# Process Cruise Control

Philipp Andelfinger
Konstantinos Dagkas

Power Management Praktikum

# Big Picture

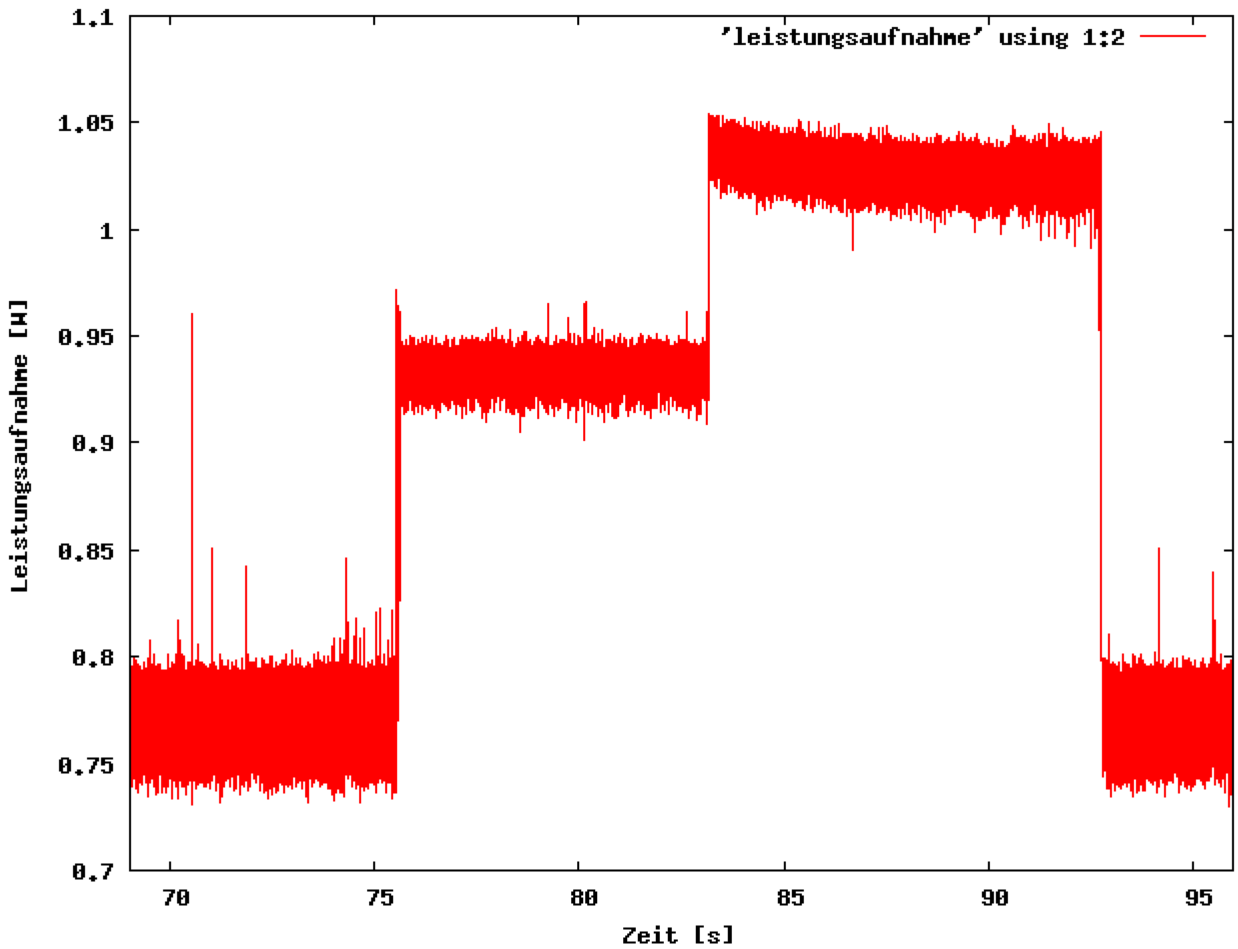
- Process Cruise Control
  - Event driven Frequency scaling
  - Power reduction Strategy:
    -  CPU intensive tasks : Max frequency
    -  Memory intensive tasks: Lower frequency

What Events do we need ?

- Instructions per second
- Memory requests per second

# PXA Platform

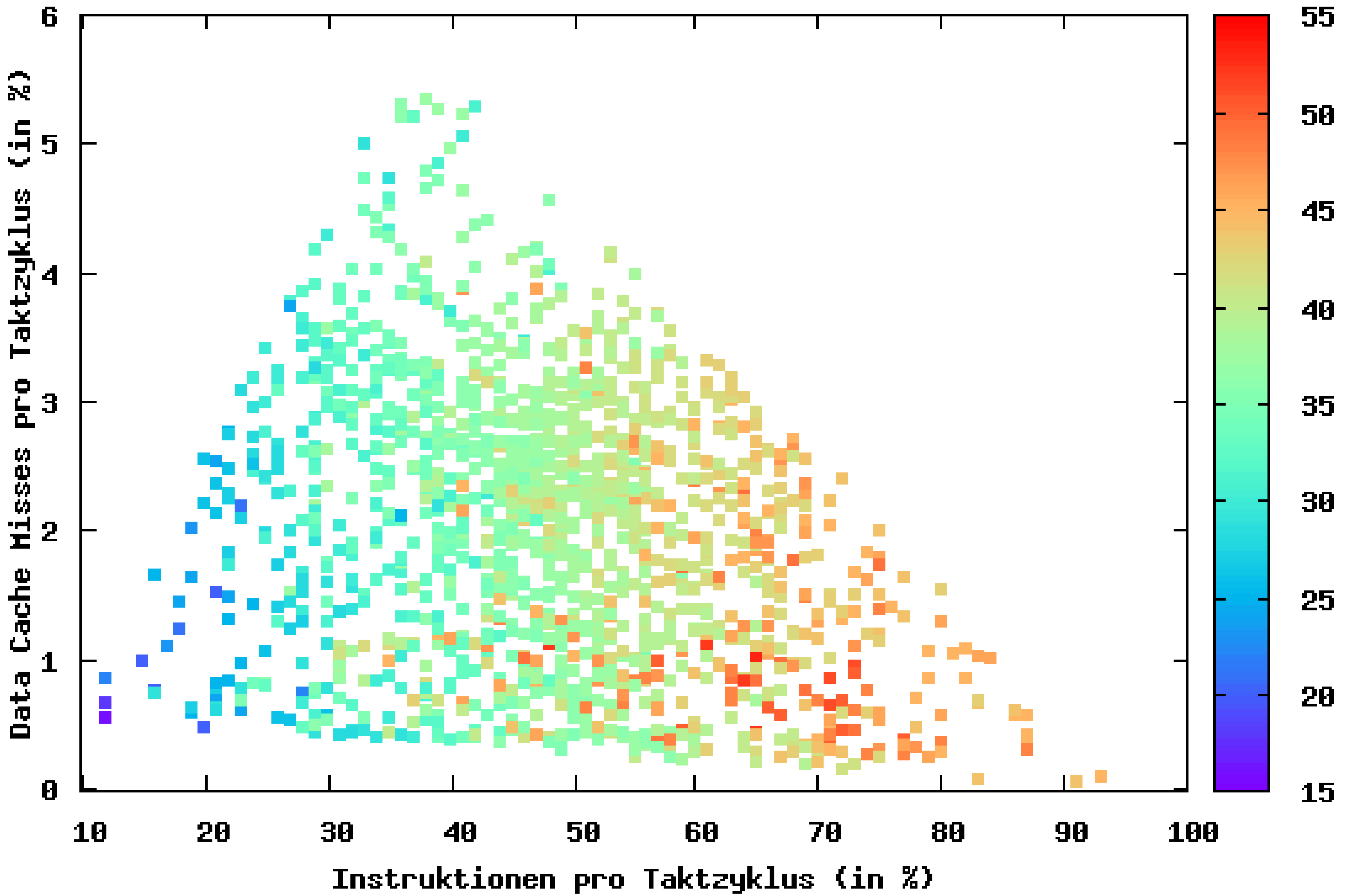
- 2 Registe
  - Instruc
  - Data c
- Time sta
- PXA built
  - Run-Tu ... en two pr
  - Pre-pro ... )



# Methodology

- Implementation process
  - PXA Hardware setup
  - Generate µBenchmarks (cpu intensive,memory intensive szenarios)
    - ✓ randomly controlled usage of preprogrammed µBenchmarks
      1. Simple memory load loop
      2. Simple asm mnemonic add
      3. Simple i/o read-write (/dev/zero->/dev/null)
    - ✓ Benchmark runs on PXA
  - Extract Perfomance Results
    - Instruction/cycle
    - Data cache misses/cycle
    - Perfomance loss: execution time (run mode) – ideal execution time (turbo mode)
  - Build frequency domains
    - Set maximum accepted perfomance loss (33%)

# Benchmark results



# Frequency decision matrix

```
int mode_table[2][20][20] =
{{{ 0,  0,  0,  0,  0,  0,  0,  0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  1,  0,  1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  0,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1, -1},
  { 0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1}},
```

-1 : no switch
0 : run mode
1 : turbo mode

# PCC Implementation

- Linux Kernel Modifications in sched.c

2. Add Event counter fields to task struct
    - initialization
3. Read perfomance counters pmn0,pmn1 and tsc
4. Calculate pmn0/tsc,pmn1/tsc
5. For each run of the scheduling function update pmn0,pmn1 for previously running process (with overflow handling)
6. Lookup appropriate frequency mode (matrix)
7. Set appropriate frequency (change run-turbo mode)