introducing

Ghassen Zegden
Manel Abdellatif

Highly Parallel Architectures to accelerate Malware detection

Introduction : Parallelism & security

Motivations

- 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
- Malware detection is a highly common and computationally-intensive problem
- Improvement in parallel computation performance

Security threats

VS

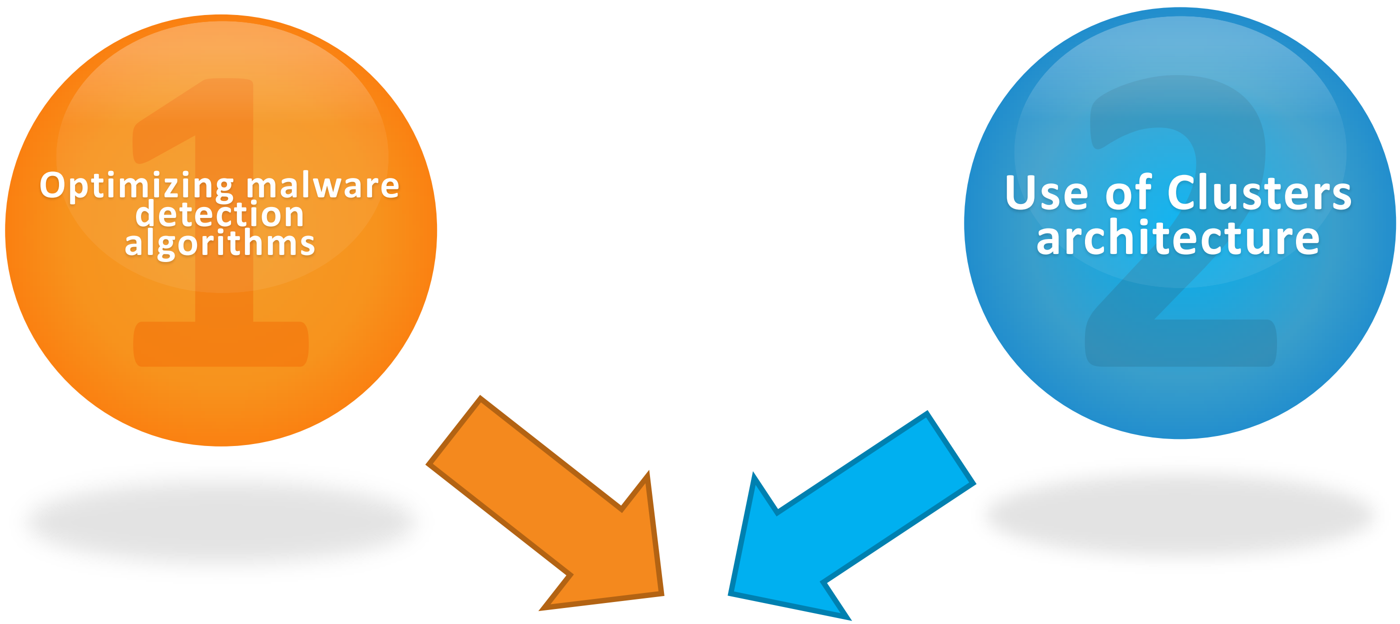
Parallel performance computation

- How to get benefit from parallel architectures on small-scale systems to accelerate malware detection?

Introduction : Parallelism & security

Focus

- In general, we make benefit from parallelism to reinforce the security level of the system by :



Achieve better computing performance
&
detection accuracy

Introduction : Parallelism & security

Work Part 1

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

Work Part 2

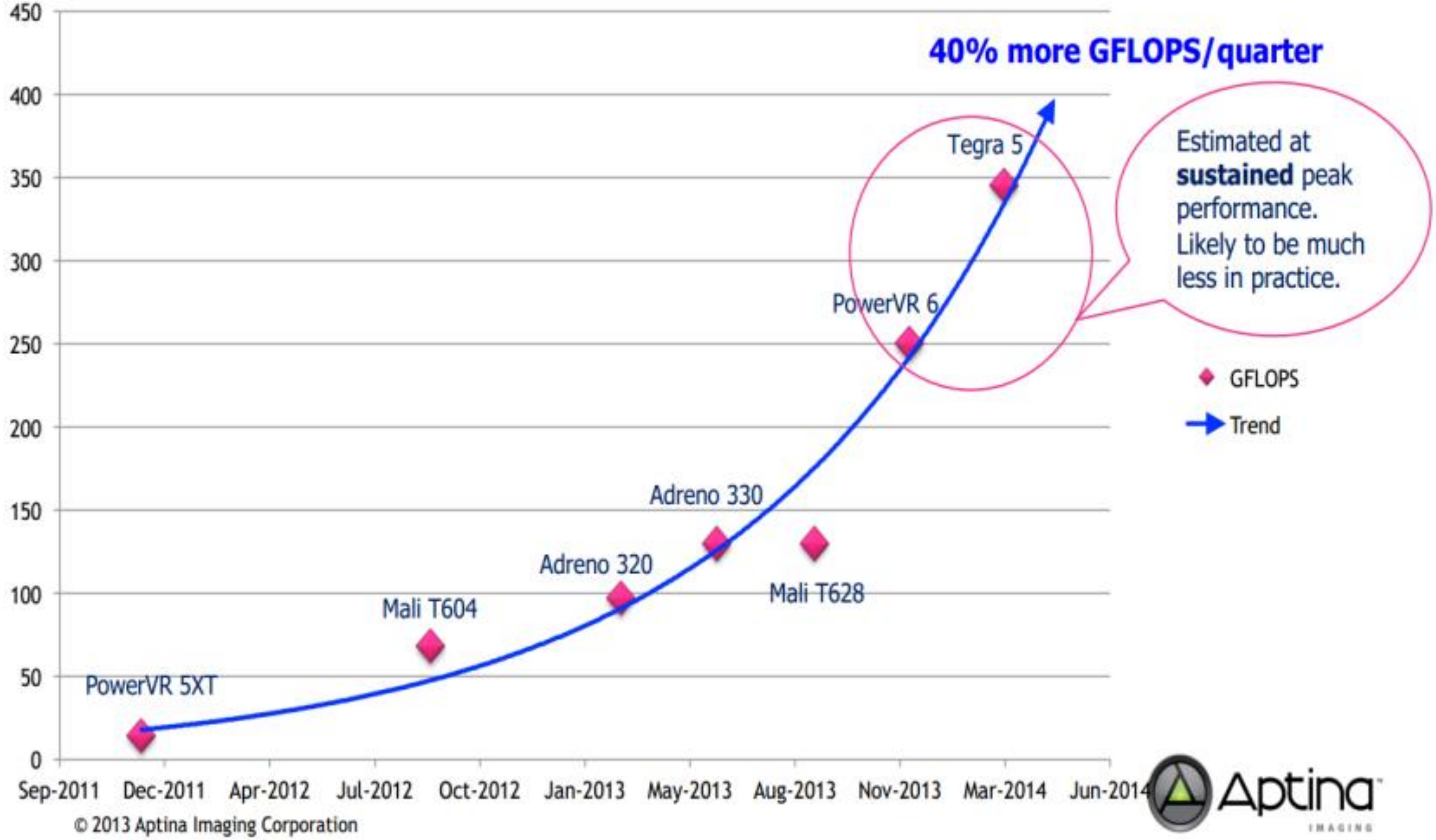
- Migration to mobile GPU platform
- Use of OpenCL
- Building of behavioral malware dataset based on syscalls patterns
- Development of memory optimization techniques
- Experimenting different scenarios to scan trace files

Work Part 3

- Migration to Parallella board in order to experiment clusters architecture
- Use of epiphany coprocessor
- Use of CO-PRocessing THReads (COPRTHR) SDK

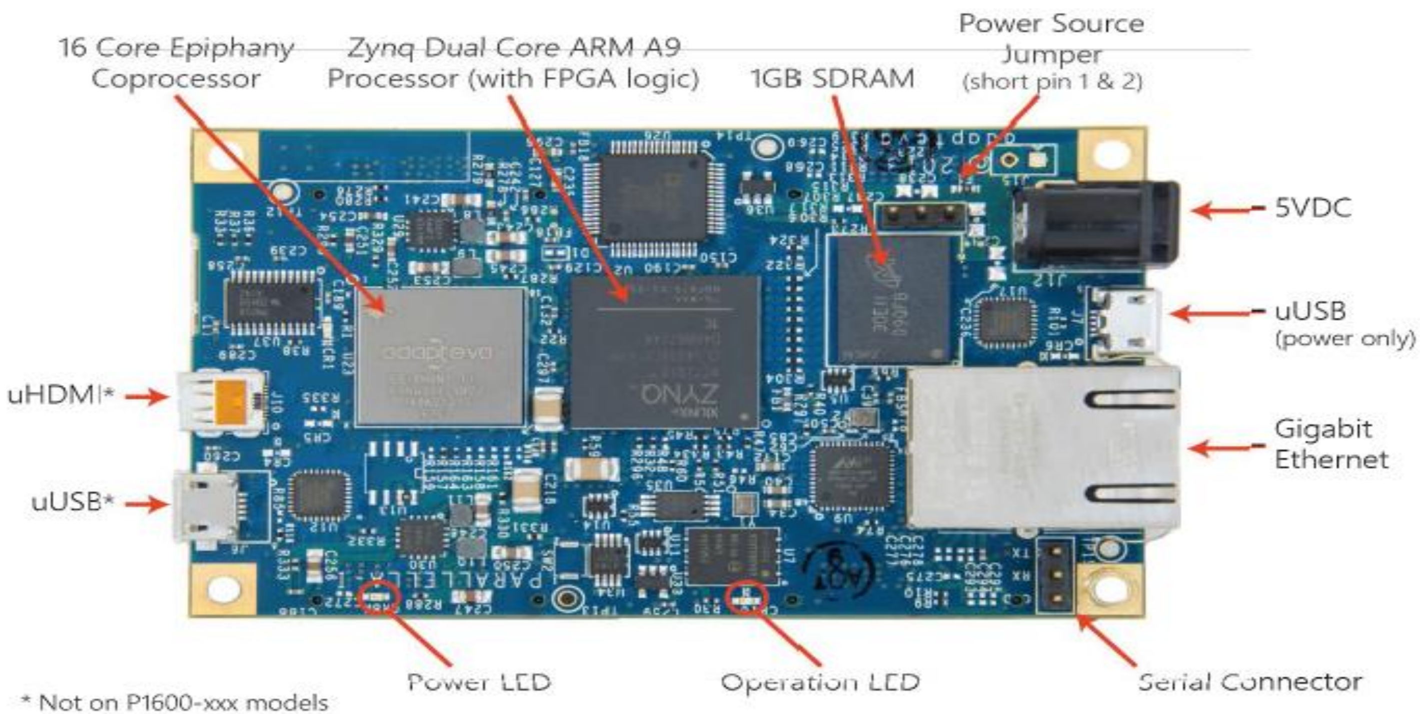
Parallel Processing architecture

Evolution of GPUs for embedded svstems



Parallella Board

Parallella: Enviromnent for parallel processing



- 100$ credit-card sized computer based on the Epiphany multicore chips developed by Adapteva
- Energy efficient
- High performance processing

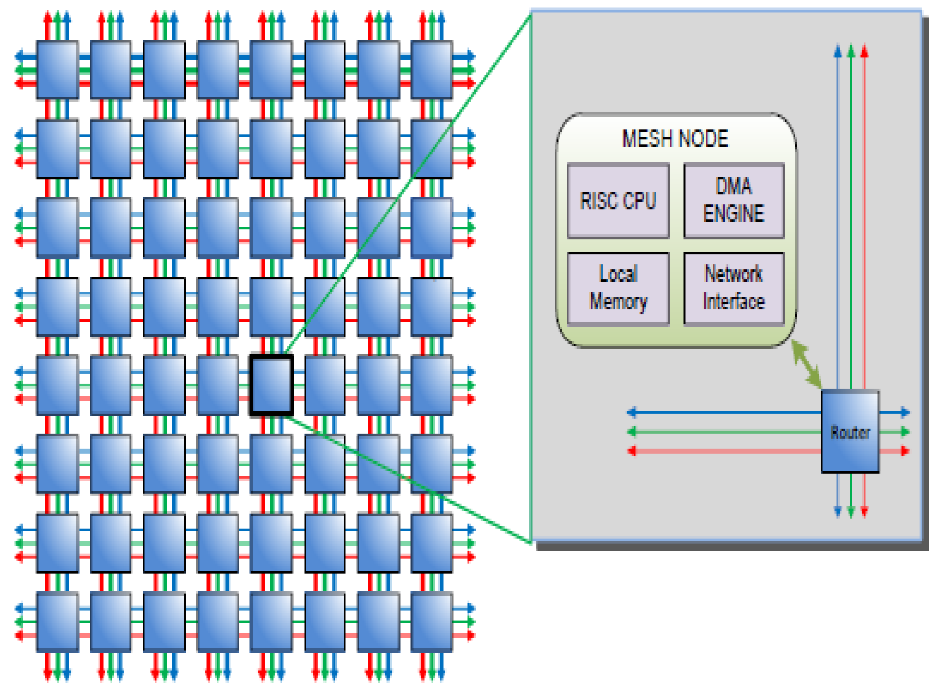
Parallella Board

Epiphany architecture

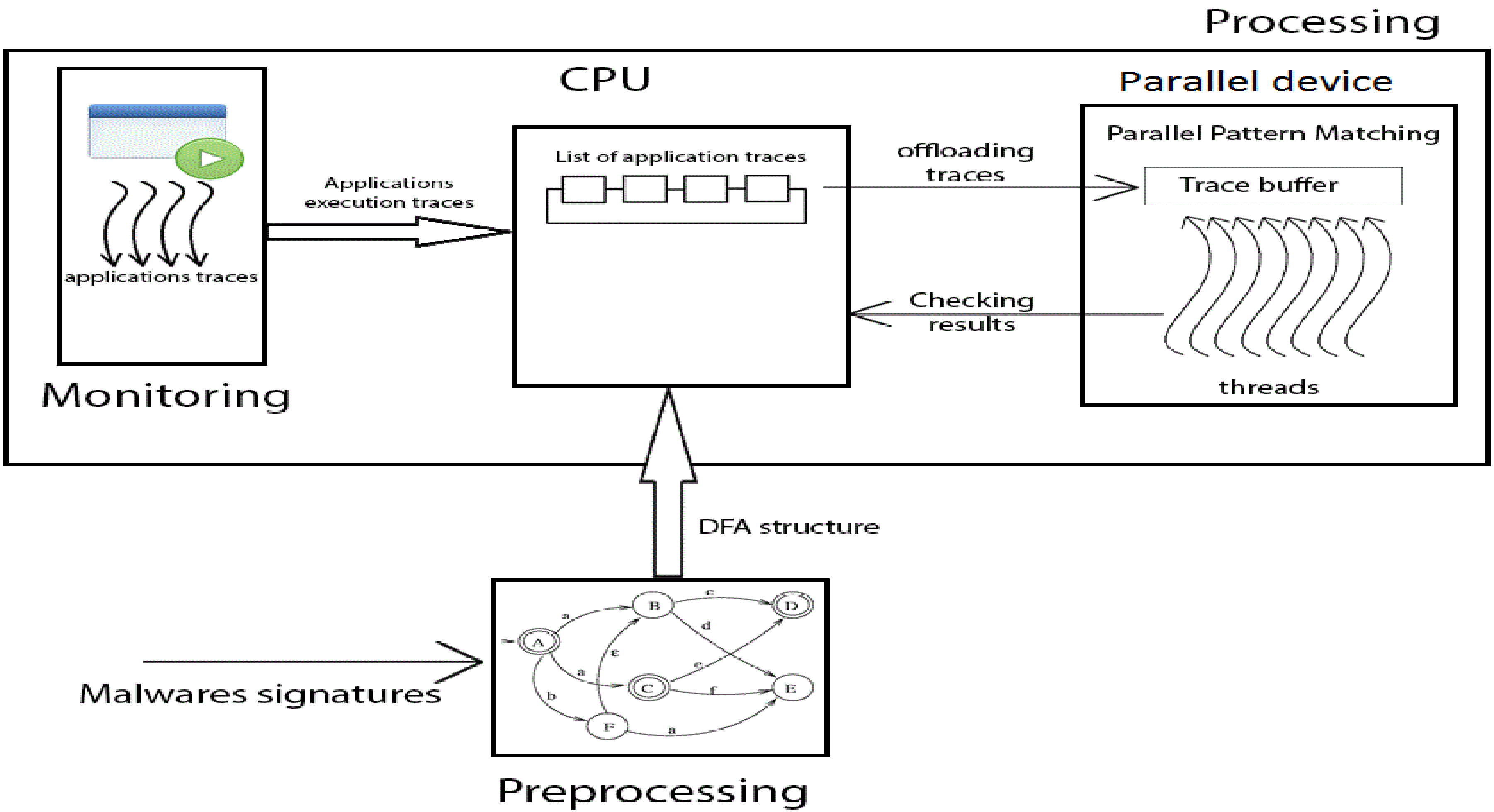
The key benefits of the Epiphany architecture are:

- **Ease of Use:** A multicore architecture that is ANSI-C/C++ programmable : accessible to every programmer
- **Effectiveness**
- **Scalability:** The architecture can scale to thousands of cores on a single chip and millions of cores within a larger system

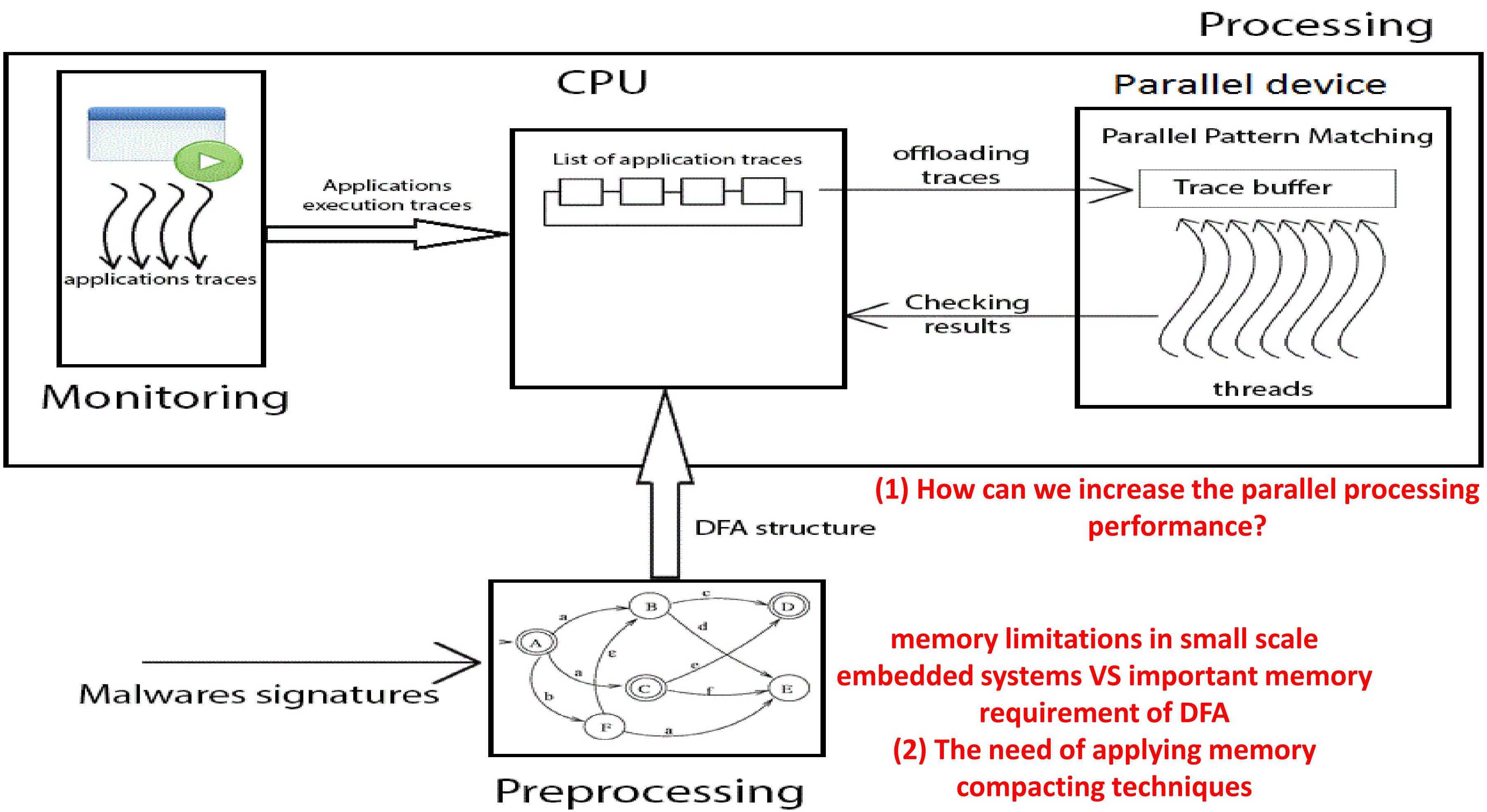
But very small local memory per Ecore (Only 32 KB for data + code)



Framework Architecture for mobile GPU



Architecture Challenges



(1) How can we increase the parallel processing performance?

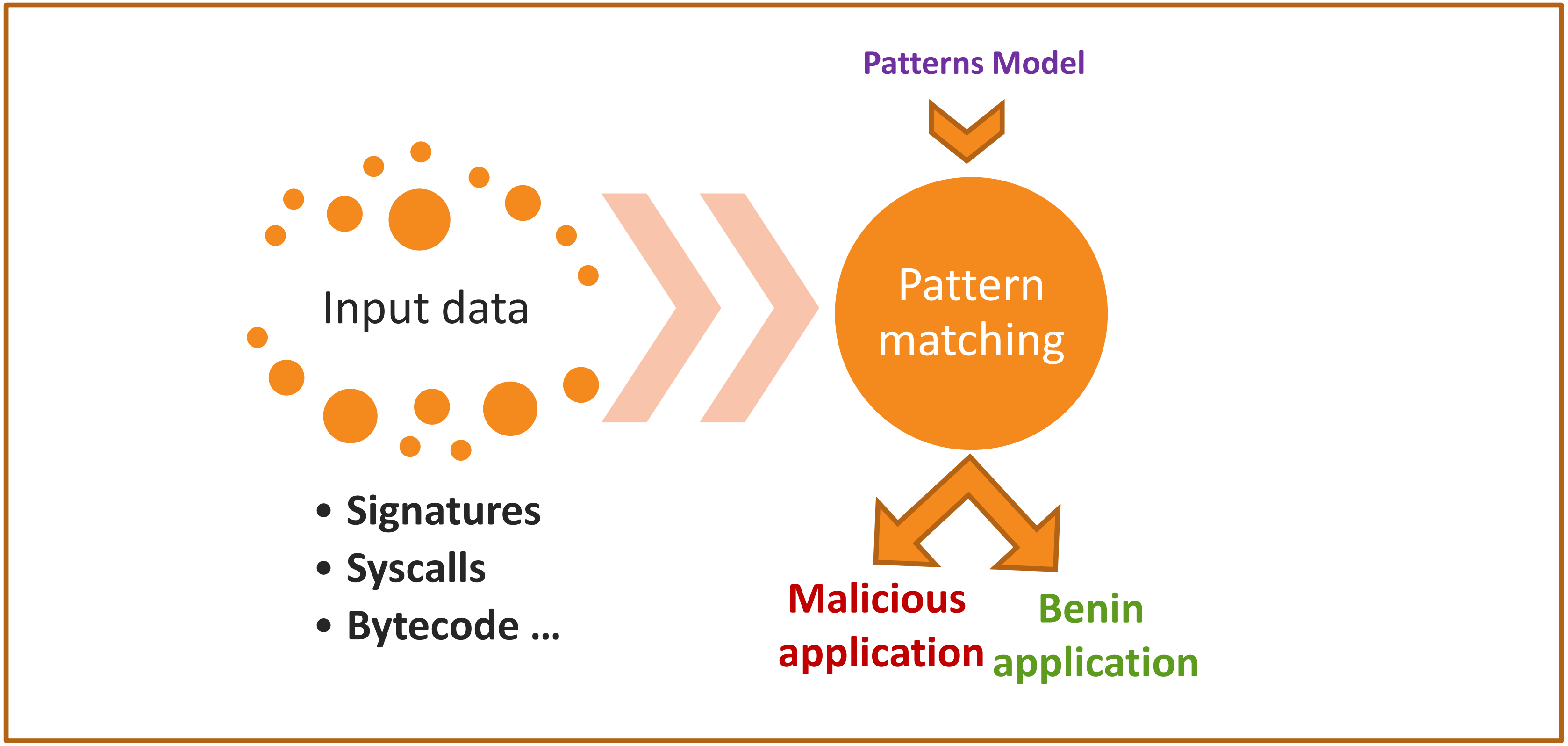
memory limitations in small scale embedded systems VS important memory requirement of DFA

(2) The need of applying memory compacting techniques

Challenge 1

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

Parallel Pattern Matching Algorithm



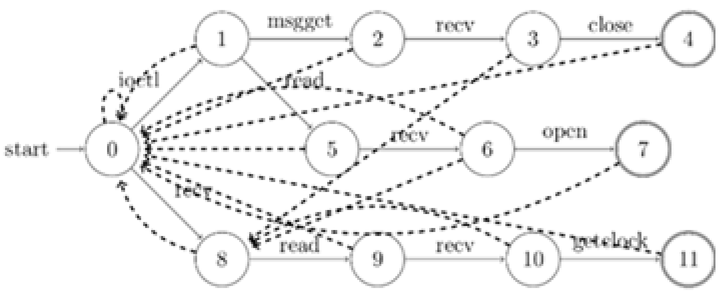
Example

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

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

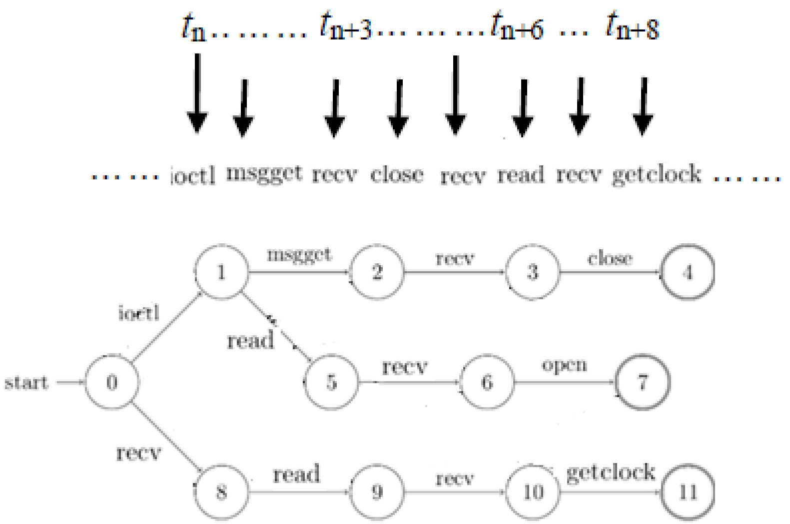
	ioctl	msgget	recv	close	read	open	getclock
0	1	fail	8	fail	fail	fail	fail
1	fail	2	fail	fail	5	fail	fail
2	fail	fail	3	fail	fail	fail	fail
3	fail	fail	fail	4	fail	fail	fail
4	fail	fail	fail	fail	fail	fail	fail
5	fail	fail	6	fail	fail	fail	fail
6	fail	fail	fail	fail	fail	7	fail
7	fail	fail	fail	fail	fail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

Parallel Pattern Matching Algorithm

Parallel Failureless Aho-Corasick

- **Gaols**
  - Increase pattern matching computation throughput via parallelization



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]
[illegible]	0
9	0
10	8
11	0

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

Parallel Pattern Matching Algorithm

Parallel Failureless Aho-Corasick

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

  - Reducing the global memory transactions of the system
  - Making benefit from memory architecture of GPU by using constant memory and local memory
  - Minimize transfers: Intermediate data can be allocated, operated on, and deallocated without ever copying them to host memory
  - Adopting an adequate scan scenario of the input stream

Challenge 2

- Malwares grows continuously
- The number of signatures is increasing proportionally

➢ Scaling problems for mobile anti-malwares due to:

- Memory Limitation of small scale embedded systems VS Important memory requirement for DFA structure

The need of applying memory compacting techniques

Memory optimization technique

1. Eliminating failure transition
2. Eliminating Final states table by performing state reordering
3. Applying P3FSM technique

Experimentations

❖ **Hardware**

❑ Mobile Phone

- Sony Xperia Z

❑ GPU

- Adreno 320

❑ CPU

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

Experimentations

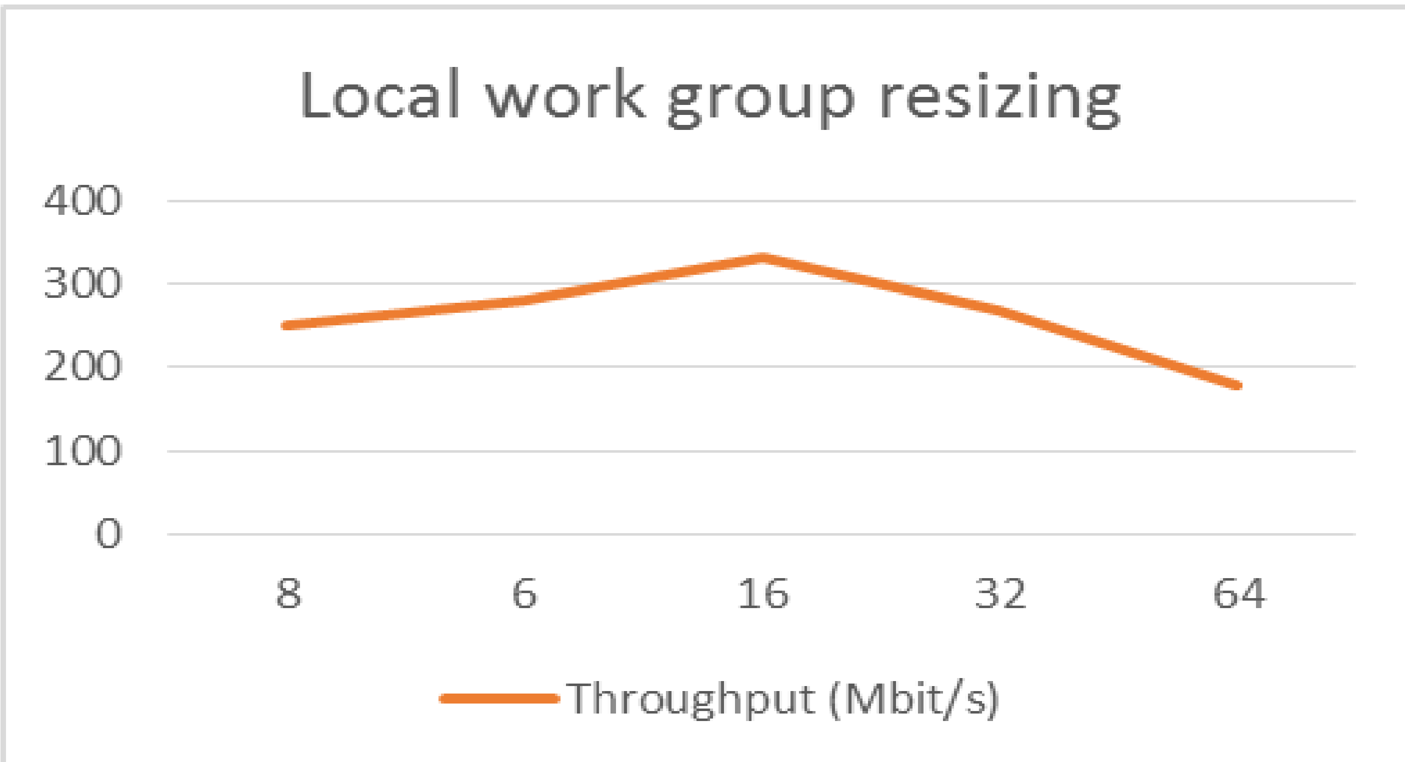
Memory requirement

Number of patterns	PFAC (KB)	P3FSM (KB)
2000	67677	8922
2200	74398	9234
10000	678937	50765
16000	806554	60432
17600	809321	74380

- ✓ 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 10 times comparing to standard PFAC DFA.

Experimentations

Thread per block resizing

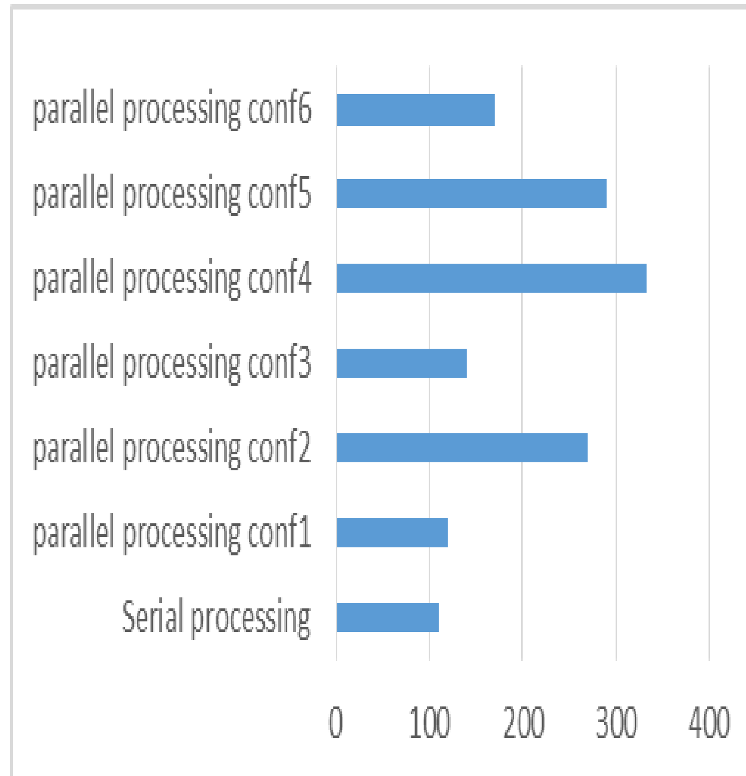


✓ Best throughput with 16 threads/block = 333Mb/s

Experimentations

Effective use of the different GPU memory types

Memory configuration	Global memory	Constant memory	Local memory
Conf1	transition_table input_buffer result_buffer		
Conf2	transition_table result_buffer		input_buffer
Conf3	transition_table result_buffer	input_buffer	
Conf4	Transition table Part 2 result_buffer	transition_tableP1	input_buffer
Conf5	transition_tableP2 input_buffer result_buffer	transition_tableP1	
Conf6	Transition table Part 2 result_buffer	input_buffer	Transition Table Part1

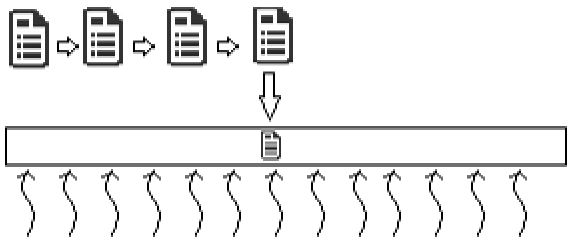


Serial processing throughput vs parallel

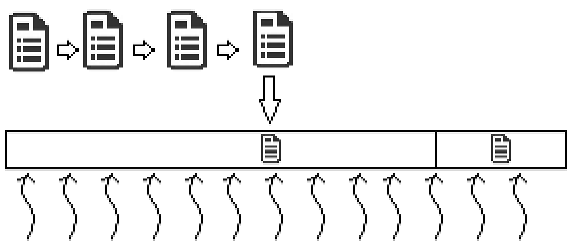
Experimentations

Traces' files Scanning Scenarios

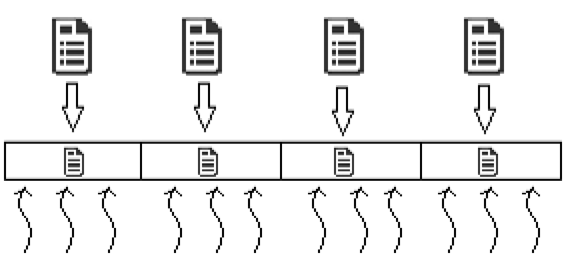
Scenario 1: Input buffer allocated to only one application trace file

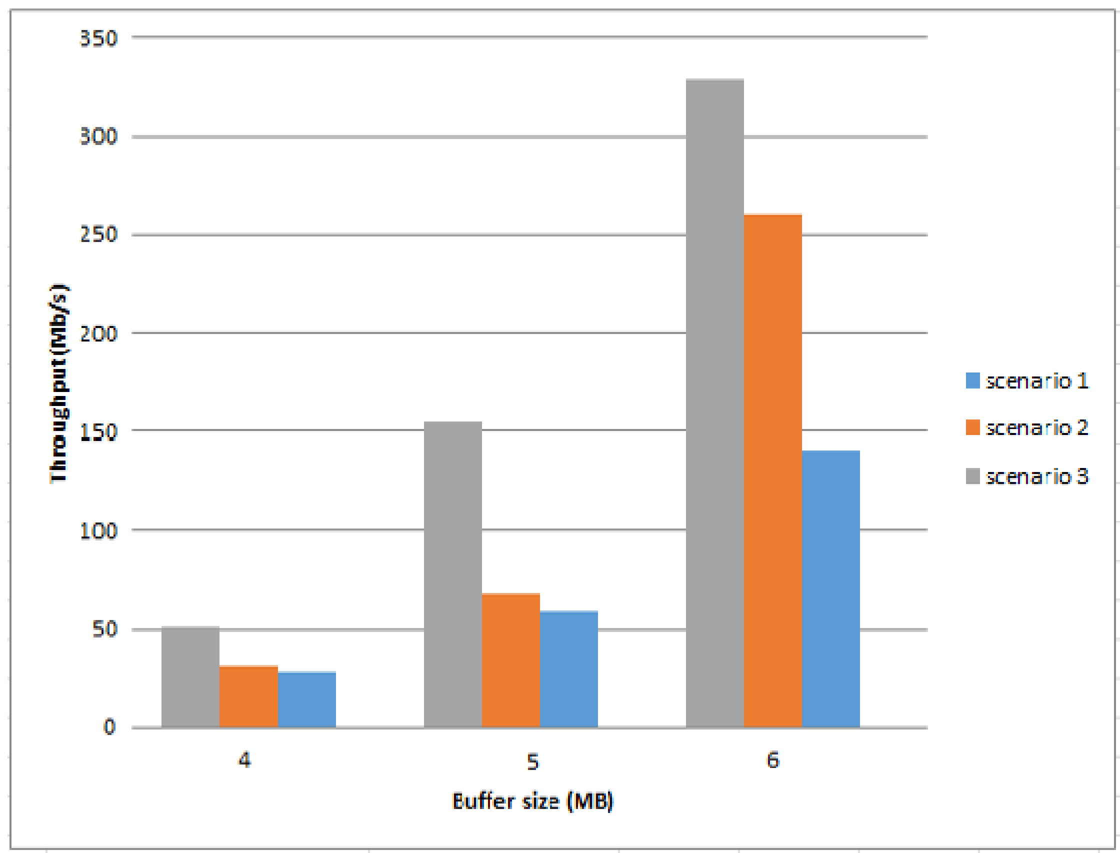


Scenario 2: Applications trace files are cocatenated in the input buffer



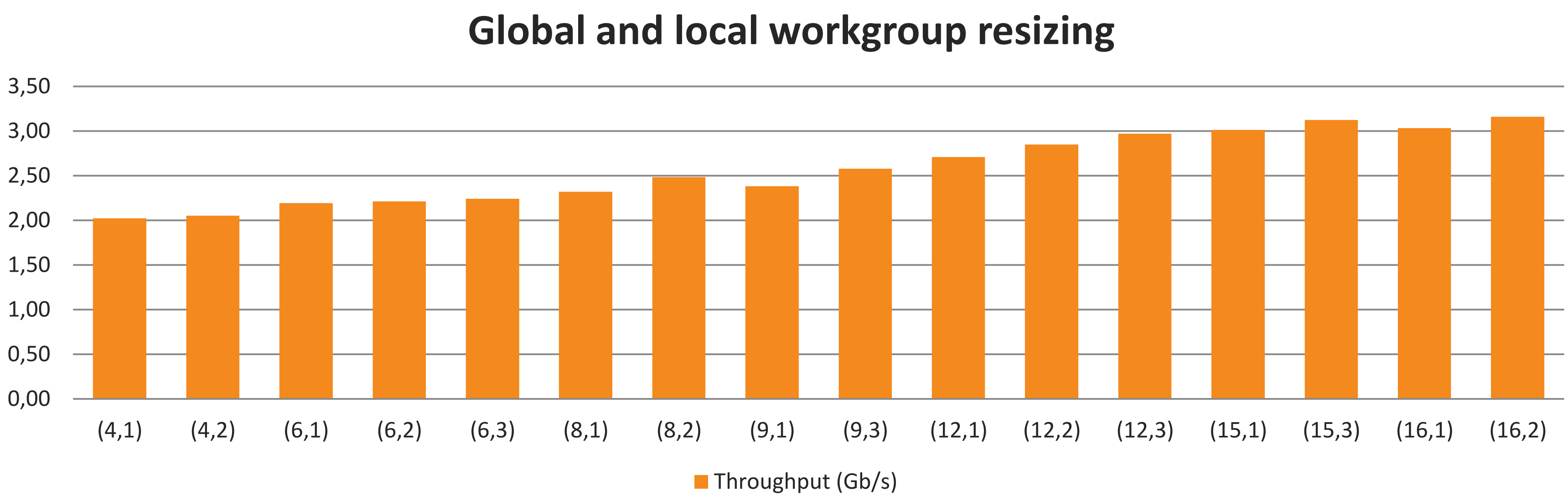
Scenario 3: Parallel processing of the applications traces simultaniously





Framework on Parallella Board

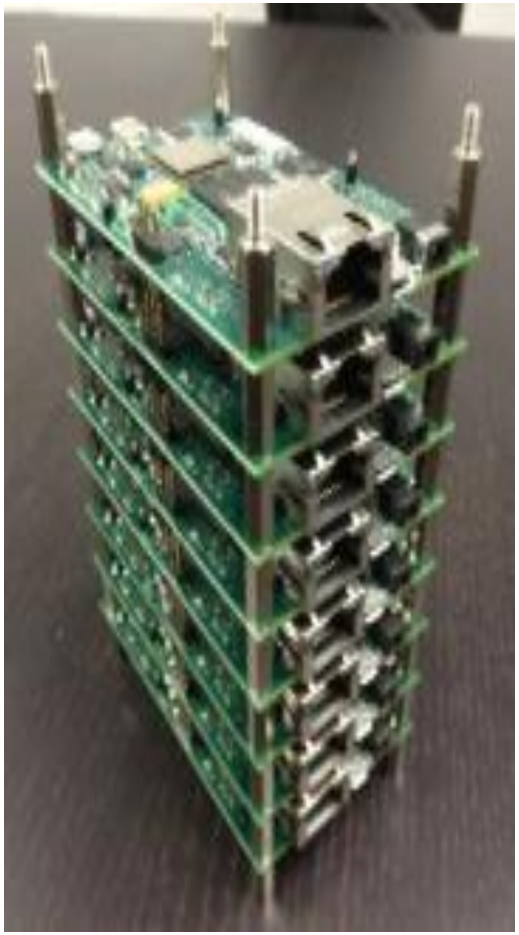
Load distribution and throughput of PFAC algorithm



- ✓ Max global workgroup size = 16 threads
- ✓ Max local workgroup size = 3 threads / ecore
- ✓ In general, the more parallel threads we have the better the throughput is.
- ✓ The best throughput is = 3.1 Gb/s with 8 ecores on which we execute 2 threads
- ✓ The more the ecores are fully exploited the better the throughput is.
- ✓ 50% of computation overhead due to data loading and platform initializing delays.
- ✓ Acceleration = 5x

Clusters

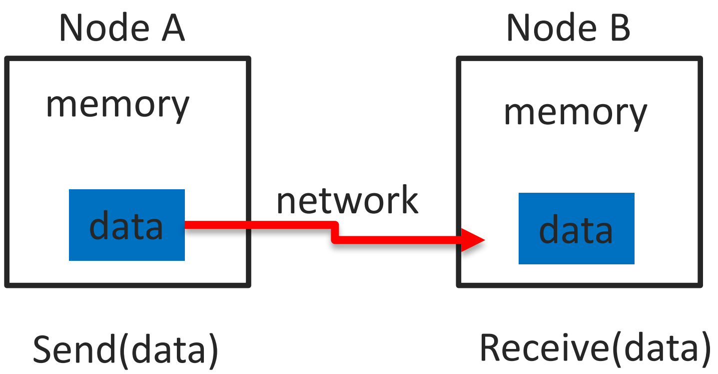
- **Beowulf Cluster**
  - consumer grade computers (not expensive)
  - MPI as a communication protocol
  - One Master, several slaves
  - Decrease the amount of time required for processor-intensive tasks

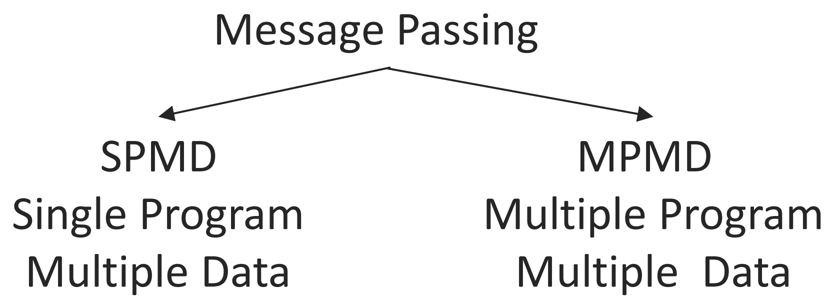


Cluster of parallella boards

Messages

- **Message Passing System**
  - In order to send/receive messages, some information has to be provided
    - Sending process
    - Source location
    - Data type
    - Data size
    - Receiving process





Messages

MPI – Message Passing Interface

- language-independent communications protocol used to program parallel computers –-> can be associated with Fortran,C,C++ and Java
- point-to-point and collective communication
- A fixed set of processes is created at program initialization
- Each process is identified by its rank
- Derived Data types can be defined to send different data types
- Open MPI as an implementation

Experimentations

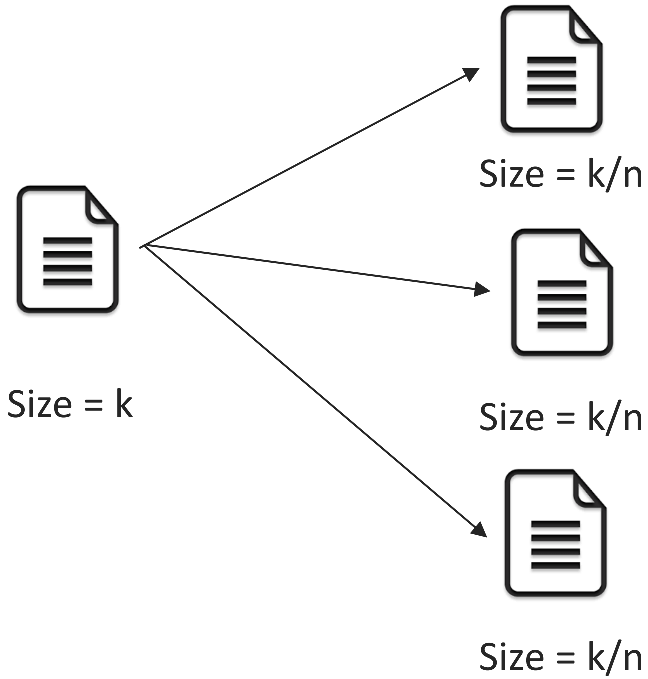
- **Hardware**
  - 4 parallella boards (total of 64 ecores)
  - Router

- **Software**
  - Ubuntu 14.04
  - Open Mpi v1.8.4

Experimentations

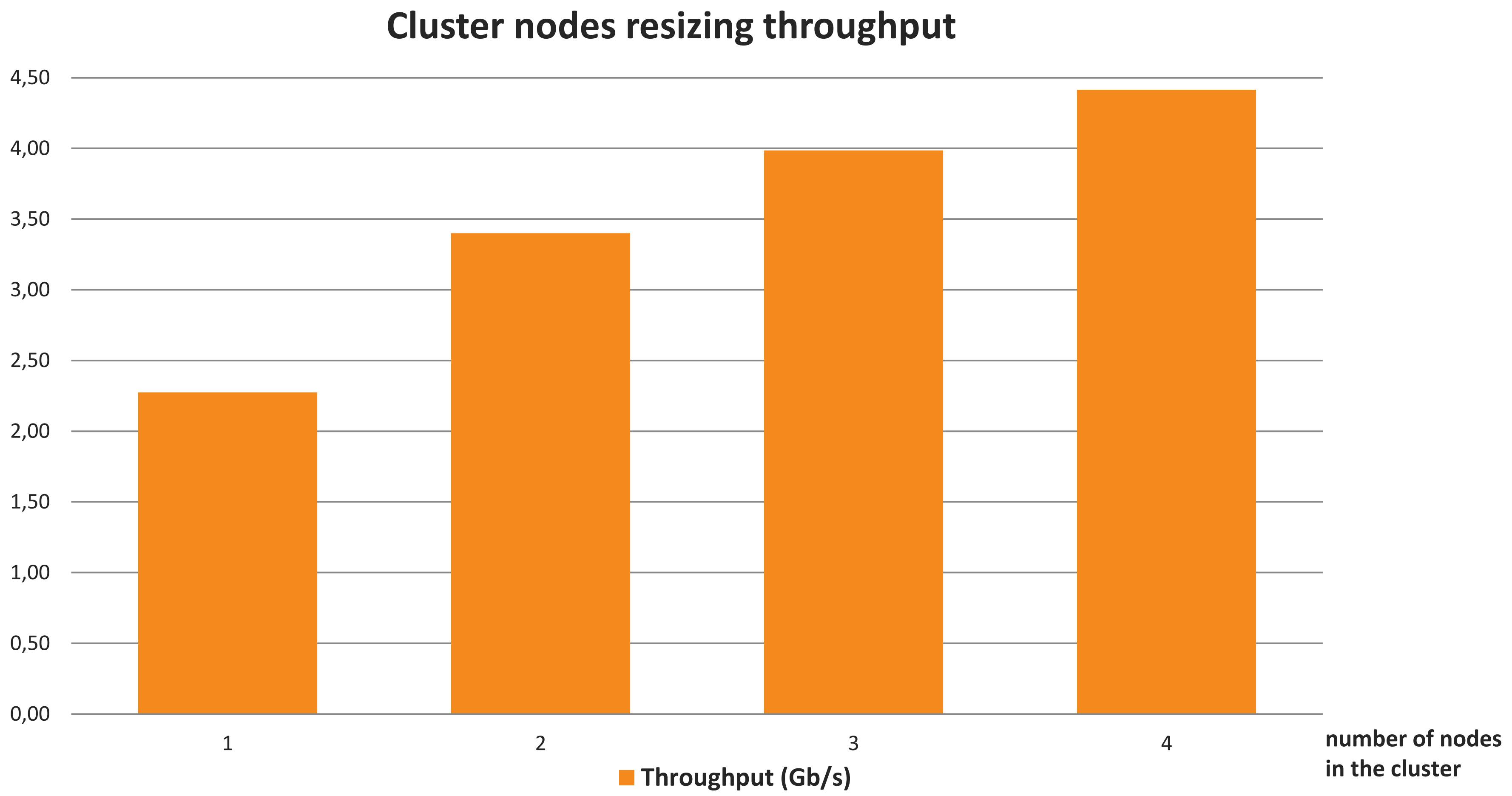
Scenario

- The input file (traces) is stored in the master node
- The master splits the file equally following the number of slaves and sends each part to a slave
- Slaves execute the algorithm on the portion of dataset they received
- Each slave sends back its results to the master



Dataset splitting

Experimentations



Conclusion

- Implementation of a parallel anti-malware framework on mobile GPU and parallella boards using behavioral detection techniques

- Series of optimizations to deal with the low memory problem of small scale embedded systems and the ever-increasing computing and memory requirements of malware detection
- Implementation of a Parallella beowulf cluster to further enhance parallelization of the anti-malware framework
- **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
- Integrating a monitor for the cluster architecture
- Expanding the cluster to a hetergenous one

Thank you for your attention