

Where Programming Language Theory meets Systems Optimization

Alcides Fonseca
amfonseca@fc.ul.pt
<http://alcidesfonseca.com/>

Alcides Fonseca
amfonseca@fc.ul.pt
<http://alcidesfonseca.com/>

Æminium



Jonathan Aldrich



Paulo Marques



Bruno Cabral



Néstor Cataño



Manuel Mohr



Sven Stork



Alcides Fonseca



Ijaz Ahmed

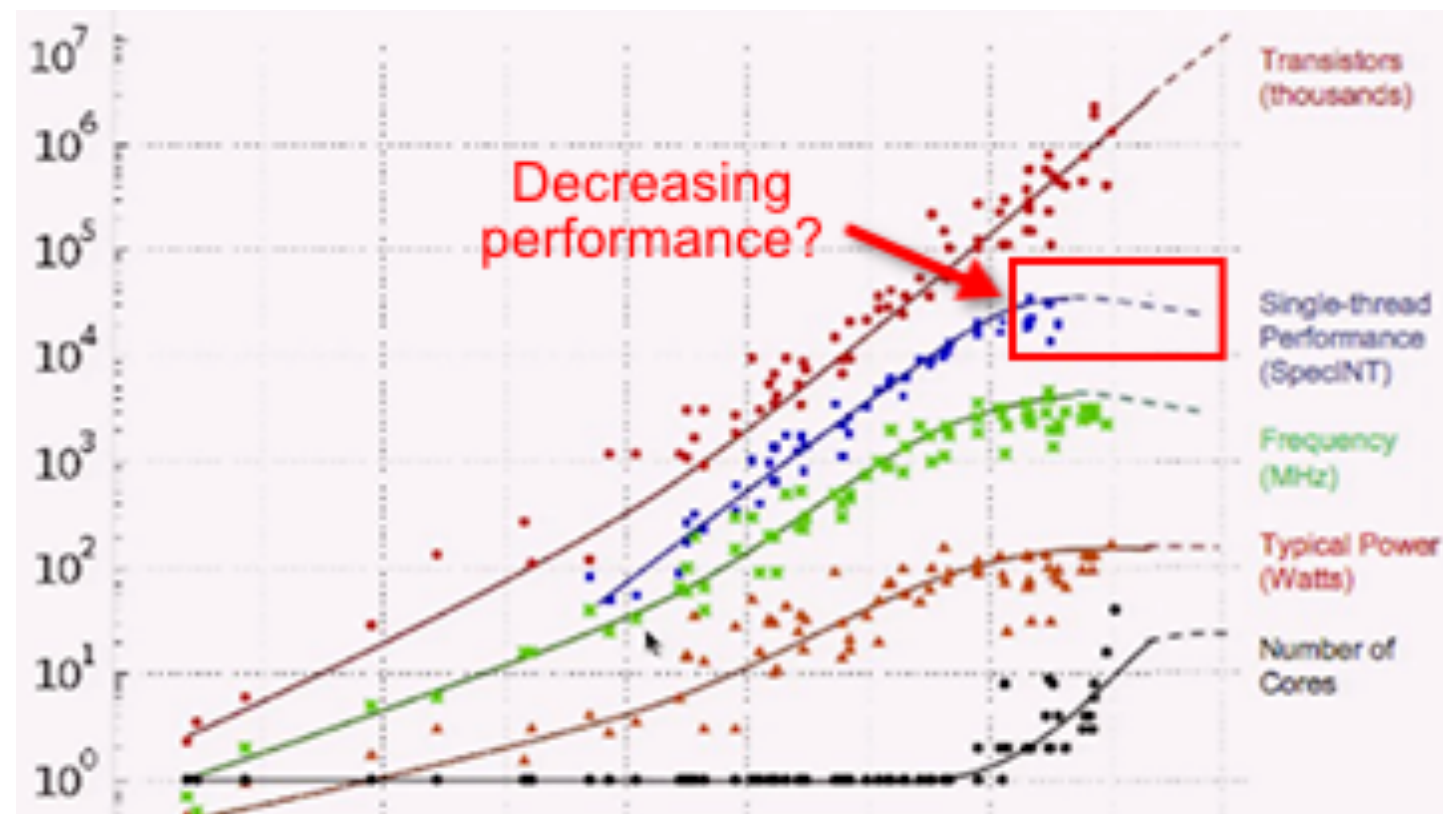
CMU

U. Coimbra

U. Madeira

<http://aeminium.dei.uc.pt>

Motivation



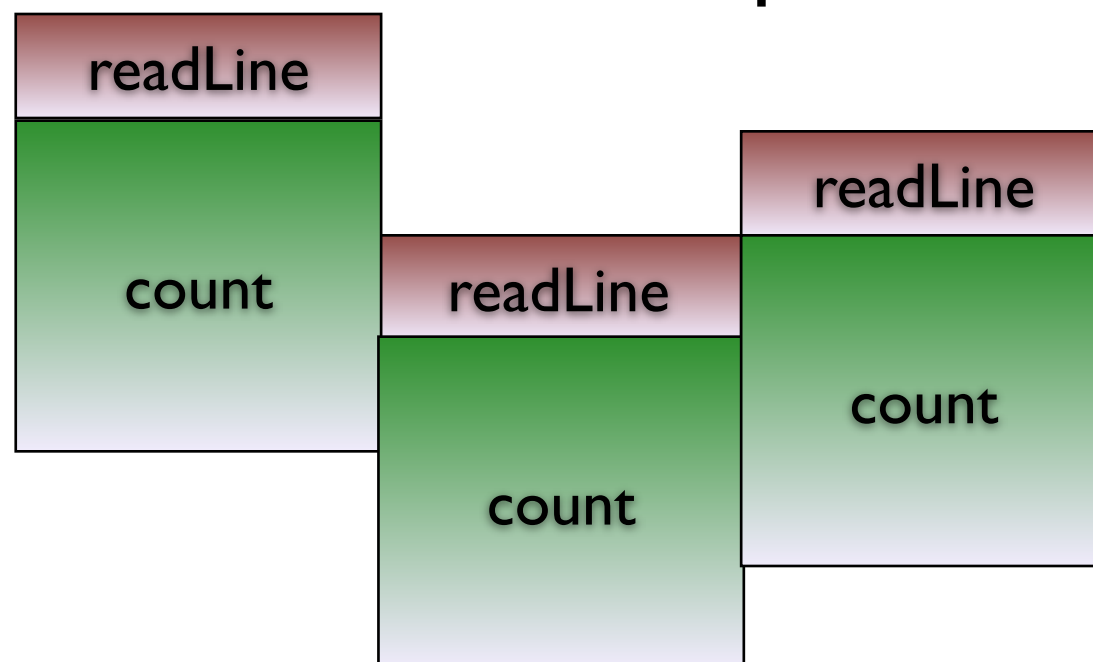
Æminium @ CMU

```
method immutable String readLine(shared File f) {  
    f.readLine();  
}
```

...

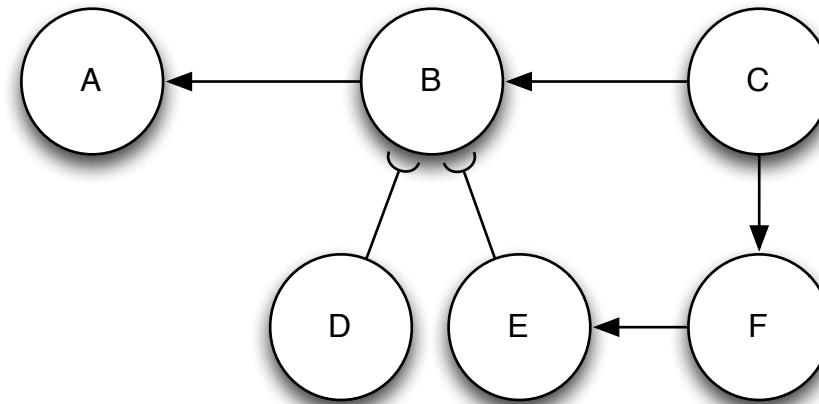
```
Array<Int> charsPerLine = new Array[lines];  
for (Int i=0; i<lines; i++) {  
    charsPerLine[i++] = count(readLine(file));  
}
```

par sequential

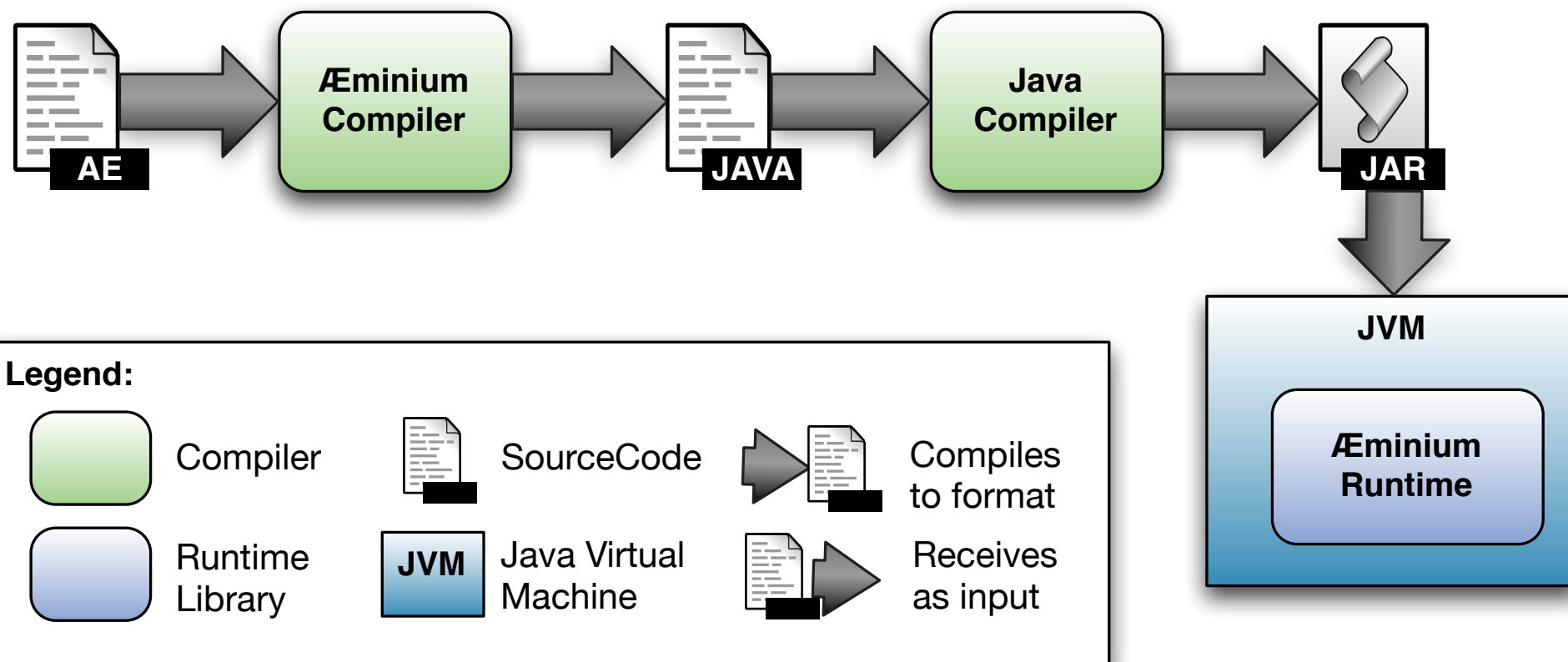
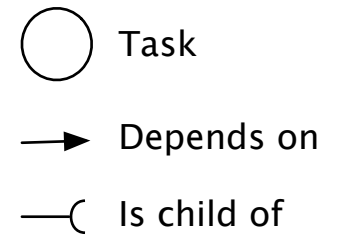


Æminium @ UCoimbra

```
method immutable String readLine(shared
File f) {
    f.readLine();
}
...
Array<Int> charsPerLine = new Array[lines];
for (Int i=0; i<lines; i++) {
    charsPerLine[i++] = count(readLine(file));
}
```



Legend:



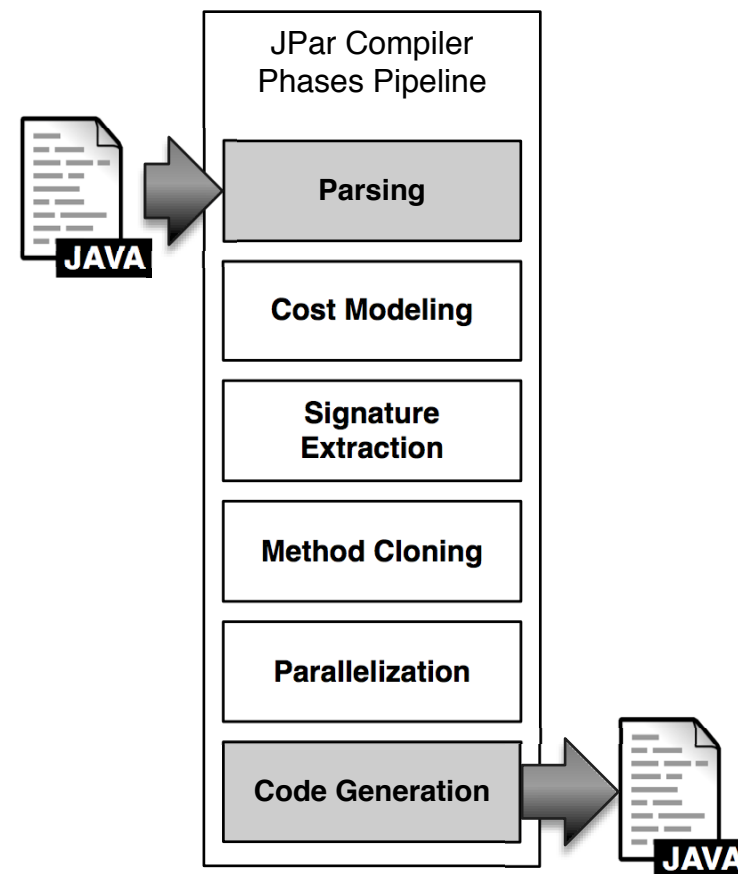
Lessons Learned

- (i) $\mathcal{AE}_{\text{minium}}$ is not practical
- (ii) $\mathcal{AE}_{\text{minium}}$ is not efficient

(i) $\mathcal{AE}$ minium is not practical

What if access permissions were inferred?

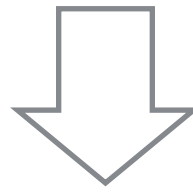
JPar Compiler



```

int f(int n) {
    if (n < 2) { // read(n), control(f)
        return n; // write(return), control(f)
    }
    int a = f(n - 1); // call(f), read(n), write(a)
    int b = f(n - 2); // call(f), read(n), write(b)
    return a + b; // read(a), read(b), control(f), write(return)
}

```



```

int f(int n) {
    if (RuntimeManager.shouldSeq())
        return jpar_sequential_version_of_f(n);

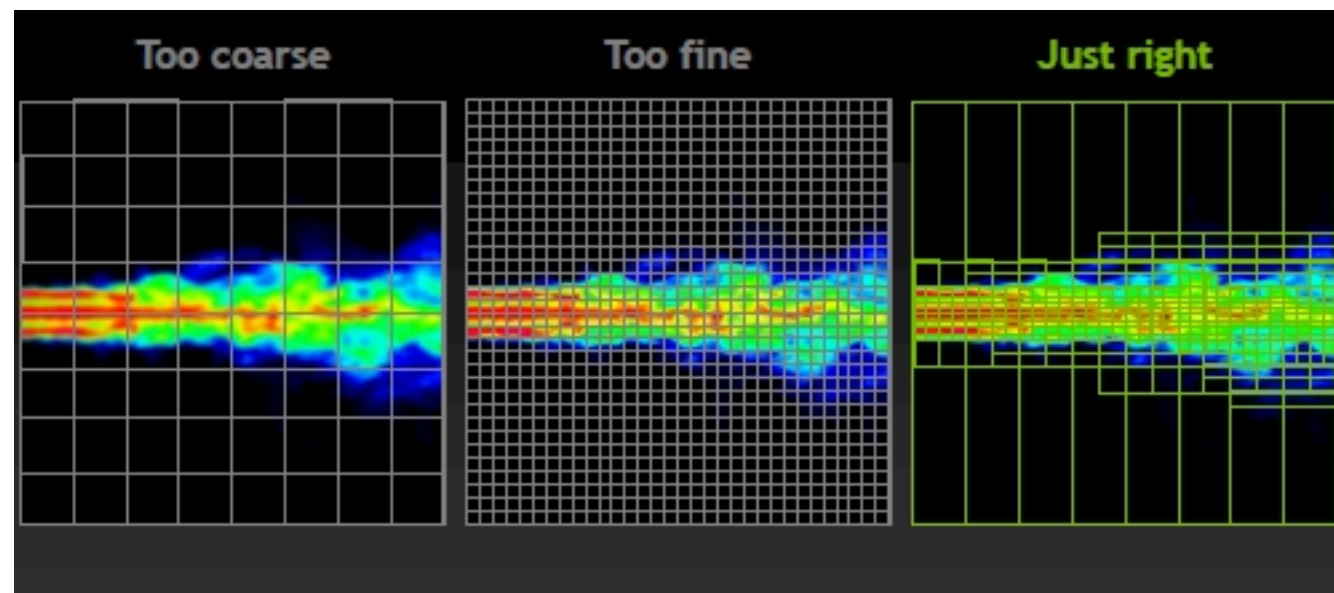
    if (n < 2) {
        return n;
    }
    Future<Integer> b_tmp = new Future<Integer>(task -> f(n-2));
    Future<Integer> a_tmp = new Future<Integer>(task -> f(n-1));
    int a = a_tmp.get();
    int b = b_tmp.get();
    return a + b;
}

```

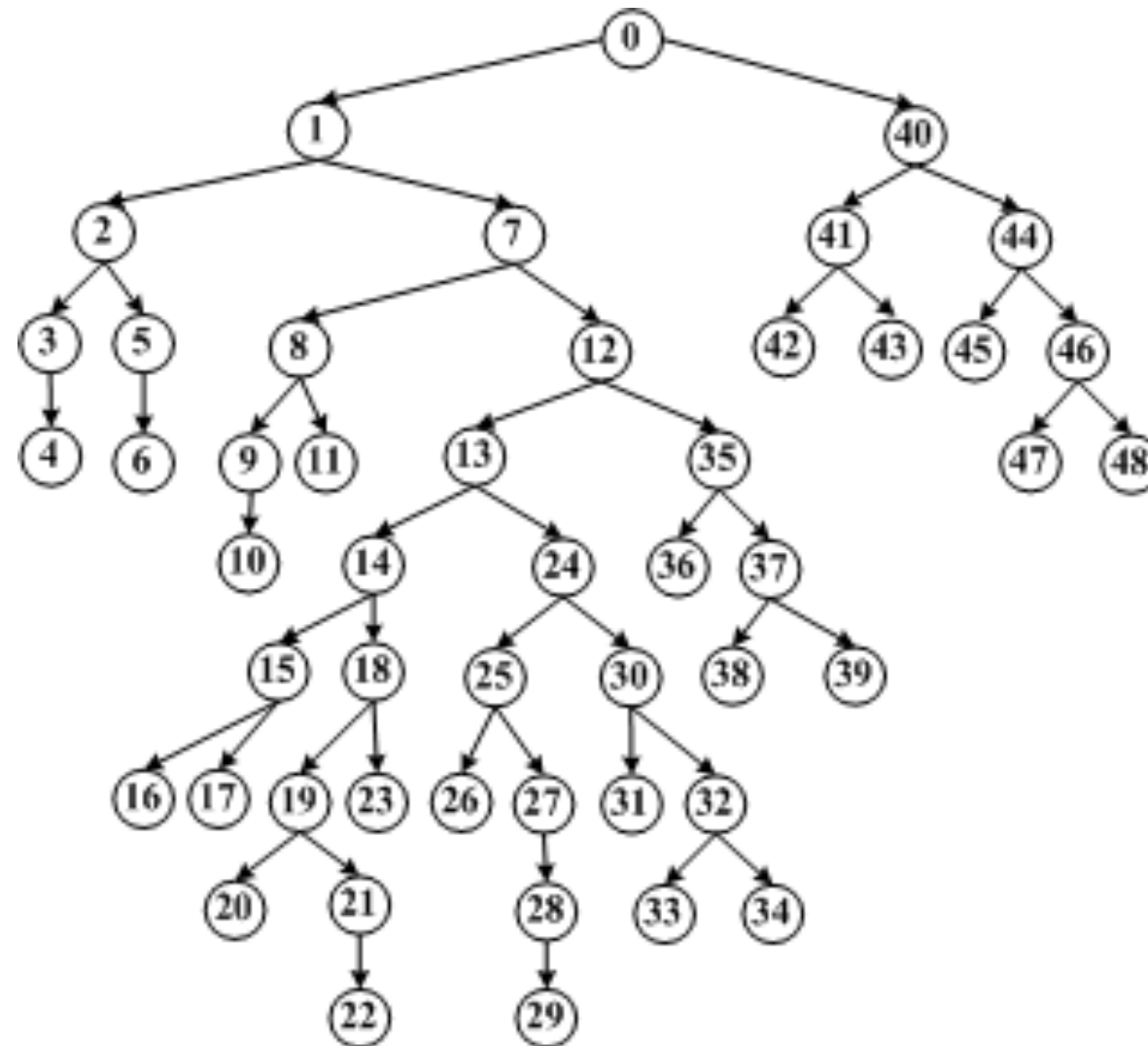
Spoiler Alert

(ii) $\mathcal{AE}_{\text{minium}}$ is not efficient

Dynamic Granularity Control

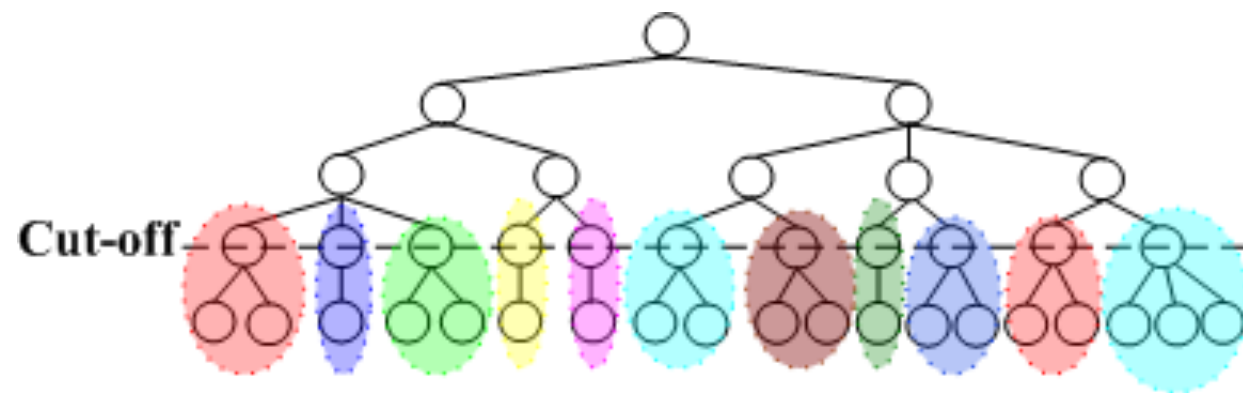


Parallel Programs

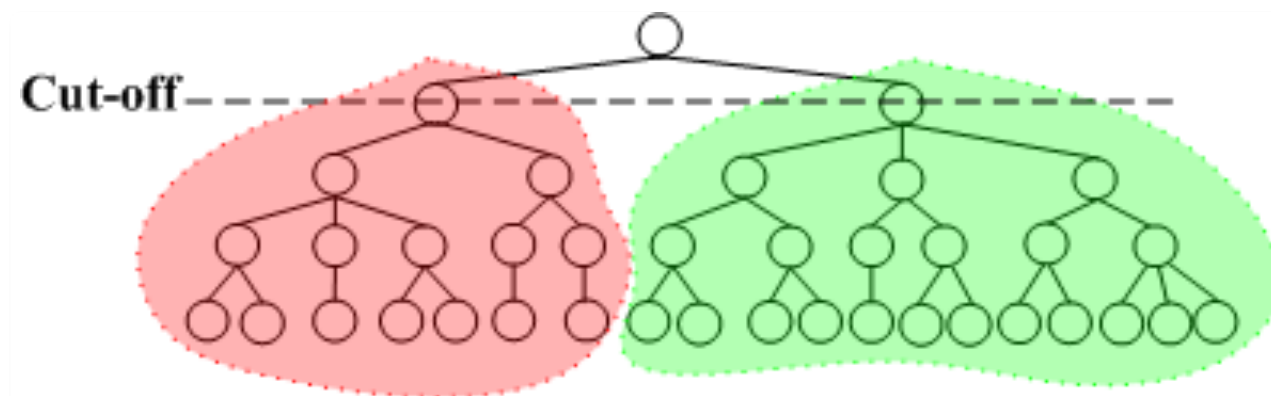


Cut-off Mechanisms

Cut-off = 3

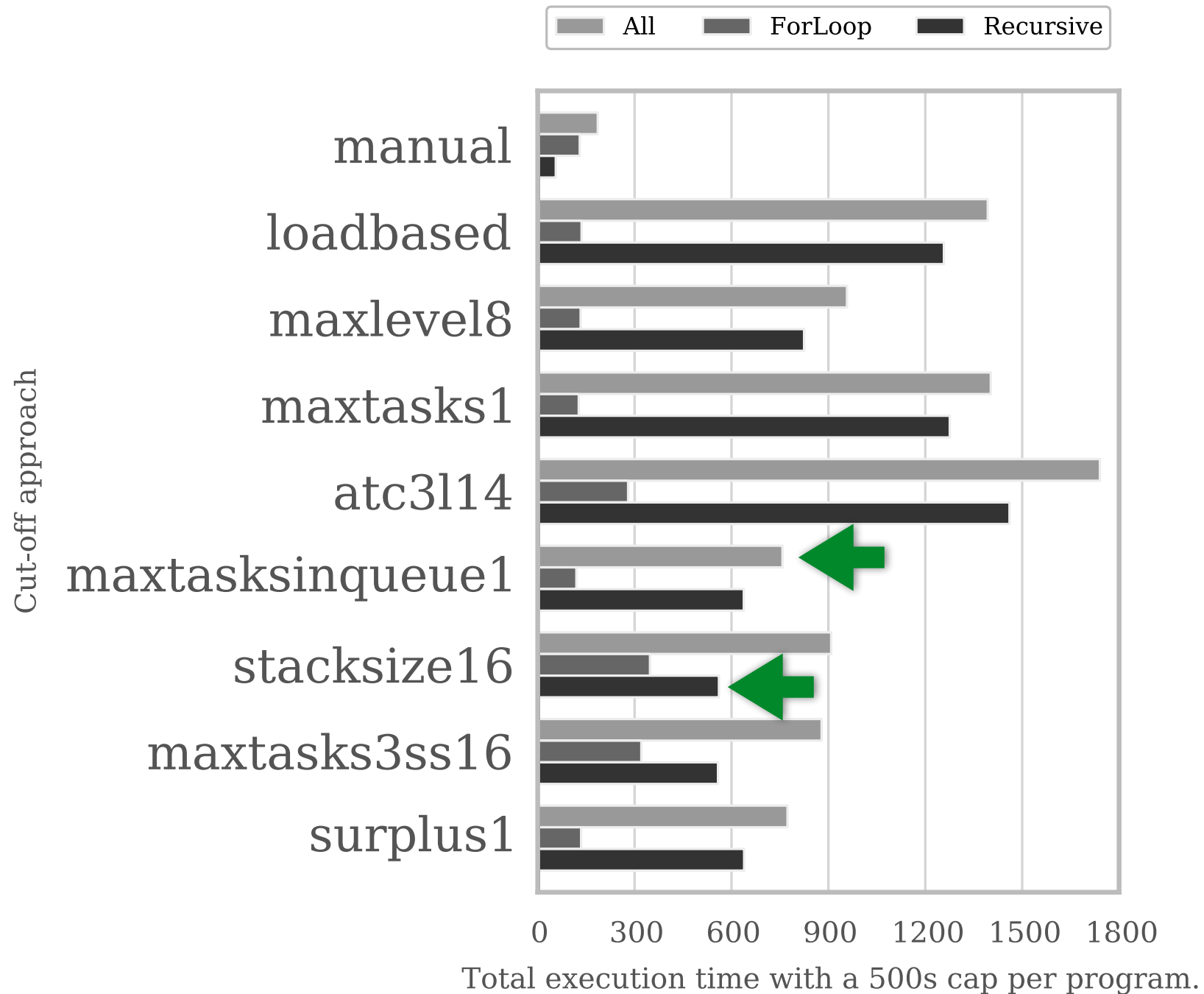


Cut-off = 1



```
int f(int n) {  
    if (RuntimeManager.shouldSeq())  
        return jpar_sequential_version_of_f(n);  
  
    if (n < 2) {  
        return n;  
    }  
    Future<Integer> b_tmp = new Future<Integer>(task -> f(n-2));  
    Future<Integer> a_tmp = new Future<Integer>(task -> f(n-1));  
    int a = a_tmp.get();  
    int b = b_tmp.get();  
    return a + b;  
}
```

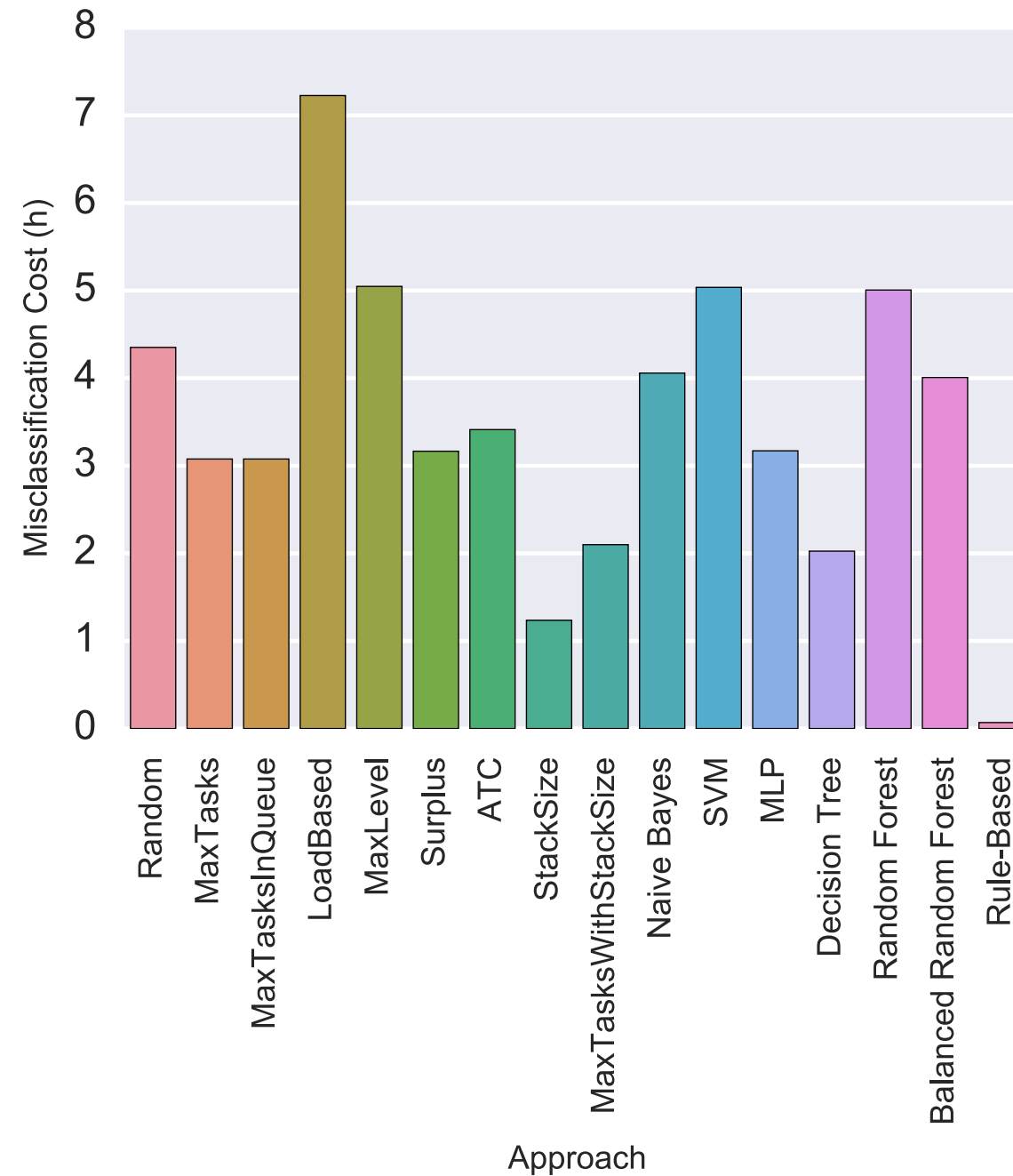
Results



Machine Learning: Program Features

- For-Loop (or not)
- Unbalance
- # Kernels
- Branching Factor
- Nesting Level

Minimizing useless execution time



Conclusion

- The ideas from *Æminium* outlived the project.
- We have adapted ideas from the PL research at CMU to other contexts (*Æminium* to Java)
- To solve performance issues, we are going back to PLT...

Future Steps

- Using type systems to infer execution time, memory and energy.
- Using dependent type systems for automated program synthesis and improvement, through genetic programming.

Alcides Fonseca
amfonseca@fc.ul.pt
<http://alcidesfonseca.com/>