Power-Aware Computing

A Kernel-Directed Compiler-Assisted Dynamic Voltage Scaling Algorithm

Stefan Büttcher

stefan@buettcher.org

Computer Science Department
FAU Erlangen-Nürnberg

Übersicht

- Motivation
- Frequenz- und Spannungsskalierung
- Struktur des kooperativen Algorithmus
- Betriebssystem-Kern
- Compiler-Unterstützung
- Energie-Messungen
- Zusammenfassung, Ausblick

Leistungsaufnahme von CPUs

Leistungsfähigkeit und Energiebedarf von Mikroprozessoren sind in den letzten 10 Jahren rasant gewachsen.

- 1994: Intel Pentium 100 MHz, max. 10 W
- 2003: AMD Athlon XP 2800+, max. 68 W
- 2003: Intel Pentium 4 3.0 GHz, max. 82 W
- 2004: Intel Pentium 4 3.4 GHz, max. 103 W

Wartezeiten & Energiebedarf

Trotz geringerer relative Leistungsaufnahme (Joule pro Instruktion) ist Energiebedarf gestiegen: Viele Rechner warten die meiste Zeit.

Offensichtliche Gründe für Wartezeiten:

- keine Arbeit vorhanden
- Peripherie (Festplatte, Modem, ...)

Nicht ganz so offensichtlich:

- Latenzzeit des Hauptspeichers

Energiespar-Möglichkeiten

Möglichkeiten, während der Wartezeiten Energie einzusparen:

- Standby-Modi
- Taktdrosselung (Clock Throttling, z.B. ACPI)
- Änderung der CPU-Frequenz (Dynamic Frequency Scaling, DFS)
- Änderung der Versorgungsspannung (Dynamic Voltage Scaling, DVS)

Übersicht

- Motivation
- Frequenz- und Spannungsskalierung
- Struktur des kooperativen Algorithmus
- Betriebssystem-Kern
- Compiler-Unterstützung
- Energie-Messungen
- Zusammenfassung, Ausblick

Dyn. Frequenzänderung (DFS)

Leistungsaufnahme eines Computersystems bei Frequenz f:

$$P_{total,f} = P_{CPU,f} + P_{other} = c \cdot f + P_{other}.$$

Halbierung der Taktfrequenz:

$$P_{total,\frac{f}{2}} = P_{CPU,\frac{f}{2}} + P_{other} = \frac{c}{2} \cdot P_{CPU} + P_{other}$$

Aber: eventuell auch Verdoppelung der Zeit!

t: Zeit f: Frequenz U: Spannung P: Leistung E: Energie

Dyn. Frequenzänderung (DFS)

Halbierung der Taktfrequenz
$\Rightarrow$ Verdoppelung der Ausführungszeit:

$$E_{total,f} = P_{total,f} \cdot t = t \cdot c \cdot f + t \cdot P_{other}$$

und

$$E_{total,\frac{f}{2}} = 2t \cdot P_{total,\frac{f}{2}} = 2t \cdot c \cdot \frac{f}{2} + 2t \cdot P_{other}$$

Der Energieverbrauch kann sich unter Umständen erhöhen. (Das ist aber nicht immer so.)

t: Zeit   f: Frequenz   U: Spannung   P: Leistung   E: Energie

Dyn. Spannungsregulierung (DVS)

Leistungsaufnahme (genauer):

$$P_{total,f} = P_{CPU,f} + P_{other} = c \cdot f \cdot U^2 + P_{other}.$$

Falls Spannung proportional zur Frequenz ist:

$$P_{total,\frac{f}{2}} = c \cdot \frac{f}{2} \cdot (\frac{U}{2})^2 + P_{other}.$$

Energiebedarf mit DVS:

$$E_{total,\frac{f}{2}} = t \cdot \frac{P_{CPU,f}}{4} + 2t \cdot P_{other}$$

t: Zeit   f: Frequenz   U: Spannung   P: Leistung   E: Energie

Auswirkungen von DFS

Messungen auf HP iPAQ H5400

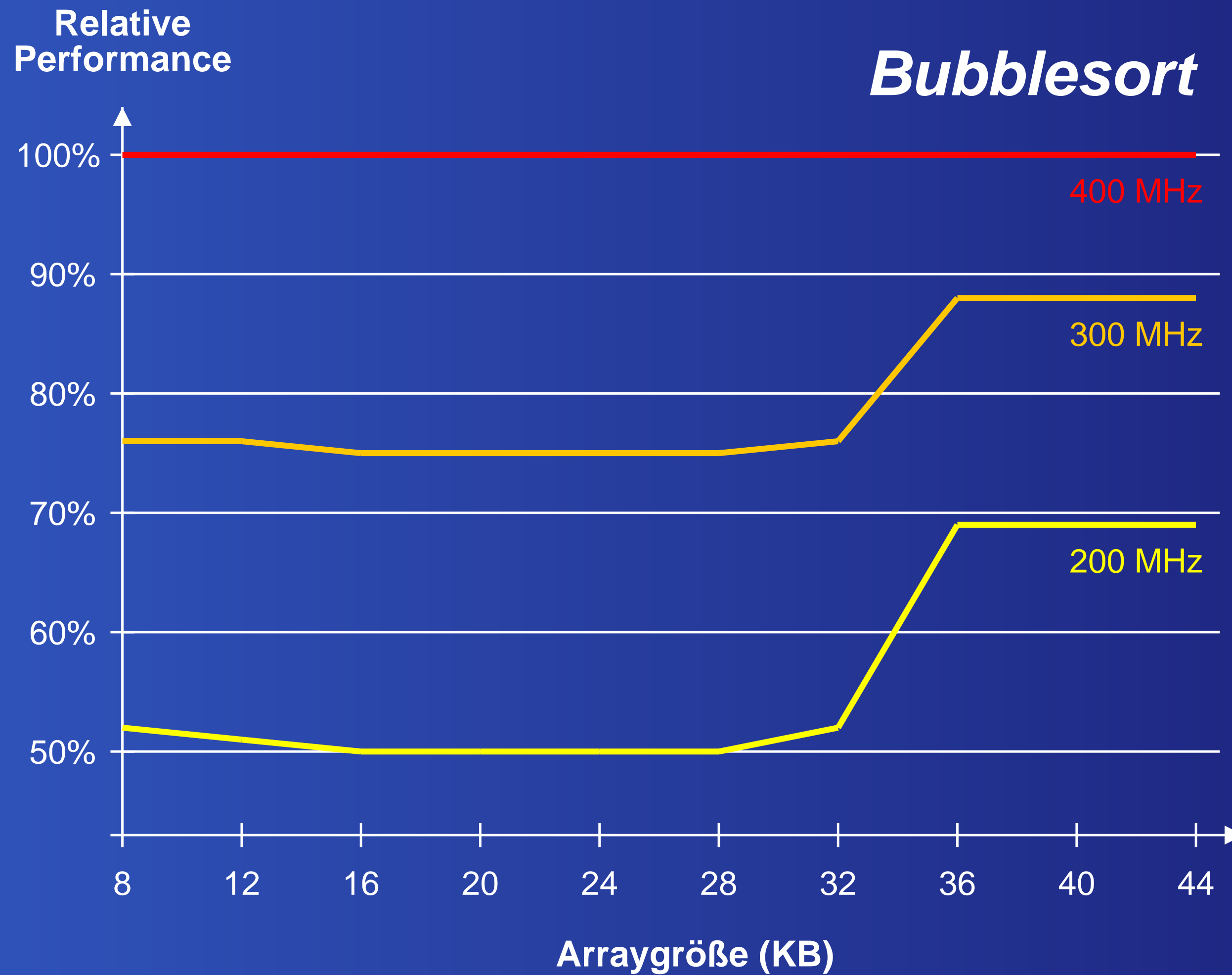
Auswirkungen von DFS

Bedeutet Halbierung der Frequenz immer Verdoppelung der Ausführungszeit?

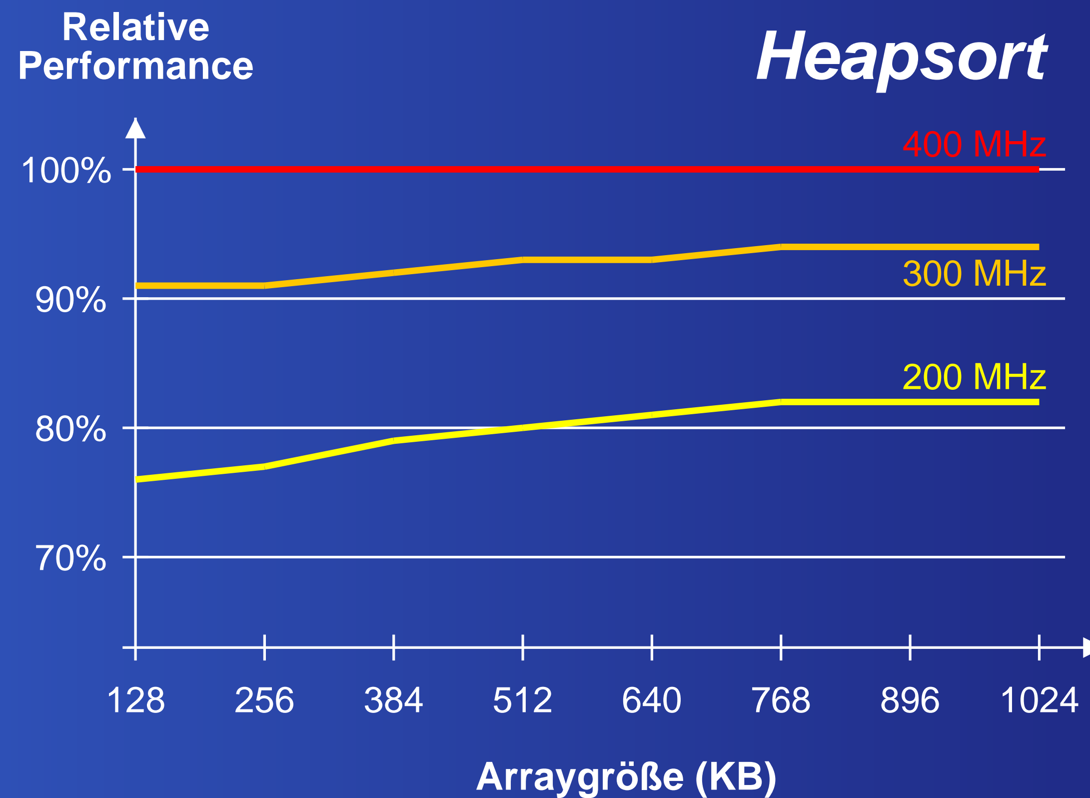
Drei Standard-Algorithmen mit ganz unterschiedlichem Hauptspeicherverhalten:

- Bubblesort (lineare Bearbeitung der Daten)
- Heapsort (wilde Sprünge durch den Speicher)
- Mergesort (Divide & Conquer)

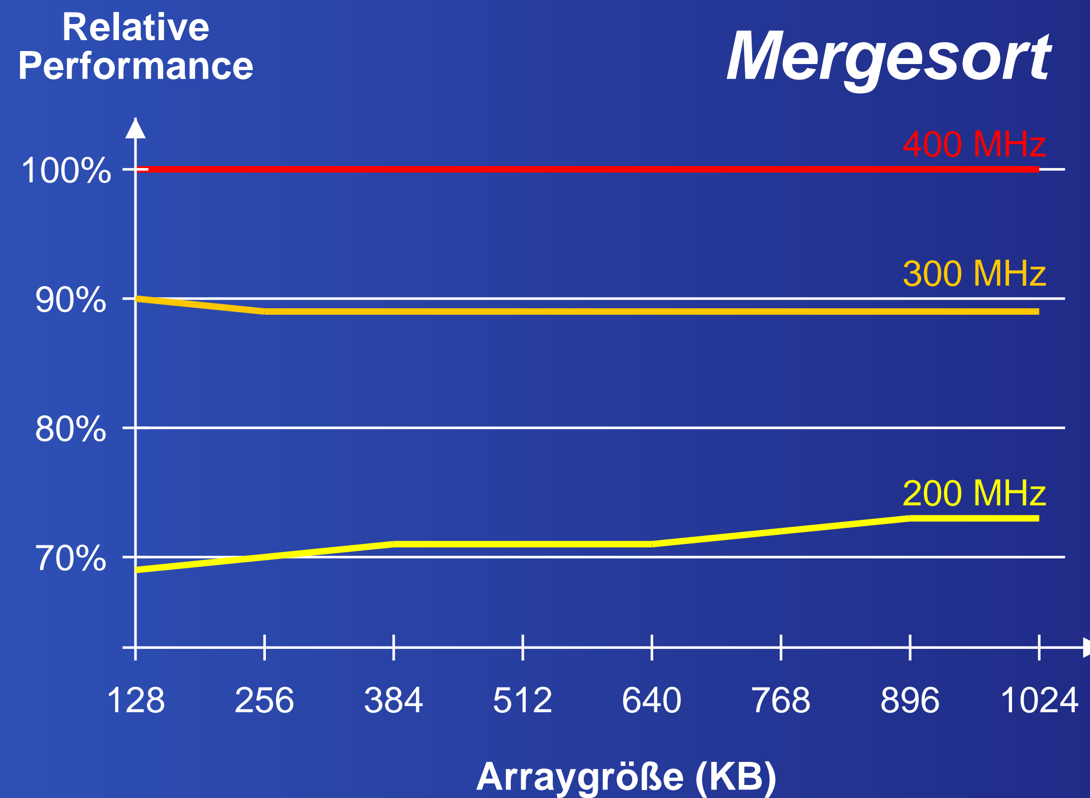
Auswirkungen von DFS



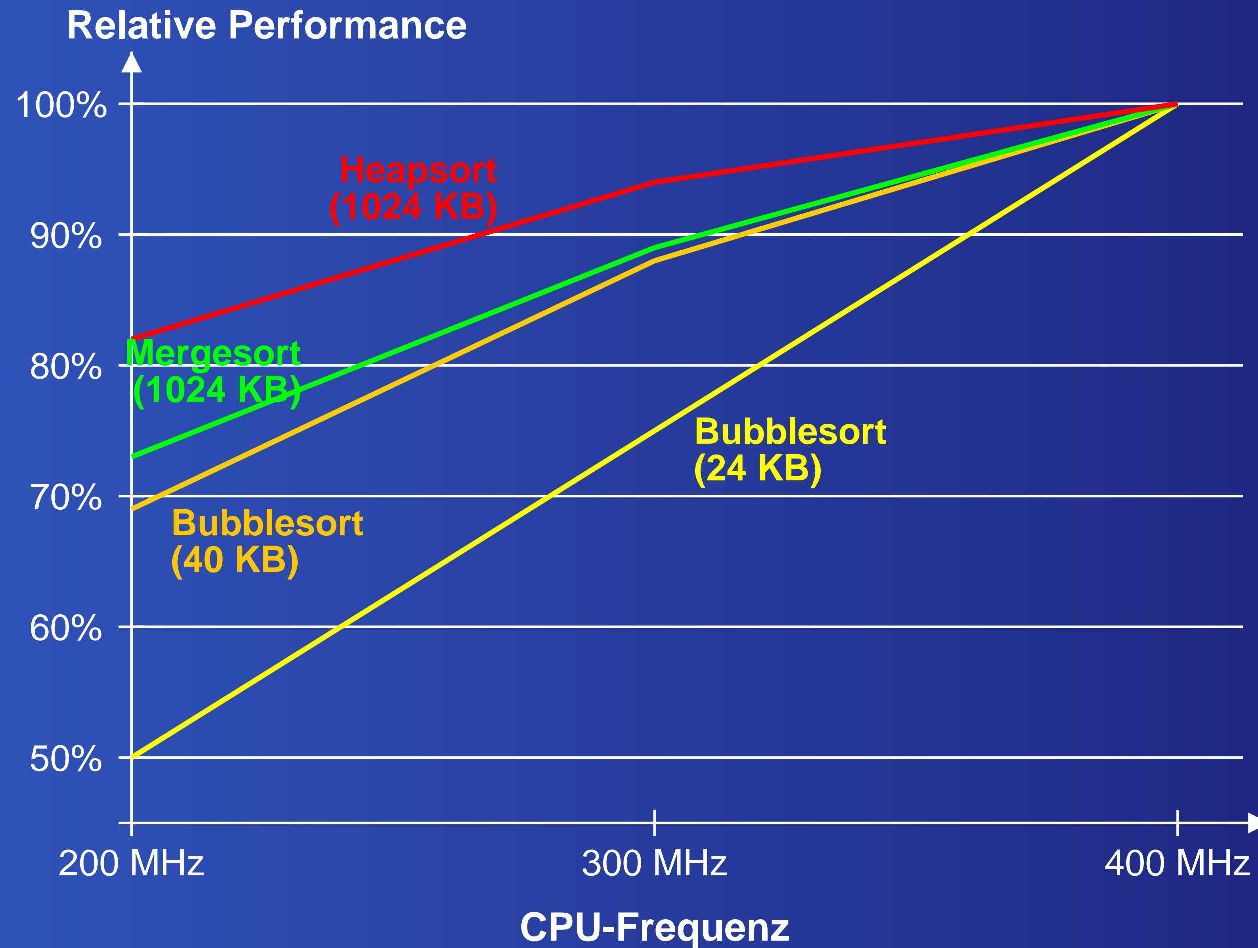
Auswirkungen von DFS



Auswirkungen von DFS



Auswirkungen von DFS



Auswirkungen von DFS

Halbierung der CPU-Frequenz bewirkt keinesfalls immer eine Verdoppelung der Ausführungszeit.

- Energieeinsparungen durch DFS möglich
- In Kombination mit DVS noch besser

DFS/DVS: Wie und wann?

DVS-Algorithmen bisher

Funktionierende DVS-Implementierungen sind bisher

- im Betriebssystem-Kern oder
- im Compiler.

Kernelhacker: „Muss im Kern sein!”

Compilerbauer: „Muss im Compiler sein!”

DVS-Algorithmen bisher

Beides hat Vor- und Nachteile:

Vorteile Kernel	Vorteile Compiler
Frequenzumschaltung erfolgt im Kernel Unterstützung durch Ereigniszähler parallele Prozesse möglich	Vorhersagen durch Programmanalyse

Warum keine Compiler-Kernel-Kooperation?

Übersicht

- Motivation
- Frequenz- und Spannungsskalierung
- Struktur des kooperativen Algorithmus
- Betriebssystem-Kern
- Compiler-Unterstützung
- Energie-Messungen
- Zusammenfassung, Ausblick

Compiler-Kernel-Kooperation

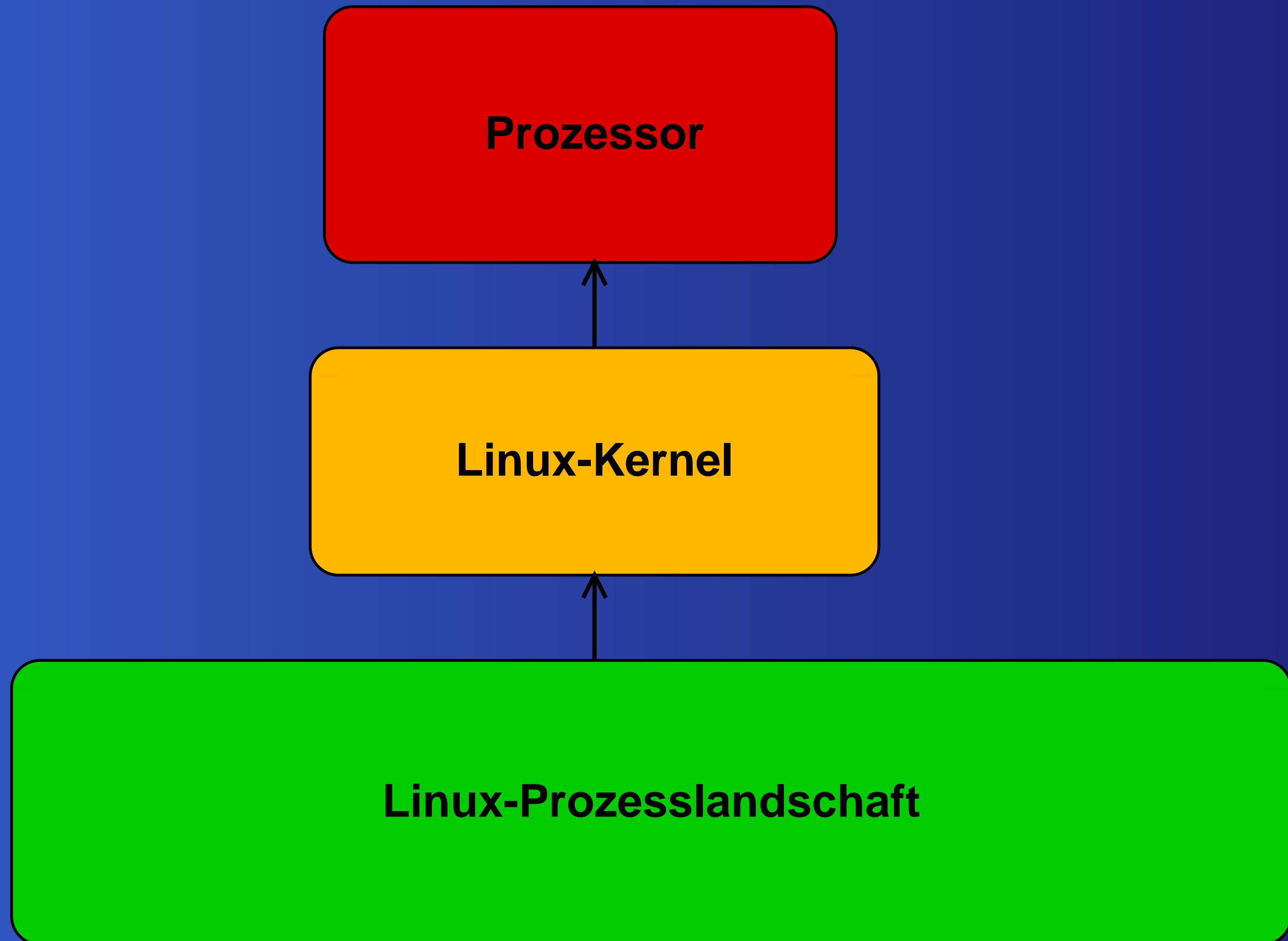
- Die wesentlichen Teile des DVS-Algorithmus sind sinnvollerweise im Betriebssystem-Kern
  - Ereigniszähler
  - globale Sicht
- Der Compiler wird so verändert, dass die erzeugten Programme dem Kernel zur Laufzeit Hilfestellungen geben.
  - Bestehende Programme werden nicht verändert!
  - Hilfestellungen sind sehr allgemein.

Compiler-Kernel-Kooperation

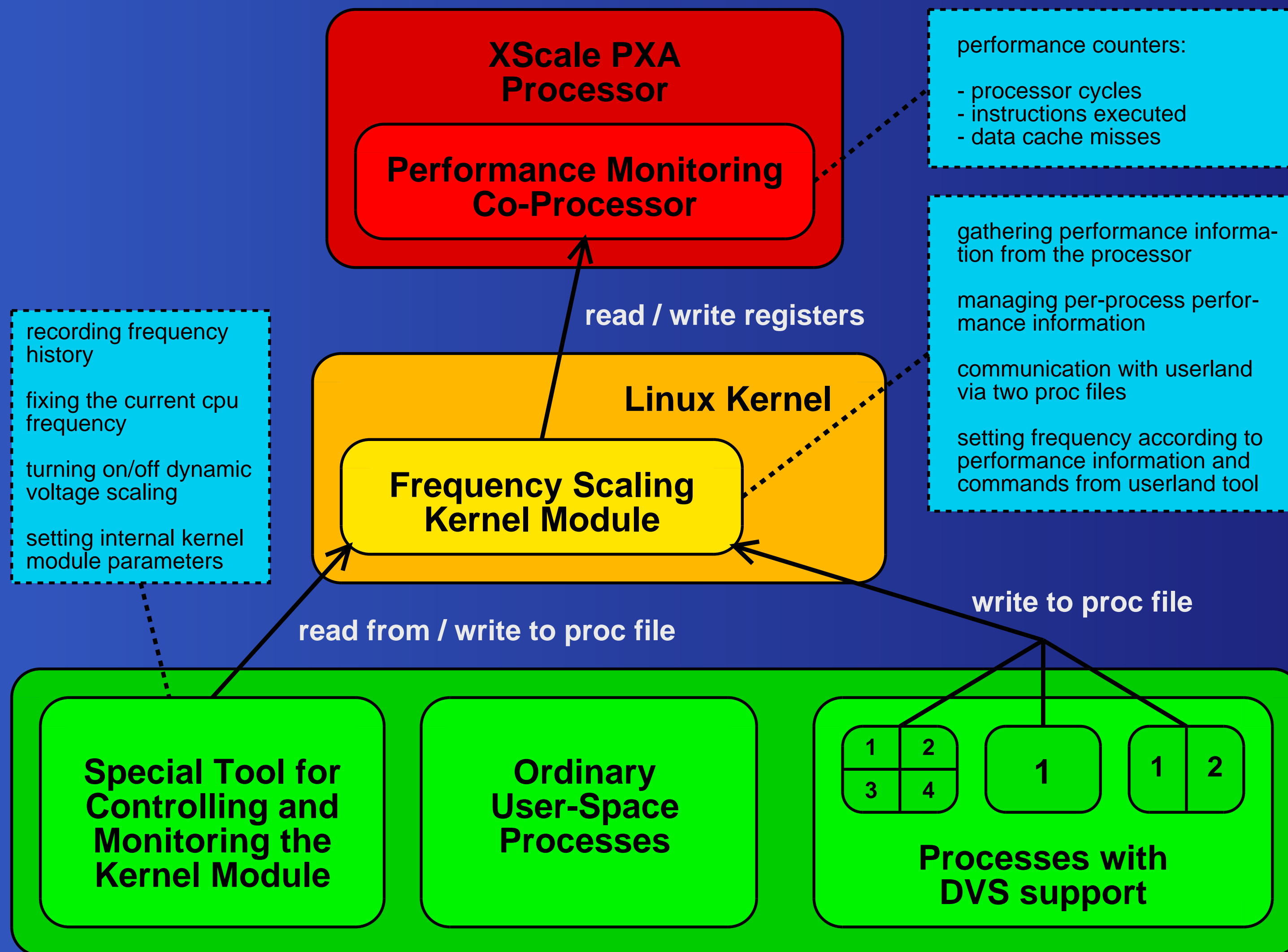
Welche Informationen erhält der Kern zur Laufzeit eines Programms?

- Nachrichten über das Betreten/Verlassen von bestimmten Programmregionen
- Mit einer Region verbundene Echtzeitbedingungen („Welche Verzögerung ist erlaubt?")

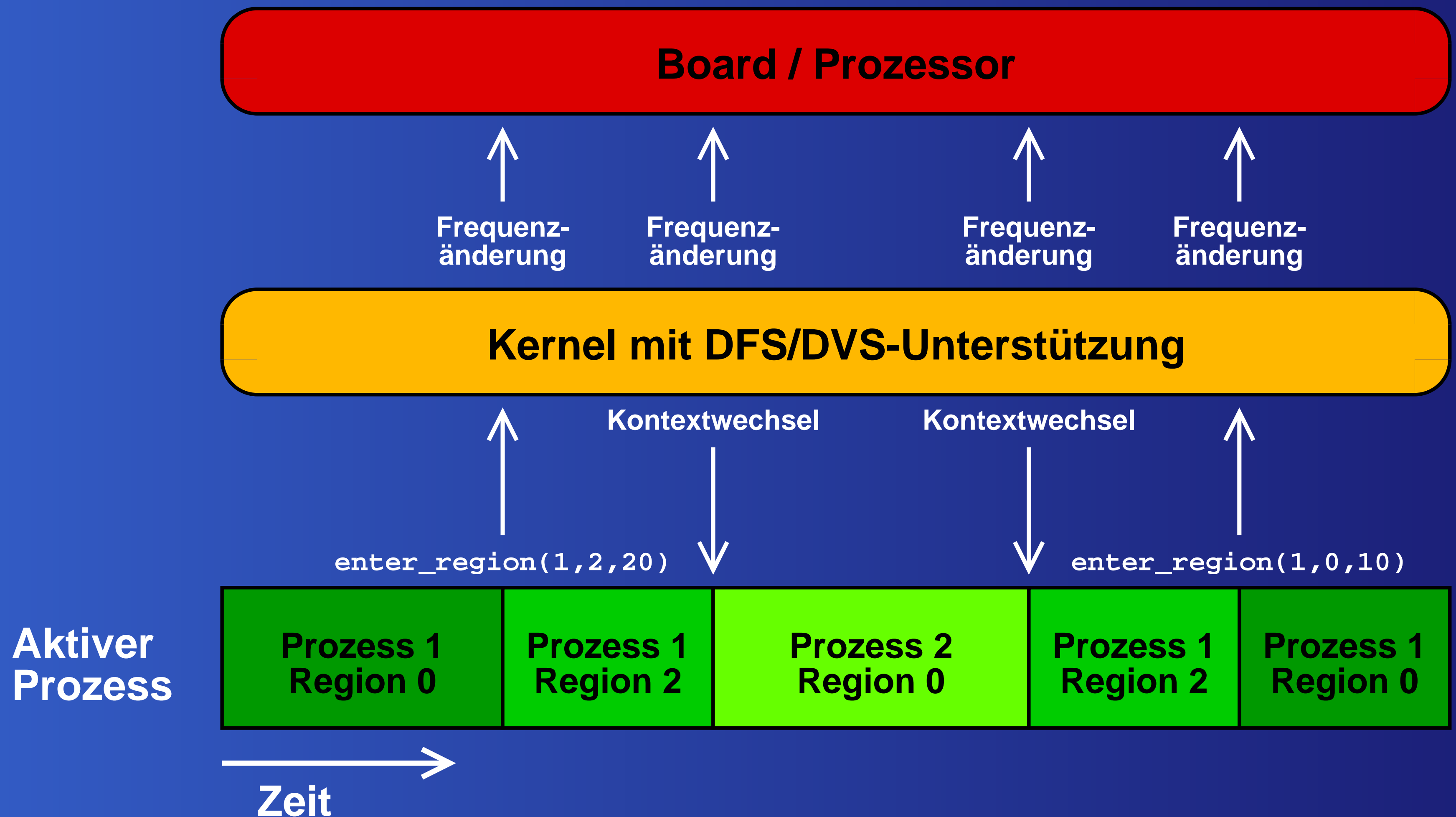
Compiler-Kernel-Kooperation



Compiler-Kernel-Kooperation



Compiler-Kernel-Kooperation



Übersicht

- Motivation
- Frequenz- und Spannungsskalierung
- Struktur des kooperativen Algorithmus
- Betriebssystem-Kern
- Compiler-Unterstützung
- Energie-Messungen
- Zusammenfassung, Ausblick

Der Kernel-Algorithmus

Grundidee:

- Der Kernel wird mit jedem System-Takt aufgerufen und ermittelt
  - ausgeführte Instruktionen (akt. Prozess)
  - aufgetretene Cache-Misses (akt. Prozess)
- Für jeden Prozess werden Frequenz und Spannung so niedrig gewählt, dass die festgelegten Echtzeitbedingungen gerade noch eingehalten werden.

Der Kernel-Algorithmus

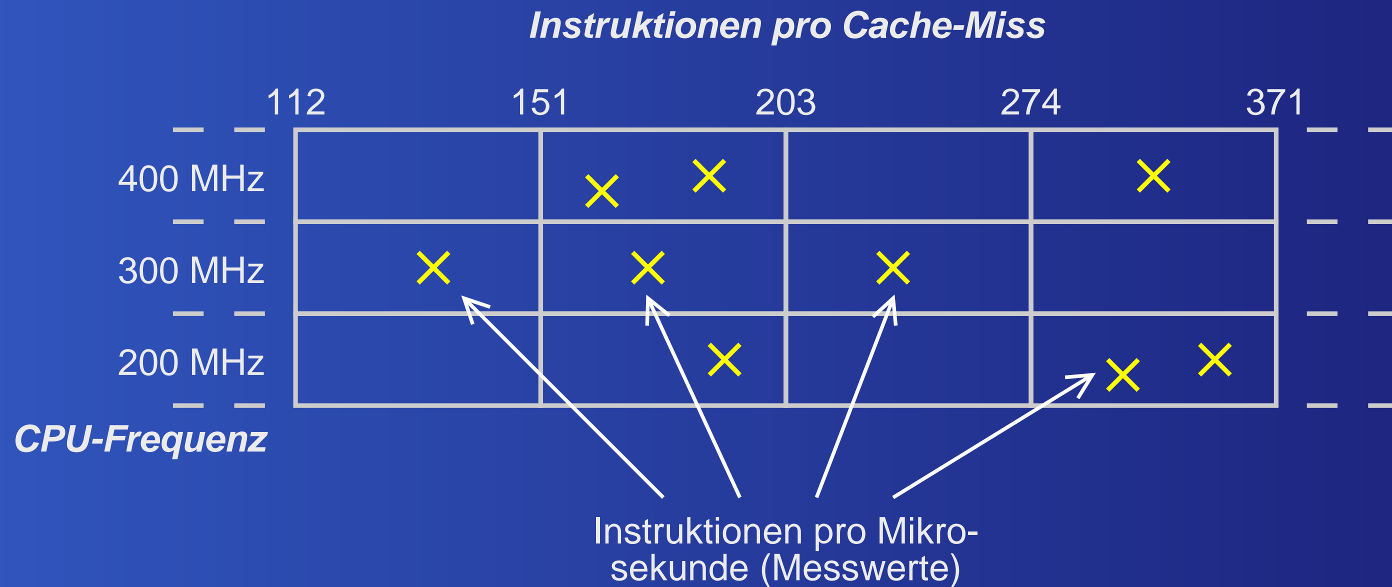
Wie kann der Kern die Auswirkungen einer Frequenzänderung abschätzen?

Approximieren einer Funktion

$$f : \left(\frac{\mathrm{Instruktionen}}{\mathrm{Cache} - \mathrm{Miss}}, \mathrm{Frequenz}\right) \mapsto \mathrm{rel.\ Performanz}$$

durch Aufbau einer Verzögerungstheorie, die auf alten Erfahrungen basiert.

Der Kernel-Algorithmus



Sind in verschiedenen Zellen der selben Spalte Messwerte vorhanden, so kann der Effekt einer Frequenzänderung vorausgesagt werden.

Übersicht

- Motivation
- Frequenz- und Spannungsskalierung
- Struktur des kooperativen Algorithmus
- Betriebssystem-Kern
- Compiler-Unterstützung
- Energie-Messungen
- Zusammenfassung, Ausblick

Compiler-Unterstützung

Grundidee:

Auswahl bestimmter Programmregionen bei der Übersetzung; Einfügen spezieller Funktionsaufrufe zur Kommunikation mit dem Kernel.

- Was ist eine *Programmregion*?
- Nach welchen Kriterien erfolgt die Auswahl? (Der Kernel wird nur über ausgewählte Regionen informiert.)

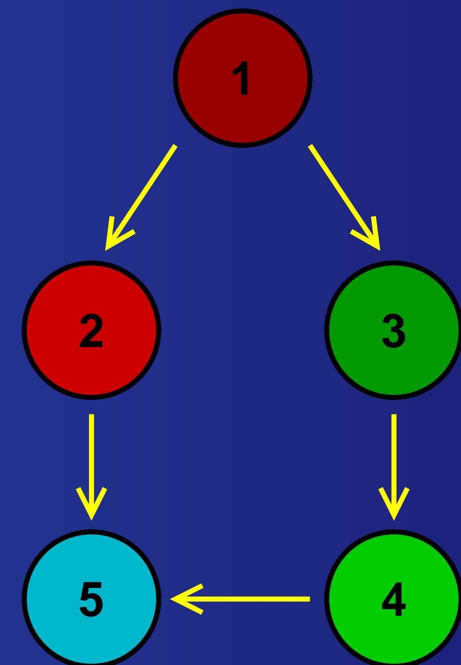
Compiler-Unterstützung

Jede Schleife ist eine Region, jede Funktion ist eine Region:

```
void heapsort(int *arr, int size) {
  buildheap(arr, 0, n);

  while (--n > 0) {
    tmp = arr[0];
    arr[0] = arr[n];
    arr[n] = tmp;
    reheap(arr, 0, n);
  }
}
```

```
void buildheap(int *arr, int root, int size) {

  for (int i = size - 1; i >= root; i--)
    reheap(arr, index, size - index + 1);
}
```

```
void reheap(int *arr, int root, int size) {
  // do something exciting...
}
```



Regionen-Graph

Compiler-Unterstützung

Die den Regionen zugeordneten Knoten bilden einen gerichteten, azyklischen Graphen (DAG).

Was ist bei Rekursionen (direkt oder indirekt)?

$\Rightarrow$ Zyklen verschmelzen, bis DAG entstanden ist.

Wenn der Graph zyklenfrei ist, entspricht jeder Knoten genau einer Programmregion.

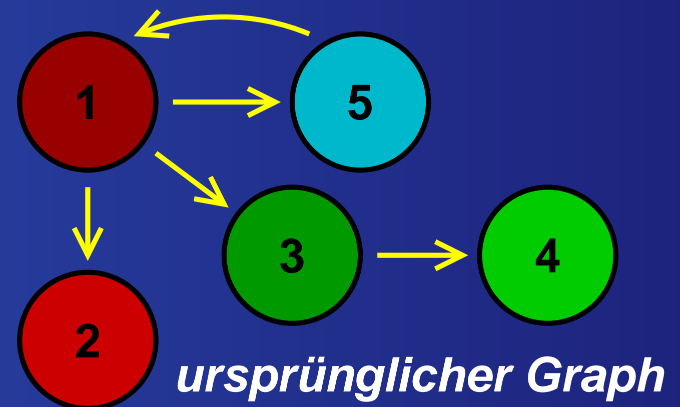
Compiler-Unterstützung

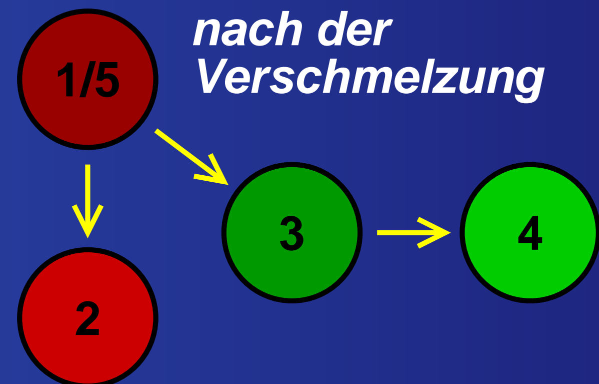
Rekursionsbehandlung

```
void foo(int i, int cnt) {
  while (i % 2 == 0) {
    i /= 2;
    cnt++;
  }

  if (i < 2)
    foobar(cnt);
  else
    bar(i, cnt + 1);
}
```

```
void bar(int i, int cnt) {
  foo(3 * i + 1, cnt + 1);
}
```

```
void foobar(int cnt) {

  for (int i = 0; i < cnt; i++)
    printf("cnt = %i\n", cnt);

}
```





Compiler-Unterstützung

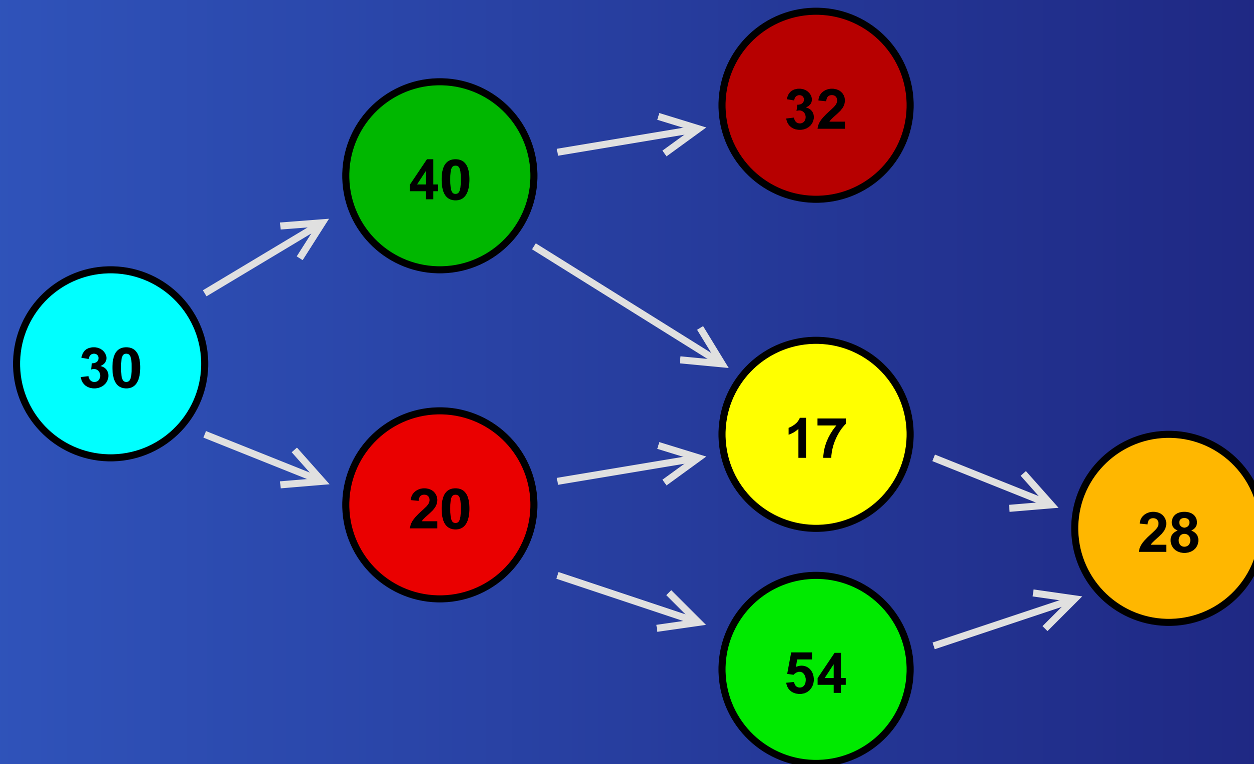
Liegt nach der Verschmelzung ein DAG vor, so kann der Graph topologisch sortiert und von unten nach oben durchlaufen werden.

Jedem Knoten v werden Kosten C_v zugeordnet (Anzahl der Instruktionen im Knoten).

Es werden Knoten ausgewählt, deren akkumulierte Kosten zwischen C_{min} und C_{max} liegen.

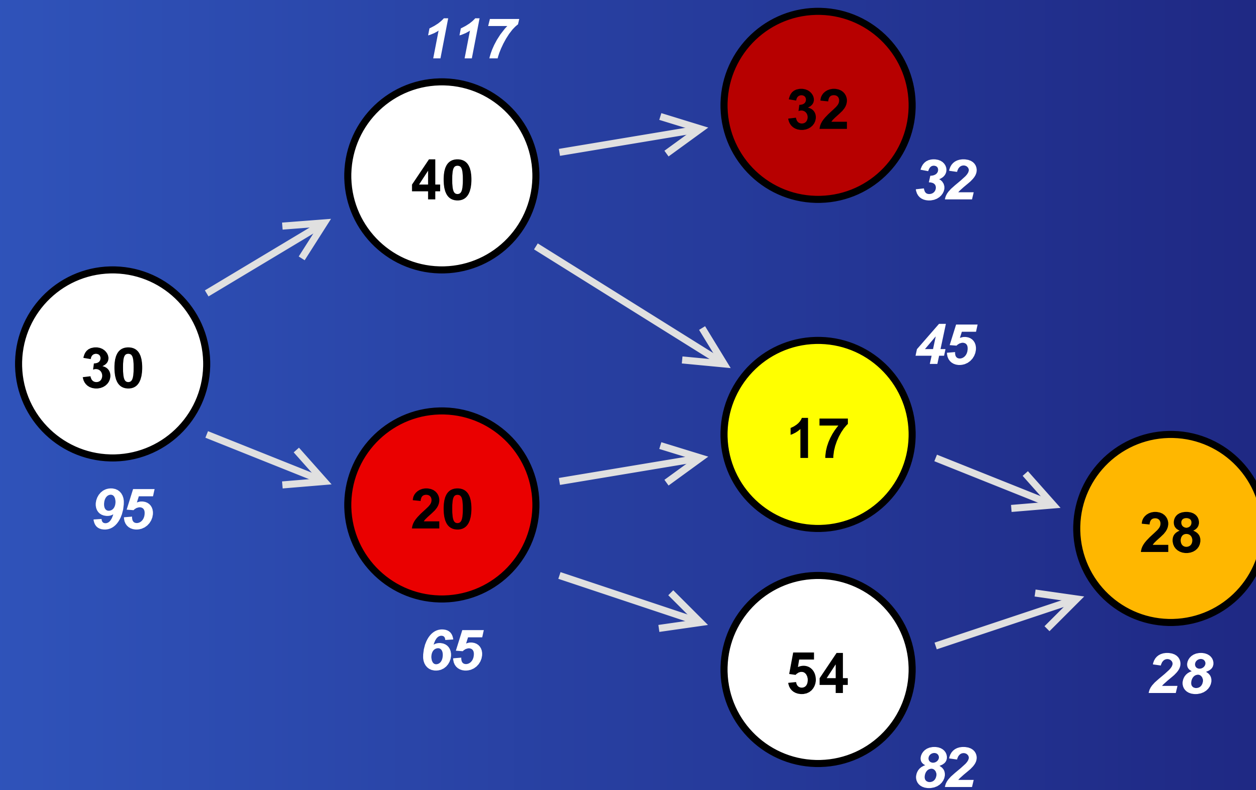
Compiler-Unterstützung

Regionenselektion mit $C_{min} = 80$, $C_{max} = 160$:



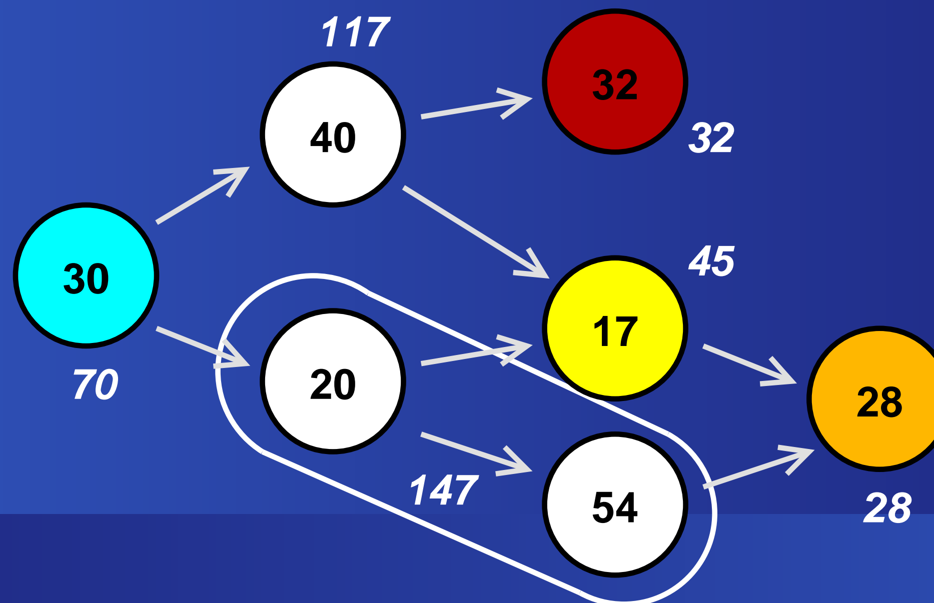
Compiler-Unterstützung

Regionenselektion mit $C_{min} = 80$, $C_{max} = 160$:

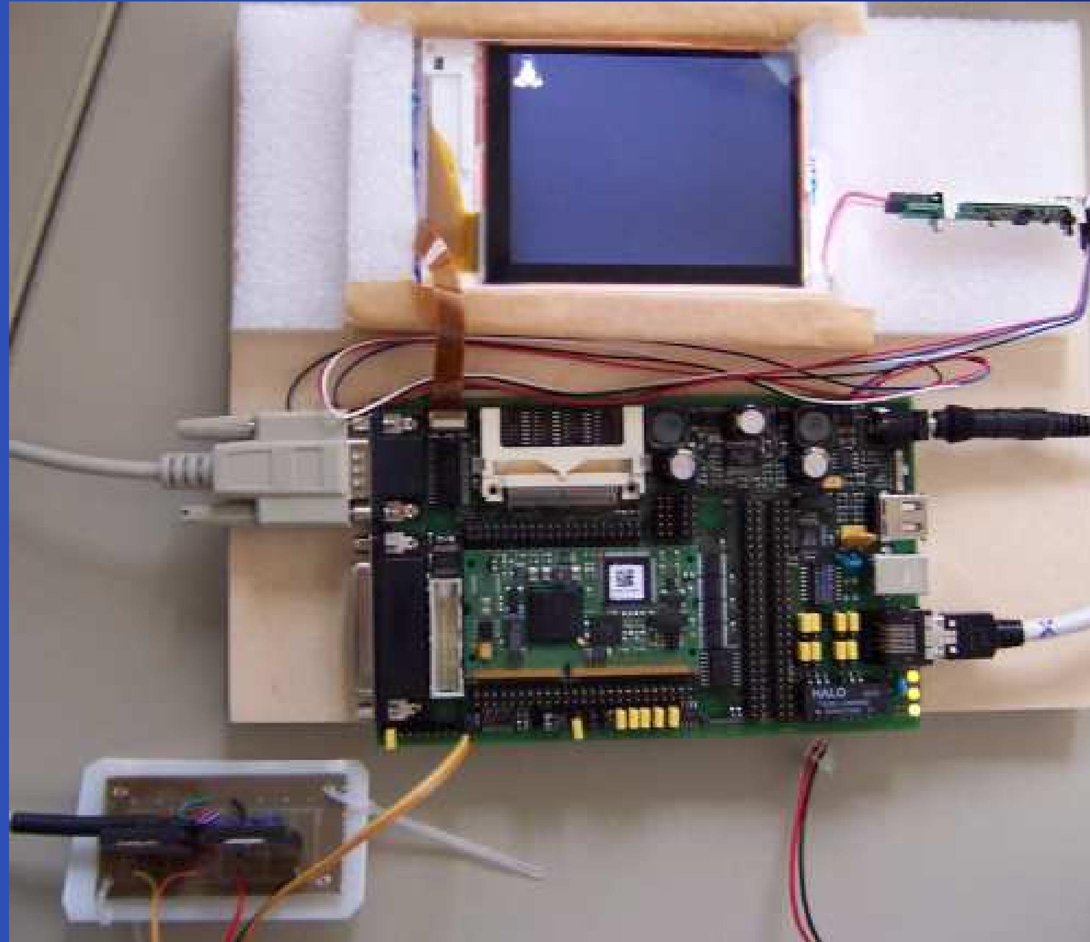


Compiler-Unterstützung

Zwei Regionen können verschmolzen werden, wenn ihre Referenzdichte (rel. Häufigkeit von Heap-Zugriffen) ungefähr gleich ist und ihre Gesamtgröße unterhalb der oberen Schranke C_{max} liegt.



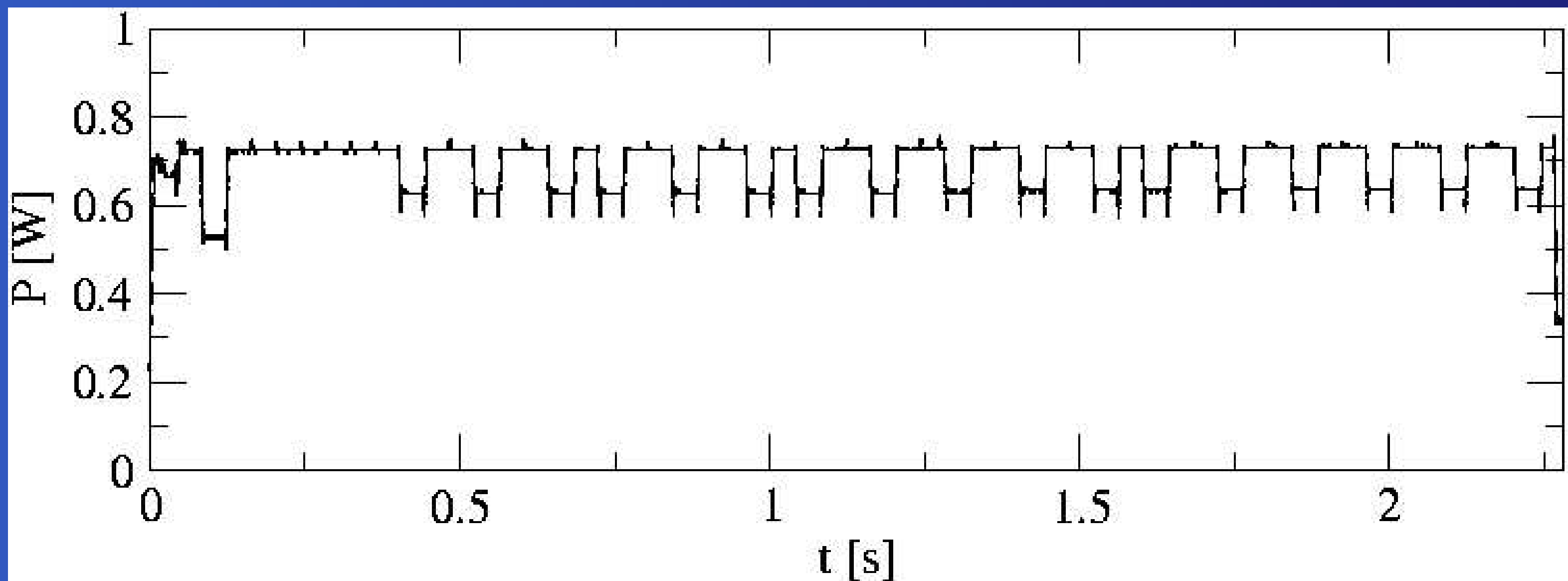
Messungen



Messungen auf Triton-LP

Messungen: Leistungsaufnahme

Beispiel: Leistungsaufnahme bei Bubblesort, nur DFS (Spannung konstant)



Messungen: Terminologie

Energieverbrauch bei Frequenz f_0:

$$E_{total,f_0} = t_{f_0} \cdot P_{active,f_0}$$

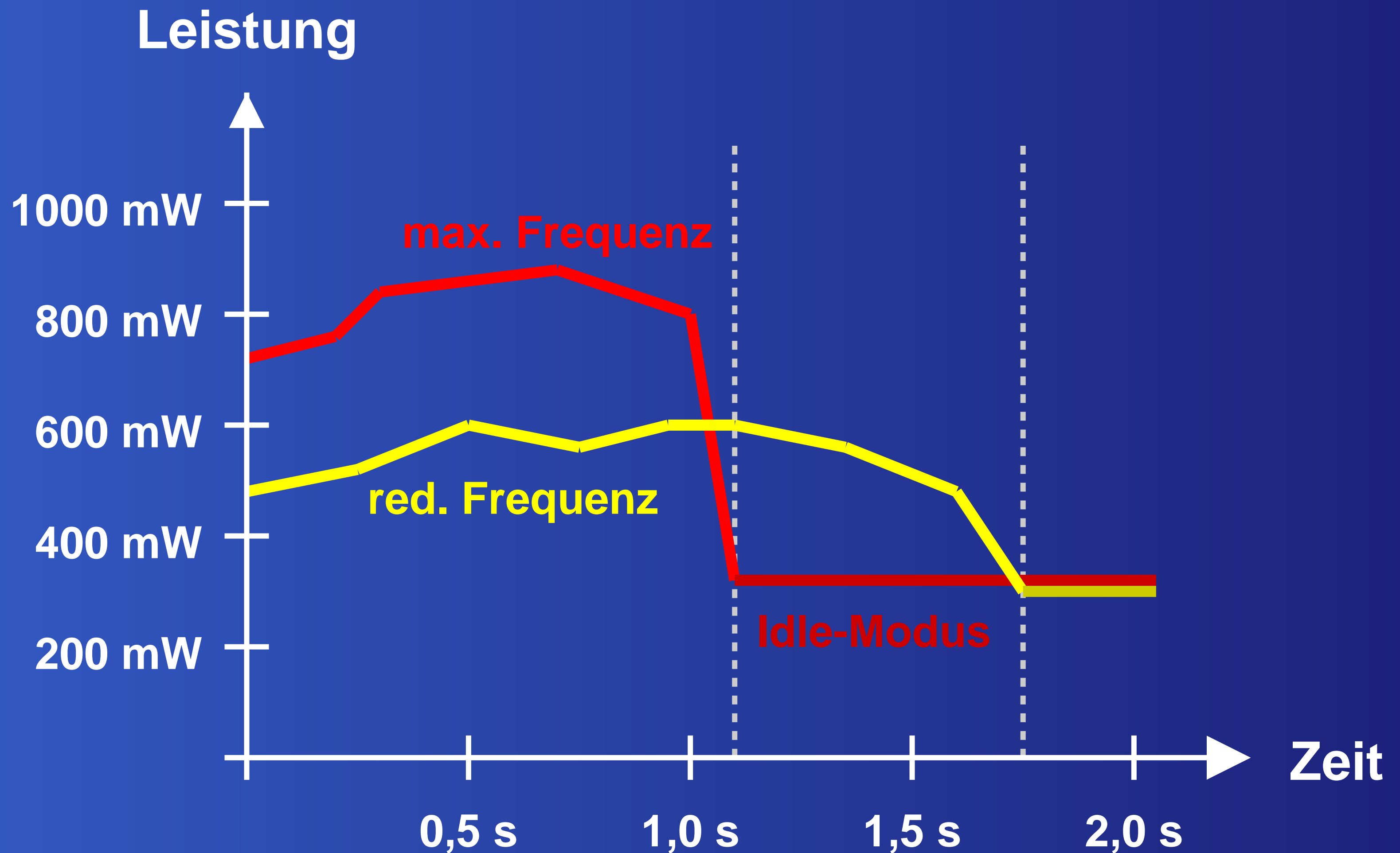
Energieverbrauch bei Frequenz f_{max}:

$$E_{total,f_{max}} = t_{f_{max}} \cdot P_{active,f_{max}} + (t_{f_0} - t_{f_{max}}) \cdot P_{idle,f_{min}}$$

- pro-forma-Energiebedarf: $\frac{E_{active,f_0}}{E_{active,f_{max}}}$
- effektiver Energiebedarf: $\frac{E_{total,f_0}}{E_{total,f_{max}}}$

t: Zeit f: Frequenz U: Spannung P: Leistung E: Energie

Messungen: Terminologie



Energie bei konstanten Frequenzen

Bubblesort (Array-Größe: 40 KB)

Frequenz	Spann.	Zeit	Energie	pro-f.	effekti
400 MHz	1,3 V	17,23 s (100%)	13,72 J	100%	100%
300 MHz	1,3 V	21,30 s (124%)	14,86 J	108%	99%
	1,1 V	21,31 s (124%)	12,90 J	94%	86%
200 MHz	1,3 V	30,03 s (174%)	17,65 J	129%	101%
	1,0 V	30,04 s (174%)	14,43 J	105%	82%

Energie bei konstanten Frequenzen

Heapsort (Array-Größe: 1600 KB)

Frequenz	Spann.	Zeit	Energie	pro-f.	effektiv
400 MHz	1,3 V	5,51 s (100%)	4,27 J	100%	100%
300 MHz	1,3 V	6,40 s (116%)	4,41 J	103%	97%
	1,1 V	6,40 s (116%)	3,89 J	91%	86%
200 MHz	1,3 V	8,15 s (148%)	4,85 J	114%	96%
	1,0 V	8,12 s (148%)	4,01 J	94%	79%

Energie bei konstanten Frequenzen

Folgerungen:

- DFS allein bringt kaum Einsparungen.
- Mit DVS sind Einparungen von über 20% möglich.
- Die Höhe der Einsparungen hängt von der Cache-Effizienz des laufenden Prozesses ab.

Kernel-Algorithmus (DFS/DVS)

Bubblesort (40 KB)

Zielslowdown	gemessen	Zeit	Energie	pro-f.	effektiv
0,0%	0,0%	17,23 s	13,72 J	100%	100%
12,0%	10,7%	19,08 s	13,65 J	99%	96%
25,0%	22,0%	21,02 s	13,65 J	97%	89%

$\Rightarrow$ effektive Energieeinsparung von 11%

Kernel-Algorithmus (DFS/DVS)

Heapsort (1600 KB)

Zielslowdown	gemessen	Zeit	Energie	pro-f.	effektiv
0,0%	0,0%	5,51 s	4,27 J	100%	100%
12,0%	12,1%	6,18 s	4,09 J	96%	91%
25,0%	31,5%	7,25 s	4,03 J	94%	84%

$\Rightarrow$ noch mehr: 16% (aber Verzögerung!)

Kernel-Algorithmus (DFS/DVS)

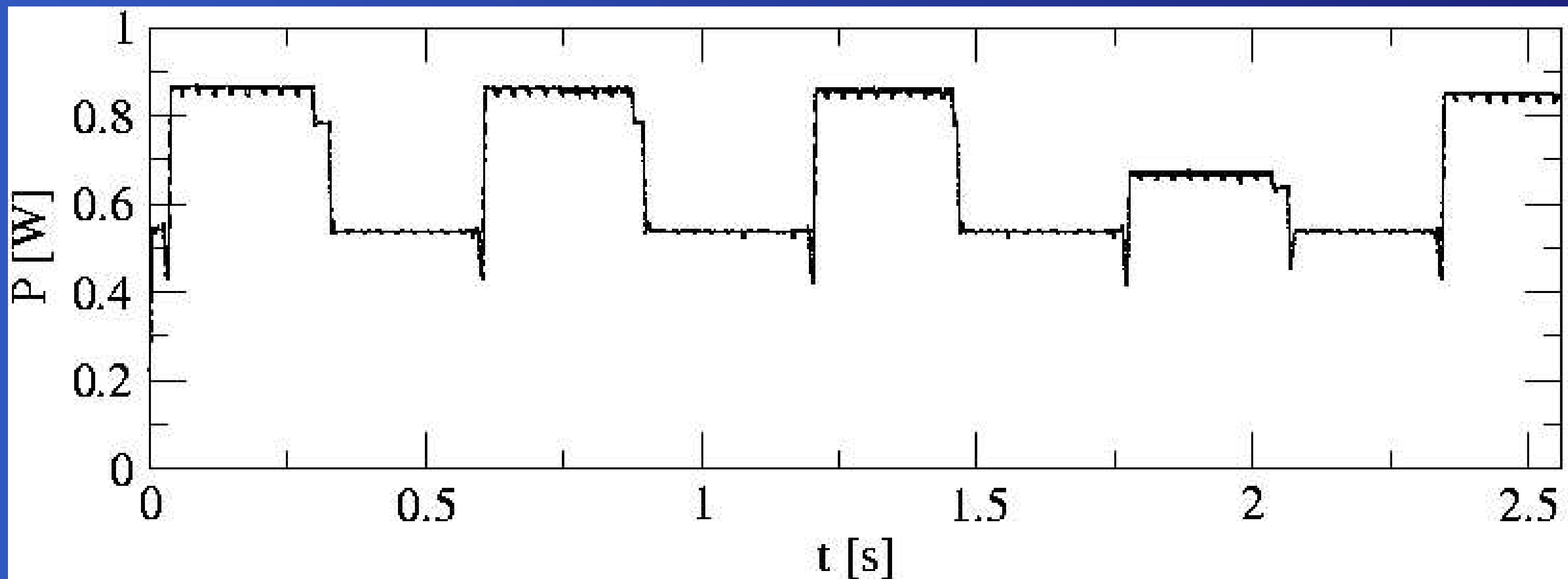
Matrix-Multiplikation (256 x 256)

Zielslowdown	gemessen	Zeit	Energie	pro-f.	effektiv
0,0%	0,0%	16,44 s	11,54 J	100%	100%
12,0%	13,8%	18,72 s	9,70 J	84%	79%
25,0%	18,8%	19,53 s	9,36 J	81%	75%

$\Rightarrow$ noch viel mehr: 25%

Parallele Prozesse

Bubblesort (10%), Heapsort (25%) parallel



tatsächliche Verzögerungsraten:
13,8% und 23,3%

Compiler-Kernel-Kooperation

Realistische Anwendung: Bildbearbeitung

3 Regionen: Lesen, Verkleinern, Schreiben

	Zeit	Energie	pro-f.	effekti
max. Frequenz	7,48 s (100%)	5,56 J	100,0%	100,0%
Kernel	8,98 s (120%)	5,59 J	100,5%	93,0%
Kernel+Compiler	8,99 s (120%)	5,61 J	100,9%	93,3%

$\Rightarrow$ keine messbare (spürbare!) Verbesserung

Compiler-Kernel-Kooperation

Realistische Anwendung: mpg123 (MP3-Spieler)

2 bzw. 3 Regionen

	Zeit	**Energie**	**pro-f.**	**effek**
max. Frequenz	62,3 s (100%)	47,3 J	100,0%	100,0
Kernel	78,5 s (126%)	47,6 J	100,5%	91,2
Kernel+Comp. (2)	78,9 s (127%)	47,5 J	100,4%	90,8
Kernel+Comp. (3)	87,7 s (141%)	52,8 J	111,5%	96,0

Compiler-Kernel-Kooperation

Künstliche Anwendung:
Bubblesort/Heapsort (abwechselnd)

3 Regionen: Bubblesort, Heapsort, Schleife

	Zeit	**Energie**	**pro-f.**	**effekti**
max. Frequenz	13,23 s (100%)	10,52 J	100,0%	100,0
Kernel	16,04 s (121%)	9,48 J	90,1%	83,4
Kernel+Comp.	16,08 s (122%)	9,30 J	88,4%	81,8

Übersicht

- Motivation
- Frequenz- und Spannungsskalierung
- Struktur des kooperativen Algorithmus
- Betriebssystem-Kern
- Compiler-Unterstützung
- Energie-Messungen
- Zusammenfassung, Ausblick

Zusammenfassung

Anfangsfrage

DVS ist gut, aber wo soll man's machen? Im Compiler oder im Kernel?

$\Rightarrow$ Compiler-Kernel-Kooperation

zentrale Bestandteile im Kernel, Compiler bietet Unterstützung

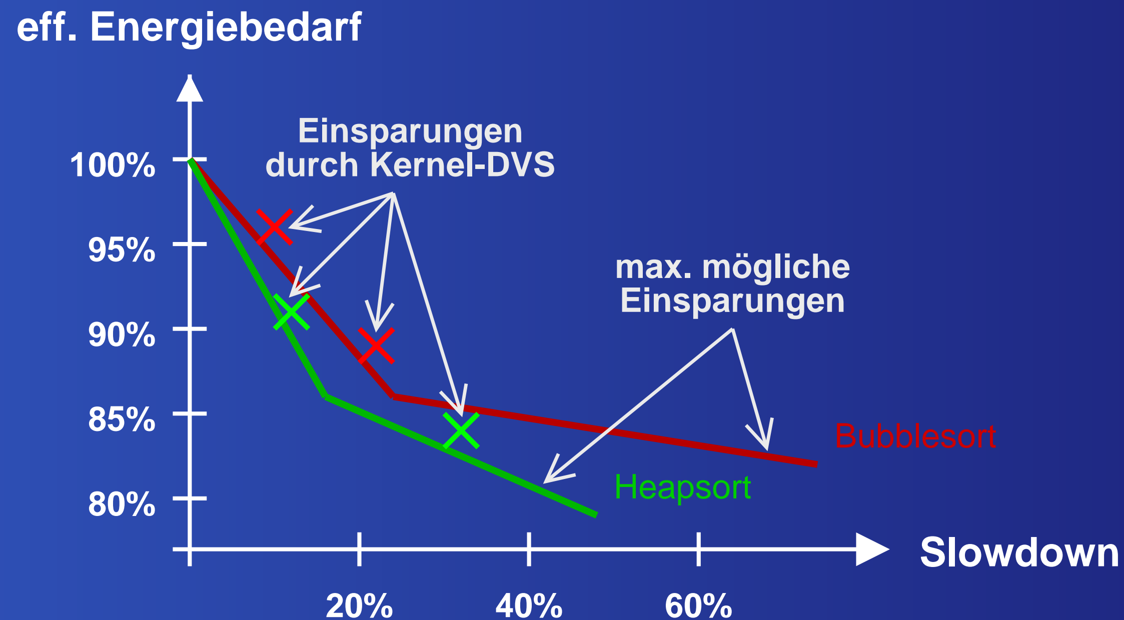
Zusammenfassung

Das Kernel-Modul

- erhält Infos über Echtzeitbedingungen,
- sammelt Performance-Daten,
- reagiert auf Verhaltensänderungen von Applikationen,
- steuert CPU-Frequenz und -Spannung mit dem Ziel maximaler Energie-Einsparung

Zusammenfassung

- Energie-Einsparungen durch Kernel-DVS liegen nah am theoretischen Optimum.



- Parallele Proz. werden adäquat behandelt.

Zusammenfassung

Der Compiler

- analysiert Programme vor der Ausführung,
- selektiert bestimmte Programmregionen,
- informiert den Kern zur Laufzeit über Regionswechsel.

Durch die Compiler-Unterstützung ist der Kernel in der Lage, Programmverhalten vorauszusehen.

Zusammenfassung

Die Compiler-Unterstützung bringt nur in Ausnahmefällen Vorteile.

- realistische, homogene Applikationen: keine Verbesserung
- manchmal: Verschlechterung durch Compiler
- konstruierte Extremfälle: 2% zusätzliche Energieeinsparung

Ausblick

Was bleibt zu tun?

- Verbesserung der Performance-Vorhersage (Echtzeitbedigungen *müssen* eingehalten werden)
- andere Regeln zur Selektion von Regionen
- Portierung auf andere Architekturen (AMD PowerNow, Intel SpeedStep, ...)
- Auswirkungen von DFS/DVS auf Batterie-Laufzeit

Vorbei

Vielen Dank!