Parallel CAD for FPGAs: A Personal Retrospective and Thoughts for the Future

Vaughn Betz
vaughn@eecg.utoronto.ca
Agenda

• Altera: 2006 - 2011
  • Motivation & constraints for parallel CAD
  • “High-Quality, Deterministic Parallel Placement for FPGAs on Commodity Hardware”, FPGA 2008
  • First commercial parallel placement algorithm for FPGAs
  • How does it work?
  • How well does it work?
• Later Parallel Placement Enhancements
• Compile Time Past, Present & Future
Part I: Motivation & Constraints
Altera 2006

- My team: responsible for most of Quartus II compile
- Major progress in speeding up
- Other SW directors: Good enough?

![Quartus II Compile Time Graph](image)

**Quartus II Compile Time**
(Relative time for fixed design on fixed CPU)

<table>
<thead>
<tr>
<th>Quartus II Version</th>
<th>Dec-02</th>
<th>Dec-03</th>
<th>Dec-04</th>
<th>Dec-05</th>
<th>Dec-06</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Compile Time (Log scale)</td>
<td>100%</td>
<td>80%</td>
<td>63%</td>
<td>50%</td>
<td>40%</td>
</tr>
</tbody>
</table>

- Stratix
- Stratix II
The Challenge

Have to Sell Your Ideas!
How to Speed Up

1. Algorithm improvements
   – Productive, but dangerous to rely on solely

2. Incremental compile
   – Software-like flow: only recompile what changed
   – Useful, but requires up-front planning and can hurt productivity

3. Go parallel

Did all 3 → Parallel Code Must Be Able To Evolve
Aside: Incremental Compilation

- Define partitions
  - CAD will not optimize across partitions
  - Can re-synthesize, place and route one partition at a time
  - Faster compile time
  - Fewer iterations because other logic unchanged

- "RapidRecompile"
  - Incremental compile without the designer identifying partitions
  - Figures out what changed automatically
  - Challenge: global optimizations
Parallel Constraints

- **Deterministic** *(required)*
  - Same results every run
  - Means no race conditions
  - Most prior work wasn’t deterministic
  - Almost impossible to test non-deterministic code
  - Many customers will not use it
    - Can’t reproduce results
    - Inherently insecure

- **Serial equivalency** *(desirable)*
  - Even better: same result no matter how many cores
Parallel Constraints

- Quality *(required)*
  - Need to achieve quality comparable to current Quartus results
    - Worse timing closure: designer productivity hurt
    - IP timing closure: DDR, PCIe *must* close timing
    - Increased wiring $\rightarrow$ unroutes?
    - Power optimization?
  - Rough annealing trade-off: 10% quality means 10X less runtime
    - Small quality loss not worth it for a moderate compile time gain
Parallel Constraints

- Maintainable *(required)*
  - Quartus is big (~20 million lines of code)
  - Place and route system over 1 million lines
  - Don’t want to shut down new algorithm work
  - Plus new devices, features need to be integrated
  - Careful how you code, and only make key pieces parallel
Algorithm Runtimes (Quartus 2011)

Must Parallelize Multiple Algorithms, but Placement is Biggest
Part II: High Quality, Deterministic Parallel Placement for FPGAs on Commodity Hardware

Overview of FPGA 2008 and TODAES 2011 Papers by Adrian Ludwin, Ketan Padalia and Vaughn Betz
Quartus Placement (2011)

- Simulated annealing-based
- Optimizes wire, timing, power, congestion
- Based on academic VPR, but with many enhancements
  - More complex cost functions
  - Directed moves
  - Multi-level placement
- Spends ~50% of time at T = 0 (quench)
Algorithm Overview

P = InitialPlacement ();
T = InitialTemperature ();

while (ExitCriterion () == False) {

    while (InnerLoopCriterion () == False) { /* One temperature */

        Pnew = PerturbPlacementViaMove (P); /* Propose move */
        ΔCost = Cost (Pnew) – Cost (P); /* Evaluate move */

        r = random (0,1);
        if (r < e^{-ΔCost/T}) {
            P = Pnew; /* Accept (Finalize) move */
        }
    } /* End one temperature */

    TimingAnalyze();
    CongestionAnalyze();
    T = UpdateTemp (T);
}

10K LoC in Propose + Evaluate

Dominate

CPU Time
Placement Improvement via Moves

<table>
<thead>
<tr>
<th></th>
<th>LAB 1</th>
<th></th>
<th></th>
<th>LAB 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAB 3</td>
<td></td>
<td>LAB 4</td>
<td>Move 2</td>
<td>LAB 6</td>
</tr>
<tr>
<td>LAB 8</td>
<td>Move 1</td>
<td>LAB 9</td>
<td></td>
<td>LAB 10</td>
</tr>
</tbody>
</table>

Move 1

Move 2
Move Evaluation & Acceptance

- Quality change of move estimated with complex cost function
  - Fast estimates of wiring, timing, power
  - Blended together into overall cost

- If cost decreases, move always accepted
  - Placement state is updated

- If costs increase, still have some chance of accepting if $T > 0$
  - Hill climbing
  - But not in the quench ($T = 0$)
move = propose(place);
cost = evaluate(place, move);
if(cost < 0) {
  accept(place, move);
}

- **Processing** (propose and evaluate)
- **Finalization** (resolve collisions and commit)

99% time
1% time
Core 0 Propose & Evaluate View

<table>
<thead>
<tr>
<th>LAB 1</th>
<th></th>
<th></th>
<th>LAB 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAB 3</td>
<td>LAB 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LAB 5</td>
<td>LAB 6</td>
<td></td>
</tr>
<tr>
<td>LAB 8</td>
<td></td>
<td></td>
<td>LAB 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LAB 10</td>
</tr>
</tbody>
</table>

Move 1
Core 1 Propose & Evaluate View

LAB 1

LAB 2

LAB 3

LAB 4

Move 2

LAB 5

LAB 6

LAB 7

LAB 8

LAB 9

LAB 10
Core 1 Finalize View
Resolving Collisions

Must detect collisions (avoid illegal placement)
When two moves have collided, we can:
   – Abandon the later moves (non-deterministic)
   – Or attempt to “fix” colliding moves

We fix it by *reproposing* it
   – This gives the same move as in the serial flow

Therefore, the placer is not only *deterministic*
   but also *serially equivalent*
   – Easier to test → results same no matter how many cores
Process (C0)  
Process (C1)  
Process (C2)  
Process (C3)  

Processing  
(propose and evaluate)  

Finalization  
(resolve collisions and commit)  

Finalize
Speedup higher in quench
  - Fewer accepted moves
  → Fewer collisions & reproposals

Cost of determinism?
  ● Modest: estimate ~12%

Memory subsystem a bigger limit
  - Parallel CAD needs memory-friendly code
Part III: Later Parallel Placement
Conflict Free Moves

- Avoid conflicts by modifying move generators & cost functions
- Each core moves blocks in a different region
- Cost function uses stale information for blocks outside region
  → No need to track conflicts or repropose moves
- Improves speedup (51x), reduces quality (10%)
- Deterministic
Dependency Checker & Fewer Conflicts

- Coding a dependency checker to **repropose** conflicting moves is hard
  - Can hardware or software (compiler) transactional memory do it for us?
  - Unfortunately, no (poor performance)

- Tweaking cost functions (ignore high fanout nets) and move generators can reduce conflicts with no quality loss
  - 5X speedup at equal quality, deterministic
  - Larger gains if you sacrifice determinism
Algorithm Changes + Parallelism

- Combine analytic placement & quenching
  - Gort and Anderson, “Analytic Placement for Heterogeneous FPGAs,” FPL 2012

- Analytic placement to get global placement
  - Parallelize x & y matrix solutions
  - 2X speedup

- Quench (iterative refinement) to fine-tune
  - Uses parallel moves
  - Avoids high-temperature part of anneal (most conflicts)
  - 1.48X speedup

- Overall parallel speedup 1.3X
Part IV: Compile Time Past, Present & Future
The Past: Parallel Success

Quartus II Compilation Time History
(Relative time for a fixed design, on a fixed CPU)

- Stratix
- Stratix II
- Stratix III (Parallel)
- Stratix IV (Parallel)

Clustering, placement, routing & delay estimation all parallel by 2009
But Compile Time Challenge Has Grown
The Present

- Large, Stratix 10 2800 High Performance Design

Compile Time (h)

<table>
<thead>
<tr>
<th>Cores</th>
<th>Synthesis</th>
<th>Plan</th>
<th>Place</th>
<th>Route</th>
<th>Retime</th>
<th>Finalize</th>
<th>Timing Closure Recom.</th>
<th>Timing, Power, Asm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1.5x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.3X Speedup</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.3x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.75X Speedup</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3.75x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overall: 2.3X Speedup

Place: 3.75X Speedup
The Present

- Parallel compile very helpful
  - 42 hours → 18 hours
- Placement longest single algorithm
  - 14 hours → 3.75 hours
- But many important algorithms
  - Extreme Amdahl’s law → must speed them all
- Synthesis not parallel
  - Lowest hanging fruit to attack (7 hours here)
  - Little published parallel synthesis research
Observations

- Algorithm speedups in 2x – 4x range on 8 cores
- Large memory footprints, complex algorithms, data transfer between algorithms
  → GPUs unlikely to help
- Machines with moderate number of fast cores best fit to current CAD tools
What to Do?

- Algorithm & parallelism co-optimization
  - Find algorithm with best parallel time
- Partition designs and compile incrementally
  - Shell & role in datacenters
  - But still not employed inside core of most designs
- FPGA architecture to reduce compile time
  - Larger logic blocks
  - More routing? (but increases cost)
  - Harden more
    - E.g. Network-on-Chip
NoC: Pre-Wired & Timing Closed

Traditional: CAD tool builds system-level interconnect

FPGA with Hard NoC: System-Level Interconnect Pre-Built
Wrap Up
Wrap Up

- FPGA capacity greatly outstripping serial CPU speed growth
- Parallelize **high-quality** algorithms
  - E.g. VPR 8 router 6x to 300x faster than earlier algorithms
- Complex flow → many algorithms to speed up
- Integrate / open-source
  - Many parallel algorithms tested in VPR
  - But only timing analysis integrated in current master
- Flat compile productive, but may not scale
  - Partition / incremental compile flows
  - But increases planning for designers
  - Do we need automatic floorplanners?
- FPGA architecture for compile time
  - Has not been a major architecture goal
  - Should be in the future