Aspects in Android: Usability and Performance

What is aspect-oriented programming

Affecting Applications in Android Using Aspects

[illegible]

Usability of AspectJ from the Performance Perspective

[illegible]

Valentino Vranić
Institute of Informatics and Software Engineering
STU FIIT SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
vranic@stuba.sk
http://fiit.sk/~vranic/

```
public class Point {
    private int x;
    private int y;

    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }
    public int getX() { return x; }
    public int getY() { return y; }
}
```

```
public aspect RangeControl {
    void around(int x): call(void Point.setX(..)) && args(x) {
        if (x < 0)
            proceed(640 + x % 640);
        else if (x > 640)
            proceed(x % 640);
        else
            proceed(x);
    }
}
```

A more advanced aspect

```
public aspect SomeAspect {
    void around(): call(void My*.main(*)) {
        invokeQueue.add(new Runnable() {
            public void run() {
                proceed();
            }
        }); // calls captured and sent to some queue
    }
}
```

- The Worker Object Creation object-oriented design pattern

AOP in AspectJ

- Not only calls, but executions, access to attributes, or even control flows can be captured
- Aspects can introduce new attributes and methods
- Aspects can be used to mechanize changes, which is very useful in customization
- All this is so-called asymmetric aspect-oriented programming (AOP)
- There's much more to AOP...
- Separation of crosscutting concerns
- Advanced modularization

Can aspect-oriented programming be utilized in Android?

[illegible]

Take the GPS sensor in terms of

[illegible]

Altering the GPS sensor output

[illegible]

What else can be done with aspects in Android?

[illegible]

Even more to be done with aspects in Android

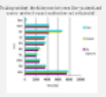
[illegible]

Our performance measurement framework (1)

- https://github.com/[illegible]/AspectJ-Performance-Measurement-Framework
- Ackermann function calculation (deep recursion)
- Fibonacci sequence calculation (branching recursion)
- Large matrix computation (matrix operations)
- Nested loop execution (loop handling)
- Random generation of double numbers (random generation)
- Prime numbers calculation (arithmetic operations)
- Text string concatenation (working with string values)
- Read of a large text file (working with I/O)
- Quicksort algorithm (sorting)
- Object instantiation (memory allocation)

Our performance measurement framework (2)

- Targeting the overhead coming from the very invocation of aspects or, more precisely, advices (before, after, and around)
- Coarse fine grained aspect application
- Tests have been performed repeatedly big number of times to decrease the imprecision of the System.nanoTime() method
- Desktop and mobile setting



Mobile setting

- Android mobile device with the ART virtual machine with a clean Android installation (9.0.0)
- Slightly altered original test suite for performance measure

Findings

- High use of aspects causes more significant performance overhead in mobile devices compared to desktop devices
- It pays off to apply aspects rather to a small number of high time complexity methods than to a large number of low time complexity methods
- The before advice generates less performance overhead than the after advice (on both mobile and desktop devices); the around advice generates the biggest performance overhead

Aspects in Android: Usability and Performance

What is aspect-oriented programming

Affecting Applications in Android Using Aspects

Ivan Martoš and Valentino Vranić

Usability of AspectJ from the Performance Perspective

Erik Šuta, Ivan Martoš, and Valentino Vranić

Valentino Vranić

Institute of Informatics and Software Engineering

STU FIIT SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

vranic@stuba.sk

http://fiit.sk/~vranic/

Aspects in Android: Usability and Performance

What is aspect-oriented programming

Affecting Applications in Android Using Aspects

Ivan Martoš and Valentino Vranić

Usability of AspectJ from the Performance Perspective

Erik Šuta, Ivan Martoš, and Valentino Vranić

What is aspect-oriented programming

A

```
public class Point {
    private int x;
    private int y;

    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }
    public int getX() { return x; }
    public int getY() { return y; }
}
```

```
public aspect RangeControl {
    void around(int x): call(void Point.setX(..)) && args(x) {
        if (x < 0)
            proceed(640 + x % 640);
        else if (x > 639)
            proceed(x % 640);
        else
            proceed(x);
    }
}
```

A more advanced aspect

```
public aspect SomeAspect {
    void around(): call(void My*.make*()) {
        invoke.Queue.add(new Runnable () {
            public void run() {
                proceed();
            }
        }); // calls captured and sent to some queue
    }
}
```

- The Worker Object Creation aspect-oriented design pattern

AOP in AspectJ

- Not only calls, but executions, access to attributes, or even control flows can be captured

- Aspects can introduce new attributes and methods

- Aspects can be used to modularize changes, which is very useful in customization

- All this is so-called asymmetric aspect-oriented programming (AOP)

- There's much more to AOP...

- Separation of crosscutting concerns

- Advanced modularization

Affecting Applications in Android Using Aspects

Ivan Martoš and Valentino Vranić

lroid:

ed

How aspect-oriented programming can be utilized in Android?

- General, application dependent application of AOP holds for mobile applications in Android, too

- An adapted build cycle is necessary in order to utilize AspectJ under Android

- Calls to Android API can be captured and affected by aspects

- Aspects can’t modify permissions that application has declared in its manifest file

Fake the GPS sensor is turned on

```
boolean around(String provider):
    call(boolean android.location.LocationManager.
        isProviderEnabled(..)) && args(provider) {

    //Additional logic...
    return true;

}
```

Altering the GPS sensor output

- GPS consumes lots of energy
- The GSM provider's location service can be used instead
- This can be managed with an aspect that modifies location update requests

```
void around(String provider, long timeChange, float distChange,
    LocationListener listener):
    call(void android.location.LocationManager. requestLocationUpdates(
    String, long, float, android.location.LocationListener))
    && args(provider, timeChange, distChange, listener) {

    provider = LocationManager.NETWORK_PROVIDER;
    proceed(provider, timeChange, distChange, listener);

}
```

What else can be done with aspects in Android?

- Add notifications to application at any place

- Affect the Context class

- Monitor customs and routines of users (e.g., in order to increase the battery life)

- By using aspects it is possible to add notifications to application at any place

- Alter, affect, or even disable sensors

Even more to be done with aspects in Android

- Provide added functionality

- Use cases can be preserved in source code by aspects

- Disable advertisement by affecting the com.google.ads package with appropriate aspects

- Note: to disable bypassing by other applications, affect system calls, not application calls

Usability of AspectJ from the Performance Perspective

Erik Šuta, Ivan Martoš, and Valentino Vranić

Our performance measurement framework (1)

- https://github.com/eriksuta/AspectJ-Performance-Measurement-Framework
- Ackermann function calculation4 (deep recursion)
- Fibonacci sequence calculation (branching recursion)
- Large matrix computation (matrix operations)
- Nested loop execution (loop handling)
- Random generation of double numbers (random generation)
- Prime numbers calculation (arithmetic operations)
- Vast string concatenation (working with string values)
- Read of a long text file (working with I/O)
- Quicksort algorithm (sorting)
- Object instantiation (memory allocation)

Our performance measurement framework (2)

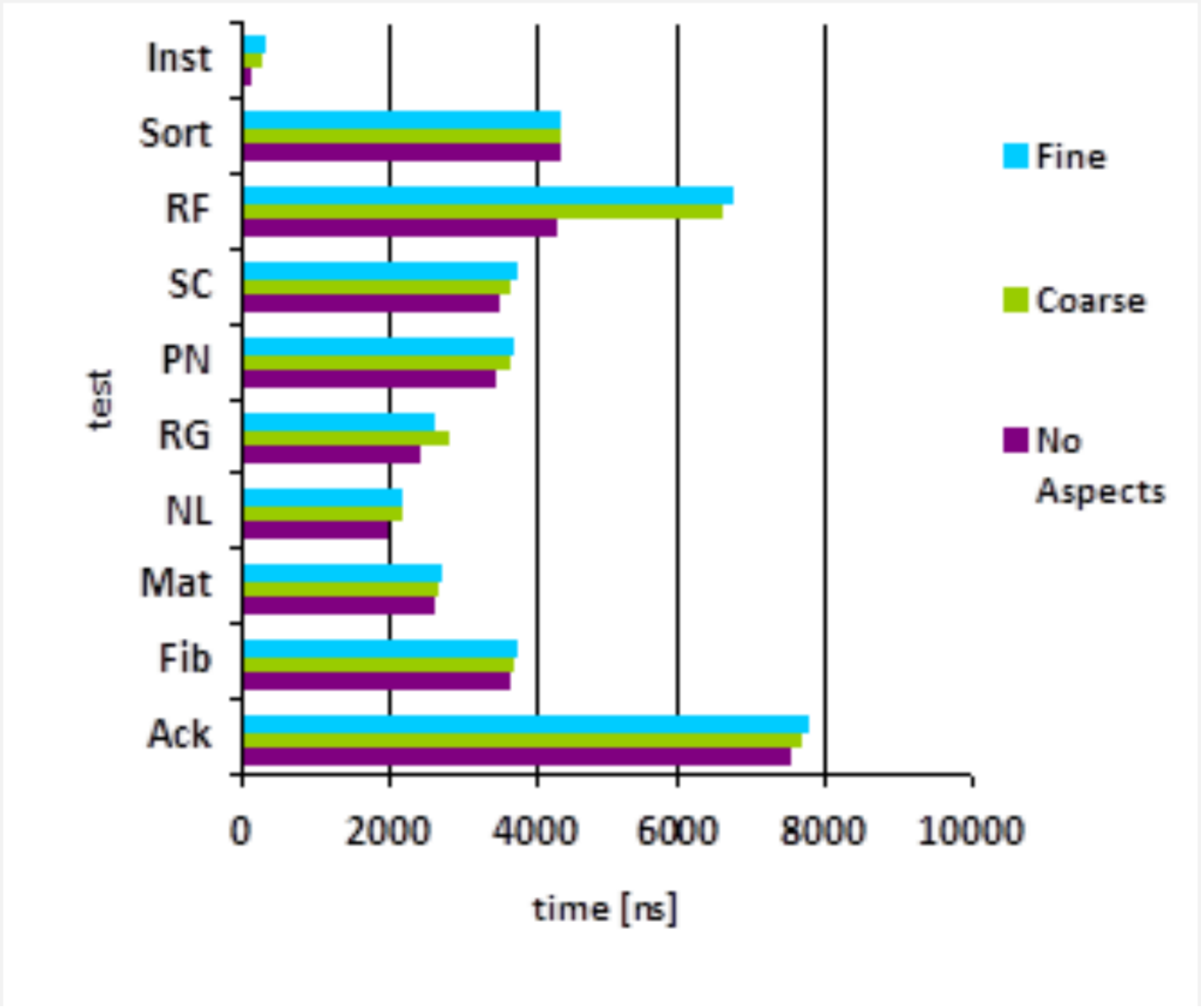
- Targeting the overhead coming from the very invocation of aspects or, more precisely, advices (before, after, and around)

- Coarse/fine grained aspect application

- Tests have been performed repeatedly big number of times to decrease the imprecison of the System.nanoTime() method

- Desktop and mobile setting

Desktop setting: the difference between fine-grained and coarse-grained aspect application not substantial

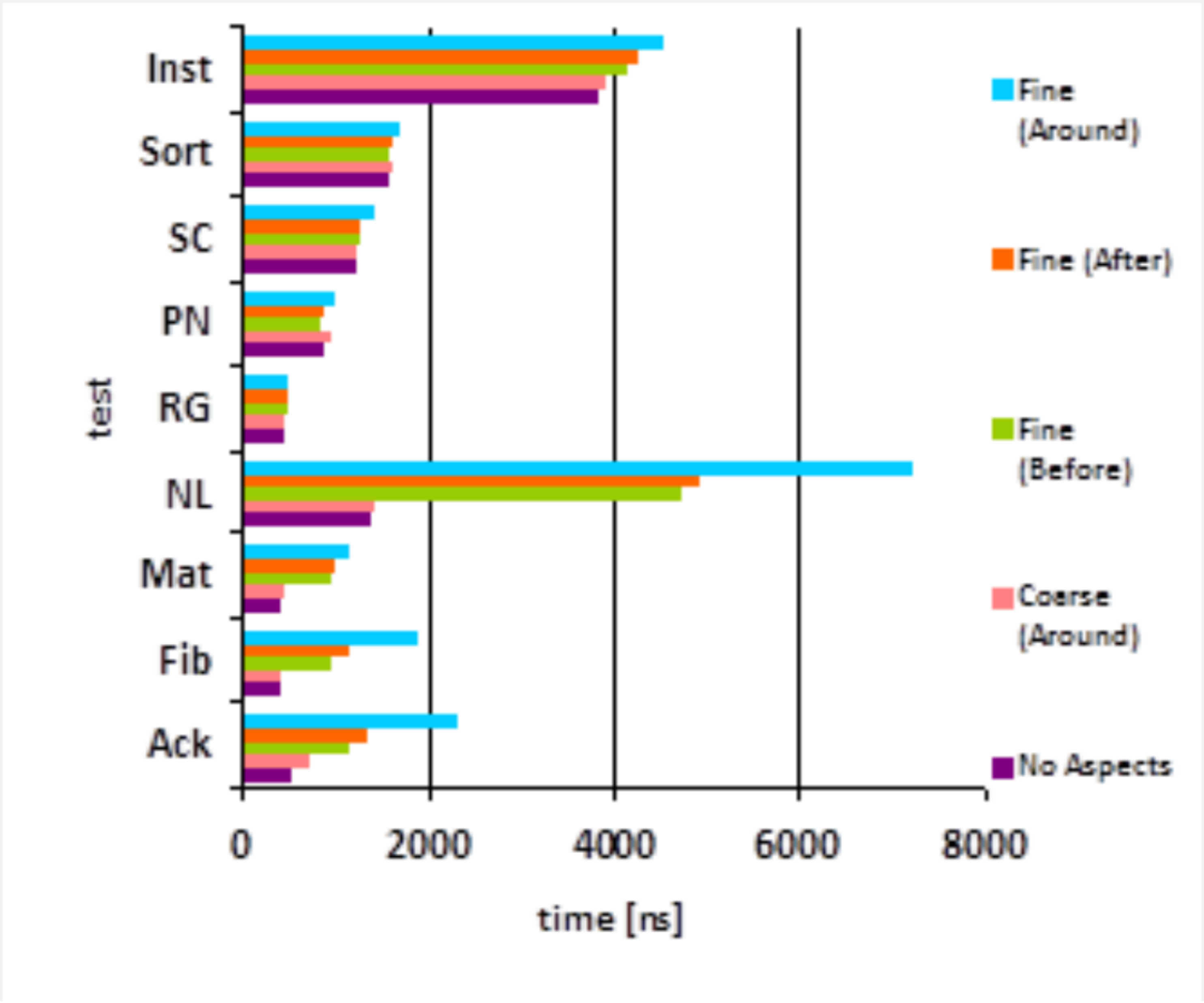


Mobile setting

- Android mobile device with the ART virtual machine with a clean Android installation (5.0.1)

- Slightly altered original test suite for performance reasons

Mobile setting (Android mobile device with AspectJ version 1.7.3): coarse-grained aspect application performs better



Findings

- Rich use of aspects causes more significant performance overhead in mobile devices compared to desktop devices

- It pays off to apply aspects rather to a small number of high time complexity methods than to a large number of low time complexity methods

- The before advice generates less performance overhead than the after advice (on both mobile and desktop devices); the around advice generates the biggest performance overhead

Aspects in Android: Usability and Performance

What is aspect-oriented programming

Affecting Applications in Android Using Aspects

Ivan Martoš and Valentino Vranić

Usability of AspectJ from the Performance Perspective

Erik Šuta, Ivan Martoš, and Valentino Vranić

Valentino Vranić

Institute of Informatics and Software Engineering

STU FIIT SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

vranic@stuba.sk

http://fiit.sk/~vranic/