The Shape of Languages to Come
A (Brief) History of ME

Formed the company in 1982.

Previously teaching Engineering in Brighton.

Met the DEC PDP-11 and was hooked.

Over the years used between 20-30 different computer languages ..... 

..... but real passion is High Performance Computing (HPC).

Run a number of “meetup” groups.
Synopsis of the Talk

Taxonomy of programming languages.
What is this LLVM about anyway?
How quick is quick?
A demo using Julia
New Languages for Old
Q & A
Mastering Julia

Master your analytical and programming skills in Julia to solve complex data processing problems

Dr Malcolm Sherrington
Taxonomy of programming languages

A programmer’s perspective

By application area

Systemic viewpoint
Systemic viewpoint

Assembly languages
Compilers
Interpreters
Intermediate code
LLVM (Jittering)
Assembly Languages

Machine specific
1-to-1 correspondence between the computer architecture and the code
Typically two passes through the code
Used for on-board systems and operating system kernels
Still common with FPGA’s
1950’s Languages

- FORTRAN (1954)
- LISP (1957)
- ALGOL (1958)
- COBOL (1959)
- JOVIAL (1959)

*The computer language paradox*
Top 10 Programming Languages Currently Used in Projects (Jan 2014)

- Java
- C#
- PHP
- C++
- JavaScript
- Python
- Objective-C
- Ruby
- C
- PL-SQL
Most Popular Coding Languages of 2014

- Python: 30.3%
- Java: 22.2%
- C: 4.1%
- C#: 5%
- Javascript: 5.2%
- Ruby: 10.6%
- PHP: 3.3%
- Go: 1.5%
- C#: 1.6%
- Haskell: 1.2%
- Lua: 0.04%
- Clojure: 0.2%
- Bash: 0.1%
- TCL: 0.03%
- Objective C: 0.4%
The LLVM project started at UIUC in 2003

Main architect was Chris Lattner, now with Apple

Is a JIT-tering system which converts IR code to specific assembly/machine code

Currently supports: X86, X86-64, ARM, AArch64, Mips, SystemZ, PowerPC

http://www.llvm.org
LLVM processing chain
What systems are now using LLVM?

As a compiler:
  Clang, Swift, GNU, Haskell, Ruby
  LDC, Clasp, LLgo

As a “bolt-on” module / extension
  Python (Numba), Tcl

As a complete system
  Julia, Javascript (V8), LuaJIT, Rust
  SML, Pure

How Fast is Fast?

Regular derivative (stock option) pays out at the termination of the contract.

There is a formula based on the work of Black & Scholes in the 1970’s

A up or down turn at the end of the contract period can be disastrous

The Asian option uses an average price over the contract rather than the final one

This is computationally intense
“Simple” Asian Option
\[ S = \text{zeros}(\text{Float64}, N, T) \]
\[ A = \text{zeros}(\text{Float64}, N) \]

for \( n=1:N \)
\[
S[n,1] = S0
\]
\[
dW = \text{randn}(T) \times \sqrt{dt}
\]
for \( t=2:T \)
\[
z0 = (r - q - 0.5 \times v \times v) \times S[n, t-1] \times dt
\]
\[
z1 = v \times S[n, t-1] \times dW[t]
\]
\[
z2 = 0.5 \times v \times v \times S[n, t-1] \times dW[t] \times dW[t]
\]
\[
S[n, t] = S[n, t-1] + z0 + z1 + z2
\]
end
\[
A[n] = \text{mean}(S[n, :])
\]
end

\[ P = \text{zeros}(\text{Float64}, N) \]
\[ P[n] = \max(A[n] - K, 0) \text{ for } n = 1:N \]
price = \( \text{exp}(-r \times tma) \times \text{mean}(P) \)
Asian Benchmarks

Results for 100,000 runs of 100 steps, \((c \sim 0.73\ s)\)

<table>
<thead>
<tr>
<th>Language</th>
<th>Timing ((c = 1))</th>
<th>Asian Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1.0</td>
<td>1.681</td>
</tr>
<tr>
<td>julia</td>
<td>1.41</td>
<td>1.680</td>
</tr>
<tr>
<td>python (v3)</td>
<td>32.67</td>
<td>1.671</td>
</tr>
<tr>
<td>R</td>
<td>154.3</td>
<td>1.646</td>
</tr>
<tr>
<td>Octave</td>
<td>789.3</td>
<td>1.632</td>
</tr>
</tbody>
</table>

Samsung RV711 laptop with an i5 processor and 4Gb RAM running Centos 6.5 (Final)
Once established programming languages are long lived but the time at the top is short!

Advances in hardware and software may new approaches possible.

Emergence of new environments such as cloud computing.

Changes in application areas, such as big data and mobile apps.

Shift to analyst coding rather than specialist programmers.
The Two Language Problem?

Because of this dichotomy, a two-tier compromise is standard:

- for **convenience**, use a scripting language (Matlab, R, Python)
- but do all the **hard stuff** in a systems language (C, C++, Fortran)

Pragmatic for many applications, but has drawbacks

- aren’t the **hard parts** exactly where you need an **easier** language?
- forces **vectorization** everywhere, even when awkward or wasteful
- creates a **social barrier** – a wall between users and developers
Quo Vadis LLVM?
http://llvm.org

- Dragonegg (code generation for GCC)
- LLDB (native debugger)
- Libc++ ABI (including C++11)
- OpenMPI support
- Vmkit (JVM and .NET)
- Libclc (OpenCL)
- Kee (intelligent bug finder)
The Genie is out of the bottle, things will not be the same again.

All (new) major languages in the last five years incorporate some form of JIT-tering.

LLVM project is the main O/S and has an exciting agenda planned.

Parallelism and HPC is becoming increasing important.

Never again will languages as slow as Matlab, R and (even) Python be created and certainly not paid for.
thank you