Extending the cppyy support in Numba

Aaron Jomy
INTRODUCTION

- **Cppyy**: An automatic, run-time, Python-C++ bindings generator

- **Cling**: is used in backend since an interactive C++ interpreter provides a runtime exec approach to C++ code

- **Numba**: JIT compiler that translates Python and NumPy code into fast machine code.

WHY USE NUMBA?

- The compute time overhead while switching between languages accumulates in loops with cppyy objects.

- Numba optimizes the loop and compiles it into machine code which crosses the language barrier only once
Numba Pipeline

- **Typing**
  Numba core has a type inference algorithm which assigns a nb_type for a variable

- **Lowering**
  high-level Python operations into low-level LLVM code.
  Exploits typing to map to LLVM type

- **Boxing and unboxing**
  convert PyObject* 's into native values, and vice-versa.

We utilise the runtime numba compilation process to lower C++ code cppdef’ed in Python

How? →
• Typing is one of the largest problems posed: Template function utilization, reference types and correct function matching depend on the type resolution system.

• Type Inference solution:
  A mechanism to handle implicit casting based on propagated type info and the cppyy reflection layer.

• Note: Typing does not backtrack since the numba extension will only ever obtain the numba type inference result.

<table>
<thead>
<tr>
<th>Python</th>
<th>Numba Type</th>
<th>LLVM Type used in Numba lowering</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 (int)</td>
<td>int64</td>
<td>i64</td>
</tr>
<tr>
<td>3.14 (float)</td>
<td>float64</td>
<td>double</td>
</tr>
<tr>
<td>(1, 2, 3)</td>
<td>UniTuple(int64, 3)</td>
<td>[3 x i64]</td>
</tr>
<tr>
<td>(1, 2.5)</td>
<td>Tuple(int64, float64)</td>
<td>{i64, double}</td>
</tr>
<tr>
<td>np.array([1, 2], dtype=np.int32)</td>
<td>array(int64, 1d, C)</td>
<td>{i8*, i8*, i64, i64, i32*, [1 x i64]}</td>
</tr>
<tr>
<td>“Hello”</td>
<td>unicode_type</td>
<td>{i8*, i64, i32, i32, i64, i8*, i8*}</td>
</tr>
</tbody>
</table>
Primary Deliverables:

- Add general support for C++ templates in Numba through Cppyy
- Add support for C++ reference types in Numba through Cppyy
Some Examples

```
def ref_test():
    cppy.cppdef(""
    int64_t& ref_add_8(int64_t x) {
        static int64_t result = x+8;
        return result;
    }
    "")
@numba.njit()
def run_add(a):
    k = cppy.gb1.ref_add_8(a)
    result = k[0]
    return result
x = 17
    print("Result of ref_add_8", run_add(x))

Matched CPyCppyy Signature
2:(int64_t)
Reference return type detected
Performing lowering

Obtaining the function __overload__
in get_pointer:
Matched CPyCppyy Signature
2:(int64_t)
Successful arg combo match in
get_pointer= ('int64_t',)
Result of ref_add_8: 25
```

cppy.cppdef(""
namespace NumbaSupportExample{
    template <typename T, typename U>
    T multiply(T t, U u) { return t * u; }
}
")""
SOME EXAMPLES

cppyy.cppdef(""
namespace NumbaSupportExample{
    template <typename T>
    T sum(const T &container)
    {
        T total = T(0);
        for (const T &value : container)
        {
            total += value;
        }
        return total;
    }
}")

cppyy.cppdef(""
namespace NumbaSupportExample{
    template <typename T, int N>
    T power(T t)
    {
        T result = 1;
        for (int i = 0; i < N; ++i)
            result *= t;
        return result;
    }
}")

Template template parameters

Non-type template parameters
Thank You!