Efficient, Large-Scale Computation Techniques for the Evaluation of Side Channel Attacks

MOHAMMED MOHIUDDIN

FRANKFURT UNIVERSITY OF APPLIED SCIENCES
Informatik and Ingenieurwissenschaften

INDUSTRIAL SUPERVISOR
Mr. **ROBERT SZERWINSKI**

ACADEMIC SUPERVISOR
Prof. Dr. **CHRISTIAN BAUN**
Prof. Dr. **EICKE GODEHARDT**

May 30, 2016

Contents

1. Introduction
   - 1.1 Need for Standard Testing Methodology
   - 1.2 Motivation of Thesis
2. Concept and Design
   - 2.1 Design for Implementation on Single Node
   - 2.2 Design for Implementation on Cluster
3. Implementation
   - 3.1 Implementation on Single Node
   - 3.2 Implementation on Cluster
4. Results and Conclusion
   - 4.1 Results
   - 4.2 Conclusion

Side Channel Attacks

- Side channel attacks pose a threat to the security of cryptographic devices.
- Types of Attacks: Timing attacks, power monitoring attacks, electromagnetic attacks etc.

Testing the Vulnerability of Cryptographic Devices

- Evaluation labs and producers need to test the vulnerability of the devices in reasonable time and effort.
- Welch’s t test is recommended by the Cryptographic Research Inc.
- Size of power traces is large.
- Amount of time required to compute t parameters is large.

Motivation

- Design and develop and algorithm to perform t test using parallel computations to reduce the time of execution.
- Reduce the execution time in three ways.
    - No of passes
    - Chunk size for test
    - Parallel Execution

Concept and Design

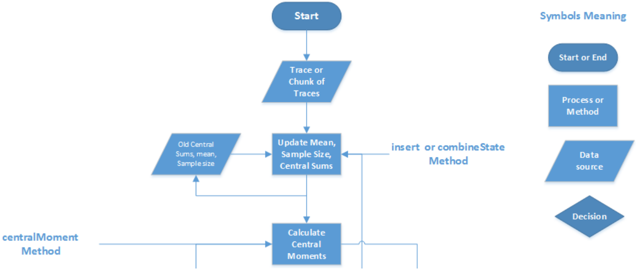
One Pass Approach

- One pass algorithm using raw moments has stability problem.
- Tobias Schneider and Amir Moradi's suggested one pass approach, avoids instabilities.

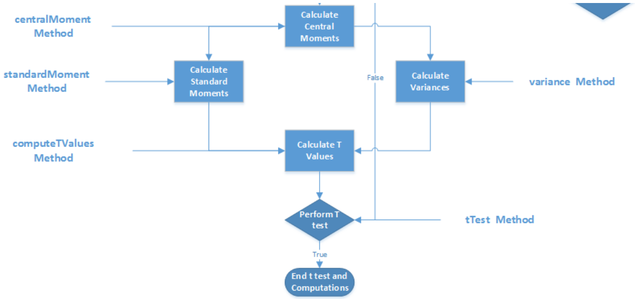
Steps Involved in Performing t test

- Central Sums
- Central Moments
- Standard Moments
- Variance
- Calculate t values
- Perform t test

Flow Diagram

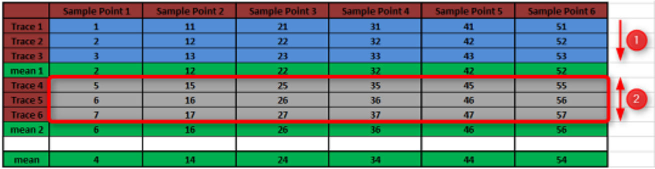

Start
Trace or Chunk of Traces
Update Mean, Sample Size, Central Sums
Old Central Sums, mean, Sample size
insert or combineState Method
Calculate Central Moments
centralMoment Method
Symbols Meaning
Start or End
Process or Method
Data source
Decision

Flow Diagram



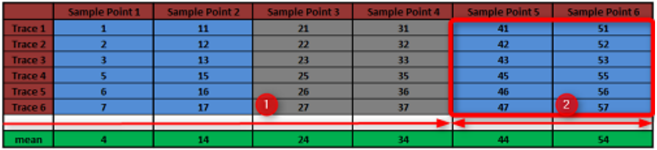
Traces

- Usual size of power traces: 100’s of millions of traces and thousands of sample points.
- Suppose: 100 million traces and 100 thousand sample points.



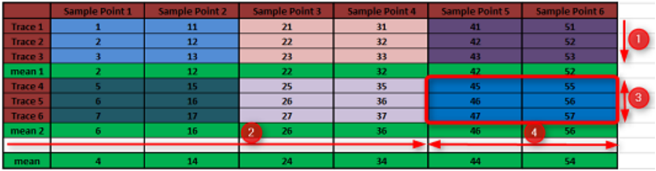
	Sample Point 1	Sample Point 2	Sample Point 3	Sample Point 4	Sample Point 5	Sample Point 6
Trace 1	1	11	21	31	41	51
Trace 2	2	12	22	32	42	52
Trace 3	3	13	23	33	43	53
mean 1	2	12	22	32	42	52
Trace 4	5	15	25	35	45	55
Trace 5	6	16	26	36	46	56
Trace 6	7	17	27	37	47	57
mean 2	6	16	26	36	46	56
mean	4	14	24	34	44	54

Sample Points



	Sample Point 1	Sample Point 2	Sample Point 3	Sample Point 4	Sample Point 5	Sample Point 6
Trace 1	1	11	21	31	41	51
Trace 2	2	12	22	32	42	52
Trace 3	3	13	23	33	43	53
Trace 4	5	15	25	35	45	55
Trace 5	6	16	26	36	46	56
Trace 6	7	17	27	37	47	57
mean	4	14	24	34	44	54

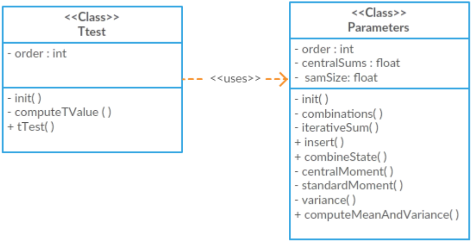
Traces and Sample Points



	Sample Point 1	Sample Point 2	Sample Point 3	Sample Point 4	Sample Point 5	Sample Point 6
Trace 1	1	11	21	31	41	51
Trace 2	2	12	22	32	42	52
Trace 3	3	13	23	33	43	53
mean 1	2	12	22	32	42	52
Trace 4	5	15	25	35	45	55
Trace 5	6	16	26	36	46	56
Trace 6	7	17	27	37	47	57
mean 2	6	16	26	36	46	56
mean	4	14	24	34	44	54

Implementation

Class Diagram



<<Class>>
Ttest

- order : int

- init( )
- computeTValue ( )
+ tTest( )

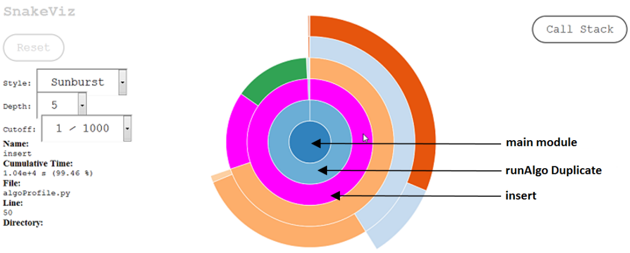
<<uses>>

<<Class>>
Parameters

- order : int
- centralSums : float
- samSize: float

- init( )
- combinations( )
- iterativeSum( )
+ insert( )
+ combineState( )
- centralMoment( )
- standardMoment( )
- variance( )
+ computeMeanAndVariance( )

Profiling Algorithm

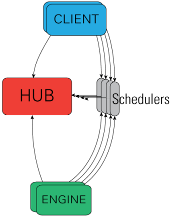


SnakeViz

Style: Sunburst
Depth: 5
Cutoff: 1 / 1000

Name:
insert
Cumulative Time:
1.04e+4 s (99.46 %)
File:
algoProfile.py
Line:
50
Directory:

IPython Parallel Environment



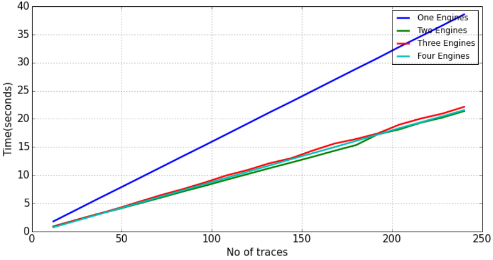
- Direct and Load balanced Views
- Client for single node
- Client for cluster

Results and Conclusion

Sample Execution of the Algorithm

- Size of the power traces considered is 2000 traces and 8400 sample points.
- Single Node: Time taken is approximately 110 minutes.
- On Cluster: Time taken is approximately 21 minutes.
- Speed-Up achieved on cluster is around 5 times faster to that of single node.

Hyper-threading



Conclusion and Future Enhancements

- Hyper-threading is not recommended for the developed algorithm.
- Considerable execution time can be reduced by increasing the number of machines used.
- This work tests the univariate leakages, it can be extended to test multivariate leakages.
- Same work can be implemented on a cluster of Graphical Processing Unit processors.