PLANNED IMPROVEMENTS TO THE CHAPEL COMPILER

Michael Ferguson, HPE
June 4, 2021
COMPILER IMPROVEMENTS:
OUTLINE

• Where We Are
• Missing Features
• Proposed Direction
• Progress Report
WHERE WE ARE
PROBLEMS WITH THE CURRENT COMPILER

• **Speed**
  - The current compiler is generally slow, and extremely so for large programs (~7s to 15 minutes)

• **Structure and Program Representation**
  - The compiler is structured only for whole-program analysis, preventing separate/incremental compilation
  - Unclear how to integrate an interpreter, provide IDE support, or ‘eval’ Chapel snippets

• **Development**
  - The modularity of the compiler implementation needs improvement
  - There is a steep learning curve to become familiar with the compiler implementation
WHAT IS WORKING WELL

• The current compiler code base has enabled the design and evolution of Chapel to date
• The compiler includes key optimizations that support program performance
• The compiler is relatively fast to build and very portable
• Chapel’s internal and library modules are extensive and largely independent of the compiler
• The runtime libraries are well-architected and not in need of major changes
## SUMMARY OF CURRENT COMPILER PASSES

<table>
<thead>
<tr>
<th>pass</th>
<th>time for Hello World</th>
<th>time for Arkouda</th>
<th>approx lines of .cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse+</td>
<td>0.5s</td>
<td>0.8s</td>
<td>10,000 lines</td>
</tr>
<tr>
<td>scopeResolve</td>
<td>0.4s</td>
<td>0.7s</td>
<td>4,500 lines</td>
</tr>
<tr>
<td>normalize+</td>
<td>0.9s</td>
<td>2.2s</td>
<td>9,000 lines</td>
</tr>
<tr>
<td>resolve</td>
<td>2.0s</td>
<td>165s</td>
<td>35,000 lines</td>
</tr>
<tr>
<td>post-resolve</td>
<td>0.3s</td>
<td>16s</td>
<td>16,000 lines</td>
</tr>
<tr>
<td>lowerIterators</td>
<td>0.1s</td>
<td>7.1s</td>
<td>6,000 lines</td>
</tr>
<tr>
<td>parallel</td>
<td>0.1s</td>
<td>17s</td>
<td>2,000 lines</td>
</tr>
<tr>
<td>optimization</td>
<td>0.5s</td>
<td>46s</td>
<td>7,000 lines</td>
</tr>
<tr>
<td>insertWideRefs+</td>
<td>0.1s</td>
<td>15s</td>
<td>6,000 lines</td>
</tr>
<tr>
<td>codegen</td>
<td>0.5s</td>
<td>48s</td>
<td>20,000 lines</td>
</tr>
<tr>
<td>makeBinary</td>
<td>1.1s</td>
<td>422s</td>
<td>-</td>
</tr>
<tr>
<td><strong>TOTAL OF ABOVE</strong></td>
<td>6.0s</td>
<td>724s</td>
<td>114,000 lines</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>6.4s</td>
<td>743s</td>
<td>170,000 lines in ‘compiler/*’</td>
</tr>
</tbody>
</table>
MISSING FEATURES
MISSING FEATURES

- Some commonly requested features:
  - IDE support (beyond syntax highlighting)
  - Separate and/or incremental compilation

- Good IDE integration requires the compiler to
  - behave more like a server
  - respond quickly to limited queries (e.g. code completion or mouse-over)

- Incremental compilation and separate compilation require
  - a more flexible compiler that can instantiate some generics but re-use other instantiations

- Compiler architecture improvements can make these problems easier to solve
PROPOSED DIRECTION
At a very high level, this design is similar to other compilers:
- Swift, Rust, Julia, Flang use the pattern:
  - parse → AST → mid-level IR (and progressively lower)
- Earliest part of compilation is most relevant for IDE integration
  - so is designed to be very incremental
CHANGES TO GET TO IMPROVED ARCHITECTURE

- Create a new AST more faithful to source code for early passes

- Develop a new pass architecture with less rigid ordering
  - make passes typically run per-function rather than whole-program and otherwise be idempotent

- Convert the new AST into the old AST to enable incremental development

- Gradually port later passes over to a new IR more suited for optimization
Matsakis’ talk, Responsive Compilers\(^1\), presents a vision for good IDE support:

- highly incremental and demand-driven—just process enough to answer a query
  - e.g., how to complete newBlock<tab>
  - fast response times are key for a satisfying experience

The strategy relies on:

- structuring compilation in terms of many fine-grained queries
  - e.g., what is the type of this variable?
- framework uses these queries to manage dependencies among results
- each query saves its result and is re-run when necessary
- query results are represented separately from the input—which tends to mean a lot of maps
- AST elements are given IDs to support these maps
RESPONSIVE COMPILER QUERIES

- `parse(filePath) → AST for file (which also establishes IDs)`
- `locate(AST) → (line number, column number)`

- `getDefinedIn(Expr, name) → Symbols defined in ‘Expr’ named ‘name’`
- `getVisible(Expr, name) → Symbols visible from ‘Expr’ named ‘name’`
- `types(Expr) → map from Symbols to Types for Symbols defined in ‘Expr’`
- `resolve(Expr) → map from Identifiers to Symbols they refer to`
FUTURE COMPILER ARCHITECTURE

source \(\rightarrow\) parse \(\rightarrow\) uAST \(\rightarrow\) resolve \(\rightarrow\) uAST + maps of type & named symbol \(\rightarrow\) lower \(\rightarrow\) oAST (future: MLIR)

“Responsive Compiler”

“Mid Level IR”

progressive lowering, mostly function-at-a-time
PROGRESS REPORT
Developing new code as a library

New uAST nodes have documentation!

These features will enable community members to contribute Chapel tools

Library Use Cases:
- Linter
- Documentation tools
- IDE integrations
IMPLEMENTATION PROGRESS

- Part-way through implementing the new uAST, parsing it, and translating it into the old AST
- Have demonstrated incremental re-compilation with simple examples

```chpl
// mymodule.chpl

module M {  
    proc f() {  
        writeln();  
    }  
}
```

```
prompt % testInteractive mymodule.chpl

mymodule.chpl:3: error: 'writeln' undeclared (first use this function)
Module M:
Module 0x7fc962406250 M
    Function M 0x7fc962406140 f
        FnCall M.f@1 0x7fc9624060a0
        Identifier M.f@0 0x7fc962406020 writeln

Would you like to incrementally parse again? [Y]:
```
QUESTIONS?

source → parse → uAST → resolve → uAST + maps of type & named symbol → lower → oAST (future: MLIR)

“Responsive Compiler”

“Mid Level IR”

progressive lowering, mostly function-at-a-time
THANK YOU

https://chapel-lang.org
@ChapelLanguage