

# GRIT: Forschungsansätze

David Büttner

**Seminar: Grüne Informationstechnologie**

Prof. Dr. T. Ludwig  
SS 08

Universität Heidelberg

7. August 2008

## 1 Einführung

## 2 Forschungsansätze - CPU

- DVFS
- Compiler Algorithmus
- Energieeinsparung bei MPI-Programmen
- Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen

## 3 Forschungsansatz - Netzwerk

## 4 Zusammenfassung

## 1 Einführung

### 2 Forschungsansätze - CPU

- DVFS
- Compiler Algorithmus
- Energieeinsparung bei MPI-Programmen
- Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen

### 3 Forschungsansatz - Netzwerk

### 4 Zusammenfassung

# Energiesparen und Forschung

Es gibt viele Möglichkeiten und Ansätze in der Informationstechnologie Energie zu sparen.

Hauptgründe der Umsetzung sind im

- Mobile-Computing: Batterielaufzeit
- High-Performance-Computing (HPC): Kosten und Ausfallsicherheit

## 1 Einführung

## 2 Forschungsansätze - CPU

- DVFS
- Compiler Algorithmus
- Energieeinsparung bei MPI-Programmen
- Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen

## 3 Forschungsansatz - Netzwerk

## 4 Zusammenfassung

## 1 Einführung

## 2 Forschungsansätze - CPU

- DVFS
  - Compiler Algorithmus
  - Energieeinsparung bei MPI-Programmen
  - Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen

## 3 Forschungsansatz - Netzwerk

## 4 Zusammenfassung

# Was ist DVFS?

DVFS steht für Dynamic Voltage and Frequency Scaling.

- Hauptanteil von Energieverbrauch eines CMOS-Schaltkreises skaliert quadratisch mit der Betriebsspannung
- Frage nach optimaler Zuordnung von Frequenz-Spannungs-Paaren zu Programmabschnitten ist NP-schwer  
⇒ Viele Heuristiken werden erstellt
- Neben der CPU müssen auch beachtet werden:
  - Zusatzaufwand durch Frequenzveränderungen
  - Auswirkungen auf andere Hardwarekomponenten

Ein optimaler DVFS Algorithmus nutzt von einem limitierten Satz von Frequenzen für die Optimallösung nur 2 dieser Frequenzen.

# Arten von DVFS Algorithmen

Die Entscheidungen in DVFS Algorithmen können

- online
- offline

getroffen werden.

Es gibt 4 Arten, auf die anhand von unterschiedlichen Informationen Entscheidungen getroffen werden:

- Manuell
- Compiler-Analyse in Kombination mit *profiling*
- MPI-Bibliothek basierte Code-Erweiterungen
- Sich anpassende Laufzeit-Systeme

## 1 Einführung

## 2 Forschungsansätze - CPU

- DVFS
- **Compiler Algorithmus**
- Energieeinsparung bei MPI-Programmen
- Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen

## 3 Forschungsansatz - Netzwerk

## 4 Zusammenfassung

# Quelle

**Titel** The Design, Implementation, and Evaluation of a  
Compiler Algorithm for CPU Energy Reduction

**Autoren** Chung-Hsing Hsu, Ulrich Kremer  
Department of Computer-Science  
Rutgers, The State University of New Jersey

**Erschienen** Proceedings of the ACM SIGPLAN 2003 conference  
on Programming language design and implementation  
San Diego, California, USA - 2003

# Die Idee

## Überblick

- Algorithmus-Idee:
  - 1 Identifiziere Programmregionen mit niedriger CPU-Auslastung durch Speicherzugriffe
  - 2 Verringere CPU-Frequenz und Spannung in diesen Phasen ohne Laufzeit stark zu beeinflussen
- Design durch Minimierungsproblem
- Implementierung und Ergebnisse

# Design

Skalierungspunkte und Skalierungsfaktoren werden off-line bestimmt:

- Erstelle ein Profil des Programms (Schleifen, Funktionsaufrufe, Befehlssequenzen, etc.)
- Löse Minimierungsproblem
- Führe Programm mit gefundenen Einstellungen aus

# Das Minimierungsproblem

Variablen:

- $P$  - Programm
- $R$  - Programmregion
- $f$  - Frequenz
- $f_{max}$  - maximal mögliche Frequenz
- $T(R, f)$  - Zeit für Region  $R$  bei Frequenz  $f$
- $T_{trans}, P_{trans}$  - Overhead der Frequenzveränderung bezogen auf Zeit bzw. Energieverbrauch
- $P_f$  - Verlustleistung bei Frequenz  $f$
- $N(R)$  - Anzahl Ausführung von  $R$
- $r$  - maximal zulässige Ausführungsverzögerung

# Das Minimierungsproblem

- Die vorgestellten Ergebnisse gehen davon aus, dass nur eine Region verlangsamt werden kann
- Es werden aus den gefundenen Regionen  $R$  alle möglichen Kombinationen gebildet und als neue Region zu  $R$  hinzugefügt.
- Enumeration über alle Regionen in  $R$  führt zu optimaler Lösung von je einem Minimierungsproblem
- Wähle beste Region aus und instrumentiere den Code entsprechend

Der allgemeine Fall ist auch möglich:

- Jede Region kann verlangsamt werden
- Das Minimierungsproblem wird dementsprechend schwerer

# Das Minimierungsproblem - 1 Region

## Bestimmung der Frequenz für Region $R$

$$\min_{R, f} P_f T(R, f) + P_{f_{max}} T(P - R, f_{max}) + P_{trans} 2N(R) \quad (1a)$$

s.t.

$$T(R, f) + T(P - R, f_{max}) + T_{trans} 2N(R) \leq (1 + r) T(P, f_{max}) \quad (1b)$$

$$\frac{T(R, f_{max})}{T(P, f_{max})} \geq p \quad (1c)$$

# Das Minimierungsproblem - Allgemein

Zusätzliche Variablen:

- $\theta(R_i, f) = \begin{cases} 1, & R_i \text{ ausgeführt mit Frequenz } f \\ 0, & \text{sonst} \end{cases}$
- $N(R_i, R_j)$  als Anzahl Übergänge von  $R_i$  zu  $R_j$
- $N_{trans} = \sum_{i,j} (N(R_i, R_j) \frac{1}{2} \sum_f |\theta(R_i, f) - \theta(R_j, f)|)$

# Das Minimierungsproblem - Allgemein

## Bestimmung der Frequenzen für Regionen aus $R$

$$\min_{\theta} \sum_{i,f} \theta(R_i, f) P_f T(R_i, f) + P_{trans} N_{trans} \quad (2a)$$

s. t.

$$\sum_{i,f} \theta(R_i, f) T(R_i, f) + T_{trans} N_{trans} \leq (1 + r) T(P, f_{max}) \quad (2b)$$

$$\sum_f \theta(R_i, f) = 1 \quad (2c)$$

# Testumgebung

## Hardware:

- Compaq Presario 715US notebook
- high-performance microprocessor (mobile AMD Athlon 4)  
(unterstützt DVFS)
- hardware data prefetching unit
- von 600MHz/1.15V bis 1200MHz/1.45V mit 7 DVFS Stufen

## Software:

- Linux 2.4.18 Kernel

## Benchmarkwahl:

- SPECfp95
- Grund: Vielfalt an CPU-Nutzungsgrad

# Ergebnisse

- Die Energieeinsparung eines Benchmarks ist negativ proportional zu seinem CPU-Nutzungsgrad  $\beta$ .  
Wobei  $0 < \beta_{cpu} \leq 1$  definiert ist als:

$$\frac{T_f}{T_{f_{max}}} = \beta_{cpu} \frac{f_{max}}{f} + c_0$$

- Compilierzeiten belaufen sich auf Minuten. Zeit steigt mit Anzahl der möglichen Regionen.
- Für die Fälle mit  $\beta \leq 0.3$  sind Energieeinsparungen des Systems bis zu 28% erreicht worden.
- Die Nebenbedingungen (1c) sind vorteilhaft für die Ergebnisse.

## 1 Einführung

## 2 Forschungsansätze - CPU

- DVFS
- Compiler Algorithmus
- **Energieeinsparung bei MPI-Programmen**
- Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen

## 3 Forschungsansatz - Netzwerk

## 4 Zusammenfassung

# Quelle

**Titel** Bounding Energy Consumption in Large-Scale MPI Programs

**Autoren** Barry Rountree et. al.  
University of Georgia  
Athens, GA

**Erschienen** Proceedings of the 2007 ACM/IEEE conference on Supercomputing  
Copyright von 2005, Zitiert schon in Paper von SC05

# Motivation

## Überblick

- Es gibt unterschiedliche Systeme die DVFS nutzen um Energy bei der Ausführung von MPI Programmen zu sparen
- Diesen fehlen zwei Dinge:
  - Die maximal mögliche Energieeinsparung zu einem gegebenen Zeitverlust-Limit
  - Ein Zeit/Frequenz-Plan der das Maximum erreichen kann
- Viele Ideen müssen den Quellcode des MPI-Programms verändern/anpassen.

⇒ Wunsch nach einem System, welches das Optimum zusammen mit einer Umsetzung berechnet, um Heuristiken beurteilen zu können.

# Die Idee

- 1 Bestimme alle *tasks* als sequentielle Code-Teile zwischen MPI Aufrufen für jeden Prozessor
- 2 Bestimme Inter-*task*-Beziehungen die durch MPI-Kommunikation gegeben werden und stelle den Kommunikationsgraphen auf
- 3 Berechne mindest Gesamtzeit und Energieverbrauch durch lösen eines LP

# Modellierung - Variablen

Variable	Bedeutung
$P$	Anzahl CPUs
$\tau$	Menge aller <i>tasks</i>
$S_i$	Startzeit von <i>task</i> $i$
$T_i$	Ausführungszeit von <i>task</i> $i$
$C_i^f$	Ausführungszeit von <i>task</i> $i$ mit Frequenz $f$
$M_i^f$	Latenz von Nachricht von <i>task</i> $i$ zu <i>task</i> $j$
$Pred_i$	Menge direkter Vorgänger von <i>task</i> $i$
$F$	Menge aller Frequenzen
$\delta_i^f$	Für <i>task</i> $i$ der mit Frequenz $f$ auszuführende Anteil
$W_i^f$	Für <i>task</i> $i$ die bei Frequenz $f$ verbrauchte Energie
$W_I$	In <i>Idle-Time</i> verbrauchte Energie
$I$	<i>Idle-Time</i> des gesamten <i>Task-Graphen</i>
$\omega$	Laufzeit des kompletten Programms

# Modellierung - Variablen

Die Entscheidungsvariablen sind:

- $S_i$  ( $S_0 = S_{source} = 0$ )
- $\delta_i^f$

Alle anderen Werte können berechnet oder gemessen werden.

Im folgenden Modell werden die Zeiten und Kosten für die Frequenzveränderungen ignoriert, da sonst kein LP, sondern ein MIP vorliegen würde.

# Modellierung - Variablen

Die Entscheidungsvariablen sind:

- $S_i$  ( $S_0 = S_{source} = 0$ )
- $\delta_i^f$

Alle anderen Werte können berechnet oder gemessen werden. Im folgenden Modell werden die Zeiten und Kosten für die Frequenzveränderungen ignoriert, da sonst kein LP, sondern ein MIP vorliegen würde.

# Modellierung - LP

## LP-Formulierung

$$\min E = W_I l + \sum_{i \in \tau} \sum_{f \in F} W_i^f \delta_i^f C_i^f \quad (3a)$$

s.t.

$$S_j + T_j - S_i \leq 0 \quad \forall i \in \tau, j \in \text{Pred}_i \quad (3b)$$

$$S_i + T_i - \omega \leq 0 \quad \forall i \in \tau \quad (3c)$$

$$P\omega - \sum_{i \in \tau} T_i = l \quad (3d)$$

$$\sum_{f \in F} \delta_i^f = 1 \quad \forall i \in \tau \quad (3e)$$

$$S_i \geq 0 \quad \forall i \in \tau \quad (3f)$$

$$\delta_i^f \geq 0 \quad \forall i \in \tau \quad (3g)$$

# Umsetzung

- 1 Programmausführung zur *Trace*-Generierung mit Hilfe der PMPI Schnittstellen auf jeder möglichen Frequenz. Erhalte so auch *communication-slack* und *memory-slack* Informationen.
- 2 Aufstellen von *Task*-Graphen
- 3 Lösen des LP. Erhalte ein *energy-schedule*
- 4 Verbiete Frequenzveränderungen für zu kurze *tasks*
- 5 Validierung durch Wiederausführung des Programms mit dem *energy-schedule* unter Messung von Laufzeit und Energieverbrauch

# Implementierung und Testumgebung

## Einschränkungen der Implementierung:

- Code in MPI-Bibliotheken wird als Kommunikation modelliert
- Asynchrone Kommunikation ist noch nicht betrachtet
- Nicht alle MPI-Befehle werden instrumentiert

## Testumgebung:

- 8 \* 2 AMD Opteron 256 dual-core Prozessoren (jeweils nur einer genutzt)
- Gigabit Ethernet
- 2GB RAM
- 1000MHz/105W bis 1800MHz/153W in 6 Stufen
- Fedora Core 2 mit OpenMPI
- Präzisionsmultimeter an den Steckdosen

# Implementierung und Testumgebung

## Einschränkungen der Implementierung:

- Code in MPI-Bibliotheken wird als Kommunikation modelliert
- Asynchrone Kommunikation ist noch nicht betrachtet
- Nicht alle MPI-Befehle werden instrumentiert

## Testumgebung:

- 8 \* 2 AMD Opteron 256 dual-core Prozessoren (jeweils nur einer genutzt)
- Gigabit Ethernet
- 2GB RAM
- 1000MHz/105W bis 1800MHz/153W in 6 Stufen
- Fedora Core 2 mit OpenMPI
- Präzisionsmultimeter an den Steckdosen

# Ergebnisse

Energieersparnisse bis zu:

- Jacobi-Iterationen: Keine Ersparnisse.
- UMT2K (ASC Purple Suite: Photonen-Transport-Code): 3.3%
- Teilchensimulationen: 15%

Jede Anwendung hatte mindestens 10.000 Aufrufe von MPI-Kommunikationsfunktionen.

## 1 Einführung

## 2 Forschungsansätze - CPU

- DVFS
- Compiler Algorithmus
- Energieeinsparung bei MPI-Programmen
- **Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen**

## 3 Forschungsansatz - Netzwerk

## 4 Zusammenfassung

# Quelle

**Titel** A Power-Aware Run-Time System for High-Performance Computing

**Autoren** Chung-hsing Hsu, Wu-chun Feng  
Los Alamos National Laboratory  
Los Alamos, NM 87545

**Erschienen** Proceedings of the 2005 ACM/IEEE conference on Supercomputing

# Die Idee

## Überblick

- Algorithmen, die bei interaktiver Nutzung gut funktionieren, scheitern bei wissenschaftlichen Anwendungen
- Motivation sind Nachteile anderer Ansätze:
  - Programmcode muss modifiziert werden
  - Abhängigkeiten von Eingabedaten
  - Verhaltensprofile von Programmen können grundlegend verändert werden

⇒ Notwendigkeit nach einem transparenten, selbst anpassenden System, welches zur Laufzeit den Energieverbrauch regelt

## Bisher...

Bisherige Ansätze für Systeme die zur Laufzeit die CPU-Frequenzen anpassen...

- basieren vor allem auf der CPU-Auslastung
- verändern die CPU-Taktung wenn die Auslastung unter/über eine bestimmte Marke kommt
- sind trotz Anwendungs- und Eingabeunabhängigkeit nur für interaktive Nutzung gut. Sie nutzen vor allem CPU-*idle-time* aus, die es in HPC Programmen wenig gibt.
- bieten nur wenig Kontrolle über die Laufzeitverlängerung.

⇒ Notwendigkeit eines Systems, welches unabhängig von Eingabe und Quellcode, zur Laufzeit Entscheidungen zum Energiesparen trifft und trotzdem eine enge Kontrolle über die Auswirkungen auf die Verzögerung bietet.

# Die Idee

- Angabe  $\delta$  als maximale Ausführungsverlängerung
- Algorithmus passt zur Laufzeit Frequenzen so an, dass diese Verlängerung nicht überschritten wird
- Intervall-basierendes Entscheidungsverfahren
- Kriterien: Menge an *off-chip* Zugriffe im vorherigen Intervall

⇒ Frage nach Zusammenhang zwischen *off-chip* Zugriffen und gesamter Ausführungszeit.

⇒ Einführung eines Wertes  $\beta$ .

# $\beta$ -Angleichung

$$\frac{T(f)}{T(f_{max})} \approx \frac{mips(f_{max})}{mips(f)} \approx \beta \left( \frac{f_{max}}{f} - 1 \right) + 1$$

- $mips(f)$  wird pro Intervall gemessen und genutzt  $\beta$  zu bestimmen

Für die Frequenzen  $f_i \in \{f_1, \dots, f_n\}$  :

$$\min \sum_{i=1}^n \left\| \frac{mips(f_{max})}{mips(f_i)} - \beta \left( \frac{f_{max}}{f_i} - 1 \right) - 1 \right\|^2$$

$$\Rightarrow \beta = \frac{\sum_{i=1}^n \left( \frac{f_{max}}{f_i} - 1 \right) \left( \frac{mips(f_{max})}{mips(f_i)} - 1 \right)}{\sum_{i=1}^n \left( \frac{f_{max}}{f_i} - 1 \right)^2}$$

# $\beta$ -Angleichungs-Algorithmus

## Der Algorithmus

- 1 **Initialisierung von  $mips(f_i)$ :** Lasse Programm für  $I$  Sekunden bei  $f_i$  laufen
- 2 **Wiederhole**
- 3 **Berechne  $\beta$**
- 4 **Berechne gewünschte Frequenz  $f^*$ :**

$$f^* = \max\left(f_{min}, \frac{f_{max}}{1 + \frac{\delta}{\beta}}\right)$$

- 5 **Führe Interall  $I$  bei Frequenz  $f^*$  aus**
- 6 **Aktualisiere  $mips(f^*)$**
- 7 **bis Programm beendet ist.**

# Testumgebung

## Testumgebung:

- 4 \* AMD Athlon64 3200+ Prozessoren
- Gigabit Ethernet
- 1GB RAM
- 1-MB L2-Cache
- 800MHz/0.9V bis 2000MHz/1.5V in 4 Stufen
- SuSE Linux 2.6.7
- DVFS kernel module powernow-k8
- LAM/MPI v7.0.6
- Benchmark: NAS-MPI v3.2

# Ergebnisse

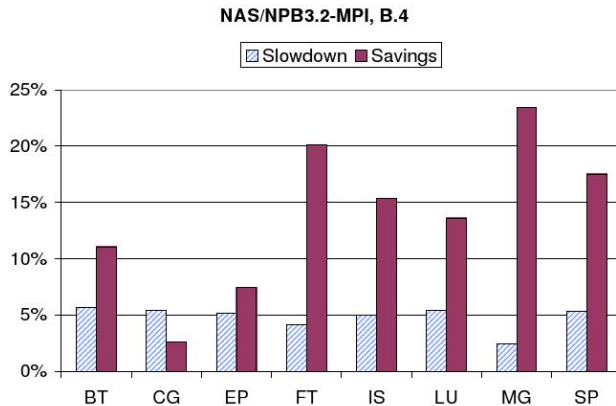


Abbildung: Ergebnisse für NAS-MPI, Class B Workload

## 1 Einführung

## 2 Forschungsansätze - CPU

- DVFS
- Compiler Algorithmus
- Energieeinsparung bei MPI-Programmen
- Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen

## 3 Forschungsansatz - Netzwerk

## 4 Zusammenfassung

# Quelle

**Titel** Performance Adaptive Power-Aware Reconfigurable Optical Interconnects for High-Performance Computing (HPC) Systems

**Autoren** Avinash Kodi, Ahmed Louri  
Ohio University / University of Arizona  
Athens, OH / Tucson, Arizona

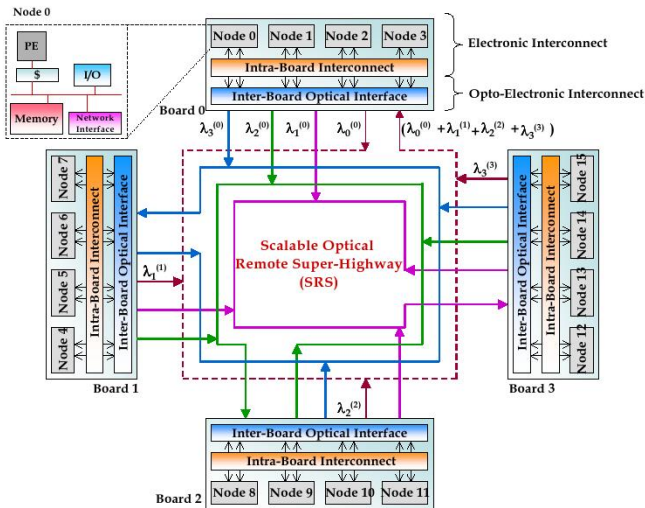
**Erschienen** Proceedings of the 2007 ACM/IEEE conference on Supercomputing

# Motivation

- Kommunikationsstrecken werden größer
- Bit-Raten wachsen
- Kommunikationsmuster sind zwischen unterschiedlichen Knoten unterschiedlich stark
- Dynamic Bandwidth Re-Allocation (DBR) Techniken existieren
  - DBR vergibt ungenutzte Verbindungen an ausgelastete Verbindungen
  - Energieverbrauch steigt stark mit an

⇒ Dynamic Power Management (DPM) soll helfen zusammen mit DBR die Leistung zu verbessern und dabei den Energieverbrauch zu minimieren.

## Die Idee



# Energieverbrauch

Die Netzwerkarchitektur besteht aus

- Sender
- Empfänger
- Kanal

Auf jedem Kanal kann mit mehreren Wellenlängen gesendet werden.

Energie wird verbraucht bei dem

- Sender durch
  - Laser
  - Laser-Driver
- Empfänger durch
  - Fotodetektor
  - Transimpedance-Amplifier
  - Taktgeber
  - Datenwiederherstellungsschaltkreisen

# Design-Möglichkeiten

Mögliche Kombinationen von DBR und DPM sind:

- Non-Power Aware, Non-Bandwidth Re-Allocation (NP-NB)
- Power Aware, Non-Bandwidth Re-Allocation (P-NB)
- Non-Power Aware, Bandwidth Re-Allocation (NP-B)
- Power Aware, Bandwidth Re-Allocation (P-B)

# Energie- und Bandbreitennutzung - NP-NB

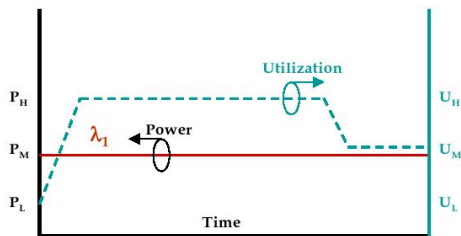


Abbildung: Non-Power Aware, Non-Bandwidth Re-Allocation

## Energie- und Bandbreitennutzung - P-NB

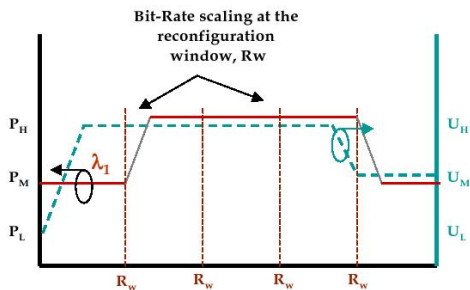


Abbildung: Power Aware, Non-Bandwidth Re-Allocation

## Energie- und Bandbreitennutzung - NP-B

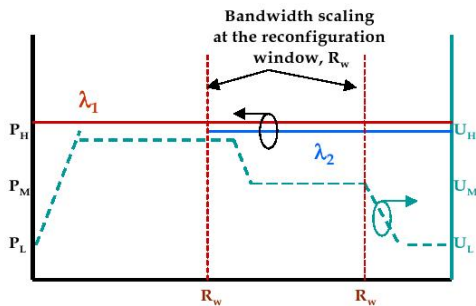


Abbildung: Non-Power Aware, Bandwidth Re-Allocation

## Energie- und Bandbreitennutzung - P-B

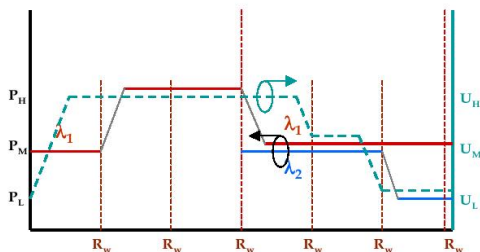
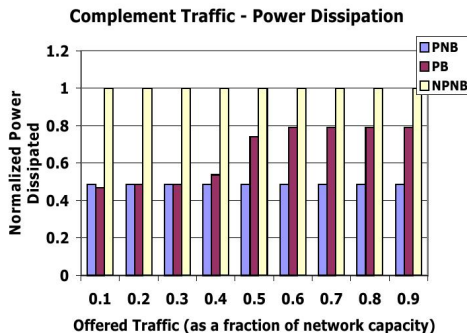


Abbildung: Power Aware, Bandwidth Re-Allocation

# Ergebnisse

Die Umsetzung der Idee wurde mit Hilfe eines Simulators durchgeführt.



## 1 Einführung

## 2 Forschungsansätze - CPU

- DVFS
- Compiler Algorithmus
- Energieeinsparung bei MPI-Programmen
- Energiebewusste Laufzeit-Systeme im Hochleistungsrechnen

## 3 Forschungsansatz - Netzwerk

## 4 Zusammenfassung

# Forschungsansätze allgemein

Zusammenfassend gilt:

- Energiesparen ist aktuell noch ein Forschungsthema
- Energiesparmechanismen für interaktiv genutzte Systeme funktionieren nicht gut für HPC Anwendungen
- Leistungsverminderung ist bei HPC Anwendungen nicht akzeptiert
- Viele Ansätze nutzen DVFS aus um den Energieverbrauch zu optimieren
- Es existieren Ansätze bei anderen Komponenten Energie zu sparen