Scout: Using Clang/LLVM to Build a Domain-Specific Language For In-Situ Data Analysis and Visualization on Emerging Architectures

Patrick McCormick*, James Jablinski*, Nick Moss*, Christine Ahrens*, Dean Prichard*, Marion (Kei) Davis*
Los Alamos National Laboratory, “Brown University

Objective of Scout Project

Today's large-scale scientific applications must be able to run on rapidly changing processor architectures and require computation, data analysis and visualization of increasingly large amounts of data.

The purpose of the Scout project is to explore building a domain-specific programming language and development toolchain that can support existing scientific applications on emerging architectures without having to significantly rewrite or refactor code.

In situ Scout code can do computation, numerical or visual analysis on the data without storing it to file or post-processing.

The Scout Language

Scout's conservative extensions to C/C++ currently provide:

- A computational mesh abstraction that supports 1, 2 and 3D-mesh elements including mesh members (fields) for cells, vertices and points.
- Parallel for all over meshes or arrays for general processing.
- Parallel render all over meshes for visualization of 2 or 3D datasets.
- Access to mesh element neighbors via cshift operation.
- Filtering ability for parallel constructs via where clause.
- Support for two-, three- and four-component vector types.
- Stand-alone or in situ Scout programs.
- Parallel constructs running on single or multiple CPU cores or GPU.

How Scout Uses Clang/LLVM

- Scout language constructs (conservative extensions to C/C++) are added to the Clang lexer and are parsed into an AST representation containing Clang tree nodes and domain-specific tree nodes.
- When the AST is lowered to LLVM IR the domain-specific regions of code are translated to various constructs:
  - Mesh declarations are translated into specialized data structures and mesh element setters and getters are created.
  - For all and 2D mesh render all constructs are handled by creating a closure for the body of the construct. The runtime library places them in a work-stealing queue in the multi-core case.
  - 3D mesh render all constructs become closures that are passed to a runtime library volume renderer. Rewriter is used for this.
  - If compiling for GPU, GPU kernel metadata is saved for later use.
- LLVM passes handle GPU kernels and enable vectorization for CPU.

Conclusions

Using Clang/LLVM for implementing Scout provides/allows:

- Code generation built-in for different targets
- Reduced learning curve and overall development time
- Future integration of LLVM toolchain elements such as LLDB and JIT
- More focus on research rather than building compiler from scratch

Would like to be able to modify the Clang AST for the implementation of some Scout constructs, but can use rewriter.

Acknowledgments

This work was supported in full by the DOE Office of Science, Advanced Scientific Computing Research, program manager Lucy Nowell.