ParallelClosure: A Parallel Design Optimizer for Timing Closure

Yi-Shan Lu\textsuperscript{1}, Wenmian Hua\textsuperscript{2}, Rajit Manohar\textsuperscript{2}, Keshav Pingali\textsuperscript{1}

\textsuperscript{1}University of Texas at Austin, \textsuperscript{2}Yale University

March 22\textsuperscript{nd}, 2019 at TAU 2019 Workshop
ParallelClosure

• Our design optimizer for TAU 2019 contest

• Design optimizations considered
  • Buffer insertion for fixing hold time violations [1]
  • Gate sizing by slew targeting [2] for minimizing area, leakage power & clock period
  • All algorithms are generalized for multi-corner, multi-mode (MCMM) optimizations

• Parallelization of static timing analysis (STA) & gate sizing
  • Parallelism analyses using the operator formulation [3]
  • Parallel implementation using the shared-memory Galois framework [4]

2. S. Held. “Gate sizing for large cell-based designs,” in DATE’09.
Outline

• Optimization flow – the algorithms
• Parallelization – boosting tool runtime
• Limitation
• Conclusions
Parallel Closure

Buffer insertion for removing max. cap. violations

Buffer insertion for removing hold time violations \[1\]

Gate sizing by slew targeting \[2\]

optimized .v
optimized .spef
ECOs

2. S. Held. “Gate sizing for large cell-based designs,” in DATE’09.
• We generalize the approach in the following paper to MCMM:

• Gate sizing in multi-mode optimization

<table>
<thead>
<tr>
<th>Gate position</th>
<th>Setup time</th>
<th>Hold time</th>
</tr>
</thead>
<tbody>
<tr>
<td>On critical paths</td>
<td>Upsize</td>
<td>Downsize</td>
</tr>
<tr>
<td>Not on critical paths</td>
<td>Downsize</td>
<td>Upsize</td>
</tr>
</tbody>
</table>

• Each gate output has a slew target per combination of (corner, mode)
• Use slew targets (slewt) to guide the sizing process

<table>
<thead>
<tr>
<th>Sizing operation</th>
<th>Slew target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upsize</td>
<td>Decrease</td>
</tr>
<tr>
<td>Downsize</td>
<td>Increase</td>
</tr>
</tbody>
</table>

2. S. Held. “Gate sizing for large cell-based designs,” in DATE’09.
2. S. Held. “Gate sizing for large cell-based designs,” in DATE’09.
- Initialize slew targets as slews from STA
- Update slew targets
  - Globally critical: slack(p) < 0
  - Locally critical: whether p is on a critical path
  - Adjust the slew targets for p based on modes & p’s criticality

<table>
<thead>
<tr>
<th>Gate position</th>
<th>Setup time slewt</th>
<th>Hold time slewt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Globally &amp; locally critical</td>
<td>Decrease</td>
<td>Increase</td>
</tr>
<tr>
<td>Otherwise</td>
<td>Increase</td>
<td>Decrease</td>
</tr>
</tbody>
</table>

2. S. Held. “Gate sizing for large cell-based designs,” in DATE’09.
What values to update slew targets?

- **Slew possibilities**
  - Values by table lookup into the slew table w/ current slew & different cap.
  - Upper bound \((ub)\): cap. = max cap. of the pin
  - Lower bound \((lb)\): cap. = 0
  - Values considered: \(lb*(ub/lb)^{(n/k)}\)
    - In ParallelClosure, \(k = 20\); \(n = 0, 1, 3, 5, 8, 11, 15, 20\)

- **Update slew targets of pin \(p\) based on**
  - Setup/hold time mode
  - \(p\)’s criticality & previous slew targets

- **No max slew violation by construction**

---

### Output rising slew for BUF_X1, Nangate 45 nm, typical corner

<table>
<thead>
<tr>
<th>(TC)</th>
<th>0.365616</th>
<th>1.895430</th>
<th>3.790860</th>
<th>7.581710</th>
<th>15.163400</th>
<th>30.326900</th>
<th>60.653700</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.23599</td>
<td>3.33809</td>
<td>5.59725</td>
<td>8.60523</td>
<td>14.8575</td>
<td>27.5164</td>
<td>52.8765</td>
<td>103.604</td>
</tr>
<tr>
<td>4.43724</td>
<td>3.33727</td>
<td>5.59699</td>
<td>8.60578</td>
<td>14.8576</td>
<td>27.5188</td>
<td>52.8775</td>
<td>103.599</td>
</tr>
<tr>
<td>15.6743</td>
<td>3.40246</td>
<td>5.62543</td>
<td>8.61689</td>
<td>14.8582</td>
<td>27.5170</td>
<td>52.8787</td>
<td>103.599</td>
</tr>
<tr>
<td>37.1331</td>
<td>4.36023</td>
<td>6.10464</td>
<td>8.84317</td>
<td>14.9465</td>
<td>27.5247</td>
<td>52.8726</td>
<td>103.605</td>
</tr>
<tr>
<td>70.5649</td>
<td>5.85455</td>
<td>7.27833</td>
<td>9.43026</td>
<td>15.0988</td>
<td>27.6409</td>
<td>52.9322</td>
<td>103.603</td>
</tr>
<tr>
<td>117.474</td>
<td>7.61897</td>
<td>9.14083</td>
<td>10.8314</td>
<td>15.5462</td>
<td>27.6912</td>
<td>53.0238</td>
<td>103.669</td>
</tr>
<tr>
<td>179.199</td>
<td>9.58764</td>
<td>11.3565</td>
<td>13.0249</td>
<td>16.7347</td>
<td>27.8716</td>
<td>53.0513</td>
<td>103.775</td>
</tr>
</tbody>
</table>

---

### Output rising slew for BUF_X2, Nangate 45 nm, typical corner

<table>
<thead>
<tr>
<th>(TC)</th>
<th>0.365616</th>
<th>3.786090</th>
<th>7.572190</th>
<th>15.144400</th>
<th>30.288800</th>
<th>60.577500</th>
<th>121.155000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.23599</td>
<td>3.10917</td>
<td>5.67693</td>
<td>8.71288</td>
<td>14.9785</td>
<td>27.6350</td>
<td>52.9690</td>
<td>103.657</td>
</tr>
<tr>
<td>4.43724</td>
<td>3.10875</td>
<td>5.67786</td>
<td>8.71402</td>
<td>14.9788</td>
<td>27.6339</td>
<td>52.9719</td>
<td>103.660</td>
</tr>
<tr>
<td>15.6743</td>
<td>3.20354</td>
<td>5.70984</td>
<td>8.72471</td>
<td>14.9811</td>
<td>27.6310</td>
<td>52.9744</td>
<td>103.651</td>
</tr>
<tr>
<td>37.1331</td>
<td>4.20264</td>
<td>6.15463</td>
<td>8.94062</td>
<td>15.0761</td>
<td>27.6468</td>
<td>52.9670</td>
<td>103.666</td>
</tr>
<tr>
<td>70.5649</td>
<td>5.70174</td>
<td>7.27713</td>
<td>9.47332</td>
<td>15.2076</td>
<td>27.7634</td>
<td>53.0379</td>
<td>103.659</td>
</tr>
<tr>
<td>117.474</td>
<td>7.47026</td>
<td>9.13720</td>
<td>10.8172</td>
<td>15.6132</td>
<td>27.8134</td>
<td>53.1232</td>
<td>103.735</td>
</tr>
<tr>
<td>179.199</td>
<td>9.44195</td>
<td>11.3787</td>
<td>12.9969</td>
<td>16.7387</td>
<td>27.9813</td>
<td>53.1620</td>
<td>103.831</td>
</tr>
</tbody>
</table>
• **Order of sizing**
  • Want to fix fanout gates of $g$ before sizing $g$
    • Output load matters more than input slew
  • Reverse topological order for gates
    • Cut cycles of gates at edges to register data inputs

• **Slew estimation**: see [2] for details

2. S. Held. “Gate sizing for large cell-based designs,” in *DATE*’09.
How to select cells for gates?

<table>
<thead>
<tr>
<th>Mode</th>
<th>For a given corner $cnr$</th>
<th>Across corners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup time</td>
<td>The smallest size that satisfies all slew targets</td>
<td>$size_s(g) = \max_{cnr}{size_{s,cnr}(g)}$</td>
</tr>
<tr>
<td>Hold time</td>
<td>The largest size that satisfies all slew targets</td>
<td>$size_h(g) = \min_{cnr}{size_{h,cnr}(g)}$</td>
</tr>
</tbody>
</table>

- If $size_s(g) \leq size_h(g)$, assign $g$ to the cell of size $size_s(g)$
  - Reduce area & leakage power
- If $size_s(g) > size_h(g)$, assign $g$ to the cell of size $size_h(g)$
  - Honor hold time constraints while limiting the impact to setup time
• The new cell assignment (state) is better if
  • The worst negative slack improves for all corners and modes; or
  • The area is reduced w/o the following metrics significantly worsened in any corner and mode:
    • Worst negative slack
    • Average total negative slack over all path endpoints, e.g., register data inputs

2. S. Held. “Gate sizing for large cell-based designs,” in DATE’09.
Outline

• Optimization flow – the algorithms
• Parallelization – boosting tool runtime
• Limitation
• Conclusions
Parallelization w/ operator formulation \cite{3}

- **Active elements**
  - Nodes/edges/subgraphs where computation is needed

- **Operator**
  - Computation at active elements
  - Neighborhood: set of nodes/edges read/written by the update
  - Morph operators may change graph topology
  - Label-computation operators only update node/edge labels

- **Schedules**
  - The ordering to apply operators on active elements
  - May be constrained for correctness
  - Some ordering may perform better than the others

- **Parallelism**
  - Disjoint updates
  - Read-only operators


Features of Galois

• Parallel data structures
  • Graphs, bags, etc.

• Parallel loops over active elements
  • for_each, do_all, etc.

• Support for
  • Load balancing
  • Scheduling
  • Dynamic work
  • Transactional execution

Successes in EDA

• FPGA routing
  [Moctar & Brisk, DAC 2014]

• AIG rewriting
  [Possani et al., ICCAD 2018]

• Timing closure
  [Lu et al., TAU 2019 contest]

#include "TimingGraph.h"

using GNode = TimingGraph::GraphNode;
using GNodeBag = galois::InsertBag<GNode>;

void propagateForward(TimingGraph& g) {
    GNodeBag fFront;
    initForward(g, fFront);
    computeForward(g, fFront);
}

int main(int argc, char** argv) {
    galois::SharedMemSys G;
    // instantiate a timing graph
    TimingGraph g;
    // construct g using cell libraries
    // & Verilog netlist
    // initialize g using SDC commands
    propagateForward(g);
    propagateBackward(g);
    reportCriticalPath(g);
    return 0;
}
STA Speedup over OpenTimer 2.0 for Best-time Runs

<table>
<thead>
<tr>
<th>Circuit</th>
<th># Gates</th>
<th># Nets</th>
<th># Pins</th>
<th>Sequential Runtime (OT, $G_{lv}$, $G_{dag}$)</th>
<th>Best Runtime (OT, $G_{lv}$, $G_{dag}$) (and # Threads Used)</th>
<th>Speedup over OT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac97_ctrl</td>
<td>14,131</td>
<td>14,407</td>
<td>40,238</td>
<td>390.0, 114.0, 101.3</td>
<td>312.0 (21), 62.3 (7), 35.3 (7)</td>
<td>5.01, 8.83</td>
</tr>
<tr>
<td>aes_core</td>
<td>22,938</td>
<td>23,199</td>
<td>66,221</td>
<td>623.3, 226.3, 196.7</td>
<td>493.3 (7), 90.3 (7), 53.0 (7)</td>
<td>5.46, 9.31</td>
</tr>
<tr>
<td>des_perf</td>
<td>105,371</td>
<td>106,532</td>
<td>295,808</td>
<td>3,453.0, 1,173.3, 956.3</td>
<td>2,762.7 (14), 266.7 (14), 155.0 (14)</td>
<td>10.36, 17.82</td>
</tr>
<tr>
<td>vga_lcd</td>
<td>139,529</td>
<td>139,631</td>
<td>380,730</td>
<td>4,700.3, 1,495.7, 1,232.3</td>
<td>3,660.7 (28), 319.3 (14), 187.7 (14)</td>
<td>11.46, 19.51</td>
</tr>
<tr>
<td>des_perf*10</td>
<td>1,053,710</td>
<td>1,065,311</td>
<td>2,958,071</td>
<td>34,853.0, 12,765.3, 10,441.7</td>
<td>29,923.3 (14), 2,222.0 (14), 1,366.7 (14)</td>
<td>13.47, 21.90</td>
</tr>
<tr>
<td>vga_lcd*10</td>
<td>1,395,290</td>
<td>1,396,301</td>
<td>3,807,291</td>
<td>49,284.3, 16,063.7, 13,084.0</td>
<td>31,212.7 (35), 2,768.7 (14), 1,708.7 (14)</td>
<td>11.27, 18.27</td>
</tr>
</tbody>
</table>
Outline

• Optimization flow – the algorithms
• Parallelization – boosting tool runtime
• Limitation
• Conclusions
Limitation

Quality of results

• Lots of buffers are inserted when there is a large number of paths w/ hold-time violations
  • Clock network synthesis [5] may help

• Not considering net topology and optimal buffer insertion for a net
  • Topology: C-tree algorithm [6]
  • Optimal buffer insertion: van Ginneken’s algorithm [7]

• Need more parameter tuning
  • E.g., convergence criteria of sizing

Performance of ParallelClosure

• Buffer insertion is purely sequential
  • Consistency of name-object mappings
  • The algorithm for fixing hold-time violations has no parallelism

Outline

- Optimization flow – the algorithms
- Parallelization – boosting tool runtime
- Limitation
  - Conclusions
Conclusions

• ParallelClosure is effective for designs w/ a small # hold-time violations
  • Buffer insertion for fixing hold time violations \[1\]
  • Gate sizing by slew targeting \[2\] for minimizing area, leakage power & clock period
  • All algorithms are generalized for multi-corner, multi-mode (MCMM) optimizations

• ParallelClosure is efficient through parallelizing STA & gate sizing
  • Parallelism analyses using the operator formulation \[3\]
  • Parallel implementation using the shared-memory Galois framework \[4\]

2. S. Held. “Gate sizing for large cell-based designs,” in DATE’09.
Thanks!

Questions? Comments?