# Parallel vs Serial Fault Simulator

Michalis Christou

May 7, 2020

## Introduction

The purpose of this project is to write a fault simulator in C++ that has both parallel and serial modes of operation. The goal is to examine the speedup of the parallel implementation compared to the serial version as well as compare any potantial speeups that can occur with or without fault dropping in both cases.

## Backround

### Single Stuck at Fault Model

Faults in circuits can be modeled in many ways. The model that was used for the purposes of this project was the single stuck at fault model. In this case the circuit, which is a gate level netlist, has only one faulty line that is either stuck permanently at 1 or 0 and the fault can be at an input or output of a gate. The number of possible fault sites in a given netlist is $\#PI + \#GATES + \#(fanoutbranches)$ and subsequentltly since a fault site has 2 potential faults, the number of possible faults in a circuit is $2 \cdot \#(faultsites)$.

### Fault Equivalence

Despite a circuit having a maximum number of possible stuck at faults, using some mechanisms we can choose a subset of these faults and if we are able to detect these it means that we can also detect the ones we skipped. For example, fault equivalence means that two faults $f_1, f_2$ are equivalent if all tests that detect $f_1$ also detect $f_2$ and vice versa. This means that if we are able to detect one of the two faults then we know doe sure that we can detect the other. Thus we can keep only one of these equivalent faults in our list. This is called fault collapsing. The results is a collapsed fault set that contains one fault from each equivalence subset.

### Fault Dominance

The definition of fault dominance is as follows, if all tests of some fault $f_1$ detect another fault $f_2$ then $f_2$ is said to dominate $f_1$. The rule that we use is if fault $f_2$ dominates $f_1$ then $f_2$ is removed from the fault list.

### Checkpoint Theorem

The checkpoint theorem is the result of Fault Equivalence and Fault Dominance. Primary inputs and fanout branches of a combinational circuit are called checkpoints. The theorem suggests that a test set that detects all single stuck at faults on all checkpoints of a combinational circuit also detects all single stuck at faults in that circuit. The checkpoints in a circuit are considered to be primary inputs and fanout branches. This means that if we only consider these fault sites and detect all faults in them, we can also detect all possible stuck at faults in that circuit.

## Fault Simulation

Fault simulation is done in order to detect possible faults in a Boolean circuit. The algorithm is fairly simple and intuitive. First a set of test vectors is obtained. Then using these test vectors the non-faulty circuit response is computed. Then for each stuck at fault that is considered, the fault is injected in the circuit, the fault output is computed and compared to the normal non-faulty output. If the outputs differ, then the fault can be detected. In this case a detected fault counter is incremented. If fault dropping is enabled, then the stuck at fault is deleted from the set. After this is repeated for all test vectors and all faults in the set the fault coverage is computed which is the number of detected faults divided by the total number of faults considered.

$$FC = \frac{\#fd}{\#tf}$$

The pseudocode for the serial algorithm is depicted below.

**Algorithm 1** Serial Fault Simulation

```
#fd = 0, #tf = |F|
for every test t ∈ T do
    Rt = true_value_simulation(t, C)
    for every fault f ∈ F do
        inject_fault(C, f)
        Rf = fault_simulation(C, f)
        if Rt ≠ Rf then
            #rf++
            F = drop_fault(t, F)
        end if
    end for
end for
return FC = compute_fault_coverage(#fd, #tf)
```

# Implementation

This section is dedicated to explain the details and inner workings of the specific C++ implementation. The code is designed to parse `.bench` files that represent gate level netlists as well as test vectors of different formats. Then the code generates all faults according to the Checkpoint Theorem. Depending on the user input the program either runs the serial algorithm or parallel with or without fault dropping.

## Netlist Representation

The netlist is internally represented as a DAG (Directed Acyclic Graph). Specifically, since the purpose of the program is to simulate faults, the graph model that was chosen was $G_2$. This model uses one graph node per circuit line and one edge per pair of gates/fanout stems with direct connection, in order to explicitly represent fanout stems. This is necessary in this case because faults could happen in any circuit line thus explicit representation of every line is needed. The data structure used in this case was a $1 - D$ array with linked lists, since generally speaking, Boolean circuits are sparse thus more memory is saved compared to using a $2 - D$ array.

## Parallelizing Fault Simulation

When parallelizing algorithms a lot has to be considered. Data dependencies is one aspect that leads to race conditions which in turn, leads to unpredictable code behaviour. In this case, the most important part of parallelizing the algorithm is to ensure that no data dependencies or race conditions would occur. In this case this was achieved easily by splitting the work in equal chunks to $N$ threads. Since each iteration is almost fully independent splitting the workflow to any number of threads was trivial once the serial algorithm was implemented. The workflow was split as follows: Each thread got a subset of the test vectors, then independently worked on identifying all faults that were previously generated with the checkpoint theorem and stored in a global variable. When using fault dropping threads that detect faults need to modify the global variable (that stores faults) in order to save time for other threads. To ensure that threads would not interfere with one another the global data structure was protected with a mutex since it was part of a critical region, but this was only a problem when using fault dropping. Without fault dropping each thread can work fully independent from any other one.

# Results

## Netlist Stats

The table below shows some useful netlist statiastics for all simulated circuits. The table includes circuit data from both ISCAS85 and ISCAS89.

| Circuit | NAND | AND | NOR | NOT | BUFF | PI | PO | Paths | KB |
|---|---|---|---|---|---|---|---|---|---|
| s27 | 1 | 1 | 4 | 2 | 0 | 7 | 4 | 28 | 5 |
| s386 | 0 | 83 | 0 | 41 | 0 | 13 | 13 | 207 | 77 |
| s510 | 61 | 34 | 55 | 32 | 0 | 25 | 13 | 369 | 100 |
| s641 | 4 | 90 | 0 | 272 | 0 | 54 | 42 | 1722 | 130 |
| s838.1 | 57 | 105 | 70 | 158 | 0 | 66 | 33 | 1714 | 186 |
| s298 | 9 | 31 | 19 | 44 | 0 | 17 | 20 | 231 | 61 |
| s382 | 30 | 11 | 34 | 59 | 0 | 24 | 27 | 400 | 78 |
| s400 | 36 | 11 | 34 | 56 | 0 | 24 | 27 | 448 | 82 |
| s526 | 22 | 56 | 35 | 52 | 0 | 24 | 27 | 410 | 106 |
| s526 | 22 | 56 | 35 | 52 | 0 | 24 | 27 | 410 | 106 |
| s713 | 28 | 94 | 0 | 254 | 0 | 54 | 42 | 21812 | 145 |
| s344 | 18 | 44 | 30 | 59 | 0 | 24 | 17 | 344 | 65 |
| s344.1 | 18 | 44 | 30 | 59 | 0 | 24 | 17 | 344 | 65 |
| s420.1 | 29 | 49 | 34 | 78 | 0 | 34 | 17 | 474 | 91 |
| s820 | 54 | 76 | 66 | 33 | 0 | 23 | 24 | 492 | 163 |
| s953 | 114 | 49 | 112 | 84 | 0 | 45 | 52 | 1156 | 193 |
| s208.1 | 15 | 21 | 16 | 38 | 0 | 18 | 9 | 142 | 43 |
| s349 | 19 | 44 | 31 | 57 | 0 | 24 | 17 | 354 | 66 |
| s444 | 58 | 13 | 34 | 62 | 0 | 24 | 27 | 535 | 90 |
| s1238 | 125 | 134 | 57 | 80 | 0 | 32 | 32 | 3559 | 245 |
| s1494 | 0 | 354 | 0 | 89 | 0 | 14 | 25 | 976 | 293 |
| s5378 | 0 | 0 | 765 | 1775 | 15 | 214 | 228 | 13542 | 1064 |
| s832 | 54 | 78 | 66 | 25 | 0 | 23 | 24 | 506 | 165 |
| s1423 | 64 | 197 | 92 | 167 | 0 | 91 | 79 | 44726 | 289 |
| s1196.1 | 119 | 118 | 50 | 141 | 0 | 32 | 32 | 3098 | 236 |
| s1488 | 0 | 350 | 0 | 103 | 0 | 14 | 25 | 962 | 292 |
| s9234 | 528 | 955 | 113 | 3570 | 0 | 247 | 250 | 244854 | 1823 |
| s13207 | 849 | 1114 | 98 | 5378 | 0 | 700 | 790 | 1345369 | 2683 |
| s15850 | 968 | 1619 | 151 | 6324 | 0 | 611 | 684 | 164738046 | 3176 |
| s38417 | 2050 | 4154 | 2279 | 13470 | 0 | 1664 | 1742 | 1391579 | 7703 |
| s38584 | 2126 | 5516 | 1185 | 7805 | 0 | 1464 | 1730 | 1080723 | 7734 |
| s35932 | 7020 | 4032 | 0 | 3861 | 0 | 1763 | 2048 | 197141 | 7249 |
| c17 | 6 | 0 | 0 | 0 | 0 | 5 | 2 | 11 | 3 |
| c880 | 87 | 117 | 61 | 63 | 26 | 60 | 26 | 8642 | 174 |
| c1355 | 416 | 56 | 0 | 40 | 32 | 41 | 32 | 4173216 | 267 |
| c1908 | 377 | 63 | 1 | 277 | 187 | 33 | 25 | 729056 | 376 |
| c2670 | 254 | 332 | 12 | 321 | 305 | 233 | 140 | 679954 | 560 |
| c3540 | 298 | 495 | 68 | 490 | 246 | 50 | 22 | 28265874 | 689 |
| c5315 | 454 | 718 | 27 | 581 | 419 | 178 | 123 | 1341305 | 1068 |
| c6288 | 0 | 256 | 2128 | 32 | 0 | 32 | 32 | 1101055638 | 1218 |
| c7552 | 1028 | 776 | 54 | 876 | 593 | 207 | 108 | 726494 | 1486 |

### Serial Results

| Circuit | Time (s) | FC(%) | FD | Thr |
|---|---|---|---|---|
| s27 | 0.020549 | 87.50% | NO | 1 |
| s386 | 19.3314 | 25.99% | NO | 1 |
| s510 | 29.8353 | 88.96% | NO | 1 |
| s641 | 12.2275 | 62.21% | NO | 1 |
| s838.1 | 119.042 | 32.52% | NO | 1 |
| s298 | 4.39425 | 74.02% | NO | 1 |
| s382 | 7.73878 | 72.77% | NO | 1 |
| s400 | 8.28644 | 66.60% | NO | 1 |
| s526 | 30.6785 | 57.81% | NO | 1 |
| s526 | 30.6707 | 64.11% | NO | 1 |
| s713 | 16.9303 | 61.72% | NO | 1 |
| s344 | 2.4745 | 70.18% | NO | 1 |
| s344.1 | 2.47505 | 70.18% | NO | 1 |
| s420.1 | 16.2829 | 41.67% | NO | 1 |
| s820 | 140.13 | 42.18% | NO | 1 |
| s953 | 144.841 | 57.80% | NO | 1 |
| s208.1 | 2.34383 | 58.33% | NO | 1 |
| s349 | 2.57983 | 72.35% | NO | 1 |
| s444 | 9.96745 | 68.44% | NO | 1 |
| s27 | 0.008762 | 87.50% | YES | 1 |
| s386 | 16.0373 | 25.99% | YES | 1 |
| s510 | 9.66769 | 88.96% | YES | 1 |
| s641 | 6.6703 | 62.21% | YES | 1 |
| s838.1 | 83.4562 | 32.52% | YES | 1 |
| s298 | 2.30052 | 74.02% | YES | 1 |
| s382 | 3.67129 | 72.77% | YES | 1 |
| s400 | 4.37297 | 66.60% | YES | 1 |
| s526 | 16.823 | 57.81% | YES | 1 |
| s526 | 16.3637 | 64.11% | YES | 1 |
| s713 | 9.17699 | 61.72% | YES | 1 |
| s344 | 1.29709 | 70.18% | YES | 1 |
| s344.1 | 1.29469 | 70.18% | YES | 1 |
| s420.1 | 10.5001 | 41.67% | YES | 1 |
| s820 | 106.441 | 42.18% | YES | 1 |
| s953 | 80.7145 | 57.80% | YES | 1 |
| s208.1 | 1.27442 | 58.33% | YES | 1 |
| s349 | 1.54368 | 72.35% | YES | 1 |
| s444 | 5.74283 | 68.44% | YES | 1 |

### Parallel Results

| Circuit | Time (s) | FC(%) | FD | Thr |
|---|---|---|---|---|
| s27 | 0.018411 | 87.50% | NO | 2 |
| s386 | 17.3113 | 25.99% | NO | 2 |
| s510 | 26.352 | 88.96% | NO | 2 |
| s641 | 10.3201 | 62.21% | NO | 2 |
| s838.1 | 109.733 | 32.52% | NO | 2 |
| s298 | 3.82556 | 74.02% | NO | 2 |
| s382 | 6.72619 | 72.77% | NO | 2 |
| s400 | 7.27337 | 66.60% | NO | 2 |
| s526 | 28.301 | 57.81% | NO | 2 |
| s526 | 28.087 | 64.11% | NO | 2 |
| s713 | 16.0003 | 61.72% | NO | 2 |
| s344 | 2.16093 | 70.18% | NO | 2 |
| s344.1 | 2.22647 | 70.18% | NO | 2 |
| s420.1 | 15.1997 | 41.67% | NO | 2 |
| s820 | 130.983 | 42.18% | NO | 2 |
| s953 | 133.918 | 57.80% | NO | 2 |
| s208.1 | 2.19149 | 58.33% | NO | 2 |
| s349 | 2.15274 | 72.35% | NO | 2 |
| s444 | 7.97324 | 68.44% | NO | 2 |
| s27 | 0.006698 | 87.50% | YES | 2 |
| s386 | 15.0778 | 25.99% | YES | 2 |
| s510 | 9.21763 | 88.96% | YES | 2 |
| s641 | 5.77095 | 62.21% | YES | 2 |
| s838.1 | 79.2006 | 32.52% | YES | 2 |
| s298 | 1.7277 | 74.02% | YES | 2 |
| s382 | 3.10285 | 72.77% | YES | 2 |
| s400 | 3.93602 | 66.60% | YES | 2 |
| s526 | 15.9112 | 57.81% | YES | 2 |
| s526 | 15.0249 | 64.11% | YES | 2 |
| s713 | 8.43757 | 61.72% | YES | 2 |
| s344 | 1.42344 | 70.18% | YES | 2 |
| s344.1 | 1.13892 | 70.18% | YES | 2 |
| s420.1 | 10.4531 | 41.67% | YES | 2 |
| s820 | 95.0126 | 42.18% | YES | 2 |
| s953 | 71.9942 | 57.80% | YES | 2 |
| s208.1 | 0.943637 | 58.33% | YES | 2 |
| s349 | 1.38595 | 72.35% | YES | 2 |
| s444 | 5.07519 | 68.44% | YES | 2 |