ETH zürich

Terra: A Multi-Stage Language for High-Performance Computing

Zachary DeVito*, James Hegarty*, Alex Aiken*, Pat Hanrahan*, Jan Vitek+

* Stanford University, +Purdue University

Presented by Gregor Wegberg

Content

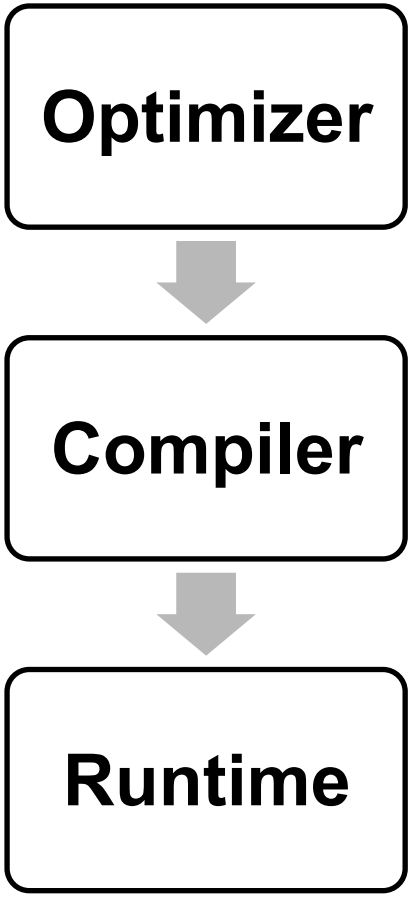
- The Problem
- Our Aim
- What is Terra?
- Terra Core
- Multi-Stage Programming
- Performance
- Future Work
- My Personal Comment

The Problem: Reality

- Increasing demand for high-performance and power-efficiency
- Range of target architectures
- Range of available high-performance libraries

The Problem: Current Solutions

- Auto-Tuners
- Domain-Specific Languages (DSL)



Our Aim

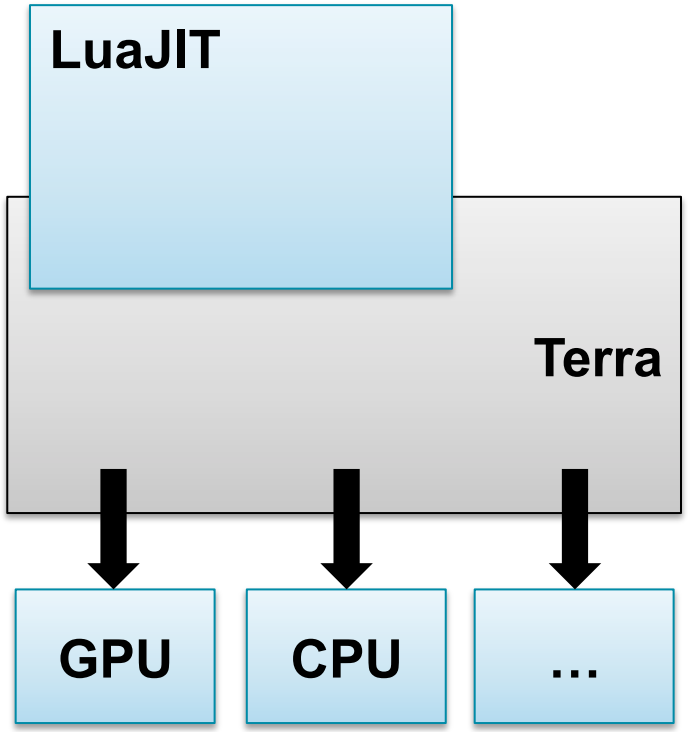
- Easy prototyping & experimenting
- Generate high-performance code dynamically
- Runtime feedback to improve performance
- Use available libraries

The Big Picture: What is Terra?

- New programming language for HPC
- Extends Lua with C like «low-level» features
- Multi-Stage Programming
- Type Reflection
- APIs: DSL & C Libraries

Terra: Lua for HPC

- Terra extends Lua
- Generates code
- Can run independently



Terra: Basics

```
function doubleLua(n)
        return 2 * n
end
```

```
terra doubleTerra(n : int)
        return 2 * n
end
```

```
doubleLua(5)    -- 5
doubleTerra(5)  -- 5
```

- Dynamically typed
- First-class functions
- Garbage Collector

- Statically typed
- Manual memory management
- C like

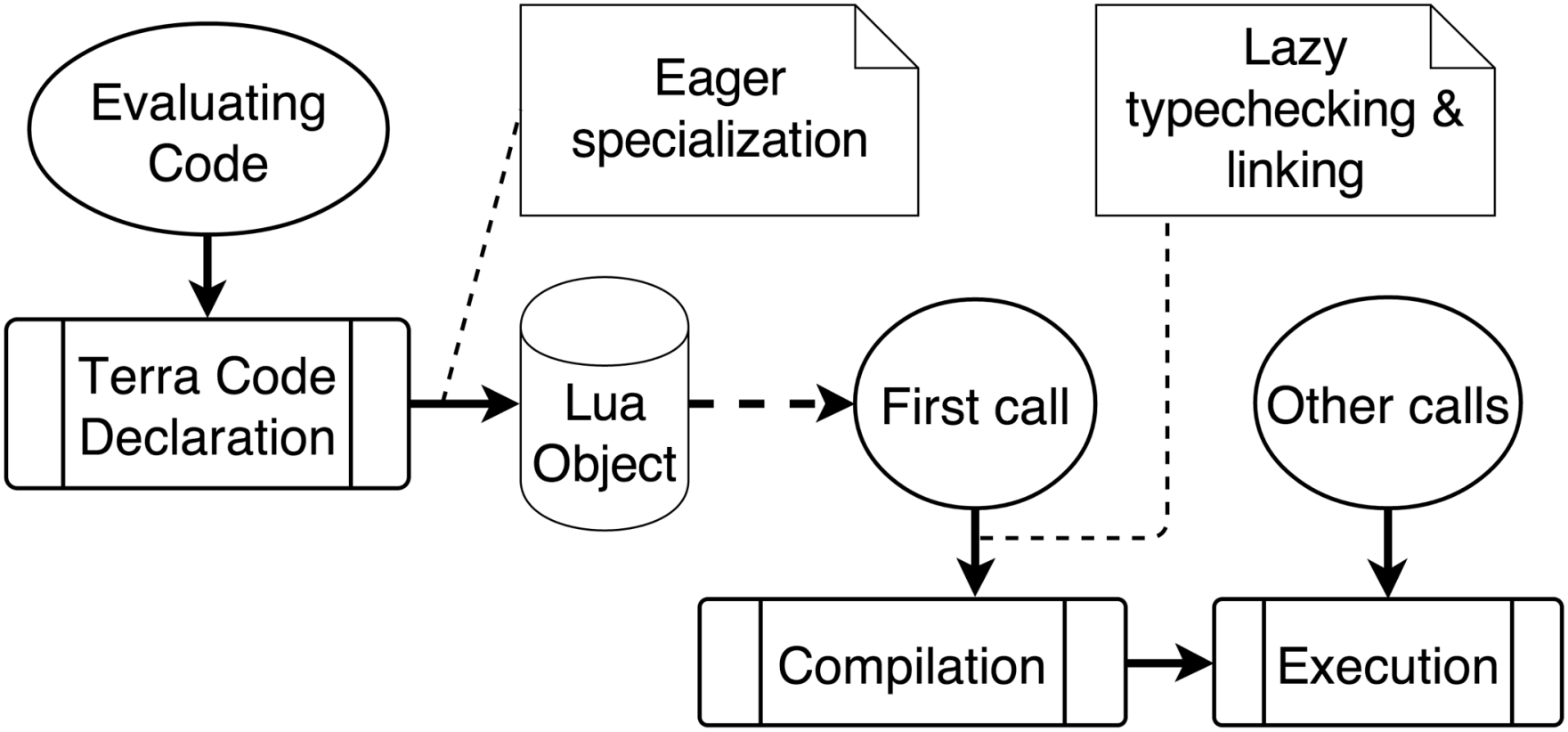
Terra: Basics

```
> doubleTerra:disas()

define i32 @doubleTerra0(i32) {
entry:
  %1 = shl i32 %0, 1
  ret i32 %1
}

definition   {int32}->{int32}
assembly for function at address 0xe0ed070
0xe0ed070(+0):      lea    EAX, DWORD PTR [EDI + EDI]
0xe0ed074(+4):      ret
```

Terra : Execution Steps



Terra: Eager Specialization

```
local v = 10

terra doTerra()
  print(v)
end

function doLua()
  print(v)
end
```

```
doTerra() -- output: 10
doLua() -- output: 10

v = 20 -- change `v`

doTerra() -- output: 10
doLua() -- output: 20
```

Terra Core

- Big step operational semantics
- Formalizes how Terra and Lua interact

Multi-Stage Programming (MSP)

- Runtime program generation & execution
- Don’t pay runtime penalty for your generality!

Multi-Stage Programming (MSP)

General Function

1. How big is my L1 & L2 cache?
2. Allocate resources accordingly
3. Do the calculation
4. Repeat

The MSP Way of Life

1. How big is my L1 & L2 cache?
2. Create actual function accordingly
3. Call generated program

Terra: MSP Example

```
function Matrix(saveType)
  local struct MatrixImpl {
    data : &double,
    size : int
  }

-- omitted:
-- init(), free(),
-- get(), set()

  return MatrixImpl
end
```

```
if saveType == "row" then -- row-major

    terra MatrixImpl:_calcIndex(row : int,
      col : int)

      return row * self.size + col
    end

  else -- column-major

    terra MatrixImpl:_calcIndex(row : int,
      col : int)

      return row + col * self.size
    end

  end
```

Terra: MSP Example

```
MyMatrixType = Matrix("row")
```

```
MyMatrixType = Matrix("col")
```

```
terra doStuff()
  var mat = MyMatrixType {}
  mat:init(10)

  mat:set(0, 0, 10)
  mat:set(5, 4, 20)

  print(mat:get(0, 0)) -- 10
  print(mat:get(5, 4)) -- 20

  mat:free()
end
```

Performance

- Double-precision general matrix multiply (DGEMM):
  - Terra in range of 20% of ATLAS' performance
  - (ATLAS: Automatically Tuned Linear Algebra Software, auto-tuner)
- Single-precision general matrix multiply (SGEMM):
  - Terra in same range as ATLAS' performance

Future Work

- Support for more architectures (e.g. GPUs, Intel's MIC)
- Tool support (e.g. Terra aware debugger)
- Libraries (e.g. inheritance system)

My Personal Comment: Other Projects

- «Liquid Metal» by IBM Research
- Runs on CPUs, GPUs, FPGAs
- Move computation across hardware
- New Language called «Lime»
- Extends Java

My Personal Comment

- Multi-Stage Programming is very interesting
- Don’t forget DSL and Type Reflection

ETH zürich

Thank You!

Questions & Discussion

MSP: Quotations & Escape

- Quotation: Splice Terra code into other Terra code
- Escape: Splice result of a Lua expression into Terra code

```
local stdio =
  terralib.includec("stdio.h")

function doN(N, body)
    return quote
        for i = 0, N do
            body
        end -- for
    end -- quote
end -- function
```

```
terra doIt()
    [doN(10,
         `stdio.printf(
            "Hello World\n")
      )]
end

doIt()
```

Terra Example: Base 10 to Base K

- Take a base 10 integer
- Convert it to a string of digits for a basis K $\in$ [2, 9]

Terra Example: Base 10 to Base K

```
function toBase(base, n)
    if base < 2 then
        error("base must be > 1")
    end

    local digits = ""
    while n > 0 do
        digits = (n % base) .. digits
        n = math.floor(n / base)
    end

    return digits
end
```

- Implemented in Lua
- Pay for generality
- Easy to understand

Terra Example: Base 10 to Base K

```
local cstdlib = terralib.includec("stdlib.h")
local cmath = terralib.includec("math.h")
```

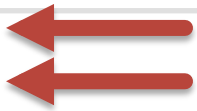
- cstdlib.calloc()
- cmath.floor()

Terra Example: Base 10 to Base K

```
local mymath = {}
for base = 2, 9 do
    mymath["toBase" .. base] = terra(n : int)
        var digitsNeeded = [int32](
            (cmath.log(n) / cmath.log(base)) + 1)
        var digits = [&int8](
            cstdlib.calloc(digitsNeeded + 1, sizeof(int8)))

        for index = 0, digitsNeeded do
            digits[digitsNeeded - 1 - index] = 48 + (n % base)
            n = [int](cmath.floor(n / base))
        end
        return digits
    end
end
```

Terra Example: Base 10 to Base K



```
local mymath = {}
for base = 2, 9 do
    mymath["toBase" .. base] = terra(n : int)
        var digitsNeeded = [int32](
            (cmath.log(n) / cmath.log(base)) + 1)
        var digits = [&int8](
            cstdlib.calloc(digitsNeeded + 1, sizeof(int8)))

        for index = 0, digitsNeeded do
            digits[digitsNeeded - 1 - index] = 48 + (n % base)
            n = [int](cmath.floor(n / base))
        end
        return digits
    end
end
```

Terra Example: Base 10 to Base K

```
local mymath = {}
for base = 2, 9 do
    mymath["toBase" .. base] = terra(n : int)
        var digitsNeeded = [int32](
            (cmath.log(n) / cmath.log(base)) + 1)
        var digits = [&int8](
            cstdlib.calloc(digitsNeeded + 1, sizeof(int8)))

        for index = 0, digitsNeeded do
            digits[digitsNeeded - 1 - index] = 48 + (n % base)
            n = [int](cmath.floor(n / base))
        end
        return digits
    end
end
```

Terra Example: Base 10 to Base K

```
local mymath = {}
for base = 2, 9 do
    mymath["toBase" .. base] = terra(n : int)
        var digitsNeeded = [int32](
            (cmath.log(n) / cmath.log(base)) + 1)
        var digits = [&int8](
            cstdlib.calloc(digitsNeeded + 1, sizeof(int8)))

        for index = 0, digitsNeeded do
            digits[digitsNeeded - 1 - index] = 48 + (n % base)
            n = [int](cmath.floor(n / base))
        end
        return digits
    end
end
```

Allocate char array for digits

Cast to int

Pointer arithmetic

Terra Example: Base 10 to Base K

```
local mymath = {}
for base = 2, 9 do
    mymath["toBase" .. base] = terra(n : int)
        var digitsNeeded = [int32](
            cmath.log(n) / cmath.log(base)) + 1)
        var digits = [&int8](
            cstdlib.calloc(digitsNeeded + 1, sizeof(int8)))

        for index = 0, digitsNeeded do
            digits[digitsNeeded - 1 - index] = 48 + (n % base )
            n = [int](cmath.floor(n / base ))
        end
        return digits
    end
end
```

Lua Variable declaration & assignment

captured Lua Variable: use current value

Terra Example: Base 10 to Base K

```
local mymath = {}
for base = 2, 9 do
    mymath["toBase" .. base] = terra(n : int)
        var digitsNeeded = [int32](
            (cmath.log(n) / cmath.log(base)) + 1)
        var digits = [&int8](
            cstdlib.calloc(digitsNeeded + 1, sizeof(int8)))

        for index = 0, digitsNeeded do
            digits[digitsNeeded - 1 - index] = 48 + (n % base)
            n = [int](cmath.floor(n / base))
        end
        return digits
    end
end
```

Terra Example: Base 10 to Base K

```
function toBase(base, n)
    -- create string of digits
end

local mymath = {}
for base = 2, 9 do
    mymath["toBase" .. base] = terra(n : int)
        -- create string of digits
    end
end

local luajit = require "ffi"
print(luajit.string(mymath.toBase5(535))) -- 4120
print(toBase(5, 535)) -- 4120
```

Lua

Terra

Lua