UCLA COMPUTER SCIENCE DEPARTMENT

From OO to FPGA:

Fitting Round Objects into Square Hardware?

Stephen Kou

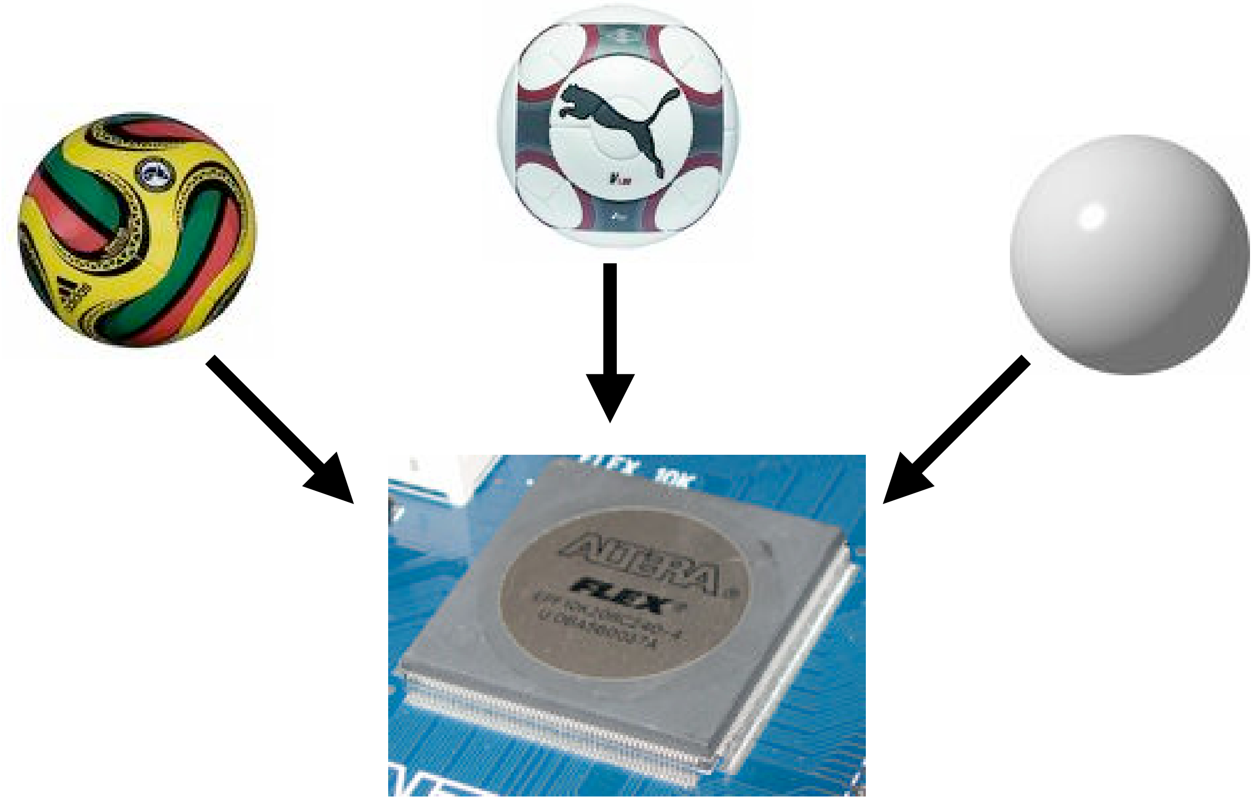
➔ Jens Palsberg

UCLA Computer Science Department
University of California, Los Angeles

Presented at OOPSLA 2010

Our tool: from OO to FPGA; big energy savings

- **OO = object oriented language**
- **FPGA = field programmable gate array**



CPU vs. FPGA vs. ASIC

	energy use	flexibility	programmability
CPU:	high	high	easy
FPGA:	medium	medium	hard
ASIC:	low	low	extremely hard

- 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:

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: VDHL → 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)

  OOPSLA 2010 (DES + JPEG decoder)

Our goals:

• work with an existing language

• low energy use, good performance, small area

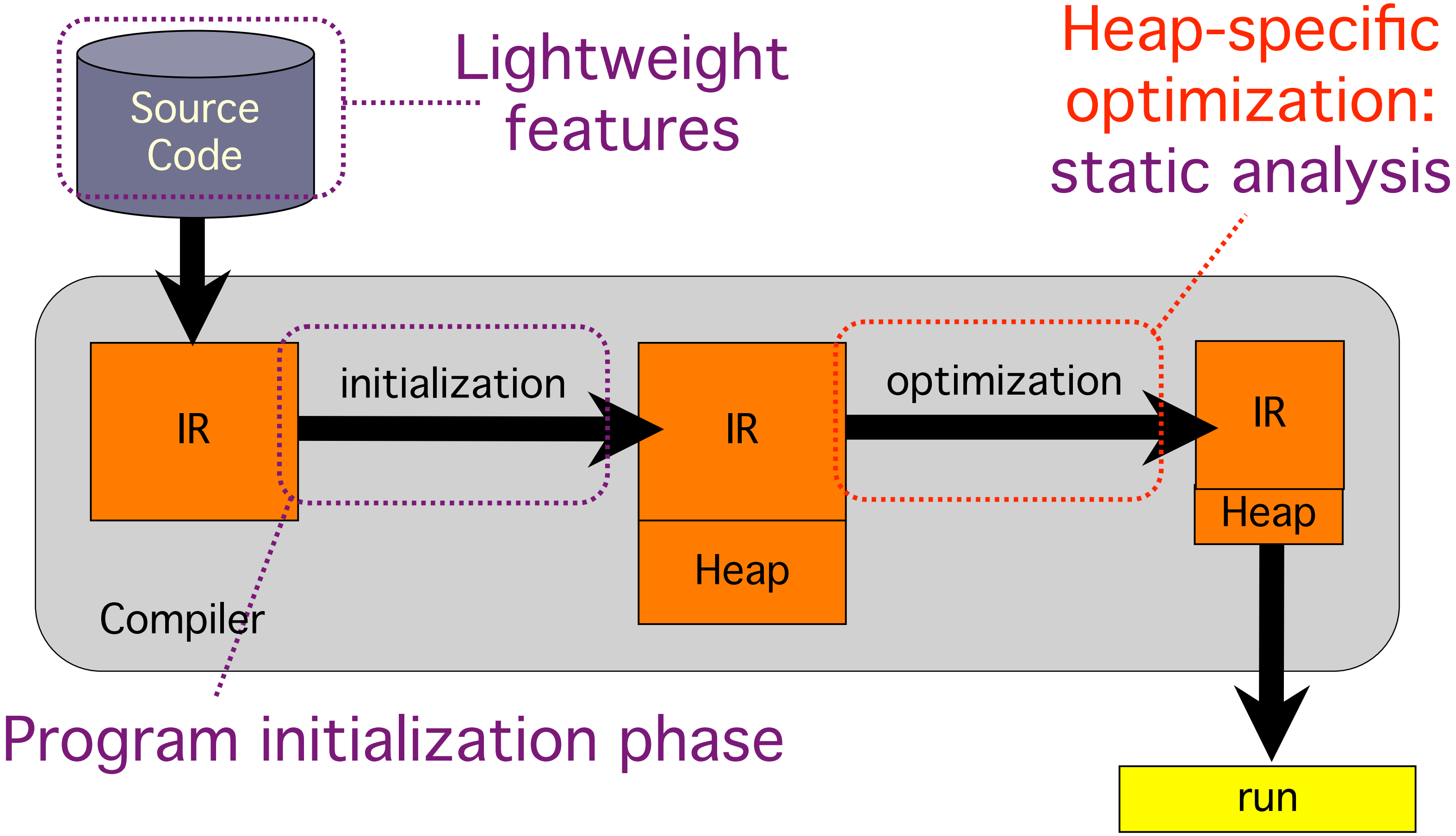
UCLA
COMPUTER SCIENCE DEPARTMENT

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

$$\text{Virgil} \xrightarrow{\quad ?? \quad} \text{C} \xrightarrow{\quad \text{AutoPilot} \quad} \text{FPGA}$$

Virgil



Source Code

Lightweight features

Heap-specific optimization: static analysis

IR

initialization

IR

Heap

optimization

IR

Heap

Compiler

run

Program initialization phase

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

```
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

```
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

```
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

```
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

```
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

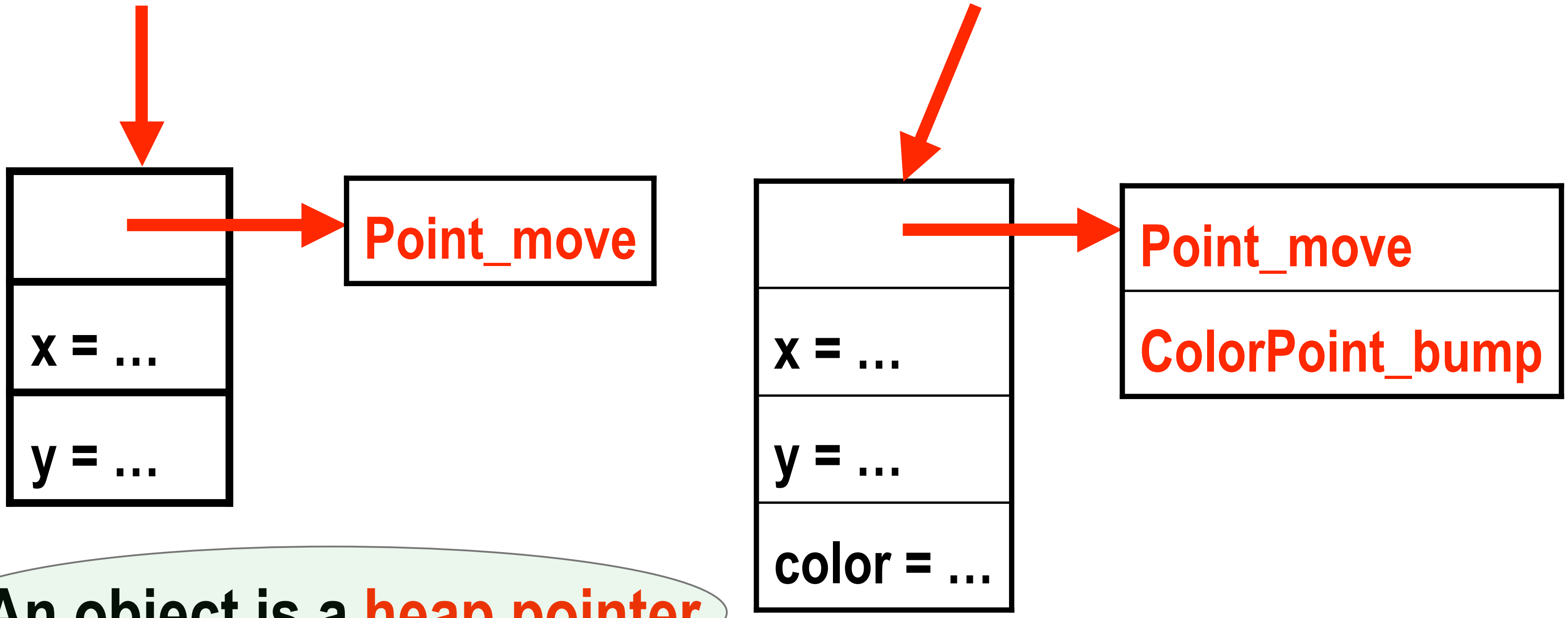
- Classes, extends, fields, constructors, methods

```
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);
    }
}
```

Two objects, standard (horizontal) layout

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



x = …

y = …

Point_move

x = …

y = …

color = …

Point_move

ColorPoint_bump

An object is a heap pointer

Problem: pointers! Not supported by AutoPilot

Five objects, vertical layout [Titzer & P., 2007]

	point1	point2	point3	colorpoint1	colorpoint2
Row_x :	7	4	5	2	8
Row_y :	1	6	4	7	12
Row_color :	----	----	-----	10	5

An object is an integer

Idea for saving space: an extra table (!! :-)

Row_x :	7	4	5	2	8
Row_y :	1	6	4	7	12
Row_color :	10	5			

	point1	point2	point3	colorpoint1	colorpoint2
Row_x :	0	1	2	3	4
Row_y :	0	1	2	3	4
Row_color :	----	----	-----	0	1

Improved idea: drop extra table, keep tuples

An object is a tuple

Row_x :	7	4	5	2	8
Row_y :	1	6	4	7	12
Row_color :	10	5			

point1	point2	point3	colorpoint1	colorpoint2
0	1	2	3	4
0	1	2	3	4
----	----	-----	0	1

Ultimate idea: condensed rows

An object is a tuple

Row_Point :

7	4	5	2	8
1	6	4	7	12

Row_ColorPoint :

10	5

point1	point2	point3	colorpoint1	colorpoint2
0	1	2	3	4
----	----	------	0	1

Instead of function pointers: custom dispatcher

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

We added a field TypeId 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

	Lines of code	
	Original	Virgil
Originally in C:		
AES	791	669
Blowfish	1,320	1,548
SHA	1,349	1,187
Originally in C++:		
Richards	705	437

Similar!

Experimental results: C vs. Virgil

SHA1	CPU (xeon)		CPU (atom)		FPGA		
	time (us)	energy (mJ)	time (us)	energy (mJ)	time (us)	energy (mJ)	area (slices)
C	319	25.4	1,093	4.37	1,565	2.07	5,715
Virgil	1,074	85.9	2,630	10.52	1,525	2.04	4,890

Experimental results: C++ vs. Virgil

Richards

	CPU (xeon)		CPU (atom)		FPGA		
	time (us)	energy (mJ)	time (us)	energy (mJ)	time (us)	energy (mJ)	area (slices)
C++	10,065	805.2	39,900	159.60	N/A	N/A	N/A
Virgil	29,135	2,330.8	61,622	246.49	14,433	18.91	4,317

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