VIRTUALIZATION IN THE AGE OF HETEROGENEOUS MACHINES

David F. Bacon

The ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE ’11)

Keynote - March 9, 2011
What Heterogeneous Machines?
WHAT HETEROGENEOUS MACHINES?
What Heterogeneous Machines?

It's The Multicore Era!
The Path of Least Imagination

For years, computer architects have been saying that a big new idea in computing was needed. Indeed, as transistors have continued to shrink, rather than continuing to innovate, computer designers have simply adopted a so-called “multicore” approach, where multiple processors are added as more chip real estate became available.

Source: Remapping Computer Circuitry to Avert Impending Bottlenecks, February 28, 2011
MEANWHILE...

GPU

FPGA

Tilera 64

Cell BE

IBM WSP
### What is Performance?

<table>
<thead>
<tr>
<th>Chip</th>
<th>flops/cycle</th>
<th>freq (GHz)</th>
<th>Gflops (peak)</th>
<th>Gflops/watt</th>
<th>Mflops $</th>
<th>Mflops watt/$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i7</td>
<td>32</td>
<td>3.2</td>
<td>102</td>
<td>0.8</td>
<td>70</td>
<td>0.7</td>
</tr>
<tr>
<td>AMD 9270</td>
<td>1600</td>
<td>0.75</td>
<td>1200</td>
<td>5.5</td>
<td>800</td>
<td>3.6</td>
</tr>
<tr>
<td>Xilinx V5 LX330</td>
<td>1040</td>
<td>0.55</td>
<td>550</td>
<td>13.7</td>
<td>138</td>
<td>8.1</td>
</tr>
</tbody>
</table>

**Peak Performance**

## Performance, Take 2

### Actual Performance (Random Number Generation)

<table>
<thead>
<tr>
<th>Chip</th>
<th>Gsamples/sec</th>
<th>Msamples/joule</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Intel Core 2</td>
<td>1.4</td>
<td>5</td>
</tr>
<tr>
<td>GPU nVidia GTX280</td>
<td>14</td>
<td>115</td>
</tr>
<tr>
<td>FPGA Xilinx Virtex 5</td>
<td>44</td>
<td>1461</td>
</tr>
</tbody>
</table>

Source: Thomas et al, A comparison of CPUs, GPUs, FPGAs and massively parallel processor arrays for random number generation, 2009
What’s the Best Use of Transistors?

Homogeneous vs. Heterogeneous
WHAT’S THE BEST USE OF TRANSISTORS?

Homogeneous vs. Heterogeneous

Keep Transistors Busy vs. Turn Transistors Off
Should We Virtualize Heterogeneous Architectures?
How?
At the Language or System Level?

Or Both?
What is “Virtualization” Anyway?
The Only 2 Ideas in Computer Science
THE ONLY 2 IDEAS IN COMPUTER SCIENCE

Hashing $f(x)$
The Only 2 Ideas in Computer Science

Hashing

\[ f(x) \]

Indirection
THE ONLY 2 IDEAS IN COMPUTER SCIENCE

Hashing  $f(x)$

Indirection
SYSTEM VS. LANGUAGE VMs

User-Mode
ISA

Supervisor-Mode
ISA

Device Drivers
SYSTEM VS. LANGUAGE VMs
**SYSTEM vs. LANGUAGE VMs**

- **Device Drivers**
  - Supervisor-Mode ISA
  - User-Mode ISA
  - x86

- **System vs. Language VMs**
  - Supervisor-Mode ISA
  - User-Mode ISA
  - x86
**SYSTEM vs. LANGUAGE VMs**
SYSTEM vs. LANGUAGE VMs

- Virtualize Environment
- Hold ISA Constant

- Virtualize Processor
- Vary ISA
OVERLAP: SYNERGY AND CONFLICT
OVERLAP: SYNERGY AND CONFLICT

Memory
OVERLAP: SYNERGY AND CONFLICT

Memory

Threads
OVERLAP: SYNERGY AND CONFLICT

Memory

Threads

User-Mode ISA

Supervisor-Mode ISA
DEVICE OR PROCESSOR?

GPU

FPGA
THE “ACCELERATOR” MODEL

GPU

CPU

Main Program

FPGA
THE “ACCELERATOR” MODEL

GPU

Force Calc

Main Program

CPU

FPGA
THE “ACCELERATOR” MODEL

Force Calc

Main Program

FFT

GPU

CPU

FPGA
The Reality
THE REALITY
THE REALITY
THE REALITY
The Reality
THE REALITY
THE REALITY
THE REALITY
Language VMs for Heterogenous Systems
<table>
<thead>
<tr>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joshua Auerbach</td>
</tr>
<tr>
<td>Perry Cheng</td>
</tr>
<tr>
<td>Rodric Rabbah</td>
</tr>
<tr>
<td>Christophe Dubach</td>
</tr>
<tr>
<td>David F. Bacon</td>
</tr>
<tr>
<td>Steven Fink</td>
</tr>
<tr>
<td>Sunil Shukla</td>
</tr>
<tr>
<td>Yu Zhang</td>
</tr>
</tbody>
</table>
HETEROGENEOUS PROGRAMMING

- Java
- C++
- Python

- Cuda
- OpenCL

- C+Intrinsics

- VHDL
- Verilog
- SystemC

- CPU Compiler
- GPU Compiler
- Node Compiler
- Synthesis

- binary
- binary
- binary
- bitfile

- CPU
- GPU
- WSP
- FPGA
GPU Compiler ➔ binary ➔ GPU ➔ Node Compiler ➔ binary ➔ WSP ➔ Synthesis ➔ bitfile ➔ FPGA
THE LIQUID METAL PROGRAMMING LANGUAGE

Lime

Lime Compiler

CPU Backend
- bytecode
- CPU

GPU Backend
- binary
- GPU

Node Backend
- binary
- WSP

Verilog Backend
- bitfile
- FPGA
THE ARTIFACT STORE & EXCLUSION

Lime

Lime Compiler

bytecode
binary
binary
bitfile

Artifact Store

CPU
GPU
WSP
FPGA
THE ARTIFACT STORE & EXCLUSION

Lime

Lime Compiler

Artifact Store

bytecode

binary

binary

bitfile

CPU

GPU

WSP

FPGA
THE ARTIFACT STORE & EXCLUSION

Lime

Lime Compiler

Artifact Store

bytecode
binary
bitfile

CPU
GPU
WSP
FPGA
THE ARTIFACT STORE & EXCLUSION

Lime

Lime Compiler

Artifact Store

bytocode

binary

bitfile

CUDA

GPU

WSP

FPGA
Heterogeneous Execution of Lime

Lime

Lime Compiler

Artifact Store

bytecode

binary

bitfile

CPU

GPU

WSP

FPGA
HETEROGENEOUS EXECUTION OF LIME

Lime

Lime Compiler

Artifact Store

bytecode

binary

bitfile

Lime Virtual Machine (LmVM)

CPU

GPU

WSP

FPGA
HETEROGENEOUS EXECUTION OF LIME

Lime

Lime Compiler

Artifact Store

bytecode

binary

bitfile

Lime Virtual Machine (LmVM)

CPU

GPU

WSP

FPGA

“Bus”
HETEROGENEOUS EXECUTION OF LIME

Lime

Lime Compiler

Artifact Store

bytecode

binary

bitfile

Lime Virtual Machine (LmVM)

CPU

GPU

WSP

FPGA

“Bus”
HETEROGENEOUS EXECUTION OF LIME

Lime

Lime Compiler

Artifact Store

Lime Virtual Machine (LmVM)

CPU
bytecode

GPU

WSP

FPGA
bitfile

“Bus”
THE LIME LANGUAGE: VIRTUALIZING HETEROGENEOUS COMPUTATION
LIME: JAVA IS (ALMOST) A SUBSET

% javac MyClass.java
% java MyClass
LIME: JAVA IS (ALMOST) A SUBSET

% javac MyClass.java
% java MyClass

% mv MyClass.java MyClass.lime
LIME: JAVA IS (ALMOST) A SUBSET

% javac MyClass.java
% java MyClass

% mv MyClass.java MyClass.lime
% limec MyClass.lime
LIME: JAVA IS (ALMOST) A SUBSET

% javac MyClass.java
% java MyClass

% mv MyClass.java MyClass.lime
% limec MyClass.lime
% java MyClass
LIME: JAVA IS (ALMOST) A SUBSET

% javac MyClass.java
% java MyClass

% mv MyClass.java MyClass.lime
% limec MyClass.lime
% java MyClass

INCREMENTALLY USE LIME FEATURES
LIME LANGUAGE OVERVIEW

Core Features
- Programmable Primitives
- Stream Programming
- Collective Operations

Supporting Features
- Reifiable Generics
- Ranges, Bounded “for”
- User-defined operators
- Typedefs
- Local type inference
- Tuples

Features:
- Immutable Types
- Bounded Types
- Bounded Arrays
- Primitive Supertypes

Parallelism Types:
- BIT-LEVEL PARALLELISM
- PIPELINE PARALLELISM
- DATA PARALLELISM

Other Features:
- Graph Construction
- Isolation Enforcement
- Closed World Support
- Rate Matching
- Messaging
## Virtualization Comparison

<table>
<thead>
<tr>
<th>Physical</th>
<th>Virtual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>Thread</td>
</tr>
<tr>
<td>RAM</td>
<td>Heap</td>
</tr>
<tr>
<td>byte</td>
<td>word</td>
</tr>
<tr>
<td>float</td>
<td>dfloat</td>
</tr>
<tr>
<td>Compare&amp;Swap</td>
<td>synchronized</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical</th>
<th>Virtual</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>bit</td>
</tr>
<tr>
<td>CLBs</td>
<td>User Primitives</td>
</tr>
<tr>
<td>Slices</td>
<td>tasks</td>
</tr>
<tr>
<td>Block RAM</td>
<td>Map</td>
</tr>
<tr>
<td>Routing Network</td>
<td>Streams</td>
</tr>
<tr>
<td>DMA, SerDes</td>
<td>Reduce</td>
</tr>
<tr>
<td>DSP Multipliers</td>
<td>integer&lt;N&gt;</td>
</tr>
</tbody>
</table>
var uppercaser = task work;

local char work(char c) {
    return toUpper(c);
}
task creation (stateless)

```plaintext
local char work(char c) {
    return toUpper(c);
}

stream
primitive filter
worker
method
port
type
var.uppercaser = task work;

Isolation Keywords
Task Creation
port
type
port
worker
method
stream
type
stream
primitive filter
```
var pipeline = task worker1 => task worker2 => task worker3;
var pipeline = task worker1 => task worker2 => task worker3;
SOURCES AND SINKS

Heap

File System

source

filter

sink

reader(…) { … }

worker(…) { … }

writer(…) { … }

/tmp/mydata

/dev/tty
public static void main(string[][] args) {

    char[][] msg = {
        'H', 'E', 'L', 'L', 'O', ',', ',', ',',
        'W', 'O', 'R', 'L', 'D', '!', ',', ' \n'};

    var hello = msg.source(1) =>
        task Character.toLowerCase(char) =>
            task System.out.print(char);

    hello.finish();
}
DEMO
HELLO WORLD
LIME/ECLIPSE ENVIRONMENT
package helloworld;

public class HelloWorld3b {

    public static void main(String[] args) {
        char[] msg = {'H', 'E', 'L', 'L', 'O', ',', ',', 'W', 'O', 'R', 'L', 'D', ',', '!', ',', '

        var hello = msg.source(1) =>
            task Character.toLowerCase(char) =>
                task System.out.print(char);

        hello.rendezvous();
    }
}
var averager = task Averager().avg;

double avg(double x) {
    total += x;
    return total/++count;
}
```
var matchedpipe = task AddStuff().work1 => # => task work2;
```
DEMO

N-BODY SIMULATION: CPU VS. GPU
9x SPEEDUP (9.26 GFLOPS) ON LAPTOP
VIRTUALIZATION OF DATA MOVEMENT
VIRTUALIZATION OF DATA MOVEMENT

+
VIRTUALIZATION OF DATA MOVEMENT

=》

3 months

+  x

months
WHAT’S SO HARD?

- Projects routinely spend 60-80% of time on data xfer
- Verilog is hard, but driving devices and buses is harder
- Motherboard chips, BIOS settings: 4x slowdown each
- Signal transitions are considered an “interface” spec
- “IP” extremely sensitive to configurations
  - and often broken when it mismatches the original
- Device drivers also highly sensitive
  - Often specialized to one I/O style; fall off cliff otherwise
- Hardware design culture accepts this as C.O.D.B.
COLLECTIVE OPERATIONS
DATA PARALLELISM
Array Parallelism

```c
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```
Array Parallelism

```c
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```

Indexable<int,float>
ARRAY PARALLELISM

float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;

Indexable<int,float>

index

0
1
2

a

1.2
0.0
3.14

b

0.0
99.0
-3.0
**ARRAY PARALLELISM**

```plaintext
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;

Indexable<int, float> domain()
```

<table>
<thead>
<tr>
<th>index</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.2</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>99.0</td>
</tr>
<tr>
<td>2</td>
<td>3.14</td>
<td>-3.0</td>
</tr>
</tbody>
</table>
Array Parallelism

```c
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```

Indexable<int, float>

![Diagram showing array parallelism with values and domains](image)

Indexable domain:

- a:
  - 0: 1.2
  - 1: 0.0
  - 2: 3.14

- b:
  - 0: 0.0
  - 1: 99.0
  - 2: -3.0
**ARRAY PARALLELISM**

```plaintext
float[ ] a = ...;
fput[ ] b = ...;
float[ ] c = a @+ b;
```

Indexable<int, float>

Indexable domain( )

```
<table>
<thead>
<tr>
<th>index</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.2</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>99.0</td>
</tr>
<tr>
<td>2</td>
<td>3.14</td>
<td>-3.0</td>
</tr>
</tbody>
</table>
```

0 :: 2 == 0 :: 2
ARRAY PARALLELISM

float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;

Indexable<int, float>
Collectable<int, float>

```
float[ ] a = [1.2, 0.0, 3.14];
float[ ] b = [0.0, 99.0, -3.0];
float[ ] c = a @+ b;
```

domain() == 0 :: 2

Indexable<int, float>
Collectable<int, float>
**Array Parallelism**

```plaintext
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```

Indexable<int,float>
Collectable<int,float>

```
<table>
<thead>
<tr>
<th>index</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.2</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
<td>99.0</td>
</tr>
<tr>
<td>2</td>
<td>3.14</td>
<td>-3.0</td>
</tr>
</tbody>
</table>
```

collector
domain(
)
? ==
0 :: 2

```
**ARRAY PARALLELISM**

float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;

Indexable<int,float>
Collectable<int,float>

```
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```

```
Indexable<int,float>
Collectable<int,float>
```
Array Parallelism

float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;

Indexable<int,float>
Collectable<int,float>

```
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```

```
float[ ] a = ...;
float[ ] b = ...;
float[ ] c = a @+ b;
```
Many Things are Collectable

```csharp
var a = new Matrix<3, quaternion>(100, 200, 100);
var b = new Matrix<3, quaternion>(100, 200, 100);
...
var c = a @ + b;
```

```csharp
Map<string, List<int>> a = ...
Map<string, List<int>> b = ...
Map<string, List<int>> c = a @ append(b);
```

```csharp
string foo = Character.toLowerCase(@ "FOO");
```

package lime.util.synthesizable;

public class FixedHashMap<K extends Value, V extends Value, BUCKETS extends ordinal<BUCKETS>, LINKS extends ordinal<LINKS>> extends AbstractMap<K,V> {
    protected final nodes = new Node<K,V>[BUCKETS][LINKS];
}
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return ! vals;
}
Get Operation, Step 1: Select Row

```java
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return | ! vals;
}
```
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return | ! vals;
}

**STEP 2: COMPARE ALL KEYS**
Step 2: Compare All Keys

public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return | ! vals;
}
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return | ! vals;
}
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return ! vals;
}
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return | ! vals;
}
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return ! vals;
}
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return | ! vals;
}
FilteredValues/Zeros

```java
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return ! vals;
}
```
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return ! ! vals;
}
Step 4: Or-Reduce For Result

public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return ! vals;
}

vals V 0
Step 4: Or-Reduce For Result

```java
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row @ compareKey(key);
    V[LINKS] vals = row @ getValueOrDefault(selections);
    return ! vals;
}
```
public local V get(K key) {
    Node[LINKS] row = nodes[hash(key)];
    boolean[LINKS] selections = row.compareKey(key);
    V[LINKS] vals = row.getValueOrDefault(selections);
    return vals;
}
VIRTUALIZATION OF DATA PARALLELISM

@!
CURRENT RESULTS
How Do we Evaluate Performance?

- Speedup for Naïve Users
  - How much faster than Java?

- Slowdown for Expert Users?
  - How much slower than hand-tuned low-level code?

- Our methodology:
  - Write/tune/compare 4 versions of each benchmark:
    - Java, Lime, OpenCL, Verilog
  - Doesn’t address flops/watt, flops/watt/$, productivity
EXPERT VS NAÏVE SPEEDUP: END-TO-END

(JAVA BASELINE)

- Photomosaic: GPU
  - Expert: 6.67x
  - Lime: 6.10x

- N-body: GPU
  - Expert: 136x
  - Lime: 66x

- IDCT: FPGA
  - Expert: 1.06x
  - Lime: 1.06x

- DES: FPGA
  - Expert: 1.0x
  - Lime: 0.04x
EXPERT VS NAÏVE SPEEDUP: KERNEL TIME

(JAVA BASELINE)

speedup normalized to manual implementations

37x  193x  10x  150x

Expert  Lime

26x  126x  10x  0.02x

phatomosaic  n-body  idct  des

GPU  GPU  FPGA  FPGA
Should We Virtualize Heterogeneous Architectures?
Yes We Can!
“User” vs. “Supervisor”
ACCELERATOR VS. FIRST-CLASS DEVICE
Another Unsolved Problem
SUMMARY

• Heterogeneity is Here to Stay
SUMMARY

• Heterogeneity is Here to Stay

• Multicore Isn’t Dead - Just Less Important
SUMMARY

• Heterogeneity is Here to Stay

• Multicore Isn’t Dead - Just Less Important

• HLLs Can Insulate Programmer from Low-Level Details
SUMMARY

• Heterogeneity is Here to Stay

• Multicore Isn’t Dead - Just Less Important

• HLLs Can Insulate Programmer from Low-Level Details

• But not fundamental computational structure
SUMMARY

• Heterogeneity is Here to Stay

• Multicore Isn’t Dead - Just Less Important

• HLLs Can Insulate Programmer from Low-Level Details
  • But not fundamental computational structure

• System-Level Virtualization is an Open (Hard) Problem
REAL-TIME PHOTO MOSAIC
REAL-TIME PHOTO MOSAIC
Questions?