Efficient Support for Complex Numbers in Java

Peng Wu

Polaris Group
University of Illinois, Urbana-Champaign

Sam Midkiff, José Moreira, Manish Gupta

IBM T.J.Watson Research Center

Java Performance for Technical Computing

- VM Overhead
- Language Issues
  - Inefficient implementation of some numerical types
  - Precise exception capture semantics
  - Object-oriented programming
  - Strongly-typed references everywhere
  - multi-threaded programming
- Compiler issues
  - Efficient support of basic numerical types

Complex Numbers in Java

- Java does not support complex as primitive data type

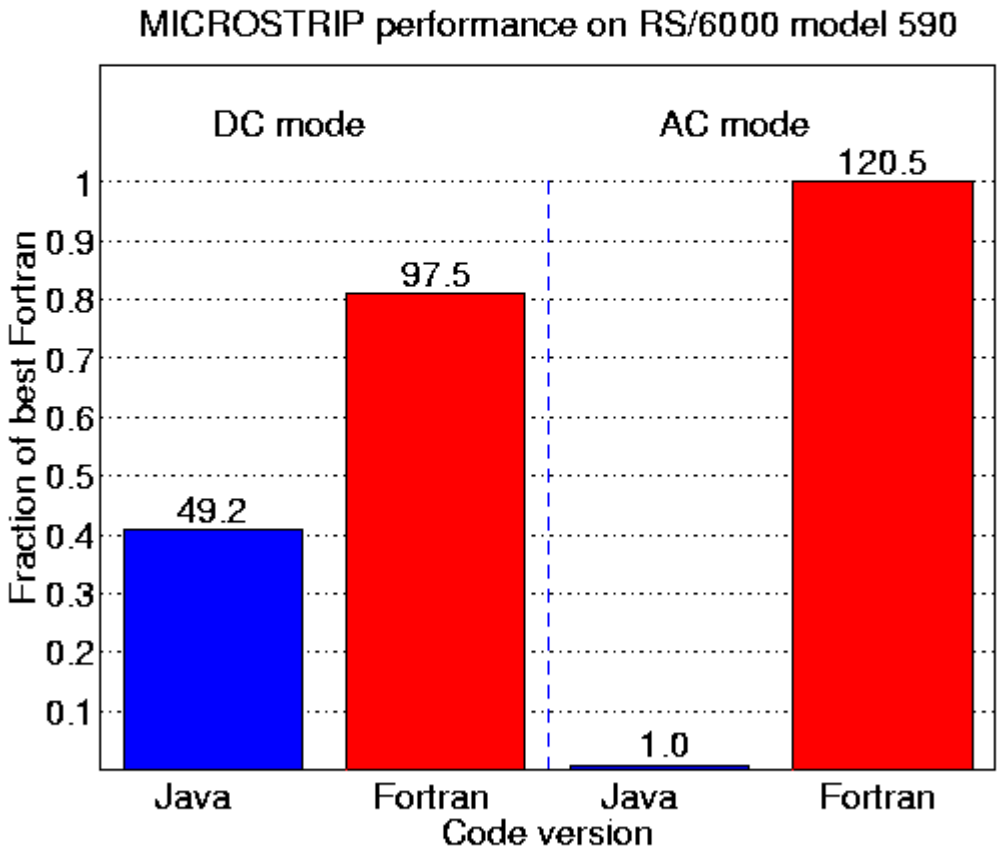
```
public final class Complex {
   public double re, im;
   public Complex(double r, double i);
   public Complex assign(Complex z);
   public Complex plus(Complex z);
   public Complex conjugate();
   …
}
```

```
a = new Complex(0,0)
```

MicroStrip Benchmark

```
do i=1,w-1
  do j=1,h-1
    B[i][j]=0.25*(A[i+1][j]+A[i-1][j]+
              A[i][j+1]+A[i][j-1])
  end do
end do
r=0;
do i=0,w
  do j=0,h
     r=r+|B[i][j]-A[i][j]|
  end do
end do
w=h=999
```

How about the Performance?



Why is Java so slow?

- Complex objects are very inefficient
  - object overhead
  - storage inefficiency
- Complicate traditional optimizations, especially for loop regions
  - Optimizing objects is far more difficult than optimizing scalars
  - When dealing with objects, pointers (references) are involved
  - Potential null-pointer exception and out-of-bound exceptions

Object Overhead

- Arithmetic operations on complex numbers creates many short lived objects

- Consider dot-product of two complex number vectors:

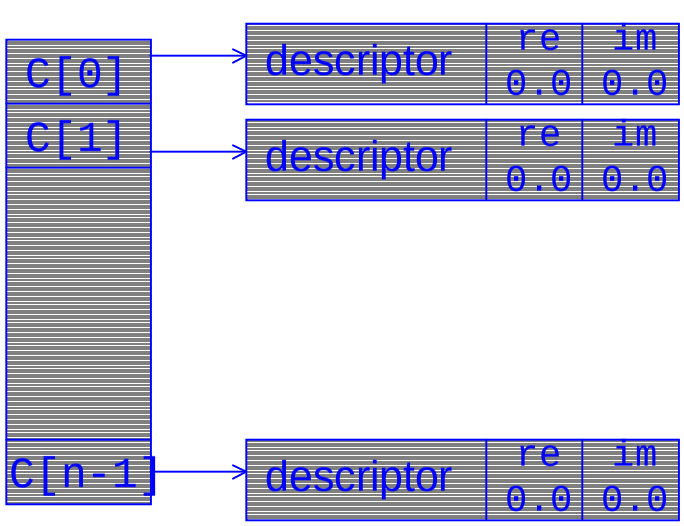
```
Complex[] a = new Complex[n];
Complex[] b = new Complex[n];
Complex s = new Complex(0,0);

for (int i=0; i<n; i++)
   s.assign(s.plus(a[i].times(b[i].conjugate())));
```

- Generate 3n Complex objects that only hold intermediate results

Storage Inefficiency

- Inefficient memory representation of Complex and Complex arrays



  - 50% storage used for object descriptor
  - extra levels of indirection to get the complex element in the array

An Alternative Complex Array Package

```
public final class ComplexArray2D {
  private double[] data;
  public ComplexArray2D(int m, int n) {
       data = new double[2*m*n];
  }
  public Complex get(int i, int j) {
       return new Complex(data[2*(I*n+j)],
   data[2*(I*n+j)+I]);
  }
  public void set(int i, int j, Complex z) {
       data[2*(I*n+j)] = z.re;
       data[2*(I*n+j)+1] = z.im;
  }
}
```

Using Complex Array Package

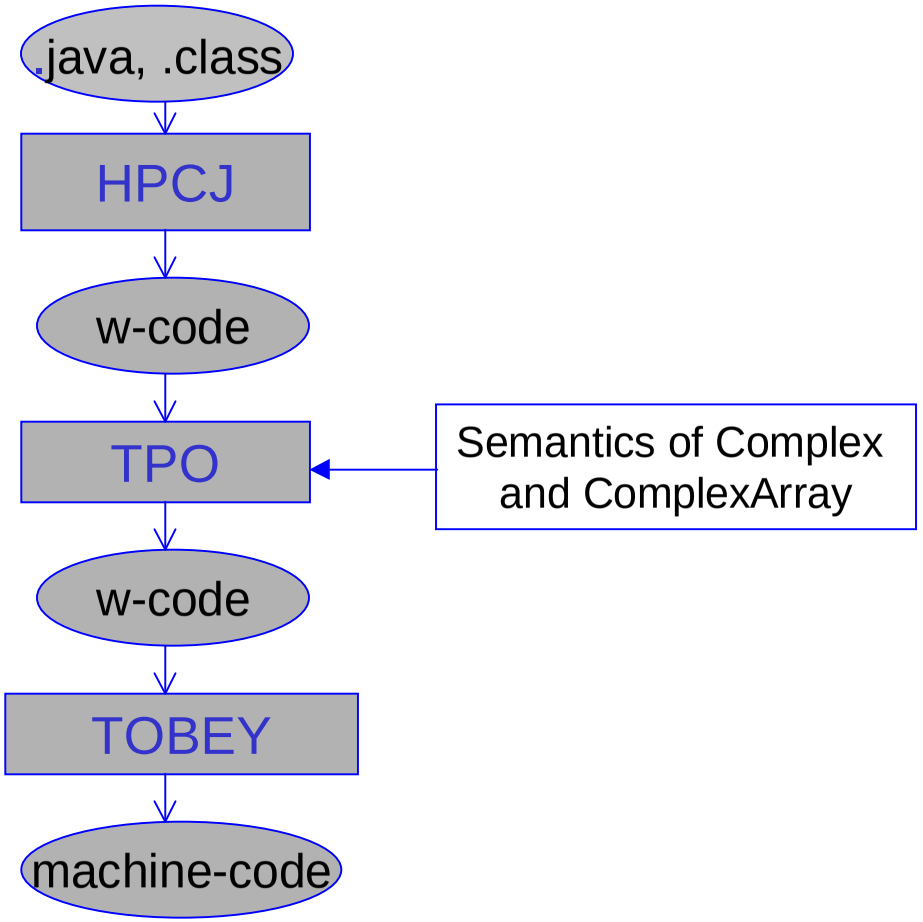
- More efficient memory layout
- The same object overhead or even more !!!
- Better behaved than arrays of complex
  - Rectangular shape
  - Less references and less aliasing
  - Almost no null-pointer exception, ease bounds checking

Our Solution

"To achieve Fortran-like performance on complex arithmetic, we must move beyond treating complex numbers as objects … "

- Compiler optimization to replace Complex objects by a more lightweight representation (pair of real and imaginary values)
  - reduce object overhead
  - makes ComplexArray class very attractive
  - enjoy traditional optimizations and better memory layout of ComplexArray

The Implementation



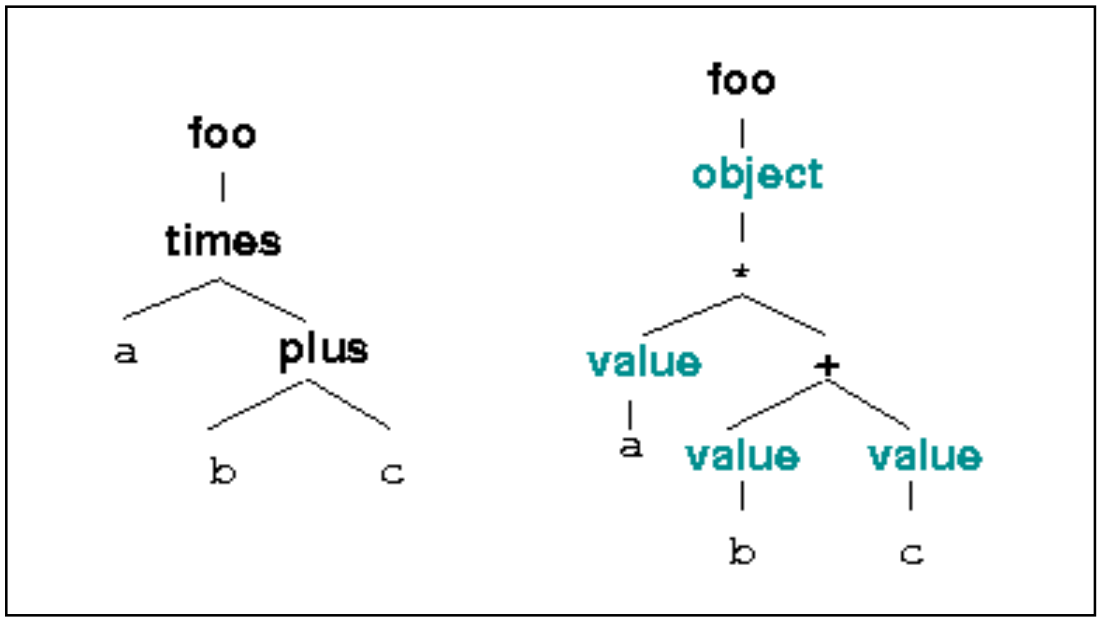
Semantic Expansion -- the Rationale

- Semantic expansion is a compilation strategy that the compiler treats selected classes as primitive data types
- We apply semantic expansion to Complex and ComplexArray
  - Complex numbers are essential in many areas
  - Semantics of complex numbers are simple and well-defined
  - Complex and ComplexArray classes are declared final
  - Mathematical nature makes special compiler handling profitable
- Semantic expansion greatly simplified the compiler optimization

Reducing Object Overhead

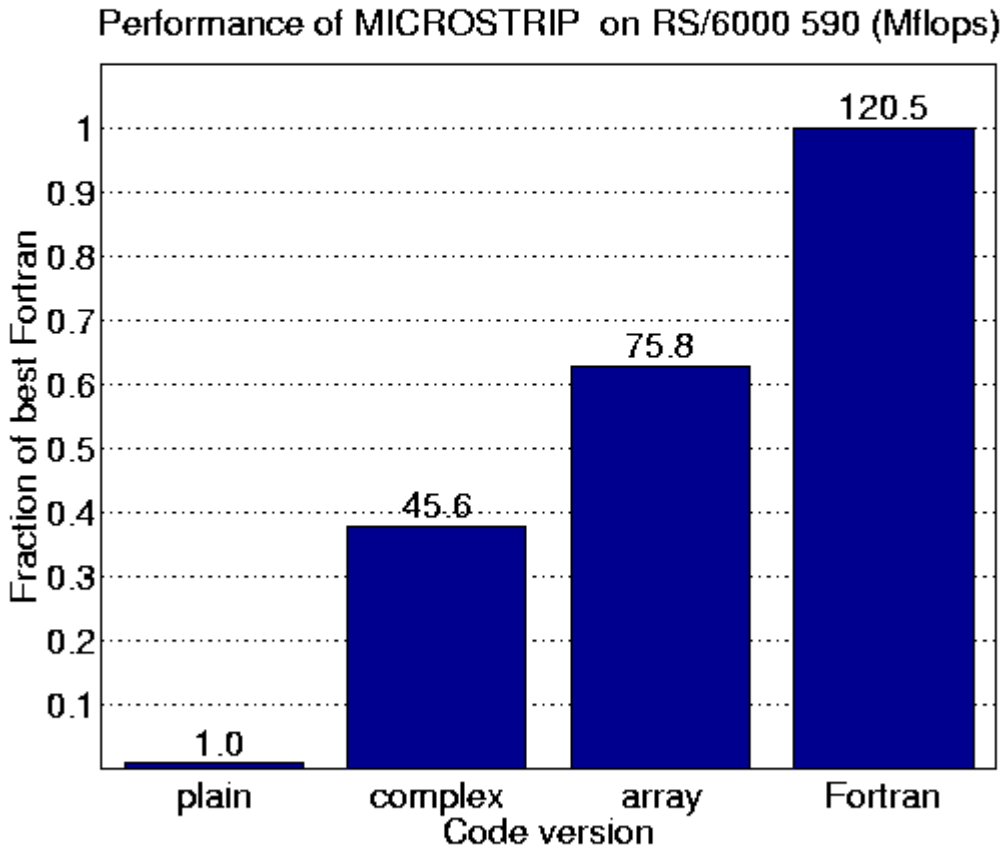
- Inlining Complex and ComplexArray method calls based on semantics
- Make methods of Complex and ComplexArray produce wcomplex instead of Complex objects
- Make methods of Complex and ComplexArray consume wcomplex instead of Complex objects
- Conversion from wcomplex to Complex if an object-oriented operation is performed
  - the “=“ operation
  - user-defined method expecting a Complex object

An Example

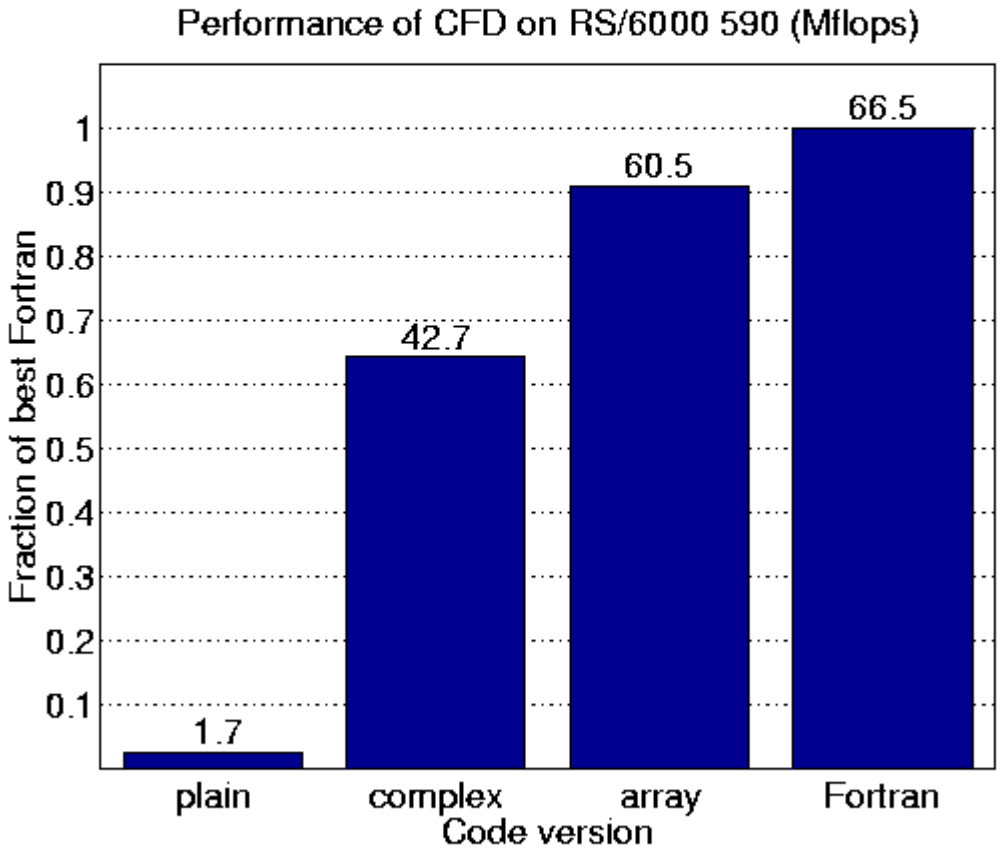


The AST for a complex arithmetic expression

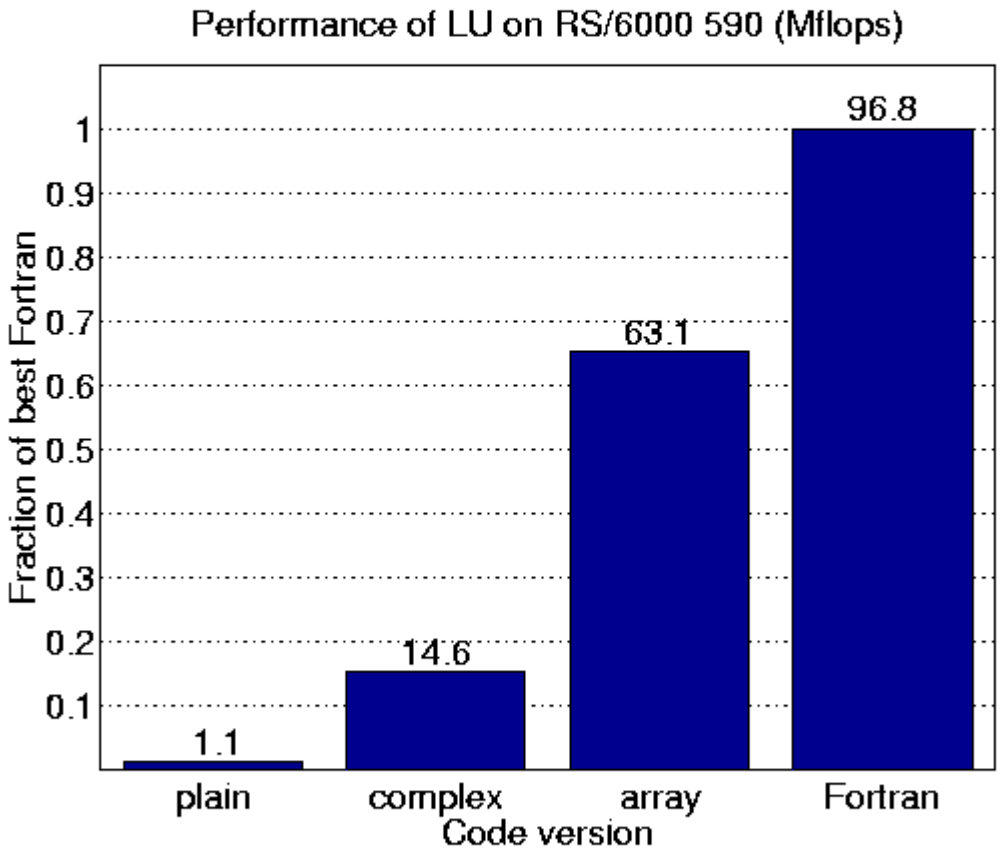
MicroStrip


Performance of MICROSTRIP on RS/6000 590 (Mflops)
Fraction of best Fortran
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
120.5
75.8
45.6
1.0
plain
complex
array
Fortran
Code version

Complex CFD code (256 x 256 grid)



Complex LU decomposition (500 x 500 grid)



Related Work

- Optimizing Java array package (Pedro Artigas, etc)
- Parallelization based on containers (Peng Wu and David Padua)
- Object Inlining (J. Dolby and Andrew Chien)
- Unboxing (C. Hall, etc)

Conclusion

- A simple yet efficient solution to support complex numbers for Java in technical computing
- Java performance can be 60% ~ 90% of Fortran for complex arithmetic
- A combined effort of compiler optimization and programmers' using "standard" classes