# ZS3:
# Marrying Static Analyzers and Constraint Solvers to Parallelize Loops in Managed Runtimes

Rishi Sharma
EPFL
rishi.sharma@epfl.ch

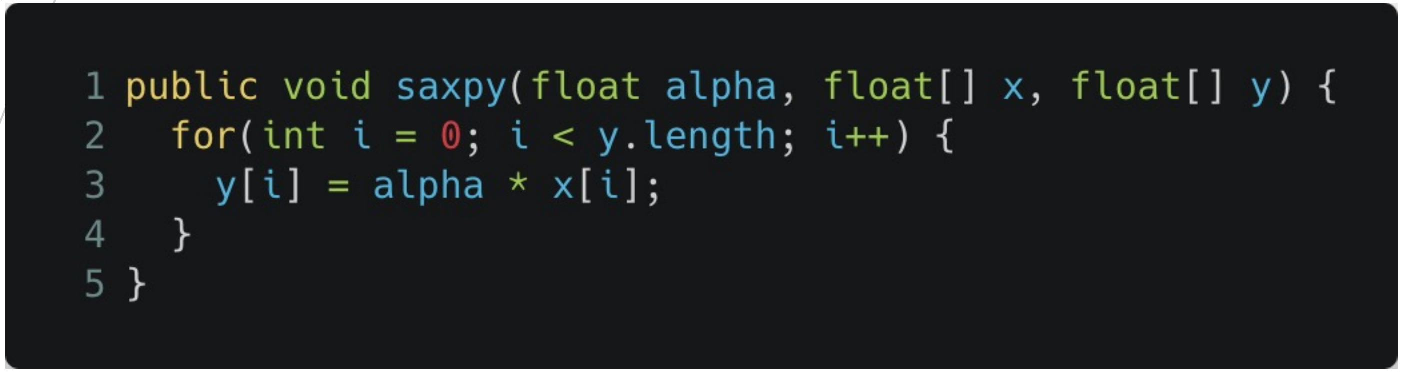
Shreyansh Kulshreshtha
Publicis Sapient
shreyanshkuls@outlook.com

Manas Thakur
IIT Mandi
manas@iitmandi.ac.in

ZS3

"A first of its kind loop parallelizer for Java programs that combines constraint solving and static analysis to mark parallelizable loops for heterogeneous architectures using TornadoVM. "

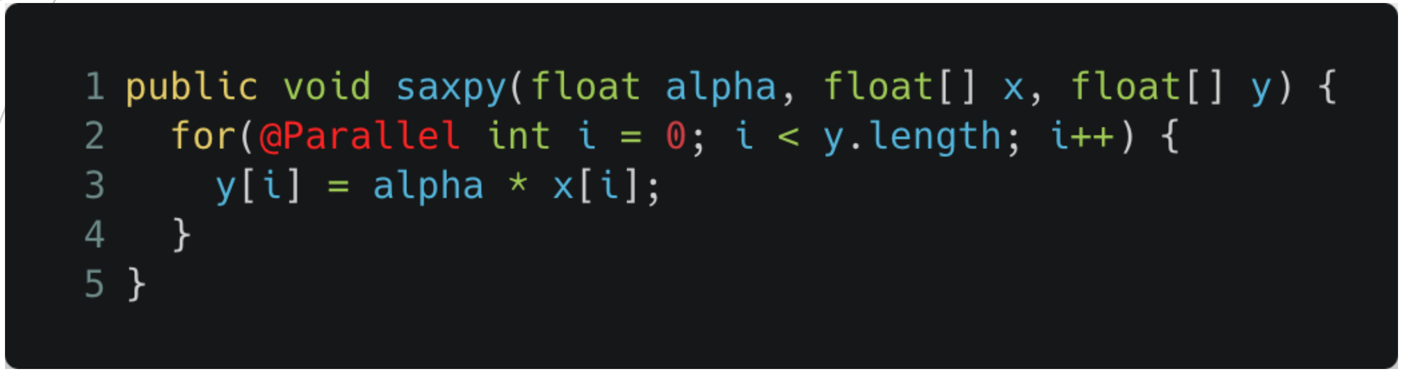
# Motivation



```java
public void saxpy(float alpha, float[] x, float[] y) {
  for(int i = 0; i < y.length; i++) {
    y[i] = alpha * x[i];
  }
}
```

Simple scalar multiplication

# Motivation

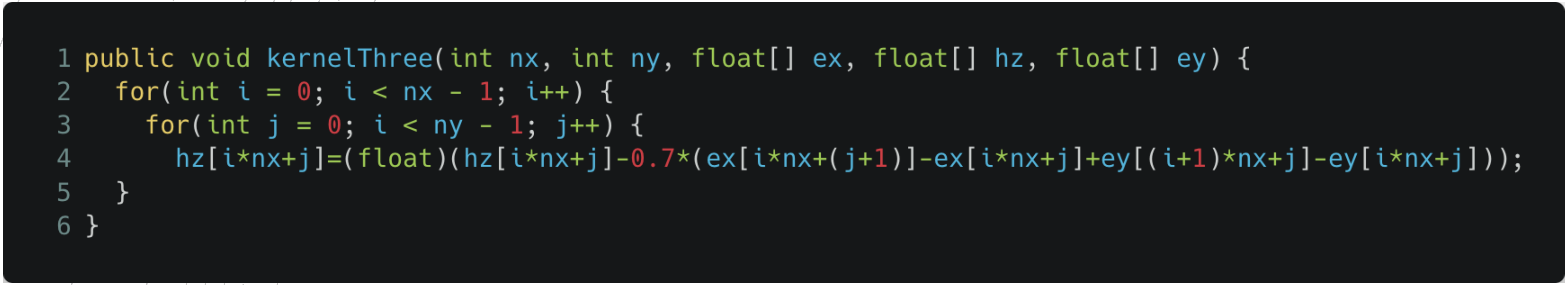


```java
public void saxpy(float alpha, float[] x, float[] y) {
  for(@Parallel int i = 0; i < y.length; i++) {
    y[i] = alpha * x[i];
  }
}
```

Simple scalar multiplication

Parallelizable, and easy to manually annotate!

# Motivation

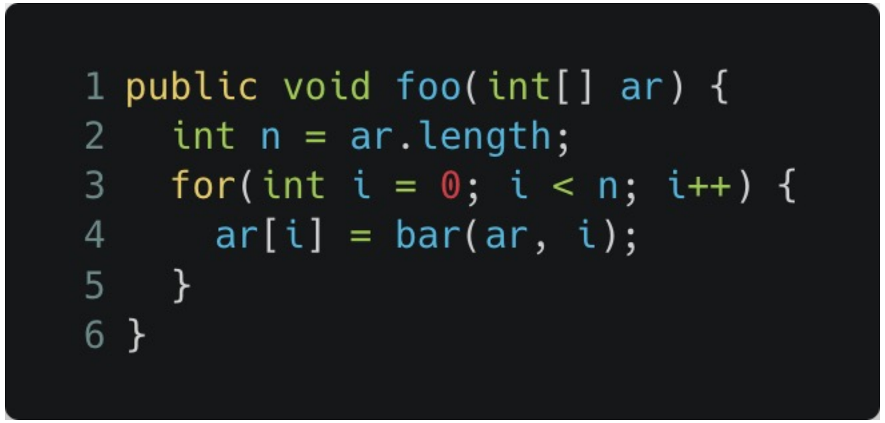


```java
public void kernelThree(int nx, int ny, float[] ex, float[] hz, float[] ey) {
  for(int i = 0; i < nx - 1; i++) {
    for(int j = 0; i < ny - 1; j++) {
      hz[i*nx+j]=(float)(hz[i*nx+j]-0.7*(ex[i*nx+(j+1)]-ex[i*nx+j]+ey[(i+1)*nx+j]-ey[i*nx+j]));
  }
}
```

A convolutional kernel function

What about this?

# Motivation

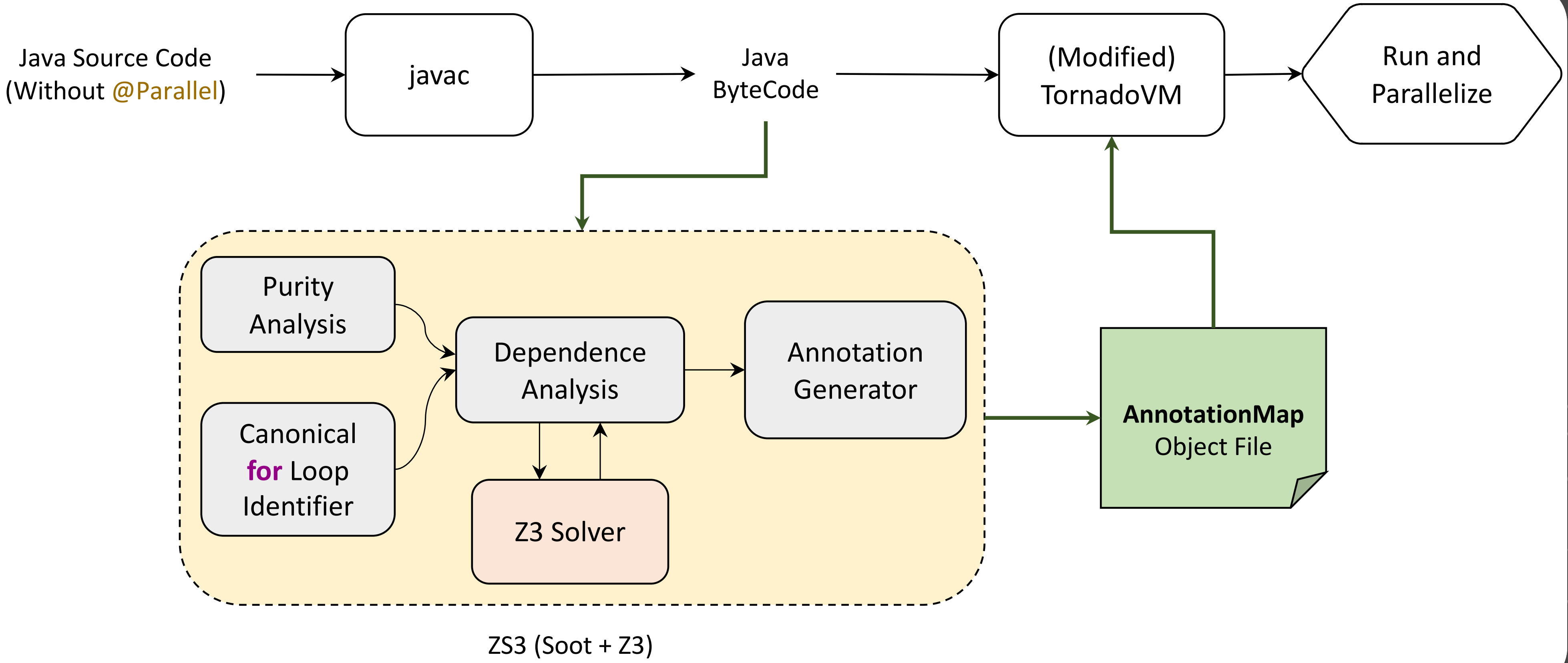


```java
public void foo(int[] ar) {
  int n = ar.length;
  for(int i = 0; i < n; i++) {
    ar[i] = bar(ar, i);
  }
}
```

Function calls

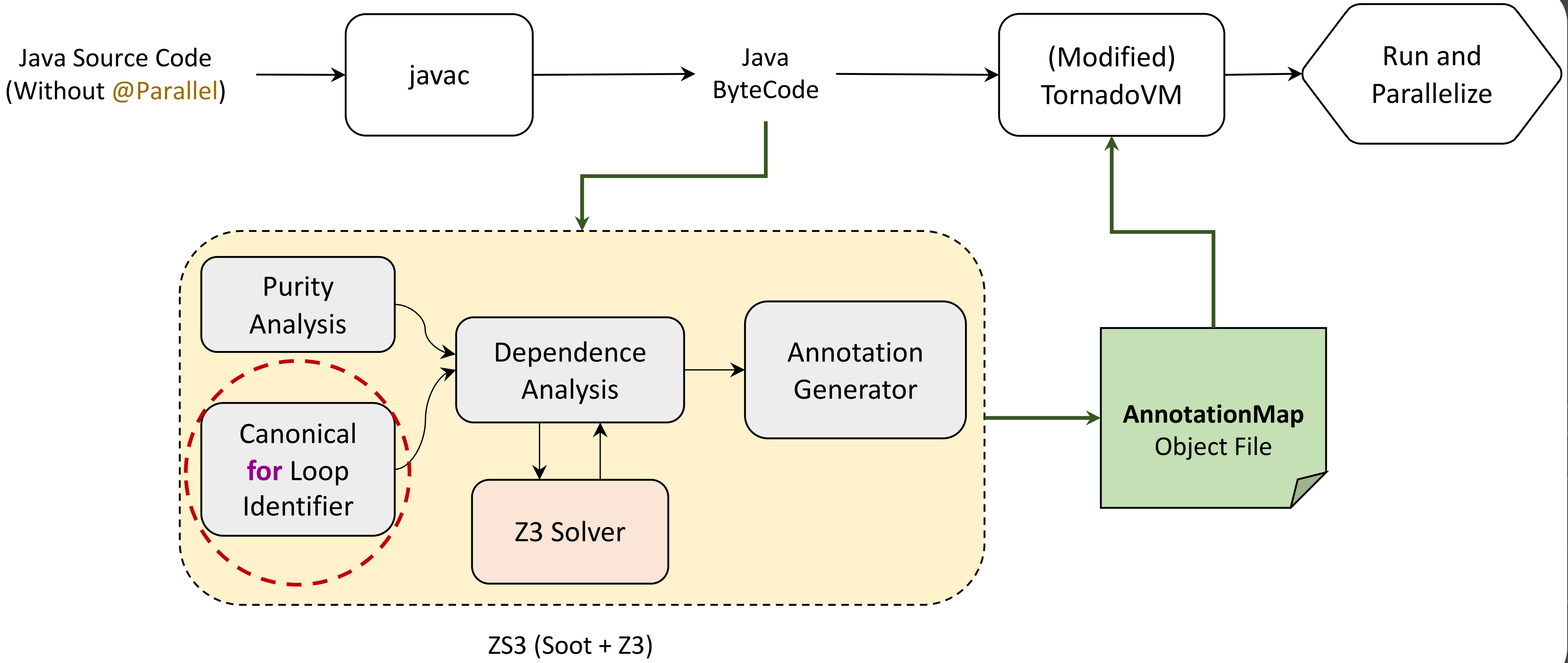
And this?

# The System



ZS3 Architecture

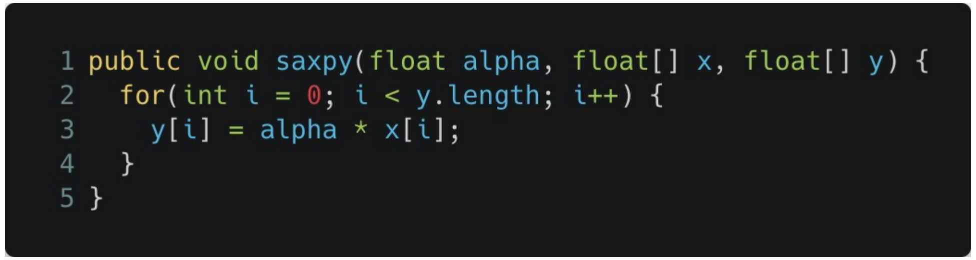
# The System


Java Source Code
(Without @Parallel)
javac
Java
ByteCode
(Modified)
TornadoVM
Run and
Parallelize
Purity
Analysis
Canonical
for Loop
Identifier
Dependence
Analysis
Z3 Solver
Annotation
Generator
ZS3 (Soot + Z3)
AnnotationMap
Object File

# The System

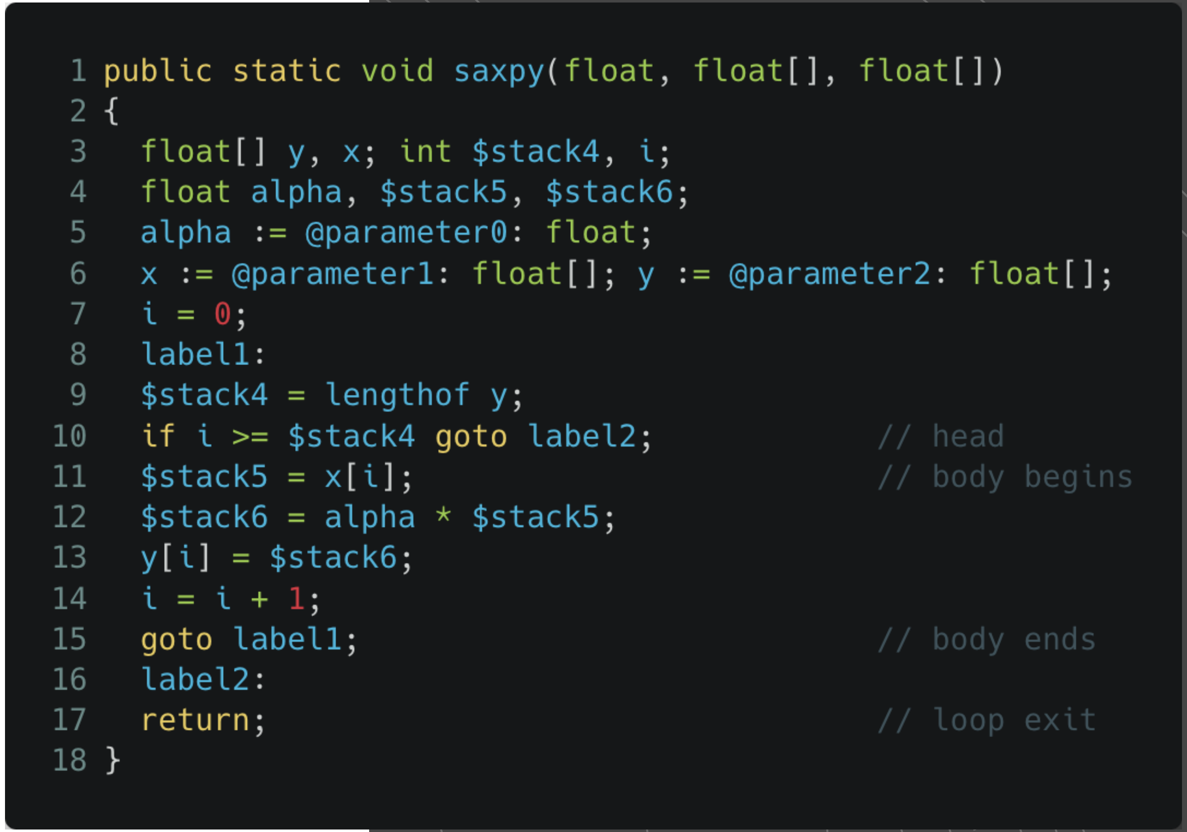
ZS3

## Canonical for loop identifier



```
public void saxpy(float alpha, float[] x, float[] y) {
  for(int i = 0; i < y.length; i++) {
    y[i] = alpha * x[i];
  }
}
```

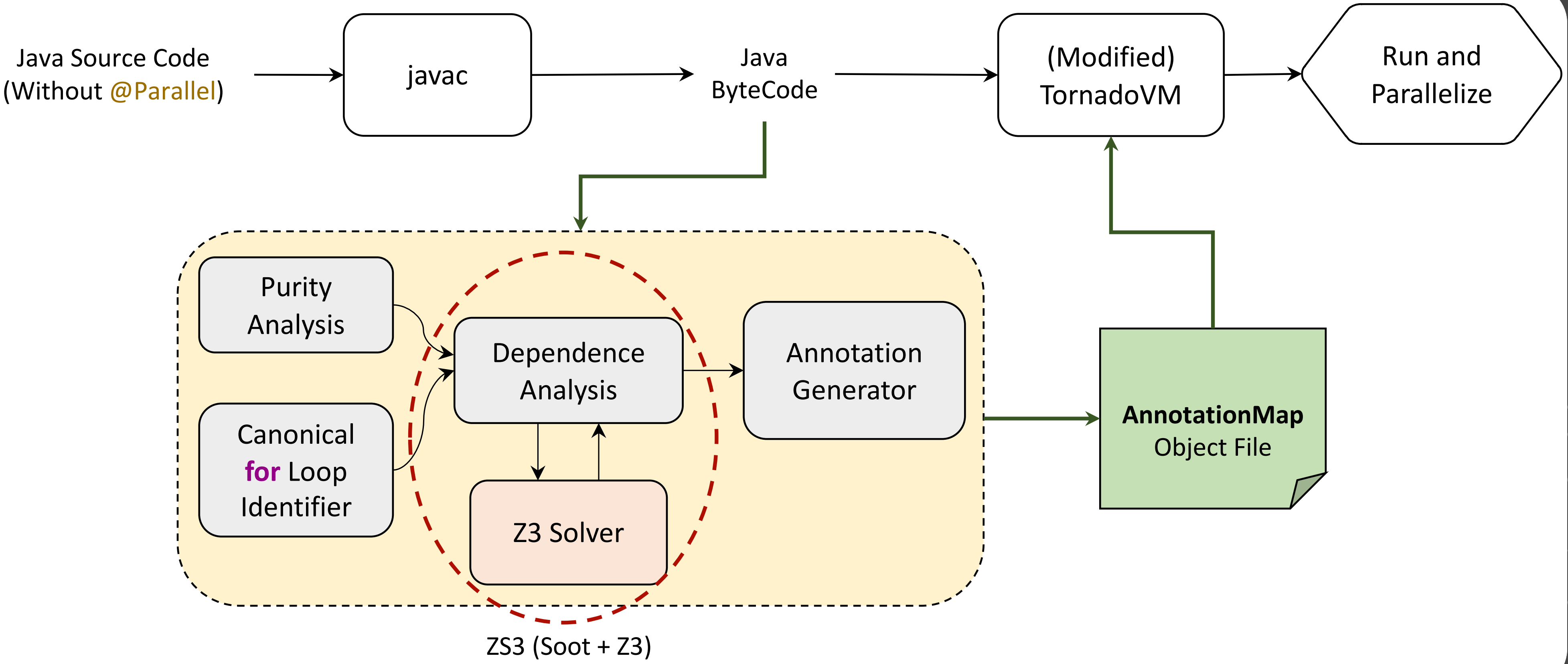
Java



```
public static void saxpy(float, float[], float[])
{
  float[] y, x; int $stack4, i;
  float alpha, $stack5, $stack6;
  alpha := @parameter0: float;
  x := @parameter1: float[]; y := @parameter2: float[];
  i = 0;
  label1:
  $stack4 = lengthof y;
  if i >= $stack4 goto label2;              // head
  $stack5 = x[i];                           // body begins
  $stack6 = alpha * $stack5;
  y[i] = $stack6;
  i = i + 1;
  goto label1;                              // body ends
  label2:
  return;                                   // loop exit
}
```

Jimple

# The System


Java Source Code
(Without @Parallel)
javac
Java
ByteCode
(Modified)
TornadoVM
Run and
Parallelize
Purity
Analysis
Canonical
for Loop
Identifier
Dependence
Analysis
Z3 Solver
Annotation
Generator
AnnotationMap
Object File
ZS3 (Soot + Z3)

# The System

## Dependence Analysis

### Scalar and Field References

- Writes to field-refs = Dependence!
- Writes to non-local variables = Dependence!
- Writes to local variables = No dependence!
- Reads to * = No dependence!

Need variable scopes

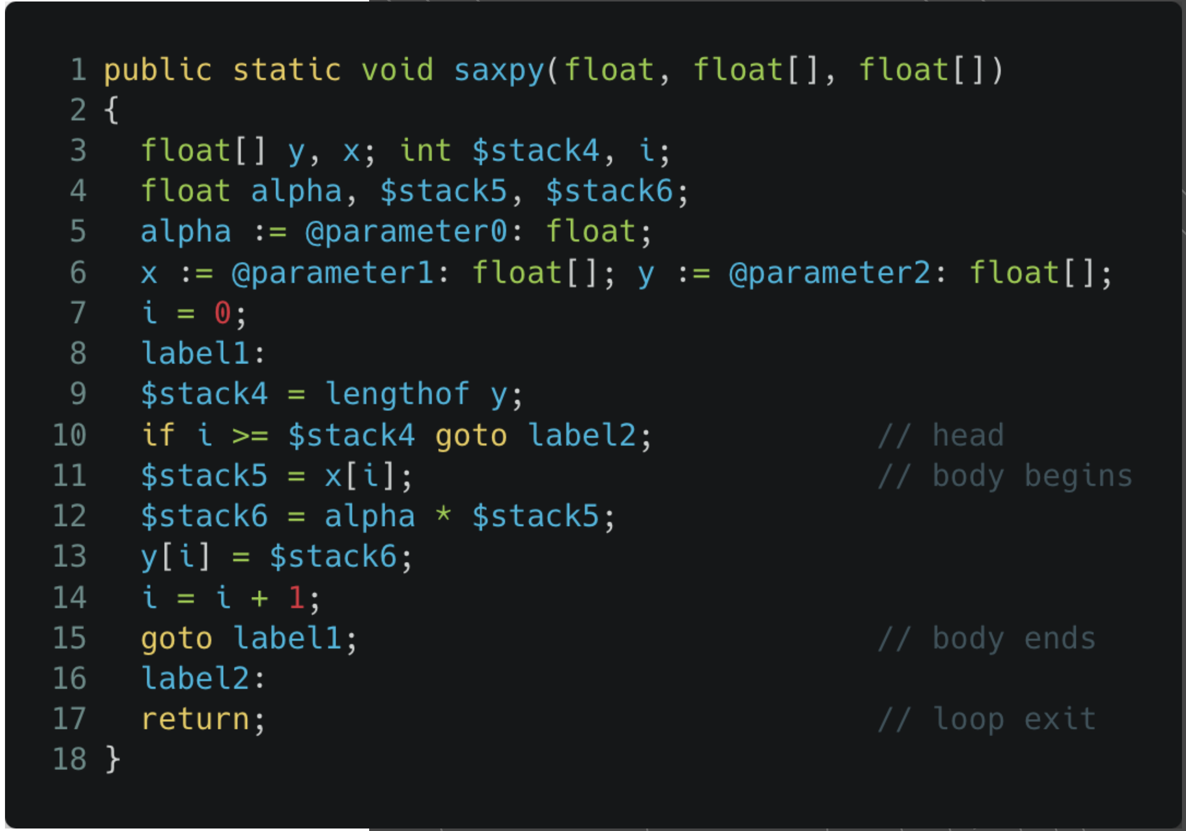
# The System

## Variable Scoping

| Start | Length | Slot | Name | Signature |
|---|---|---|---|---|
| 2 | 20 | 3 | **i** | I |
| 0 | 23 | 0 | **alpha** | F |
| 0 | 23 | 1 | **x** | [F |
| 0 | 23 | 2 | **y** | [F |

Compilation: `javac -g Saxpy.java`

View: `javap -p -v  Saxpy`



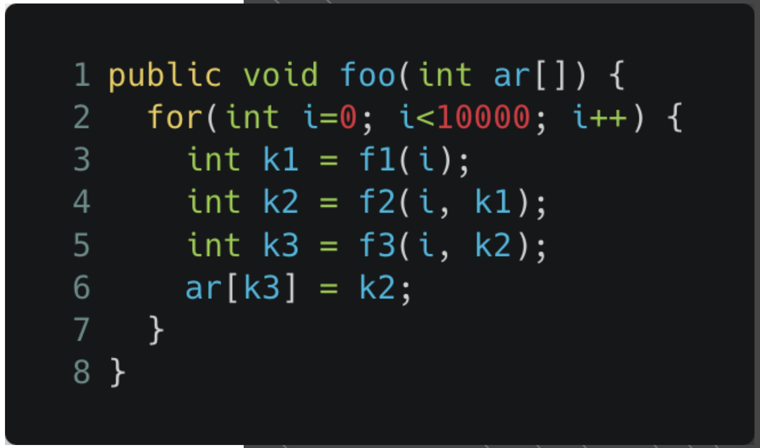
```
public static void saxpy(float, float[], float[])
{
  float[] y, x; int $stack4, i;
  float alpha, $stack5, $stack6;
  alpha := @parameter0: float;
  x := @parameter1: float[]; y := @parameter2: float[];
  i = 0;
  label1:
  $stack4 = lengthof y;
  if i >= $stack4 goto label2;            // head
  $stack5 = x[i];                         // body begins
  $stack6 = alpha * $stack5;
  y[i] = $stack6;
  i = i + 1;
  goto label1;                            // body ends
  label2:
  return;                                 // loop exit
}
```

# The System

## Dependence Analysis

### Array Indexing

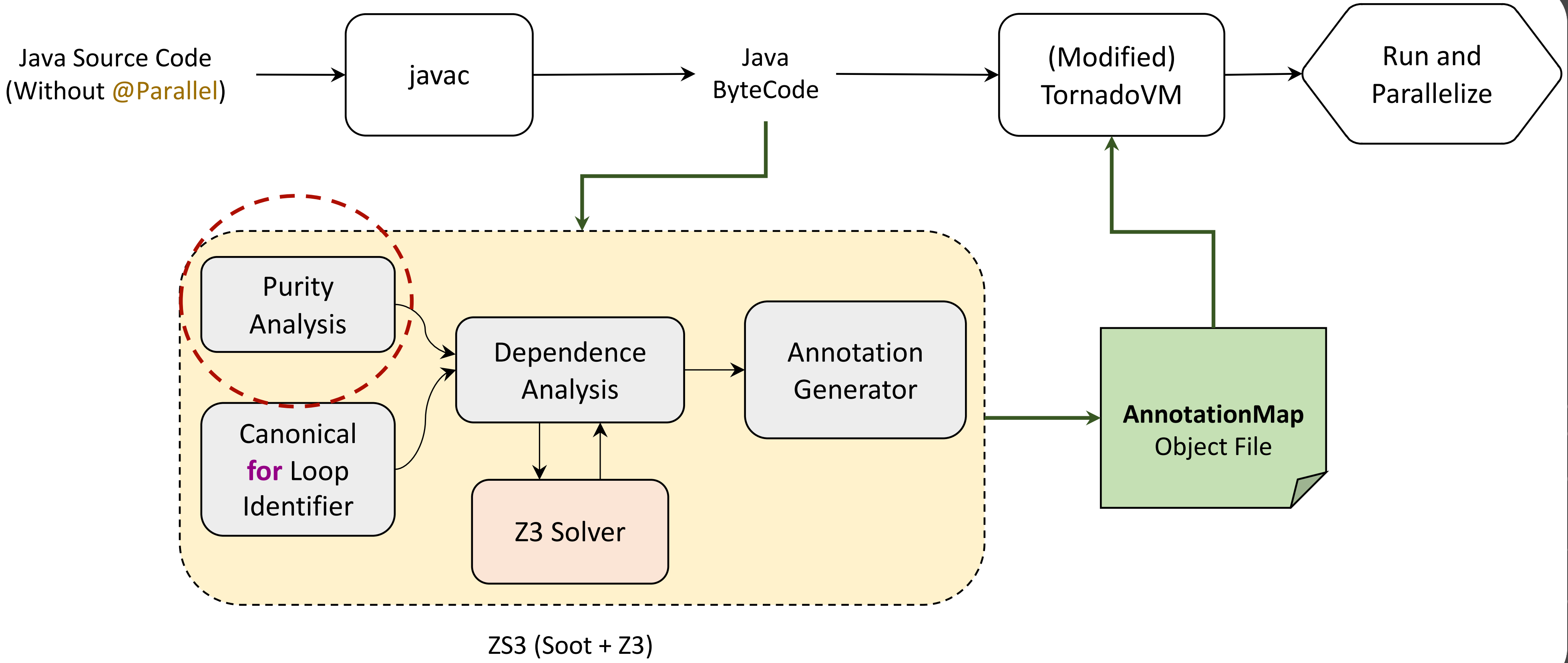
- Multiple array references in loop: PointsToAnalysis.
- For each array-write, check the dependence with all aliasing array-write and array-read.
- Using Z3! Encoding the program into logic: Def Chains.



```java
public void foo(int ar[]) {
  for(int i=0; i<10000; i++) {
    int k1 = f1(i);
    int k2 = f2(i, k1);
    int k3 = f3(i, k2);
    ar[k3] = k2;
  }
}
```

$$(k3^u == f3(i^u, k2^u)) \land (k2^u == f2(i^u, k1^u)) \land (k1^u == f1(i^u)) \land$$
$$(i^u \geq 0) \land (i^u < 10000) \land$$
$$(k3^v == f3(i^v, k2^v)) \land (k2^v == f2(i^v, k1^v)) \land (k1^v == f1(i^v)) \land$$
$$(i^v \geq 0) \land (i^v < 10000) \land$$
$$(i^u \neq i^v) \land (k3^u == k3^v)$$

# The System


Java Source Code
(Without @Parallel)
javac
Java
ByteCode
(Modified)
TornadoVM
Run and
Parallelize
Purity
Analysis
Canonical
for Loop
Identifier
Dependence
Analysis
Annotation
Generator
Z3 Solver
AnnotationMap
Object File
ZS3 (Soot + Z3)

# The System

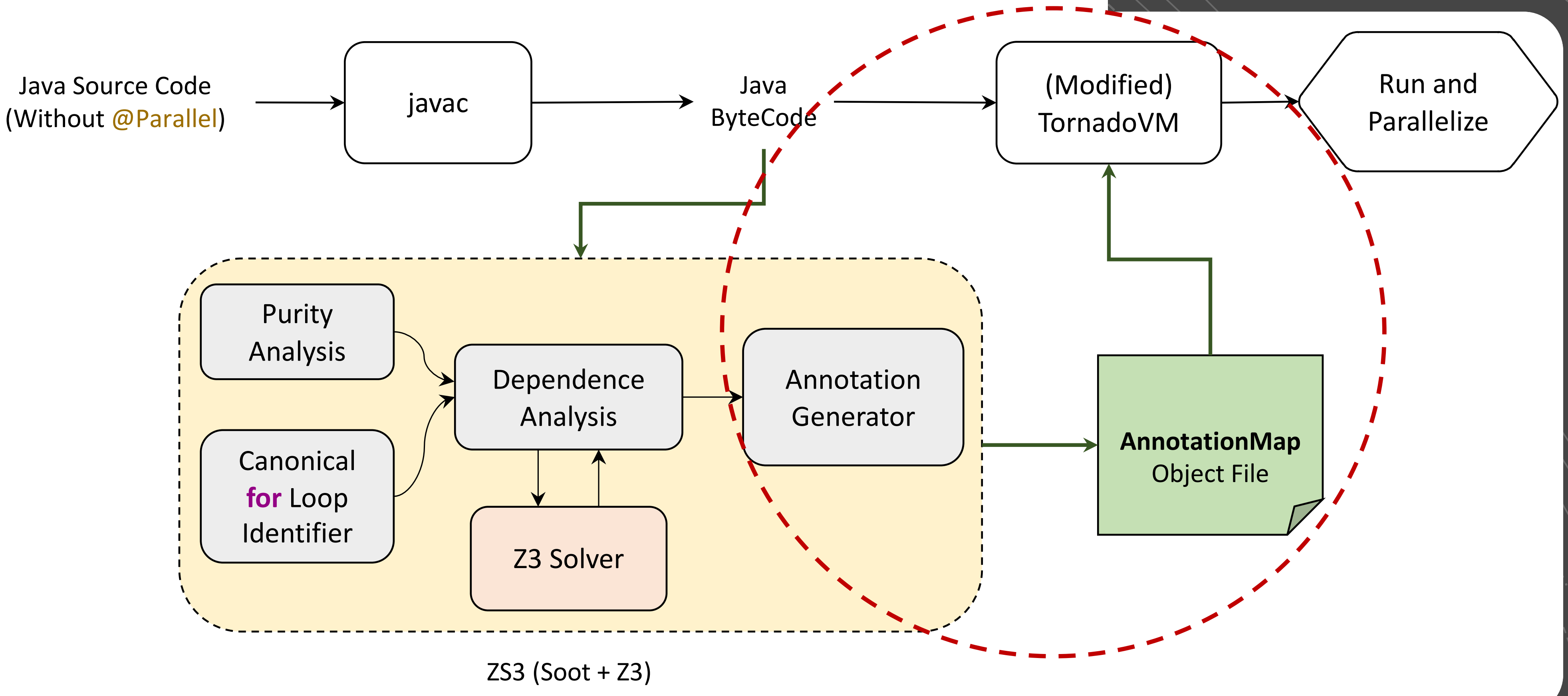
ZS3

## Purity Analysis

*A method is pure if it does not mutate any location that exists in the program state right before method invocation*

- Impure function call ⇒ Non-parallelizable.
- Sources of impurity:
    - Static field references
    - References to pre-existing objects
    - Impure function calls

# The System


Java Source Code
(Without @Parallel)
javac
Java
ByteCode
(Modified)
TornadoVM
Run and
Parallelize
Purity
Analysis
Canonical
for Loop
Identifier
Dependence
Analysis
Z3 Solver
Annotation
Generator
AnnotationMap
Object File
ZS3 (Soot + Z3)

# Evaluation

ZS3

## Research Question 1

How many of manually parallelized loops are marked as parallelizable by ZS3?

| | Marked Parallelizable | Marked Non-Parallelizable |
|---|---|---|
| Parallelizable | 61.3% | 38.7% |
| Non-Parallelizable | 0% | 100% |

No False-Positives!

# Evaluation

## Research Question 2

Are the overheads of static-analysis, those of storing the $AnnotationMap$, and the time spent in the VM significant?

| Name | Analysis Time (s) | Class File Size (B) | AnnotationMap Size (B) | Size Overhead % | Parallel Runtime (ms) | AnnotationMap Read Time (ms) |
|---|---:|---:|---:|---:|---:|---:|
| GeoMean | 1.22 | 4647.22 | 516.09 | 11.43 | 479.48 | 11.44 |
| Max | 4 | 7609.00 | 886.00 | 15.84 | 1546.23 | 25.99 |

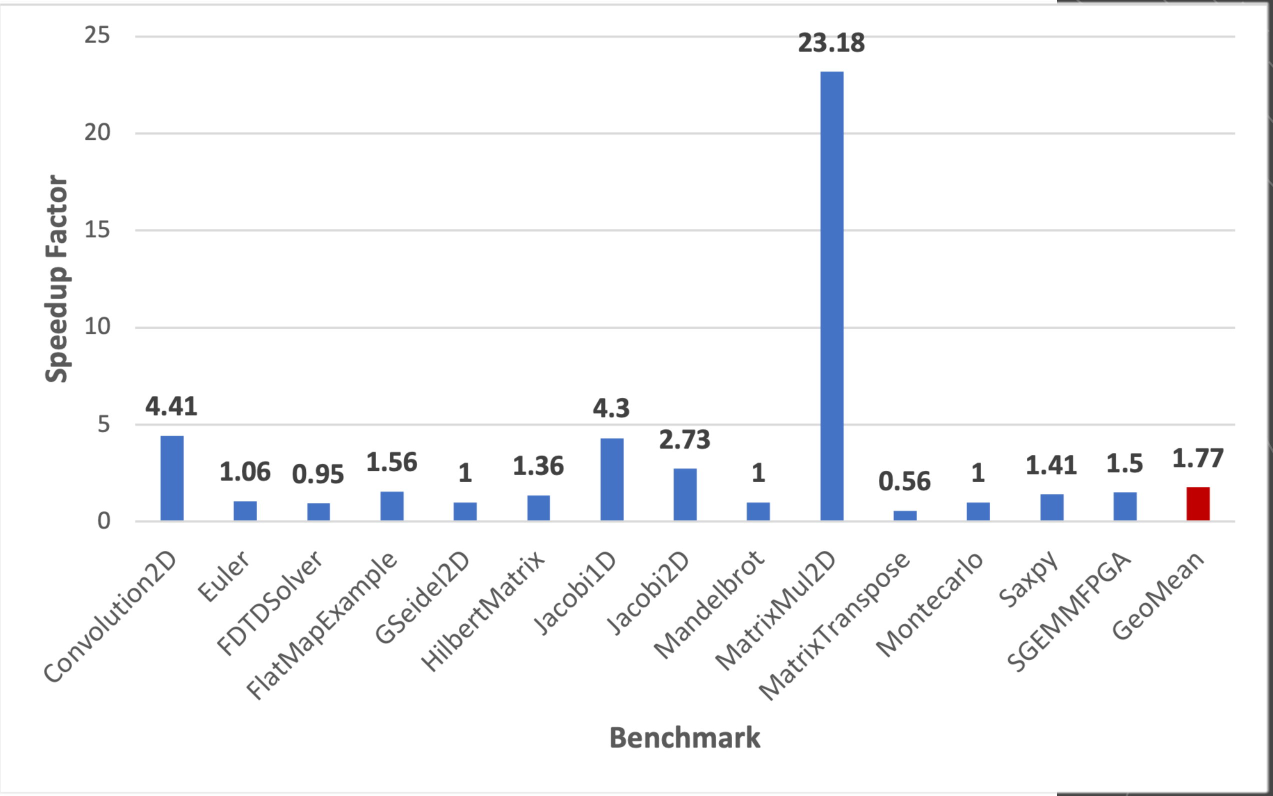
Negligible overheads

# Evaluation

ZS3

## Research Question 3

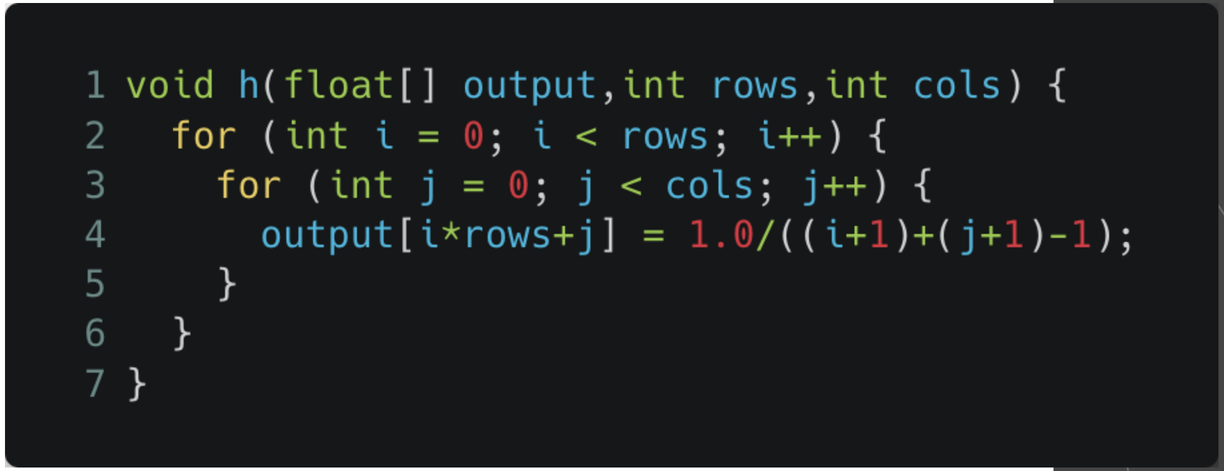
How good are the speedups of ZS3-marked parallel loops?



# Evaluation

## Research Question 4

What are the challenges yet to be handled by future static-analysis guided loop parallelizers?



```
void h(float[] output,int rows,int cols) {
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      output[i*rows+j] = 1.0/((i+1)+(j+1)-1);
    }
  }
}
```

when $j >= rows$, for $i^u = 0$ and $i^v = 1$

# Takeaways

- First of its kind loop parallelizer for managed runtimes.
- Integration of Z3 with Soot ⇒ New possibilities.
- Precision can still be improved by integrating other techniques.

# Thank You