**FLEXJAVA: Language Support for Safe and Modular Approximate Programming**

**Jongse Park**, Hadi Esmaeilzadeh, Xin Zhang, Mayur Naik, William Harris

Alternative Computing Technologies (ACT) Lab
Georgia Institute of Technology

ESEC/FSE 2015
Energy is a primary constraint

Data Center

Mobile

Internet of Things
Data growth vs performance

Adding a third dimension
Embracing Error

Energy

Performance

Error

IoT

Mobile

Desktop

Data center
Navigating a three dimensional space
Approximate computing
Embracing error

Relax the abstraction of “near perfect” accuracy in

- Processing
- Storage
- Communication

Allows errors to happen to improve performance
resource utilization efficiency
New landscape of computing
Personalized and targeted computing
Classes of approximate applications

Programs with analog inputs
  • Sensors, scene reconstruction

Programs with analog outputs
  • Multimedia

Programs with multiple possible answers
  • Web search, machine learning

Convergent programs
  • Gradient descent, big data analytics
Avoiding overkill design
Approximate Computing

Efficiency
Performance

Cost
Precision
Reliability

Cost

Application
Programming Language
Compiler
Architecture
Microarchitecture
Circuit
Physical Device
Cross-stack approach
$WH^2$ of Approximation

**What** to approximate?

**How much** to approximate?

**How** to approximate?

<table>
<thead>
<tr>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
</tr>
<tr>
<td>Runtime</td>
</tr>
<tr>
<td>Hardware</td>
</tr>
</tbody>
</table>
You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.

Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de redémarrage.

Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.

コンピュータを再起動する必要があります。電源ボタンを数秒間押し続けるか、リセットボタンを押してください。
Approximate program
Separation
Goal

Design an automated and high-level programming language for approximate computing

Criteria

1) Safety
2) Scalability
3) Modularity and reusability
4) Generality
**FlexJava**

Lightweight set of language extensions
Reducing annotating effort
Language-compiler co-design

- Safety
- Scalability
- Modularity and reusability
- Generality
**FLEXJAVA Annotations**

1. Approximation for individual data and operations
   - `loosen` (variable)
   - `loosen_invasive` (variable)
   - `tightly` (variable)

2. Approximation for a code block
   - `begin_loose` (“TECHNIQUE”, ARGUMENTS)
   - `end_loose` (ARGUMENTS)

3. Expressing the quality requirement
   - `loosen` (variable, QUALITY_REQUIREMENT);
   - `loosen_invasive` (variable, QUALITY_REQUIREMENT);
   - `end_loose` (ARGUMENTS, QUALITY_REQUIREMENT);
Safe and scalable approximation

```cpp
float computeLuminance (float r,
                        float g,
                        float b) {

    float luminance = r * 0.3f + g * 0.6f + b * 0.1f;

    loosen (luminance);

    return luminance;
}
```
Safe and scalable approximation

```c
float computeLuminance (float r,
                        float g,
                        float b) {
    float luminance = r * 0.3f + g * 0.6f + b * 0.1f;
    loosen (luminance);
    return luminance;
}
```
float computeLuminance (float r, float g, float b) {
    float luminance = r * 0.3f + g * 0.6f + b * 0.1f;
    loosen (luminance);
    return luminance;
}
Safe and scalable approximation

float computeLuminance(float r, float g, float b) {
    float luminance = r * 0.3f + g * 0.6f + b * 0.1f;
    loosen(luminance);
    return luminance;
}
Safe and scalable approximation

```c
int fibonacci (int n) {
    int r;
    if (n <= 1)
        r = n;
    else
        r = fibonacci (n - 1) +
            fibonacci (n - 2);

    loosen (r);
    return r;
}
```
Safe and scalable approximation

```c
int fibonacci (int n) {
    int r;
    if (n <= 1)
        r = n;
    else
        r = fibonacci (n - 1) + fibonacci (n - 2);
    loosen (r);
    return r;
}
```
Modularity

```java
int p = 1;
for (int i = 0; i < a.length; i++) {
    p *= a[i];
}
for (int i = 0; i < b.length; i++) {
    p += b[i];
    loosen(p);
}
```
Reuse and library support

```java
static int square(int a) {
    int s = a * a;
    loosen(s);
    return s;
}

main () {
    int x = 2 * square(3);
    System.out.println(x);
}
```
Reuse and library support

```java
static int square(int a) {
    int s = a * a;
    loosen(s);
    return s;
}

main () {
    int x = 2 * square(3);
    loosen(x);
    System.out.println(x);
}
```
Reuse and library support

```java
static int square(int a) {
    int s = a * a;
    loosen(s);
    return s;
}

main () {
    int x = 2 * square(3);
    loosen(x);
    System.out.println(x);
}
```
Reuse and library support

```java
static int square(int a) {
    int s = a * a;
    loosen(s);
    return s;
}

main () {
    int x = 2 * square(3);
    loosen(x);
    System.out.println(x);
}
```
Reuse and library support

```java
static int square(int a) {
    int s = a * a;
    return s;
}

main () {
    int x = 2 * square(3);
    loosen(x);
    System.out.println(x);
}
```
Reuse and library support

```java
static int square(int a) {
    int s = a * a;
    return s;
}

main () {
    int x = 2 * square(3);
    loosen(x);
    System.out.println(x);
}
```
reuse and library support

```java
static int square(int a) {
    int s = a * a;
    return s;
}

main () {
    int x = 2 * square(3);
    loosen_invasive(x);
    System.out.println(x);
}
```
Reuse and library support

```c
static int square(int a) {
    int s = a * a;
    return s;
}

main () {
    int x = 2 * square(3);
    loosen_invasive(x);
    System.out.println(x);
}
```
Reuse and library support

```java
static int square(int a) {
    int s = a * a;
    return s;
}

main () {
    int x = 2 * square(3);
    loosen_invasive(x);
    System.out.println(x);
}
```
static int square(int a) {
    int s = a * a;
    return s;
}

main () {
    int x = 2 * square(3);
    loosen_invasive(x);
    System.out.println(x);
}
Reuse and library support

```java
static int square(int a) {
    int s = a * a;
    return s;
}

main () {
    int x = 2 * square(3);
    loosen_invasive(x);
    System.out.println(x);
}
```
## Generality

<table>
<thead>
<tr>
<th>Software</th>
<th>Loop perforation and loop early-termination [ESEC/FSE'11, PLDI'10]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Function substitution [PLDI'10]</td>
</tr>
<tr>
<td>Hardware</td>
<td>Truffle: Architecture support for approximate computing [ASPLOS'12]</td>
</tr>
<tr>
<td></td>
<td>NPU: Neural acceleration [MICRO'12]</td>
</tr>
<tr>
<td>Memory</td>
<td>Flikker: Saving DRAM refresh power [ASPLOS'11]</td>
</tr>
<tr>
<td></td>
<td>RFVP and load value prediction [PACT'14, MICRO'14]</td>
</tr>
</tbody>
</table>
Generality

\begin{verbatim}
begin_loose("PERFORATION", 0.10);
for (int i = 0; i < n; i++) {
    ...
}
end_loose();
\end{verbatim}
Generality

\begin{align*}
\text{begin\_loose(”NPU”);} \\
p &= \text{Math.sin}(x) + \text{Math.cos}(y) \\
q &= 2 \times \text{Math.sin}(x + y) \\
\text{end\_loose();}
\end{align*}
Approximation Safety Analysis

Find the maximal set of safe-to-approximate data/operations

1. For each annotation, $\text{loosen}(v)$, find the set of data and operations that affect $v$
2. Merge all the sets
3. Exclude the data and operations that affect control flow
4. Exclude the data and operations that affect address calculations

The rest of the program data and operations are precise
Workflow
## Benchmark

<table>
<thead>
<tr>
<th>Name</th>
<th>SciMark2 Benchmark</th>
<th>Description</th>
<th>Code Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>sor</td>
<td># lines: 36</td>
<td>Successive Over-relaxation</td>
<td>36</td>
</tr>
<tr>
<td>smm</td>
<td># lines: 38</td>
<td>Matrix-vector multiplication</td>
<td>38</td>
</tr>
<tr>
<td>mc</td>
<td># lines: 59</td>
<td>Mathematical approximation</td>
<td>59</td>
</tr>
<tr>
<td>fft</td>
<td># lines: 168</td>
<td>Signal processing</td>
<td>168</td>
</tr>
<tr>
<td>lu</td>
<td># lines: 283</td>
<td>Matrix factorization</td>
<td>283</td>
</tr>
<tr>
<td>sobel</td>
<td># lines: 163</td>
<td>Image processing</td>
<td>163</td>
</tr>
<tr>
<td>raytracer</td>
<td># lines: 174</td>
<td>Graphics</td>
<td>174</td>
</tr>
<tr>
<td>jmeint</td>
<td># lines: 5,962</td>
<td>3D gaming</td>
<td>5,962</td>
</tr>
<tr>
<td>lu</td>
<td># lines: 283</td>
<td>Matrix factorization</td>
<td>283</td>
</tr>
<tr>
<td>hessian</td>
<td># lines: 10,174</td>
<td>Image processing</td>
<td>10,174</td>
</tr>
<tr>
<td>zxing</td>
<td># lines: 26,171</td>
<td>Image processing</td>
<td>26,171</td>
</tr>
</tbody>
</table>
Number of Annotations

- **EnerJ**: Blue bars
- **FLEXJAVA**: Green bars

**Benchmarks**:
- sor
- smm
- mc
- fft
- lu
- sobel
- raytracer
- jmeint
- hessian
- zxing

Lower is Better

2× to 17× reduction in the number of annotations
User-study: Annotation Time

10 subjects (programmers)
Time for annotating fft using **EnerJ** and **FLEXJAVA**

![Bar chart showing annotation time for subjects using EnerJ and FLEXJAVA. Lower is Better.]

**8x** average reduction in annotation time
FLEXJAVA

Language-compiler **co-design** to

**Automate** approximate programming

**Reduce** programmer effort

- Safety
- Scalability
- Modularity and reusability
- Generality
Replication Package

http://act-lab.org/artifacts/flexjava/

1. Annotated benchmarks
2. Compiler with approximate safety analysis
3. Source code highlighting tool

Open source git repository:
https://bitbucket.org/act-lab/flexjava.code