Leveraging Software Testing to Explore Input Dependence for Approximate Computing

Abdulrahman Mahmoud, Radha Venkatagiri, Khalique Ahmed, Sarita Adve, Darko Marinov, Sasa Misailovic

University of Illinois at Urbana-Champaign

swat@cs.illinois.edu

WAX 2017, Xi'an, China

Approximate Computing Across the Stack

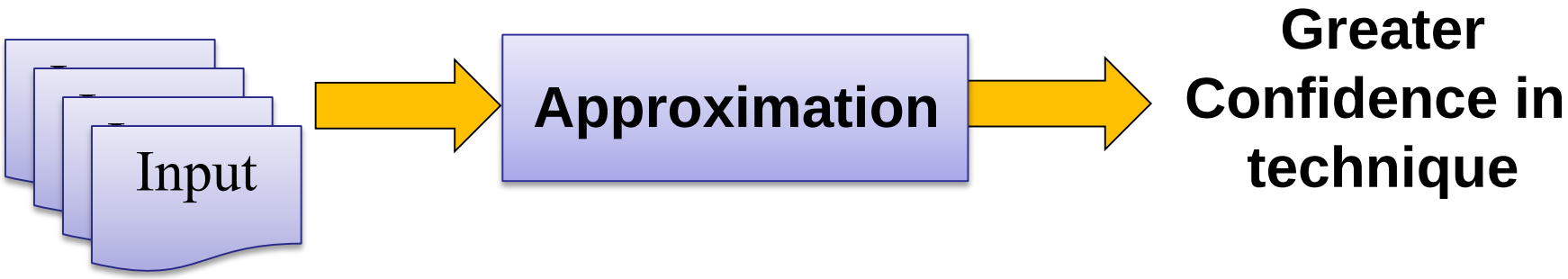
- Techniques
  - Loop Perforation, Value Approximation, …
- Frameworks
  - Approxilyzer, Chisel, Expax, IRA, Rumba, …
- Software
  - Approximate kernels, EnerJ, FlexJava, …
- Hardware
  - Approximate adders, neural networks, voltage scaling, …

Approximate Computing Across the Stack

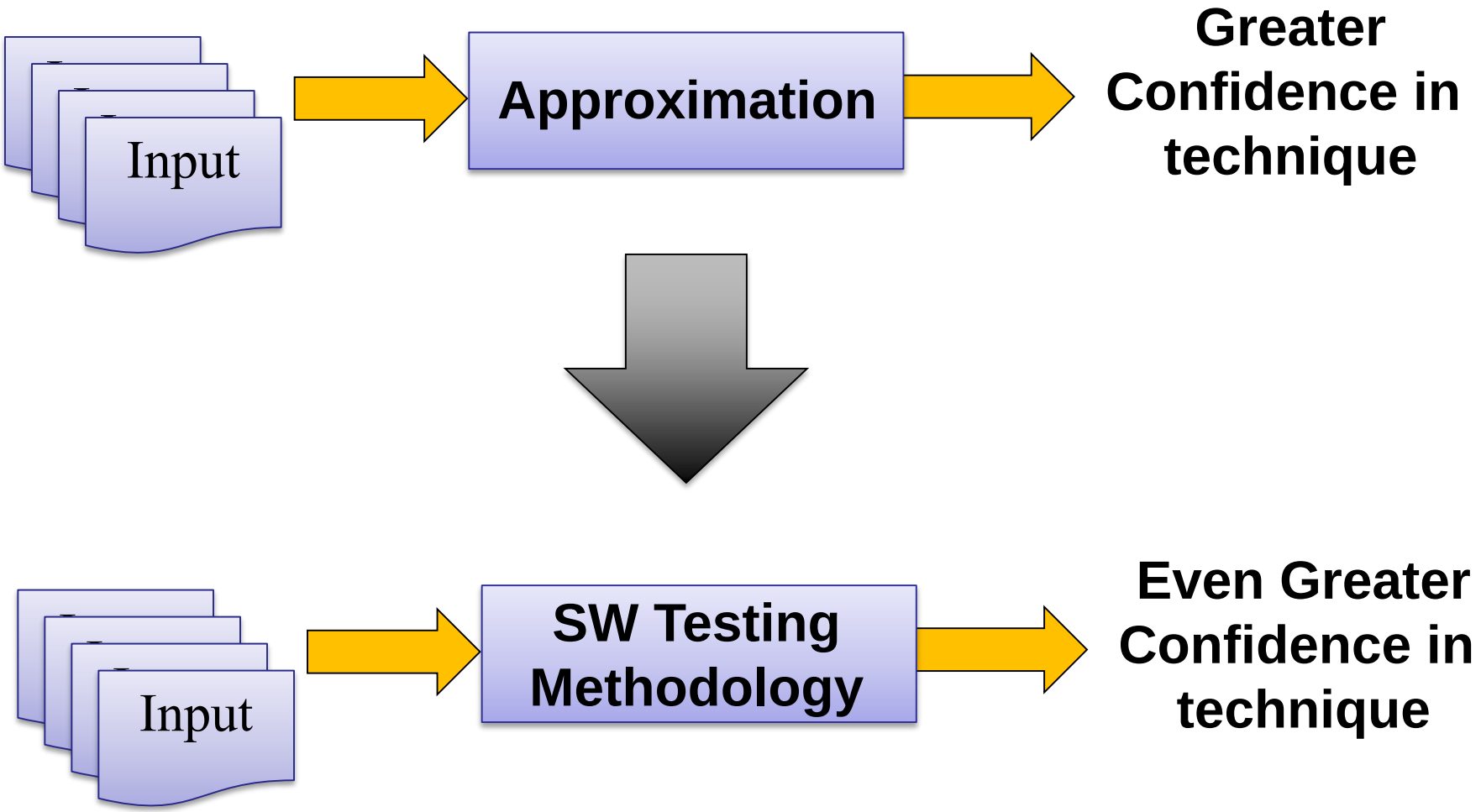
- Techniques ← Input Dependent
  - Loop Perforation, Value Approximation, …
- Frameworks ← Input Dependent
  - Approxilyzer, Chisel, Expax, IRA, Rumba, …
- Software ← Input Dependent
  - Approximate kernels, EnerJ, FlexJava, …
- Hardware ← Input Dependent
  - Approximate adders, neural networks, voltage scaling, …

Input Dependence

Input Dependence



Input Dependence



Software Development Workflow

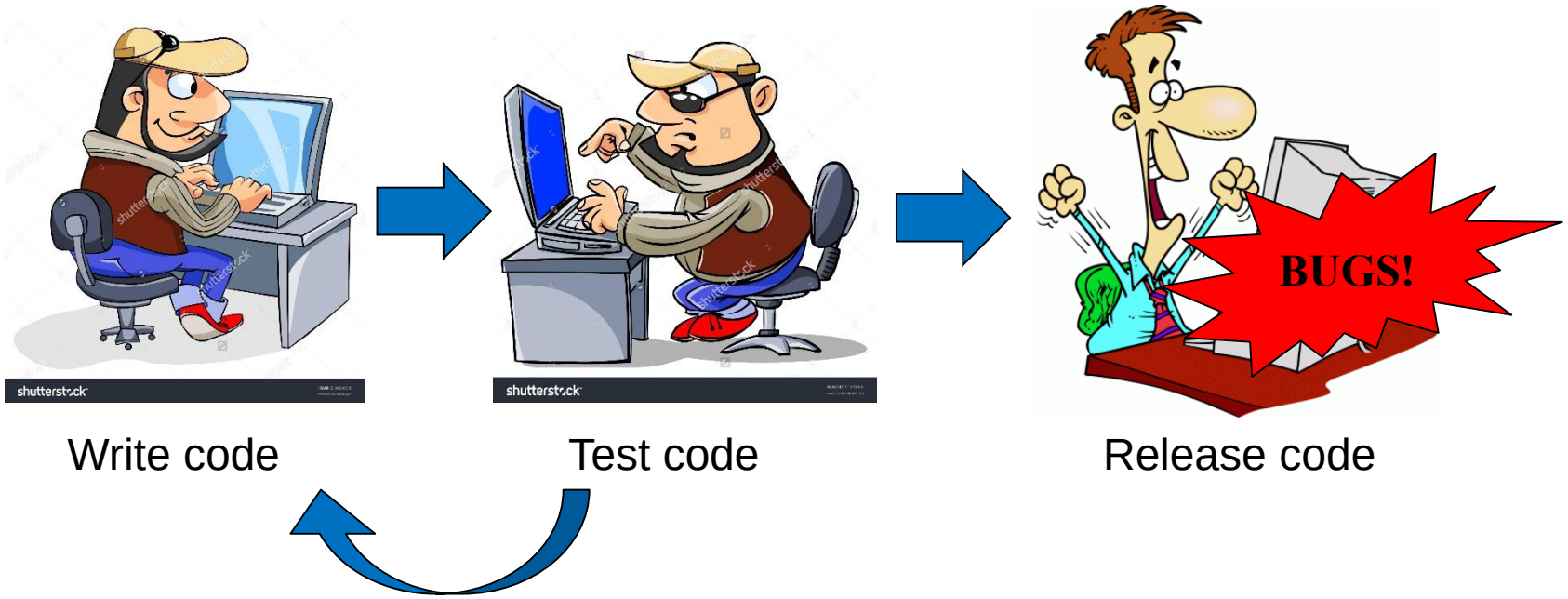
Typical workflow of code development

1. **Write code**
2. **Test code**
3. **Release code**

Software Development Workflow

Typical workflow of code development

1. **Write code**
2. **Test code** — **Systematic approach**
3. **Release code**



Software Development Workflow

Typical workflow of code development

1. **Write code**
2. **Test code**
3. **Release code**

Systematic approach

Approximate Computing

1. **Approx Technique**
2. **Test Technique**
3. **Release Technique**

“Good Enough?”

- Metrics and techniques for assessing “good enough” include:
  - Coverage Analysis
  - Mutation Testing

- Metrics and techniques for speeding up testing:
  - Test Selection
  - Test Prioritization
  - Test Minimization

Coverage Analysis

- How representative are the inputs of the input space?

Coverage Analysis

- How representative are the inputs of the input space?
- Measured using a “Coverage Criterion”
  - Function coverage
  - Statement coverage
  - Branch coverage
  - Path coverage

Coverage Analysis

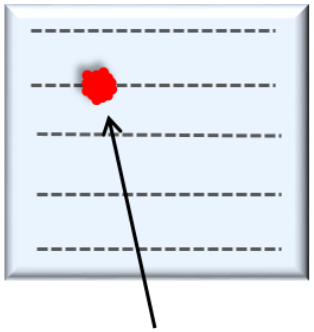
- How representative are the inputs of the input space?
- Measured using a “Coverage Criterion”
  - Function coverage
  - Statement coverage
  - Branch coverage
  - Path coverage

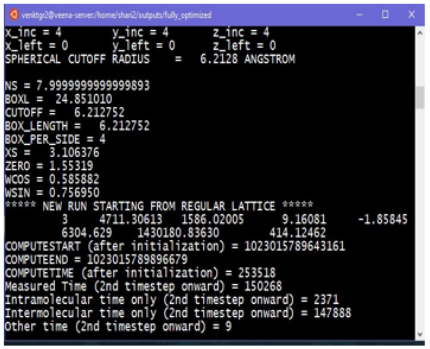
```
1. while(...) {
2.     if(A):
3.         F(X)
4.     else:
5.         G(Y)
   }
```

Application to an Approximation Analysis Tool

Error in program → APPROXILYZER → Quality of output

11% degradation





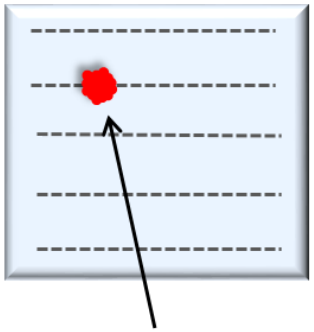
Single-bit error in register + dynamic instruction

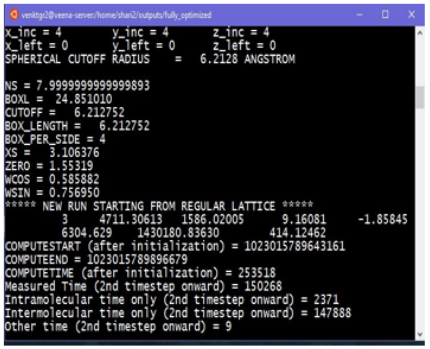
- Approxilyzer: General-purpose tool for identifying first order approx. insts
  - With a large input, 100% analysis is difficult

Application to an Approximation Analysis Tool

Error in program → APPROXILYZER → Quality of output

11% degradation





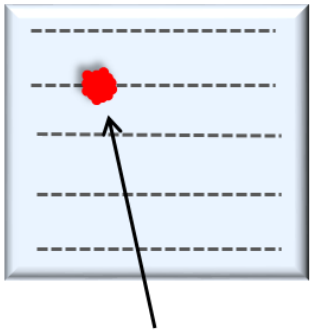
Single-bit error in register + dynamic instruction

- Approxilyzer: General-purpose tool for identifying first order approx. insts
  - With a large input, 100% analysis is difficult
- Input Minimization: Find a smaller, representative input

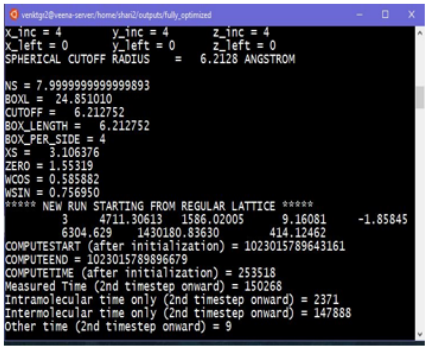
Application to an Approximation Analysis Tool

Error in program → APPROXILYZER → Quality of output

11% degradation



Single-bit error in register + dynamic instruction

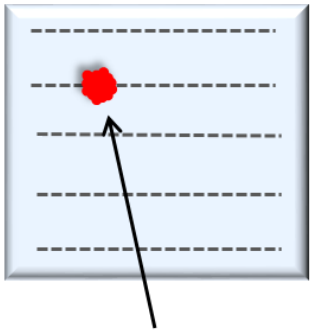


- Approxilyzer: General-purpose tool for identifying first order approx. insts
  - With a large input, 100% analysis is difficult
- Input Minimization: Find a smaller, representative input. How?

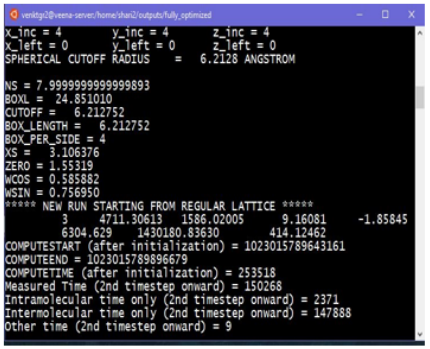
Application to an Approximation Analysis Tool

Error in program → APPROXILYZER → Quality of output

11% degradation



Single-bit error in register + dynamic instruction



- Approxilyzer: General-purpose tool for identifying first order approx. insts
  - With a large input, 100% analysis is difficult
- Input Minimization: Find a smaller, representative input. How?
- Minimize along a suitable coverage criterion

PC Coverage

- **Coverage criteria: PC Coverage**
  - **Similar to statement coverage, at *instruction level***
- **Approxilyzer operates at the inst level → Instruction-centric criterion**

```
if(A or B) { … }
```

PC Coverage

- Coverage criteria: PC Coverage
  - Similar to statement coverage, at *instruction level*
- Approxilyzer operates at the inst level → Instruction-centric criterion

```
if(A or B) { … }
```

- For example:
  - PARSEC Blackscholes runs on 64K option prices
  - 1000 unique options

PC Coverage

- Coverage criteria: PC Coverage
  - Similar to statement coverage, at *instruction level*
- Approxilyzer operates at the inst level → Instruction-centric criterion

```
if(A or B) { … }
```

- For example:
  - PARSEC Blackscholes runs on 64K option prices
  - 1000 unique options
  - Only 21 options needed for 100% PC coverage

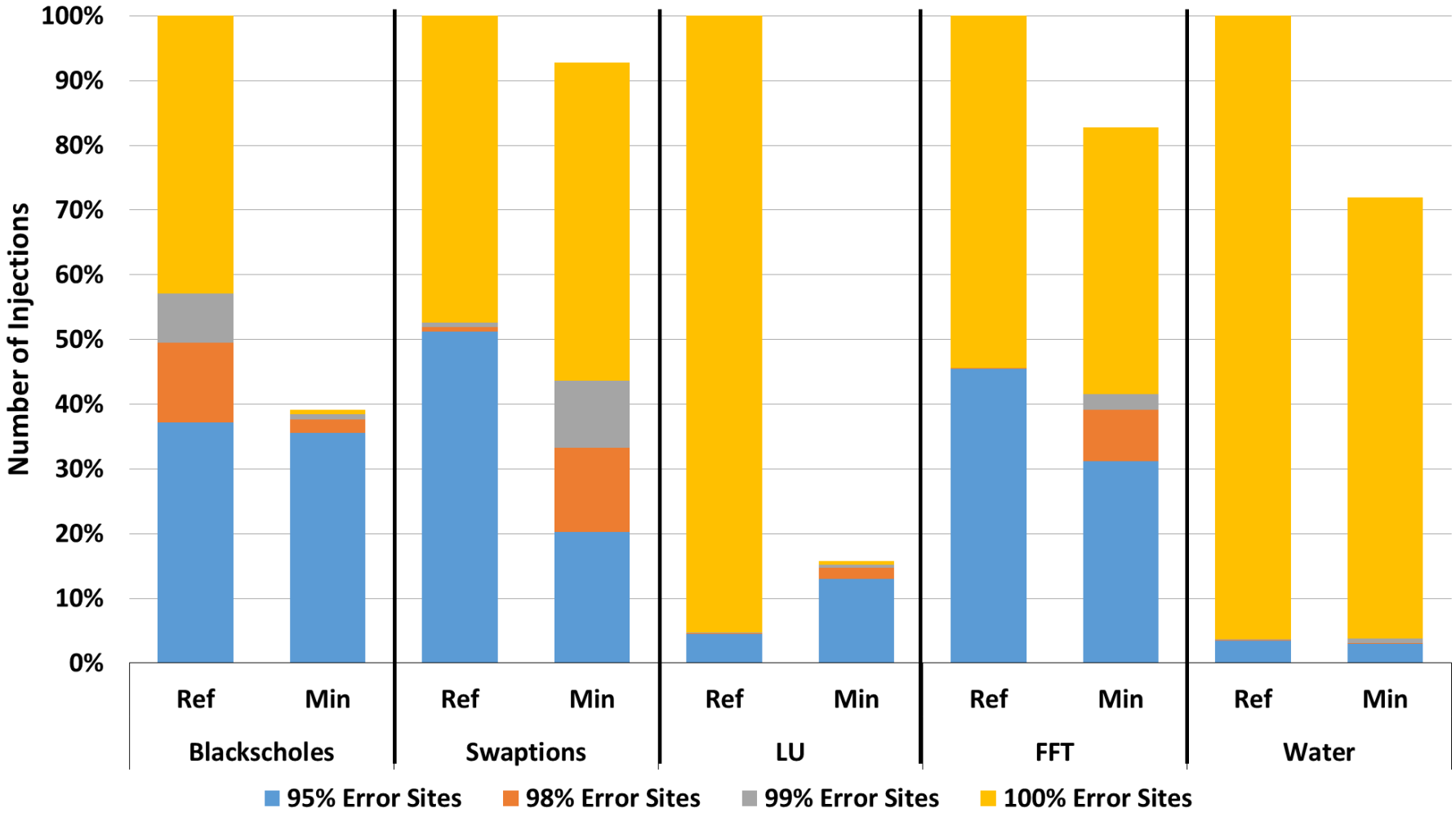
PC Coverage

- Coverage criteria: PC Coverage
  - Similar to statement coverage, at *instruction level*
- Approxilyzer operates at the inst level → Instruction-centric criterion

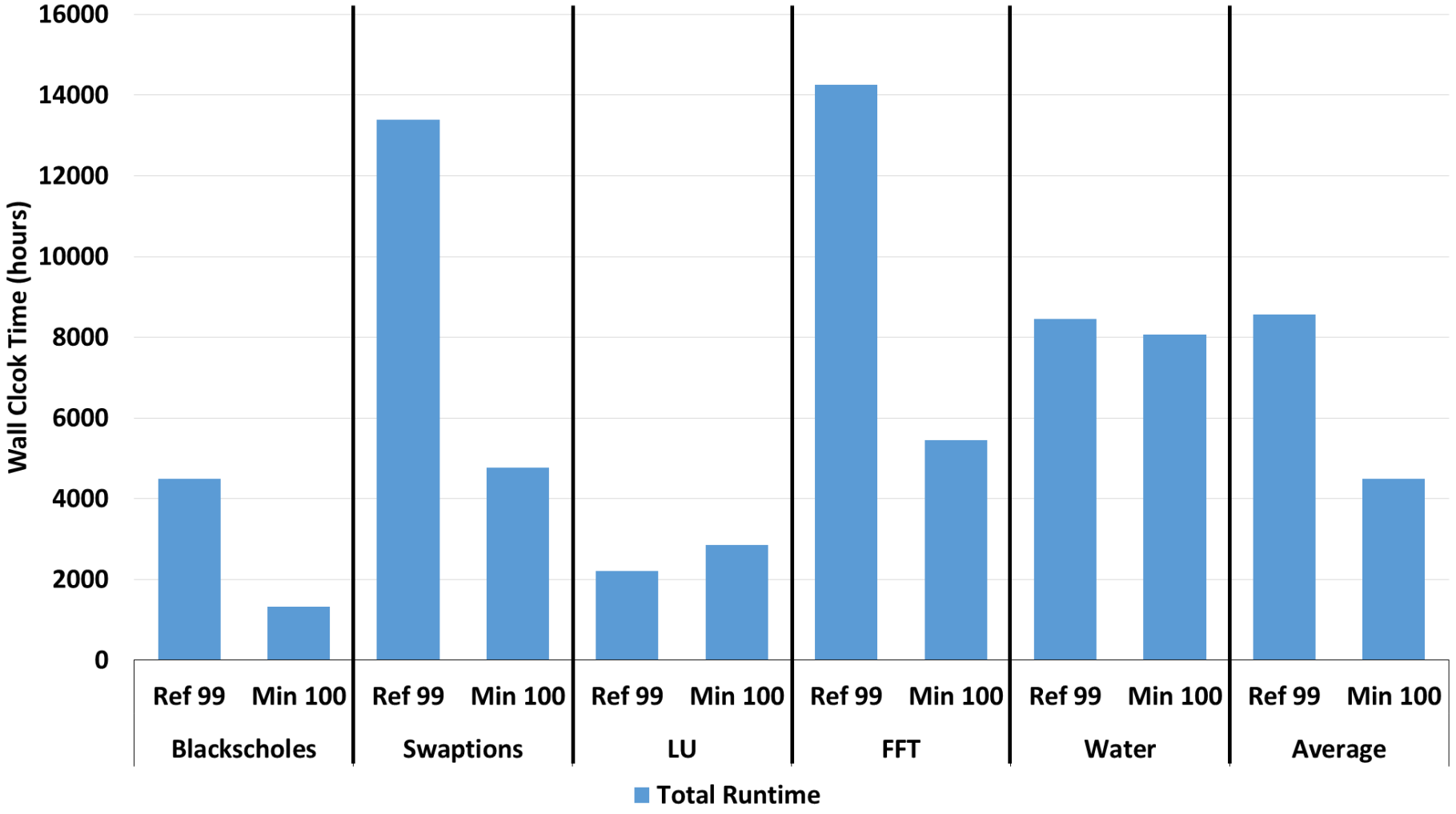
```
if(A or B) { … }
```

- For example:
  - PARSEC Blackscholes runs on 64K option prices
  - 1000 unique options
  - Only 21 options needed for 100% PC coverage
- Benefits:
  - Faster runtime of simulation (21 instead of 64k input)
  - Fewer dynamic instructions → Fewer error sites
  - Can target 100% analysis

Input Minimization



Speedup


16000
14000
12000
10000
8000
6000
4000
2000
0
Wall Clcok Time (hours)
Ref 99
Min 100
Blackscholes
Swaptions
LU
FFT
Water
Average
Total Runtime

Conclusion

- Addressing inputs should be a first-class citizen for approximation

- Software testing techniques can be used:
  - To assess quality, given a criterion for measurement
  - To improve performance of techniques and tools

- Going forward:
  - How can we incorporate the SW Testing workflow into the Approximation pipeline?