SAT Solving with GPU-Accelerated Inprocessing

TACAS'21

Muhammad Osama[1] – Anton Wijs[1] – Armin Biere[2]

[1] Eindhoven University of Technology, The Netherlands
[2] Johannes Kepler University, Austria

Netherlands Organisation for Scientific Research

GEARS:GPU-Enabled Accelerated Reasoning about Systems design

Satisfiability (SAT) Solving – Warm up

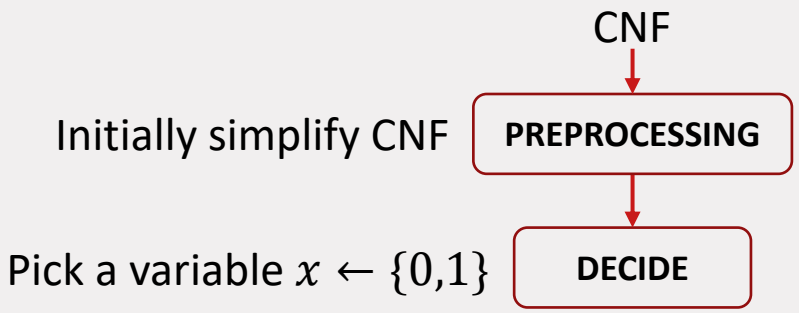
- A problem of finding a satisfying assignment for a Boolean formula
  - If such assignment exists, the problem is SATISFIABLE (SAT)
  - Otherwise, it's UNSATISFIABLE (UNSAT)
  - Usually presented in Conjunctive Normal Form (CNF)

$$F = (a \vee b \vee c) \wedge (\neg a \vee b) \wedge (\neg b \vee c)$$

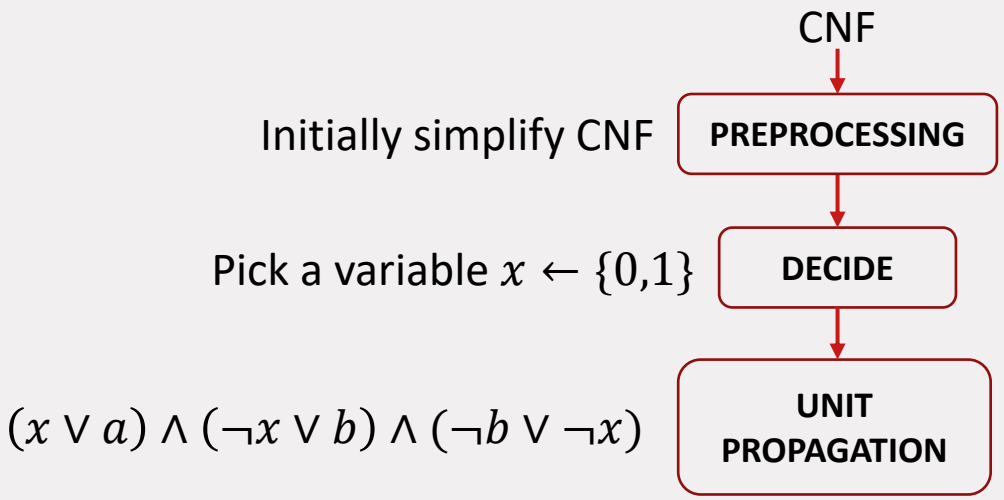
  - Modern SAT solvers use **Conflict-Driven Clause-Learning (CDCL)** search algorithm + simplifications

SAT Solving – Warm up

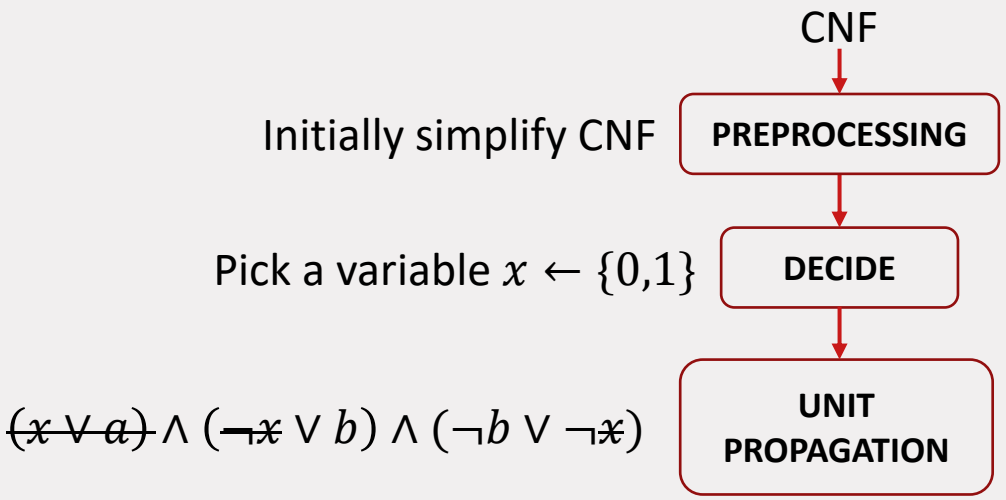
SAT Solving – Warm up



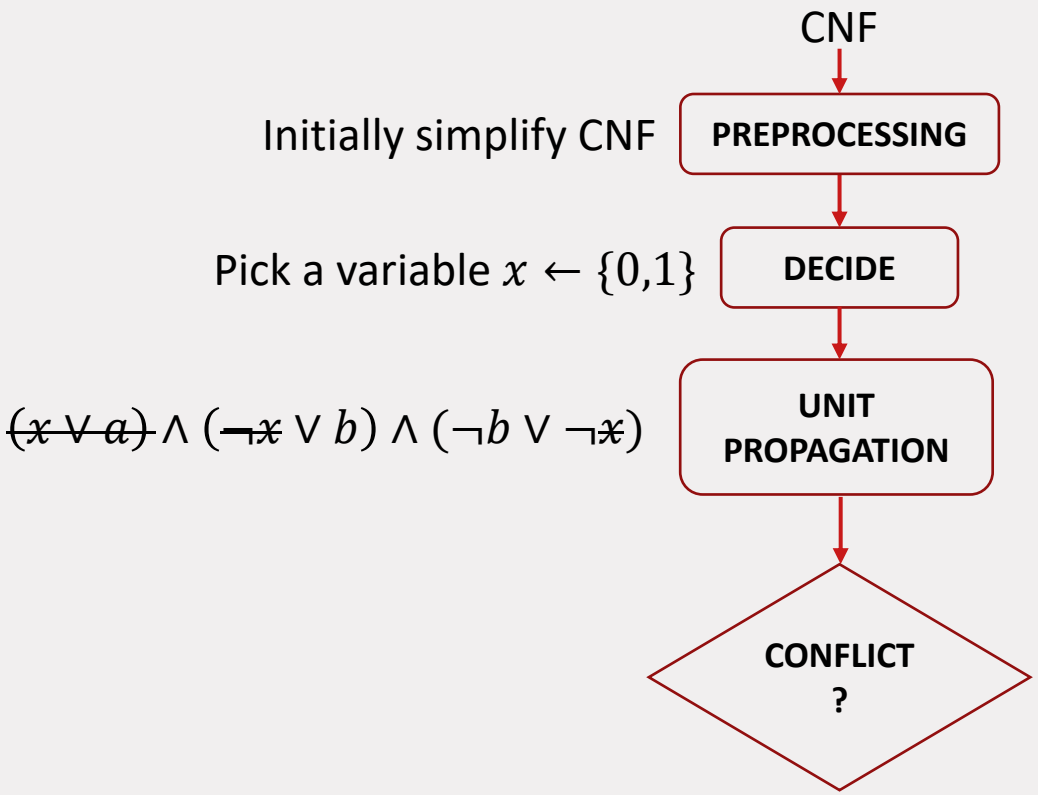
SAT Solving – Warm up



SAT Solving – Warm up

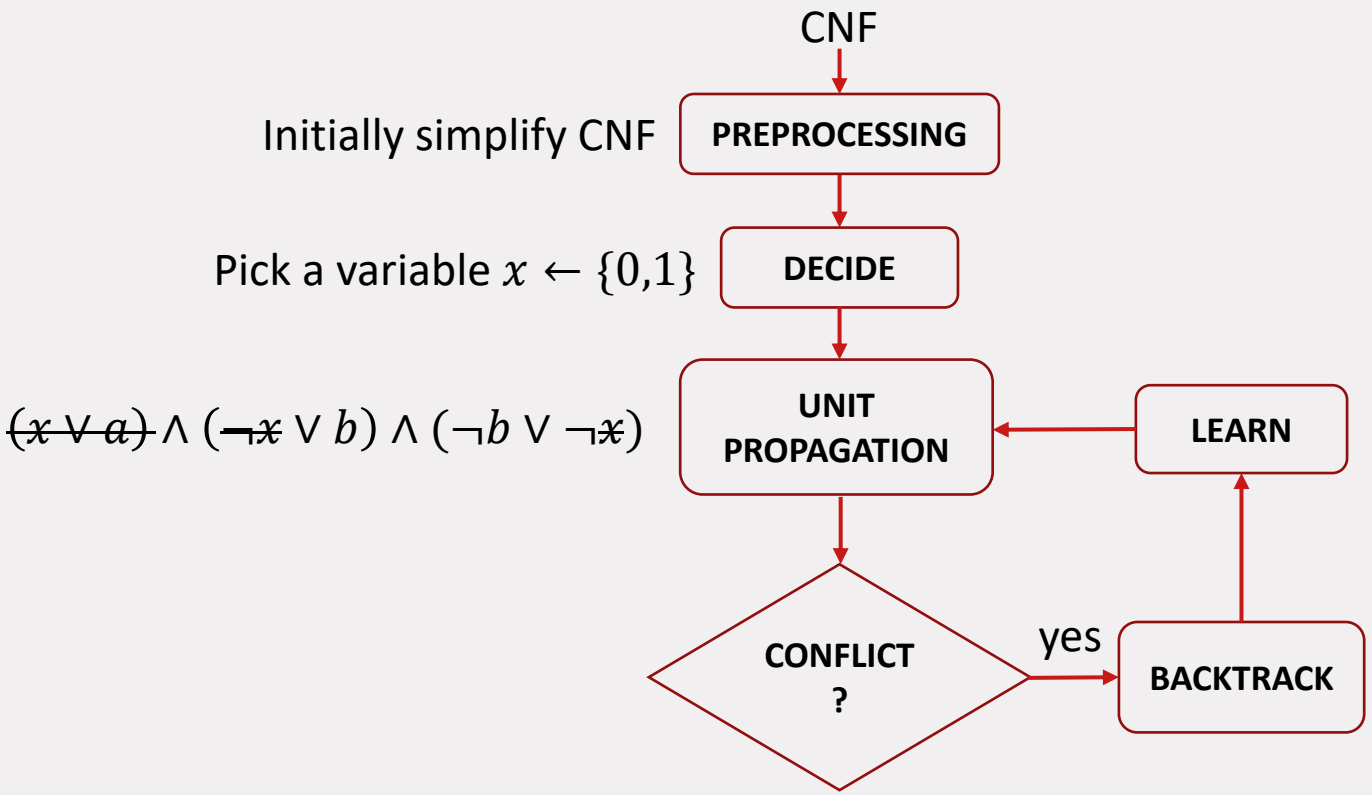


SAT Solving – Warm up



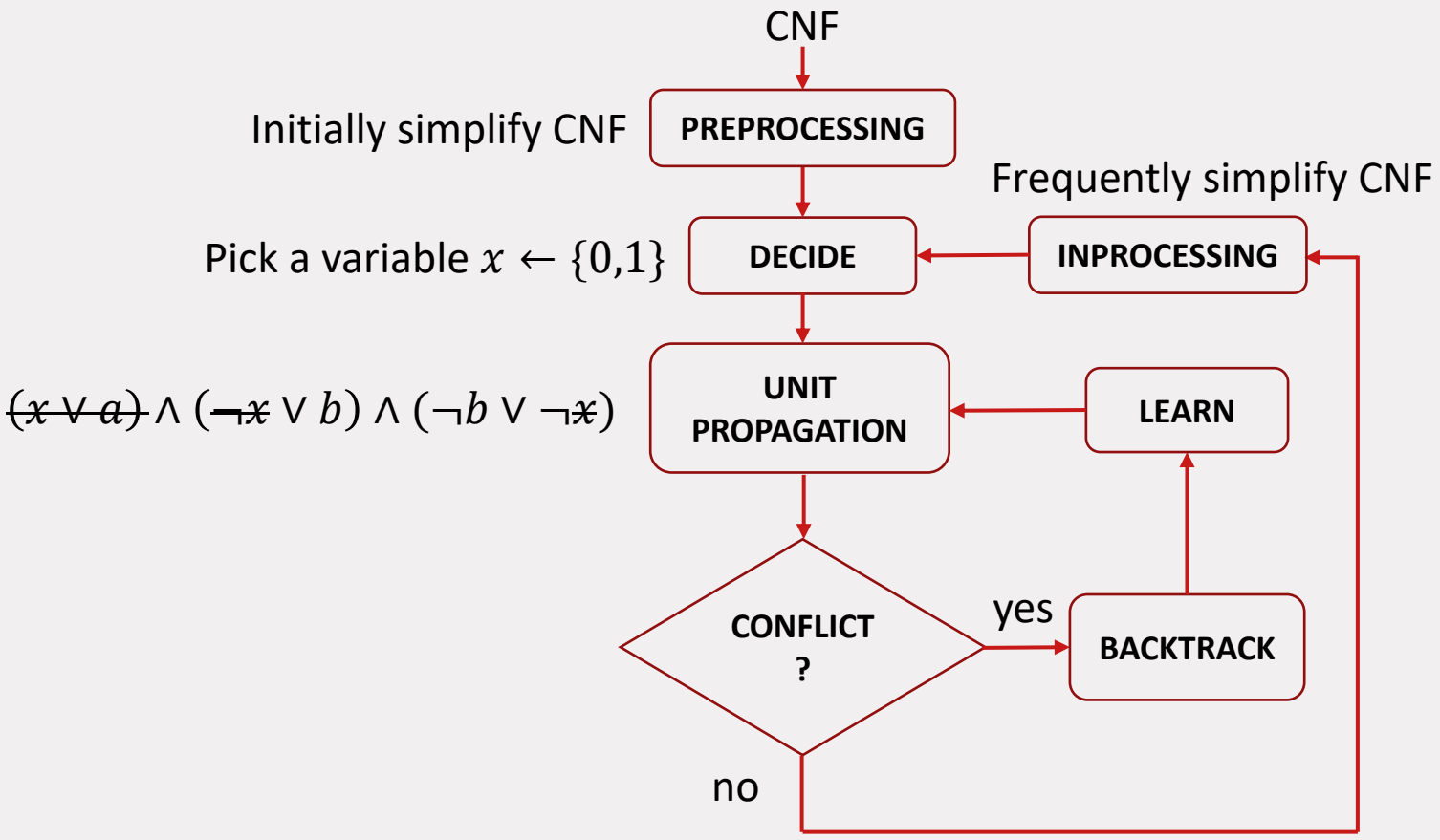
TU/e

SAT Solving – Warm up



TU/e

SAT Solving – Warm up



Contributions

GPU-accelerated Inprocessing

Challenge	Proposed solution
GPU has a very limited memory (24 GBs in powerful GPUs)	✓ New **space-efficient data structure** ✓ **Parallel garbage collection**
Our previous work in parallel variable elimination couldn't reproduce results and heavily relies on atomics	✓ Garbage collector designed to preserve the order of clauses ✓ **3-phase parallel variable elimination**
Many redundant clauses but not all simplifications are suitable for GPU	✓ New simplification method called **eager redundancy elimination** suitable for data parallelism

Data Structure

```
class SCLAUSE {
    char state, used;
    char flag, added;
    uint32 lbd, sig;
    int size;
    uint32 literals[1];
};
```

Clause container

```
class CNF {
    struct {
        uint32* memory;
        uint64 size, cap;
    } clauses;
    struct {
        uint64* memory;
        uint64 size, cap;
    } references;
}
```

CNF container

Data Structure

```
class SCLAUSE {
    char state, used;
    char flag, added;
    uint32 lbd, sig;
    int size;
    uint32 literals[1];
};
```

Clause container

```
class CNF {
    struct {
        uint32* memory;
        uint64 size, cap;
    } clauses;
    struct {
        uint64* memory;
        uint64 size, cap;
    } references;
}
```

CNF container

Example: $C = \{ a \vee \neg b \vee c \}$

Data Structure

```
class SCLAUSE {
    char state, used;
    char flag, added;
    uint32 lbd, sig;
    int size;
    uint32 literals[1];
};
```

Clause container

```
class CNF {
    struct {
        uint32* memory;
        uint64 size, cap;
    } clauses;
    struct {
        uint64* memory;
        uint64 size, cap;
    } references;
}
```

CNF container

Example: $C = \{ a \vee \neg b \vee c \}$

```
bytes = 20 + 4(size - 1) = 28
buckets = bytes / 4
        = size + 4 = 7
```

Data Structure

```
class SCLAUSE {
    char state, used;
    char flag, added;
    uint32 lbd, sig;
    int size;
    uint32 literals[1];
};
```

Clause container

```
class CNF {
    struct {
        uint32* memory;
        uint64 size, cap;
    } clauses;
    struct {
        uint64* memory;
        uint64 size, cap;
    } references;
}
```

CNF container

Example: $C = \{ a \vee \neg b \vee c \}$

```
bytes = 20 + 4(size - 1) = 28
buckets = bytes / 4
        = size + 4 = 7
```

Data Structure

```
class SCLAUSE {
    char state, used;
    char flag, added;
    uint32 lbd, sig;
    int size;
    uint32 literals[1];
};
```

Clause container

```
class CNF {
    struct {
        uint32* memory;
        uint64 size, cap;
    } clauses;
    struct {
        uint64* memory;
        uint64 size, cap;
    } references;
}
```

CNF container

Example: $C = \{ a \vee \neg b \vee c \}$

```
bytes = 20 + 4(size - 1) = 28
buckets = bytes / 4
        = size + 4 = 7
```

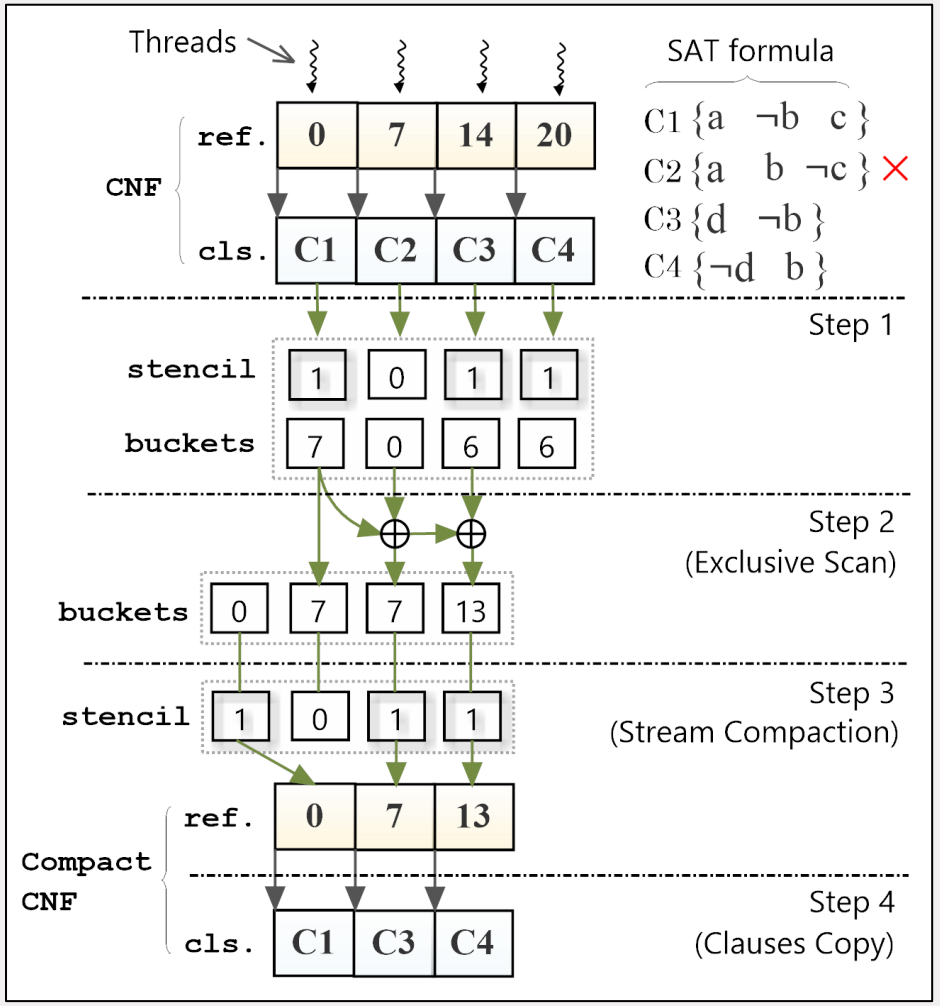
```
cptr =
(SCLAUSE*)(clauses.memory + 0)

  references.memory[0] = 0
```


TU/e

Parallel Garbage Collection

- Stencil: a binary array to mark the existence of an element
- Exclusive scan[1]: a parallel prefix sum offset by one position to the right
- Stream compaction[1]: compacts sparse data in parallel to consecutive elements

[1] https://nvlabs.github.io/cub/



Bounded Variable Elimination (BVE) - Recap

- Resolution rule: $\dfrac{(x \vee \ell \vee \dots) \qquad (\neg x \vee \ell' \vee \dots)}{(\ell \vee \dots \vee \ell' \vee \dots)}$

Bounded Variable Elimination (BVE) - Recap

- Resolution rule: $$\frac{(x \vee \ell \vee \dots) \qquad (\neg x \vee \ell' \vee \dots)}{(\ell \vee \dots \vee \ell' \vee \dots)}$$

- Example: $S = (x \vee a) \wedge (x \vee b) \wedge (\neg x \vee c)$

Bounded Variable Elimination (BVE) - Recap

- Resolution rule: $$\frac{(x \vee \ell \vee \dots) \qquad (\neg x \vee \ell' \vee \dots)}{(\ell \vee \dots \vee \ell' \vee \dots)}$$

- Example: $S = (x \vee a) \wedge (x \vee b) \wedge (\neg x \vee c)$

$$S = \underbrace{(a \vee c) \wedge (b \vee c)}$$

resolvents

Bounded Variable Elimination (BVE) - Recap

- Resolution rule: $\dfrac{(x \vee \ell \vee \ldots) \qquad (\neg x \vee \ell' \vee \ldots)}{(\ell \vee \ldots \vee \ell' \vee \ldots)}$

- Gate equivalence reasoning can be also used to find patterns of logical gates (e.g. AND, OR, XOR, …) and substitute them with their definitions
- Gate substitution reduces the number of resolvents

Parallel BVE

- In previous work (TACAS19), we proposed our parallel version of BVE on the GPU where threads can eliminate *mutually-independent* variables in parallel

Parallel BVE

- In previous work (TACAS19), we proposed our parallel version of BVE on the GPU where threads can eliminate *mutually-independent* variables in parallel
- Achieved a speedup of 32x compared to sequential version
- Had two shortcomings:
  - Heavily relies on atomics to add new resolvents
  - Resolvents are added to random positions (side effect of parallel atomics)

3-Phase Parallel BVE

3-Phase Parallel BVE

0: Initially

- occurrence table (T) for all literals in formula (S) is created on the GPU
- A list (ϕ) of independent variables is generated by the variable scheduler on the CPU side
- **For all** $x \in \phi$ **in parallel**, sort(T, *key*)

3-Phase Parallel BVE

0: Initially

- occurrence table (T) for all literals in formula (S) is created on the GPU
- A list (ϕ) of independent variables is generated by the variable scheduler on the CPU side
- **For all** $x \in \phi$ **in parallel**, sort(T, *key*)

1: **For all** $x \in \phi$ **in parallel**, Identify the type of elimination and calculate the resolvents info.

type, *added*, *buckets* $\leftarrow$ **`variableSweep`**(ϕ, S, T)

3-Phase Parallel BVE

0: Initially

- occurrence table (T) for all literals in formula (S) is created on the GPU
- A list (ϕ) of independent variables is generated by the variable scheduler on the CPU side
- **For all** $x \in \phi$ **in parallel**, sort(T, *key*)

1: **For all** $x \in \phi$ **in parallel**, Identify the type of elimination and calculate the resolvents info.

type, *added*, *buckets* ← **`variableSweep`**(ϕ, S, T)

2: Compute the new positions of resolvents using parallel prefix sum

buckets ← **`exclusiveSum`**(*buckets*, **`size`**(*clauses*));

old size of all clauses in buckets

added ← **`exclusiveSum`**(*added*, **`size`**(*references*));

old number of clauses

3-Phase Parallel BVE

0: Initially

- occurrence table (T) for all literals in formula (S) is created on the GPU
- A list (ϕ) of independent variables is generated by the variable scheduler on the CPU side
- **For all** $x \in \phi$ **in parallel**, sort(T, *key*)

1: **For all** $x \in \phi$ **in parallel**, Identify the type of elimination and calculate the resolvents info.

type, *added*, *buckets* ← **`variableSweep`**(ϕ, S, T)

2: Compute the new positions of resolvents using parallel prefix sum

buckets ← **`exclusiveSum`**(*buckets*, **`size`**(*clauses*));

added ← **`exclusiveSum`**(*added*, **`size`**(*references*));

old size of all clauses in buckets

old number of clauses

3: **For all** $x \in \phi$ **in parallel**, resolve or substitute x

S, *units* ← **`variableResolvent`**(ϕ, S, T, *type*, *added*, *buckets*)

Eager Redundancy Elimination (ERE)

- We observed that unit propagation results in lots of redundancies
- ERE is a combination of resolution + equivalence:

> For a given formula S and $C_1 \in S, C_2 \in S$ with $x \in C_1, \neg x \in C_2$ for some variable x, if there exists a clause $C \in S$ such that $C \equiv C_1 \otimes_x C_2$ then $S = S \setminus C$

- Since we learn clauses during SAT solving, another constraint is checked before the removal of the redundancy

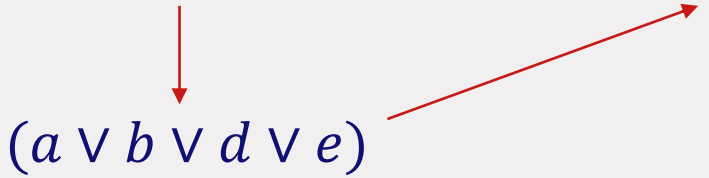
> (C_1 is LEARNT ∨ C_2 is LEARNT) ⇒ C is LEARNT

Eager Redundancy Elimination (ERE)

- Example: $(a \vee b \vee \neg c) \wedge (d \vee e \vee c) \wedge (a \vee b \vee d \vee e)$

Eager Redundancy Elimination (ERE)

- Example: $(a \vee b \vee \neg c) \wedge (d \vee e \vee c) \wedge (a \vee b \vee d \vee e)$



Eager Redundancy Elimination (ERE)

- Example: $(a \vee b \vee \neg c) \wedge (d \vee e \vee c) \wedge \cancel{(a \vee b \vee d \vee e)}$

Eager Redundancy Elimination (ERE)

- Example: $(a \vee b \vee \neg c) \wedge (d \vee e \vee c) \wedge \cancel{(a \vee b \vee d \vee e)}$

- **For all** $x \in \phi$ **in parallel**, resolve(x) to obtain the resolvents

TU/e

Eager Redundancy Elimination (ERE)

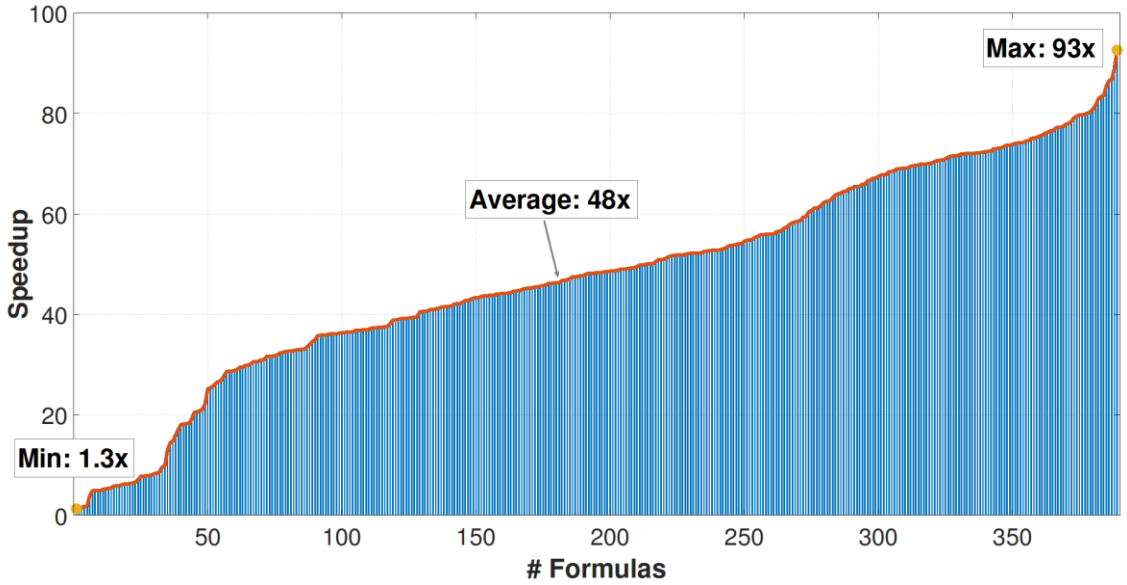
- Example: $(a \vee b \vee \neg c) \wedge (d \vee e \vee c) \wedge \cancel{(a \vee b \vee d \vee e)}$

- **For all** $x \in \phi$ **in parallel**, resolve(x) to obtain the resolvents

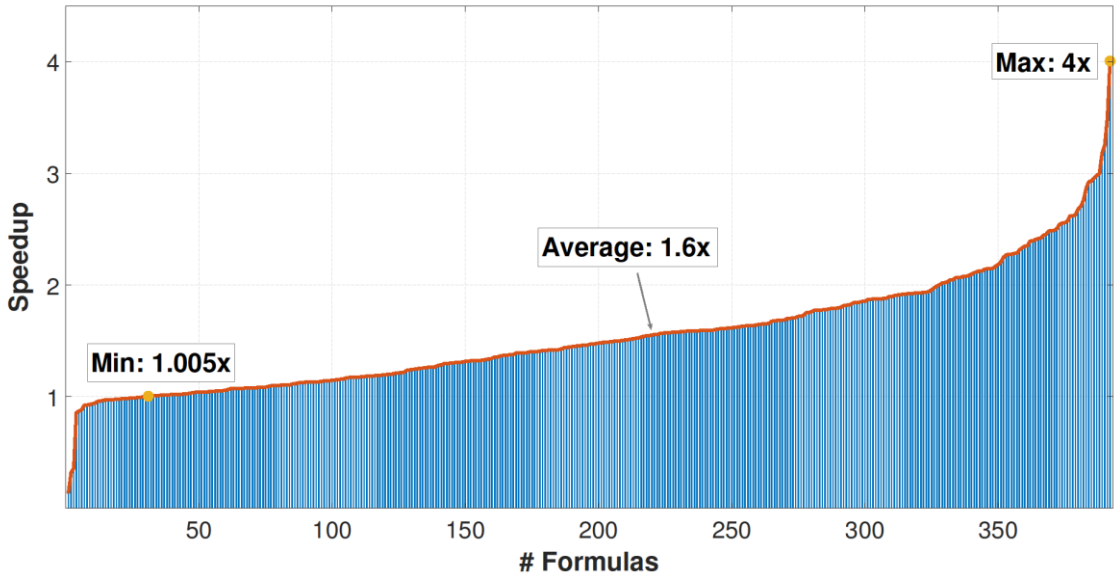
- For each resolvent, search for an equivalent clause in parallel on the fly

Benchmarks

Parallel GC vs a sequential version

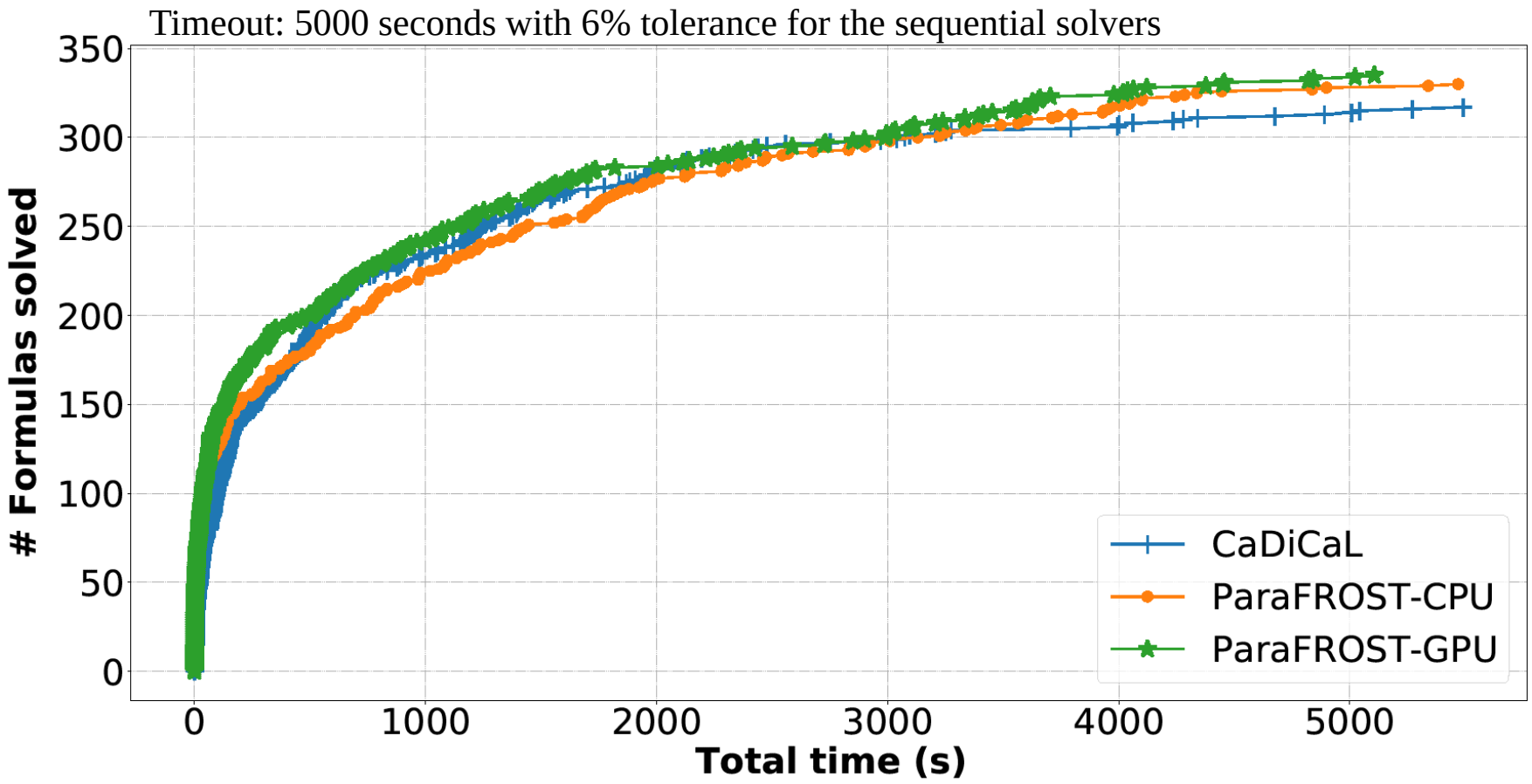


3-Phase BVE vs atomic version



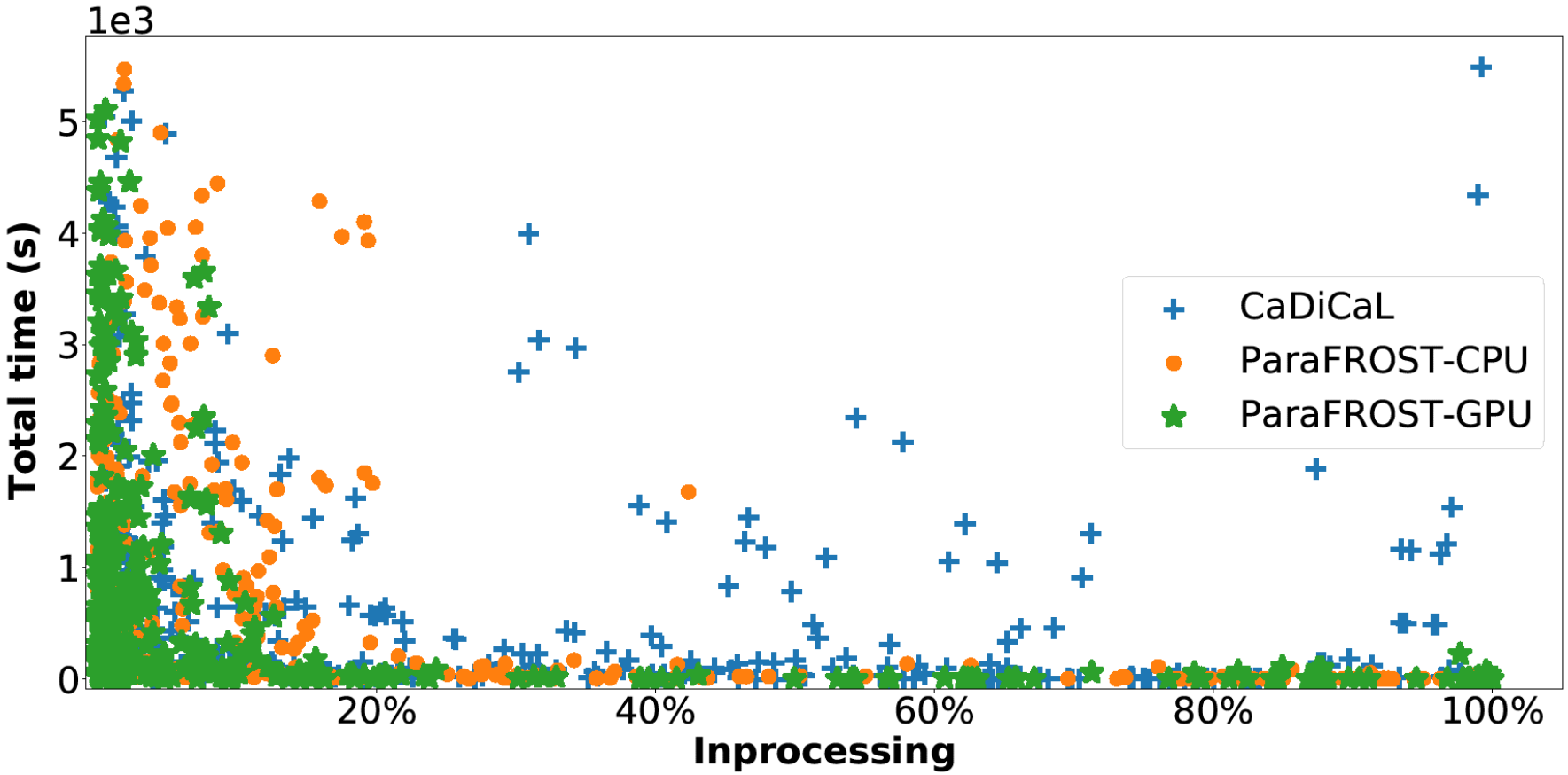
TU/e

Benchmarks



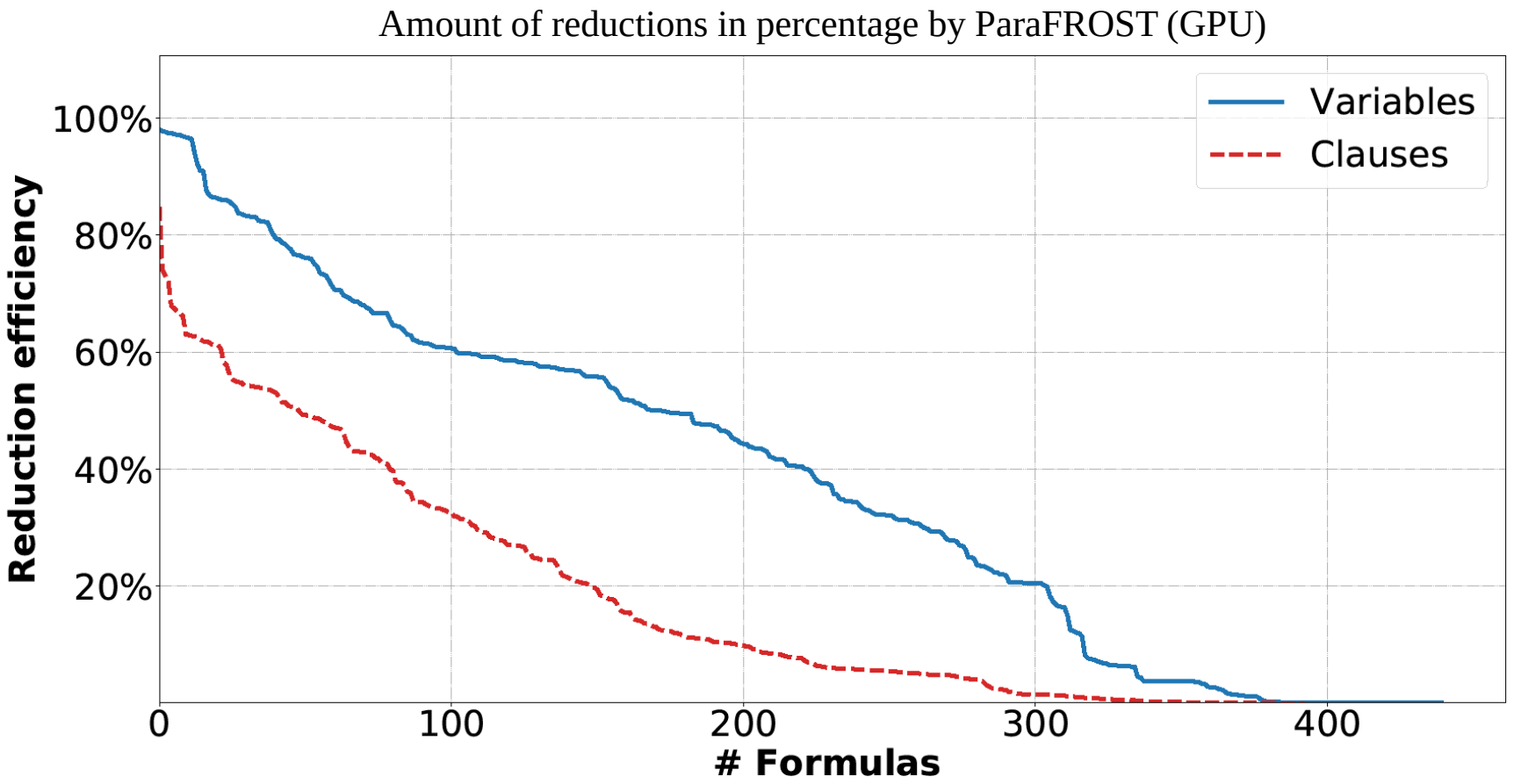
TU/e

Benchmarks



TU/e

Benchmarks



TU/e

Summary

- We implemented a new GPU solver with space-efficient data structure
- A parallel garbage collector was introduced for the first time for SAT applications
- The new parallel BVE was twice as fast and results can be reproduced
- Future work may include generating proofs for the GPU work (idea by Armin) and implementing more simplfications on the GPU

Thank you

Artifact: https://gears.win.tue.nl/software/parafrost

ParaFROST: https://github.com/muhos/ParaFROST

NWO Netherlands Organisation for Scientific Research

GEARS:GPU-Enabled Accelerated Reasoning about Systems design

TU/e