Static Java Program Features for Intelligent Squash Prediction

Jeremy Singer, <u>Paraskevas Yiapanis</u>, Adam Pocock, Mikel Lujan, Gavin Brown, Nikolas Ioannou, Marcelo Cintra

Thread-level Speculation...

- Aim to use parallel multi-core resources in execution of sequential program

- Assume low likelihood for certain data dependence conflicts, and parallelize accordingly

- Runtime safety mechanisms to detect conflict and roll-back

Our TLS Model

- Method-level speculation: at a method call site, execute the callee method (non-speculative) in parallel with the caller continuation (speculative)

Method – Level Speculation

```
public void m_A() {
    //code before
    m_B();
   //code after
}
```

```
public void m_B() {
  //do stuff
}
```


Paraskevas Yiapanis

Method – Level Speculation

```
public void m_A() {
    //code before
    m_B();
   //code after
}
```

```
public void m_B() {
  //do staff
}
```

Method – Level Speculation

Time

Non – speculative thread

```
public void m_A() {
    //code before
    m_B();
    //code after
}
```

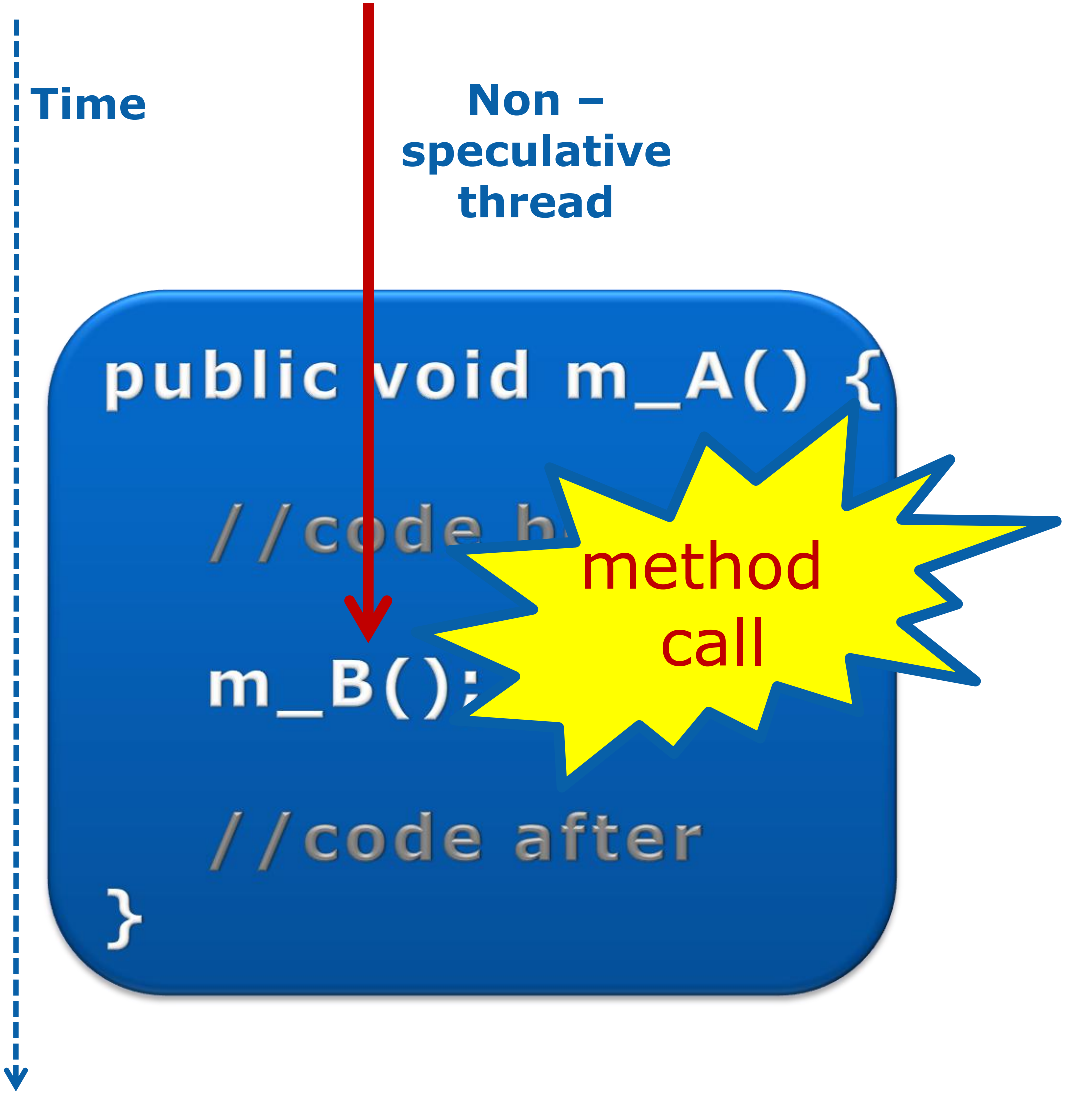
Method – Level Speculation

Time

Non – speculative thread

```
public void m_A() {
    //code before
    m_B();
    //code after
}
```

Method – Level Speculation



Method – Level Speculation

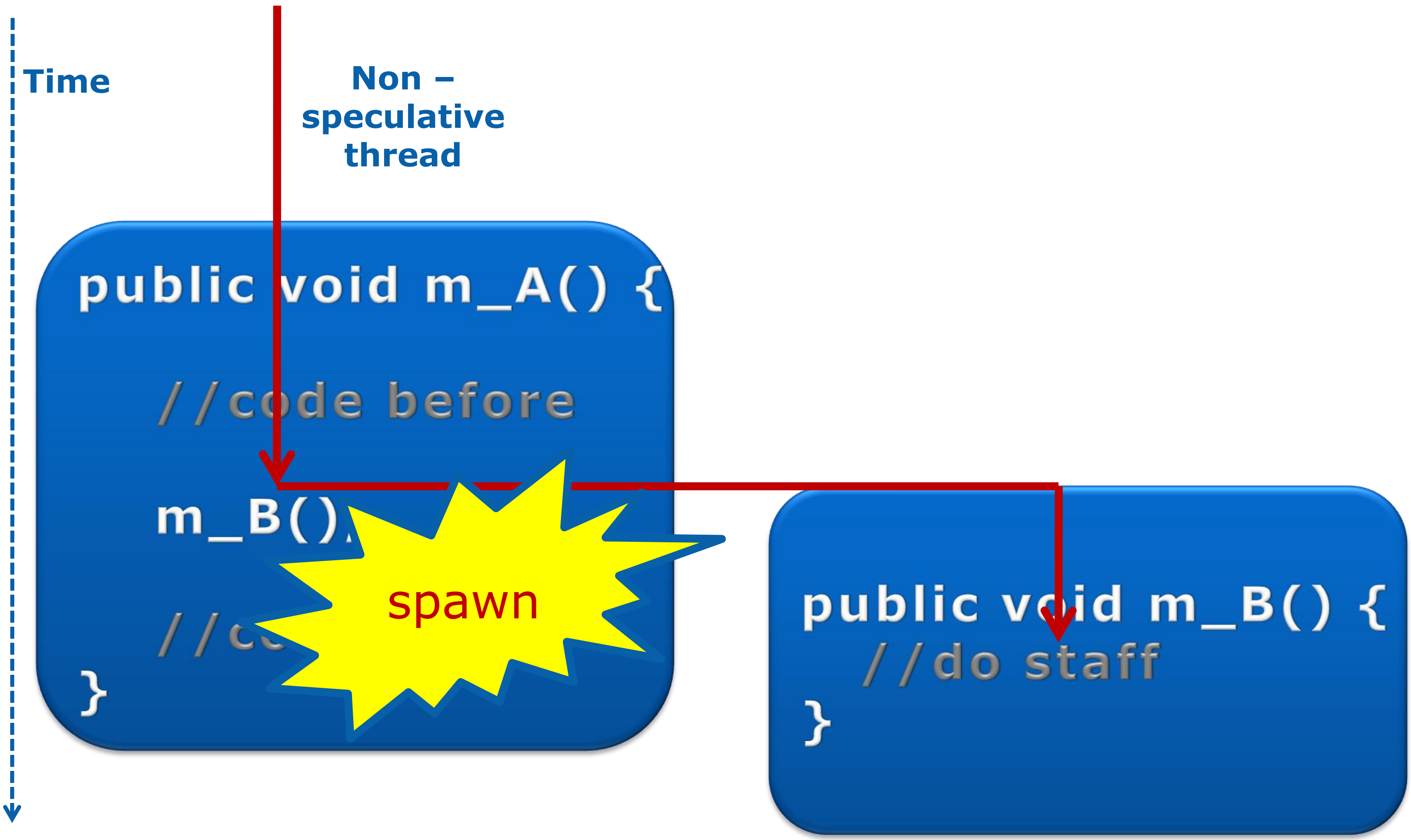
Time

Non – speculative thread

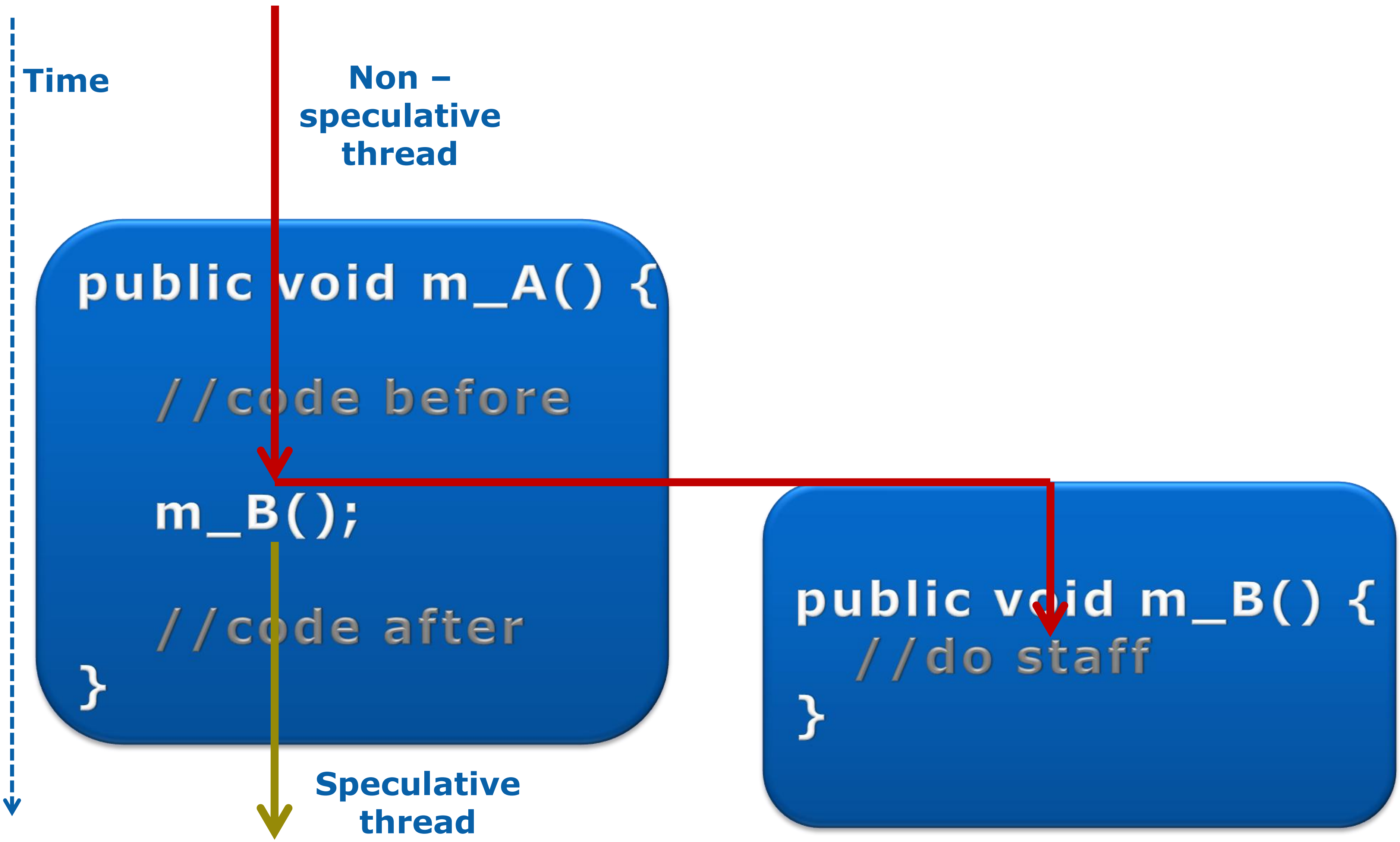
```
public void m_A() {
    //code before
    m_B();
    //code after
}
```

```
public void m_B() {
    //do staff
}
```

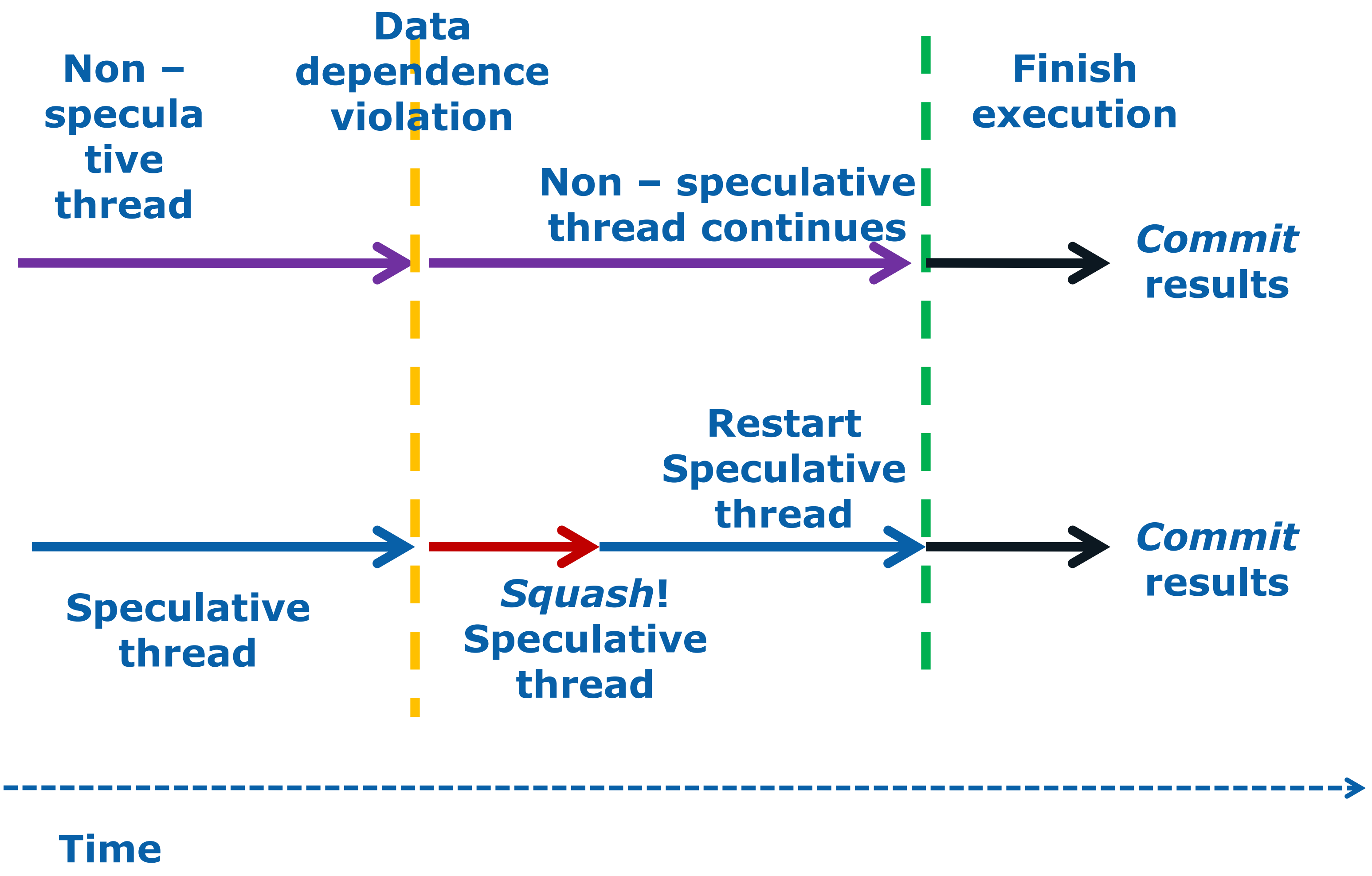
Method – Level Speculation



Method – Level Speculation



Method – level Speculation



Particular Problem: *Squash Prediction*

Squashes (caused by data dependence conflicts at runtime) are expensive

- Runtime overhead of rollback / re-execution
- Wasted parallel resource that could be used for executing alternative parallel threads without conflicts

Ideally, we could *predict squashes* ahead-of-time, and avoid spawning conflicting threads

Data Collection

Characterize two Java methods (caller and callee) using standard metrics - features

Execute these two methods in parallel and determine whether there is a data dependence violation - class

Store the vector of features, and the class as a row in the learning database - example

Java Method Features

- **All static characteristics of methods**
- **23 real-valued features from the MILEPOST gcc compiler**
  - e.g. Number of CFG basic blocks with more than 2 successors
- **22 binary features from our nano-pattern catalogue**
  - e.g. method may write value to an array
- **For each potential TLS spawn, we have 90 features (45 caller + 45 callee)**

TLS Emulation Infrastructure

- **Java benchmarks (SPECjvm98 / DaCapo)**
- **On top of instrumented Jikes RVM**
  - record method entry/exit, memory read/write
- **On top of Simics full-system simulator**
- **Generate sequential execution trace files with timings**
- **Feed into custom trace-based TLS emulator**

TLS Execution Parameters

- Method-level speculation
- Spawn on all methods longer than threshold runlength
- Parameterizable costs for TLS spawn/commit/squash events
- 2 cores, so maximum of 1 in-flight speculation
- This is the simplest scenario for learning

Learning Technique

- Generate a set of rules, using decision tree learner (C5.0 algorithm)
- Order rules based on confidence (accuracy)
- Only consider rules above threshold confidence
- example rule:

Learning Technique

- **Generate a set of rules, using decision tree learner (C5.0 algorithm)**
- **Order rules based on confidence (accuracy)**
- **Only consider rules above threshold confidence**
- **example rule:**

if *noparams* = 1
and *arrReader* = 1
and *objCreator*' = 0
and *thisInstanceFieldWriter*' = 0
and *methodInstrNum*' <= 136

Paraskevas Yiapanis

Learning Technique

- **Generate a set of rules, using decision tree learner (C5.0 algorithm)**
- **Order rules based on confidence (accuracy)**
- **Only consider rules above threshold confidence**
- **example rule:**

Squash!

if *noparams* = 1
and *arrReader* = 1
and *objCreator*' = 0
and *thisInstanceFieldWriter*' = 0
and *methodInstrNum*' <= 136

Application of Rule-Sets

leave-one-out cross-validation:

- learn rules one a set of Java benchmarks (training set)
- apply these rules on a different benchmark (testing set)

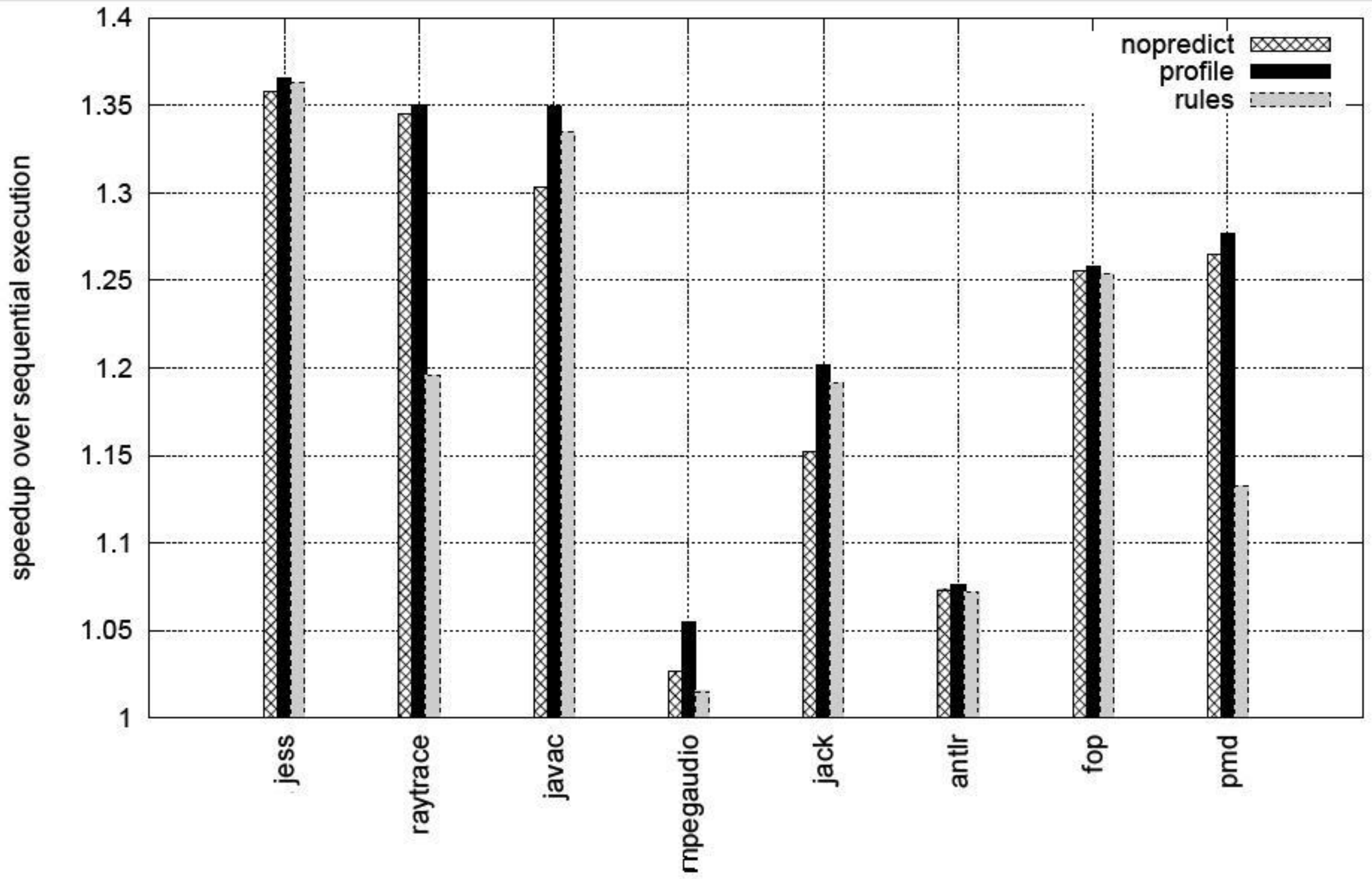
Paraskevas Yiapanis

Evaluation

Three thread-spawning strategies

- ***no prediction*** (spawn for all methods above threshold runlength)
- ***profile-based*** spawning (spawn for call-sites where majority of spawns committed successfully, on profile run of that benchmark)
- ***rules-based*** spawning (for each benchmark, spawn where rules predict no squash, based on LOOCV)

Results



speedup over sequential execution
1.4
1.35
1.3
1.25
1.2
1.15
1.1
1.05
1
nopredict
profile
rules
jess
raytrace
javac
mpegaudio
jack
antlr
fop
pmd

Observations (1)

In all cases, profile-based spawning gives best results

Is it feasible to learn TLS behaviour on one benchmark, then expect to be able to apply it to another benchmark?

- Yes, because of shared library / runtime code
- Yes, because of standard object-oriented design patterns

Observations (2)

In 2 cases, jess and jack, rules-based spawning is comparable with profile-based.

In 2 cases, raytrace and pmd, rules-based spawning is much worse than the other policies.

Observations (3)

Our rules-based squash prediction works well when there is a relatively high level of data dependences

- true for jess and jack

When there are few data dependences, rules-based prediction suffers from a high *false positive rate* (predicted squashes that would actually commit ok) inhibiting actual parallelism

- true for raytrace and pmd

We should *tweak* parameters for the learning algorithm to *reduce* the false positive rate.

Conclusions

Static characteristics may provide useful features for learning about Java methods

Some further steps need to be taken to improve squash prediction using ML

Next steps…

A better feature set is needed (incorporate dynamic characteristics of methods)

A larger training set is needed (more, and more diverse Java benchmarks for learning)

Perhaps rephrase the learning problem to give scope for better speedups (loop-level speculation?)