From OO to FPGA: Fitting Round Objects into Square Hardware?

Stephen Kou
→ Jens Palsberg

UCLA Computer Science Department
University of California, Los Angeles

Presented at OOPSLA 2010
Our tool: from \textit{OO} to FPGA; \textbf{big energy savings}

- \textit{OO} = object oriented language
- FPGA = field programmable gate array
CPU vs. FPGA vs. ASIC

<table>
<thead>
<tr>
<th></th>
<th>energy use</th>
<th>flexibility</th>
<th>programmability</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>high</td>
<td>high</td>
<td>easy</td>
</tr>
<tr>
<td>FPGA</td>
<td>medium</td>
<td>medium</td>
<td>hard</td>
</tr>
<tr>
<td>ASIC</td>
<td>low</td>
<td>low</td>
<td>extremely hard</td>
</tr>
</tbody>
</table>

So: use ASICs to increase battery lifetime
  - Example: cell phones

But: use FPGAs if you predict lots of modifications
ASIC and FPGA cheat sheet

- Finished ASIC designs: 3,408 in 2006; 3,275 in 2007; then fell 9.5% in 2008 and fell again about 22% in 2009
- Now: 30x more design starts in FPGA over ASIC
- Projected market for FPGAs in 2016: $9.6 billion
- Feature sizes:
  - 2002 Virtex-2 90 nm
  - 2008 Virtex-5 65 nm
  - 2009 Virtex-6 40 nm
  - 2010 Virtex-7 28 nm
The Challenge

Compile a bare object-oriented program to an FPGA with significant energy savings compared to a CPU, while still maintaining acceptable performance and space usage.
How people traditionally program FPGAs

- Write in a hardware description language
  - VHDL
  - Verilog

- Compile with a synthesis tool: VHDL → FPGA
  1. Mapping
  2. Clustering
  3. Placement
  4. Routing
How some people program FPGAs nowadays

- Program in a small subset of C
- Compile to VHDL or Verilog with a high-level synthesis tool
  - AutoESL: AutoPilot (based on xPilot [Cong et al., UCLA])
  - Synopsys: Synphony C Compiler
  - Mentor Graphics: Catapult
- Ponder whether writing directly in VHDL is better
  - Fine-tune speed?
  - Fine-tune energy use?
  - Fine-tune area
  - Really?
From OO to FPGA: a JVM on an FPGA

- Schoeberl [2004]: execute bytecodes on a FPGA
- No comparisons with a CPU
From OO to FPGA: state of the art

- Liquid Metal (Auerbach, Bacon, Cheng, Rabbah, IBM)
- Goal: one language for all platforms
- Approach: careful language design
- Key papers: ECOOP 2008 (DES)

  OOPSLA 2010 (DES + JPEG decoder)
From OO to FPGA: state of the art

- Liquid Metal (Auerbach, Bacon, Cheng, Rabbah, IBM)
- Goal: one language for all platforms
- Approach: careful language design
- Key papers: ECOOP 2008 (DES)

Our goals:

- work with an existing language
- low energy use, good performance, small area
A match made in heaven?

- Virgil is an object-oriented language developed at UCLA [Titzer, OOPSLA 2006; Titzer & P., CASES 2007], targeted to programming small devices, e.g., sensor nodes.

- The Virgil compiler translates to C.

- AutoPilot is a C to FPGA synthesizer.

- Can we do
Virgil

Program initialization phase

Lightweight features

Heap-specific optimization: static analysis

Source Code

IR

initialization

IR

optimization

IR

Heap

Compiler

run
The AutoPilot subset of C

- Places severe limitations on many C constructs
  - Pointers
  - Struct casting
  - Contents of structs

- Rules out the traditional way of compiling OO languages
  - Cannot represent objects with method tables
  - Cannot use structs
Our technique

- OO to FPGA = type case for method dispatch +
grouped arrays +
hybrid object layout
Key features of OO

- Classes, extends, fields, constructors, methods

```java
class Point {
    int x, y;
    Point(int a, int b) {
        x = a; y = b;
    }
    void move(int d) {
        x = x + d; y = y + d;
    }
}

class ColorPoint extends Point {
    int color;
    ColorPoint(int a, int b, int c) {
        super(a, b); color = c;
    }
    void bump(int c) {
        color = c;
        this.move(1);
    }
}
```
Key features of OO

- **Classes**, extends, fields, constructors, methods

```java
class Point {
    int x, y;
    Point(int a, int b) {
        x = a; y = b;
    }
    void move(int d) {
        x = x + d; y = y + d;
    }
}
```

```java
class ColorPoint extends Point {
    int color;
    ColorPoint(int a, int b, int c) {
        super(a, b); color = c;
    }
    void bump(int c) {
        color = c;
        this.move(1);
    }
}
```
Key features of OO

- Classes, `extends`, fields, constructors, methods

```java
class Point {
    int x, y;
    Point(int a, int b) {
        x = a; y = b;
    }
    void move(int d) {
        x = x + d; y = y + d;
    }
}

class ColorPoint extends Point {
    int color;
    ColorPoint(int a, int b, int c) {
        super(a, b); color = c;
    }
    void bump(int c) {
        color = c;
        this.move(1);
    }
}
```
Key features of OO

- Classes, extends, fields, constructors, methods

```java
class Point {
    int x,y;
    Point(int a, int b) {
        x=a; y=b;
    }
    void move(int d) {
        x=x+d; y=y+d;
    }
}

class ColorPoint extends Point {
    int color;
    ColorPoint(int a, int b, int c) {
        super(a,b); color=c;
    }
    void bump(int c) {
        color=c;
        this.move(1);
    }
}
```
Key features of OO

- Classes, extends, fields, **constructors**, methods

```java
class Point {
    int x, y;
    Point(int a, int b) {
        x = a; y = b;
    }
    void move(int d) {
        x = x + d; y = y + d;
    }
}
class ColorPoint extends Point {
    int color;
    ColorPoint(int a, int b, int c) {
        super(a, b); color = c;
    }
    void bump(int c) {
        color = c;
        this.move(1);
    }
}
```
Key features of OO

- Classes, extends, fields, constructors, methods

```java
class Point {
    int x,y;
    Point(int a, int b) {
        x=a; y=b;
    }
    void move(int d) {
        x=x+d; y=y+d;
    }
}
class ColorPoint extends Point {
    int color;
    ColorPoint(int a, int b, int c) {
        super(a,b); color=c;
    }
    void bump(int c) {
        color=c;
        this.move(1);
    }
}
```

UCLA
COMPUTER SCIENCE DEPARTMENT
Two objects, standard (horizontal) layout

- Point p = new Point();
- ColorPoint cp = new ColorPoint();

An object is a heap pointer

Problem: pointers! Not supported by AutoPilot
An object is an integer

<table>
<thead>
<tr>
<th></th>
<th>point1</th>
<th>point2</th>
<th>point3</th>
<th>colorpoint1</th>
<th>colorpoint2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_x</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Row_y</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Row_color</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
### Idea for saving space: an extra table (!! :-)

<table>
<thead>
<tr>
<th>Row_x</th>
<th>7</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_y</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Row_color</td>
<td>10</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>point1</th>
<th>point2</th>
<th>point3</th>
<th>colorpoint1</th>
<th>colorpoint2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row_x</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Row_y</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Row_color</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>0</td>
</tr>
</tbody>
</table>
Improved idea: drop extra table, keep tuples

An object is a tuple

Row_x:

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>8</th>
</tr>
</thead>
</table>
Row_y:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>6</th>
<th>4</th>
<th>7</th>
<th>12</th>
</tr>
</thead>
</table>
Row_color:

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>5</th>
</tr>
</thead>
</table>

point1  point2  point3  colorpoint1  colorpoint2

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Ultimate idea: condensed rows

An object is a **tuple**

<table>
<thead>
<tr>
<th>Row_Point</th>
<th>7</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Row_ColorPoint</td>
<td>10</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>point1</th>
<th>point2</th>
<th>point3</th>
<th>colorpoint1</th>
<th>colorpoint2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>--------------</td>
</tr>
</tbody>
</table>
Instead of function pointers: custom dispatcher

```c
void move_dispatch(struct Tuple __this, int d) {
    switch( Row_Point[__this.f0].Typeld ) {
        case 101:    // Point, ColorPoint
            return Point_move(__this, d);
    }
}
```

We added a field Typeld to each entry of Row_Point
Experimental results: our platforms

- CPU (xeon) 2.66 GHz TDP = 80 W
- CPU (atom) 1.6 GHz TDP = 4 W
- FPGA (Xilinx Virtex-II) 100 MHz N/A

Auerbach et al. [previous paper] run on a Xilinx Virtex-5

- TDP = Thermal Design Power (can be viewed as a max)
  - Excludes power for memory, storage drives, etc.
## Experimental results: our benchmarks

<table>
<thead>
<tr>
<th>Originally in C:</th>
<th>Lines of code</th>
<th>Originally in C++:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>791</td>
<td>669</td>
</tr>
<tr>
<td>Blowfish</td>
<td>1,320</td>
<td>1,548</td>
</tr>
<tr>
<td>SHA</td>
<td>1,349</td>
<td>1,187</td>
</tr>
<tr>
<td>Richards</td>
<td>705</td>
<td>437</td>
</tr>
</tbody>
</table>
## Experimental results: C vs. Virgil

<table>
<thead>
<tr>
<th>SHA1</th>
<th>CPU (xeon)</th>
<th>CPU (atom)</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (us)</td>
<td>energy (mJ)</td>
<td>time (us)</td>
</tr>
<tr>
<td>C</td>
<td>319</td>
<td>25.4</td>
<td>1,093</td>
</tr>
<tr>
<td>Virgil</td>
<td>1,074</td>
<td>85.9</td>
<td>2,630</td>
</tr>
</tbody>
</table>
## Experimental results: C++ vs. Virgil

### Richards

<table>
<thead>
<tr>
<th></th>
<th>CPU (xeon)</th>
<th></th>
<th></th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (us)</td>
<td>energy (mJ)</td>
<td>time (us)</td>
<td>time (us)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>energy (mJ)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>area (slices)</td>
</tr>
<tr>
<td>C++</td>
<td>10,065</td>
<td>805.2</td>
<td>39,900</td>
<td>159.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>Virgil</td>
<td>29,135</td>
<td>2,330.8</td>
<td>61,622</td>
<td>246.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14,433</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,317</td>
</tr>
</tbody>
</table>
Conclusion

- OO to FPGA is possible
- Energy savings!
  - Virgil on an FPGA beats C++ on an Atom by 8x
- Faster OO code!
  - Virgil on an FPGA beats C++ on an Atom by 3x
- Competitive area