POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Tracing embedded heterogeneous systems

PROGRESS REPORT MEETING, MAY 2016

THOMAS BERTAULD

DIRECTED BY MICHEL DAGENAIS

Presentation plan

1. Introduction
2. The Keystone 2 architecture
3. BareCTF
4. Tracing embedded heterogeneous systems
5. The synchronization process
6. Use-case
7. Conclusion

Introduction -

Why tracing heterogeneous embedded systems ?

Have you ever wondered how images get processed inside these cameras?

Introduction -

Why tracing heterogeneous embedded systems ?

- Systems designed for specific needs/tasks
- Often used for real-time applications like signal processing
- Can be used anywhere
- Power-efficient
- Used inside much more complex systems

Introduction -

Challenges

- Different kind of processors
  - Some may be « unconventional » ones
  - Some may be « bare-metal » ones
- Complex and specialized hardwares
- Limited resources
  - No internal storage
  - Little RAM
- Lack of traditionnal tools

The Keystone 2 -

Specifications

- 66AK2H TI SoC
  - 4 ARM Cortex A15 running Linux (1.4 GHz)
  - 8 C66x TI CorePacs DSPs (1.2 GHz)
- 2 GB DDR3
- 6 MB Multicore Shared Memory
- http://www.ti.com/product/66AK2H12

The Keystone 2 -

Benefits and drawbacks

- Broadly used TI DSPs
- Powerful SoC
- 8 processors with built-in signal processing abilities
- TI's SYS/BIOS modules
- Full C support on the DSPs

- No way of tracing the DSPs
- Complex to use

BareCTF -

Tracing bare-metal systems

- Python tool created by Philippe Proulx (*EfficiOS*)
- Targets bare-metal systems
- Generates CTF traces
- Easy-to-use (configuration by YAML files)
- Lightweight
- https://github.com/efficios/barectf

Tracing embedded heterogeneous systems -

Facts and goal

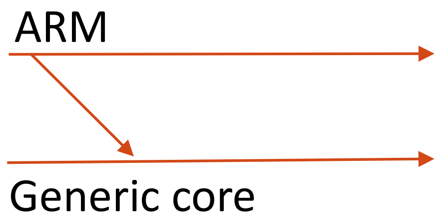
- LTTng can be used to trace the ARM side of any board
- BareCTF can be used to trace every other type of cores
- For what end?
  - Trace the whole application’s chain
  - Detect anomalies, bottlenecks, latencies…
  - Have a global view of a process distributed between different type of cores

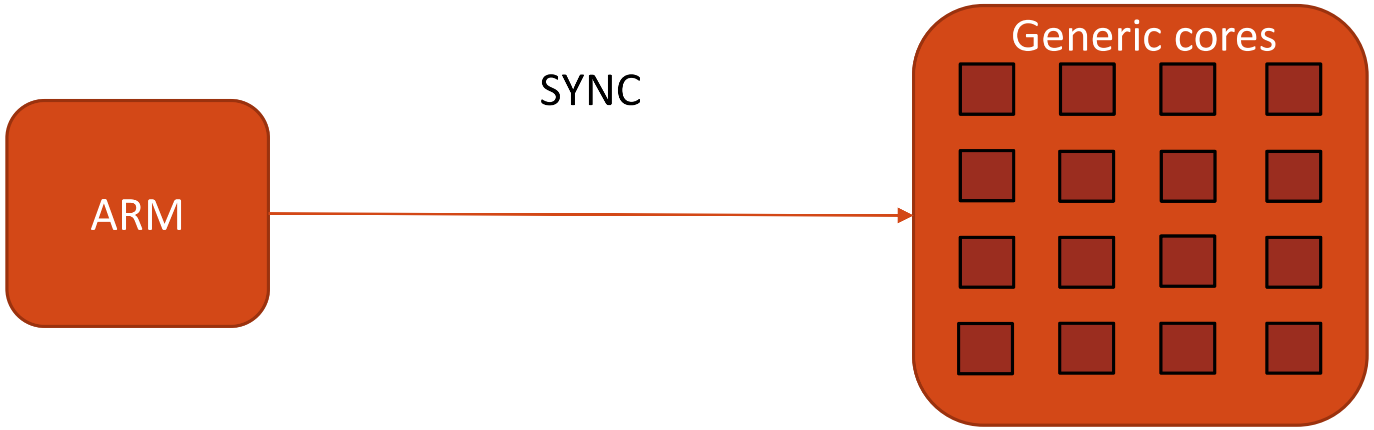
Tracing embedded heterogeneous systems -

Challenges

- BareCTF must be ported to any new platform
- The traces obtained from different processors must be synchronized
  - Necessity to generate matching events in each trace
  - Interrupt-based mechanism

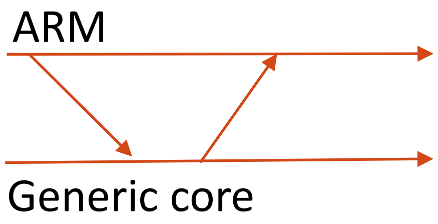
The synchronization process -
Description

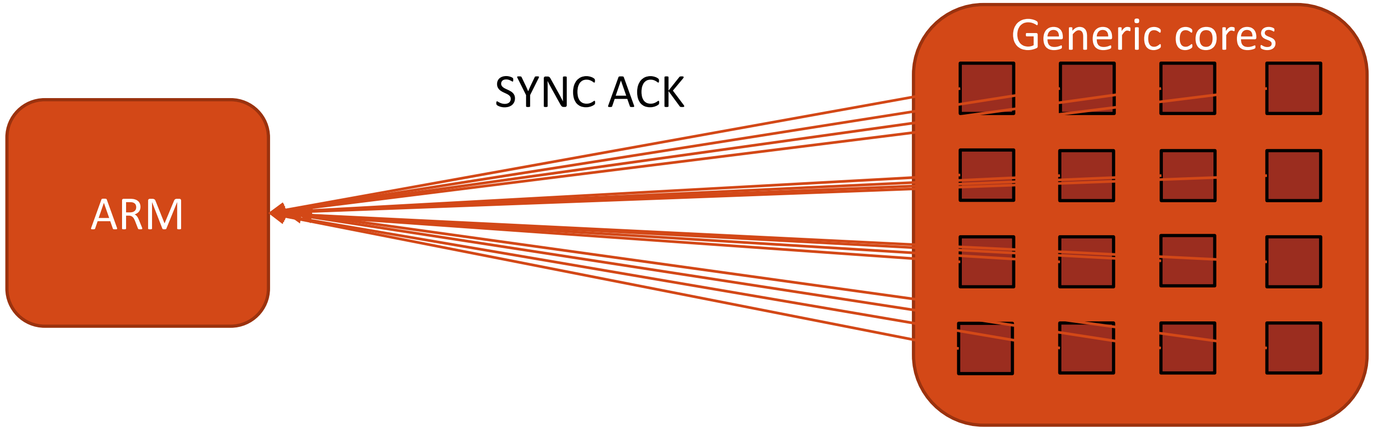




The synchronization process -

Description

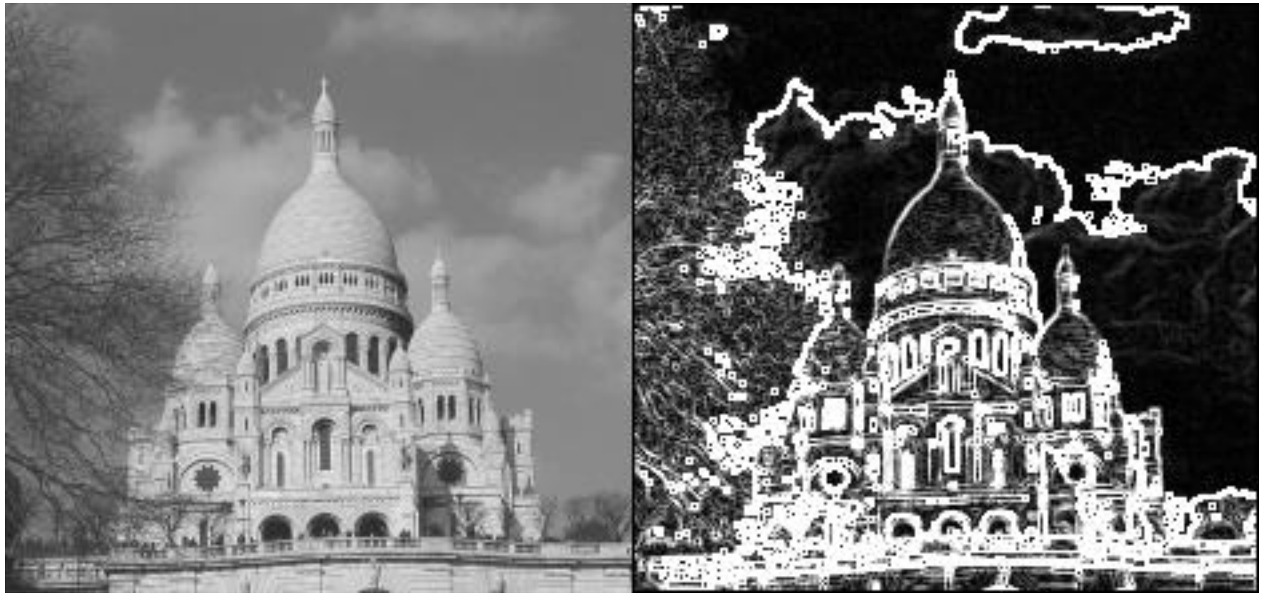




Use-case -

Description

- Instrumentation of an image processing algorithm
  - Edge detection
  - Sobel's filter



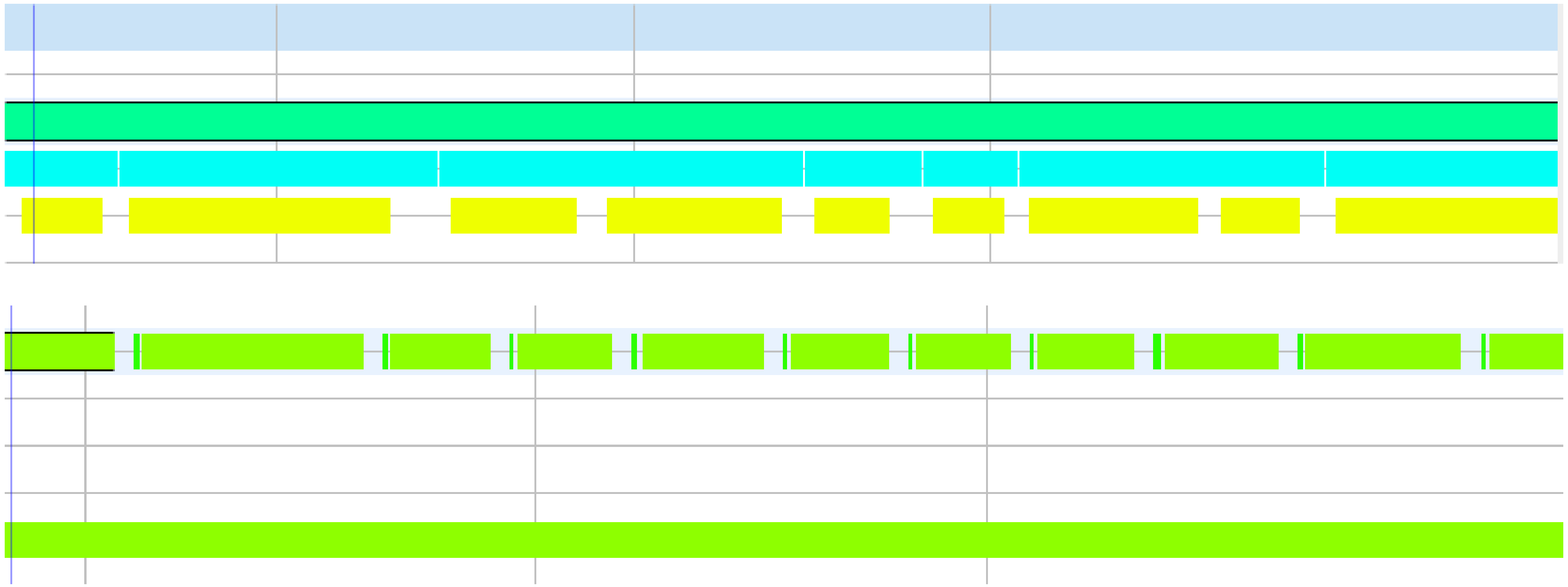
Use-case -

Setup

- 3000*3000 bmp image
- 1 ARM process acting as *master*
  - Gives commands
  - Sends input image and receives result
- 8 DSPs running acting as *slaves*
  - Wait for commands
  - Use TI’s ImgLib for image processing
  - In charge of memory management

Use-case -

Results



Use-case -

Results

- Low impact
  - ~95ms to ~96ms of processing time
- Effective
  - Can show if the work isn’t well balanced
  - Allows to keep track of the overall process
- Uses TraceCompass internal traces synchronization mechanism

Conclusion -

Limitations

- The barectf platform can be improved
  - Heavy API
  - High latency
  - Wasted memory
- The synchronization doesn’t take drift in account

Conclusion -

Future work

- Switch to more efficient message-passing methods
- Determine an optimal synchronization rate
- Improve the overall overhead of the barectf platform
- Tests on more complex systems

Thank you for your attention !

Contact : thomas.bertauld@gmail.com