

# Summary

**Sink States:** $0(0 \times 10^0)$

Table 1: Pulse Analysis Summary

Classes	Methods	States	Unsatisfiable Clauses	Unreachable States	Possible concurrent Methods	Total. no. of pairs	No. of concurrent pairs	Percentage of concurrent Methods
JGFMolDynBenchSizeA	2	1	0	0	1	3	1	33
JGFInstrumentor	4	1	0	0	3	10	3	30
JGFMolDynBench	5	1	0	0	0	15	0	0
JGFTimer	2	1	0	0	0	3	0	0
md	3	1	0	0	0	6	0	0
particle	6	1	0	0	5	21	5	24
random	3	1	0	0	0	6	0	0
Total Classes=7	25	7	0	0	9	64	9	14

## Contents

<b>1</b>	<b>JGFLUFactBenchSizeB</b>	<b>3</b>
<b>2</b>	<b>JGFLUFactBench</b>	<b>4</b>
<b>3</b>	<b>JGFInstrumentor</b>	<b>5</b>
<b>4</b>	<b>Linpack</b>	<b>6</b>
<b>5</b>	<b>JGFTimer</b>	<b>7</b>
<b>6</b>	<b>Abbreviation</b>	<b>8</b>
<b>7</b>	<b>Annotated Version of Sequential Java Program generated by Sip4j</b>	<b>9</b>

# 1 JGFMolDynBenchSizeA

Table 2: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFMolDynBenchSizeA	✓
main	✓

Table 3: State Transition Matrix

	alive
alive	↑

Table 4: Methods Concurrency Matrix

	JGFMolDynBenchSizeA	main
JGFMolDynBenchSizeA	⌘	⌘
main	⌘	

## 2 JGFInstrumentor

Table 5: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFInstrumentor	✓
printHeader	✓
stopTimer	✓
addOpsToTimer	✓

Table 6: State Transition Matrix

	alive
alive	↑

Table 7: Methods Concurrency Matrix

	JGFInstrumentor	printHeader	stopTimer	addOpsToTimer
JGFInstrumentor	✗	✗	✗	✗
printHeader	✗			
stopTimer	✗		✗	✗
addOpsToTimer	✗		✗	✗

### 3 JGMolDynBench

Table 8: Methods Requires Clause Satisfiability

Method	Satisfiability
JGMolDynBench	✓
JGRun	✓
JGApplication	✓
JGtidyup	✓
JGInitialise	✓

Table 9: State Transition Matrix

	alive
alive	↑

Table 10: Methods Concurrency Matrix

	JGMolDynBench	JGRun	JGApplication	JGtidyup	JGInitialise
JGMolDynBench	⌘	⌘	⌘	⌘	⌘
JGRun	⌘	⌘	⌘	⌘	⌘
JGApplication	⌘	⌘	⌘	⌘	⌘
JGtidyup	⌘	⌘	⌘	⌘	⌘
JGInitialise	⌘	⌘	⌘	⌘	⌘

## 4 JGFTimer

Table 11: Methods Requires Clause Satisfiability

Method	Satisfiability
JGFTimer	✓
stop	✓

Table 12: State Transition Matrix

	alive
alive	↑

Table 13: Methods Concurrency Matrix

	JGFTimer	stop
JGFTimer	⧻	⧻
stop	⧻	⧻

## 5 **md**

Table 14: Methods Requires Clause Satisfiability

Method	Satisfiability
md	✓
initialise	✓
runiters	✓

Table 15: State Transition Matrix

	alive
alive	↑

Table 16: Methods Concurrency Matrix

	md	initialise	runiters
md	⧻	⧻	⧻
initialise	⧻	⧻	⧻
runiters	⧻	⧻	⧻

## 6 particle

Table 17: Methods Requires Clause Satisfiability

Method	Satisfiability
particle	✓
force	✓
mkekin	✓
velavg	✓
dscal	✓
domove	✓

Table 18: State Transition Matrix

	alive
alive	↑

Table 19: Methods Concurrency Matrix

	particle	force	mkekin	velavg	dscal	domove
particle	⌘	⌘	⌘	⌘	⌘	⌘
force	⌘	⌘	⌘		⌘	⌘
mkekin	⌘	⌘	⌘		⌘	⌘
velavg	⌘					
dscal	⌘	⌘	⌘		⌘	⌘
domove	⌘	⌘	⌘		⌘	⌘



## 7 random

Table 20: Methods Requires Clause Satisfiability

Method	Satisfiability
random	✓
seed	✓
update	✓

Table 21: State Transition Matrix

	alive
alive	↑

Table 22: Methods Concurrency Matrix

	random	seed	update
random	⌘	⌘	⌘
seed	⌘	⌘	⌘
update	⌘	⌘	⌘

## 8 Abbreviation

Table 23: Used Abbreviation

Symbol	Meaning
✓	requires clause of the method is satisfiable
✗	requires clause of the method is unsatisfiable
↑	The row-state can be transitioned to the column-state
✕	The row-state cannot be transitioned to the column-state
	The row-method can be possibly executed parallel with the column-method
⧻	The row-method cannot be executed parallel with the column-method

## 9 Annotated Version of Sequential Java Program generated by Sip4j

```
1 package outputs;
2 import edu.cmu.cs.plural.annot.*;
3
4 @ClassStates({@State(name = "alive")})
5 class JGFMolDynBenchSizeA {
6   @Perm(ensures="unique(this) in alive")
7   JGFMolDynBenchSizeA() { }
8
9   @Perm(requires="none(this) in alive",
10  ensures="unique(this) in alive")
11   void main(String argv[]) {
12   }
13
14 }ENDOFCLASS
15
16 @ClassStates({@State(name = "alive")})
17
18 class JGFInstrumentor {
19   @Perm(ensures="unique(this) in alive")
20   JGFInstrumentor() { }
21
22   void printHeader(int section, int size) {
23   }
24   @Perm(requires="full(this) in alive",
25  ensures="full(this) in alive")
26   void stopTimer(String name) {
27   }
28   @Perm(requires="full(this) in alive",
29  ensures="full(this) in alive")
30   void addOpsToTimer(String name, double count) {
31   }
32
33 }ENDOFCLASS
34
35 @ClassStates({@State(name = "alive")})
36
37 class JGFMolDynBench {
38   @Perm(ensures="unique(this) in alive")
39   JGFMolDynBench() { }
40
41   @Perm(requires="unique(this) in alive",
42  ensures="unique(this) in alive")
43   public void JGFrunk(int size) {
44   }
45   @Perm(requires="full(this) in alive",
46  ensures="full(this) in alive")
47   public void JGFApplication() {
48   }
49   @Perm(requires="unique(this) in alive",
50  ensures="unique(this) in alive")
51   public void JGFtidyup() {
52   }
53   @Perm(requires="unique(this) in alive",
54  ensures="unique(this) in alive")
55   public void JGFinitialise() {
56   }
57
58 }ENDOFCLASS
59
60 @ClassStates({@State(name = "alive")})
61
62 class JGFTimer {
63   @Perm(ensures="unique(this) in alive")
64   JGFTimer() { }
65
66   @Perm(requires="full(this) in alive",
67  ensures="full(this) in alive")
68   public void stop() {
69   }
70
71 }ENDOFCLASS
72
73 @ClassStates({@State(name = "alive")})
```

```

76 class md {
77   @Perm(ensures="unique(this) in alive")
78   md() {   }

80   @Perm(requires="unique(this) in alive",
81     ensures="unique(this) in alive")
82   public void initialise() {
83   }
84   @Perm(requires="full(this) in alive",
85     ensures="full(this) in alive")
86   public void runiters() {
87   }

89 }ENDOFCLASS

91 @ClassStates({@State(name = "alive")})

93 class particle {
94   @Perm(ensures="unique(this) in alive")
95   particle() {   }

97   @Perm(requires="full(this) in alive * pure(#0) in alive * pure(#1) in alive * pure(#2) in alive * pure
98     (#3) in alive",
99     ensures="full(this) in alive * pure(#0) in alive * pure(#1) in alive * pure(#2) in alive * pure(#3) in
100     alive")
101   public void force(double side, double rcoeff, int mdsiz, int x) {
102   }
103   @Perm(requires="full(this) in alive * pure(#0) in alive",
104     ensures="full(this) in alive * pure(#0) in alive")
105   public double mkekin(double hsq2) {
106     return 0;
107   }
108   @Perm(requires="pure(this) in alive * pure(#0) in alive * pure(#1) in alive",
109     ensures="pure(this) in alive * pure(#0) in alive * pure(#1) in alive")
110   public double velavg(double vaverh, double h) {
111     return 0;
112   }
113   @Perm(requires="full(this) in alive * pure(#0) in alive",
114     ensures="full(this) in alive * pure(#0) in alive")
115   public void dscal(double sc, int incx) {
116   }
117   @Perm(requires="full(this) in alive * pure(#0) in alive",
118     ensures="full(this) in alive * pure(#0) in alive")
119   public void domove(double side) {
120   }

122 }ENDOFCLASS

124 @ClassStates({@State(name = "alive")})

126 class random {
127   @Perm(ensures="unique(this) in alive")
128   random() {   }

130   @Perm(requires="full(this) in alive",
131     ensures="full(this) in alive")
132   public double seed() {
133     return 0;
134   }
135   @Perm(requires="full(this) in alive",
136     ensures="full(this) in alive")
137   public double update() {
138     return 0;
139   }

141 }ENDOFCLASS

```