of theoretical work over last 20 years and various algorithms for #SAT

## Research Question
Are (theoretical) algorithms from parameterized complexity even useful for implementations in #SAT/WMC solving?
Parameterized Algorithmics

**Topic of the Talk**

Solve \#SAT/WMC by means of an implementation of a parameterized algorithm that explicitly exploits small treewidth.

**Presentation:**

1. Ideas towards a GPU model counter [FHWoltranZ'18]
2. Improved Architecture for \#SAT (POS paper [FHZ'19])

**Purpose:**

There are other architectures out there and it might fit for certain algorithms. NOT: outperforming everything else.
Parameterized Algorithmics

Topic of the Talk

Solve \#SAT/WMC by means of an implementation of a parameterized algorithm that \textit{explicitly} exploits small treewidth.

Presentation:

1. Ideas towards a GPU model counter \cite{FHWolftranZ18}
2. Improved Architecture for \#SAT (POS paper \cite{FHZ19})

Purpose:

There are other architectures out there and it might fit for certain algorithms. NOT: outperforming everything else.
Parameterized Algorithmics

**Topic of the Talk**

Solve #SAT/WMC by means of an implementation of a parameterized algorithm that **explicitly** exploits small treewidth.

**Presentation:**

1. Ideas towards a GPU model counter [FHWoltranZ’18]
2. Improved Architecture for #SAT (POS paper [FHZ’19])

**Purpose:**

There are other architectures out there and it might fit for certain algorithms. **NOT:** outperforming everything else.
Tree Decompositions

*Definition & Example*

- Most prominent graph invariant
- Small treewidth indicates tree-likeness and sparsity
- Can be used to solve $\#\text{SAT}/WMC$ by defining graph representations of the input formula
Tree Decompositions

Treewidth

- Treewidth defined in terms of tree decompositions (TD)
- TD: arrangement of graph into a tree + bags s.t. ...
- Treewidth: width of a TD of smallest width
Tree Decompositions

Treewidth

- Treewidth defined in terms of tree decompositions (TD)
- TD: arrangement of graph into a tree + bags s.t. ...
- Treewidth: width of a TD of smallest width
Definition

A tree decomposition is a tree obtained from an arbitrary graph s.t.
1. Each vertex must occur in some bag
2. For each edge, there is a bag containing both endpoints
3. Connected: If vertex \( v \) appears in bags of nodes \( t_0 \) and \( t_1 \), then \( v \) is also in the bag of each node on the path between \( t_0 \) and \( t_1 \).

Tree Decomposition \( \mathcal{T} \) of \( G \):

- \( G : x \quad a \quad c \quad b \quad c \quad y \)
- \( \mathcal{T} : b, c \quad b, c \quad b, c, y \quad b, x, c \quad b, x, a \)

Width: 

\[ \text{width} = \max \{ \text{size of bags} \} \]
1. Build graph $G$ of $F$
2. Create TD $\mathcal{T}$ of $G$
3. Dynamic Programming
   - Store results in table $\tau_t$
   - Apply $A$ to $F_t$
   - done? no
   - Visit next node $t$ of $\mathcal{T}$ in post-order
   - yes
4. Output count

Part:
A) Background & Basic Concepts
   - Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA’10]
B) Finding TDs (2)
C) Dynamic Programming (3) on the GPU
Outline (Basic Architecture)

1. Build graph \( G \) of \( F \)
2. Create TD \( \mathcal{T} \) of \( G \)
3. Dynamic Programming
   - Store results in table \( \tau_t \)
   - Apply \( A \) to \( F_t \)
   - Visit next node \( t \) of \( \mathcal{T} \) in post-order
4. Output count

Part:
A) Background & Basic Concepts
   - Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA’10]
B) Finding TDs (2)
C) Dynamic Programming (3) on the GPU
Outline (Basic Architecture)

1. Build graph $G$ of $F$
2. Create TD $T$ of $G$
3. Dynamic Programming
   - Store results in table $\tau_t$
   - Apply $A$ to $F_t$
4. Output count

Part:
A) Background & Basic Concepts
   - Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA’10]
B) Finding TDs (2)
C) Dynamic Programming (3) on the GPU
Outline (Basic Architecture)

1. Build graph \( G \) of \( F \)
2. Create TD \( T \) of \( G \)
3. Dynamic Programming
   - Store results in table \( \tau_t \)
   - Apply \( A \) to \( F_t \)
   - done? no
   - Visit next node \( t \) of \( T \) in post-order
   - yes
4. Output count

Part:
A) Background & Basic Concepts
   - Treewidth, Graph Representation (1) + Dynamic Programming (3) [Samer & Szeider JDA’10]
B) Finding TDs (2)
C) Dynamic Programming (3) on the GPU
How to “use” tree decompositions for #SAT/WMC?
Solving #SAT [SamerSzeider10]

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

\[ \text{Mod}(\varphi) = \{ \{b\}, \{a, b\}, \{b, c\}, \{a, b, c\}, \{b, c, x\}, \{a, b, c, x\}, \{b, y\}, \{a, b, y\} \} \]

1. Create graph representation
2. Decompose graph
3. Solve problems via S
4. Combine solutions

Diagram: Graph representation of the solution set.
Solving \#SAT \cite{SamerSzeider10}

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via $S$
4. Combine solutions
Solving #SAT [SamerSzeider10]

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via \( S \)
4. Combine solutions
Solving #SAT [SamerSzeider10]

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via \( S \)
4. Combine solutions

“Local formula” \( F_t \) clauses whose variables are contained in the bag (colored in red above)
Solving #SAT [SamerSzeider10]

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via \( S \)
4. Combine solutions
Solving #SAT [SamerSzeider10]

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via \( S \)
4. Combine solutions
Solving #SAT [SamerSzeider10]

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via \( S \)
4. Combine solutions
Solving \#SAT [SamerSzeider10]

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via \( S \)
4. Combine solutions
Solving #SAT [SamerSzeider10]

\[ \varphi = \neg a \vee b \vee x \land (a \vee b) \land (c \vee \neg x) \land (b \vee \neg c) \land (\neg b \vee \neg c \vee \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via \( S \)
4. Combine solutions
Solving \#SAT [SamerSzeider10]

\[ \varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y) \]

1. Create graph representation
2. Decompose graph
3. Solve problems via \( S \)
4. Combine solutions
φ = (¬a ∨ b ∨ x) ∧ (a ∨ b) ∧ (c ∨ ¬x) ∧ (b ∨ ¬c) ∧ (¬b ∨ ¬c ∨ ¬y)

1. Create graph representation
2. Decompose graph
3. Solve problems via S
4. Combine solutions
Solving $\#\text{SAT}$ [SamerSzeider10]

$\varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $S$
4. Combine solutions
Solving $\#\text{SAT}$ [SamerSzeider10]

$\varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y)$

1. Create graph representation
2. Decompose graph
3. Solve problems via $S$
4. Combine solutions

Runtime: $2^{O(tw)} \cdot \text{poly}(|\varphi|)$
“Find” tree decompositions of small width?

Works well even for relatively large instances.

Thanks to the Parameterized Algorithms and Computational Experiments Challenge (PACE) ’16/’17!!!
“Find” tree decompositions of small width?

Works well even for relatively large instances.

Thanks to the Parameterized Algorithms and Computational Experiments Challenge (PACE) ’16/’17!!!
A GPU-based #SAT/WMC-solver

OR how to go parallel?
Dynamic Programming on the GPU

How to parallelize DP?

1. Compute tables for multiple nodes in parallel
   ⇒ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units
   ⇒ Allows with right hindsight for massive parallelization

Why: computation of rows are independent
Dynamic Programming on the GPU

How to parallelize DP?

1. Compute tables for multiple nodes in parallel
   ⇒ Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units
   ⇒ Allows with right hindsight for massive parallelization

Why: computation of rows are independent
Dynamic Programming on the GPU

How to parallelize DP?

1. Compute tables for multiple nodes in parallel
   
   → Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units
   
   → Allows with right hindsight for massive parallelization

Why: computation of rows are independent
Dynamic Programming on the GPU

How to parallelize DP?

1. Compute tables for multiple nodes in parallel
   \[ \Rightarrow \text{Does not allow for immediate massive parallelization due to dependencies to children} \]

2. Distribute computation of rows among different computation units
   \[ \Rightarrow \text{Allows with right hindsight for massive parallelization} \]

Why: computation of rows are independent
Dynamic Programming on the GPU

How to parallelize DP?

1. Compute tables for multiple nodes in parallel
   - Does not allow for immediate massive parallelization due to dependencies to children

2. Distribute computation of rows among different computation units
   - Allows with right hindsight for massive parallelization

Why: computation of rows are independent
Implementation

Disclaimer for theorists: you need to get your hands dirty

+ Right hindsight
Implementation Ideas

Right hindsight?

1. Data structures: a “pixel” represents #solutions store data as
   a. Array (gpuSAT1); improved in gpuSAT2
   b. Compressed partial assignments in BST (gpuSAT2)

2. Avoid Copying:
   Merge small bags (gpuSAT1 < 14, gpuSAT2 hardware dep.)

3. Handle potential VRAM overflow (gpuSAT2):
   Split bags and previously computed solutions
   (if $2^w$ assignments do not fit into the VRAM)

4. Get counters right
(1) Data Structures

a. Array: memory address (plus offset) identifies assignment
   ⇒ Issue: produces lots of memory cells that contain value 0

b. BST (gpuSAT2):
   - Compress Assignments (or address assignments not just by a memory cell)
   - Store only where ≠ 0
   - Idea: use BST; simulate this in an array
     (implement manually on GPU; no libs)
(4) Counters:

- WMC: double or double4 (gpuSAT1)
- \#SAT
  - a. run WMC and use uniform factor (gpuSAT1)
  - b. use logarithmic counters (gpuSAT2)
    - Store floating log-counters
    - Numbers stored in relation to exponent $2^e$ (largest exponent)
    - Dynamically change exponent (keep highest possible precision)

In Practice

- Available on github (GPL3)
- OpenCL: vendor and hardware independent computation framework; C++11
- Works for two graph types: primal, incidence, dual graph
New Architecture (gpuSAT2) [FHZ19]

0. Preprocess $F$

1. Build $G_F$

2. Choose TD $\mathcal{T}$

2b. Preprocess $\mathcal{T}$

3. DP on GPUs

3a. Solution space splitting

3b. Chunk handler

4. Output count

0. Instance Preprocessing

2. Customized Tree Decompositions

3a. Solution Space Splitting

3b. Execute a small GPU-program in a GPU thread (kernel) for each element in $S$

Compress the data and store it in the VRAM (separate GPU-programs)

After all chunks are processed memory regions are merged
0. Instance Preprocessing

2. Customized Tree Decompositions
   (#30; minimize max. card. of intersection of bags at node and its children)

3a. Solution Space Splitting

3b. Execute a small GPU-program in a GPU thread (kernel) for each element in $S$
Compress the data and store it in the VRAM (separate GPU-programs)
After all chunks are processed memory regions are merged
New Architecture (gpuSAT2) [FHZ19]

0. Instance Preprocessing
2. Customized Tree Decompositions
3a. Solution Space Splitting
   (Split larger solutions into smaller portions ⇒ avoid OOM)
3b. Execute a small GPU-program in a GPU thread (kernel) for each element in S
   Compress the data and store it in the VRAM (separate GPU-programs)
   After all chunks are processed memory regions are merged
New Architecture (gpuSAT2) [FHZ19]

0. Instance Preprocessing
1. Customized Tree Decompositions
2. Solution Space Splitting
3a. Execute a small GPU-program in a GPU thread (kernel) for each element in $S$
3b. Compress the data and store it in the VRAM (separate GPU-programs)
   After all chunks are processed memory regions are merged.
Experimental Work

Instances
- 2585 instances from public benchmarks
- \#SAT and WMC

Limits
- Cannot expect to solve instances of high treewidth.

Experiments
1. Distribution of width
2. Benchmarked all solvers that are publicly available
#SAT: Width Comparison (Preprocessing comp.)

- Runtime well below a second (max. 2.5) 0–40; timeout (900s) on 41
- 54% primal treewidth below 30; 70% below 40
- Preprocessing produces TDs of significantly smaller width
WMC: Width Comparison (w/o Preprocessing)

⇒ Produce decompositions of significantly smaller width
Experimental Work (Runtime)

Setting (Runtime Comparision)

Take gpuSAT1, gpuSAT2, and versions as well as sequential and parallel solvers. Consider Wallclock

Hardware

- non-GPU solving: cluster of 9 nodes; each E5-2650 CPUs(12cores) 2.2 GHz, 256 GB RAM; disabled HT, kernel 4.4
- GPU-solving: i3-3245 3.4 GHz; 16 GB RAM; GPU: Sapphire Pulse ITX Radeon RX 570 GPU; 1.24 GHz with 32 compute units, 2048 shader units, 4GB VRAM
Experimental Work (Runtime Disclaimer)

Questionable Setting?
Aren’t you comparing apples and oranges? YES.

Problems of the Setting
- We compare on different hardware
  - Soon, new cluster node with the same specs and two GPUs
- Wallclock is unfair.
  - Usually user is interested in getting things done quickly (+ fairly cheap)
- Power consumption (Joule) and price of investment better measure
  (BUT not accessible with the current framework)
- We use cheap consumer hardware (200 EUR) for the GPU
  not a Tesla K80 (8k EUR) or DGX2 (400k EUR)
- Parallel vs. sequential: No excuse, sorry
<table>
<thead>
<tr>
<th>pmc preprocessing</th>
<th>solver</th>
<th>0-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
<th>51-60</th>
<th>&gt;60</th>
<th>best</th>
<th>unique</th>
<th>( \sum )</th>
<th>time[h]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>miniC2D</td>
<td>1193</td>
<td>29</td>
<td><strong>10</strong></td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>13</td>
<td>0</td>
<td>1242</td>
<td>68.77</td>
</tr>
<tr>
<td></td>
<td>gpuSAT2</td>
<td><strong>1196</strong></td>
<td>32</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>250</td>
<td>8</td>
<td>1229</td>
<td>71.27</td>
</tr>
<tr>
<td></td>
<td>d4</td>
<td>1163</td>
<td>20</td>
<td><strong>10</strong></td>
<td>2</td>
<td>4</td>
<td>28</td>
<td>52</td>
<td>1</td>
<td>1227</td>
<td>76.86</td>
</tr>
<tr>
<td></td>
<td>gpuSAT2(A+B)</td>
<td>1187</td>
<td>18</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>120</td>
<td>7</td>
<td>1206</td>
<td>74.56</td>
</tr>
<tr>
<td></td>
<td>countAntom 12</td>
<td>1141</td>
<td>18</td>
<td><strong>10</strong></td>
<td>5</td>
<td>4</td>
<td>13</td>
<td>101</td>
<td>0</td>
<td>1191</td>
<td>84.39</td>
</tr>
<tr>
<td></td>
<td>c2d</td>
<td>1124</td>
<td>31</td>
<td><strong>10</strong></td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>20</td>
<td>0</td>
<td>1181</td>
<td>84.41</td>
</tr>
<tr>
<td></td>
<td>sharpSAT</td>
<td>1029</td>
<td>16</td>
<td><strong>10</strong></td>
<td>2</td>
<td>4</td>
<td>30</td>
<td><strong>253</strong></td>
<td>1</td>
<td>1091</td>
<td>106.88</td>
</tr>
<tr>
<td></td>
<td>gpuSAT1</td>
<td>1020</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>106</td>
<td>7</td>
<td>1036</td>
<td>114.86</td>
</tr>
<tr>
<td></td>
<td>sdd</td>
<td>1014</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1028</td>
<td>124.23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>without preprocessing</th>
<th>solver</th>
<th>0-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
<th>51-60</th>
<th>&gt;60</th>
<th>best</th>
<th>unique</th>
<th>( \sum )</th>
<th>time[h]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>countAntom 12</td>
<td>118</td>
<td>511</td>
<td>139</td>
<td><strong>175</strong></td>
<td>21</td>
<td><strong>181</strong></td>
<td>318</td>
<td>15</td>
<td>1145</td>
<td>96.64</td>
</tr>
<tr>
<td></td>
<td>d4</td>
<td>124</td>
<td>514</td>
<td>148</td>
<td>162</td>
<td><strong>21</strong></td>
<td>168</td>
<td>69</td>
<td>15</td>
<td>1137</td>
<td>104.94</td>
</tr>
<tr>
<td></td>
<td>c2d</td>
<td>119</td>
<td>525</td>
<td><strong>165</strong></td>
<td>161</td>
<td>18</td>
<td>120</td>
<td>48</td>
<td>15</td>
<td>1108</td>
<td>110.53</td>
</tr>
<tr>
<td></td>
<td>miniC2D</td>
<td>122</td>
<td>514</td>
<td>128</td>
<td>149</td>
<td>9</td>
<td>62</td>
<td>0</td>
<td>0</td>
<td>984</td>
<td>141.22</td>
</tr>
<tr>
<td></td>
<td>sharpSAT</td>
<td>100</td>
<td>467</td>
<td>124</td>
<td>156</td>
<td>12</td>
<td>123</td>
<td><strong>390</strong></td>
<td>4</td>
<td>982</td>
<td>135.41</td>
</tr>
<tr>
<td></td>
<td>gpuSAT2(A+B)</td>
<td><strong>125</strong></td>
<td><strong>539</strong></td>
<td>96</td>
<td>138</td>
<td>0</td>
<td>0</td>
<td>94</td>
<td><strong>19</strong></td>
<td>898</td>
<td>151.16</td>
</tr>
<tr>
<td></td>
<td>gpuSAT2</td>
<td><strong>125</strong></td>
<td>523</td>
<td>96</td>
<td>138</td>
<td>0</td>
<td>0</td>
<td>78</td>
<td>17</td>
<td>882</td>
<td>155.43</td>
</tr>
<tr>
<td></td>
<td>gpuSAT1</td>
<td><strong>125</strong></td>
<td>524</td>
<td>67</td>
<td>140</td>
<td>0</td>
<td>0</td>
<td>82</td>
<td>9</td>
<td>856</td>
<td>162.03</td>
</tr>
<tr>
<td></td>
<td>cachet</td>
<td>99</td>
<td>430</td>
<td>71</td>
<td>152</td>
<td>8</td>
<td>57</td>
<td>3</td>
<td>0</td>
<td>817</td>
<td>176.26</td>
</tr>
</tbody>
</table>
Techniques pay off after preprocessing
<table>
<thead>
<tr>
<th>solver</th>
<th>0-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
<th>51-60</th>
<th>&gt;60</th>
<th>best</th>
<th>unique</th>
<th>∑</th>
<th>time[h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>with pmc*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>miniC2D</td>
<td>858</td>
<td>164</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>13</td>
<td>8</td>
<td>1031</td>
<td>21.29</td>
</tr>
<tr>
<td>gpuSAT1</td>
<td>866</td>
<td>158</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>348</td>
<td>4</td>
<td>1024</td>
<td>18.03</td>
</tr>
<tr>
<td>gpuSAT2(A+B)</td>
<td>866</td>
<td>156</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>343</td>
<td>4</td>
<td>1022</td>
<td>17.86</td>
</tr>
<tr>
<td>gpuSAT2</td>
<td>866</td>
<td>138</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>299</td>
<td>4</td>
<td>1004</td>
<td>22.43</td>
</tr>
<tr>
<td>d4</td>
<td>810</td>
<td>106</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46</td>
<td>0</td>
<td>916</td>
<td>55.36</td>
</tr>
<tr>
<td>cachet</td>
<td>617</td>
<td>128</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>106</td>
<td>1</td>
<td>749</td>
<td>93.65</td>
</tr>
<tr>
<td>d4</td>
<td>82</td>
<td>501</td>
<td>142</td>
<td>156</td>
<td>10</td>
<td>19</td>
<td>111</td>
<td>24</td>
<td>910</td>
<td>53.97</td>
</tr>
<tr>
<td>miniC2D</td>
<td>84</td>
<td>517</td>
<td>134</td>
<td>152</td>
<td>3</td>
<td>4</td>
<td>19</td>
<td>7</td>
<td>894</td>
<td>59.69</td>
</tr>
<tr>
<td>gpuSAT2(A+B)</td>
<td>86</td>
<td>527</td>
<td>98</td>
<td>138</td>
<td>0</td>
<td>0</td>
<td>167</td>
<td>19</td>
<td>849</td>
<td>64.40</td>
</tr>
<tr>
<td>gpuSAT2</td>
<td>86</td>
<td>511</td>
<td>98</td>
<td>138</td>
<td>0</td>
<td>0</td>
<td>131</td>
<td>7</td>
<td>833</td>
<td>68.61</td>
</tr>
<tr>
<td>gpuSAT1</td>
<td>86</td>
<td>513</td>
<td>68</td>
<td>140</td>
<td>0</td>
<td>0</td>
<td>182</td>
<td>10</td>
<td>807</td>
<td>73.78</td>
</tr>
<tr>
<td>cachet</td>
<td>60</td>
<td>447</td>
<td>100</td>
<td>145</td>
<td>2</td>
<td>9</td>
<td>118</td>
<td>1</td>
<td>763</td>
<td>89.80</td>
</tr>
</tbody>
</table>
Contributions

- Established Architecture for DP on the GPU
- Competitive Implementation for #SAT/WMC solving

Benchmark: Comparing apples and oranges

BUT: you compare parallel and sequential solvers.

1. We run on cheap consumer hardware (200 EUR).
2. Cannot measure speedup due to OpenCL limitations
   ⇒ migrate to cuda
Take Home Messages

1. Parameterized Algorithms can actually work
   (Preprocessing is key; some techniques pay only off with right preprocessing)
2. Does it work for SAT? ⇒ we don’t expect so.

Future Work

- Improve current setup by:
  - Portfolio solving; Parallel Usage of GPUs; Alternative Frameworks
- Consider whether stable among different GPU hardware
- Parameters (pswidth)

Sponsors: FWF Y698 & P26696; DFG HO 1294/11-1
Summary contd.

Take Home Messages

1. Parameterized Algorithms can actually work
   (Preprocessing is key; some techniques pay only off with right preprocessing)
2. Does it work for SAT? ⇒ we don’t expect so.

Future Work

- Improve current setup by:
  Portfolio solving; Parallel Usage of GPUs; Alternative Frameworks
- Consider whether stable among different GPU hardware
- Parameters (pswidth)

Thanks for listening!

Sponsors: FWF Y698 & P26696; DFG HO 1294/11-1
References

[AMW17]: Abseher, Musliu, Woltran. htd – A Free, Open-Source Framework for (Customized) Tree Decompositions and Beyond. CPAIOR’17. 2017. doi: 10.1007/978-3-319-59776-8_30


[FHZ19]: Fichte, Hecher, Zisser. gpusat2 – An Improved GPU Model Counter. POS 2019.


gpusat is available at: https://github.com/daajoe/gpusat
Backup Slides
Solving (Width: 0–30): #SAT

kc/cdcl: c2d, d4, dsharp
dp: gpusat, dynQBF, dynasp parallel: countAntom, gpusat
cdcl: Cachet, sharpSAT, clasp
bdd: sdd
approx: approxmc, sts
### Solving: 

<table>
<thead>
<tr>
<th>solver</th>
<th>0-20</th>
<th>21-30</th>
<th>31-40</th>
<th>41-50</th>
<th>51-60</th>
<th>&gt;60</th>
<th>best</th>
<th>∑ rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2d</td>
<td>164</td>
<td>519</td>
<td><strong>175</strong></td>
<td>116</td>
<td>20</td>
<td>118</td>
<td>120</td>
<td>1112</td>
</tr>
<tr>
<td>Cachet</td>
<td>133</td>
<td>421</td>
<td>91</td>
<td>109</td>
<td>8</td>
<td>58</td>
<td>13</td>
<td>820</td>
</tr>
<tr>
<td>d4</td>
<td><strong>169</strong></td>
<td>510</td>
<td>156</td>
<td>119</td>
<td><strong>23</strong></td>
<td><strong>162</strong></td>
<td>191</td>
<td><strong>1139</strong></td>
</tr>
<tr>
<td>gpusat(p)</td>
<td><strong>169</strong></td>
<td>523</td>
<td>79</td>
<td>104</td>
<td>0</td>
<td>0</td>
<td>88</td>
<td>875</td>
</tr>
<tr>
<td>miniC2D</td>
<td>167</td>
<td>491</td>
<td>137</td>
<td>103</td>
<td>8</td>
<td>67</td>
<td>2</td>
<td>973</td>
</tr>
<tr>
<td>sharpSAT</td>
<td>136</td>
<td>465</td>
<td>136</td>
<td>112</td>
<td>11</td>
<td>124</td>
<td>483</td>
<td>984</td>
</tr>
<tr>
<td>sts</td>
<td>162</td>
<td>448</td>
<td>101</td>
<td><strong>146</strong></td>
<td>10</td>
<td>45</td>
<td>252</td>
<td>912</td>
</tr>
</tbody>
</table>

**Table:** Number of counting instances solved by solver and interval.
Empirical Work (first approach)

Observations

- Implementation is fairly naive
- Still: competitive up to width 30
- Requirement: obtain decompositions fast
- Width was surprisingly small (different for SAT)
(1) Data Structures

b. BST (details):

- Continuous sequence 64-bit unsigned integers (cells)
- Cell: empty, index, and value (counter)
- Index cells: lower 32 bits index to the next cell (lower bits assignment 0, upper 1)
- Handle Sync (between parallel threads) by keeping track of the current size (number of allocated cells; prevent to allocate cell again)
Solving #SAT [SamerSzeider10]

$$\varphi = (\neg a \lor b \lor x) \land (a \lor b) \land (c \lor \neg x) \land (b \lor \neg c) \land (\neg b \lor \neg c \lor \neg y)$$

1. Create graph representation
2. Decompose graph
3. Solve problems via $S$
4. Combine solutions

Nice Tree Decompositions
(note example left is not nice)

LEAF.: Put empty set and counter 1

INTR.: Guess truth value and check satisfiability

REMOVE: Remove $a$ from each assignment (row) in the table and sum up the counters if we get multiple assignments with the same data

JOIN: Match rows with the same assignment and multiply the counters

“Local formula” $F_t$ clauses whose variables are contained in the bag (colored in red above)
Algorithm for Primal Graph

**In:** Node $t$, bag $\chi_t$, clauses $F_t$, sequence $C$ of tables.

**Out:** Table $\text{tab}_t$

1. **if** type($t$) = leaf **then**
   
2. \[ \text{tab}_t \leftarrow \{ \emptyset \} \]

3. **else if** type($t$) = intr and $a \in \chi_t \setminus \chi_{t'}$, **then**

4. \[ \text{tab}_t \leftarrow \left\{ \tau \cup \{a\} \mid \tau \in \text{tab}'' \land \tau \cup \{a\} \models F_t \right\} \cup \]

5. \[ \left\{ \tau \mid \tau \in \text{tab}'' \land \tau \models F_t \right\} \]

6. **else if** type($t$) = rem and $a \in \chi_{t'} \setminus \chi_t$ **then**

7. \[ \text{tab}_t \leftarrow \left\{ \tau \setminus \{a\} \mid \tau \in \text{tab}'' \right\} \]

8. **else if** type($t$) = join **then**

9. \[ \text{tab}_t \leftarrow \left\{ \tau \mid \tau \in \text{tab}'' \land \tau \in \text{tab}'' \right\} \]

10. **return** $\text{tab}_t$