Spectrum of languages

Ralf W. Grosse-Kunstleve
Computational Crystallography Initiative
Lawrence Berkeley National Laboratory

Spectrum of implementation languages

• Python
  – Interpreted, Object Oriented, Exception handling
• C++
  – Compiled, Object Oriented, Exception handling
• C
  – Compiled, User defined data types, Dynamic memory management
• Fortran
  – Compiled, Some high-level data types (N-dim arrays, complex numbers)
• Assembler
  – Computer program is needed to translate to machine code
• Machine code
  – Directly executed by the CPU

Matrix of language properties

<table>
<thead>
<tr>
<th></th>
<th>Dynamically typed -&gt; convenience</th>
<th>Statically typed -&gt; speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted</td>
<td>Python</td>
<td>Java</td>
</tr>
<tr>
<td>Compiled to machine code</td>
<td>Psyco</td>
<td>C++</td>
</tr>
</tbody>
</table>

Programmer Efficiency & Performance

- Maintainability
- Reusability
- Modularity
- Performance
- Programmer Efficiency

Choice of implementation languages

• Python
  – Very high-level programming
  – Easy to use (dynamic typing)
  – Fast development cycle (no compilation required)
  – Too slow for certain tasks

• C++
  – High-level or medium-level programming
  – Many arcane details (strong static typing)
  – Largely automatic dynamic memory management (templates)
  – Much faster than Python
  – With enough attention, performance even rivals that of FORTRAN

Happy marriage: Python and C++

• Syntactic differences put aside, Python and C++ objects and functions are very similar.
• Flexibility (interpreted, dynamically typed) and Efficiency (compiled, statically typed) are complementary.
• Boost.Python (C++ library) provides the link:
  – Non-intrusive on the C++ design
  – Pseudo-automatic wrapping using C++ template techniques
  – No external tools needed
  – Creates sub-classable Python types
  – Python bindings are very maintainable
  – Tutorial and reference documentation

```cpp
class_<unit_cell>("unit_cell")
  .def("volume", &unit_cell::volume)
  .def("fractionalize", &unit_cell::fractionalize)
;```
Vector operations

- Computer Science wisdom:
  - Typically 90% of the time is spent in 10% of the code
- Similar to idea behind vector computers:
  - Python = Scalar Unit
  - C++ = Vector Unit
- Loading the vector unit: (8.7 seconds)
  ```python
  miller_indices = flex.miller_index()
  for h in xrange(100):
    for k in xrange(100):
      for l in xrange(100):
        miller_indices.append((h,k,l))
  ```
- Go! (0.65 seconds)
  ```python
  space_group = sgtbx.space_group_info("P 41 21 2").group()
  epsilons = space_group.epsilon(miller_indices)
  ```
  Computing 1 million epsilons takes only 0.65 seconds!

Compiled vs. Interpreted

- Compiler
  - generates fast machine code
- Interpreter (Python, Perl, TCL/TK, Java)
  - may generate byte-code but not machine code

Compiled vs. Interpreted

- Compiler
  - generates fast machine code
  - needs arcane compilation commands
  - needs arcane link commands
  - generates object files (where?)
  - may generate a template repository (where?)
  - generates libraries (where?)
  - generates executables (where?)
  - all this is platform dependent
- Interpreter (Python, Perl, TCL/TK, Java)
  - may generate byte-code but not machine code

Conclusion languages

- It is important to know the modern concepts
  - Especially for ambitious projects
- Syntax is secondary
  - Anything that does the job is acceptable
  - Python, C++, csh, sh, bat, Perl, Java
- There is no one size fits all solution
  - But Python & C++ covers the entire spectrum
- Carefully weigh programmer efficiency vs. runtime efficiency
  - Prefer a scripting language unless runtime efficiency is essential

Acknowledgements

- Organizers of this meeting
- Paul Adams
- Pavel Afonine
- Peter Zwart
- Nick Sauter
- Nigel Moriarty
- Erik McKee
- Kevin Cowtan
- David Abrahams
- Open source community

http://www.phenix-online.org/ http://cctbx.sourceforge.net/