Relieving Capacity Limits on FPGA-Based SAT Solvers

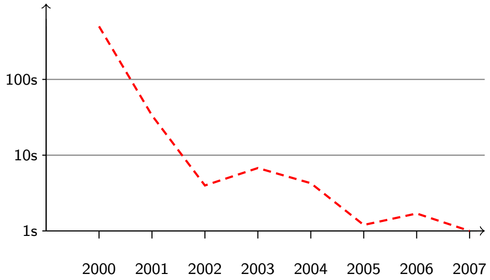
Leopold Haller [1] Satnam Singh [2]

[1]Oxford University Computing Laboratory

[2]Microsoft Research Cambridge

FMCAD 2010

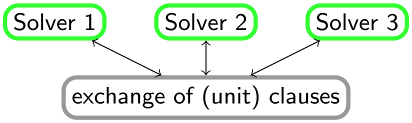
The quest for ever more efficient SAT solvers



SAT solving performance (from [MZ09])

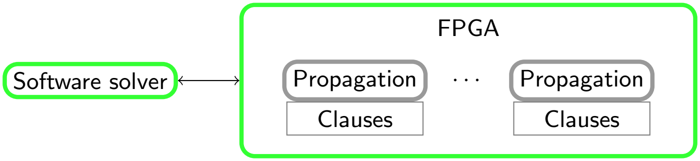
Parallelising SAT

Software (ManySAT, Plingeling):



Coarse-grain parallelism

Hardware (e.g., [DTYZ08]):



Fine-grain parallelism

Challenges in Hardware-Based CDCL

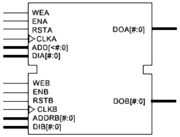
```
while true do
    if ¬decide() then
        return SAT;
    while BCP() = conflict do
        analyseAndLearn();
        if ¬backtrack() then
            return UNSAT;
```

Challenges in Hardware-Based CDCL

```
while true do
    if ¬decide() then
        return SAT;
    while BCP() = conflict do
        analyseAndLearn();
        if ¬backtrack() then
            return UNSAT;
```

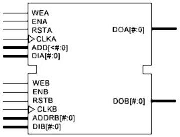
- Memory access in Boolean Constraint Propagation is hard to predict. Existing approaches rely on *fast on-chip memory*.

Memory resources



on-chip	off-chip
Block RAM	SDRAM
hundreds of small composable blocks parallel access	large atomic unit of memory sequential access
one-cycle access	non-deterministic delay
uniform access speed	streaming access faster
0.5-7MB (Virtex-5)	multiple GB

Memory resources

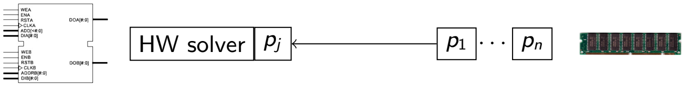


on-chip	**off-chip**
Block RAM	SDRAM
hundreds of small composable blocks parallel access	large atomic unit of memory sequential access
one-cycle access	non-deterministic delay
uniform access speed	streaming access faster
0.5-7MB (Virtex-5)	multiple GB

Existing approaches store instance information on-chip
Capacity limits: e.g., [DTYZ08] - 64K variables/clauses

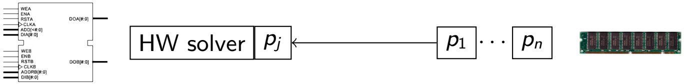
Relieving capacity limits

Previously suggested: Instance partitioning



Relieving capacity limits

Previously suggested: Instance partitioning

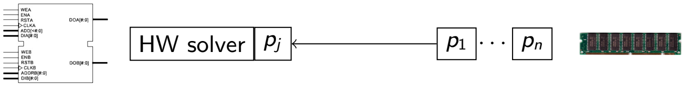


We explore the feasibility of

- *directly utilizing off-chip storage*
- by building a *custom memory hierarchy* for SAT algorithms.

Relieving capacity limits

Previously suggested: Instance partitioning



We explore the feasibility of

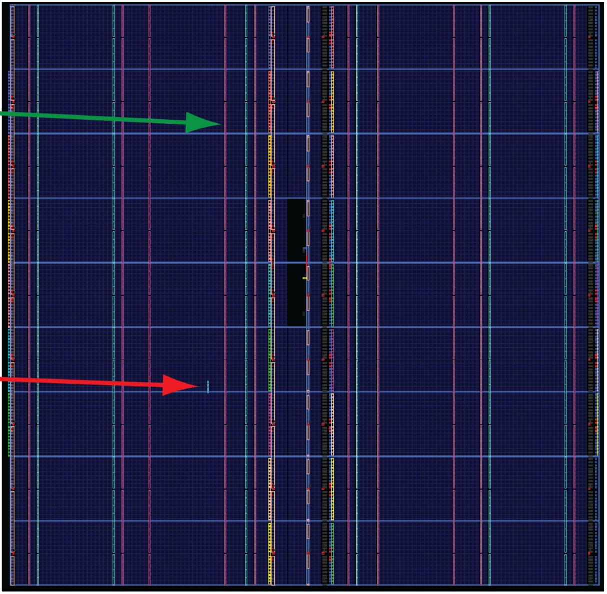
- *directly utilizing off-chip storage*
- by building a *custom memory hierarchy* for SAT algorithms.

Our platform is the
BEE3 (Berkeley Emulation Engine v3)

- 4 interconnected Virtex5 FPGAs
- 2 independent memory channels
- 8GB per channel (64GB in total)
- Ethernet, RS232, etc.

Xilinx Virtex-6 FPGA
XCVLX760
758,784 logic cells
864 DSP blocks
1,440 dual ported 18Kb RAMs

32-bit adder
(32/474,240 LUTs)
> 700MHz



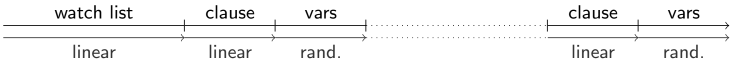
BCP in CDCL algorithms

```
procedure bcp($l$: $literal$)
    $wl$ ← readWatchlist($l$);
    $r \leftarrow \emptyset$;
    for $address \in wl$ do
        clause ← readClause($address$);
        vals ← readVarValues($vals$);
        $r \leftarrow r \cup$ propagate($clause$, $vals$);
        //returns conflict, deduction or move WL
    processResults($r$); writeResults($r$);
end procedure
```

BCP in CDCL algorithms

```
procedure bcp($l$: $literal$)
    $wl$ ← readWatchlist($l$);
    $r$ ← $\emptyset$;
    for $address \in wl$ do
        clause ← readClause($address$);
        vals ← readVarValues($vals$);
        $r$ ← $r \cup$ propagate($clause$, $vals$);
        //returns conflict, deduction or move WL
    processResults($r$); writeResults($r$);
end procedure
```

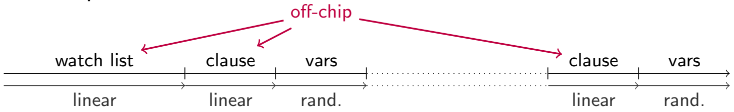
Read access pattern:



BCP in CDCL algorithms

```
procedure bcp($l$: $literal$)
    $wl$ ←readWatchlist($l$);
    $r \leftarrow \emptyset$;
    for $address \in wl$ do
        clause ← readClause($address$);
        vals ← readVarValues($vals$);
        $r \leftarrow r \cup$ propagate($clause$, $vals$);
        //returns conflict, deduction or move WL
    processResults($r$); writeResults($r$);
end procedure
```

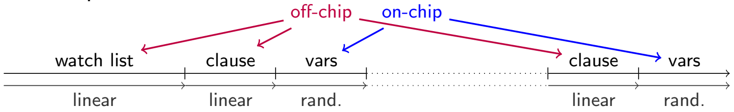
Read access pattern:



BCP in CDCL algorithms

```
procedure bcp(l: literal)
    wl ← readWatchlist(l);
    r ← ∅;
    for address ∈ wl do
        clause ← readClause(address);
        vals ← readVarValues(vals);
        r ← r ∪ propagate(clause, vals);
        //returns conflict, deduction or move WL
    processResults(r); writeResults(r);
end procedure
```

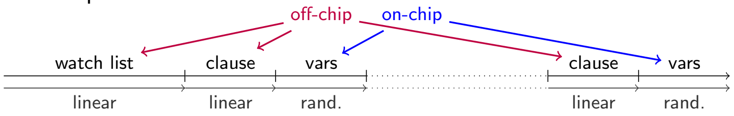
Read access pattern:



BCP in CDCL algorithms

```
procedure bcp($l$: $literal$)
    $wl$ ← readWatchlist($l$);
    $r$ ← $\emptyset$;
    for $address \in wl$ do
        clause ← readClause($address$);
        vals ← readVarValues($vals$);
        $r$ ← $r \cup$ propagate($clause$, $vals$);
        //returns conflict, deduction or move WL
    processResults($r$); writeResults($r$);
end procedure
```

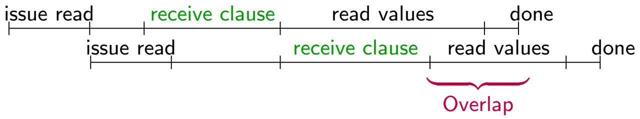
Read access pattern:



Parallelise?

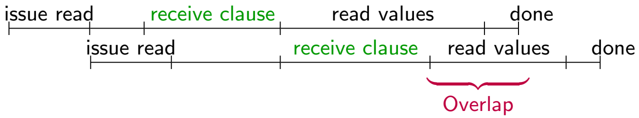
Parallel Deduction

Possible overlap while processing clauses:

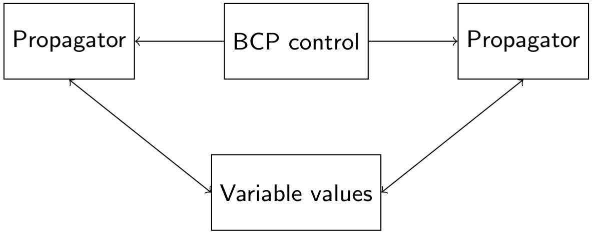


Parallel Deduction

Possible overlap while processing clauses:



Use of multiple propagation units:

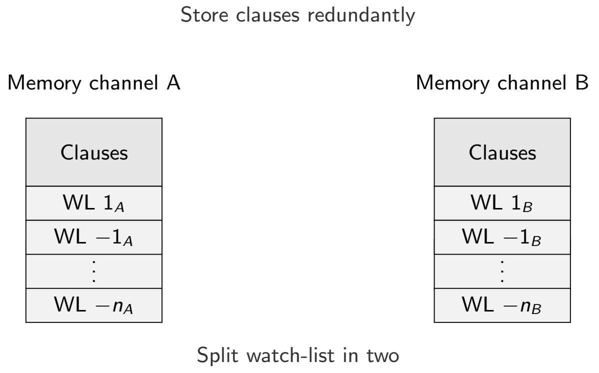


Parallel watched literals

Utilize the two independent memory channels.

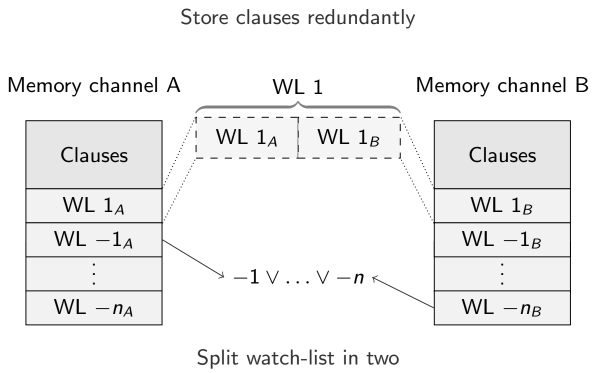
Parallel watched literals

Utilize the two independent memory channels.



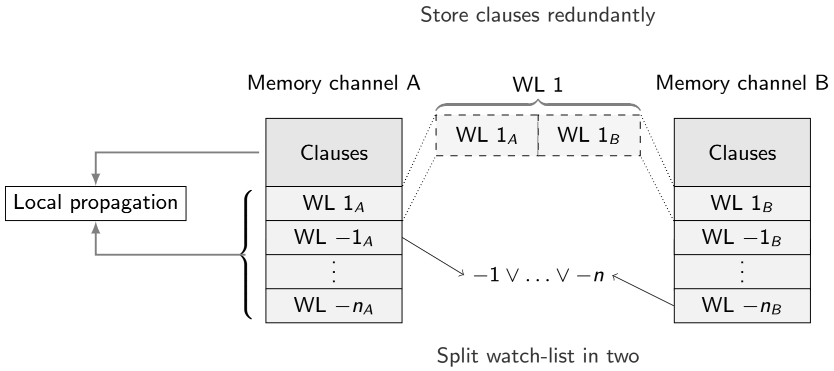
Parallel watched literals

Utilize the two independent memory channels.

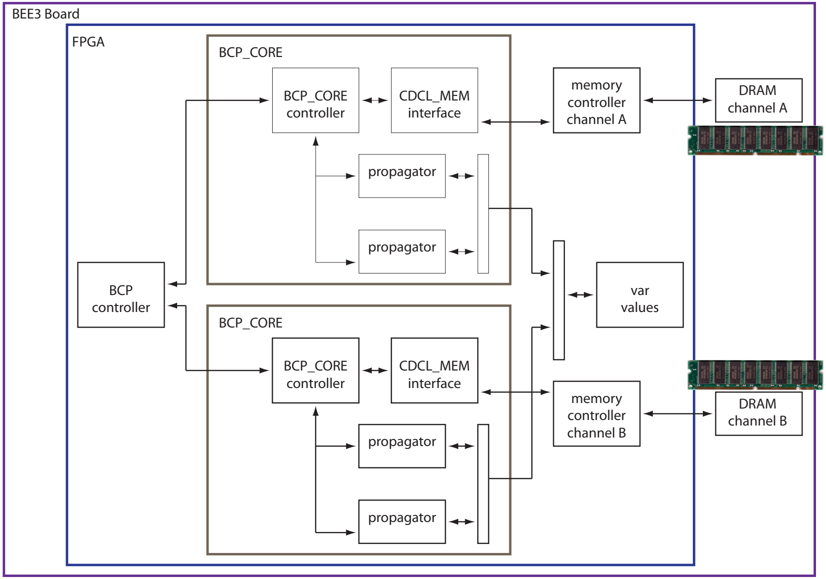


Parallel watched literals

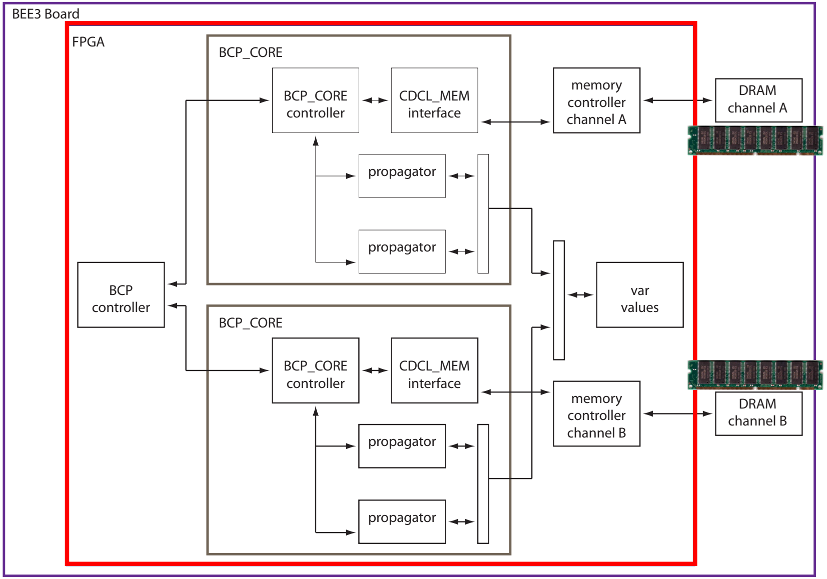
Utilize the two independent memory channels.



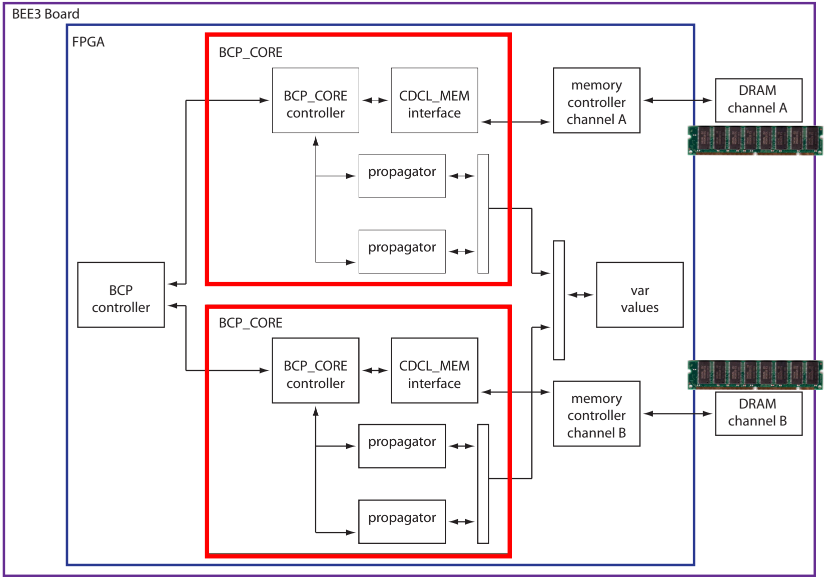
Architecture



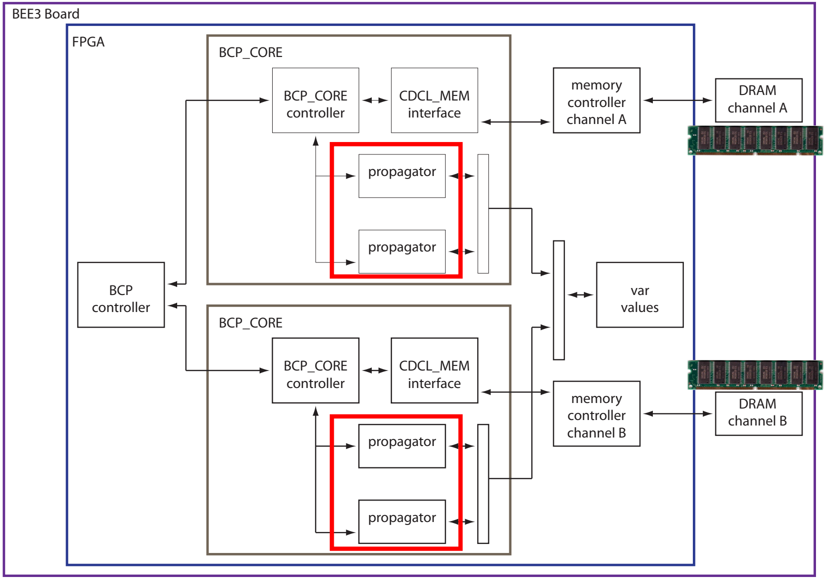
Architecture



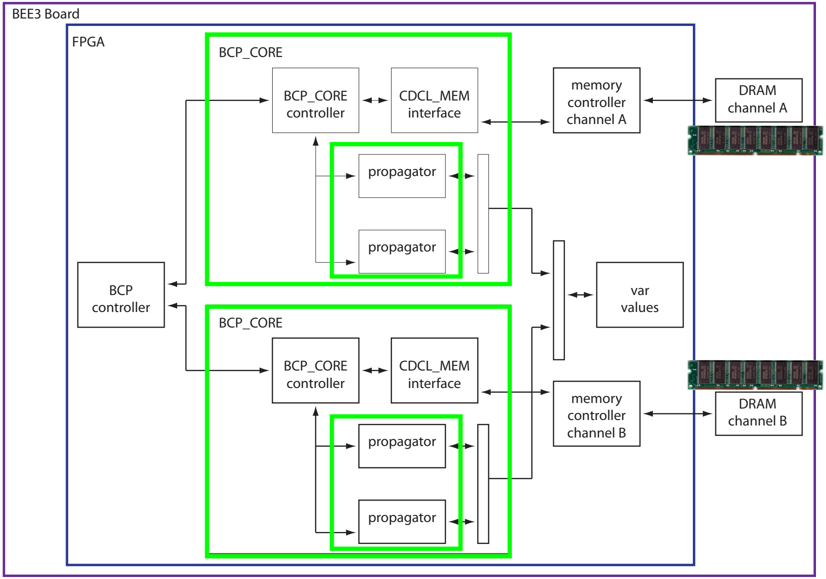
Architecture



Architecture



Architecture



Progress

- Implementation of a BCP core on the BEE3 board.

- Validated in simulation, synthesized on the board
  - 15000 LUTs (20%)
  - 100MHz control logic, 250MHz memory

Size limits (Virtex5 w. 16GB)	
clauses (max. 24 literals)	ca. 70.000.000
variables	ca. 1.000.000

- Obtaining benchmark results is work in progress

Future work & Conclusion

Future work

- Application specific caches
- Parallelization through use of multiple FPGA chips

Future work & Conclusion

Future work

- Application specific caches
- Parallelization through use of multiple FPGA chips

Architecture for a hardware-based BCP core that:

- Achieves significantly *higher capacity* than existing approaches by directly accessing *large off-chip memory resources*.
- Employs fine-grain parallelisation strategies.

Thank you for your attention.

John D. Davis, Zhangxi Tan, Fang Yu, and Lintao Zhang.
A practical reconfigurable hardware accelerator for boolean satisfiability solvers.
In *DAC*, pages 780–785, 2008.

Sharad Malik and Lintao Zhang.
Boolean satisfiability from theoretical hardness to practical success.
Commun. ACM, 52(8):76–82, 2009.