Praktikum
Power Management

Performance Counters

Philipp Kern, Michael Nagel
Universität Karlsruhe
und Rechtsnachfolger

Endergebnis I

Anzeige analog zu "top"

```
root@i30pm6:~$ python ./watchpower.py
------------------------------
    0       57.457209 W      GLOBAL VALUE
    1        0.001551 W      init [2]
 1089        0.000000 W      /sbin/syslogd
 1300        0.000000 W      /sbin/getty
 1301       24.157816 W      ./bench/bench_branch_multi
 1302       25.110561 W      ./bench/bench_branch_multi
 1303        8.140440 W      /usr/bin/python ./watchpower.py
 1285        0.000000 W      -zsh
 1192        0.000000 W      /usr/sbin/cron
 1215        0.000000 W      /sbin/getty 38400 tty1
 1216        0.000000 W      /sbin/getty 38400 tty2
 1217        0.000000 W      /sbin/getty 38400 tty3
 1218        0.000000 W      /sbin/getty 38400 tty4
 1219        0.000000 W      /sbin/getty 38400 tty5
 1220        0.000000 W      /sbin/getty 38400 tty6
 1098        0.000000 W      /sbin/klogd -x
 1107        0.000000 W      /usr/sbin/acpid
  989        0.000000 W      dhclient3
 1119        0.000000 W      /usr/sbin/sshd
 1129        0.000000 W      /usr/sbin/inetd
 1282        0.027798 W      sshd: root@pts/0
 1144        0.003523 W      /usr/sbin/ntpd
   -1        0.000000 W      IDLE THREAD
* * *        0.002668 W      >>> ALL KLTs <<<
*sum*       57.453082 W
------------------------------
```

Endergebnis II

Anzeige analog zu "time"

```
root@i30pm6:~$ python ./power.py bench/bench_branch_multi <<< 50
Child process exited, starting calculations...

             Counter                   nJ             J          Counter value
      unhalted_cycles:     566492112240 nJ ==   566.49 J           49260183673
       memory_retired:      46042077166 nJ ==    46.04 J           16385080842
                  tsc:     514898859753 nJ ==   514.90 J           63255388176
     retired_branches:       3834670905 nJ ==     3.83 J            4195482391
       floating_point:          1019052 nJ ==     0.00 J               1462054
mispredicted_branches:     162310857216 nJ ==   162.00 J             294041408
   ld_miss_l1_retired:       2132373507 nJ ==     2.13 J              96487489
      mob_load_replay:     298041823177 nJ ==   298.04 J            6132547802
     mop_queue_writes:     501752872147 nJ ==   501.75 J           65078193534
Sum:            2.09550666516e+12 nJ == 2095.507 J
Average:        33.128 W (based on tsc)
Average:        42.540 W (based on unhalted_cycles)

difference tsc over unhalted_cycles: 13995204503
```

Performance-Counter I

- CPU zählt gewisse Ereignisse mit
  - eigentlich für Compiler-Bauer
- Performance Counter Interface
  - Initialisieren der Zähler
  - Auswahl der Ereignisse
  - Auslesen der Zählerwerte

Performance-Counter II

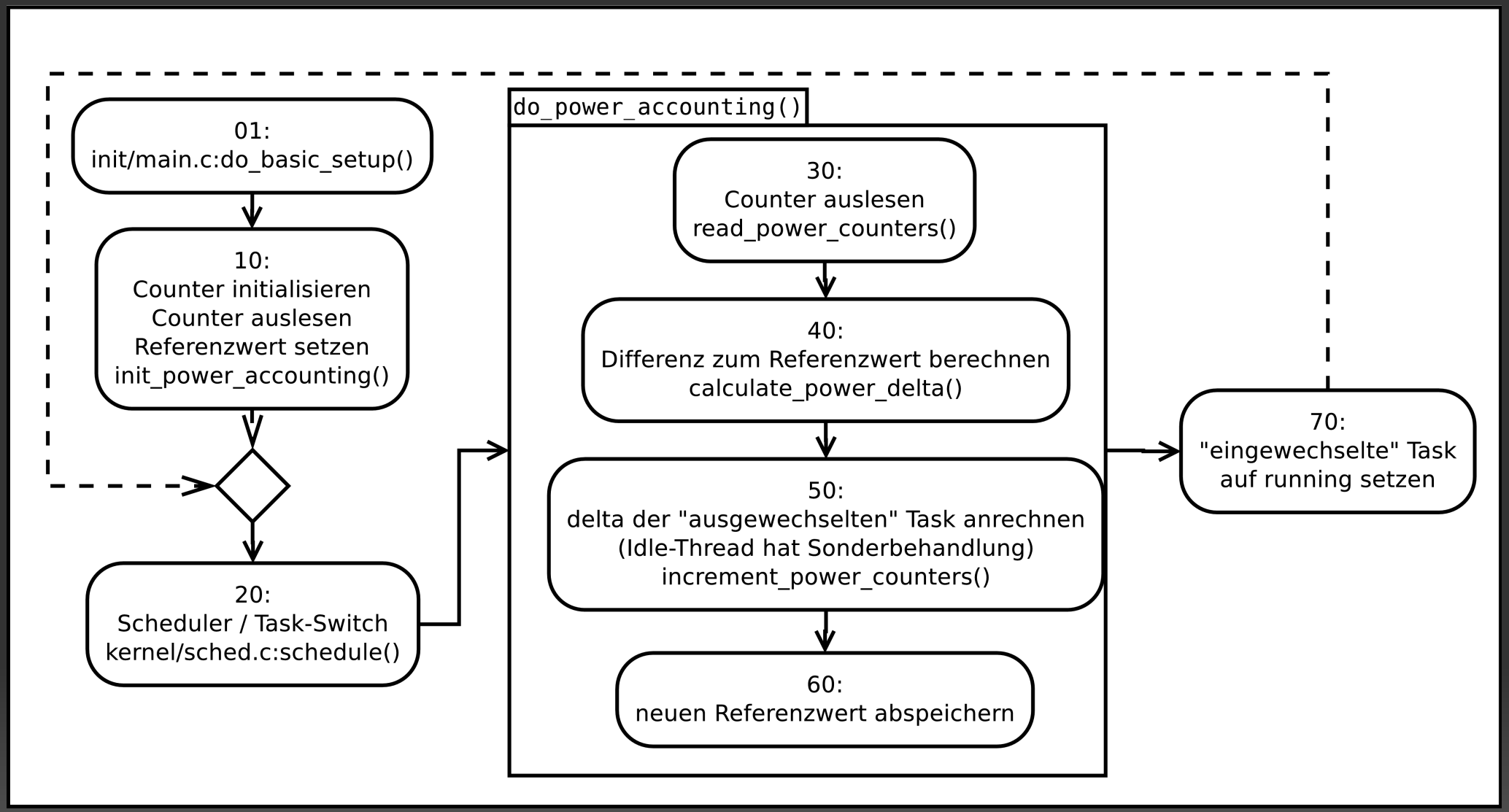
$$Energieverbrauch = \sum EreignisCount * Energieverbrauch(Einzelereignis)$$

- Zählung pro Thread
  - alle Grundlagen für die Tools vorhanden

Ereignis-Auswahl & Gewichte

- Somebody Else's Problem
  - Auswahl der gezählten Ereignisse und Ereignis-Gewichte übernommen aus Studienarbeit von Simon Kellner
- Ziele
  - möglichst geringer Fehler
  - durchführbar
  - damals: Energieverbrauch überschätzen

Flowchart



- basierend auf 2.6.31
  - aktuell zu Beginn des Praktikums

SysFS, ProcFS

- SysFS
  - globaler Zähler
  - Idle-Pseudo-Thread
  - `/sys/kernel/power_counters/{current,idle}/$COUNTER_NAME`
- ProcFS
  - 9 Einträge pro Prozess
  - `/proc/$PID/power/$COUNTER_NAME`

unsere Struct

```
struct power_counters {
        u64 tsc;
        u64 unhalted_cycles;
        u64 mop_queue_writes;
        u64 retired_branches;
        u64 mispredicted_branches;
        u64 memory_retired;
        u64 mob_load_replay;
        u64 ld_miss_l1_retired;
        u64 floating_point;
};
```

- Hardware-Genauigkeit außer `tsc` eigentlich nur 40 Bit
- 64 Bit vereinfacht das Overflow-Handling

unsere Funktionen

```
/* arch/x86/kernel/power_accounting.c */

// Zählung initialisieren
extern void init_power_accounting(void);

// aktuellen Stand in Struct einlesen
extern void read_power_counters(struct power_counters* pcs);

// Differenz zwischen zwei Structs berechnen
extern void calculate_power_delta(
    struct power_counters* res,
    struct power_counters* before,
    struct power_counters* after);
// Structs addieren (res += value)
extern void increment_power_counters(
    struct power_counters* res,
    struct power_counters* value);

// Referenz-Struct, verwendet do_power_accounting beim Taskswitch
extern struct power_counters power_counters_ref;

// Delta seit letztem Aufruf auf prev addieren
extern void do_power_accounting(
    struct task_struct *prev,
    bool is_idle);
```

Userspace Programme I

- analog zu top
  - Zähler regelmäßig aus ProcFS auslesen
  - FloatingPoint Rechnungen durchführen
  - Aging der Werte
  - Ausgabe der Watt-Zahl pro Prozess

Userspace Programme II

- analog zu time
  - Kind forken
  - auf sigchld warten
  - Zähler einsammeln und anzeigen
  - Zombie freigeben

Endergebnis I

Anzeige analog zu "top"

```
root@i30pm6:~$ python ./watchpower.py
------------------------------
    0       57.457209 W      GLOBAL VALUE
    1        0.001551 W      init [2]
 1089        0.000000 W      /sbin/syslogd
 1300        0.000000 W      /sbin/getty
 1301       24.157816 W      ./bench/bench_branch_multi
 1302       25.110561 W      ./bench/bench_branch_multi
 1303        8.140440 W      /usr/bin/python ./watchpower.py
 1285        0.000000 W      -zsh
 1192        0.000000 W      /usr/sbin/cron
 1215        0.000000 W      /sbin/getty 38400 tty1
 1216        0.000000 W      /sbin/getty 38400 tty2
 1217        0.000000 W      /sbin/getty 38400 tty3
 1218        0.000000 W      /sbin/getty 38400 tty4
 1219        0.000000 W      /sbin/getty 38400 tty5
 1220        0.000000 W      /sbin/getty 38400 tty6
 1098        0.000000 W      /sbin/klogd -x
 1107        0.000000 W      /usr/sbin/acpid
  989        0.000000 W      dhclient3
 1119        0.000000 W      /usr/sbin/sshd
 1129        0.000000 W      /usr/sbin/inetd
 1282        0.027798 W      sshd: root@pts/0
 1144        0.003523 W      /usr/sbin/ntpd
   -1        0.000000 W      IDLE THREAD
* * *        0.002668 W      >>> ALL KLTs <<<
*sum*       57.453082 W
------------------------------
```

Endergebnis II

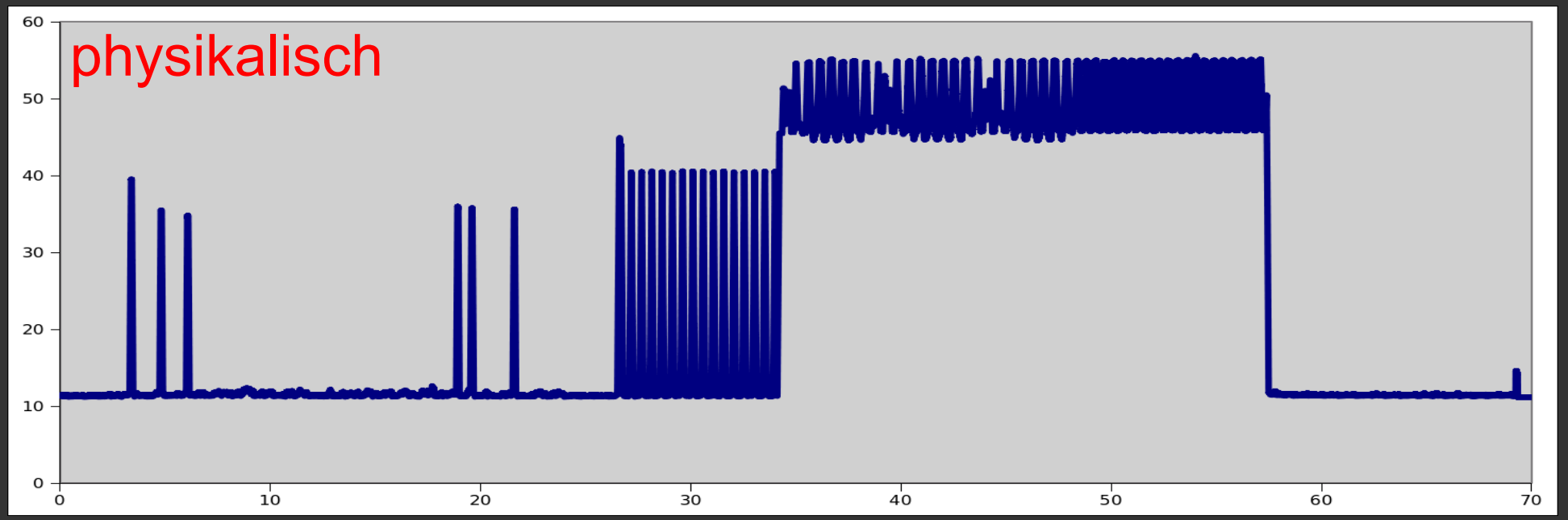
Anzeige analog zu "time"

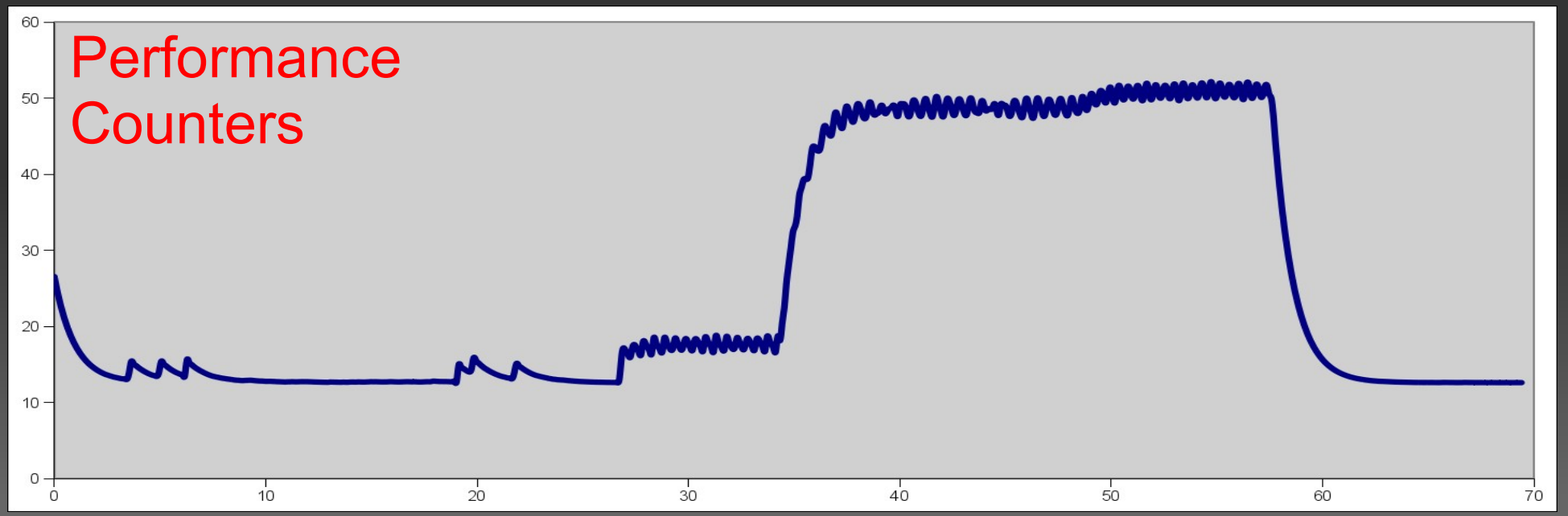
```
root@i30pm6:~$ python ./power.py bench/bench_branch_multi <<< 50
Child process exited, starting calculations...

             Counter                   nJ             J          Counter value
      unhalted_cycles:     566492112240 nJ ==   566.49 J           49260183673
       memory_retired:      46042077166 nJ ==    46.04 J           16385080842
                  tsc:     514898859753 nJ ==   514.90 J           63255388176
     retired_branches:       3834670905 nJ ==     3.83 J            4195482391
       floating_point:          1019052 nJ ==     0.00 J               1462054
mispredicted_branches:     162310857216 nJ ==   162.00 J             294041408
   ld_miss_l1_retired:       2132373507 nJ ==     2.13 J              96487489
      mob_load_replay:     298041823177 nJ ==   298.04 J            6132547802
     mop_queue_writes:     501752872147 nJ ==   501.75 J           65078193534
Sum:            2.09550666516e+12 nJ == 2095.507 J
Average:        33.128 W (based on tsc)
Average:        42.540 W (based on unhalted_cycles)

difference tsc over unhalted_cycles: 13995204503
```

Messung


physikalisch
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70


Performance
Counters
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70

Offene Punkte

- Zählerstände im Userspace interpretiert
  - kein Scheduling anhand der Schätzung
- Code ist stark architektur-abhängig
  - läuft nur auf P4/NetBurst
- wrmsr/Zählersetup blind übernommen
- Interrupt-Abrechnung an laufende Task
- globaler Referenzwert
  - kein Hyperthreading, kein Multicore
- Laufzeitoverhead (~8W)
  - Python, mehrfaches Auslesen, stupide Ausgabe, …

Backup-Folien

iteratives Aging:

$decay = 0.9$

$assert\ 0 \leq decay \leq 1$

$wert_i := wert_{i\_1} * decay + messwert_i * (1 - decay)$

setzt monotone Taktung vorraus und ergibt Halbwertszeit von n Takten mit

$decay^n = 0.5$

$n := \log_{decay} 0.5$