A Source-to-Source Transformation Framework for Architecture-Dependent Performance Optimization

Daniel Quinlan
Markus Schordan

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
ROSE Talk Outline

- **Goal:** Simplify Scientific Software Development
  - Use High-level Abstractions in Libraries
  - High-Level Abstractions hide complexity
  - Optimize the Use of High-Level Abstractions in applications at Compile-Time

- **Treat Libraries as Domain-Specific Languages**
  - e.g. MPI source-to-source compiler
  - Source-to-source compiler for generating code for a machine specific API (e.g. BG/L low level APIs)

- **ROSE:** Source-to-source Compiler Framework
  - Automated Recognition of high-level abstractions
  - Specification of Transformations
  - Language Independent (currently C and C++)

- **Simple Example Problem**
- **Results**
- **Conclusions**
Telescoping Languages are defined by grammars that are an extension of the base language grammar.

The extension of a base level grammar is the addition of library abstractions as terminals in the telescoping language grammar.

User-defined types become part of the type system within the telescoping language. This step can be automated.

Library 1 ---> Telescoping Language: T1
Library 2 ---> Telescoping Language: T2
Library 3 uses library 2: ----> T2:T3
Overview of ROSE Approach

Library header files are used to build source code representing preprocessor’s Recognition Mechanism.

ROSE Extensible Annotation Mechanism
Starting the Two Processor per node fist fight
  — L1 Cache is has no coherency
  — Special API is defined for manual sweeping L1 cache to L3
  — It might not be *fun* to write or debug code specific to two processor use of BG/L nodes

Management can’t restrain from multiplication by 2

It might be possible to automate the generation of code to use the two processor API.
  — Need more hardware information
  — Need examples of what code might look like
  — Need to understand what additional info is required
```cpp
int main() {
    int a[10];
    for(int i=0;i<10;i++)
        a[i]=i*i;
    return 0;
}
```

**ROSE 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
- Source code generation
int main() {
    Range I(1, 98, 1), J(1, 98, 1);
    doubleArray A(100, 100);
    doubleArray B(100, 100);
    A(I, J) = B(I, J) + B(I, J) + B(I, J);
    return 0;
}
Example High Level Abstraction: P++ parallel array class

• Developed in 1990-91, published 1992
• Stencil operations on structured grids are naturally expressed in terms of array operations
• Details of parallel implementation can be hidden from the user by the parallel array class

```
int n=100;
Index I(1,n-2), J(1,n-2);
floatArray u(n,n);

// update stencil and communicate between processors
u(I,J) = .25*(u(I-1,J) + u(I+1,J) + u(I,J-1) + u(I,J+1));
```
**ROSE Transformation Example**

**High-Level abstractions mapped to low-level implementation**

### P++ Code

```cpp
Index I (1, n, 1);
doubleArray Solution(n+1), old_Solution(n+1);
doubleArray RHS(n+1);
old_Solution(I) =
((h*h)*RHS(I) + Solution(I+1) + Solution(I-1)) / 0.5;
```

### Automated ROSE Transformation

```cpp
Index I (1, n, 1);
doubleArray RHS(n+1);
doubleArray Solution(n+1);
double* restrict RHS_data = RHS.getDataPointer();
double* restrict Solution_data = Solution.getDataPointer();

int I_index = 0;
int I_base = I.getBase();
int I_bound = I.getBound();

for (I_index = I_base; I_index < I_bound; I_index++)
    old_Solution_data[I_index] = ((h*h)*RHS_data[I_index] +
                                  Solution_data[I_index+1] +
                                  Solution_data[I_index-1]) / 0.5;
```
Example Problem

\[ u_t + u_x + u_y = f \]

```cpp
for (int k = 0; k<100; k++)
{
    temp(ix1,iy1) = old_A(ix1,iy1) -
    2.*dt*((A(ix1+1,iy1)-A(ix1-1,iy1))/(2*dx) +
    (A(ix1,iy1+1)-A(ix1,iy1-1))/(2*dy) -
    (4+2*t+x(ix1,iy1)+y(ix1,iy1)) );
    old_A = A;
    A(ix1,iy1)= temp(ix1,iy1);
    A(all,jL)=(1+(t+dt))*(2+x(all,jL)+y(all,jL));
    A(all,jU)=(1+(t+dt))*(2+x(all,jU)+y(all,jU));
    A(iL,iy1)=(1+(t+dt))*(2+x(iL,iy1)+y(iL,iy1));
    A(iU,iy1)=(1+(t+dt))*(2+x(iU,iy1)+y(iU,iy1));
    A.updateGhostBoundaries();
    t +=dt;
}
```
Relative Performance Improvement
(Using Preprocessor Build with ROSE)

Scaling of Array Statement Abstraction
(2nd Order Linear Advection Test Problem)

Execution Time vs. Number of Processors
Slope representing ideal scaling
Automated Cache-Transformations

Speedup of Two Transformations for Cache Optimization

Time Iterations

Speedup

Temporal Tiling
Spatial Tiling
Conclusions

- High Level Abstractions Simplify Applications
- Semantics of User-Defined Abstractions can be used
- Performance is at least as good as lower level C/F77/HPF
- Many Cache-based optimizations provide better performance than vendor compilers
- Should simplify (automate) use of both processors on BG/L nodes

Future Work
- Use of better program analysis (RICE)
- Leverage General Compiler Optimizations (Broadway)
- More Cache Based Optimizations
- Parallel Communication Optimizations
Unparsed Example
Takes applications apart and puts them back together

Original Input C++ Source code
```
#include "A++.h"
#include "./include/ROSE_TRANSFORMATION_SOURCE.h"
#include <iostream.h>

int main() {

    int x = 4;

    //these comments are difficult
    for (int i = 0; i < 10; i++) {
        while (x) {
            x = x + 1;

            if (false) { x++; x = 7+x; }
            else {
                x = x - 1;
                x--;
            }
        } // comments!

        x++;
        x += 1;
    }

    return 0;
}
```

Unparsed Output C++ Source code
```
#include "A++.h"
#include "./include/ROSE_TRANSFORMATION_SOURCE.h"
#include <iostream.h>

int main() {

    int x=4;

    //these comments are difficult
    for (int i=0; i < 10; i++){
        while(x){
            x = x + 1;

            if (FALSE){ x++; x = 7 + x; }
            else {
                x = x - 1;
                x--;
            }

        } // comments!

        x++;
        x += 1;
    }

    return 0;
}
```