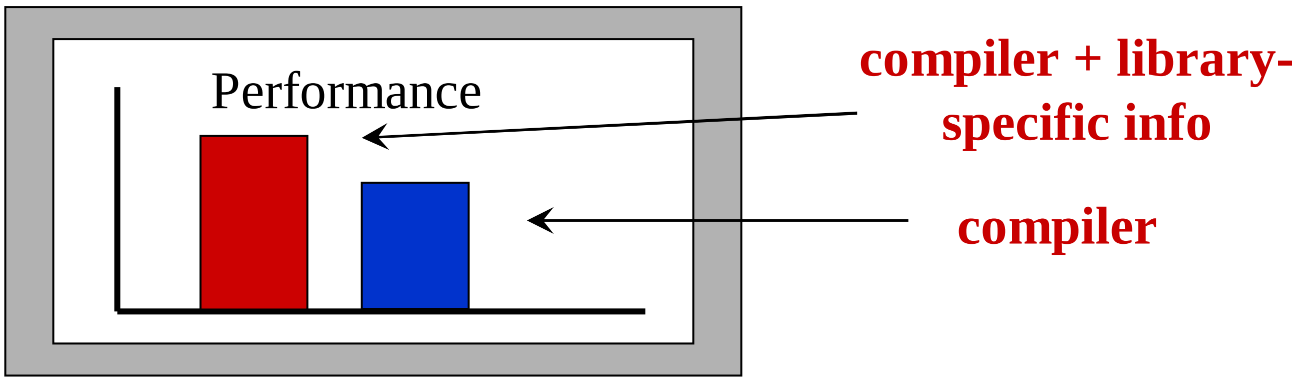
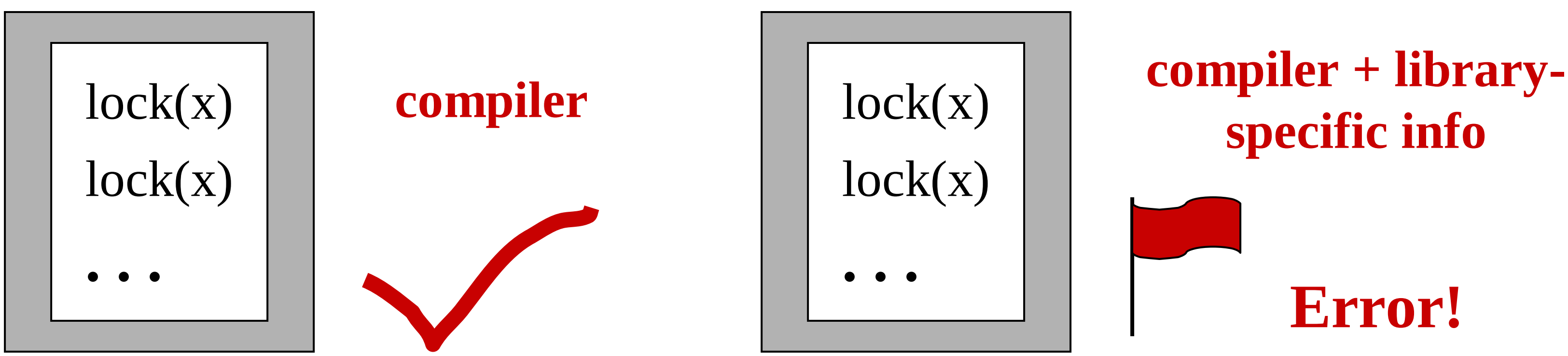
Compiler Support for Software Libraries

Calvin Lin
Samuel Guyer
University of Texas at Austin
June 22, 2002

Motivation

- Numerous libraries exist
- There's a huge benefit to providing compiler support for libraries





Outline: Compiler Support for Libraries

- Requirements
- Our Solution
- Conclusions

Optimization Example

- Consider a dot product routine

```
void dot-product(X,Y,Z){
  for (int i=0,i<n; i++){
    X[i] = Y[i]* Z[i];
  }
}
```

- Consider a common transformation to improve locality

```
dot-product(T1,A,B);
dot-product(T2,T1,C);
```

➡

```
for (i=0; i<n; i++){
  t1    = A[i] * B[i];


  2T[i] = t1   * C[i];
}
```

Syntactic Manipulation is Limited

- We must preserve data dependences

```
dot-product(T1,A,B);
A[0] = 10;
dot-product(T2,T1,C);
```

?

```
for (i=0; i<n; i++){
  t1    = A[i] * B[i];
  T2[i] = t1   * C[i];
}
A[0] = 10;
```

Syntactic Manipulation is Limited

- We must recognize aliases

```
dot-product(T1,A,B);
*p = 10;
dot-product(T2,T1,C);
```

?

```
for (i=0; i<n; i++){
  t1    = A[i] * B[i];
  T2[i] = t1   * C[i];
}
*p = 10;
```

- We must correctly handle interactions between the library and the application program

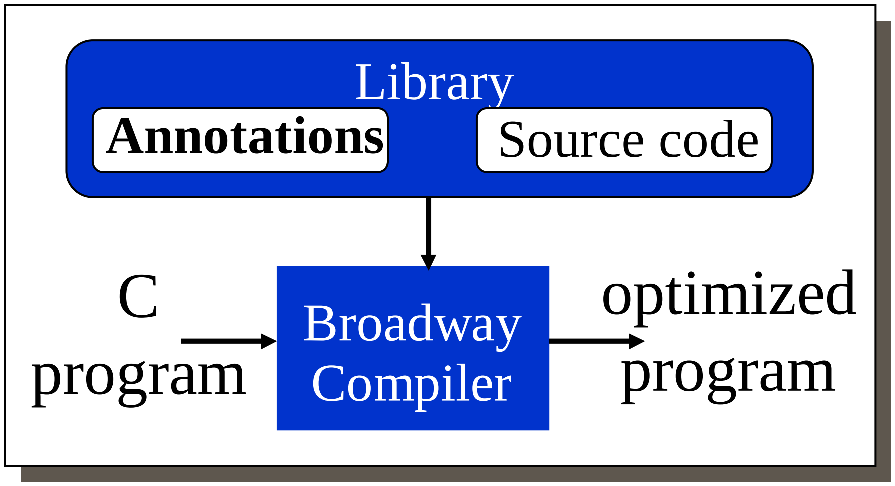
What Do We Need?

- Barriers to optimization
  - Data dependences
  - Pointers and aliasing
  - Control flow
  - Complex data structures
- We need the same analyses that traditional compilers use
  - Control flow analysis
  - Data-flow analysis
  - Pointer and dependence analysis

Are Traditional Compilers Sufficient?

- Libraries are lightweight domain-specific languages
  - ⇒ Compilers need to understand the semantics of these languages
- Each library has its own semantics
  - ⇒ We'd like one compiler for all libraries
- Each domain specific language is embedded in a base language
  - ⇒ We'd like our compiler to understand both languages and the interactions between them

Our Solution: The Broadway Compiler



- Annotations provide library-specific knowledge

- One compiler for all libraries
- Common theme:
  - Expose traditional compiler facilities so that they can be **easily** configured
  - Integrate the use of these facilities to apply to both libraries and the base language

Optimization Opportunities

I. Traditional optimizations on library operators

II. Specializations of library routines

III. Extensions of traditional optimizations to library operators

requires increasing library-specific information

I. Traditional Optimizations

- Trivial example
  - Loop invariant code motion

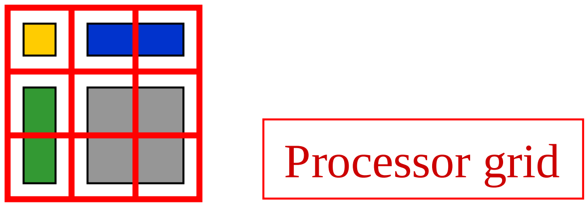
```
while (c)
{
  CheckState(x);
  . . .
}
```

```
CheckState(x);
while (c)
{
  . . .
}
```

- Requires dependence analysis (or annotations)

II. Library Specialization

- Idea
  - Analyze dynamic program properties
  - Use this information to specialize routines
- Consider a parallel matrix computation
  - Submatrices can have special properties



  - Can replace a parallel algorithm with a sequential one
- Requires library-specific data-flow analysis

III. Extensions of Traditional Optimizations

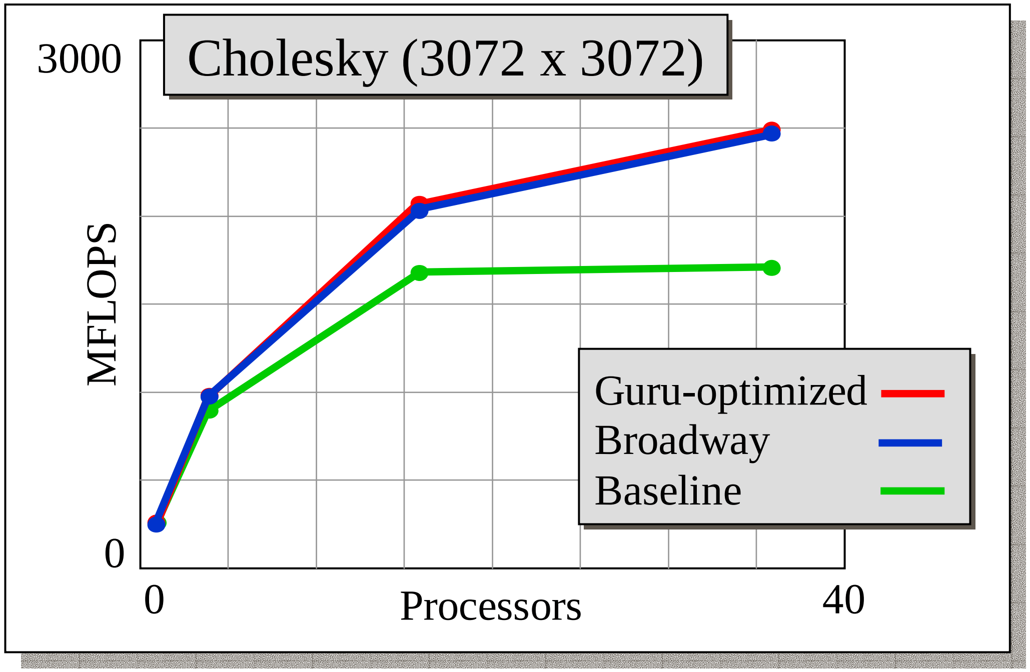
- Example
  - Constant propagation
- Objects often store state and libraries provide routines to access this state
  - If we can statically determine the state of this object, we can replace function calls with the constant itself
- Requires dependence analysis
- Requires annotations if the state is not stored in an easily accessible form

The Broadway Compiler

- Two configurable mechanisms
  - Configurable dependence analysis
    - Procedure side effects
    - Pointer relationships
  - Configurable data-flow analysis
- Configurations specified through annotations
- Integrated with built-in mechanisms
  - Aggressive context- and flow-sensitive pointer analysis
  - Various standard optimizations

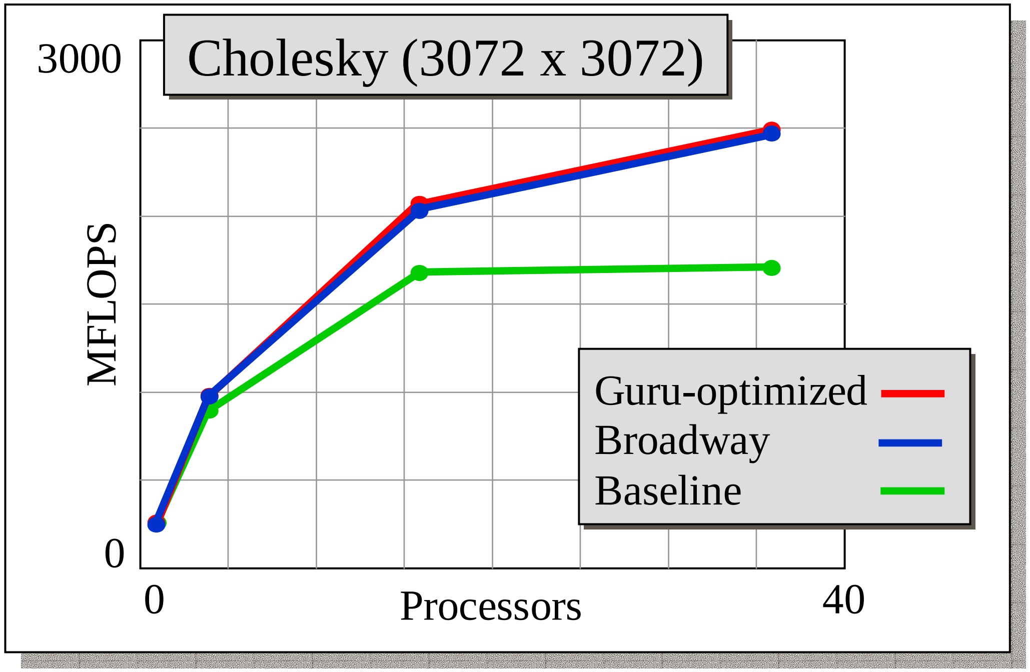
Performance Results

- Applied to unmodified PLAPACK parallel dense linear algebra library [van de Geijn 1997]
- Unmodified application and library source code



Observations from PLAPACK Results

- Interactions among multiple optimizations are essential
- Interaction between library and base language are important
- There is benefit to optimizing at multiple levels of abstraction



Future Work

- Many other uses of domain-specific compilation
  - Can check for program errors
    - Broadway has been used to identify security holes (Format String Vulnerabilities)
    - More precise than other approaches [Berger,Guyer,Lin 2001]
  - Can remove overhead of language interoperability for PETSc

Conclusions

- Aggressive program analysis is important
- Significant performance gains possible
- The big picture: