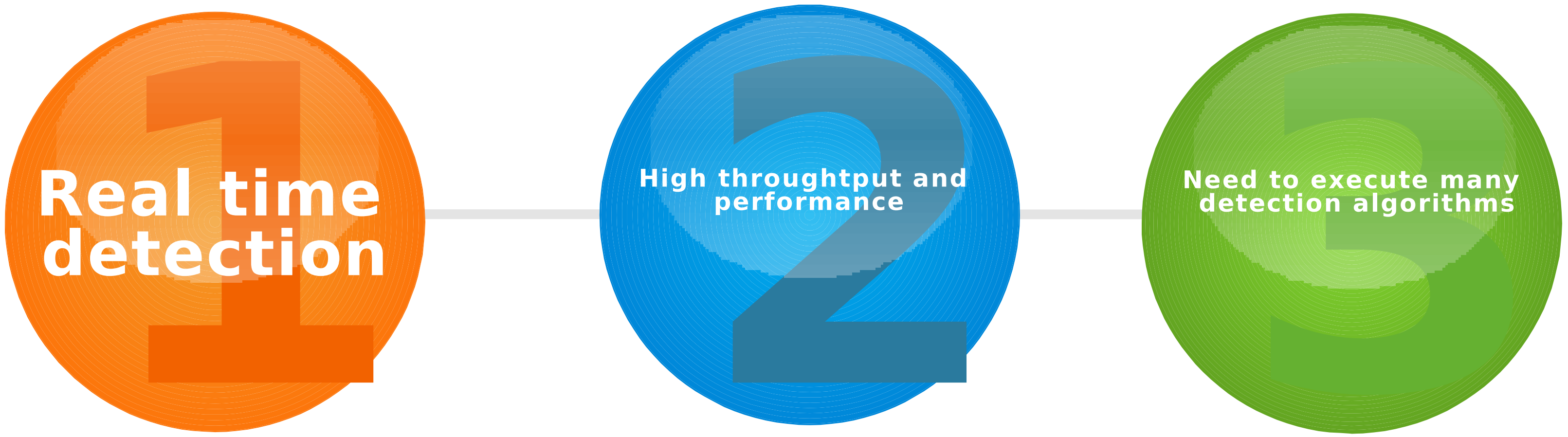
Manel Abdellatif

introducing

Parallel programming for accelerating Malware detection

Motivations

- Ever-growing number of threats
- The market boost of embedded systems
- The increasing variety of operating systems
- Malware detection is a highly common and computationally-intensive problem in intrusion detection systems


1
Real time detection
2
High throughtput and performance
3
Need to execute many detection algorithms

What about small-scale systems?

- Great use of small-scale systems : mobile phones, gaming consoles, SoC etc.
- Memory and computation performance constraints
- Ever growth of attacks on small-scale systems
- Improvement in parallel computation performance

Security threats

vs

Parallel performance computation

- How to get benefit from parallel architectures to monitor small-scale systems?

Previous Work

- Development of parallel architecture for malware detection based on pattern matching technique
- Achieving better computing performance
- Use of Cuda and desktop GPU

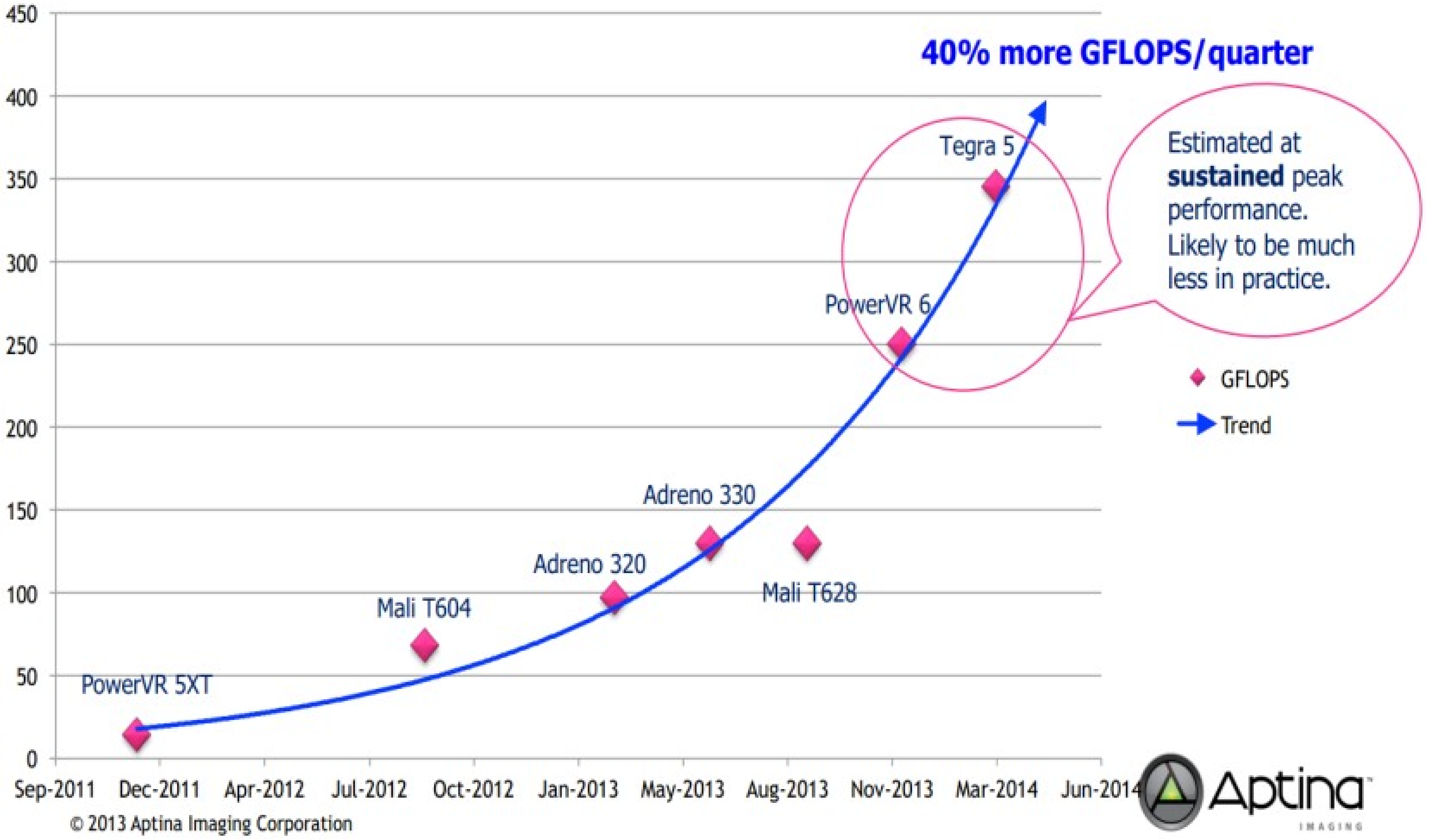
Current Work

- Migration to mobile platform
- Use of OpenCL
- Building of behavioral malware dataset based on syscalls patterns
- Development of memory optimization techniques

Benefits of mobile GPGPU

- ✓ To ensure high security level of mobile devices, accelerating malware detection can be provided by GPU parallel processing
- ✓ Offering a complementary processing unit to the CPU
- ✓ Adapted to SIMD architecture
- ✓ Fast memory types access (shared memory , constant memory)
- ✓ More and more evolving
- ✓ GPUs driven by high-end applications: prepared to

Evolution of GPUs for embedded systems



Popular mobile GPUs

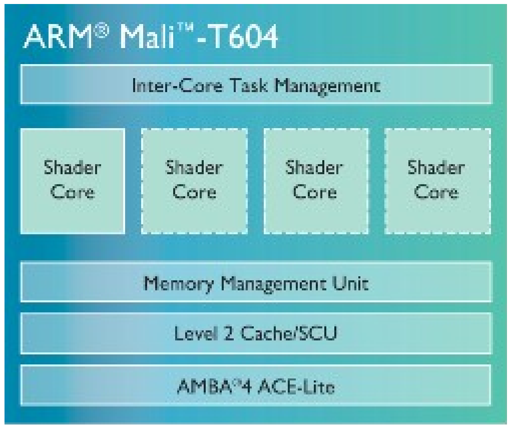
Adreno 330

-Inbuilt in Snapdragon™ 800 Series Processors.
-speed can push to 3.6 gig pixels/s
-Used in HTC one, Xperia z ultra

Power VR SGX544mp3

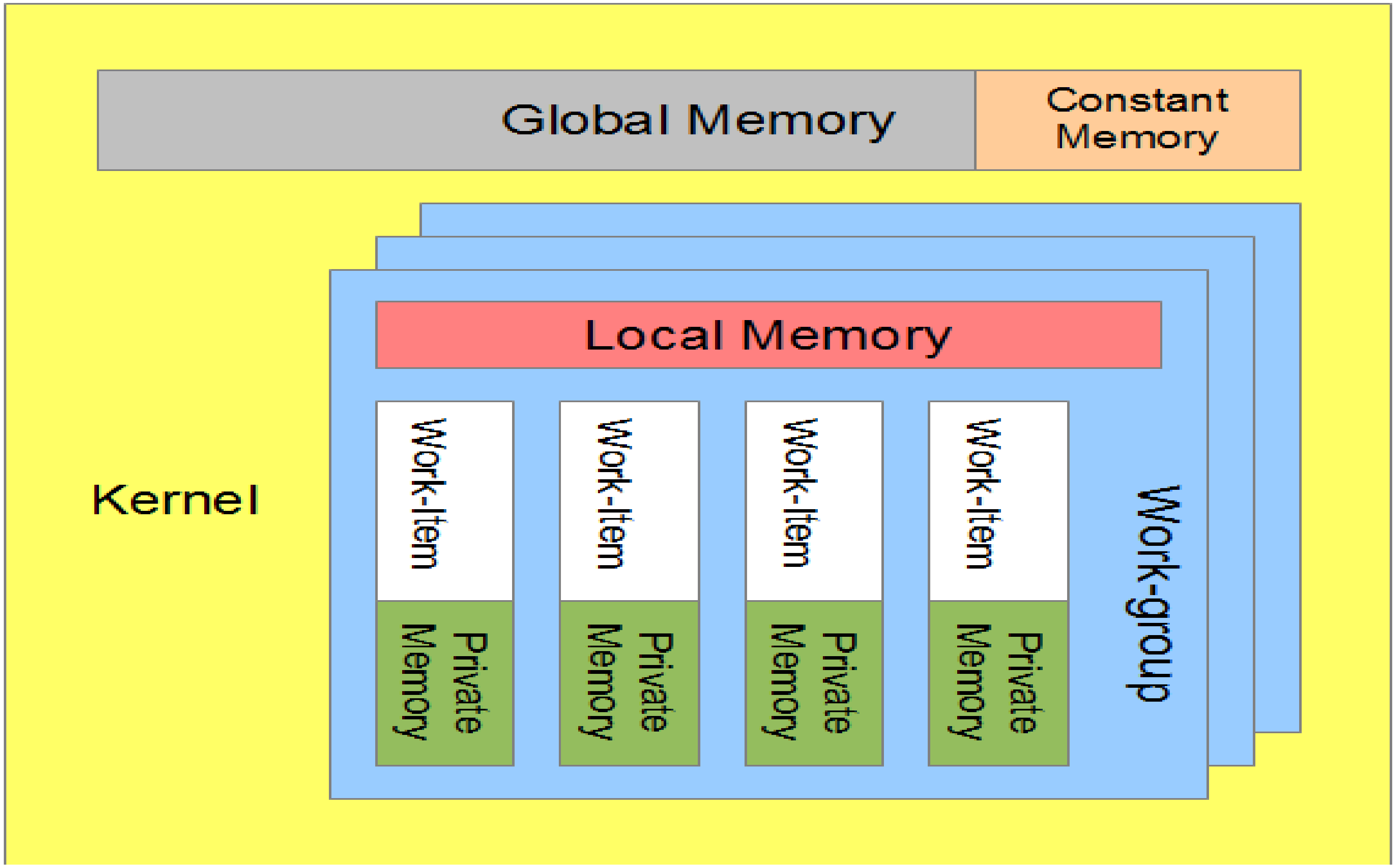
-Inbuild in Exynos 5 Octa processor
-used in galaxy s4 @ 533 MHz clock speed

Mali T604

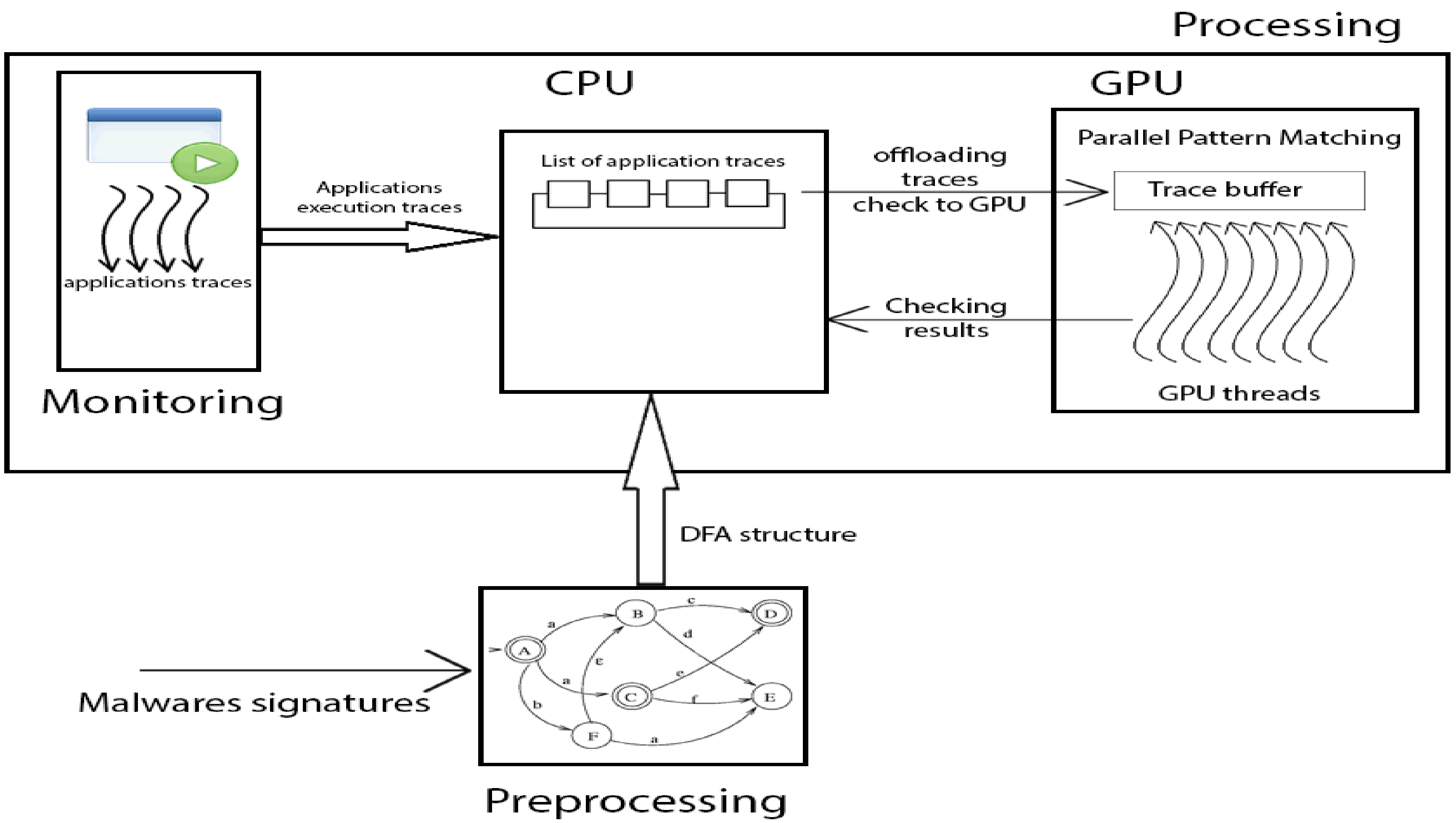


-First time used in Exynos 5
-The 1st Midgard architecture gpu for arm
-Used in famous series of Google tablet nexus 10.

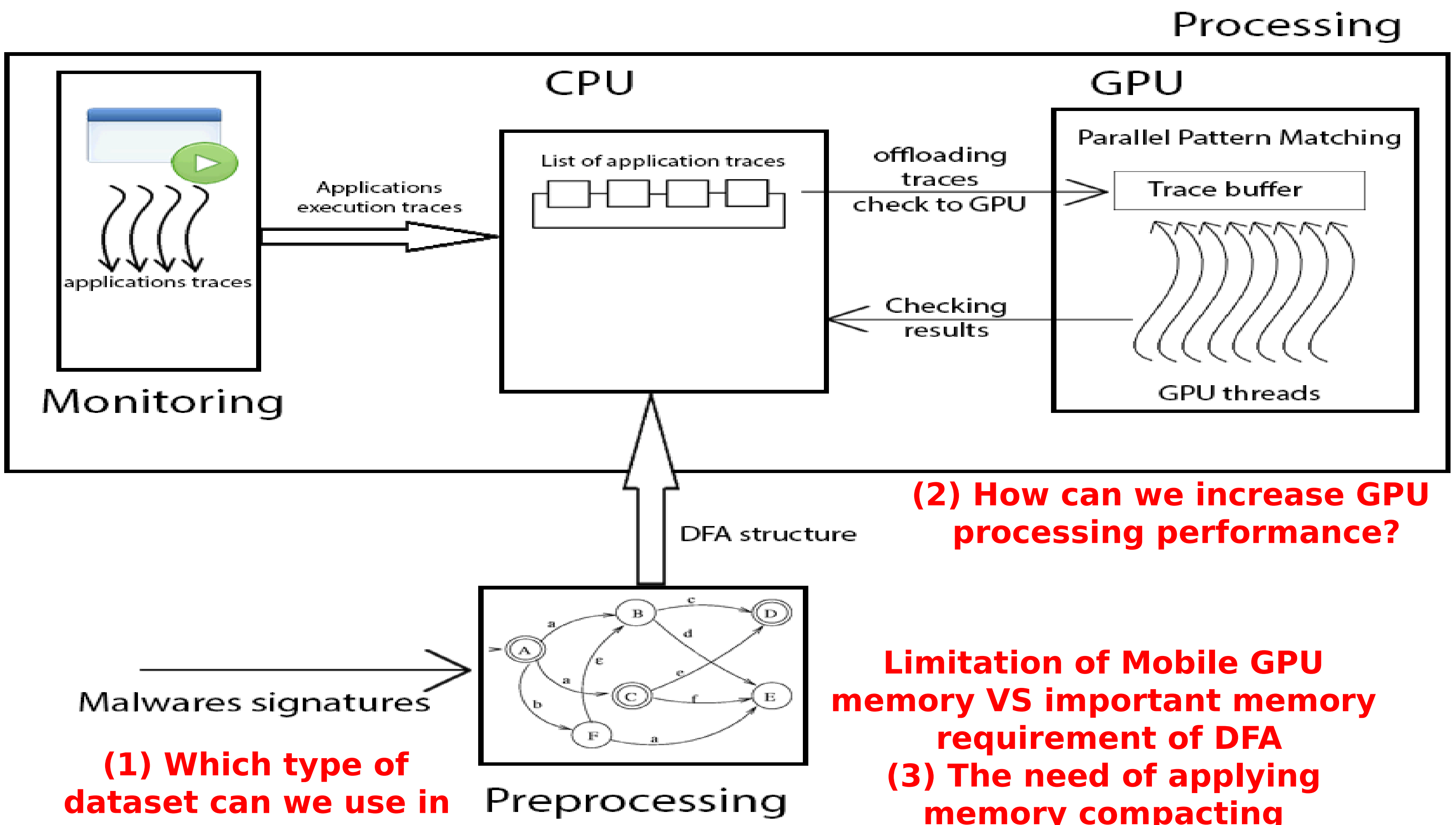
GPU architecture



Architecture



Architecture Challenges



(2) How can we increase GPU processing performance?

(1) Which type of dataset can we use in order to have good detection accuracy?

Limitation of Mobile GPU memory VS important memory requirement of DFA
(3) The need of applying memory compacting techniques

Challenge 1

Which type of dataset can we use in order to have good detection accuracy?

Malware detection

Extraction of malicious behavior based on syscalls sequences with the thread-grained extraction technique

- Key concept: Malwares which have the same malicious code embedded on benign applications, will have common malicious behaviour
- Tracking the tree architecture of applications threads
- Malicious behaviour is likely to appear at the same thread level on applications having the same malware

Application 1

Application 2

Thread tree architecture of tow applications belo
To the same malware family

Malware detection

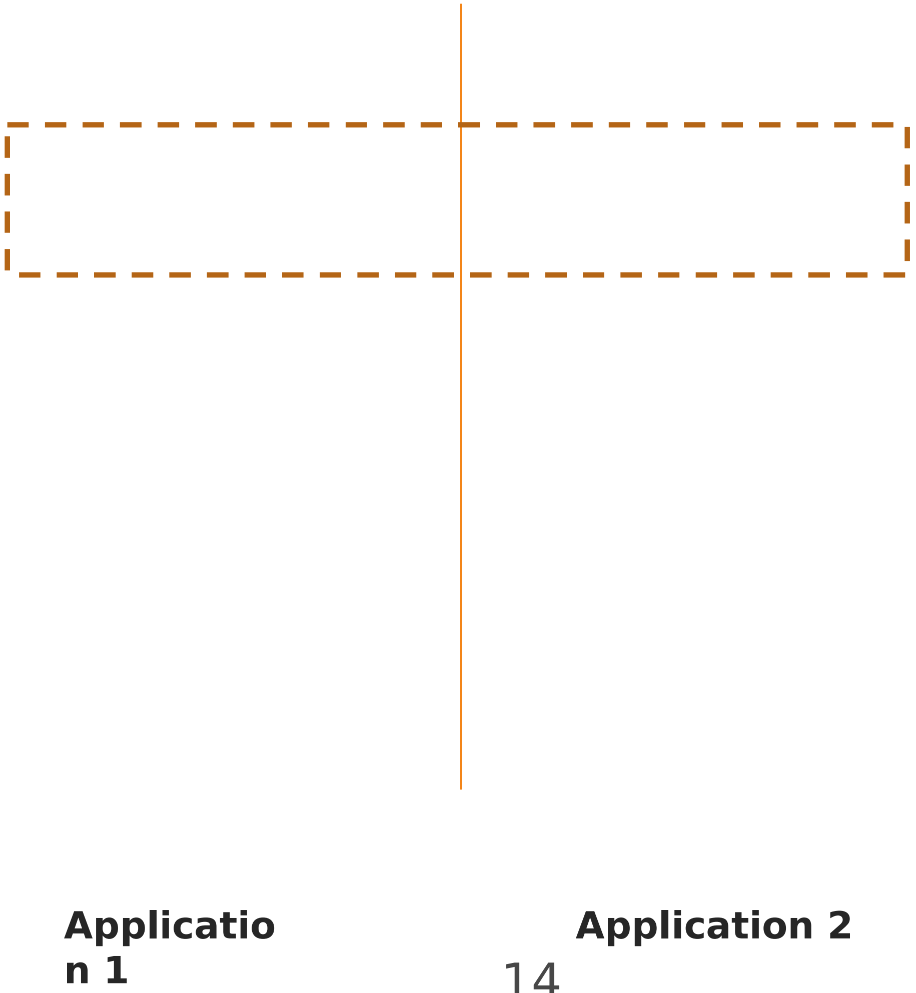
Training phase

❖ Recording Phase

- Having different applications that belongs to the same malware family
- Execution of every application
- Tracking thread tree structure of the application
- Recording syscalls sequences for every thread created by the application

❖ Extraction phase

- Extraction of common syscalls subsequences belonging to threads from the same family and having the same level
- Storage of common subsequences



Malware detection

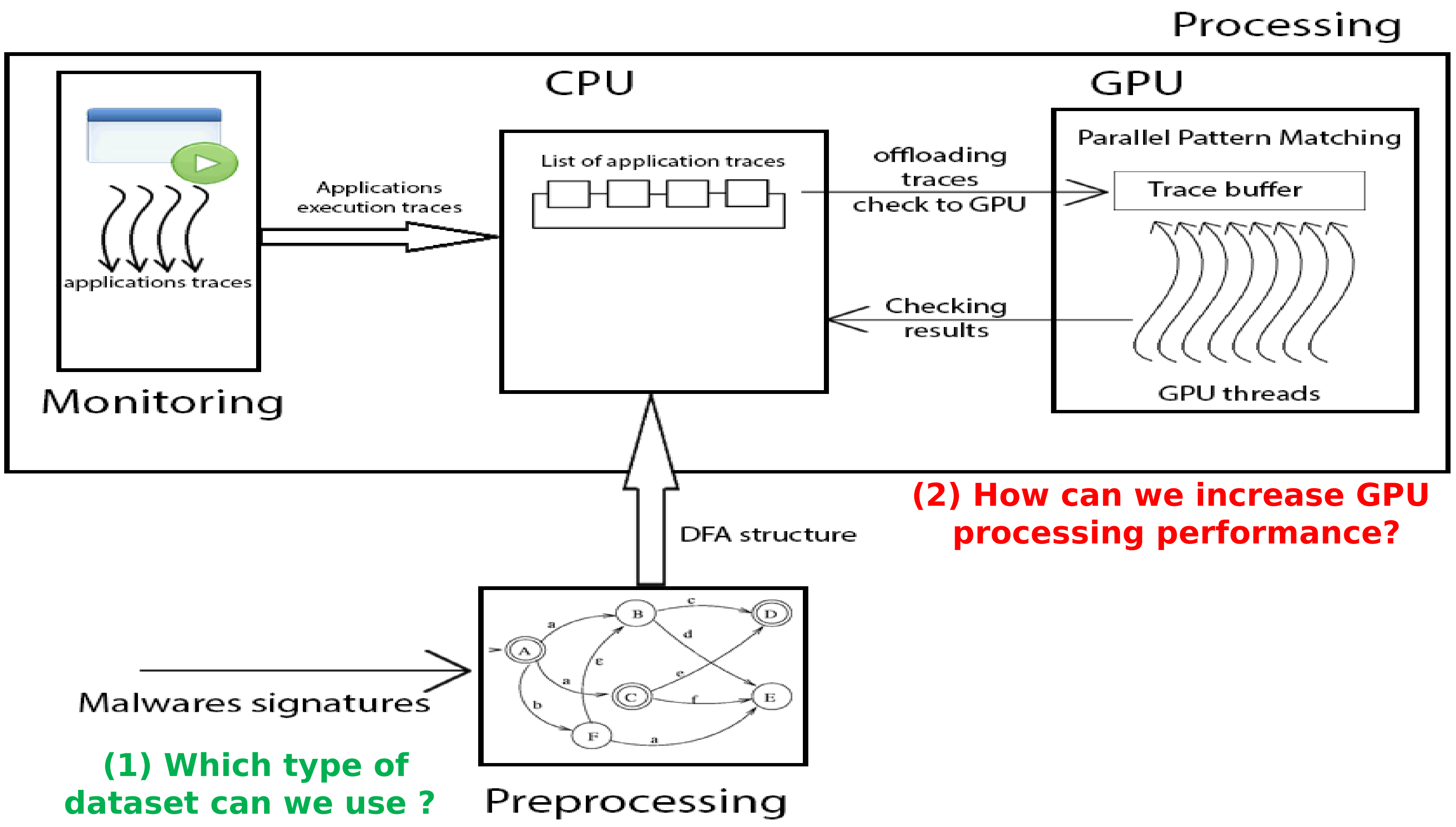
Training phase

❖Filtering Phase

- Get syscall patterns from benign applications B
- For every Csi in our malware dataset M
    - Counting the number of common subsequence Csi appearing in B and in M
    - Calculate malicious probability of Csi
    - Storing Csi into Malware common subsequences if Csi haw high malicious probability
- We choose to work with Csi having malicious probability = 1 (Csi appearing in M and not in B)

The result of traning phase= Malware behavioural dataset build of syscalls patterns

Architecture



(2) How can we increase GPU processing performance?

(1) Which type of dataset can we use ?

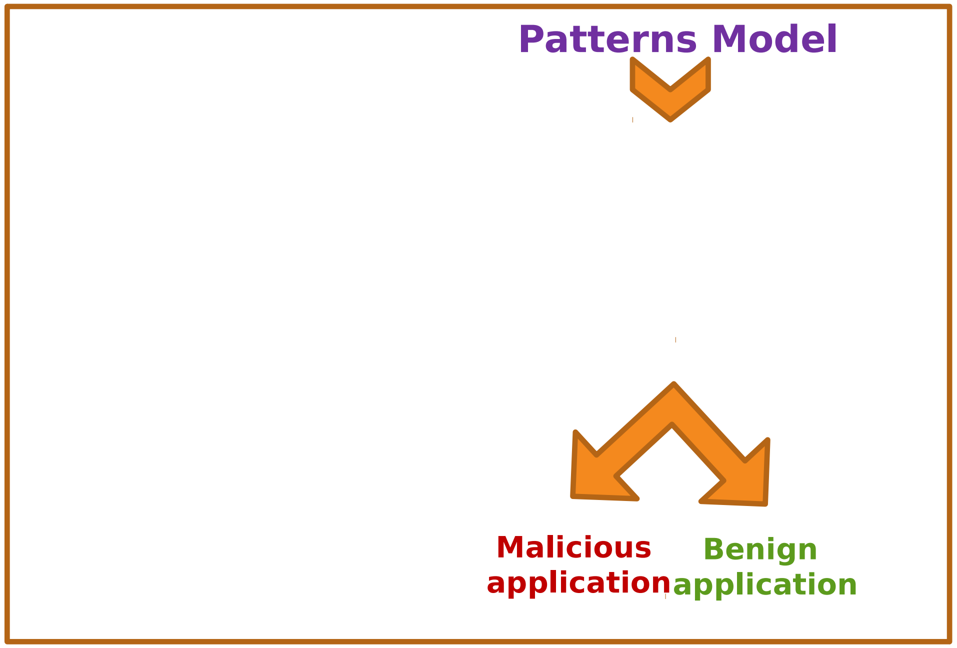
✓

Challenge 2

How can we increase the parallell processing performance of pattern matching on the GPU?

Parallel Pattern Matching Algorithm

- Match of data streams by malware scanner against a large set of known signatures, using a pattern matching algorithm.
- Pattern matching algorithms analyze the data stream and compare it against a database of signatures to detect known malware.
- Fairly complex signature patterns composed of different-size strings, range constraints, and sometimes recursive forms.



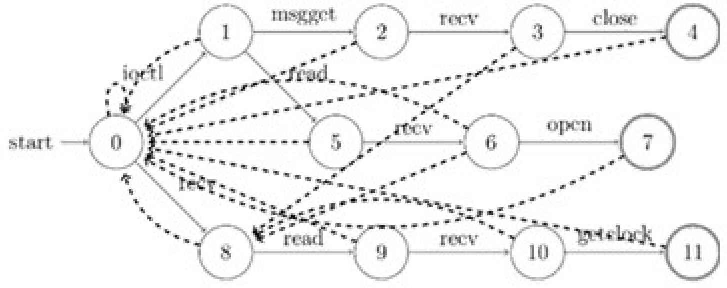
Example
- Aho-corasick
- Wu-manber
- Knuth-Morris-Pratt

Parallel Pattern Matching Algorithm

Aho-Corasick

- AC algorithm is based on a DFA structure built from reference patterns.
- The construction of automaton is done in pre-processing phase.
- The matching process is done in processing phase.
- The automaton structure can be essentially described by tow tables: transition table and failure state table.

ioctl	msgget	recv	close
ioctl	read	recv	open
recv	read	recv	getclock



state	Final state
0	0
1	0
2	0
3	0
4	1
5	0
6	0
7	1
8	0
9	0
10	0
11	1

	ioctl	msgget	recv	close	read	open	getclock
0	1	fail	8	fail	fail	fail	fail
1	fail	2	fail	fail	5	fail	fail
2	fail	fail	3	fail	fail	fail	fail
3	fail	fail	fail	4	fail	fail	fail
4	fail	fail	fail	fail	fail	fail	fail
5	fail	fail	6	fail	fail	fail	fail
6	fail	fail	fail	fail	fail	7	fail
7	fail	fail	fail	fail	fail	fail	fail
8	fail	fail	fail	fail	9	fail	fail
9	fail	fail	10	fail	fail	fail	fail
10	fail	fail	fail	fail	fail	fail	11

state	Failure state
0	0
1	0
2	0
3	8
4	0
5	0
6	8
7	0
8	0
9	0
10	8
11	0

Direct implementation of parallel pattern matching

- Idea
  - Input stream segmentation
  - For every segment we associate a thread
  - Problem of boundary detection

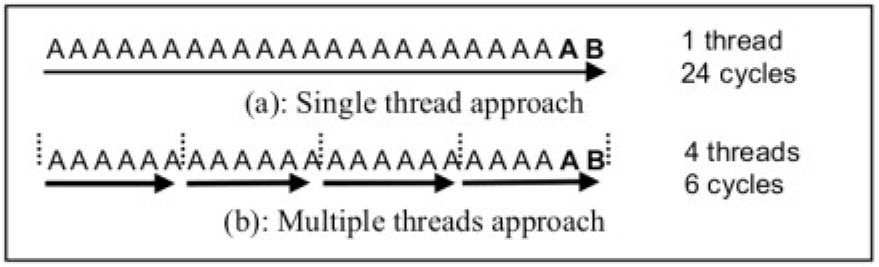


Figure 1. Single vs. multiple thread approach

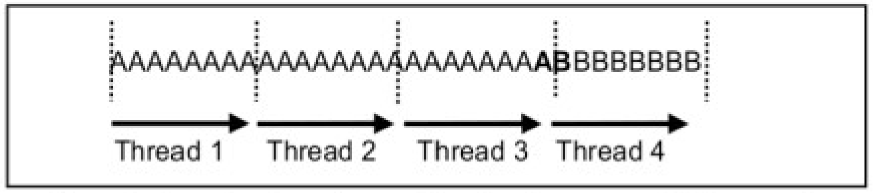


Figure 2. Boundary detection problem that the pattern "AB" cannot be identified by Thread 3 and 4.

- Possible solution
  - Every thread check the pattern presence on the edges.
  - Each thread must scan for a minimum length which is s almost equal to the segment length plus the longest pattern length of an AC state machine

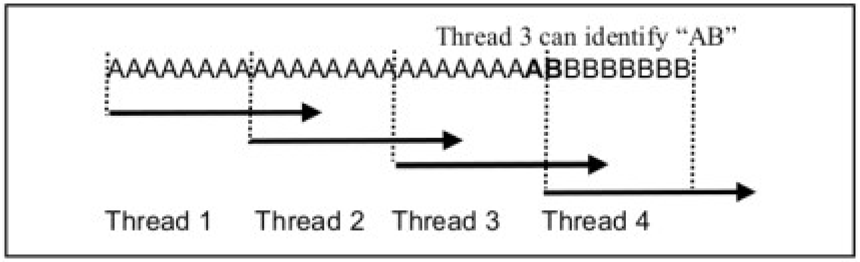


Figure 3. Every thread scans across the boundary to resolve the boundary detection problem.

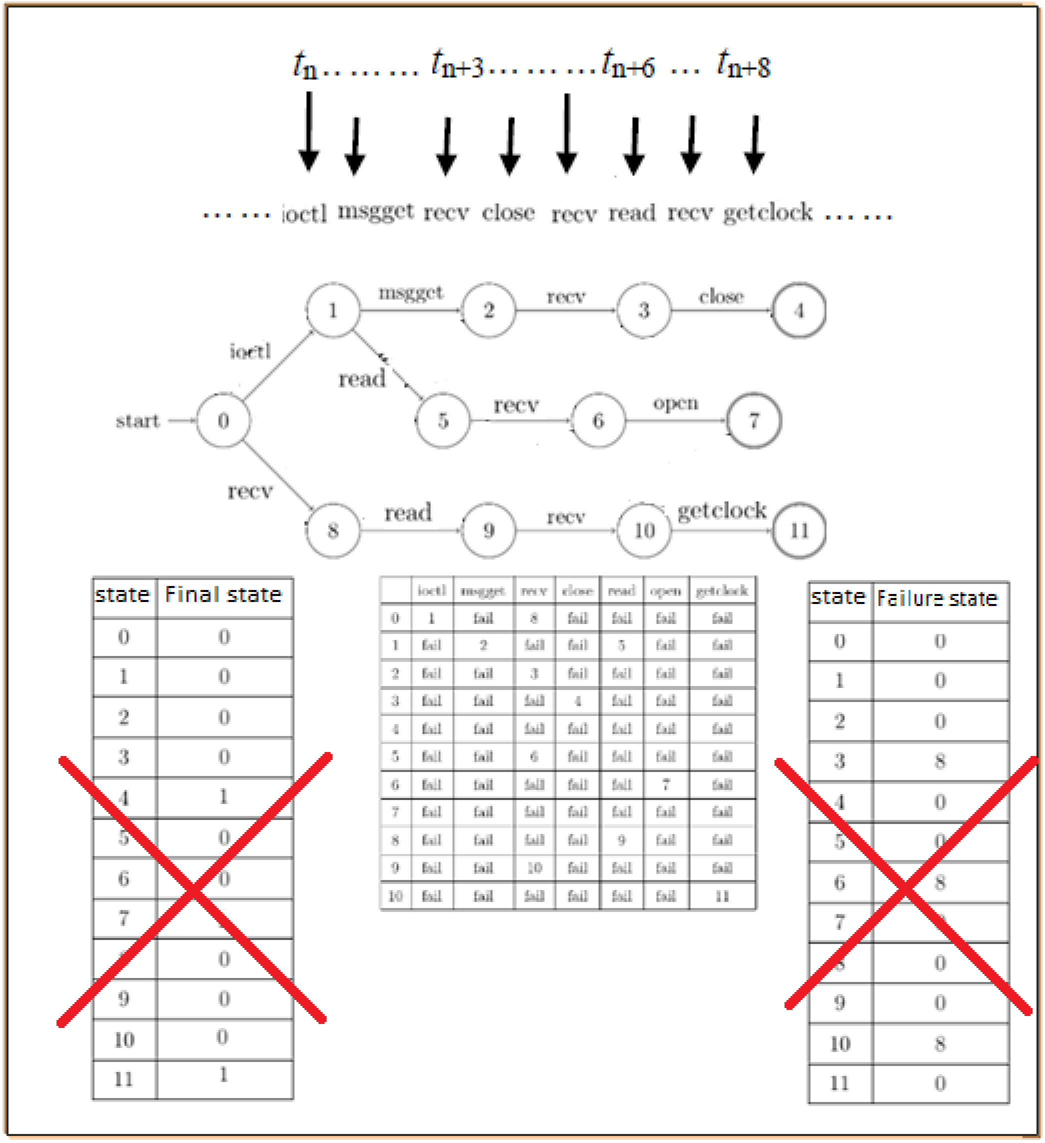
Parallel Pattern Matching Algorithm

Aho-Corasick

Lin, C. H., Liu, C. H., Chien, L. S., & Chang, S. C. (2013). Accelerating pattern matching using a novel parallel algorithm on gpus. Computers, IEEE Transactions on, 62(1 1906-1916.

- **Gaols**
  - Increase pattern matching computation throughput via parallelization.
  - resolve the throughput bottleneck caused by the overlapped computation.

- **Idea**
  - Byte allocation per thread
  - Failure transitions elimination
  - The thread stops his work if no valid transition is found.



t_n …… t_{n+3} …… t_{n+6} … t_{n+8}
…… ioctl msgget recv close recv read recv getclock ……
start
0
1
2
3
4
5
6
7
8
9
10
11
ioctl
msgget
recv
close
read
recv
open
recv
read
recv
getclock

state	Final state
0	0
1	0
2	0
3	0
4	1
5	0
6	0
7	[illegible]
[illegible]	0
9	0
10	0
11	1

	ioctl	msgget	recv	close	read	open	getclock
0	1	fail	8	fail	fail	fail	fail
1	fail	2	fail	fail	5	fail	fail
2	fail	fail	3	fail	fail	fail	fail
3	fail	fail	fail	4	fail	fail	fail
4	fail	fail	fail	fail	fail	fail	fail
5	fail	fail	6	fail	fail	fail	fail
6	fail	fail	fail	fail	fail	7	fail
7	fail	fail	fail	fail	fail	fail	fail
8	fail	fail	fail	fail	9	fail	fail
9	fail	fail	10	fail	fail	fail	fail
10	fail	fail	fail	fail	fail	fail	11

state	Failure state
0	0
1	0
2	0
3	8
4	0
5	0
6	8
7	[illegible]
8	0
9	0
10	8
11	0

Parallel Pattern Matching Algorithm

Parallel Failureless Aho-Corasick

Parallel Pattern Matching Algorithm

Parallel Failureless Aho-Corasick

- **Increase of the algorithm performance on GPU**

Challenge 3

- Malwares grows continuously

- The number of signatures is increasing proportionally

➢ Scaling problems for mobile anti-malwares due to:

- Limitation of Mobile GPU memory VS Important memory requirement for DFA structure

The need of applying memory compacting techniques

Memory optimization technique

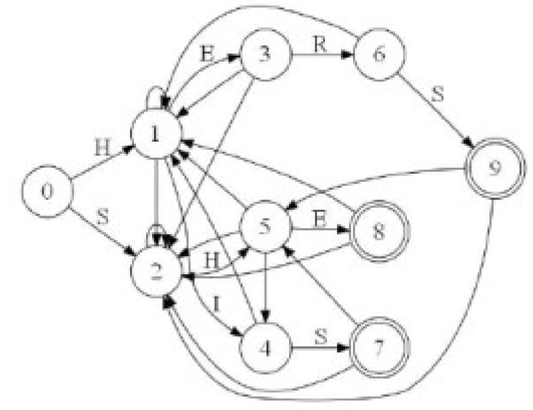
1

2

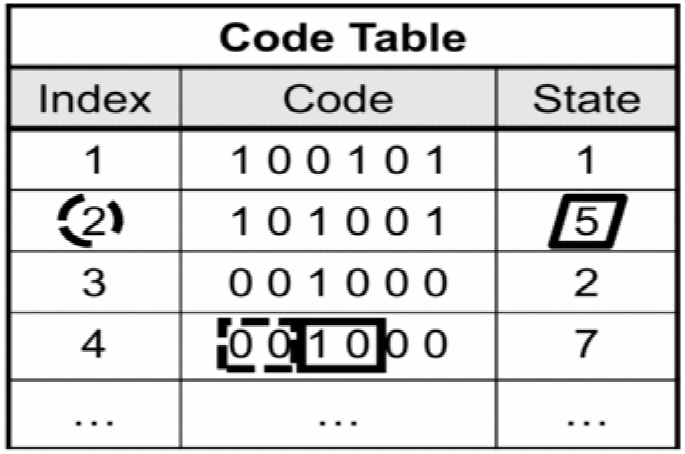
3

Memory optimization technique

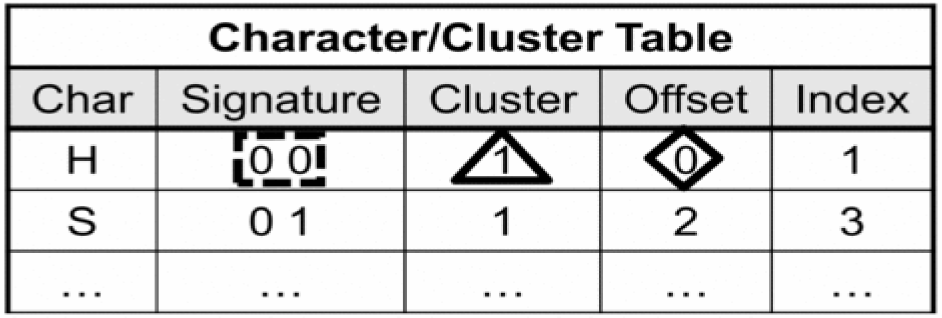
P3FSM: Portable Predictive Pattern Matching Finite State Machine



(a) DFA for Patterns “SHE”, “HERS”, “HIS”



Code Table		
Index	Code	State
1	1 0 0 1 0 1	1
2	1 0 1 0 0 1	5
3	0 0 1 0 0 0	2
4	0 0 1 0 0 0	7
…	…	…



Character/Cluster Table				
Char	Signature	Cluster	Offset	Index
H	0 0	1	0	1
S	0 1	1	2	3
…	…	…	…	…

1. Check cluster 1 of state code 7 for character signature of H 00 = 00
2. Compute next state: state signature + character offset
   $\boxed{10}$ + 0 = 2 -> next state index = 2, next state = state 5

Experimentations

❖ **Hardware**

❑ Mobile Phone
- HTC one

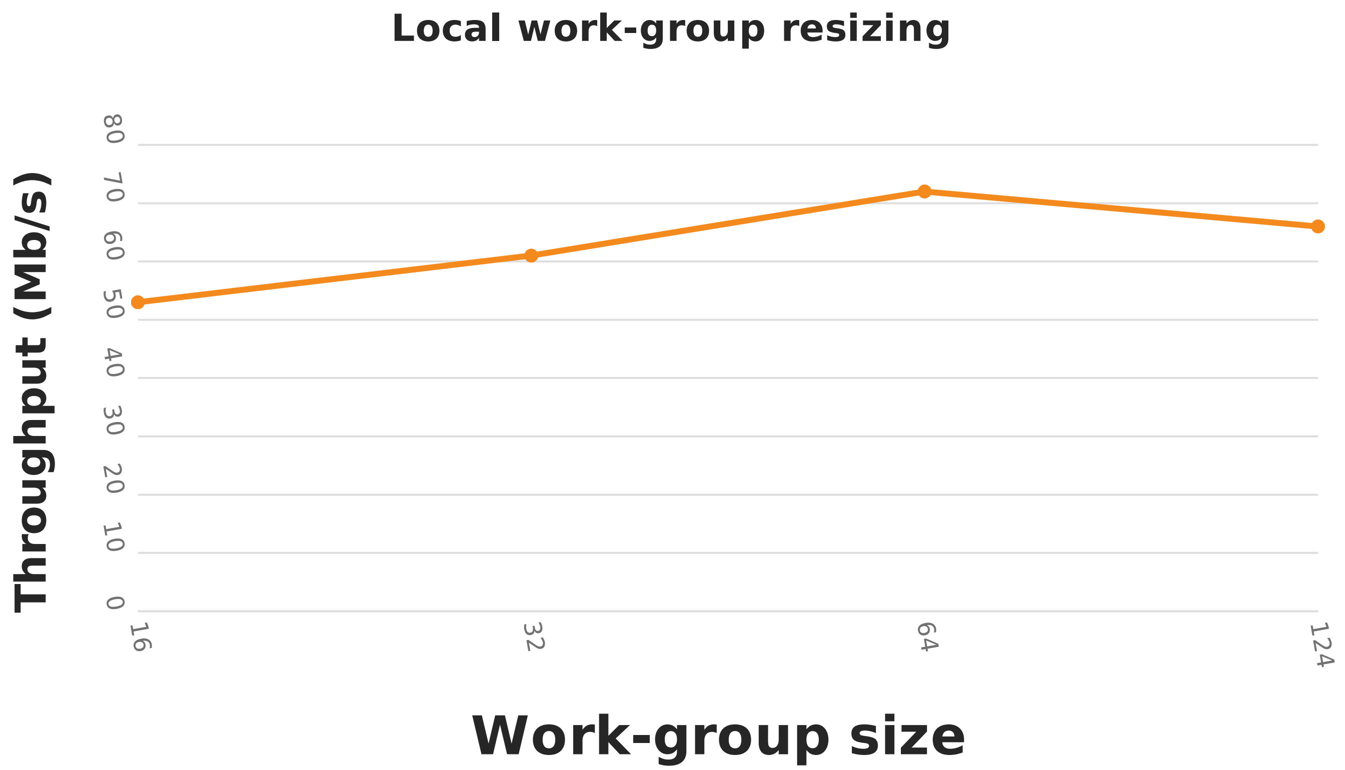
❑ GPU
- Adreno 320

❑ CPU
- Qualcomm Snapdragon 600, quad-core CPU @ 1.7GHz

❖ **Benchmark**
- 600 Malicious syscalls patterns

Experimentations

Thread per block resizing

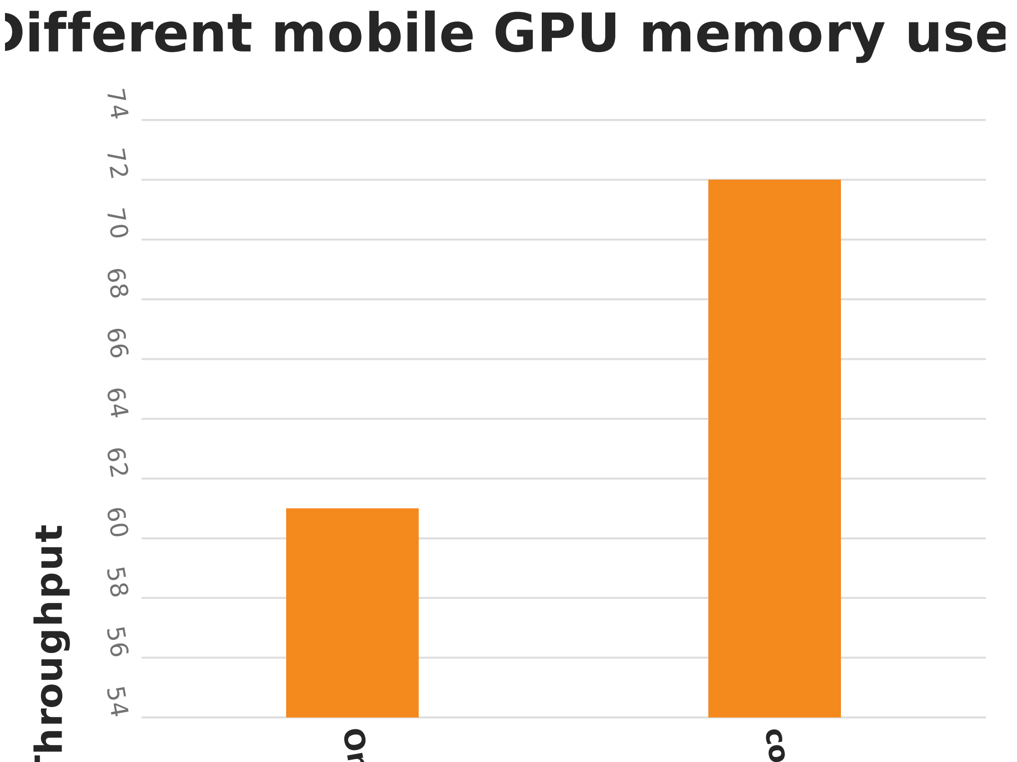


✓ Best throughput with 64x64 threads = 72Mb/s

Experimentations

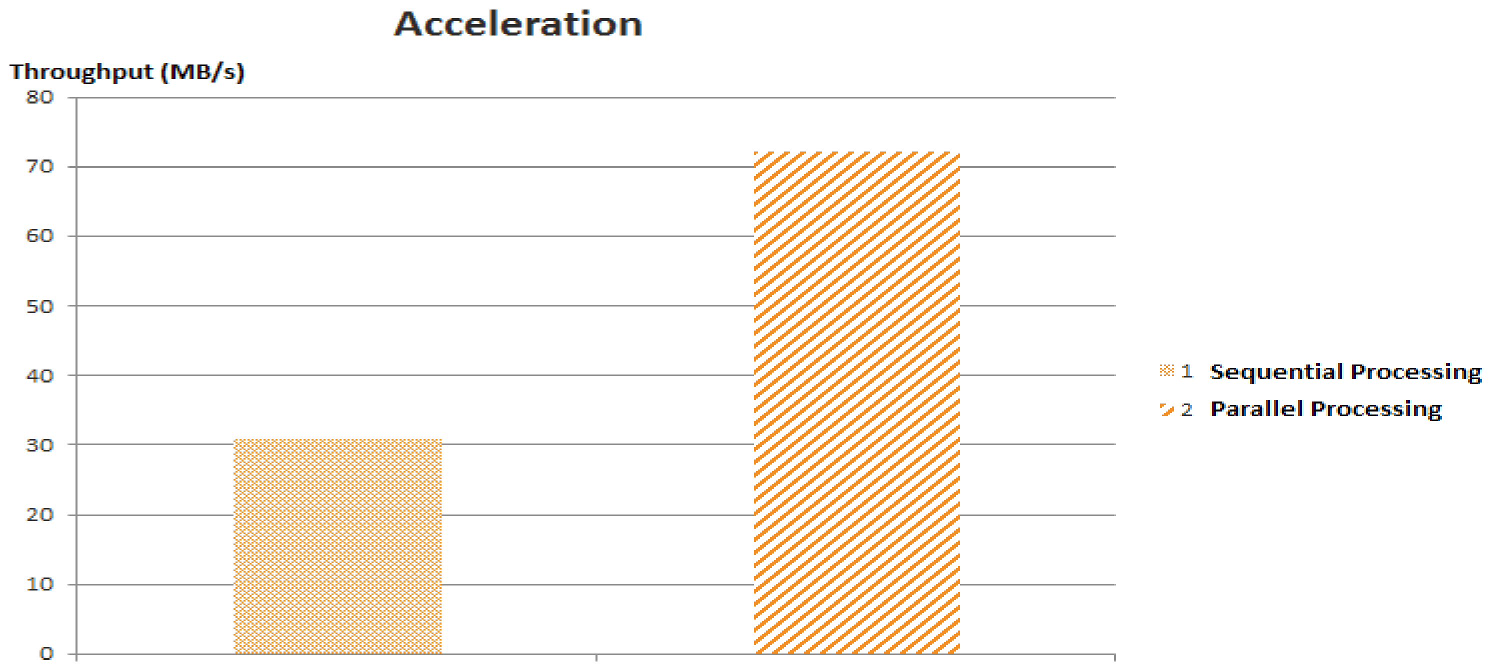
Effective use of the different GPU memory types

- ✓ Around 15% gain of performance with the use of constant and shared memories
- ✓ Applying shared memory to improve the latency of global memory accesses



Experimentations

Acceleration



- ✓ An acceleration of around 2.3x is obtained with the parallel processing on the mobile GPU over serial processing
- ➢ The framework throughput is dominated by data transfers between the host /device which consist of 60% of the total processing time

Experimentations

Memory requirement

Number of patterns	PFAC (KB)	P3FSM (KB)
100	446	37
200	703	83
300	1086	197
600	2843	462

- ✓ Storing DFA structure on the GPU is memory consuming especially that mobile GPU memory is small.
- ✓ Difference in memory requirement between PFAC DFA and P3FSM.
- ✓ P3FSM that compacts the DFA structure by many times comparing to standard PFAC DFA.

Conclusion

- Implementation of a parallel host-based anti-malware on mobile GPU using behavioral detection techniques

- Series of optimizations to deal with the low memory problem of mobile devices and the ever-increasing computing and memory requirements of malware detection

- **Perspectives:**
  - Integrating a GPU monitor which tracks down the GPU memory usage and allows the automaton adjustment in real-time to fit the reduced GPU memory
  - Use of mobile GPU clusters
  - Working on malware dataset to achieve better detection accuracy

Thank you for your attention