Quantifying Academic Placer Performance on Custom Designs

Datapath Placement Benchmarks

Samuel Ward and Earl Swartzlander
The University of Texas at Austin

David A. Papa, Zhuo Li, Cliff Sze, and Charles Alpert
IBM Austin Research

© 2011 IBM Corporation
Outline

- Background
- Motivation
- What is Dataflow or Structured Placement Design?
- Dataflow Circuit Design Examples
- Structured Placement Benchmark A
- Structured Placement Benchmark B
- Errata to the Paper
- Results
Modern Processor Design – Power7

- **Chip**
  - 45nm SOI process, 567 mm²
  - 1.2 billion Transistors
  - 3.0 – 4.25 GHz clock speed

- **Core**
  - 4, 6 or 8 cores per chip
  - 4 SMT threads per core
  - 10+ execution units per core:

- **Unit**
  - Collection of Fine-Grained Macros
  - Random Logic Macros (RLMs)
  - Dataflow / Structured Macros
  - RLMs automated
  - Dataflow still mostly done manually
Motivation

- **Bridging the Gap:**

  - **Driving Higher Productivity**
    - 10x difference in # transistors per designer for RLM vs Dataflow
    - Need improved automation tools because of growing costs
  
  - **Reason: Sub Optimality on Dataflow Macros**
    - Area: generate less dense designs
    - Delay: present 10%-20% reduction post routed delays
    - Power: 20%-30% increase in power for similar delays
Goals

- Often claimed that current placement tools do not perform well on datapath designs

- **BUT WHY?? AND HOW MUCH??** Is it the...
  - Regular structure?
  - Highly compact layouts?

- **GOAL:** Compare State-of-the-art placement tools against manual placement on REAL designs
  - Is placement the problem or is it something else?

- **This work:**
  - Presents two custom constructed datapath designs that perform common logic functions
  - Presents hand-designed layouts for each to compare the “known optimal solution”
  - Compares latest generation of academic placers against them
  - Testcases released publically
What’s Different?

- There have been past attempts to quantify suboptimality of placement heuristics
  - Hagen, et al. [7]
    - Copy small circuits and replicate them
    - Loosely connecting their ports together, in order to create a much larger benchmark.
    - Problems
      - Defined connections between the copies do not correspond to real logic functions.
      - No pin locations are defined for the circuit
  - PEKO/PEKU Benchmarks - Chang, et al.
    - Placement examples with known optima (PEKO)
    - Placement examples with known upperbounds (PEKU)
    - Optimality achieved by adding nets to cells in configurations that cannot be shortened.
    - though the pin distributions of cells matched that of a typical VLSI circuit.
    - Problems
      - these netlists did not correspond to any logic function at all.
      - It could be argued that the PEKO and PEKU testcases are artificially hard

- What is Needed:
  - Need real logic function
  - Need to now how close placers are to optimal
What is Dataflow Design?

- **Logic Function Types:**
  - Load / Store queue
  - Decoders
  - Encoders
  - Crossbar Switch
  - Adders
  - Muxes
  - Latch Banks for Buses
  - Memories / CAMs

- Many designs have regular datapaths, placers have no regular structure
- Placement failures: high utilization, irregular shapes
- Gate Sizing: Larger wire lengths cause increased gate sizes
- Routing: Difficult to route even they are placed
- Let’s look at some examples...
Dataflow Example 1: Fixed Latches

- Select Lines Share Routing Tracks
- Bitstack Compressed to Save Area / Routing
- >30% Area Growth
- Input / Output Pins Lined up to Reduce Integration Level Congestion

**Custom Solution:**

**Placed Solution:**
Dataflow Example 2: Unfixed Latches

- **Impact**
  - To achieve similar wire length:
  - Area: >40%
  - Timing: >20%

- **Why Fixed Latches?**
  - Provide “hints” to the placer for improved results
  - Improve clock routing
  - Overall timing is better
  - Unfixed more Unstable -> multiple tool flows
Dataflow Examples: Design 3

- **Custom Placement**
  - Util = 95%
  - Meets Timing
  - Design Time: 14-18 weeks
  - Highly Stable

- **Automated Placement**
  - Util < 70%
  - Routing Congestion Problems
  - Larger WL drives more power, larger gate sizes
  - ~40% Area Growth
  - Design Time: 6-8 weeks
Dataflow Examples: Design 4

- **Custom Placement**
  - Util > 95%, Meets Timing
  - Compact Placement, careful whitespace usage
  - Design Time: 8-12 Weeks

- **Automated Placement**
  - Util < 70%, Timing Critical
  - Larger WL drives more power, larger GW
  - ~30% Area Growth
  - Design Time: 2-4 Weeks
Why are Placers Bad?

- Generally speculated that poor performance of placers on datapath designs is due to very tight density constraints.
- Perhaps placers could find the right structures but simply had trouble with the legalization?

Experiment:
- Two dataflow designs built
- Eight variants of each created
  - Additional whitespace inserted
  - Provides more opportunity for the placers

\[ \eta = \alpha + \beta + n\varepsilon \]
How Were They Built?

- Generally, Custom Design uses a Different Library
- Implemented Common Dataflow Structures
  - Custom Design Environment
  - Used Standard Cell Library
- Manually Placed Custom
- Converted Layout Netlist to Bookshelf Format
- Compared Wire Length between Manual and Placed Solutions
- Let’s take a look at the designs...
Design 1: Rotator (Barrel Shifter)

- **Rotate circuits**
  - Also known as cyclic shifters
  - A simple and common bit operation

- **Found throughout**
  - microprocessors,
  - cryptography,
  - imaging,
  - biometrics

- **Traditionally, custom designed because of**
  - Highly regular structure
  - Significant routing complexity (local and global)
Design 1: Logical Overview

\[ k[i, j] = r[i] \& k[i-1, j] + ! r[i] \& k[i-1, j + 2i] \]

where \( i = 0, \ldots, m-1 \), \( j = 1, \ldots, n-1 \)

\[ k[i, j] = k[i, j + z \times n], \text{ where } z \text{ is } 0, 1, 2, \ldots, \]

**Example:**
- \( d:01110101 \)
- \( R:101 \)
- \( S:10101110 \)
Design 1 Placement

Logical Design

Physical Design
Design 1 Placement Cont.

- **Highlighted rows:**
  - Helps track changes between manual solution and placed solution
  - Highlights areas of suboptimality

- **Observations**
  - Clustering impact high in areas of more available whitespace
    - Ex: Red stack highly segmented
  - Legalization an issue in areas of little whitespace
    - Ex: Left green stack
Design 2: Structured Trees such as AND/OR Logic

- **Logic 101...**
  - Load/Store Queue (simple memory)
  - Content Addressable Memories
  - Greater than/Less than
  - Basic ALU Operations

- **Common structure repeated regularly**
- **Standard cells can be interchanged to match any of these functions**

**Any 1’s?**

```
```

**All 1’s?**

```
```

**Does A equal B?**

```
```

**OR_REDUCE(A[0:7])**

```
```

**AND_REDUCE(A[0:7])**

```
```

`A[0:3] == B[0:3]`
Design 2: Placement

- Replicable with
  - Large OR-trees
  - Large AND-trees
  - Compare Logic

- Characteristics
  - Many global connections between each bit stack
  - Few local connections between each bit stack
Design 2 Placement Continued

- Custom design verses automatic placement

- Results:
  - Clustering causes logic to clump together
  - Timing and Congestion increase in the process of being quantified
  - Placement unaware of logical partitions
  - Currently: Fixed Latches Improve Overall Placement Results
Errata to Published Benchmarks

- **Goal:** generate an end to end benchmark flow solving the dataflow design problem

- **Changes**
  - Added Clocking Signals
    - reduces HPWL Ratio because latches are fixed
    - Needed for future timing work
  - Pin Locations and Pin Count:
    - Improved Placement
    - Improved Pin Count, more pins for a flatter netlist
  - Simplification – Removed some control logic to focus on datapath placement

- **Why?**
  - Ongoing Research Quantifying Other Areas of Suboptimality
    - Routing
    - Power
    - Delay

- **Published at:** [http://vlsicad.eecs.umich.edu/BK/spb](http://vlsicad.eecs.umich.edu/BK/spb)
  - Special thanks to Professor Igor Markov
# Results: Base Case

## Design 1
- Most placers generate overlaps
- ntuPlace3 Aborted
- CAPO: best overall HPWL

## Design 2
- Most placers generate overlaps
- Overall better than design 1
- ntuPlace3: best overall HPWL

<table>
<thead>
<tr>
<th>Placer</th>
<th>TWL</th>
<th>TWL Ratio</th>
<th>Run Time (s)</th>
<th>TWL</th>
<th>TWL Ratio</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>11000365</td>
<td>1.00</td>
<td>n/a</td>
<td>8642097</td>
<td>1.00</td>
<td>n/a</td>
</tr>
<tr>
<td>Capo</td>
<td>15945589*</td>
<td>1.45*</td>
<td>1453.9</td>
<td>14381067*</td>
<td>1.66*</td>
<td>1430.6</td>
</tr>
<tr>
<td>mPL6</td>
<td>18290965*</td>
<td>1.66*</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>ntuPlace3</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>11110251*</td>
<td>1.29*</td>
<td>533.0</td>
</tr>
<tr>
<td>APlace*</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Dragon</td>
<td>52926316</td>
<td>4.81*</td>
<td>2350.18</td>
<td>34711167</td>
<td>4.02*</td>
<td>2692.0</td>
</tr>
<tr>
<td>FastPlace</td>
<td>16336840*</td>
<td>1.49*</td>
<td>194.9</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

* Completed with Overlaps
ntuPlace3: Aborts during global placement for Design 1
n/a: Did not complete
Results: Whitespace

Design 1 Test Cases
- * generated overlaps
- ntuPlace3 Aborted
- CAPO: best overall HPWL

<table>
<thead>
<tr>
<th>Whitespace</th>
<th>92.5</th>
<th>89.0</th>
<th>85.8</th>
<th>82.8</th>
<th>80.1</th>
<th>77.4</th>
<th>74.0</th>
<th>71.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capo</td>
<td>1.45*</td>
<td>1.49*</td>
<td>1.24*</td>
<td>1.28*</td>
<td>1.14*</td>
<td>1.18*</td>
<td>1.12*</td>
<td>1.11*</td>
</tr>
<tr>
<td>mPL6</td>
<td>1.66*</td>
<td>1.65*</td>
<td>1.64*</td>
<td>1.66*</td>
<td>1.64*</td>
<td>1.63*</td>
<td>1.76*</td>
<td>1.73*</td>
</tr>
<tr>
<td>ntuPlace3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>aPlace</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dragon</td>
<td>4.81</td>
<td>5.00</td>
<td>5.39</td>
<td>5.88</td>
<td>5.83</td>
<td>5.91</td>
<td>6.56</td>
<td>7.37</td>
</tr>
<tr>
<td>FastPlace</td>
<td>1.49*</td>
<td>1.33*</td>
<td>1.31*</td>
<td>1.30*</td>
<td>1.27*</td>
<td>1.27*</td>
<td>1.29*</td>
<td>1.30*</td>
</tr>
</tbody>
</table>

Design 2 Test Cases
- Best HPWL seen at 10% to 15% whitespace
- ntuPlace3: best overall HPWL

<table>
<thead>
<tr>
<th>Whitespace</th>
<th>95.5</th>
<th>93.6</th>
<th>89.5</th>
<th>85.3</th>
<th>81.5</th>
<th>78.1</th>
<th>75.2</th>
<th>72.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capo</td>
<td>1.66*</td>
<td>1.24*</td>
<td>1.17*</td>
<td>1.18*</td>
<td>1.18*</td>
<td>1.20*</td>
<td>1.20*</td>
<td>1.21*</td>
</tr>
<tr>
<td>mPL6</td>
<td>-</td>
<td>1.19*</td>
<td>1.15*</td>
<td>1.72*</td>
<td>1.15*</td>
<td>1.16*</td>
<td>1.17*</td>
<td>1.18*</td>
</tr>
<tr>
<td>ntuPlace3</td>
<td>1.29*</td>
<td>1.12</td>
<td>1.14</td>
<td>1.13</td>
<td>1.20</td>
<td>1.15</td>
<td>1.16</td>
<td>1.24</td>
</tr>
<tr>
<td>aPlace</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dragon</td>
<td>4.02</td>
<td>4.24</td>
<td>4.49</td>
<td>4.81</td>
<td>5.09</td>
<td>5.33</td>
<td>5.60</td>
<td>5.93</td>
</tr>
<tr>
<td>FastPlace</td>
<td>-</td>
<td>1.26</td>
<td>1.15</td>
<td>1.15</td>
<td>1.17</td>
<td>1.19</td>
<td>1.20</td>
<td>1.21</td>
</tr>
</tbody>
</table>
Future Work

- High Density Legalization
  - High utilization of datapath design difficult to solve efficiently

- Routing Aware Placement
  - Easy to pack, hard to route

- Structural Analysis
  - Understanding logic structure can improve results
  - How do we quickly evaluate the logical structure?

- Datapath Extraction
  - Datapath Components in Traditional Random Logic
  - Simultaneous Optimization of Both Styles