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

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

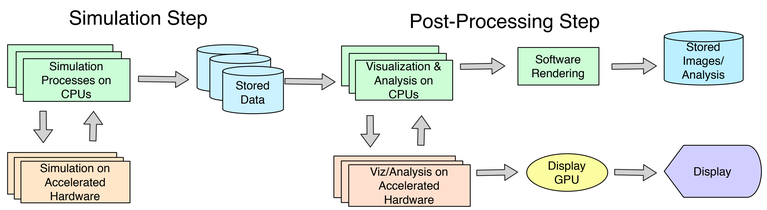
## Objective of Scout Project

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.

## Traditional Scientific Workflow




## In Situ Scientific Workflow

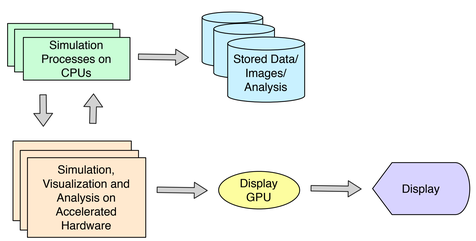



Figure 1. Post-processing model vs. In Situ via Scout

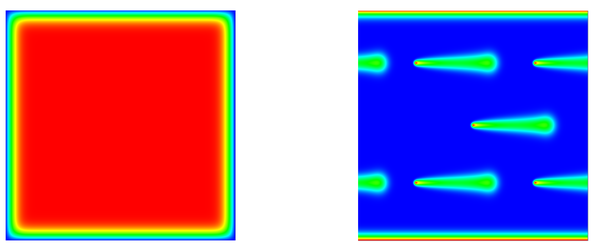


Figure 2. Heat transfer applications using Scout run on CPU, multicore or GPU

## 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 `forall` over meshes or arrays for general processing.
- Parallel `renderall` 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.

```
// Declare a uniform mesh with fields stored
// at cell centers and mesh vertices.
uniform mesh UniMesh {
 cells    : float temp, rho;
 vertices: float3  velocity;
};

// Define a 2D instance of the mesh.
UniMesh umesh[256,256];

// Initialize all mesh fields to zero.
forall cells c of umesh
  c.temp = c.rho = 0.0;
forall vertices v of umesh
  v.velocity = 0.0;
```

Figure 3. Declare, define and initialize uniform mesh.

```
// Render every cell of in the mesh 'umesh'
// assigning a color to each cell.
renderall cells c of umesh {

  // Map temperature into hue range of
  // blue (cold) to red (hot).
  float hue = 240.0 -
     (240.0 * (c.temp/MAX_TEMP));

  // Assign color to the current cell (c)
  // by converting hue-saturation-value
  // color to the built-in rgba 'color'.
  color = hsv(hue, 1.0, 1.0);
}
```

Figure 4. Use renderall to produce an image of the mesh cells.

## 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.
  - Forall and 2D mesh renderall 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 renderall 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.

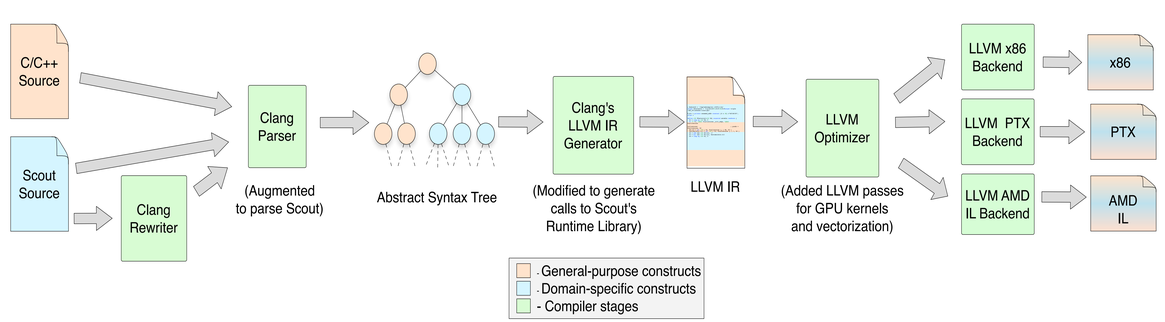



Figure 5. Scout's augmented Clang/LLVM compiler.

## 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.

LA-UR-12-25846