AKBP/ES II: Team 4

Cruise Control

Holger Scherl & Matthias Faerber

snhosche@cip.informatik.uni-erlangen.de

simafaer@informatik.stud.uni-erlangen.de

Friedrich-Alexander-Universität Erlangen-Nürnberg

Öko-Informatik – Warum?

Heutige Anwendungen fordern

- Leistungsfähigere Prozessoren
- Große Hauptspeicher
- Energieintensive I/O-Geräte (Wireless LAN, Bluetooth, etc.)

Folgen:

- Kurze Betriebsdauer von Mobilgeräten
- Hohe Kosten im Serverbetrieb

Versuchsaufbau

Intel 80200 basierend auf Intel XScale Microarchitecture

- variable Taktfrequenz der CPU
- 9 Geschwindigkeitsstufen (0-733 MHz, 6 benutzbar)
- maximale Energieaufnahme 900mW

Cyclone IQ 80310 IO Board

- Stromversorgung durch den PCI-BUS eines Desktop-PCs
- Root-Filesystem über NFS
- Console auf serieller Schnittstelle
- Netzwerkverbindung (rlogin)

Unser Ansatz (Cruise Control)

Grundsätze:

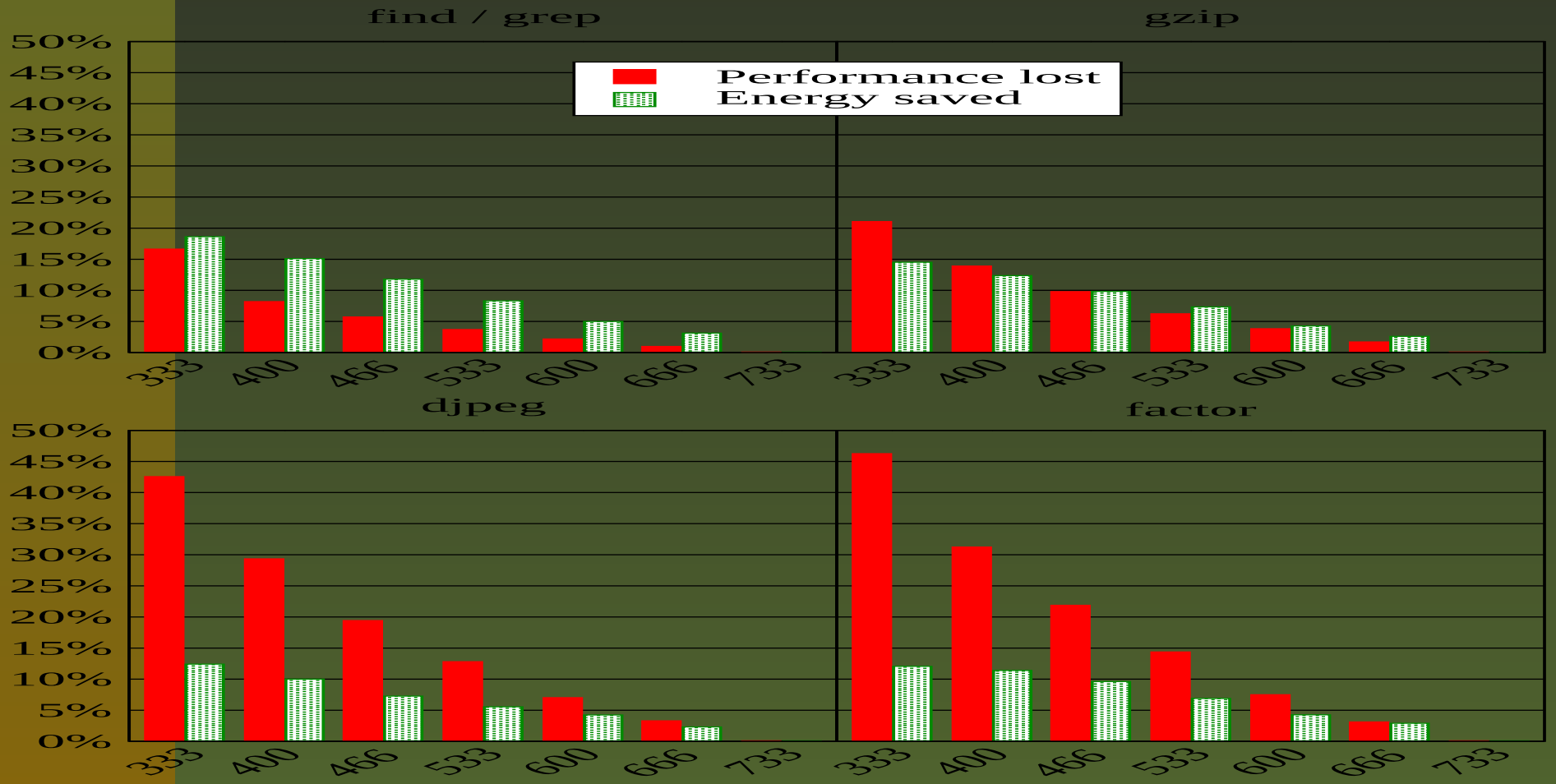
- Energieverbrauch $\sim$ Taktfrequenz
- Hauptspeicherzugriffe erfordern Wartezyklen (Busy-Wait-Loop)

$\Rightarrow$ effizientere Ausnutzung der Energie möglich

- Alle Prozesse rechnen mit eigener Taktfrequenz
- Dynamische Anpassung der Taktfrequenz bei jedem Taskwechsel
- Prozesscharakterisierung durch Performance-Counter

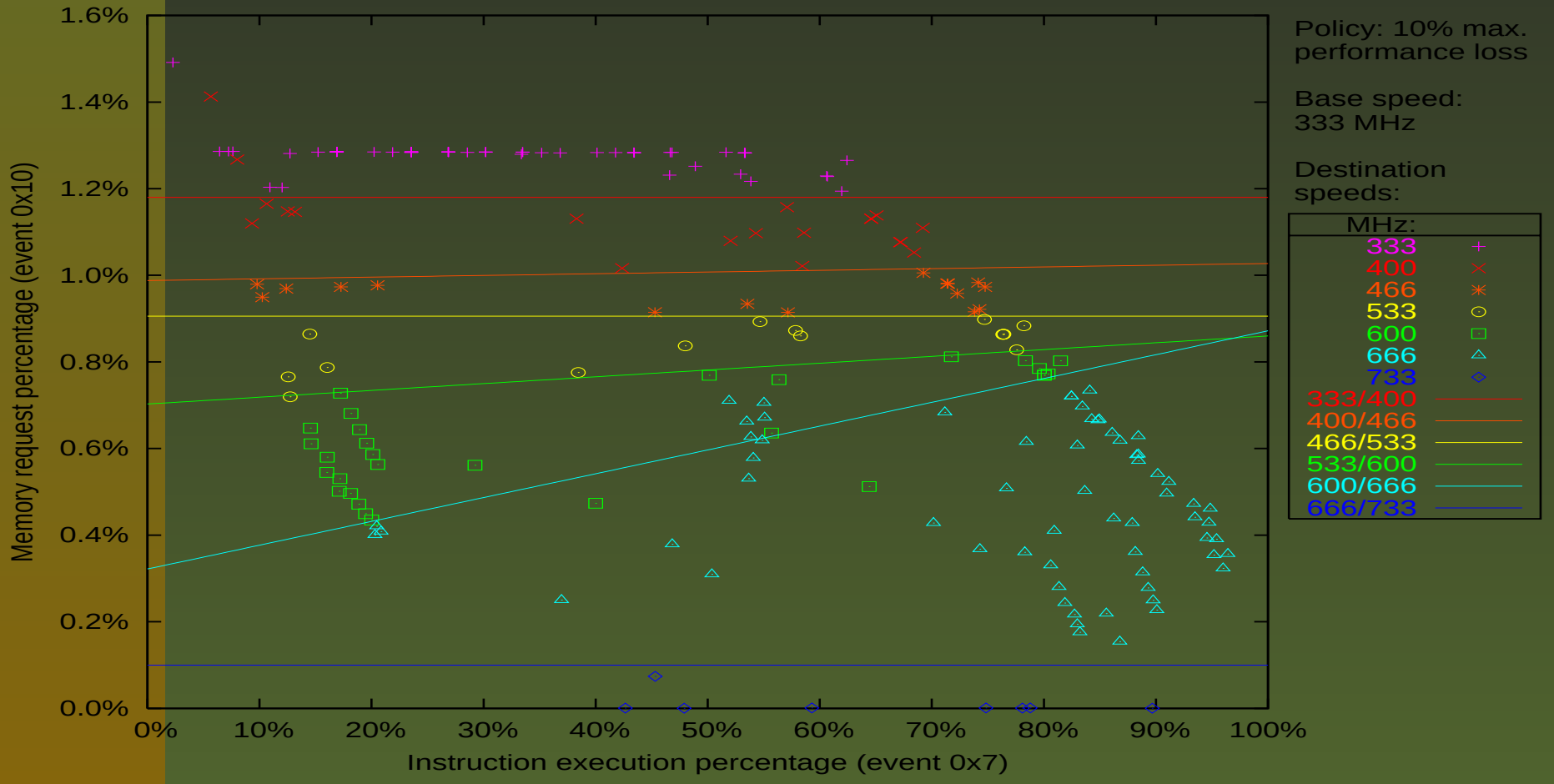
Beispiele

Energie – Performanz



Theorie

Modell zur Anpassung der Geschwindigkeit:



Modifikationen am Kern

Zusätzliche Dateien im `/proc`-Dateisystem:

- `cpuspeed` zur Ausgabe des aktuellen Status des Prozessors (Geschwindigkeit, Performance Counter)
- `policy` zum Einstellen und Ausgabe der Scheduling-Policy
- `power` mit Informationen zu allen laufenden Prozessen

Modifikationen am Kern

Ausschnitt aus der Datei `/proc/speedstep/power`:

```
Dynamic Speed Selection is enabled.
Speed range: 333-733  Default speed set to 733
  Speed     TimCtr   TimAvg    CpuCtr  CpuAvg    MemCtr MemAvg   Cmd
-------------------------------------------------------------------------
 1  333   48067695   258360   6162492   37101    303008   2045  init
 3  466     327956    30542     29358    2701      1737    161  ksoftirqd
 4  333     178107    17810     11502    1150      1107    110  kswapd
 9  333    9576443   337281    885124   30455     50434   1775  rpciod
15  733    3940548   359118    182992   16628     21413   1951  xinetd
26  733   24965616   383159   1604506   18883    140024   2134  bash
30  333  459584859   239494  31896742   23571   2792112   2014  in.rlogind
32  333  356260295   595730  29715051   45619   2107981   4826  bash
40  733  232449089   855782  22597836  100135   1202959   4278  grep
41  333  860781780 10179133 241087104 1888447 310422125 129639  mem_test
44  733 2073656107  5747509 118986737 5361518   4320134   1494  reg_test
62  533  395742892   558057  35831768   82214   2359285   3712  find
63  533  366596749   858152  31482257  148477   2174457   5565  cat
```

Modifikationen am Kern

Scheduler:

- `speedstep_adjust_thread` vor `switch_to`
  Auswertung der Performance Counter und Neuberechnung der optimalen Taktfrequenz des alten Threads
- `speedstep_switch_to` nach `switch_to`
  Wechsel auf die Geschwindigkeit des neuen Threads

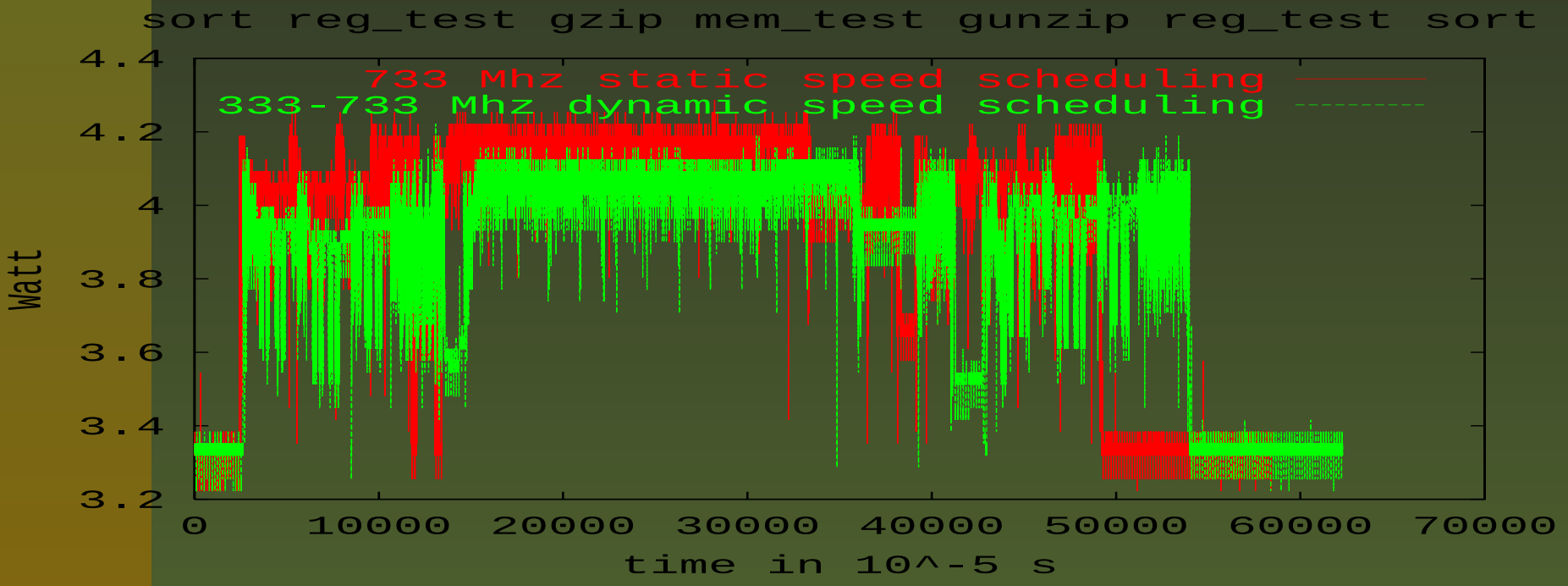
Ergebnisse

Energieschema für den Aufruf:
”reg_test; mem_test”

$\Delta E = 0,143\text{J}; \Delta t = 56\text{ms}$

Ergebnisse

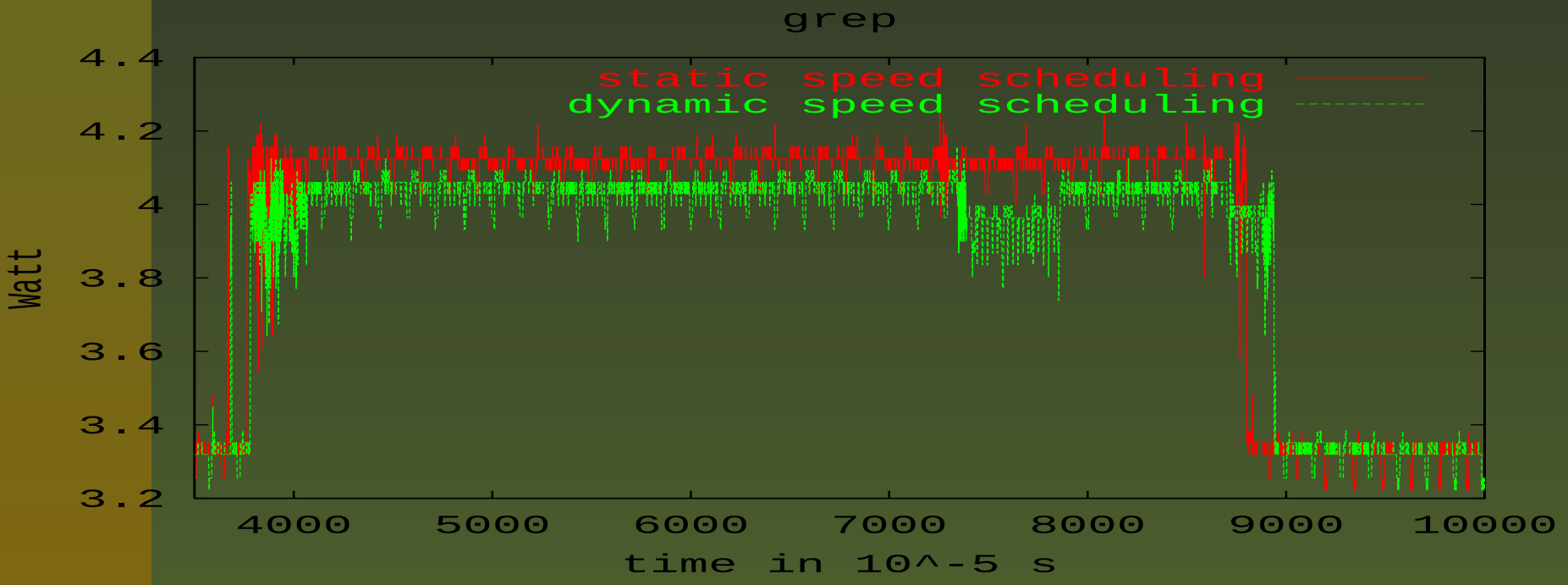
Energieschema für den Aufruf:
”sort | reg_test | gzip | mem_test | gunzip | sort”



$\Delta E = 0,307\text{J}; \Delta t = 480\text{ms}$

Ergebnisse

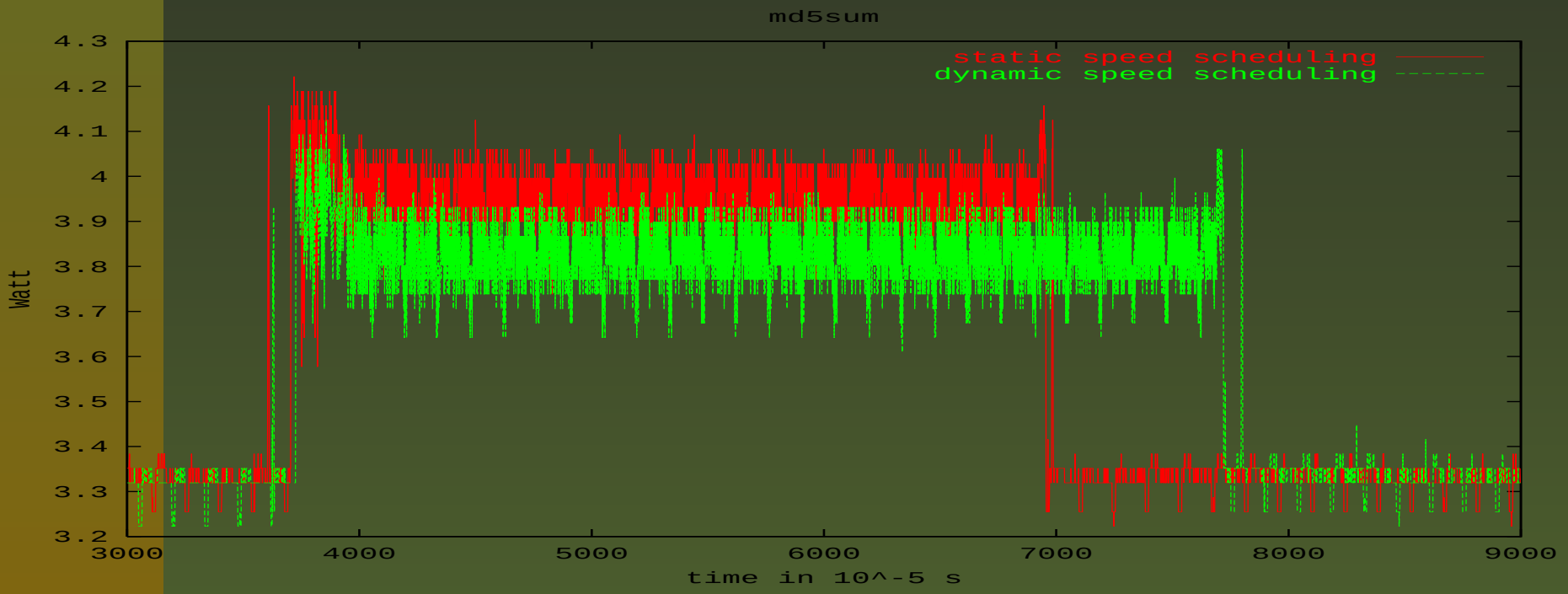
Energieschema für den Aufruf: ”grep”



$\Delta E = 0,035\text{J}; \Delta t = 14\text{ms}$

Ergebnisse

Energieschema für den Aufruf: ”md5sum”



$\Delta E = 0,010\text{J}; \Delta t = 74\text{ms}$

Meßergebnisse

Test Programm	Energieersparnis	Zeitverlust
reg_mem	5,8%	3,5%
sort_mem_reg	9,8%	10,3%
grep	9,7%	2,8%
md5sum	5,5%	22,9%
mem_test	22,4%	0,4%

Fazit

Nach grober Schätzung müssen wir den XScale 43 Jahre lang betreiben, um die zur Entwicklung des Treibers verbrauchte Energie (Pizzaofen, Monintore, etc.) wieder einzusparen !!!

Noch Fragen ???