Lawrence Livermore National Laboratory

Towards an Abstraction-Friendly Programming Model for High Productivity and High Performance Computing

Chunhua "Leo" Liao, Daniel J. Quinlan and Thomas Panas

Center for Applied Scientific Computing

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

LLNL-PRES-417743

Programming and high level abstractions

- Best practice in software engineering:
  - High level abstractions → high productivity
    - expose interface, hide implementation details
    - + code reuse, - software complexity
  - Standard or customized
    - C++ STL containers, algorithms, iterators, …
    - User-defined classes, functions, libraries, …

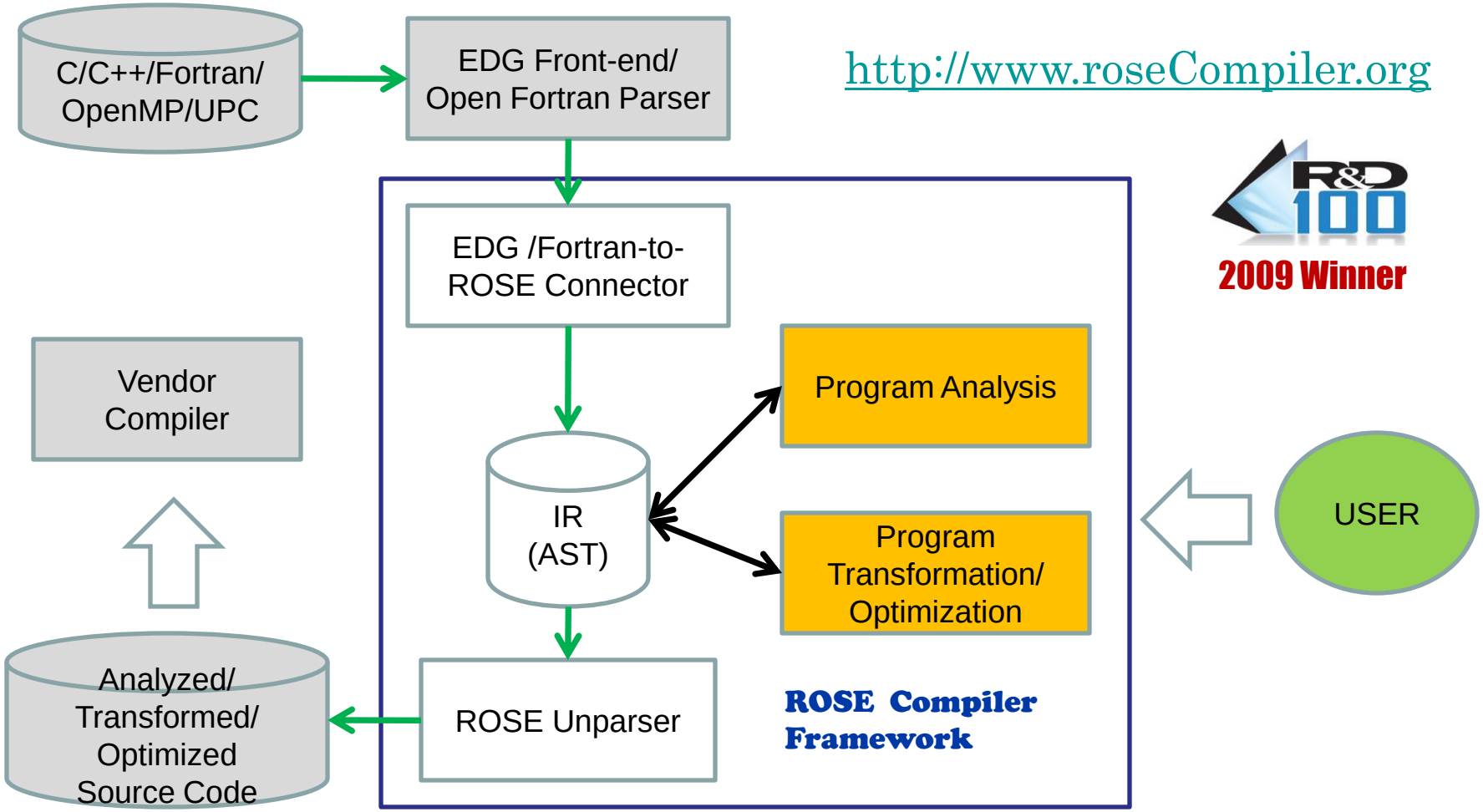
Semantics of abstractions: an unexploited gold mine

- Semantics: any standard or user-defined meanings
  - *STL::vector<T>* elements stored contiguously
  - *a->foo(x)* read only
  - *STL::Set<mytype>* order does not matter
- Not fully exploited by traditional programming models:
  - Traditional pmodel: write code and throw it to a vendor compiler
  - Low level intermediate representation (IR)
  - Info. hiding mechanism
  - Semantics: too many and too diverse
- Performance is inversely proportional to the level of abstractions used

An abstraction-friendly HPC programming model

- Goal: encourage the best programming practice while maintaining high or even better performance
- Solution:
  - User intervention
    - A specification: abstractions + semantics
    - User-defined optimizations: eliminate the dependence on compiler experts
  - An extensible source-to-source compiler framework
    - Recognize abstractions
    - Semantics-aware optimizations
    - Complement vendor compilers

ROSE: making compiler technology accessible



ROSE intermediate representation (IR)

- ROSE IR = AST + symbol tables + CFG ...
- Preserves all source level details
  - Token stream, source comments
  - C preprocessor control structure
  - C++ templates
- Rich interface
  - AST traversal, query, creation, copy, symbol lookup
  - Generic analyses, transformations, optimizations
- Fully support
  - Abstraction recognition and semantic analysis

Case 1: a vector computation loop

```
 std::vector <double > v1(SIZE);
…
 double sum = 0.0;
std::vector <double>::iterator iter;
for (iter= v1.begin(); iter!=v1.end(); iter++)
    sum = sum +*iter;
```

- std::vector : better productivity than arrays
  - Dynamic allocated, automatic de-allocation
  - Easier resizing, boundary-check
- Impede optimizations
  - E.g. Auto parallelization: primitive arrays
  - Non-canonical loop: *for (integer-init; test; increment) block*
  - Obscure element accesses: dereferencing an iterator

Semantics can help

- Semantics of *std::vector <T>*
  - An array-like container
  - Element access methods: *[], at(), *iterator*
- Loop normalization:
  - loops using iterators → loops using integers
  - **iterator → container.at(i)*
- Dependence analysis:
  - element_access_method(i)→ subscript i

```
 double  v1[SIZE];
 double sum = 0.0;
int i;
#pragma omp parallel for reduction (sum)
for (i = 0; i<SIZE; i++)
    sum = sum + v1.at(i);
```

Case 2: a domain-specific tree traversal

- Compass: A ROSE-based tool for static code analysis
  - A checker: detect a violation of MISRA* Rule 5-0-18

```
void CompassAnalyses::PointerComparison::Traversal::visit(SgNode* node){
    // Check binary operation nodes
    SgBinaryOp* bin_op = isSgBinaryOp(node);
   if (bin_op)  {
      // Check relational operations
     if (isSgGreaterThanOp(node) || isSgGreaterOrEqualOp(node) ||
        isSgLessThanOp(node) || isSgLessOrEqualOp(node))  {

      SgType* lhs_type = bin_op->get_lhs_operand()->get_type();
      SgType* rhs_type = bin_op->get_rhs_operand()->get_type();
     // Check operands of pointer types
     if (isSgPointerType(lhs_type) || isSgPointerType(rhs_type))
     // output  a violation
         output->addOutput(bin_op);
   }   }  }
```

*The Motor Industry Software Reliability Association: MISRA C++: 2008 Guidelines for the use of the C++ language in critical systems.

Semantics can help

- Enabling parallelization
  - Read-only semantics
    - Information retrieval functions: *get_*()*
    - Type casting functions: *isSg*()*
  - Order independent side effects
    - *output->addOutput(bin_op)*
    - Suitable for using *omp critical*
- Enabling customized optimization
  - Order-independent tree traversal
  - Nodes stored in memory pools
  - Recursive tree traversal → Loop over memory pools

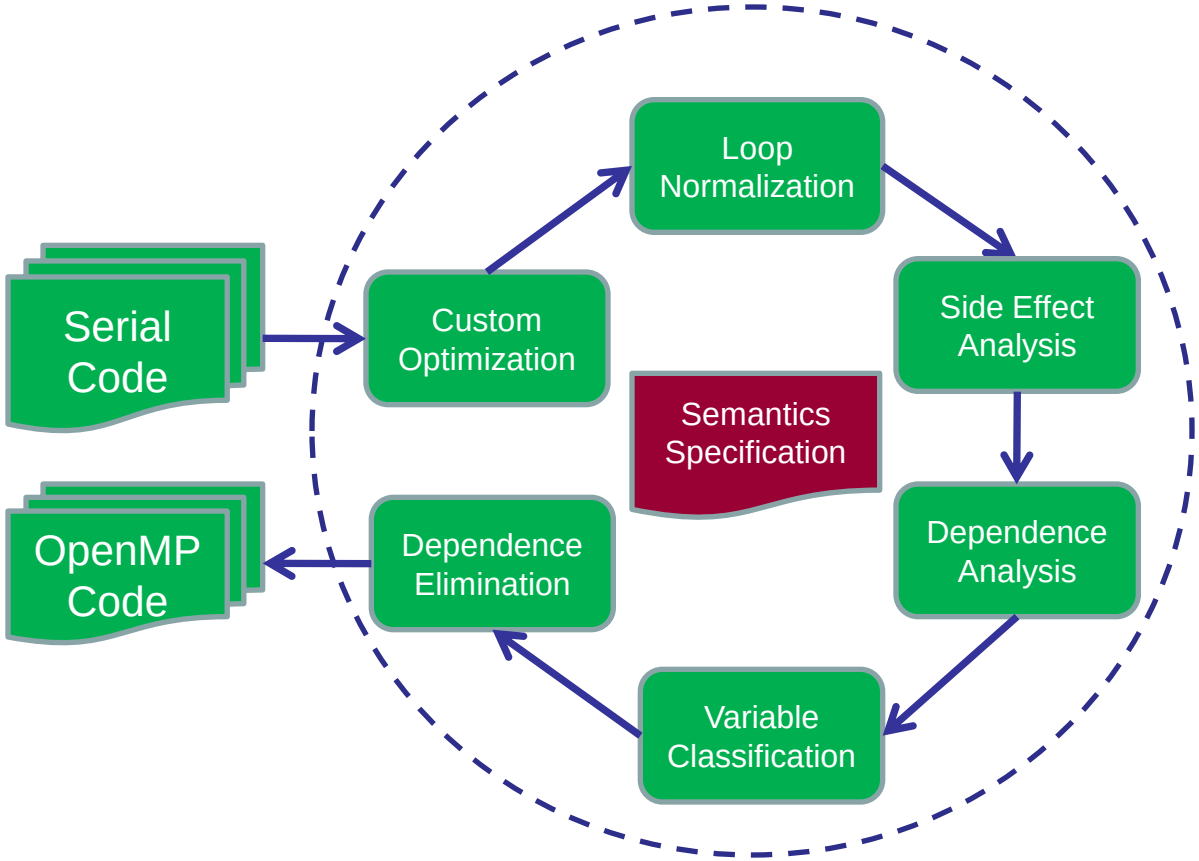
Optimized and Parallelized Code

```
#pragma omp parallel for
for (i=0; i<pool_size; i++)
{
   SgBinaryOp* bin_op = isSgBinaryOp(MEMPOOL[i]);
   if (bin_op)
   {
       if (isSgGreaterThanOp(node) || isSgGreaterOrEqualOp(node) ||
           isSgLessThanOp(node) || isSgLessOrEqualOp(node))  {

         SgType* lhs_type = bin_op->get_lhs_operand()->get_type();
         SgType* rhs_type = bin_op->get_rhs_operand()->get_type();

       if (isSgPointerType(lhs_type) || isSgPointerType(rhs_type))
       {
           #pragma omp critical
           output->addOutput(bin_op);
       }
      }
    }
}
```

Implementation: a semantics-aware parallelizer



Chunhua Liao, Daniel J. Quinlan, Jeremiah J. Willcock and Thomas Panas, **Extending Automatic Parallelization to Optimize High-Level Abstractions for Multicore**, *IWOMP 2009 - International Workshop on OpenMP*, Dresden, Germany, 3-5 June 2009

An abstraction/semantics specification file

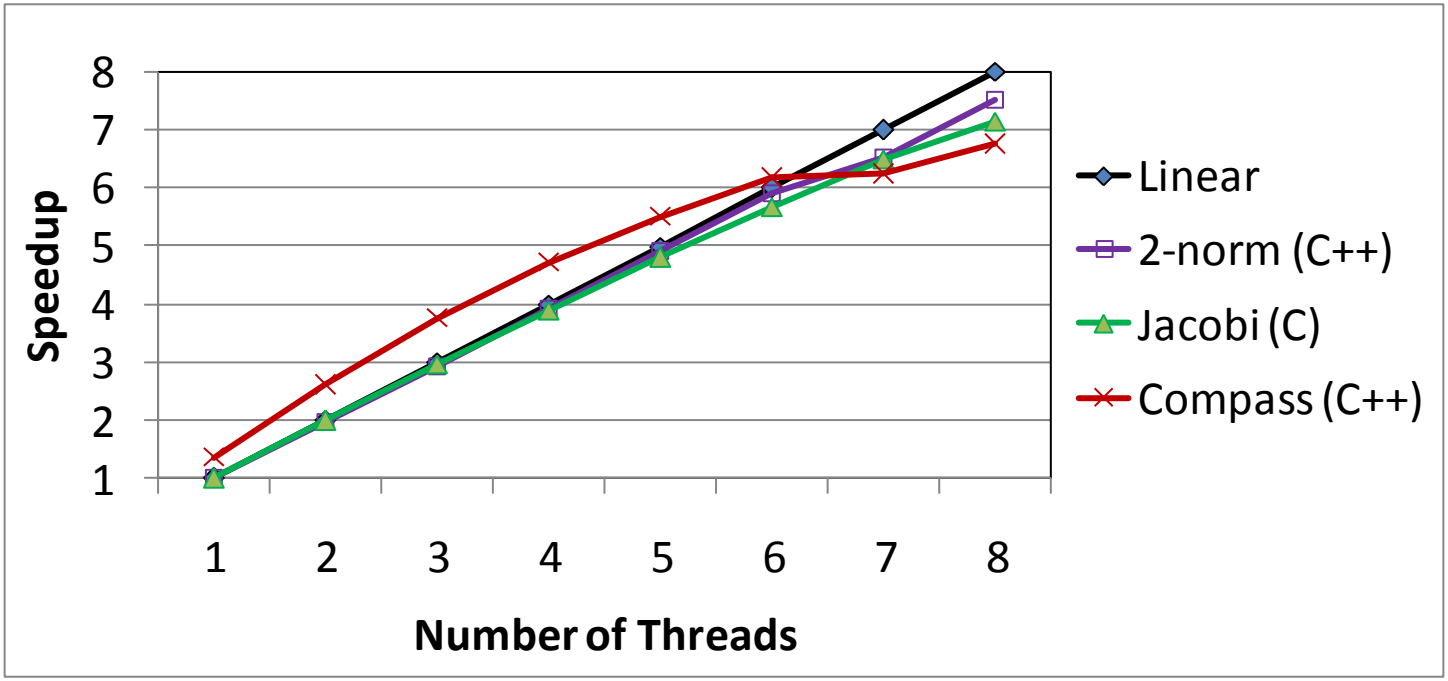
```
class std::vector<MyType> {
  alias none; overlap none; //elements are alias-free and non-overlapping
  is_fixed_sized_array { //semantic-preserving functions as a fixed-sized array
    length(i) = {this.size()};
    element(i) = {this.operator[](i); this.at(i);};
  };
};

SgXXX* isSgXXX(SgNode*node)
{ modify none; }  // read-only functions

SgNode*  SgNode::get_XXX()
{ modify none; } // read-only  member functions

void Compass::OutputObject::addOutput(SgNode* node){
   read {node};
   //order-independent side effects
   modify {Compass::OutputObject::outputList<order_independent>};
}
```

Preliminary results



Platform: Dell Precision T5400, 3.16GHz quad-core Xeon X5460 dual processor, 8GB
Compilers: ROSE OpenMP translator + Omni 1.6 Runtime + GCC 4.1.2

Related Work

- Kennedy, et.al. Telescoping languages: a system for automatic generation of domain languages, proceedings of IEEE, 2005
  - High-level scripting languages, library preprocessing
- Gregor, Schupp, STLlint: lifting static checking from languages to libraries, Softw. Pract. Exper. 2006
  - Static analysis of error use of abstractions
  - C++ syntax for specification
- Kulkarni, Pingali, et.al. Optimistic parallelism requires abstractions. PLDI 2007
  - Abstraction: un-ordered set;
  - Semantics: commutativity, inverse

Conclusions and future work

- ROSE-based abstraction-friendly programming model:
  - High productivity (use of abstractions) + high performance (semantics-aware optimizations)
  - Source-to-source: complement vendor compilers
  - User intervention: less depend on compiler experts
- Future work
  - Better specification files
  - Classify and formalize more abstractions/semantics
  - Operations on semantics to generate new semantics?

Questions?