Semantic-Driven Parallelization of Loops Operating on User-Defined Containers

Dan Quinlan, Markus Schordan, Qing Yi

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory, under contract No. W-7405-Eng-48

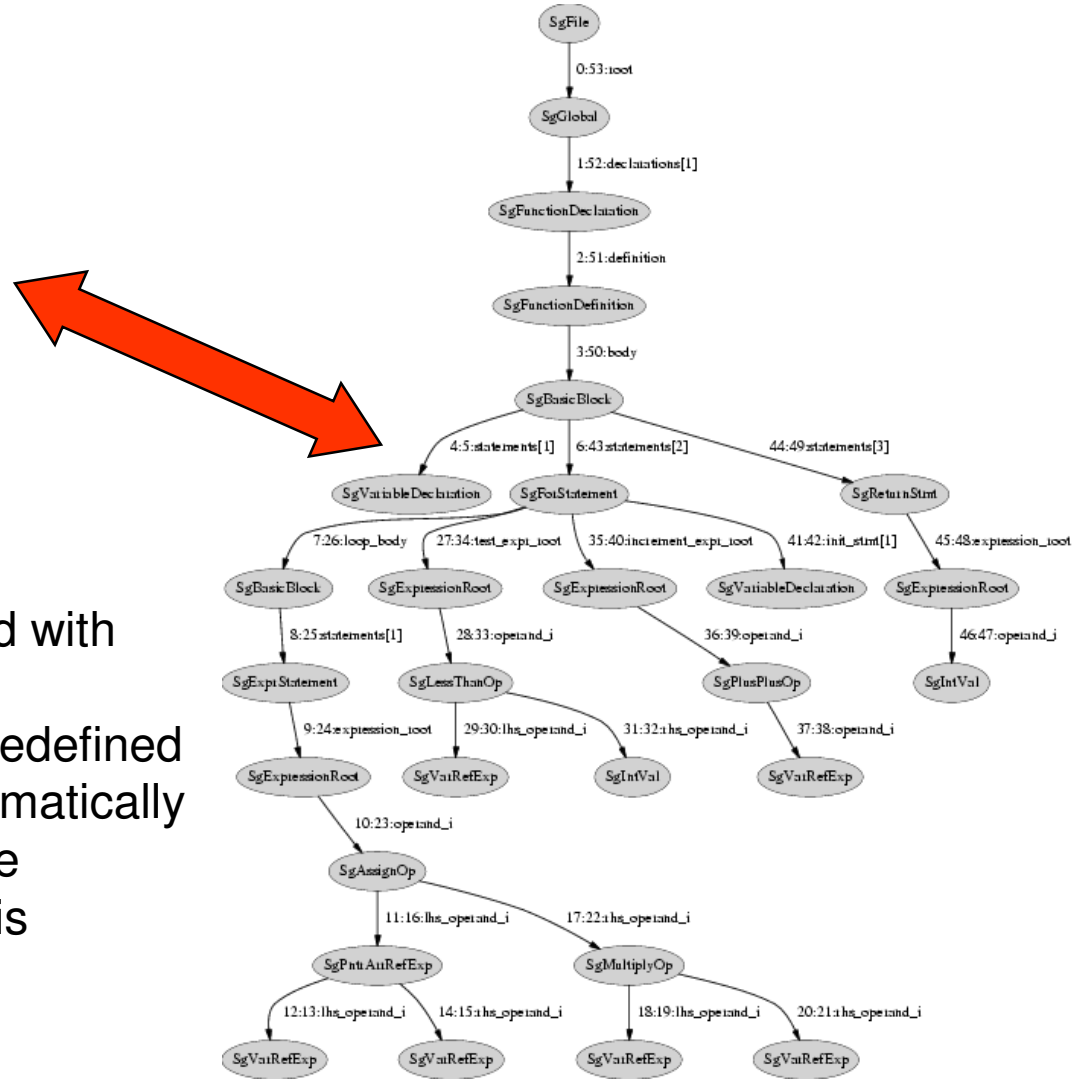
Outline of Talk

- **ROSE** Architecture
- AST Restructure Operations
- Building Domain Specific Languages from Libraries
- User-Defined Container Optimizations
  - Annotations
  - Analysis
- Example
- Related Work
- Conclusions

ROSE: Tool to Build Translators

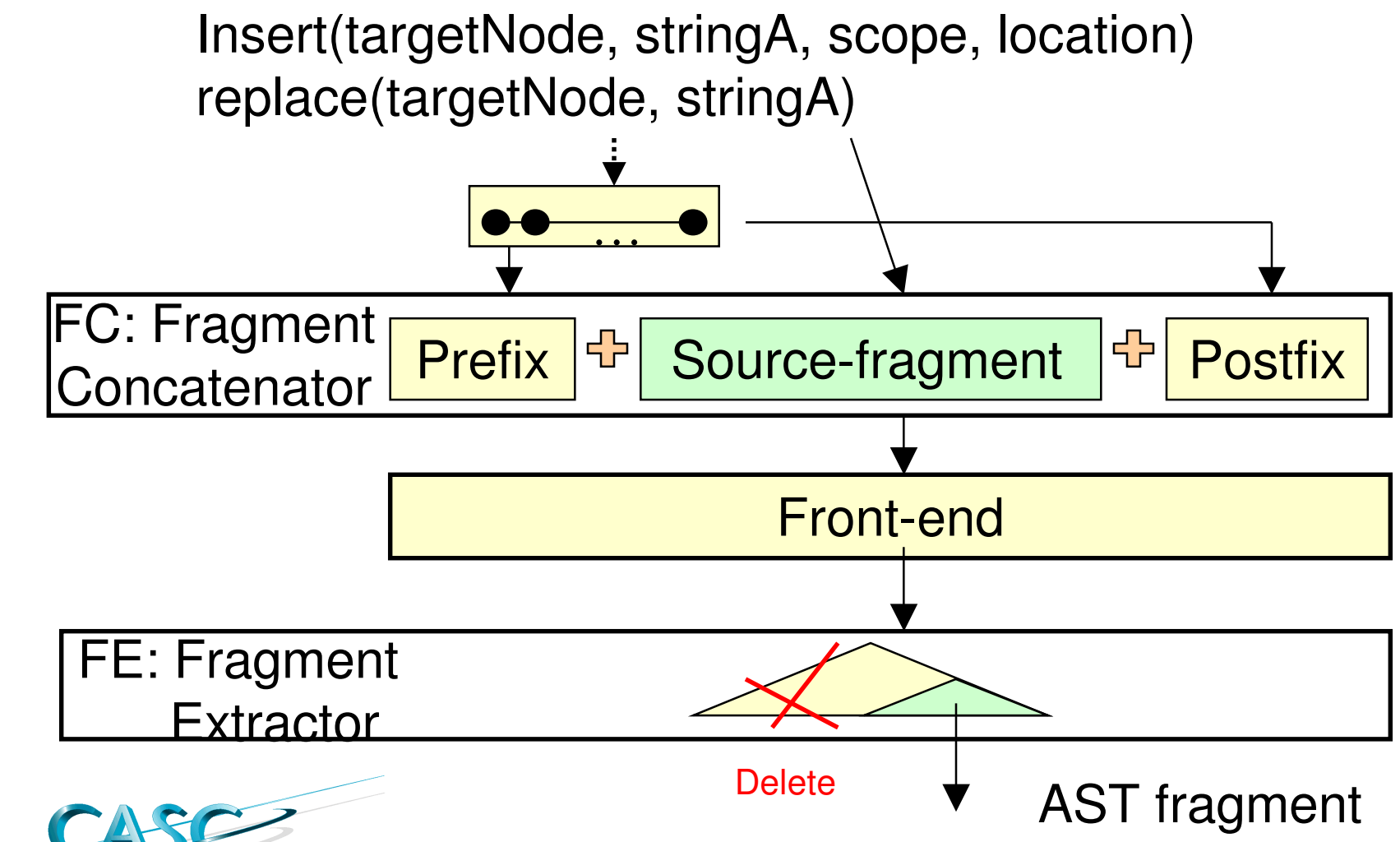
```
int main() {
  int a[10];

  for(int i=0;i<10;i++)
    a[i]=i*i;
  return 0;
}
```



- ROSE AST Features:
  - AST Query mechanisms
  - AST Rewrite mechanisms
  - Semantic actions associated with grammar rules
  - Abstract C++ grammar is predefined
  - Higher level grammars automatically generated from library source
  - Database for Global Analysis
  - Source code generation

String-Based Rewrite of AST



Domain Specific Languages

- Programming Languages exist to make programming productive
  - General purpose languages are *only generally* productive
  - Domain Specific Languages are *much more* productive
    - if used in their domain
    - if well designed
- Compilers have a role in automating the generation of Domain Specific Languages
- Observations:
  - Many Languages Support Abstractions
  - Libraries Define Abstractions
  - No Mechanism to Communicate *Semantics* of Abstractions

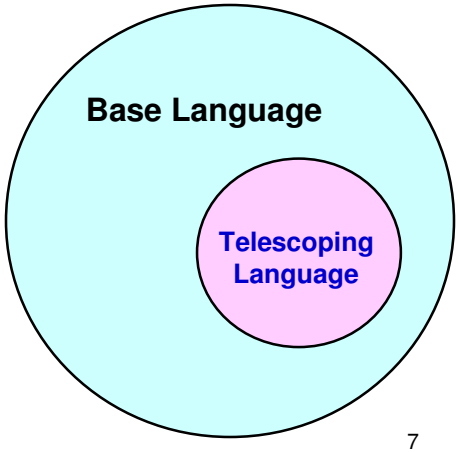
Constructive Definitions

- **Definition: Base Language is the language used to implement all libraries and the target applications that use them.**
- **Definition: Abstract Grammar is the set of product rules minus those specific to any syntax. I will use Abstract Grammar and Grammar interchangeably**
- **Definition: Domain Specific Grammar is the abstract grammar of the base language *plus* product rules specific to abstractions.**

- **Trick Question: What is a Domain Specific Language without a corresponding domain specific grammar?**

Domain Specific Languages

- Observations about this Domain Specific Grammar:
  - 1) We have defined a domain specific language
  - 2) Can't add syntax (no new keywords)
  - 3) Domain Specific Language build inside of base language
  - 4) Not a language Extension
    - Application and libraries are still written in the base language
  - 5) Parser front-end need not be modified (still use EDG)
  - 6) Process can be fully automated by processing the library's definition (found in it's header files)
- Language --> Grammar
  - Domain Specific Languages have a Domain Specific Grammar
  - Telescoping Languages have a *Telescoping Grammar*



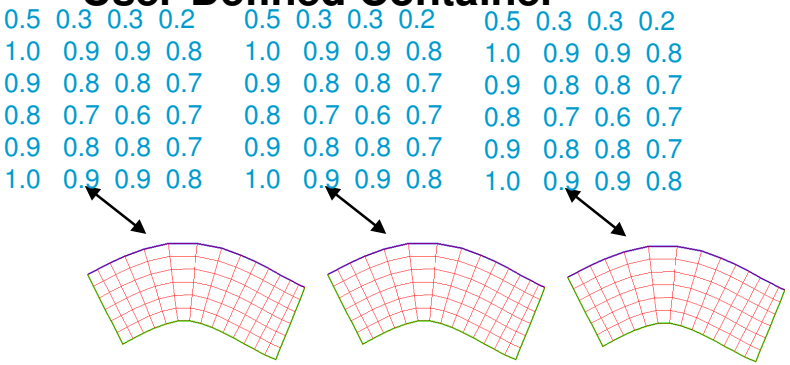
What Good is a Telescoping Grammar?

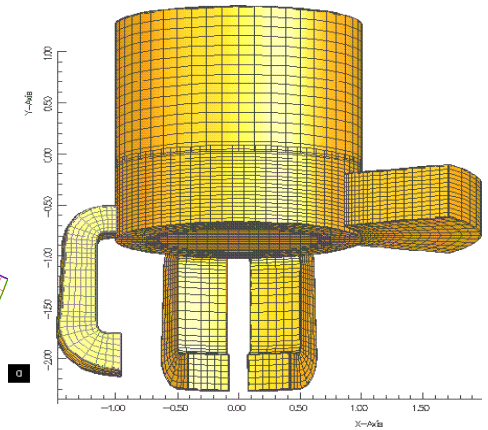
- **Can be automatically generated**
- **Don't extend the syntax of the base language**
- **Use same tools as base language**
- **Helps recognize abstractions at compile-time**
- **Simplifies transformations**
  - **High degree of resolution of language/library abstractions**
  - **Attach semantic actions to corresponding attribute grammars.**

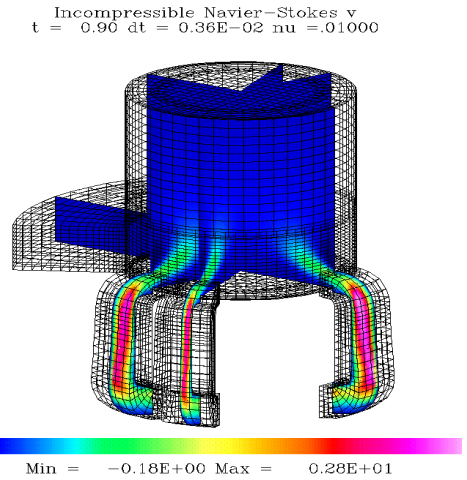
Equation: $u_{new} = u - \delta t \ ((u \bullet \nabla)u - \nu \Delta u)$

Code: uNew = u - dt*(u.convectiveDerivative() - nu*u.laplacian());

User-Defined Container







A++ Library: ~80 operators, **performance penalty: 3.6**

Simple Motivating Example

- Example Code:

```
Foo f; list mycontainer;
…
for (list::iterator i = mycontainer.begin(); i != mycontainer.end(); i++)
      f.foobar(*i);
```

- Can be parallel if:
  - *f.foobar* is safe (see safety analysis)
  - elements of *list* are not aliased
- After Transformation

```
Foo f; list mycontainer;
...
SupportingOmpContainer_list mycontainer2 (mycontainer);
#pragma omp parallel for
for (int i=0; i < mycontainer2.size(); i++)
      f.foobar(mycontainer2[i]);
```

Attached Semantic Action (Coco)

- Transformation (SgScopeStatement rule)

```
SgScopeStatement<unsigned int forNestingLevel>
  = SgForStatement
(.
   bool isOmpForQualified =
      ompTransUtil.isUserDefIteratorForStatement(astNode,forNestingLevel);
.)
"(" SgForInitStatementNT<forNestingLevel> SgExpressionRootNT
    SgExpressionRootNT SgBasicBlockNT<forNestingLevel+1>
")"
(.
    if (isOmpForQualified) {
       string iVarName = query.iteratorVariableName(astNode);
       string iContName = query.iteratorContainerName(astNode);
       string iContType = query.iteratorContainerType(astNode,iContName);
       string parTypeName = ompTransUtil.supportingParType(astNode,iContType);
       string parContName = ompTransUtil.uniqueVarName(astNode,iContName);
       string modifiedBodyString = ompTransUtil.derefToIndexBody(astNode,iVarName,iContName);
       string support = parTypeName+" "+parContName+"("+iContName+");\n";
       string beforeForStmt = "#pragma omp parallel for\n";
       string newForStmt = "for(int "+iVarName+"=0;"
                                     +iVarName+"<"+parContName+".size();"
                                     +iVarName+"++)"+modifiedBodyString;
       string subst.replace(astNode,support+beforeForStmt+newForStmt);
    }
.)
| ...
```

Evaluation of Inherited Attribute

Evaluation of Synthesized Attribute

Call AST Rewrite Mechanism

Transformation of Motivating Example

```
Foo f; list mycontainer;
...
SupportingOmpContainer_list mycontainer2 (mycontainer);
#pragma omp parallel for
for (int i=0; i < mycontainer2.size(); i++)
    f.foobar(mycontainer2[i]);
```

Annotation & Safety Analysis

- Summary of annotation Mechanism
  - Specification of qualified container classes (by type name)
  - Specification of functions side-effects
    - Function name
    - Names of modified global variables
    - Names of modified input parameters
- Summary of algorithm for safety analysis of parallelization
  - Check form of loop header:
    - *for (container::iterator p = l.begin(); p != l.end(); p++)*
  - Find modified variables in loop body
    - Check if locally declared or (*p)
  - Find function calls
    - Check if modifies global variables
    - Find modified arguments
      - Check if locally declared or (*p)

Related Work

- Large body of autoparallelizing Fortran compilers:
  - Dsystem, Fx, Vienna Fortran Compiler, Paradigm, Polaris, SUIF (F77, C, C++)
- Parallel STL like containers
  - STAPL (parallel STL library)
  - Parallel Standard Library
- Broadway Compiler (C compiler)
  - More sophisticated annotation language
- Telescoping Language work at RICE

Conclusions and Future Work

- Compile-Time optimization of a library abstraction
- Semantic-Driven parallelization of iterations over user-defined container similar to STL
- One possible approach to parallel STL?
- Demonstrates transformation specific annotation mechanism (not as powerful as that within the Broadway Compiler)
- Future Work:
  - More sophisticated annotation mechanism for function side-effects
    - automated support
    - combined with support in ROSE for global analysis
  - Specification of semantics
  - Less conservative, more precise analysis
  - Optimize performance of object-oriented scientific libraries