Using Workload Detection and Characterization for Power Management in Intel x86 MID platforms

Sujith Thomas

Intel Corporation

Acknowledgements

Harinarayanan Seshadri, Rajeev Muralidhar, Ananth R Krishna, Nithish Mahalingam, Vishwesh Rudramuni

Intel Corporation

Contents

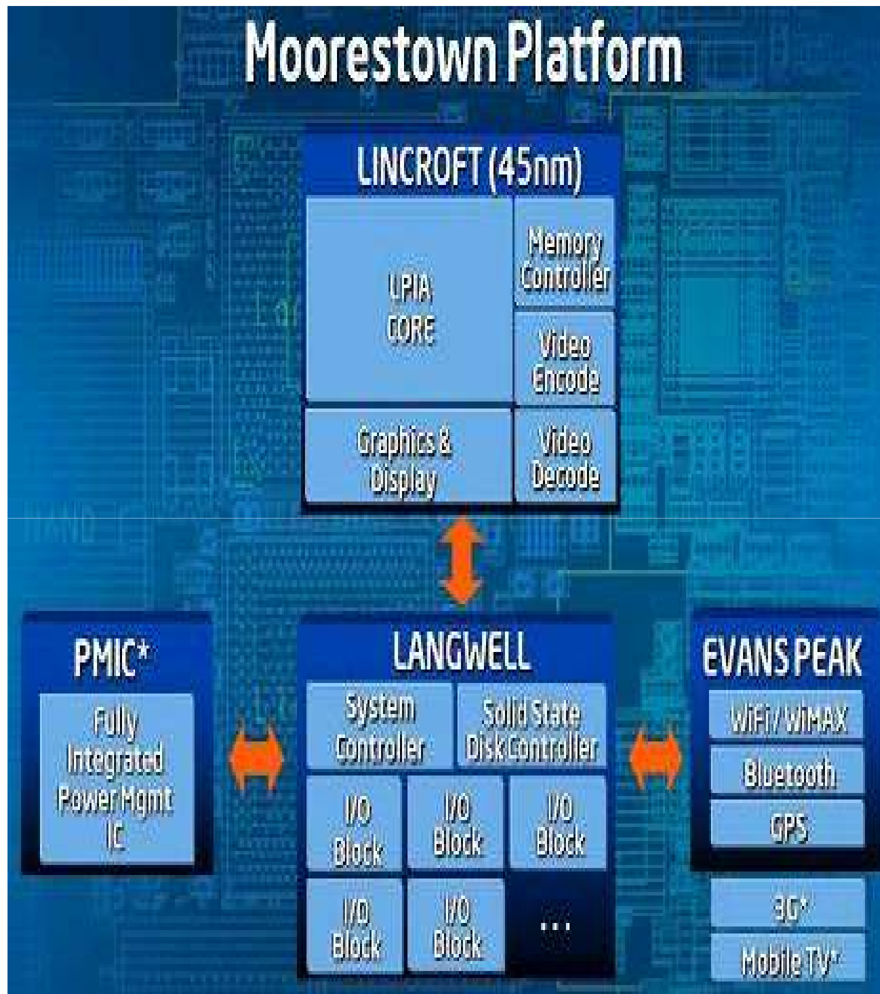
- ❑ Intel x86 MID/smartphone platform overview
- ❑ Power Management Overview & Challenges
- ❑ Dynamic workload detection
- ❑ Workload characterization and “mode” detection
- ❑ Results, future extensions

Introduction

During the past decade, Operating Systems for battery-operated portable and embedded systems have evolved significantly, addressing growing processor complexity and providing a more complete software platform for developing sophisticated applications.

Yet power management has remained an afterthought until now. With power efficiency becoming one of the dominant issues facing the electronics industry today, OSs are playing a pivotal role in energy management.

Intel© Moorestown x86 MID platform



- 3 chip MID platform
- Some kernel porting for legacy compatibility (PCI, SFI, IOAPIC emulation, etc.)
  - Refer earlier talk by Jacob Pan yesterday
  - Idle power management talk by German Monroy
- Enables different form factor devices – tablets, smartphones

Power Management in X86 MIDs

- CPU power management
  - Lincroft CPU supports deep C-states
  - Standard CPUIDLE-based C-state management through cpuidle governors

- CPU Performance management
  - Standard CPUFREQ based P-state management through cpufreq governors

- Device Power Management
  - Hardware becoming smarter and flexible wrt power management
  - Device drivers are becoming smarter wrt power management
  - Upcoming changes in Linux Runtime PM Framework – decentralized, autonomous power management by drivers

Power Management Challenges

Platform power management

- Idle power management
- Active power management

Idle power management

- Detecting subsystem/device idleness – device drivers should ideally do this on their own, in a device specific manner
- Based on this driver can autonomously manage its power state – new Linux Runtime PM

Active power management - What if:

- Devices do not detect/manage their idleness?
- Can we power down subsystems that are not actively involved in the current usage of the device? Need HW support for this
- For eg. If the workload identified on the platform is "video from hard-disk" subsystems like SDIO (WLAN), Bluetooth etc. can be transitioned to a low power state (if the devices do not do that on their own)

Can we do more by intelligently detecting "usage" of the platform?

Challenges with Active Power Management – Workload Characterization

- ❖ How can we detect a subsystem usage and selectively manage it without impacting user experience?

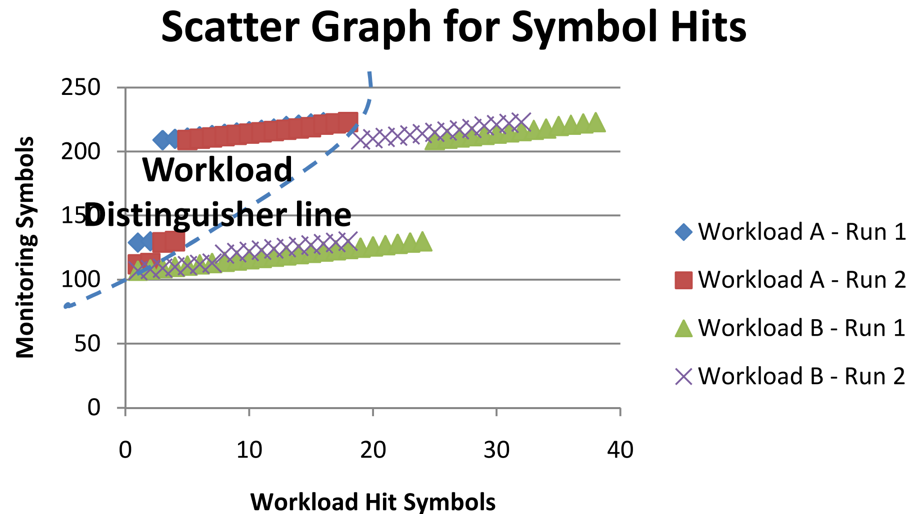
- ❖ How can we detect the type of workload running on the platform without causing overhead to the platform itself

- ❖ How can we minimize the latencies associated with recognizing workload changes in the platform?

Workload and Patterns

❖ Each workload has unique characteristics and it's a distinct signature for that work load

❖ The characteristics/signature of workload can be determined by observing certain parameters



The above plot shows how clear patterns can be established for the same workload and how we can distinguish workloads using pattern recognition techniques.

Using Kernel Kprobes

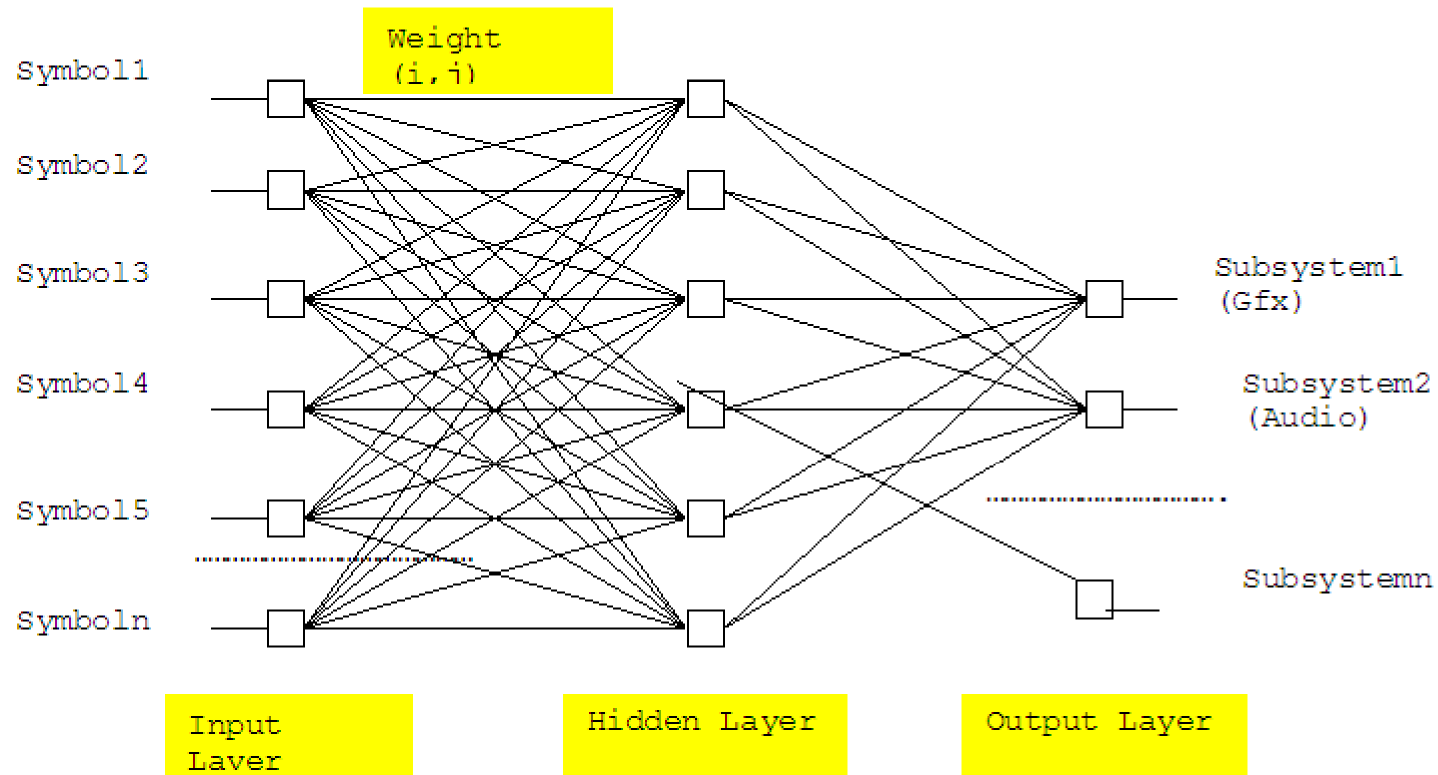
- Kprobes property, to trap at almost any kernel code address, specifying a handler routine is used in workload characterization and detection.

- Extensions to Kprobes : We are proposing a sysfs interface to set a group of symbols and another interface to read back the symbols that got hit.

The Design

1. Get the list of workloads/use-cases for the platform
2. Get the list of subsystems which can be power managed
3. Identify the core device drivers and extract the relevant symbols
4. For each use case, run a kernel profiler to collect the pattern
5. Train a Neural Network for these patterns
6. Transfer the ‘weights’ generated to the run-time platform
7. Install ‘kprobes’ for relevant symbols
8. Collect pattern at periodic intervals based on user configuration
9. Determine the inactive/active device and do mode transitions

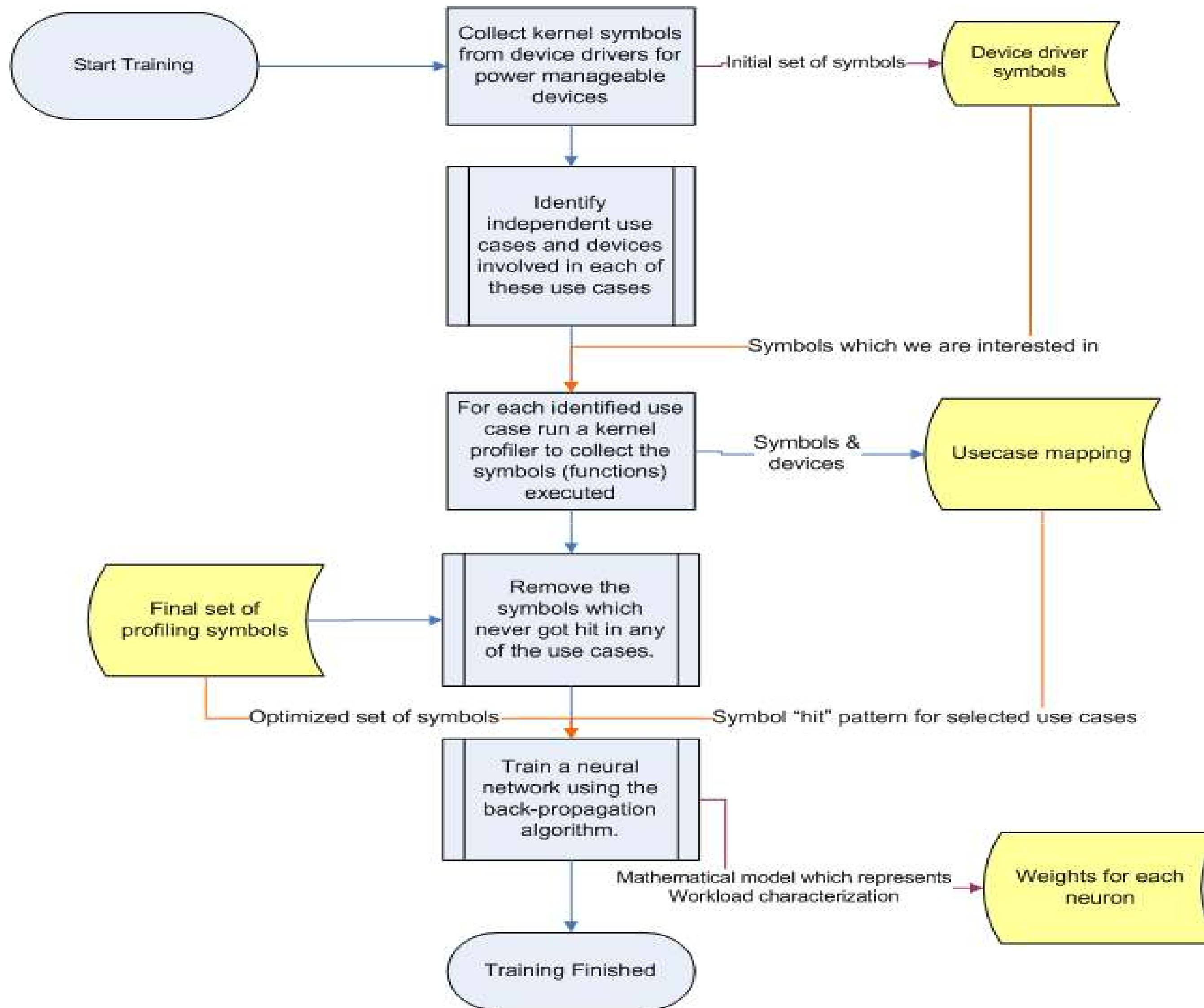
The proposed Neural Network



Training and testing the Neural Network

- Kprobes are installed for all kernel symbols which may be part of the pattern for the use cases in which we are interested
- Parameters obtained from profiling a use-case is given as input to the neural network
- Back Propagation algorithm is used to train the neural network, and to get the weights for all the links
- On training the neural network, a mathematical relationship between input and output is established
- A known set of symbols are provided to the trained neural network, and the predicted results are validated against the actual

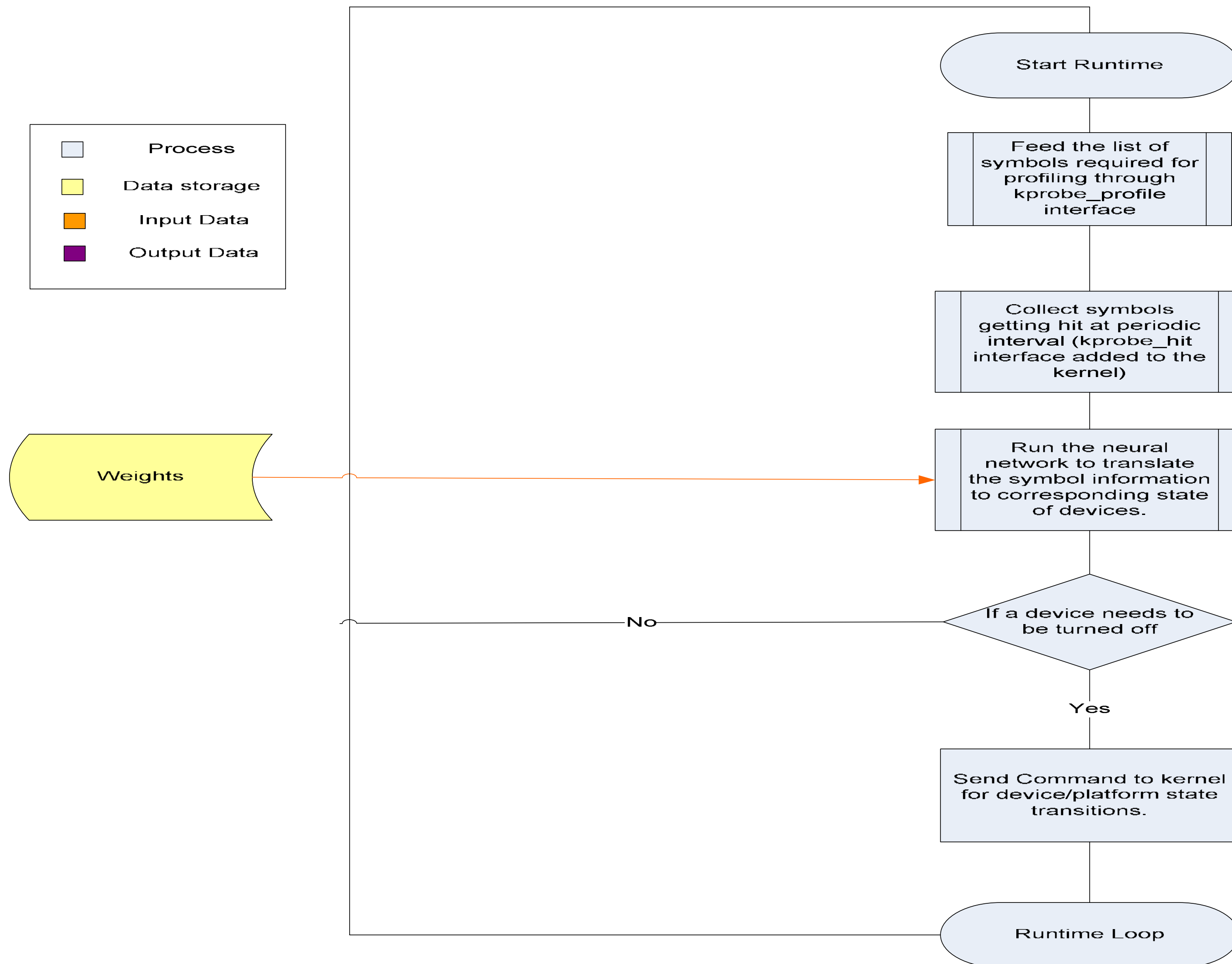
Training Phase Flow



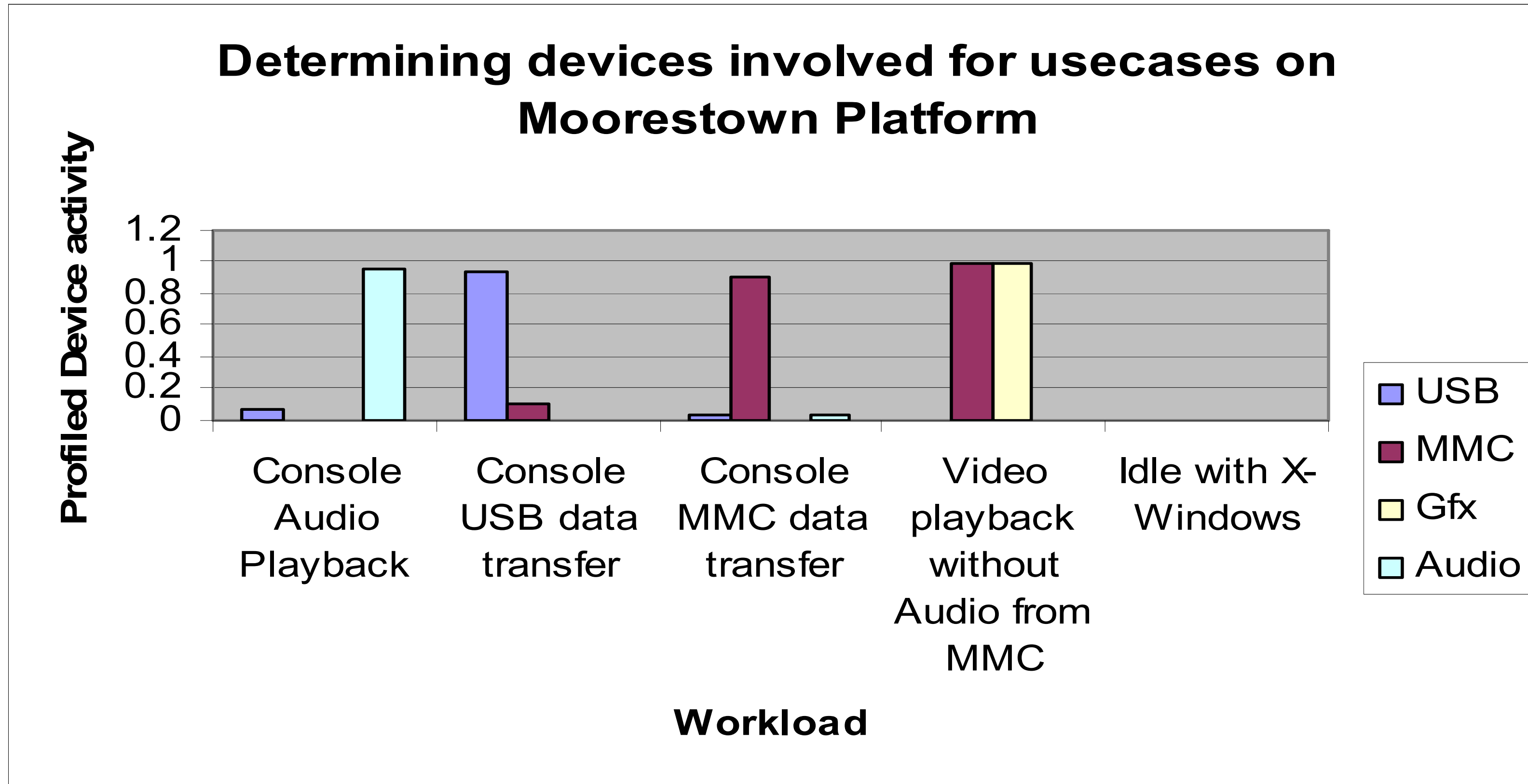
Run-time Prediction using Neural Network

- The weights obtained from the trained network is used for run-time prediction
- During a run-time usage scenario the kernel is continuously probed using Kprobe
- Fresh set of input are obtained as parameters and are passed to neural network as input
- Output from neural network aids in selective powering off the devices on the platform

Runtime Phase Flow



Results



Thank You