Rationale and Contribution

- Modern computing systems complexity is skyrocketing mainly due to:
  - Tremendous availability and integration of heterogeneous resources
  - Demand for performance and reliability
- We present a **Self–Aware Adaptive** computing system blending techniques from different research fields to exploit available resources considering the execution context
Motivation example

- Thanks to the availability of heterogeneous resources, modern computing systems provide the ability to provide many different implementation of a functionality.
- When multiple implementations of a functionality are available, taking reasonable choices at compile time about which one better suits the execution context is not a trivial task.
Outline

• Introduction on Self-Aware Adaptive computing systems
• Context definition
• Proposed approach to implement Self-Aware Adaptive computing systems
  ▪ Observe
  ▪ Decide
  ▪ Act
• Experimental results
• Conclusions and Future works
Introduction (1 of 2)

- The key idea: manage technology and its complexity using technology itself changing the computing system behavior and resources management policies.

- The key characteristics:
  - Awareness
  - Adaptation
  - Approximation
  - Goal-orientation
• Self–Aware Adaptive computing systems follow the **Observe, Decide, and Act** loop:

![Diagram of Observe, Decide, and Act loop](image-url)
The literature is filled with works taking advantage of online monitoring/profiling, decision making, or hardware/software partitioning either static (compile time defined) or dynamic (run time defined).

Many of these works does not exploit all these features resulting in a lack of Self-Aware Adaptation.
• The use of static hardware/software partitioning to select an implementation of functionality prevents any kind of Self-Adaptation.

• The use of dynamic hardware/software partitioning allows Self-Adaptation on a degree depending on Self-Awareness which depends on online monitoring/profiling.
Proposed approach

- The Self–Aware Adaptive computing system we have developed fully exploit the ODA–loop to overcome some of the limits we outlined in state of the art solutions

- The ODA–loop is implemented by means of three sub–systems:
  - **Heartbeats**: observe
  - Heuristic decision making process: decide
  - **Implementation Switch Service**: act
• Online performance assertion and monitoring are performed by means of Heartbeats
• Heartbeats is simple yet extremely powerful library used to declare performance goals and to update and monitor the overall throughput of an application
• It is based on two simple concepts:
  ▪ Heartbeat
  ▪ Heart rate

The decision making process can be implemented in many different ways:

- Heuristic methods
- Probabilistic methods
- Machine learning techniques
- Control theory techniques

We implement a heuristic that uses information collected by Heartbeats and avoids oscillations.

---

(1) J. Eastep, et al.: Smartlocks: Lock Acquisition Scheduling for Self-Aware Synchronization (ICAC ‘10)
(2) M. Maggio, et al.: Controlling software applications via resource allocation within the Heartbeats framework (to appear in CDC ’10)
The ability to hot-swap an implementation of a given functionality in favor of another one proves to be a fundamental characteristic for a Self-Aware Adaptive computing system.

The Implementation Switch Service behavior has been inspired by the hot-swap mechanism available in K42.

J. Appavoo, et al.: Experience with K42, an open-source, Linux-compatible, scalable operating-system kernel (IBM Systems Journal ‘05)
• Switchable unit: **Dynamic-Link Library**
• State translation: “on-the-fly” translation of the “canonical data structure”
Experimental results

• Testing platform:
  - Xilinx XC2VP30 FPGA, IBM PowerPC 405, 256 MB of SDRAM, and 1 GB of Flash running a Linux–based operating system

• Static analysis of both the hardware and the software implementation of the DES cryptographic algorithm

• Dynamic analysis of the Self–Aware Adaptive computing system

• Online monitoring, decision making process, and dynamic reconfiguration overheads analysis
Static analysis

![Graph showing execution time vs. blocks for software, hardware, and reconfigurable hardware.]

- Software
- Hardware
- Reconfigurable Hardware

Execution time [ms]

Blocks [#]
Dynamic analysis

\[ \Delta \text{ observation delta} \quad m \text{ minimum heart rate} \]
\[ R \text{ reconfiguration time} \quad M \text{ maximum heart rate} \]

Heart rate vs Time

- \( t_0 \): Start time
- \( t_1, t_2, t_3 \): Observations
- \( t_4, t_5, t_6, t_7 \): Time points

\[ \Delta \text{ (delta)} \]

\[ R \]
Overheads (1 of 2)

Overhead on Execution time vs Blocks [#]

- Overhead
- Average

Average overhead: 3.52%
Overheads (2 of 2)

Overhead on Execution time

Execution times [ms]

Blocks [#]

Hardware
Reconfigurable Hardware
Overhead
Average Overhead

93.87%
Conclusions

• The proposed approach merges the potential of reconfigurable architectures with online performance assertion and monitoring, and adaptation capabilities.

• Experimental results show the goodness of the proposed approach when the computing system works within an unpredictable environment.

• The overhead of the online monitoring and decision making process proved to be sustainable.
Future works

- Run the Linux-based operating system on top of a multi-core processor sided with an FPGA used as an accelerator co-processor
- Implement an hot-swap mechanism within the Linux kernel to allow the implementation switch between device drivers optimized for different execution contexts