HeAP: Heterogeneous Analytical Placement for FPGAs

FPL 2012
Marcel Gort and Jason Anderson
Motivation

• CAD for FPGAs takes too long (up to a day).
• FPGA placement contributes to a large proportion of overall CAD time.
• In ASIC domain, where millions of cells are handled by placement, fast analytical methods dominate.
• Recent work [1] adapted FastPlace to homogeneous FPGAs.
  • vs. VPR: ~13x speedup with 20% worse wirelength.
• Can we use analytical methods to place cells onto a realistic heterogeneous FPGA?

Analytical Placement (AP)

- Objective function: Half-Perimeter Wirelength (HPWL)
- Minimizing objective function:
  - Solve system of linear equations generated from connections between cells.
  1. Convert multi-pin nets to 2-pin nets.
  2. Create system of linear equations to solve \textit{weighted sum of squared distances between cells}.
  3. Solve system using off-the-shelf linear systems solver.
\[
\Phi = \text{Min} \left[ (X_B - X_A)^2 + (X_C - X_B)^2 + (X_A - 0)^2 + (3 - X_C)^2 \right]
\]
\[
\begin{bmatrix}
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 2
\end{bmatrix}
\begin{bmatrix}
X_A \\
X_B \\
X_C
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
3
\end{bmatrix}
\]
\[
\begin{bmatrix}
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 2
\end{bmatrix}
\begin{bmatrix}
X_A \\
X_B \\
X_C
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
3
\end{bmatrix}
\]

\[X_A = 0.75, \quad X_B = 1.5, \quad X_C = 2.25\]
\[
\begin{bmatrix}
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 2
\end{bmatrix}
\begin{bmatrix}
X_A \\
X_B \\
X_C
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
3
\end{bmatrix}
\]

\[X_A = 0.75, \; X_B = 1.5, \; X_C = 2.25\]
Actual Solved Solution

Need to spread cells!
AP Overview

Generate X and Y matrices based on netlist

Solve

Spread cells to legalize solution

Good enough?

Yes

Refinement

No

Perturb formulation based on spreading
Legalized Solution
Spreading

- Adapted from SimPL [2].
- Find over-utilized area.

2. Myung-Chul K., Dong-Jin L., Markov I. SimPL: An Effective Placement Algorithm. in ICCAD 2010
Spreading

- Adapted from SimPL [2].
- Find over-utilized area.
- Find a larger surrounding area that can accommodate all cells within it.
Spreading

• Adapted from SimPL [2].
• Find over-utilized area.
• Find a larger surrounding area that can accommodate all cells within it.
• Split the cells into two sets.
Spreading

- Assign an area to each cell which is proportional to the total area of the cells.
Spreading

• Assign an area to each cell which is proportional to the total area of the cells.
• Spread each set of cells separately, within the area assigned to it.
Spreading

• Assign an area to each cell which is proportional to the total area of the cells.
• Spread each set of cells separately, within the area assigned to it.
• Alternate x and y spreading directions until solution is legal.
Spreading

- Assign an area to each cell which is proportional to the total area of the cells.
- Spread each set of cells separately, within the area assigned to it.
- Alternate x and y spreading directions until solution is legal.
Spreading

- Assign an area to each cell which is proportional to the total area of the cells.
- Spread each set of cells separately, within the area assigned to it.
- Alternate x and y spreading directions until solution is legal.
AP Overview

Generate X and Y matrices based on netlist

Solve

Spread cells to legalize solution

Good enough?

Perturb formulation based on spreading

Refinement

Yes

No
Pseudo-connections

Solved solution

Legalized solution
Pseudo-connections

Solved solution

Legalized solution
Pseudo-connections

Weighted pseudo-connections between cells and their legalized placements.
AP Overview

Re-generate X and Y matrices based on perturbed formulation

Solve

Spread cells to legalize solution

Good enough?

Yes

Refinement

No

Perturb formulation based on spreading
HeAP

• Analytical placement framework that targets commercial FPGAs.
  – Currently supports Cyclone II FPGAs.
  – Supports RAMs, DSPs, LABs, hard macros (carry chains).
• Uses Quartus University Interface Program (QUIP) to replace the placer in Quartus II.
AP for FPGAs
AP for FPGAs
AP for FPGAs

FPGA

ASIC
• Snap to Grid – cut generation in spreading is multi-objective.
AP for FPGAs

- Snap to Grid – cut generation in spreading is multi-objective.
- Carry chains – relative cell placements must be maintained.
AP for FPGAs

- Snap to Grid – cut generation in spreading is multi-objective.
- Carry chains – relative cell placements must be maintained.
- Cell type can only fit in correct slot type (e.g. RAMs in RAM slots).
AP for FPGAs – Spreading

- Spread each cell type separately.
- Maintain legality.
AP for FPGAs – Spreading

- Spread each cell type separately.
- Maintain legality.
AP for FPGAs – Spreading

- Spread each cell type separately.
- Maintain legality.
AP for FPGAs – Spreading

- Spread each cell type separately.
- Maintain legality.
AP for FPGAs – Spreading

- Spread each cell type separately.
- Maintain legality.
AP for FPGAs – Spreading

- Spread each cell type separately.
- Maintain legality.
AP for FPGAs – Spreading

- Spread each cell type separately.
- Handle legality in spreading.
AP for FPGAs – Solving

- Solver doesn’t know about resource constraints but…
  - We can encourage solver to place a cell close to a legal slot by adding a pseudo-connection between that cell and the slot.
  - After enough iterations, solver will begin to generate solutions that are close to legal.
AP for FPGAs – Solving

- Solver doesn’t know about resource constraints but...
  - We can encourage solver to place a cell close to a legal slot by adding a pseudo-connection between that cell and the slot.
  - After enough iterations, solver will begin to generate solutions that are close to legal.
AP for FPGAs – Solving

- Solver doesn’t know about resource constraints but...
  - We can encourage solver to place a cell close to a legal slot by adding a pseudo-connection between that cell and the slot.
  - After enough iterations, solver will begin to generate solutions that are close to legal.
• Solver doesn’t know about resource constraints but…
  • We can encourage solver to place a cell close to a legal slot by adding a pseudo-connection between that cell and the slot.
  • After enough iterations, solver will begin to generate solutions that are close to legal.
• Solve different types of cells together or separately?
Solving Orders

All

Initial Placement

Solve all

Spread DSPs

Spread RAMs

Spread LABs

Solve all

Spread DSPs

Spread RAMs

Spread LABs
## Solving Orders

<table>
<thead>
<tr>
<th>All</th>
<th>Rotate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial Placement</strong></td>
<td><strong>Initial Placement</strong></td>
</tr>
<tr>
<td>Solve all</td>
<td>Solve DSPs</td>
</tr>
<tr>
<td>Spread DSPs</td>
<td>Spread DSPs</td>
</tr>
<tr>
<td>Spread RAMs</td>
<td>Solve RAMs</td>
</tr>
<tr>
<td>Spread LABs</td>
<td>Spread RAMs</td>
</tr>
<tr>
<td>Solve all</td>
<td>Solve LABs</td>
</tr>
<tr>
<td>Spread DSPs</td>
<td>Spread LABs</td>
</tr>
<tr>
<td>Spread RAMs</td>
<td>Solve DSPs</td>
</tr>
<tr>
<td>Spread LABs</td>
<td>Spread DSPs</td>
</tr>
</tbody>
</table>
Solving Orders

All + Rotate

Initial Placement

- Solve all
- Spread DSPs
- Spread RAMs
- Spread LABs

- Solve DSPs
- Spread DSPs
- Solve RAMs
- Spread RAMs
- Solve LABs
- Spread LABs
- Solve all

...
Solving Order Convergence Rate

Legalized

Solved
Experimental Methodology

1. Used Altera Quartus II to generate:
   - I/O placement
   - Cell packing
2. Run HeAP, targeting smallest of three Cyclone II FPGAs on which benchmark will fit.
3. Quartus II verifies legality and generates post-routing results.
Experimental Methodology

• vs. Quartus II:
  – All 4 cores of Intel core i5 using various Quartus effort levels.

• HeAP parallelization
  – During solving, x and y directions solved simultaneously.
  – During refinement, “move suggestion” phase parallelized.

• vs. Simulated Annealing (SA):
  – Code from non-timing driven VPR placement (“fast”).

• 10 largest benchmarks from CHStone, QUIP, and MCNC.
Placement Run-time

Placement Run-time (s)

- Quartus II
- HeAP
- SA

10x speedup

4x speedup
HPWL Comparison

- Quartus II
- HeAP
- SA

7% reduction
Wirelength vs. Run-time Landscape

- Varied Quartus Placement effort from 0.1, 0.5, 1.0, 2.0.
- Varied fitter effort level between auto and standard.
- Ran in timing and non-timing driven modes.
Wirelength vs. P&R Run-time Landscape
Wirelength vs. P&R Run-time Landscape
Wirelength vs. P&R Run-time Landscape
FMax vs. P&R Run-time Landscape

Effort = 0.1
Effort = 0.5

Fmax (Mhz)

Run-time (s)
FMax vs. P&R Run-time Landscape

Effort = 0.1
Effort = 0.5

- Quartus II - TD
- Quartus - NTD
- HeAP - TD
- HeAP - NTD
Conclusions

• AP can work well for Heterogeneous FPGAs.
  • Add pseudo connections to legal cell locations.
  • To obtain high quality and high speed, mix solving for each cell type separately and solving for cell types together.
• HeAP offers better run-time and wirelength than some Quartus II low effort placement runs.
• HeAP is unable to equal high-effort, high-quality results obtained using Quartus II.
• Compared to Quartus II run with default effort levels:
  – 4x faster than non-timing driven flow with 5% worse wirelength.
  – 11x faster than timing-driven flow with 4% worse wirelength and 9% worse FMax.
Future Work

• Timing-driven AP
  • 9% worse FMax when ignoring timing in placement. Can we improve on that?
• Quality-driven spreading
  • Spreading objective function does not consider wirelength.
Questions?