Clang requires ~100 patches to clang’s incremental compilation facilities. For example, CodeGen was modified to work with multiple llvm::Module instances, and to finalize per each end-of-translation unit (clang has more than one). Tweaks to the FileManager’s caching mechanism, and improvements to the SourceManager virtual and overridden files (code reached mostly from within clang’s setup) were necessary.

Our research shows that the **clang infrastructure works amazingly well** to support something which was not its main use case. The grand total of our diffs against clang-9 is: 62 files changed, 1294 insertions(+), 231 deletions(-).

A major weakness of clang’s infrastructure is that it does not work with the clang Action infrastructure due to the lack of an IncrementalAction.

**Incremental Action** allows constant compilation of partial inputs and ensures that the compiler remains active. It includes an API to access attributes of recently compiled chunks of code that can be post-processed. The REPL orchestrates existing LLVM and Clang infrastructure with a similar data flow:

---

**Incremental Compilation Support in Clang**

**Vassil Vassilev and David Lange**

Princeton University

---

### Objectives

Incremental compilation aims to support clients that need to keep a single compiler instance active across multiple compile requests. Our work focuses on:

- **Enhancement** – upstream incremental compilation extensions available in forks;
- **Generalization** – make building tools using incremental compilation easier;
- **Sustainability** – move incremental use cases upstream.

### Incremental Compilation Support in Clang

- **Background**
  - Cling requires ~100 patches to clang’s incremental compilation facilities. For example, CodeGen was modified to work with multiple llvm::Module instances, and to finalize per each end-of-translation unit (clang has more than one). Tweaks to the FileManager’s caching mechanism, and improvements to the SourceManager virtual and overridden files (code reached mostly from within clang’s setup) were necessary.
  - Our research shows that the **clang infrastructure works amazingly well** to support something which was not its main use case. The grand total of our diffs against clang-9 is: 62 files changed, 1294 insertions(+), 231 deletions(-).
  - A major weakness of clang’s infrastructure is that it does not work with the clang Action infrastructure due to the lack of an IncrementalAction.

- **Introduction**
  - Over the last decade an interactive, interpretative, C++ (aka REPL) platform has developed as part of the high-energy physics data analysis project – ROOT [1, 2]. This REPL, cling [3] supports data analysis of exabytes of particle physics data coming from the Large Hadron Collider (LHC) and other particle physics experiments.
  - Cling is a C++ interpreter built on top of clang and LLVM. In a nutshell, it uses clang’s incremental compilation facilities to process code chunk-by-chunk by assuming an ever-growing translation unit [3]. Code is lowered into LLVM’s IR and run by the LLVM JIT. Cling implements language “extensions” such as execution statements on the global scope and error recovery.
  - Cling is also available as a standalone tool, with a growing community, including users in finance, biology and in a few companies with proprietary software. For example, there is a cling jupyter kernel [4]. One of the major challenges we face to foster this community is the set of cling-related patches to its LLVM and clang forks.
  - The benefit of relying more heavily on the LLVM community with a similar data flow: the compiler remains active. It includes an API to access attributes of recently compiled chunks of code that can be post-processed. The REPL orchestrates existing LLVM and Clang infrastructure with a similar data flow.

- **High Level Tool Design**
  - **Incremental Action** allows constant compilation of partial inputs and ensures that the compiler remains active. It includes an API to access attributes of recently compiled chunks of code that can be post-processed. The REPL orchestrates existing LLVM and Clang infrastructure with a similar data flow.

- **Conclusion**
  - Over the years it has become evident that incremental processing of C++ has its users not only in the science domain but also industry. The work underway aims at making the incremental C++ facilities in clang more sustainable and easier to build tools.
  - The advancement of the interpretative technology for C++ in cling will be a basis for our contributions to the LLVM community. The development of clients based on clang incremental C++ and improved language interoperability will make C++ easier to use and friendlier to integrate in the trend Notebook-style programming.

### References

[4] Xeus is now a Jupyter subproject. https://blog.jupyter.org/xeus-is-now-a-jupyter-subproject-6e6e85f830b

Acknowledgements

This project is supported by National Science Foundation under Cooperative Agreement OAC-1931488. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.