Universidade Federal de Pelotas
Ciência da Computação
Grupo de Arquiteturas e Circuitos Integrados

Avaliação das boas práticas Android para desempenho

Aline Tonini, Marco Beckmann, Julio Mattos, Lisane Brisolara

Apoio:
FAPERGS

Sumário

- Introdução
- Boas Práticas Android
- Avaliação das boas práticas
  - Metodologia
  - Experimentos/Resultados
- Conclusões e Trabalhos Futuros

Introdução

- Alternativa para o uso de computadores pessoais
- Facilidade de uso
- Crescente demanda por novos aplicativos
- Existem várias plataformas, dentre elas a **Android**

Introdução

- Android
  - Plataforma aberta desenvolvida pela Google
  - Possui ferramentas de apoio ao desenvolvedor
  - Usa Java como linguagem de programação
  - Possui conexão com os serviços do Google

Introdução

- Desenvolver para dispositivos móveis possui restrições
- A aplicação deve **otimizar** os recursos disponíveis
- O **desempenho** é um aspecto importante
- Melhorias no código tornam a aplicação mais **eficiente**

A Google sugere Boas Práticas de programação para Android com foco em desempenho!

Introdução

- Qual o impacto destas práticas no desempenho?

Objetivo: Avaliar o impacto das boas práticas através de experimentos

Boas Práticas

- Evitar a criação de objetos desnecessários
- Preferir métodos estáticos se não for acessar o campo de um objeto
- Usar *static final* para constantes de tipos primitivos e *Strings*
- Considerar acesso *package* em atributos acessados por classes internas privadas

Boas Práticas

- Evitar o uso de Ponto Flutuante
  - O Ponto Flutuante é cerca de duas vezes mais lento do que o Inteiro em Android
- Evitar o uso de Getters/Setters
  - O tempo de acesso ao atributo diretamente é cerca de três vezes mais rápido
- Utilizar a sintaxe aprimorada do *For*
  - A utilização do For-Each para coleções

Foco em duas práticas: for e getter/setters

Sumário

- Introdução
- Boas Práticas Android
- **Avaliação das boas práticas**
  - Metodologia
  - Experimentos/Resultados
- Conclusões e Trabalhos Futuros

Metodologia

- Avaliação do desempenho
  - Do código original
  - Do código aplicando as boas práticas

- Cada teste é executado 30 vezes → tempo médio de execução

- Comparação entre as médias através do teste estatístico “t de Student”

Metodologia

- Plataforma: Android 4.1.2 API 15

- Uso da **ferramenta DDMS** para trace do código
  - Métodos *starMethodTracing()* e *stopMethodTracing()*
  - Tempo de execução

Include CPU Time: Tempo gasto para executar o método somado com o tempo de execução dos métodos filhos

Exclude CPU Time: Tempo gasto para executar o método sem considerar o tempo de execução dos métodos filhos.

Experimento 1

- Código Experimental: **Getter/Setter**

```
public class Getter {
    int getter;

    int getGetter() {
        return getter;
    }
}

void withoutGetter(Getter get) {
    int i;
    i = get.getter;
}

void withGetter(Getter get) {
    int i;
    i = get.getGetter();
}
```

Chamada ao trace

```
Getter get = new Getter();
Debug.startMethodTracing("GetterWithGet",40000000);
for (int i = 0; i < 10000; i++)
    withGetter(get);
    //withoutGetter(get);
Debug.stopMethodTracing();
```

Usado Include CPU Time

Experimento 1

- Código Experimental: **Sintaxe do for**

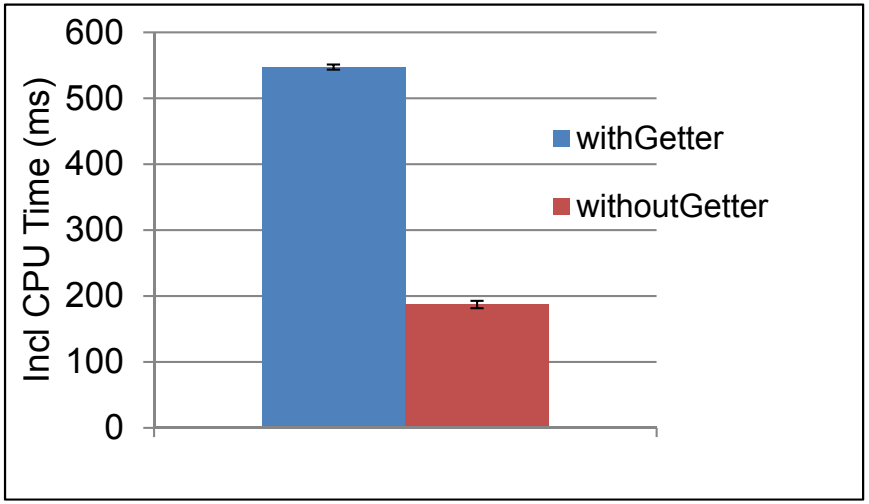
```
public void zero(FooLoop mArray[]) {
    int sum = 0;
    for (int i = 0; i < mArray.length; ++i) {
        sum += mArray[i].mSplat;
    }
}

public void one(FooLoop mArray[]) {
    int sum = 0;
    FooLoop[] localArray = mArray;
    int len = mArray.length;

    for (int i = 0; i < len; ++i) {
        sum += localArray[i].mSplat;
    }
}

public void two(FooLoop mArray[]) {
    int sum = 0;
    for (FooLoop a : mArray) {
        sum += a.mSplat;
    }
}
```

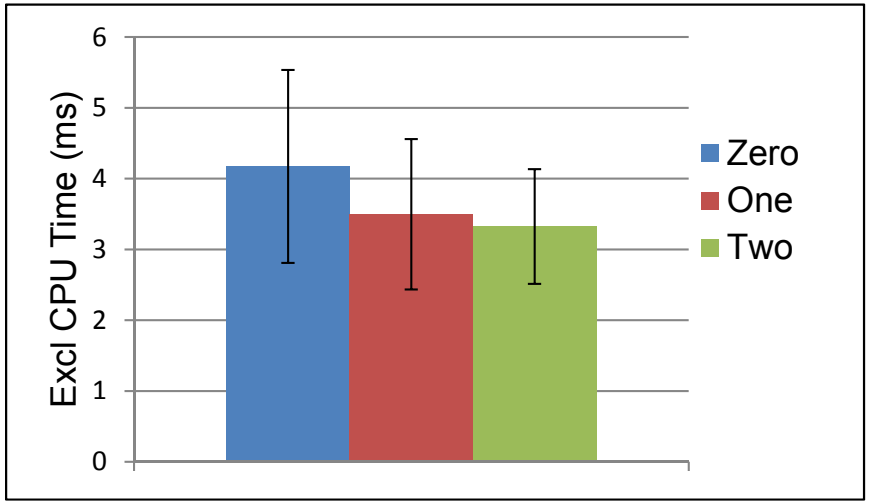
Chamada ao trace

```
Debug.startMethodTracing("ForZero", 40000000);
zero(mArray);
//one(mArray);
//two(mArray);
Debug.stopMethodTracing();
```

Experimento 1: Resultados

- Código Experimental
  - Getters/Setters
  - Uso da Melhor Sintaxe do For





65,81% mais rápido!

Two() foi 19,89% mais rápido que Zero() e 5% mais rápido que One()!

Experimento 2

- Aplicativo OpenSudoku: **Código original**

```
protected boolean validate() {
    boolean valid = true;
    Map<Integer, Cell> cellsByValue = new HashMap<Integer, Cell>();
    int len = mCells.length;
    Cell cell;
    int value;
    for (int i = 0; i < len; i++) {
        cell = mCells[i];
        value = cell.getValue();
        if (cellsByValue.get(value) != null) {
            mCells[i].setValid(false);
            cellsByValue.get(value).setValid(false);
            valid = false;
        } else {
            cellsByValue.put(value, cell);
        }
    }
    return valid;
}
```

Experimento 2

- Aplicativo OpenSudoku: **Código original**

```
protected boolean validate() {
    boolean valid = true;
    Map<Integer, Cell> cellsByValue = new HashMap<Integer, Cell>();
    int len = mCells.length;
    Cell cell;
    int value;
    for (int i = 0; i < len; i++) {
        cell = mCells[i];
        value = cell.getValue();
        if (cellsByValue.get(value) != null) {
            mCells[i].setValid(false);
            cellsByValue.get(value).setValid(false);
            valid = false;
        } else {
            cellsByValue.put(value, cell);
        }
    }
    return valid;
}
```

Experimento 2

- Aplicativo OpenSudoku: **Getters/Setters**

```
protected boolean validate() {
    boolean valid = true;
    int len = mCells.length;
    Cell cell;
    int value;
    Map<Integer, Cell> cellsByValue = new HashMap<Integer, Cell>();
    for (int i = 0; i < len; i++) {
        cell = mCells[i];
        value = cell.mValue;
        if (cellsByValue.get(value) != null) {
            mCells[i].mValid=false;
            cellsByValue.get(value).mValid=false;
            valid = false;
        } else {
            cellsByValue.put(value, cell);
        }
    }
    return valid;
}
```

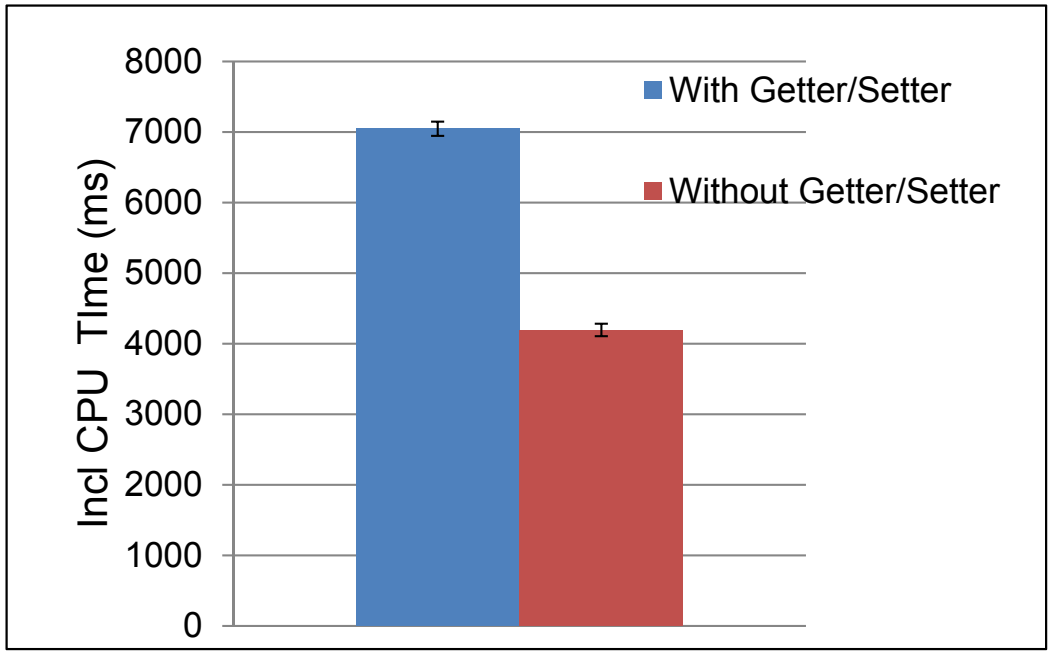
Experimento 2

- Aplicativo OpenSudoku: **Getters/Setters**

```
protected boolean validate() {
    boolean valid = true;
    int len = mCells.length;
    Cell cell;
    int value;
    Map<Integer, Cell> cellsByValue = new HashMap<Integer, Cell>();
    for (int i = 0; i < len; i++) {
        cell = mCells[i];
        value = cell.mValue;
        if (cellsByValue.get(value) != null) {
            mCells[i].mValid=false;
            cellsByValue.get(value).mValid=false;
            valid = false;
        } else {
            cellsByValue.put(value, cell);
        }
    }
    return valid;
}
```

Experimento 2: Resultados

- Aplicativo OpenSudoku
  - Código Original x **Não uso de Getters/Setters**



Com Getter/Setter:
7046,0340 ms

Sem Getter/Setter:
4195,0565 ms

40,46% mais rápido!

Uso do Include CPU Time

Experimento 2

- Aplicativo OpenSudoku: **Sintaxe do For**

```
protected boolean validate() {
    boolean valid = true;
    Cell cell;
    int value;
    Map<Integer, Cell> cellsByValue = new HashMap<Integer, Cell>();
    for (Cell a:mCells) {
        cell = a;
        value = cell.getValue();
        if (cellsByValue.get(value) != null) {
            a.setValid(false);
            cellsByValue.get(value).setValid(false);
            valid = false;
        } else {
            cellsByValue.put(value, cell);
        }
    }
    return valid;
}
```

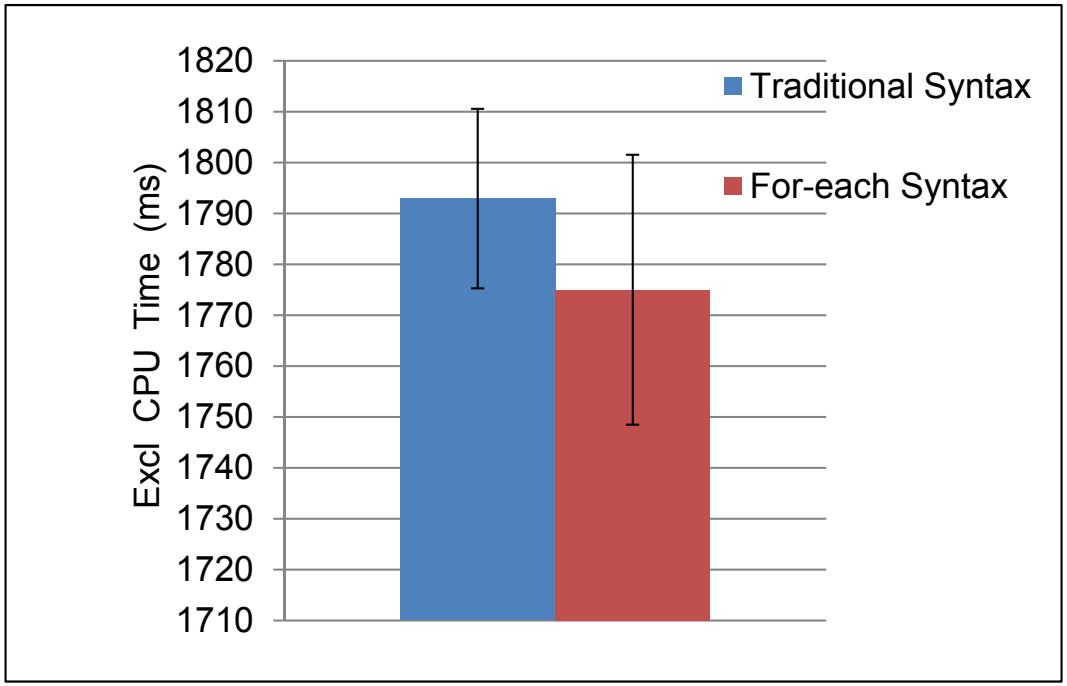
Experimento 2

- Aplicativo OpenSudoku: **Sintaxe do For**

```
protected boolean validate() {
    boolean valid = true;
    Cell cell;
    int value;
    Map<Integer, Cell> cellsByValue = new HashMap<Integer, Cell>();
    for (Cell a:mCells) {
        cell = a;
        value = cell.getValue();
        if (cellsByValue.get(value) != null) {
            a.setValid(false);
            cellsByValue.get(value).setValid(false);
            valid = false;
        } else {
            cellsByValue.put(value, cell);
        }
    }
    return valid;
}
```

Experimento 2: Resultados

- Aplicativo OpenSudoku
  - Código Original x **Uso da Sintaxe Aprimorada do For**



Original:
1792,9221 ms

For-Each:
1775,0134 ms

1% mais rápido!

Experimento 2

- Aplicativo OpenSudoku: **Ambas práticas**

```
protected boolean validate() {
    boolean valid = true;
    Map<Integer, Cell> cellsByValue = new HashMap<Integer, Cell>();
    //int len = mCells.length;
    Cell cell;
    int value;
    //for (int i = 0; i < len; i++){
    for (Cell a :mCells) {
        //cell = mCells[i];
        cell = a;
        //value = cell.getValue;
        value = cell.mValue;
        if (cellsByValue.get(value) != null) {
            //mCell[i].setValid(false);
            a.mValid=false;
            //cellsByValue.get(value).setValid(false);
            cellsByValue.get(value).mValid=false;
            valid = false;
        } else {
            cellsByValue.put(value, cell);
        }
    }
    return valid;
}
```

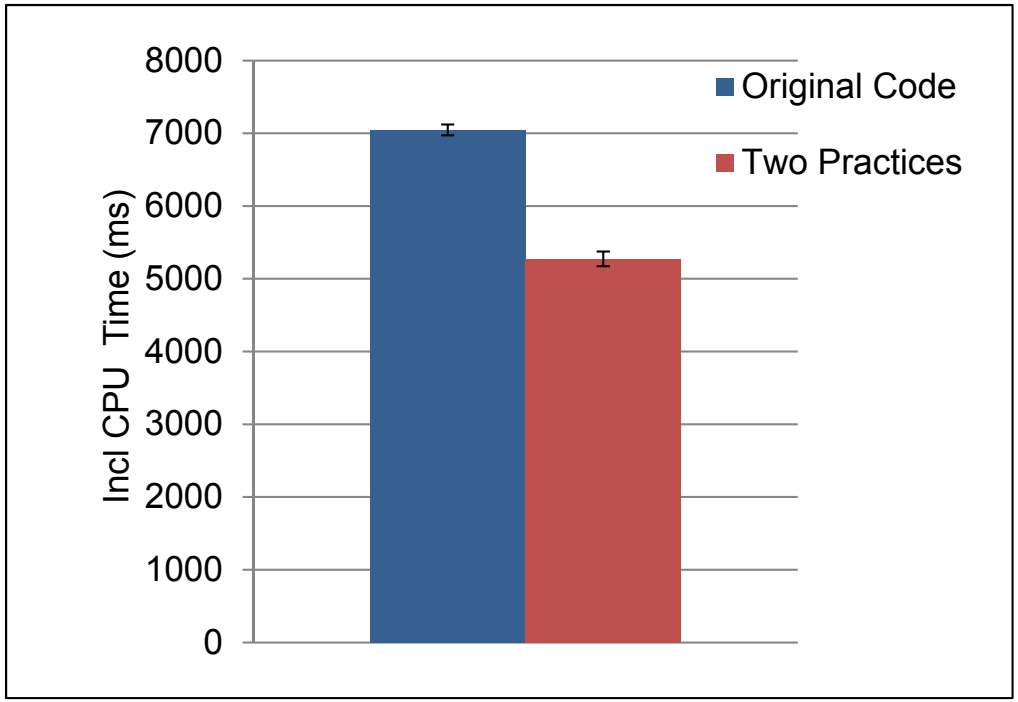
Experimento 2

- Aplicativo OpenSudoku: **Ambas práticas**

```
protected boolean validate() {
    boolean valid = true;
    Map<Integer, Cell> cellsByValue = new HashMap<Integer, Cell>();
    //int len = mCells.length;
    Cell cell;
    int value;
    //for (int i = 0; i < len; i++){
    for (Cell a :mCells) {
        //cell = mCells[i];
        cell = a;
        //value = cell.getValue;
        value = cell.mValue;
        if (cellsByValue.get(value) != null) {
            //mCell[i].setValid(false);
            a.mValid=false;
            //cellsByValue.get(value).setValid(false);
            cellsByValue.get(value).mValid=false;
            valid = false;
        } else {
            cellsByValue.put(value, cell);
        }
    }
    return valid;
}
```

Experimento 2: Resultados

- Aplicativo OpenSudoku
  - Código Original x Ambas as Práticas



Original:
7046,034 ms

Ambas práticas:
5274,1315 ms

25,16% mais rápido!

Conclusões e Trabalhos Futuros

- Através dos experimentos que a utilização das boas práticas proporcionam um **melhor desempenho** quando aplicadas!

- Estender os experimentos para avaliar o impacto das boas práticas no consumo de energia.

Universidade Federal de Pelotas
Ciência da Computação
Grupo de Arquiteturas e Circuitos Integrados

Evaluating Android best practices for performance

Obrigado!

Aline Tonini

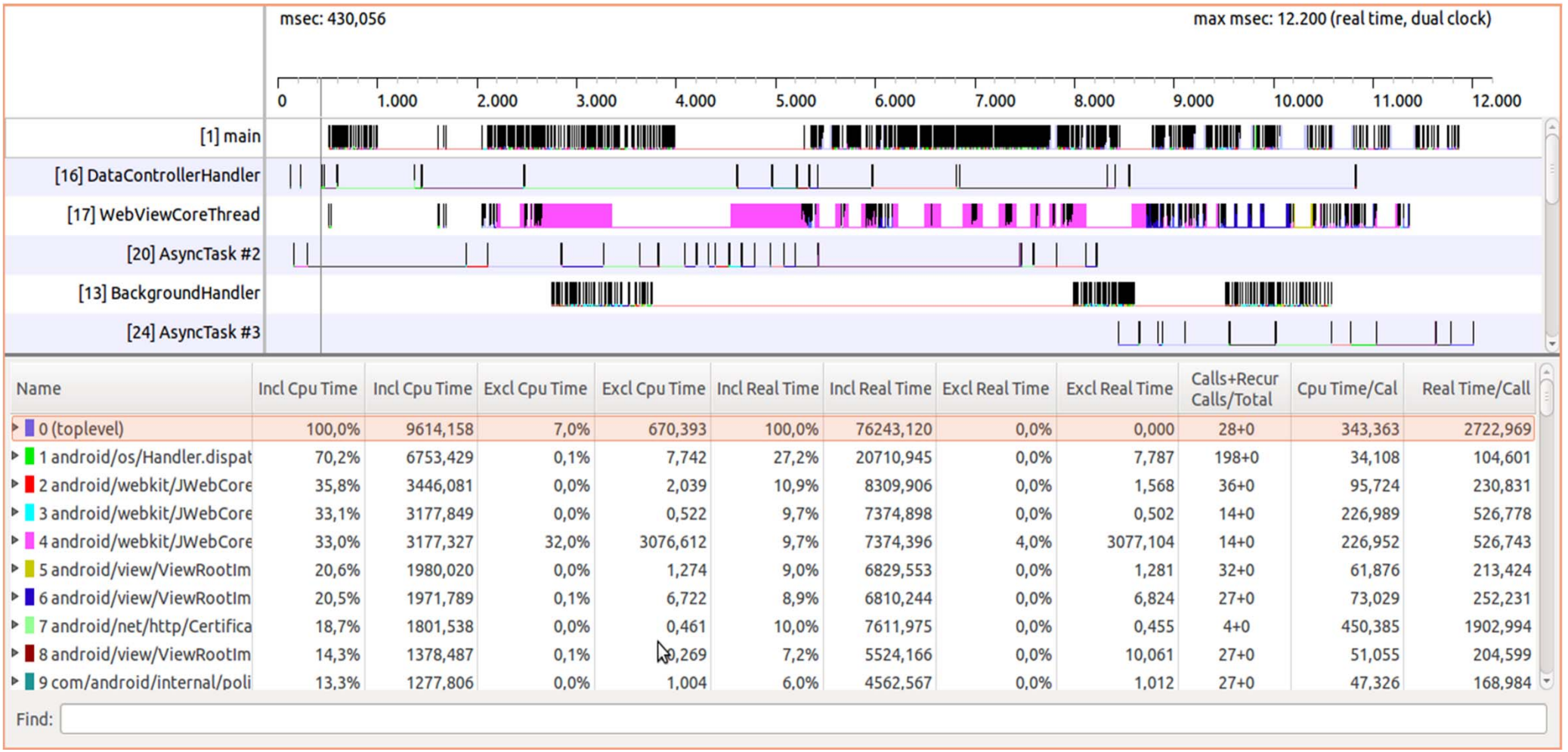
artonini@inf.ufpel.edu.br

Excl CPU time (ms)		
	Traditional Syntax	**For-each Syntax**
1	1765,875	1759,212
2	1766,537	1851,383
3	1796,082	1791,123
4	1781,555	1761,923
5	1778,929	1813,918
6	1789,912	1769,555
7	1786,354	1771,415
8	1779,532	1809,015
9	1787,874	1758,466
10	1821,943	1794,89
11	1810,186	1807,671
12	1805,116	1800,558
13	1800,057	1785,258
14	1816,636	1776,486
15	1806,309	1771,84
16	1793,994	1752,203
17	1766,796	1753,165
18	1779,677	1759,376
19	1779,226	1731,869
20	1838,509	1739,475
21	1805,523	1754,687
22	1791,348	1753,063
23	1813,277	1742,277
24	1796,486	1752,129
25	1802,396	1779,436
26	1779,182	1754,877
27	1774,964	1770,829
28	1777,974	1763,808
29	1784,655	1846,546
30	1810,759	1773,95
Média	**1792,9221**	**1775,013433**
DP	**17,64133365**	**26,53688215**

Incl CPU time (ms)		
	With Getter/Setter	Without Getter/Setter
1	7042,272	4308,745
2	6943,953	4103,22
3	7049,796	4092,465
4	6944,632	4198,53
5	6944,979	4175,352
6	6966,363	4291,452
7	6949,645	4195,892
8	6955,596	4267,209
9	7020,494	4194,221
10	6971,231	4203,683
11	7253,174	4098,668
12	6997,054	4238,705
13	7024,805	4095,221
14	7021,651	4101,767
15	7009,185	4106,69
16	7085,818	4094,322
17	7052,614	4077,468
18	7106,401	4094,79
19	7084,875	4111,771
20	7049,811	4091,746
21	7054,573	4037,11
22	7078,753	4041,406
23	7085,444	4335,127
24	7094,278	4347,417
25	7337,856	4223,93
26	7115,693	4210,337
27	7194,088	4233,469
28	7095,412	4229,125
29	6922,233	4226,187
30	6828,162	4244,482
Média	**7046,034**	**4195,0565**
DP	**101,2619866**	**87,77230576**

Incl CPU time(ms)		
	Two Practices	**Original Code**
1	5301,004	7042,272
2	5382,019	6943,953
3	5408,547	7049,796
4	5385,193	6944,632
5	5253,416	6944,979
6	5235,729	6966,363
7	5236,468	6949,645
8	5386,384	6955,596
9	5228,47	7020,494
10	5228,439	6971,231
11	5231,696	7253,174
12	5245,378	6997,054
13	5250,607	7024,805
14	5246,759	7021,651
15	5240,527	7009,185
16	5229,034	7085,818
17	5251,023	7052,614
18	5209,57	7106,401
19	5461,903	7084,875
20	5220,296	7049,811
21	5382,299	7054,573
22	5222,856	7078,753
23	5185,682	7085,444
24	5180,521	7094,278
25	5189,478	7337,856
26	5205,02	7115,693
27	5300,56	7194,088
28	5297,286	7095,412
29	5328,853	6922,233
30	5298,929	6828,162
Média	**5274,131533**	**7046,034**
DP	**74,39731901**	**101,2619866**

- Visualização do Trace



Name	Incl Cpu Time	Incl Cpu Time	Excl Cpu Time	Excl Cpu Time	Incl Real Time	Incl Real Time	Excl Real Time	Excl Real Time	Calls+Recur Calls/Total	Cpu Time/Cal	Real Time/Call
0 (toplevel)	100,0%	9614,158	7,0%	670,393	100,0%	76243,120	0,0%	0,000	28+0	343,363	2722,969
1 android/os/Handler.dispat	70,2%	6753,429	0,1%	7,742	27,2%	20710,945	0,0%	7,787	198+0	34,108	104,601
2 android/webkit/JWebCore	35,8%	3446,081	0,0%	2,039	10,9%	8309,906	0,0%	1,568	36+0	95,724	230,831
3 android/webkit/JWebCore	33,1%	3177,849	0,0%	0,522	9,7%	7374,898	0,0%	0,502	14+0	226,989	526,778
4 android/webkit/JWebCore	33,0%	3177,327	32,0%	3076,612	9,7%	7374,396	4,0%	3077,104	14+0	226,952	526,743
5 android/view/ViewRootIm	20,6%	1980,020	0,0%	1,274	9,0%	6829,553	0,0%	1,281	32+0	61,876	213,424
6 android/view/ViewRootIm	20,5%	1971,789	0,1%	6,722	8,9%	6810,244	0,0%	6,824	27+0	73,029	252,231
7 android/net/http/Certifica	18,7%	1801,538	0,0%	0,461	10,0%	7611,975	0,0%	0,455	4+0	450,385	1902,994
8 android/view/ViewRootIm	14,3%	1378,487	0,1%	0,269	7,2%	5524,166	0,0%	10,061	27+0	51,055	204,599
9 com/android/internal/poli	13,3%	1277,806	0,0%	1,004	6,0%	4562,567	0,0%	1,012	27+0	47,326	168,984

Find: