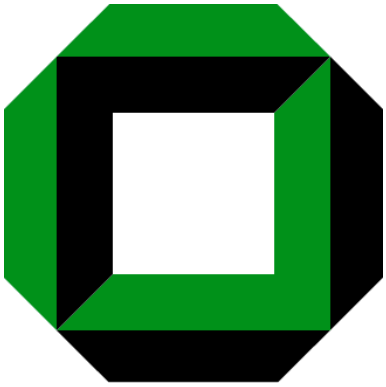


Universität Karlsruhe (TH)
Forschungsuniversität - gegründet 1825
Fakultät für Informatik
Lehrstuhl Systemarchitektur - Prof. Dr. Frank Bellosa

Energieabrechung und energiegewahre Verteilung asynchroner Ereignisse in Mehrprozessorsystemen

Diplomarbeit
von

Peter Leidinger

06. August 2007 – 05. Februar 2008

Verantwortlicher Betreuer: Prof. Dr.-Ing. Frank Bellosa
Betreuer: Dipl.-Inform. Andreas Merkel

Lehrstuhl Systemarchitektur
Universität Karlsruhe
76128 Karlsruhe

Hiermit versichere ich, die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe angefertigt zu haben. Die verwendeten Hilfsmittel und Quellen sind im Literaturverzeichnis vollständig angeführt.

Karlsruhe, den 05. Februar 2008

Peter Leidinger

Inhaltsverzeichnis

Inhaltsverzeichnis

Zusammenfassung

Energie ist in vielen Systemen eine kritische Ressource. Die hier erbrachte Arbeit baut auf bereits entwickelten Methoden zur Energieabrechnung und ihrer Verteilung auf. In bisherigen Implementierungen wird die während asynchroner Aktivitäten konsumierte Energie dem gerade laufenden Prozess zugeordnet. Dies ist nicht immer korrekt, da der aktuell laufende Prozess nicht zwingend Initiator dieser durch eine asynchrone Unterbrechung eingeleiteten Aktivität ist. Diesem Zustand wird in dieser Arbeit entgegengewirkt. Die durch ISRs (Interrupt Service Routine) einschließlich eventuell auftretender leistungsunkritischer späterer Rechenschritte (so genannte *Bottom Halves*) konsumierte Energie wird durch unsere Arbeit anstatt den unterbrochenen Prozessen den unterbrechungsinitiierenden Prozessen angerechnet. Initiiert ein Prozess eine Unterbrechung, so ist es aus energetischer und leistungsbezogener Sicht günstiger, wenn die initiierte Unterbrechung zu demjenigen NUMA-Knoten gesendet wird, dessen Prozessoren den Initiatorprozess ausführen. Die durch unsere Arbeit gesammelten Daten ermöglichen eine dynamische Unterbrechungsverteilung, die dies gewährleistet.

Kapitel 1

Einführung

1.1 Problembeschreibung

Die Erhöhung von Taktraten sowie die Erhöhung von Schaltkreisdichten heutiger Recheneinheiten erfordert einen steigenden Fokus auf Technologien, die sich der Energieabfuhr innerhalb solcher Systeme widmen. Die einer CPU zugeführte Energie lässt sich durch sich bewegende elektrische Ladung beschreiben. Dies impliziert die Emission von Energie durch elektromagnetische Wellen und Energieverlust durch Phononen-Wechselwirkung. Die hierdurch entstehende Wärme kann, sofern sie nicht abgeführt wird, zu Fehlfunktionen und Zerstörung der Recheneinheiten führen. Die thermische Energie muss folglich durch Kühlsysteme abgeführt werden. Dies wird zumeist durch Kühlsysteme realisiert, welche sich am maximalen Energieverbrauch ausrichten, den reale Applikationen konsumieren können (TDP – Thermal Design Power). Solche Kühlsysteme, die sich an der maximalen Leistung orientieren, verursachen zusätzlich einen überproportionierten Bedarf an Energie, wenn die Systemleistung unterhalb TDP liegt. Moderne Prozessoren verfügen bereits über Mechanismen, welche den Energieverbrauch der Kühlsysteme durch Auslesen von CPU-Wärmedioden regulieren [Dat06, Dat07].

Energieeinsparungen werden hauptsächlich durch Herabsetzen der anliegenden Spannung (DVS - Dynamic Voltage Scaling) oder Herabsetzen anliegender Taktraten (Clock Scaling) realisiert. Dies führt allerdings zu erheblichen Performance-Einbußen. Der Energieverbrauch heutiger Prozessoren steht in direkter Abhängigkeit zu den Instruktionen, die vom Prozessor ausgeführt werden. Somit existiert ein direkter Bezug zwischen Energieverbrauch, Prozessortemperatur und ausgeführtem Prozess [IM03, BWWK03]. Folglich lassen sich Prozesse aus Sicht ihrer Energiecharakteristika und korrespondierender CPU-Temperatur in *Hot Tasks* und *Cool Tasks* einteilen [MB06]. Entgegen der herkömmlichen Prozesseinteilung aufgrund ihrer Prozessorauslastung unterscheiden sich *Hot-* und *Cool-Task* durch ihren Energieverbrauch. Verbraucht ein Prozess viel Energie, so bezeich-

nen wir diesen als *Hot-Task*. Weißt ein Prozess einen geringeren Energieverbrauch auf, so bezeichnen wir ihn als *Cool-Task*. *Hot-Tasks* führen zu höheren Prozessortemperaturen als *Cool-Tasks*. Durch Betrachtung dieser Prozesseinteilung lassen sich Scheduling-Verfahren entwickeln, die eine gleichmäßige Energie- und Temperaturverteilung im Mehrprosessorsystem ermöglichen. Befinden sich auf einer CPU ein oder mehrere *Hot Tasks*, so können diese zu einer CPU migriert werden, welche weniger oder keinen dieser *Hot Tasks* verarbeitet. Man spricht hierbei von einem *energiegewahren Scheduler*.

Bisherige prozessspezifische Energiemessungen werden durch Differenzbildung zweier aufeinanderfolgender Energiebeträge gebildet. Der Energiebetrag ΔE_A, welcher einem Prozess A angerechnet wird, lässt sich durch

$$\Delta E_A = E_{t_2} - E_{t_1} \tag{1.1}$$

berechnen. Hierbei repräsentieren E_{t_1} bzw. E_{t_2} die zu den Zeitpunkten t_1 (Beginn der Zeitscheibe) bzw. t_2 (Ende der Zeitscheibe) gemessenen relativen Energiebeträge des Prozessors, welcher Prozess A zugeteilt wurde. Die Messzeitpunkte t_1 und t_2 sind stets an den Scheduler gekoppelt, da sie während Ein- und Ausplanung des Prozesses stattfinden. Die hierdurch erhältlichen prozessspezifischen Energiewerte entsprechen somit nicht immer den wirklich konsumierten prozessspezifischen Energien. Energieverbrauch, der durch asynchrone Unterbrechungen entsteht, wird ungerechtfertigt immer dem gerade laufenden Prozess zugeordnet. Asynchrone Unterbrechungen sind aus Sicht des Scheduler transparent. Der Scheduler wird nicht in die Abarbeitung asynchroner Ereignisse involviert, da sie während einer durch ihn zugeteilten Zeitscheibe stattfinden. Die Messzeitpunkte t_1 und t_2 liegen somit außerhalb der Unterbrechungsbehandlung. Die Zuteilung der Unterbrechungsenergie ist nicht korrekt.

Initiiert ein Prozess A einen hochlatenten Lesezugriff auf ein beliebiges Gerät, so wird ihm für die Wartezeit die CPU entzogen. Der Scheduler plant, soweit vorhanden, einen Prozess B ein. Signalisiert das angesprochene Gerät nun durch eine asynchrone Unterbrechung die Bereitstellung der angeforderten Daten, so wird die durch die Unterbrechung konsumierte Energie Prozess B anstatt dem verantwortlichen Prozess A zugeteilt. Somit können Prozesse, die hohe Datenmengen mit peripheren Geräteinstanzen austauschen, fälschlicherweise als *Cool Tasks* eingestuft werden. Im Gegensatz können andere Prozesse, welche kaum auf die Peripherie zugreifen, als *Hot Tasks* eingestuft werden. Dies sind Fehlentscheidungen, welche leicht zu thermischen Ungleichgewichten führen.

Während asynchroner Aktivitäten wird Energie in Wärme umgewandelt. Wird diese Energie nicht richtig zugeordnet, so kann ein *energiegewahrer Scheduler* keine optimale Energieverteilung gewährleisten. Eine korrekte Assoziation der Unterbrechungs-Energie zu den Initiatoren wirkt diesem Missstand entgegen. Umgekehrt kann durch diese Assoziation die energiegewahre Verteilung der Unterbrechungen stattfinden. Initiiert ein Prozess A Unterbrechungen auf einem entfernten Prozessor C, so kann ein Scheduler, welcher Prozessortemperatur und prozessspezifischen Energieverbrauch in seine Entschei-

dungen einbezieht, keine optimalen Entscheidungen treffen. Die Temperatur des Prozessors C steht in Abhängigkeit zu Prozess A, welcher auf einem anderen Prozessor eingeplant ist. Die Unterbrechungsenergie wird einem der durch C abgearbeiteten Prozessen willkürlich zugeordnet. Migriert der Scheduler diesen Prozess, so ändert dies nichts an der Tatsache, dass Prozessor C immer noch die Unterbrechungsenergie konsumiert.

1.2 Lösungsansatz

Die Energie, die während asynchroner Unterbrechungen verbraucht wird, muss anstatt den unterbrochenen Prozessen den Initiatoren dieser Unterbrechungen angerechnet werden. Hierzu muss eine systemweite Assoziation zwischen Prozessen und asynchronen Unterbrechungen geschaffen werden. Dies impliziert eine Aufhebung der systeminternen Transparenz asynchroner Unterbrechungen sowie der Transparenz von Gerätezugriffen, welche durch asynchrone Unterbrechungen realisiert werden. Die vorliegende Transparenz ermöglicht Datentransfers mit externen Geräten, ohne dass Prozesse sich um die Art und Weise wie die Daten transferiert werden beschäftigen müssen. Welches Gerät die erforderlichen Daten bereitstellt und wie es dies tut ist ebenfalls (bei erfolgreicher Datenübertragung) aus Prozesssicht irrelevant. Um diese Transparenz für den Betriebssystem-Kern ohne merklichen Leistungsverlust zu lockern, haben wir eine bereits vorhandene Schnittstelle zwischen Prozessen und Geräten erweitert: Systemaufrufe. Versucht ein Prozess auf eine periphere Ressource zuzugreifen, so muss dies in sicheren mehrfädigen Systemen über Systemaufrufe stattfinden. Initiiert ein Prozess A einen Systemaufruf, so wird in einer internen Liste überprüft, ob der Systemaufruf voraussichtlich eine asynchrone Unterbrechung auslösen wird. Ist dies der Fall, so wird ein entsprechender Eintrag im Prozess-Kontroll-Block (PCB - Process Control Block) des Initiators gesetzt. Nun können zwei Fälle eintreten:

1. Der Prozess initiiert keine asynchrone Unterbrechung,
2. Der Prozess initiiert eine asynchrone Unterbrechung.

Im ersten Fall wird der entsprechende PCB-Eintrag wieder gelöscht. Der Ablauf des Systemaufrufs folgt seinem gewöhnlichen Lauf. Im zweiten Fall wird der Prozess in einen schlafenden Zustand gesetzt. Entsprechend der Scheduler-Strategie kann ein weiterer Prozess eingeplant werden. Beim Eintritt der initiierten Unterbrechung ruft das System die entsprechende Behandlungsroutine (ISR - Interrupt Service Routine) auf. Die während der Unterbrechung konsumierte Energie ΔE_i wird berechnet und nach Beendigung der Unterbrechung dem unterbrochenen Prozess, welchem diese Energie später durch den Scheduler ungerechtfertigt angerechnet wird, im Voraus abgezogen. Die Berechnung der Prozessenergie E_A durch

$$E_A = E_A - \Delta E_i + \Delta E_i \tag{1.2}$$

ist redundant. Um diese Berechnung zu umgehen muss der Scheduler allerdings vor und nach jeder Unterbrechung aufgerufen werden. Der hierdurch entstehende Overhead übertrifft allerdings jede Zumutbarkeit[1] und würde die Unterbrechungsbehandlung ad absurdum führen.

Die Energiesumme ΔE

$$\Delta E = \sum_i \Delta E_i \tag{1.3}$$

wird in einem Puffer gesichert. Nach einer oder mehreren dieser Unterbrechungen wird dem Scheduler signalisiert, dass Prozess A wieder eingeplant werden kann. Wird Prozess A wieder eingeplant, so wird ihm die durch die Unterbrechungen konsumierte Energie ΔE angerechnet. Der von A initiierte Systemaufruf kann nun weiterlaufen. Der entsprechende PCB-Eintrag wird gelöscht.

Die energiegewahre Verteilung der Unterbrechungen wird zum Bestandteil des energiegewahren Schedulers. Der energiegewahre Scheduler erreicht ein thermische Gleichgewicht durch Migration von *Hot-Tasks*, welche auf „heißen“ Prozessoren ausgeführt werden, zu „kühlen“ Prozessoren. Entsteht durch eine solche Migration ein Lastenungleichgewicht, so tauscht der Scheduler *Hot-Tasks* eines „heißen“ Prozessors gegen *Cool-Tasks* eines „kalten“ Prozessors. Sind weniger Prozesse aktiv als Prozessoren vorhanden sind, so werden *Hot-Tasks* zu unbeschäftigten Prozessoren migriert (HTM - Hot Task Migration). Unser energie- und unterbrechungsenergiegewahrer Scheduler verringert die Anzahl solcher Migrationen in Fällen eines hohen Unterbrechungsenergieverbrauchs. Wird ein Prozess als *Hot-Task* eingestuft, so wird überprüft, ob dieser Einstufung ein hoher Anteil an Unterbrechungsenergie zugrunde liegt. Erfolgt die Abarbeitung der verantwortlichen Unterbrechung durch den gleichen Prozessor, welcher den *Hot-Task* abarbeitet, so wird die Unterbrechung zu einem Prozessor des gleichen NUMA-Knotens migriert. Liegt der restliche Energieverbrauch des *Hot-Task* unterhalb des Schwellwertes, welcher als Maß zur *Hot-Task*-Einstufung diente, so erfolgt keine Migration. Erfolgte die Einteilung als *Hot-Task* aufgrund von Unterbrechungsenergie, welche durch einen anderen Prozessor verbraucht wurde, wird der Prozess nur migriert, wenn sein Restenergieverbrauch oberhalb des Schwellwertes liegt.

Eine direkte Zuordnung zwischen Prozessen und den von ihnen initiierten asynchronen Unterbrechungen impliziert einen weiteren vorteilhaften Effekt: In NUMA-Architekturen (Non Uniform Memory Access) herrschen unterschiedliche Speicherlatenzen. Initiiert ein auf Prozessor i laufender Prozess A eine Unterbrechung n, welche auf Prozessor $j \neq i$ eintrifft, so ist dies weniger effizient als ein Eintreffen der Unterbrechung n auf Prozessor i. Die Unterbrechungsverteilung anhand der gesammelten Daten kann somit herangezogen werden, um den Durchsatz von vielen datentransferintensiven Anwendungen zu

[1] Additionen beschäftigen den Prozessor wenige Zyklen. Die Unterbrechungsbehandlung sowie die Aktivierung des Scheduler benötigen jeweils mehrere tausend Zyklen.

steigern. Aufgrund der Energieeinsparungen, welche durch die NUMA- und prozessgewahre Unterbrechungsverteilung entstehen ist diese nicht nur Nebenprodukt. Durch unsere Unterbrechungsverteilung wird der Energieverbrauch des Gesamtsystems verringert. Wir haben einen Mechanismus und eine Strategie entwickelt, welche erstmalig eine prozessspezifische energie- und NUMA-gewahre Verteilung asynchroner Unterbrechungen ermöglicht.

1.3 Strukturierung

Der weitere Verlauf dieser Arbeit ist wie folgt gegliedert: Kapitel 2 gibt einen kurzen Einblick in die Grundlagen, welche zum Verständnis der weiteren Kapitel von Vorteil sind. Das darauf folgende Kapitel 3 gibt einen Überblick über verwandte Arbeiten. Kapitel 4 beschreibt unser Design. Kapitel 5 zeigt die Durchführung einer exemplarischen Implementierung der von uns vorgestellten Arbeit anhand Erweiterungen eines Linux-Kernels. Der vorgestellten Implementierung folgt deren Evaluierung in Kapitel 6. Kapitel 7 dient zur abschließenden Beurteilung und der Diskussion zukünftiger Arbeiten.

1.4 Nomenklatur

Zur besseren Lesbarkeit haben wir nur solche Anglismen ins Deutsche übersetzt, welche sich bereits im deutschsprachigen Raum etabliert haben. Akronyme, Programmtexte, Funktionsnamen und Bezeichner sind stets in englischer Sprache gehalten.

Unsere Arbeit widmet sich asynchronen Ereignissen. Solche werden hauptsächlich durch asynchrone Unterbrechungen ausgelöst. Sofern nicht explizit widerrufen, verstehen wir aus Gründen besserer Lesbarkeit und Redundanzvermeidung, unter *Unterbrechungen* immer *asynchrone Unterbrechungen*. Unter einem Prozess verstehen wir einen lauffähigen Programmcode, welcher einen Prozess-Kontroll-Block (PCB - Process Control Block) besitzt. Wir differenzieren nicht zwischen leichtgewichtigen oder anderen Prozessen.

Energie kann weder verbraucht noch konsumiert werden. Die SI-Einheit von Energie ist das *Joule* (*J*). Es gelten die Energie- und Impulserhaltungssätze der Quantenmechanik. Energie kann von einer Energieform (mechanische Energie, Strahlung, Wärmeenergie, chemische Energie u.s.w.) in eine oder mehrere andere überführt werden. Diejenige Energie, die pro Zeiteinheit von einem System auf ein anderes übertragen wird, bezeichnet man als Leistung. Die SI-Einheit der Leistung ist das *Watt* (*W*) [Tip94]. Zur besseren Lesbarkeit sowie zur Vermeidung von Verwechslungen mit informationstechnischer Nomenklatur verwenden wir folgenden Ansatz: Sprechen wir von Leistung, so sprechen wir

von informationstechnischer Leistung[2]. Sprechen wir von *konsumierter* oder *verbrauchter* Energie, so sprechen wir von physikalischer Leistung. Wir tun dies, obwohl wir wissen, dass Energie nicht verbraucht werden kann. Sprechen wir von Energie, so meinen wir denjenigen Energiebetrag, welchen wir (z.B.: für eine Instruktion) verbrauchen werden.

[2]Dies wird häufig als Performanz bezeichnet. Solche Anglismen werden von uns generell vermieden [Rec02].

Kapitel 2

Systemarchitektonische Grundlagen

Zum Verständnis dieser Arbeit bietet dieses Kapitel eine kurze Einführung in systemarchitektonische Grundlagen. Dies führt von Multiprozessoren hin zu asynchronen Unterbrechungen in Mehrprozessorsystemen.

2.1 Multiprozessoren

Ein Multiprozessor ist ein Computersystem, in welchem sich mehrere Prozessoren einen gemeinsamen Speicher teilen, um einen Leistungsgewinn im Vergleich zu Uniprozessorsystemen zu erzielen [Tan03].

2.1.1 SMP und SMT

Symmetrische Multiprozessoren (SMP - Symmetric Multiprocessing) sind aus gleichartigen Prozessoren aufgebaute Multiprozessoren. Wir werden solche im Folgenden der Einfachheit halber als Multiprozessoren bezeichnen (asymmetrische Multiprozessoren liegen nicht im Fokus dieser Arbeit). Multiprozessoren sind prinzipiell in der Lage, gleichzeitig Prozesse zu bearbeiten, wobei Speicherkohärenz durch Hardware und Software unterstützt werden kann (Multiprozessorsynchronisation). Simultanes Multithreading (SMT - Simultaneous Multithreading) hingegen bezeichnet die Fähigkeit physischer Prozessoren, mehrere Prozesse durch integrierte logische Prozessoren gleichzeitig zu verarbeiten. Einziger Anbieter von Prozessoren mit SMT-Unterstützung ist die Firma Intel. Intel bezeichnet SMT als HyperThreading. Diese Technologie verwendet logische Prozessoren, welche Ressourcen eines physischen Prozessors exklusiv, partitioniert oder geteilt nutzen.

2.1.2 NUMA – Nonuniform Memory Access

Jeder Prozessor eines Multiprozessorsystems kann den gesamten zur Verfügung stehenden Speicher adressieren. Systeme, welche für jeden Prozessor die gleiche Speicherlatenz garantieren, werden als UMA-SMP-Systeme (UMA - Uniform Memory Access) bezeichnet. SMP-Systeme mit unterschiedlichen prozessorspezifischen Speicherlatenzen werden als NUMA-Systeme bezeichnet. Hierbei erfolgt eine Einteilung des Speichers und der Prozessoren in so genannte NUMA-Knoten. Erfolgt ein Speicherzugriff auf einen entfernten NUMA-Knoten, so muss ein Inter-Node-Zugriff durchgeführt werden. Dies führt in jedem Fall zu Leistungsverlusten.

2.1.3 Speicherhierarchie

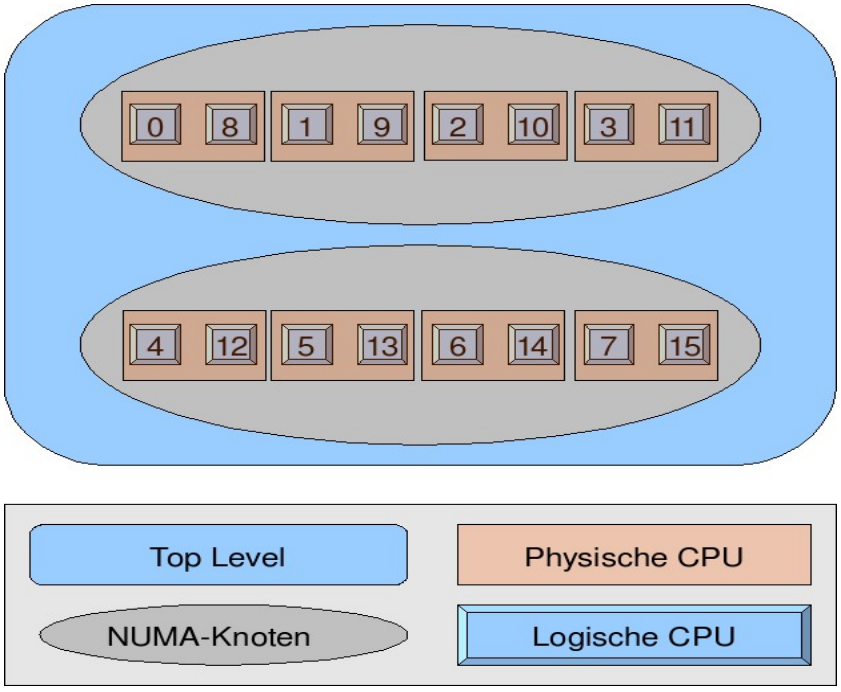
Multiprozessoren weisen im Allgemeinen eine Speicherhierarchie auf. Speicherzugriff erfolgt über ein dediziertes Bussystem, welches bei steigender Kardinalität an Prozessoren einen Flaschenhals darstellt. Um die Buszugriffe zu minimieren, werden prozessorspezifische Zwischenspeicher, so genannte Caches, verwendet. Liest ein Prozessor ein Speicherwort, so wird meist ein ganzer Datenblock in den lokalen Cache kopiert. Der Prozessor erhält anfangs lediglich Lesezugriff auf den eingelagerten Datenblock. Erfolgt nun ein Schreibzugriff auf den lokal eingelagerten Datenblock, so muss dieser in einem speicherkoherenten System mit allen anderen Prozessoren synchronisiert werden. Um dies sicherzustellen, gibt es eine Vielzahl an Protokollen [Tan03, Tan02].

Als Beispiel stellen wir ein NUMA-SMT-Multiprozessorsystem mit acht physischen Prozessoren vor (siehe Abbildung 2.1). Jeder physische Prozessor verfügt über zwei logische Prozessoren, die sich einen gemeinsamen Cache teilen. Es ergibt sich folgende Speicherhierarchie:

1. Ebene: physischer Prozessor,
2. Ebene: NUMA-Knoten,
3. Ebene: Top Level.

Die erste Ebene involviert die logischen Prozessoren eines physischen Prozessors, die sich den selben Cache (*L1*, *L2*, *L3*) teilen. Die Leistung eines Speicherzugriffs lässt sich anhand der Anzahl der Ebenen, welche durchlaufen werden müssen, feststellen. Je geringer die Distanz zwischen logischem Prozessor und Speicherwort, desto geringer die Latenz.

Abbildung 2.1: Speicherhierarchie eines NUMA-SMT-Multiprozessorsystems



2.2 Asynchrone Unterbrechungen

Unter asynchronen Unterbrechungen verstehen wir Unterbrechungen, die zu nicht an den Programmfluss gekoppelten Zeitpunkten auftreten. Zeitpunkte asynchroner Unterbrechungen können nicht reproduziert werden. Ihre Eintrittszeitpunkte sind oft von solch statistischer Natur, dass sie in die Generierung von Zufallzahlen involviert werden können. Als Beispiel hierfür sei der durch Love dokumentierte Entropie-Pool, welcher zur Generierung von echten Zufallszahlen des Linux-Kernels verwendet wird, gegeben [Lov05]. Asynchrone Unterbrechungen können zurückgestellt werden. Eine Unterbrechung wird durch ein externes Gerät (Hardware) ausgelöst. Hierbei erzeugt das Gerät ein elektrisches Signal und sendet dies zu einem Eingabe-Pin des programmierbaren Unterbrechungs-Controllers (PIC - Programable Interrupt Controler). Dieser sendet ein Signal an den Prozessor. Empfängt der Prozessor ein solches Signal, so unterbricht er seine aktuelle Tätigkeit und informiert das Betriebssystem über die eingetretene Unterbrechung.

Synchrone Unterbrechungen werden durch den Prozessor erzeugt (Division durch Null, Seitenfehler u.s.w.). Sie können nicht zurückgestellt werden. Tritt beispielsweise ein Seitenfehler auf, so macht es wenig Sinn, diesen nicht direkt zu behandeln, da der aktuell ausgeführte Prozess ohne die einzulagernde Seite nicht weiter ausgeführt werden kann. Synchrone Unterbrechungen sind an den Programmfluss gekoppelt und können reproduziert werden.

Im Folgenden werden wir zur besseren Lesbarkeit asynchrone Unterbrechungen, sofern nicht explizit angegeben, schlicht als *Unterbrechungen* bezeichnen.

2.2.1 Grundlagen

Es existieren drei grundlegende Mechanismen zur Durchführung von Ein- und Ausgabe [Tan03]:

1. Programmierte,
2. Unterbrechungsgesteuerte und
3. Direct Memory Access (DMA).

Der erstgenannte Mechanismus involviert als einziger keine Unterbrechungen. Versucht ein Prozess mit einem Gerät zu kommunizieren, so geschieht dies durch Polling (Geschäftiges Warten): Das angesteuerte Gerät wird immer wieder gefragt, ob es Daten entgegennehmen oder liefern kann. Der Prozessor ist somit insbesondere bei geringen Datenraten mit ineffizienten Polling beschäftigt. Dies führt zu Leistungsverlusten.
Der Leistungverlust wird anhand beider verbleibender Mechanismen durch Involvieren von Unterbrechungen gemildert. Die unterbrechungsgesteuerte Ein- und Ausgabe agiert

folgendermaßen: Versucht ein Prozess A lesenden oder schreibenden Zugriff auf ein peripheres Gerät G durchzuführen, so wird ein entsprechender Systemaufruf (read, write, sendfile u.s.w.) aufgerufen. Dem Aufruf entsprechend wird ein Signal auf einen Bus gelegt, welcher eine Verbindung zwischen Prozessor und Gerät realisiert. Dieses Signal informiert das anzusprechende Gerät über die von Prozess A gestellte Anfrage. Anschließend wird der Scheduler aufgerufen, welcher dem Prozessor einen neuen Prozess B zuteilt[1]. Sobald Gerät G die Anfrage bearbeitet hat, informiert es einen PIC. Dieser sendet, soweit keine höher privilegierte Unterbrechung vorliegt, ein Signal mit entsprechender Nummer (Unterbrechungslinie) zum Prozessor. Dieser unterbricht nun, sofern diese Unterbrechung nicht explizit beispielsweise durch Prozess B unterbunden wurde, den aktuellen Prozess B. Die Unterbrechung ist entweder durch prozessorspezifische Schaltlogik oder durch das Betriebssystem für Prozesse und den Scheduler transparent. Trifft eine Unterbrechung ein, so lädt der Prozessor den Programmzähler, welcher in einer speziellen Tabelle (Unterbrechungsvektor-Tabelle) unter dem Index der gesendeten Unterbrechungs-Nummer gespeichert ist. Dieser Programmzähler zeigt auf die gerätespezifische Unterbrechungsroutine (ISR - Interrupt Service Routine), welche folglich durchlaufen wird. Die Implementierung der ISR erfolgt innerhalb dedizierter Gerätetreiber. Sie wird explizit im System registriert. Während der Abarbeitung der ISR werden Unterbrechungen mit gleicher Unterbrechungs-Nummer gesperrt (maskiert). Während der Abarbeitung einer Unterbrechung können weitere Unterbrechungen des gleichen Gerätes anliegen. Diese können durch die ISR selbst initiiert werden. Die implizierte Rekursion kann dazu führen, dass keine ISR beendet werden kann. Das System kehrt nie aus der Unterbrechungsbehandlung zurück. Ähnliches geschieht, wenn die Periode zweier Unterbrechungen des gleichen Gerätes kürzer ist als die Zeit, welche die ISR benötigt, um eine Unterbrechung abzuarbeiten. Die ISR sollte grundsätzlich nur kritische Operationen, wie beispielsweise das Kopieren von Daten zwischen einem Puffer und dem Gerät, durchführen. Unkritische Operationen können außerhalb der ISR abgearbeitet werden. Die Gesamtheit solcher unkritischer Operationen, welche durch Unterbrechungen initiiert werden, bezeichnen wir als *Bottom Halves*[2]. Während der Abarbeitung der ISR oder seiner *Bottom Halves* wird dem System signalisiert, dass Prozess A wieder eingelagert werden kann. Anhand der Scheduler-Strategie kann nun Prozess A, B oder ein weiterer den Prozessor zugeteilt bekommen. Wird Prozess A wieder eingelagert, so kehrt er aus seinem Systemaufruf zurück. Der Datentransfer ist für ihn transparent.
Gerätezugriff mittels DMA erfolgt ähnlich dem unterbrechungsgesteuerten Zugriff. Der Unterschied liegt darin, dass der eigentliche Datentransfer zwischen Gerät und Prozessor ungeachtet des Prozessors in einen dedizierten Bereich des Speichers gepuffert wird. Die Synchronisation transferierter Datenblöcke wird wiederum durch Unterbrechungen reali-

[1] Ist kein lauffähiger Prozess verfügbar, so wird der Idle-Prozess eingegliedert.

[2] Die Bezeichnung *Bottom Halves* (BHs) ist historisch bedingt. Linux verwendet dies als Sammelbegriff für Tasklets und Softirqs.

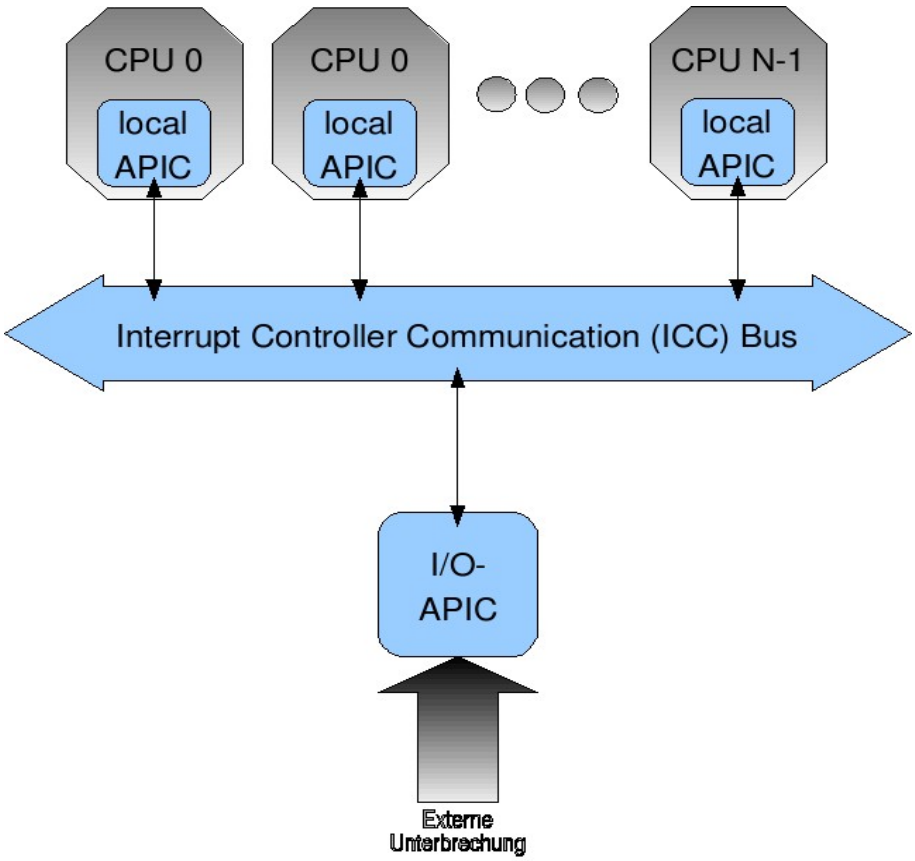
siert. Der Transfer großer Datenblöcke kann somit weitestgehend ohne Beschäftigung des Prozessors realisiert werden. Die Größe der pro Unterbrechung übertragenen Datenblöcke ist umgekehrt proportional zur Kardinalität der auftretenden Unterbrechungen.

2.2.2 Asynchrone Unterbrechungen in Mehrprozessorsystemen

Bisher wurde lediglich der Mechanismus *asynchrone Unterbrechung* behandelt. Dieser Mechanismus stößt in Mehrprozessorsystemen auf weitere Schwierigkeiten. Man betrachte hierzu die höhere Komplexität eines Mehrprozessor-Scheduler verglichen mit einem Uniprozessor-Scheduler. Die Portierung des Mechanismus *asynchrone Unterbrechung* auf solche Systeme birgt eine ähnliche Komplexität. Die Beherrschung dieser Komplexität wird allerdings durch Hardware-Mechanismen erleichtert. Es muss sichergestellt werden, dass jede Unterbrechung einen einmaligen Aufruf der entsprechenden ISR durch einen Prozessor impliziert.

Die Verteilung einer durch ein externes Gerät verursachten Unterbrechung wird vorerst an einen externen PIC gesendet. Dieser kümmert sich um die Verteilung der Unterbrechung auf die vorhandenen Prozessoren. Der externe PIC ist durch jeden der Prozessoren programmierbar. Als Fallbeispiel stellen wir hier die PIC-Architektur von Intel vor. Unsere Implementierung baut auf ein solches Multiprozessorsystem auf:
Geräte, die an einem externen Bus (PCI, ISA, u.s.w.) hängen, liefern Unterbrechungsanfragen (IRQ - Interrupt Request) über diesen BUS an einen so genannten I/O Advanced Programable Interrupt Controler (I/O APIC) [Int, Int96, BC05]. Ein I/O APIC kann bis zu 24 verschiedene Unterbrechungen verwalten. Hierbei kann jeder Unterbrechung durch ein programmierbares Register eine Referenznummer eines lokalen APIC, welcher einen festen Bestandteil eines jeden Prozessors darstellt, übergeben werden. Die Summe dieser Register wird als *Redirektions-Tabelle* (Redirection Table) bezeichnet. Der I/O APIC sendet nun die Unterbrechung über den ICC-Bus (Interrupt Controller Communication Bus) an die der Referenznummer entsprechenden lokalen APIC. Der lokale APIC unterbricht folglich seinen lokalen Prozessor (siehe Abbildung 2.2).

Abbildung 2.2: Intel Programable Interrupt Controler



Kapitel 3

Verwandte Forschung

3.1 Energiegewahre Verteilung in Multiprozessorsystemen

3.1.1 Energieverbrauchsschätzung

Zur Durchführung einer energiegewahren Prozessverteilung ist eine verlässliche Methode zur Bestimmung des Energieverbrauchs unabdingbar. Eine empirische Online-Messung prozessspezifischer Verbrauchsdaten ist aufgrund der hohen Frequenz, mit welcher der Energieverbrauch auf High-End-Prozessoren wechselt, nicht möglich. Offline-Methoden, welche durch Simulationsprogramme den Energieverbrauch bestimmen, sind aufgrund schlechter oder nicht vorhandener Modelle sowie zu niedriger Simulationsgeschwindigkeit für unser Vorhaben ebenfalls nicht geeignet [BTM00, GSI+02]. Aufgrund der bereits in Kapitel 1.1 erklärten Abhängigkeit zwischen ausgeführten Recheninstruktionen und zeitgleich konsumierter Energie bietet sich der Umweg über Ereigniszähler hin zur Energieabschätzung an [Bel00, Kel03].
Kellner verwendet zur Erfassung der Instruktionsdichte Ereigniszähler, welche in vielen Prozessoren zur Leistungsmessung integriert sind. Die Gesamtenergie ΔE, welche durch n verschiedene Ereignisse verbraucht wird, ergibt sich somit zu

$$\Delta E = \sum_{i=1}^{n} a_i \cdot c_i. \tag{3.1}$$

Hierbei konsumiert ein einzelnes Ereignis, also die zu Ereigniszähler c_i assoziierte Instruktion, die Energie a_i (Energiegewicht von Ereignis i). Die Bestimmung der jeweiligen Energiegewichte erfolgt numerisch. Die zur numerischen Berechnung notwendigen Daten ergeben sich aus der empirisch gemessenen Energiezufuhr des Prozessors und den prozessspezifischen Ereigniszählerwerten einiger ausgewählter Testprogramme. Der relative Fehler der Energieschätzung für handelsübliche Applikationen liegt hierbei bei kaum mehr als 6% [Kel03].

3.1.2 Thermales Prozessor-Modell

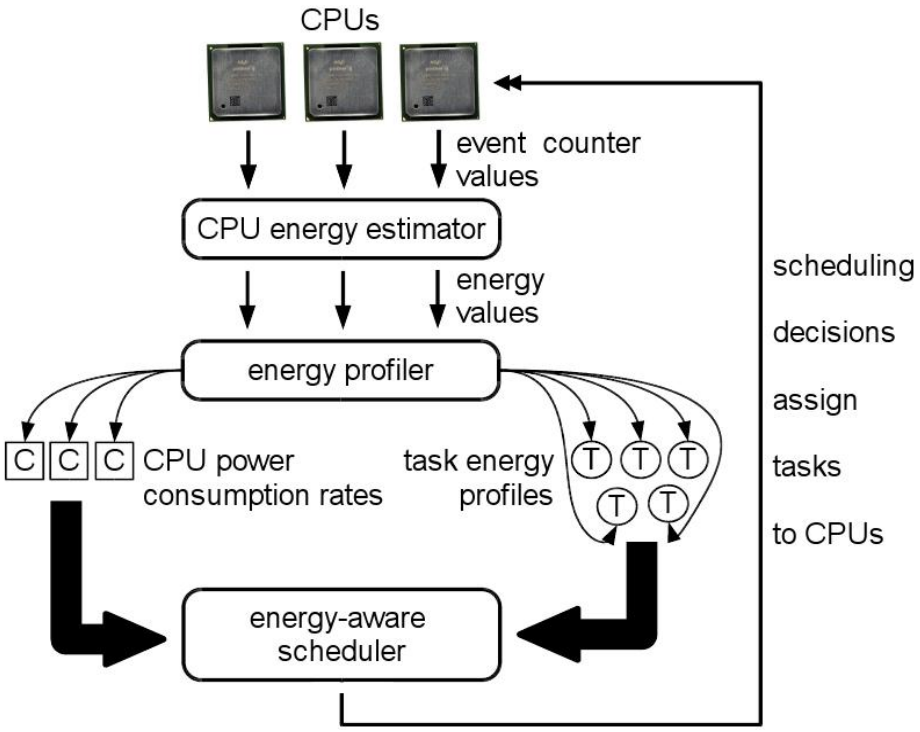
Das Vorhandensein einer adäquaten Methode zur Energieverbrauchsschätzung kann zur Berechnung der Temperatur eines Prozessors herangezogen werden [BWWK03, Kel03]. Ein Modell, welches Prozessortemperatur, Energieverbrauch und Prozessorkühlsystem in Zusammenhang stellt, wird als *Thermales Prozessor-Modell* bezeichnet. Wir werden dies im Folgenden als *Thermales Modell* bezeichnen, da wir uns ausschließlich mit thermalen Modellen von Prozessoren beschäftigen.
Die Temperaturdifferenz des durch Bellosa et al. und Kellner erarbeiteten thermalen Modells zur gemessenen Prozessor-Temperatur beträgt weniger als $1.2\ °C$. Dies ergibt einen relativen Fehler von 2.12% [BWWK03, Kel03]. Das *Thermale Modell* kann somit zur professionellen Temperaturschätzung verwendet werden.

3.1.3 Energiegewahres hierarchisches Scheduling

Verschiedene Prozesse verfügen über verschiedene Energie-Charakteristika. Dies wird durch den unterschiedlichen Energieverbrauch, der durch die Prozesse aufgerufenen Instruktionen verursacht. Prozesse, die einen erhöhten Energiebedarf aufzeigen (*Hot Tasks*), erzeugen somit mehr thermische Energie als sparsame Prozesse (*Cool Tasks*). In Mehrprozessorsystemen kann dies zu thermischen Ungleichgewichten führen. Verarbeitet Prozessor C_1 zwei *Hot Tasks*, so verbraucht er mehr Energie als Prozessor C_2, welcher lediglich zwei *Cool Tasks* verarbeitet. Dies impliziert für Prozessor C_1 eine höhere Betriebstemperatur. Übersteigt diese Temperatur einen kritischen Wert (TCB), so müssen Unternehmungen getroffen werden, um C_1 wieder abzukühlen. Dies kann durch *DVS*, *Clock Scaling* oder *Throttling* (Leerlauf des Prozessors durch Zuteilung des Idle-Threads) geschehen. Jede dieser Maßnahmen verschlechtert die Leistung aufgrund einer geringeren resultierenden Instruktionsdichte. Dieser Leistungsverlust wird durch energiegewahres Scheduling verringert [Mer05]. Energie wird als erstklassige Resource in Entscheidungen des Schedulers einbezogen. Die Drosselung von Prozessor C_1 wird durch Migration eines *Hot Tasks* zu C_2 und eventueller Migration eines *Cool Task* von C_2 zu C_1 vermieden. Hierbei werden Speicherhierarchien wie in Kapitel 2.1.3 in die Scheduler-Strategie miteinbezogen.
Merkels Ansatz besteht aus einem Framework aus Energieschätzer (*Estimator*), Energieprofil (*Profiler*) und energiegewahren Scheduler (siehe Abbildung 3.1). Der *Estimator* schätzt die pro Prozessor konsumierte Energie anhand von Ereigniszählern. Anhand des prozessorspezifischen Energieverbrauchs können Rückschlüsse auf die Prozessortemperatur getroffen werden. Der *Estimator* wird durch den *Profiler* zu jedem Prozesswechsel und jedem Timer-Tick[1] aufgerufen. Hierdurch werden einerseits prozessorspezifi-

[1] Der Scheduler wird mindestens einmal pro Zeitscheibe durch eine Unterbrechung aufgerufen. Der Zeitpunkt dieser Unterbrechung wird als Timer-Tick bezeichnet.

Abbildung 3.1: Design Energiegewahrer Scheduler [Mer05]



scher Energieverbrauch sowie prozessspezifisches Energie-Profil berechnet. Der Scheduler kann anhand dieser Informationen Prozesse und Prozessoren entsprechend ihres Energieverbrauchs einordnen. Anhand dieser Einordnungen werden Prozesse, unter Einhaltung des thermalen Gleichgewichts und Minimierung von Drosselungen, auf die verschiedenen Prozessoren verteilt. Dies impliziert zusätzlich eine Minimierung des Energieverbrauchs sowie eine Maximierung der Leistung bei Prozessen unterschiedlicher Energiecharakteristik.
Alle Ansätze, welche sich der energiegewahren Verteilung widmen, beinhalten keine explizite Betrachtung von asynchronen Ereignissen. Die während einer asynchronen Unterbrechung konsumierte Energie wird immer dem unterbrochenen Prozess zugeordnet. Dies führt, wie man in Kapitel 6 noch genauer sehen kann, zu thermalen Ungleichgewichten. Dieser Missstand wird durch unsere Arbeit erstmalig behoben.

3.2 Verteilung asynchroner Unterbrechungen in Mehrprozessorsystemen

Strategien zur Verteilung asynchroner Unterbrechungen können durch Hardware oder Software realisiert werden. Bisher entwickelte Strategien werden nun vorgestellt.

3.2.1 Hardware

Die hardwareseitige Implementierung (Mechanismus) der Verteilung asynchroner Unterbrechungen im Mehrprozessorsystem wird durch lokale APICs und externe I/O APICs (Intel-Nomenklatur) realisiert (siehe Abschnitt 2.2.2). Intels 82093AA I/O APIC implementiert drei festverdrahtete Verteilungsstrategien [Int96, Int03]. Eine statische Verteilung, welche jeder Unterbrechung eine fixe CPU zuordnet, sowie zwei dynamische. Als dynamische Strategien stehen ein prioritätsbasiertes Verfahren sowie ein *Round Robin*-Verfahren (jeder Prozessor erhält zu gleichen Anteilen nacheinander die Unterbrechung) zur Verfügung. Die prioritätsbasierte Variante sendet Unterbrechungen an denjenigen lokalen APIC, dessen programmierbares Prioritätsregister (TPR - Task Priority Register) den niedrigsten Wert besitzt. Die Prioritätsregister werden allerdings von keiner Linux-Distribution zur Prioritätsvergabe verwendet [BC05].

3.2.2 Software

Die Wiederbeschreibbarkeit der Redirektions-Tabelle ermöglicht es, Strategien auf Softwareebene zu entwickeln, um die Verteilung der Unterbrechungen zu steuern. Für Linux findet sich eine solche Implementierung unter dem Namen *irqbalance* [IB008]. *Irqbalance* verwendet die beiden Modi *Power* und *Performance*. Der *Power*-Modus wird bei geringer Systemlast und geringer Unterbrechungsdichte δ_i verwendet. In diesem Modus werden alle Unterbrechungen an den ersten Prozessor[2] gesendet, um den restlichen Prozessoren die Ausnutzung ihrer Stromsparmodi zu ermöglichen. Überschreitet Systemlast und Unterbrechungsdichte einen gewissen Schwellwert, so schaltet *irqbalance* in den *Performance*-Modus. Hierbei wird eine Einteilung von Unterbrechungen in die fünf Klassen Netzwerk, Speicher, Media, Legacy und Timer getroffen. Unterbrechungen einer Klasse (zum Beispiel die Unterbrechungen verschiedener Netzwerk-Karten) werden über alle Prozessoren gleichmäßig verteilt. Eine praktische Evaluierung von *irqbalance* bezüglich Netzwerk-Durchsatz wurde durch Arndt erbracht [Arn07]. Arndt zeigt hierbei, dass die manuelle Bindung von Netzwerk-Adaptern und den sie bedienenden Unterbrechungen

[2] Prozessor mit der kleinsten Identifikationsnummer. Dies entspricht zumeist dem Prozessor dessen lokaler APIC ebenfalls die kleinste Identifikationsnummer besitzt.

zu gleichen NUMA-Knoten den Netzwerk-Durchsatz im Vergleich zu *irqbalance* steigert und diesem vorzuziehen ist.

3.2.3 Zusammenfassung

Alle bisher erbrachten Strategien zur Verteilung asynchroner Unterbrechungen verhindern lediglich ungleichmäßige Prozessorbelastung oder ermöglichen Energieeinsparungen bei geringer Last. Affinitäten von Unterbrechungen zu Prozessen werden nicht beachtet. Die effektive energiegewahre Verteilung sowie die Ausnutzung der Speicherhierarchie ist somit im Allgemeinen nicht gegeben.

Kapitel 4

Design

4.1 Problemanalyse

4.1.1 Thermische Ungleichgewichte

Thermische Ungleichgewichte in Multiprozessorsystemen können zu Leistungseinbußen führen. Resultierend aus der direkten Abhängigkeit des Energieverbrauchs eines Prozessors von der Art der von ihm ausgeführten Instruktionen können Prozesse, welche die gleiche Prozessorauslastung (Instruktionsdichte) erreichen, verschiedene Verbrauchswerte aufweisen [BWWK03, IM03]. Wie man in Kapitel 6 sehen kann, können Prozesse, welche 100% Prozessorlast erzeugen, zwischen *38 Watt* und *67 Watt* benötigen. Solche stark unterschiedliche Energiewerte implizieren, sofern sie statisch auf verschiedenen Prozessoren ausgeführt werden, unterschiedliche Prozessor-Temperaturen. Weiterhin können verschiedene Prozessoren mehr oder weniger günstig im Luftstrom der Kühlsysteme positioniert sein. Der Temperaturunterschied kann für unseren *67-Watt*-Testfall auf einem *2.2 GHz Pentium 4 Xeon Prozessor* allein aufgrund verschiedener Kühlbedingungen 7 $°C$ betragen (37 bis 44 $°C$). Einen durch thermische Ungleichgewichte implizierten Leistungsverlust erörtern wir durch ein Beispiel:

Prozessor	TDP	$\mathrm{T}_{\mathrm{throttle}}$	T_{max}
C_1	70 W	80 $°C$	79 $°C$
C_2	70 W	80 $°C$	80 $°C$

Tabelle 4.1: Exemplarische Prozessordaten

Es sei ein symmetrisches Multiprozessorsystem mit zwei Prozessoren C_1 und C_2 gegeben (siehe Tabelle 4.1). Den Regeln der Symmetrie entsprechend sei vom Hersteller

Prozess	E	E_{irq}	E_{rest}
A	70 W	90 W	50 W
B	40 W	0 W	40 W

Tabelle 4.2: Exemplarischer prozessspezifischer Energieverbrauch

für beide Prozessoren eine TDP von *70 Watt* und eine zulässige maximale Betriebstemperatur von $80\ °C$ gegeben. Bei Überschreiten dieser Temperatur wird eine prozessorinterne Unterbrechung ausgelöst, welche durch *Clock Scaling* eine Halbierung der Taktfrequenz initiiert, um die Prozessortemperatur herabzusetzen. Beide Prozessoren sind an einen Kühler gekoppelt. Prozessor C_1 liegt räumlich etwas näher zum Kühler als dies für Prozessor C_2 der Fall ist. Bei Anliegen von TDP erreicht Prozessor C_1 eine Maximaltemperatur von $79\ °C$. Prozessor C_2 übersteigt bei Anliegen von TDP die zulässige Maximaltemperatur. Es seien zwei Prozesse A und B gegeben (siehe Tabelle 4.2). Prozess A verbraucht *70 Watt* , Prozess B verbraucht $40\ Watt$. Beide Prozesse lasten den jeweilig zugeteilten Prozessor voll aus. Ein nicht energiegewahrer Scheduler könnte nun Prozess A Prozessor C_2 und Prozess B Prozessor C_1 zuweisen. Aus Sicht des Schedulers ist dieser Entscheidung nichts entgegenzusetzen. Der Scheduler kann weder die beiden Prozesse mit gleicher Systemlast noch die Prozessoren aufgrund ihrer unterschiedlichen Kühleigenschaften unterscheiden. Nach einiger Zeit wird nun Prozessor C_2, welcher den *70 Watt*-Prozess ausführt, seine maximale Betriebstemperatur überschreiten. *Clock Scaling* wird ausgelöst. Die Leistung des voll ausgelasteten Prozessors wird halbiert. Bei Einsatz eines energiegewahren Schedulers könnte dieser Leistungsverlust folgendermaßen vermieden werden: Aufgrund der schlechten Kühlbedingungen von Prozessor C_2 könnte diesem ein geringeres Energielimit von *65 Watt* zugewiesen werden. Prozessor C_1 würde in diesem Fall ein Energielimit von TDP, also *70 Watt* erhalten. Würde Prozess A nun zu Prozessor C_2 delegiert, so könnte ein energiegewahrer Scheduler feststellen, dass dessen Energieverbrauch überhalb des Energielimits des Prozessors liegt. Prozess A würde somit zu Prozessor C_1 migriert werden. Prozess B würde Prozessor C_2 zugeteilt bekommen. Nach einiger Zeit wird nun Prozessor C_1 seine maximale Temperatur von $79\ °$ erreichen. Diese liegt unterhalb der kritischen Temperatur. Der Prozessor wird nicht gedrosselt. Aufgrund des geringen Energieverbrauchs von Prozess B wird auch Prozessor C_2 nie eine Temperatur erreichen, welche ihn zum Drosseln seiner Taktfrequenz veranlasst.
Bisherige Implementierungen energiegewahrer Scheduler weisen keine explizite Assoziationen zwischen Unterbrechungs-Initiator und Unterbrechungs-Energieverbrauch auf. Dies führt bei datentransferlastigen Prozessen zu fehlerhaften Energieberechnungen. Dies kann wiederum zu thermalen Ungleichgewichten und implizierten Leistungsverlust führen. Zur Veranschaulichung greifen wir auf obiges Beispiel zurück: Der energiegewah-

re Scheduler verhindert die Drosselung von Prozessor C_2 durch Migration von Prozess A zu Prozessor C_1. Diese Entscheidung fordert eine exakte Berechnung des Energieverbrauchs von Prozess A. Würde der berechnete Energieverbrauch unterhalb des zugewiesenen Energielimits liegen, so würde die Migration verhindert werden. Prozessor C_2 würde überhitzen und müsste gedrosselt werden. Genau dieser Fall kann eintreten, wenn Prozess A einen gewissen Anteil seiner Energie während der Abarbeitung von Unterbrechungen verbrauchen würde. Exemplarisch könnte dies folgendermaßen aussehen:
Prozess A transferiert mittels asynchroner Unterbrechungen Daten über ein Netzwerk. Der Energieverbrauch, welcher während der ISR-Behandlung entsteht, beträgt *90 Watt*. Die Ausführungszeit von Prozess A sei gleich der Ausführungszeit der Unterbrechungen. Der restliche Energieverbrauch liegt für Prozess A somit bei *50 Watt*. Wird nun die Unterbrechungsenergie Prozess B zugeordnet, so errechnet der Scheduler folgende Energieverteilung: Prozess A verbraucht *50 Watt*; Prozess B verbraucht *65 Watt*. Keiner der beiden Prozesse übersteigt das Energielimit (*65 Watt*). Der energiegewahre Scheduler handelt analog zum normalen Scheduler. Prozessor C_2 überhitzt und wird gedrosselt.
Das thermale Gleichgewicht kann durch Verfahren, welche Unterbrechungsenergie nicht korrekt zuordnen, nicht realisiert werden. Diesem Missstand wird durch die von uns erbrachte Arbeit entgegengewirkt.

4.1.2 Verteilung asynchroner Unterbrechungen

Es existieren keine energiegewahren Strategien zur Verteilung asynchroner Ereignisse. Treffen asynchrone Unterbrechungen auf einem Prozessor an, so können diese, wie oben beschrieben, zu thermalen Ungleichgewichten führen. Trifft eine, durch Prozess A auf Prozessor C_1 initiierte Unterbrechung, auf einem entfernten Prozessor C_2 ein, so kann durch unser Verfahren die Unterbrechungsenergie E_{int} Prozess A zugeordnet werden. Der Energieverbrauch E_{ges} dieses Prozesses bezieht sich nun, entgegen dem Wissen des Schedulers, nicht ausschließlich auf Prozessor C_1. Wird A durch den Scheduler, aufgrund der Unterbrechungsenergie als *Hot-Task* eingestuft, so führt die Weg-Migration von A nicht zwingend zur Senkung der Temperatur des Prozessors C_1.

Weiterhin gibt es keine Strategien zur prozessspezifischen Verteilung asynchroner Unterbrechungen, die Speicherhierarchien in Multiprozessorsystemen effizient ausnutzen. Unterbrechungsinitiierende Prozesse eines NUMA-Knotens k erreichen, wie man in Kapitel 6 sehen kann, ihre maximale Leistung genau dann, wenn die initiierte Unterbrechung ebenfalls auf k eintrifft. Je weiter entfernt von k die Unterbrechung eintrifft, desto schlechter die Prozessleistung. Der Zugriff auf entfernte Datenblöcke erfordert einen nicht vernachlässigbaren Overhead in Zuge von Inter-Node-Zugriffen bzw. notwendiger Kopien der Blöcke hin zum lokalen NUMA-Knoten.
Die angeführten Leistungsverluste lassen sich nur durch Implementierung einer Assoziation von Unterbrechung und Initiator verhindern. Ein Unterbrechungsverteiler kann

diese Information zur prozessspezifischen Unterbrechungsverteilung zu Hilfe ziehen. Initiiert ein Prozess eine Unterbrechung, so wird diese Unterbrechung zum lokalen NUMA-Knoten des Initiators gesendet.
Die Zuordnung mehrerer Unterbrechungen zu einer oligopolen Prozessoren-Untermenge kann bei hoher Unterbrechungsdichte δ_i eine erhöhte Unterbrechungslatenz nach sich ziehen. Systeme, welche Unterbrechungspreemption unterstützen, implizieren eine höhere Latenz bezüglich unterbrochener Unterbrechungen (Nested Interrupts). Eine Unterbrechung i mit Latenz T_i, welche durch die Unterbrechungen $j...n$ der Latenzen $T_j...T_n$ unterbrochen wird, weist eine Gesamtlatenz T_{g_p} von

$$T_{g_p} = T_i + \sum_{k=j}^{n} T_k \tag{4.1}$$

auf. Dieser Latenzanstieg kann durch Zuordnung der Unterbrechungen $j...n$ zu anderen Prozessoren verhindert werden.

Nichtpreemptive Systeme steigern im Gegensatz hierzu die Latenz wartender Unterbrechungen. Werden vor einer anstehenden Unterbrechung i die Unterbrechungen $j...n$ der Latenzen $T_j...T_n$ bearbeitet, so erweitert sich die Latenz $T_{g_{np}}$ der Unterbrechung i zu

$$T_{g_{np}} = T_i + \sum_{k=j}^{n} T_k \tag{4.2}$$

Auch dieser Latenzanstieg würde durch dedizierte Unterbrechungszuweisung unterschiedlicher Unterbrechungen zu verschiedenen Prozessoren verhindert.

4.2 Allgemeiner Lösungsansatz

Unser Lösungsansatz basiert auf vier Eckpfeilern: Exakter Energiegewahrer Scheduler (*Scheduler*), Assoziator (*I-Associator*), Unterbrechungsenergieschätzer (*I-Estimator*) und energie- und NUMA-gewahrer Unterbrechungsverteiler (*I-Balancer*). Der *Scheduler* ist ein Derivat des durch Merkel vorgestellten Schedulers [Mer05], welcher hingehend einer korrekten Zuordnung und Verteilung der Unterbrechungsenergie erweitert wurde. Der *I-Associator* stellt die Assoziation zwischen Unterbrechungen, Peripheriegeräten, Dateien und Systemaufrufen bereit. Der *I-Estimator* ist fester Bestandteil der Unterbrechungsbehandlung. Er misst, sofern dies notwendig ist, die pro Unterbrechung konsumierte Energie. Die Verteilung der Unterbrechungen wird durch den *I-Balancer* realisiert. Aufbau und Zusammenspiel dieser Instanzen werden wir nun erörtern:

4.3 I-Associator

Für den Nutzer spielt es keine Rolle, ob ein unterbrechungsbasierter Datentransfer stattfindet. All dies wird durch die Schnittstelle *Systemaufrufe* abstrahiert und durch Gerätetreiber implementiert. Selbst für Kernel-Prozesse (mit Ausnahme derer, die sich um die Verwaltung der Unterbrechungen beschäftigen) ist es irrelevant, welche Unterbrechung zu welchem Gerät assoziiert wurde. Prozesse kennen nur Dateien. Somit kennen sie weder Geräte, schon gar nicht asynchrone Unterbrechungen. Zur korrekten Energieabrechnung asynchroner Unterbrechungen muss folglich innerhalb des Kernels eine entprechende Zuordnung der Unterbrechungen zu den sie initiierenden Prozessen stattfinden. Dies geschieht durch den *I-Associator*.

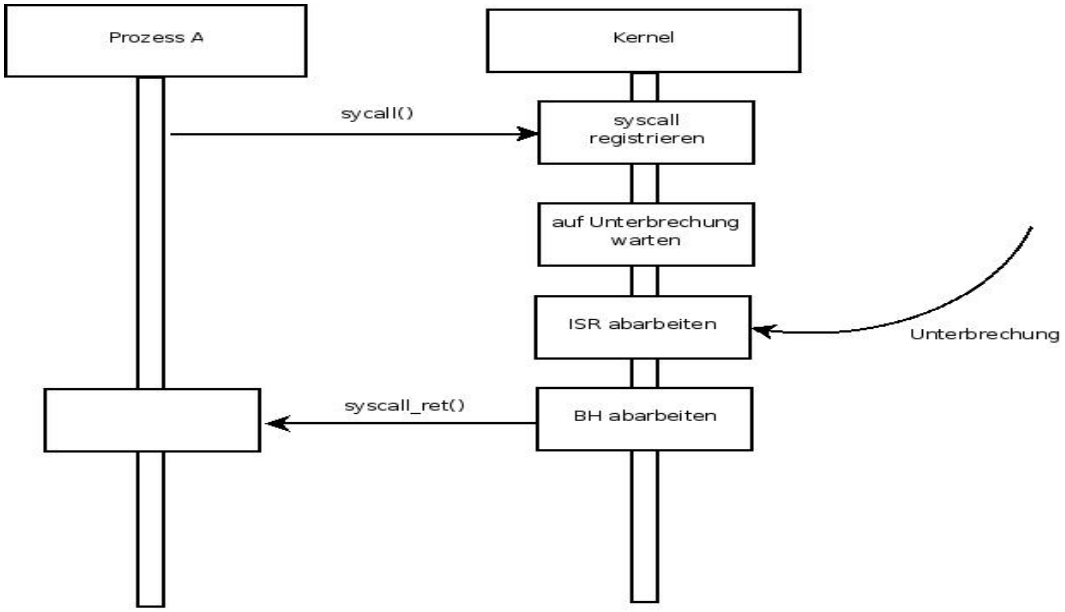
Eine solche Assoziation steht im Gegensatz zum ursprünglichen Design asynchroner Unterbrechungen. Peripheriezugriff findet für User-Prozesse durch Systemaufrufe (read, write, sendfile u.s.w.) statt. Bis auf einige Ausnahmefälle (z.B.: Boot-Vorgang) trifft dies auch auf Kernel-Prozesse zu. Aus Gründen der Sicherheit und Implementierungsfreundlichkeit für Programme werden periphere Daten durch Dateisysteme in Dateien gruppiert. Es genügt die Parameterübergabe eines eindeutigen Identifikations-Datums (Name, ID) an den entprechenden Systemaufruf, um auf die Daten von Dateien zuzugreifen. Wie, wann und wo diese Dateien bereitsgestellt werden, ist aus Nutzer-Sicht bei erfolgreichem Datentransfer irrelevant (bei Fehlschlagen des Datentransfers können anhand von Rückgabewerten nutzerspezifische Maßnahmen getroffen werden). Die Transparenz betrifft auch die in 2.2.1 erläuterten Mechanismen zur Durchführung des Datentransfers.

Zur Unterbrechungsbehandlung nutzten Betriebssysteme eine indizierte Zeiger-Tabelle. Diese beinhaltet für jede Unterbrechungsnummer einen Zeiger zur Speicheradresse der Funktion, welche die Unterbrechung abarbeitet. Die Zuweisung dieser Adressen sowie die Zuweisung der Unterbrechungsnummern (IRQ-Line – Interrupt Request Line) erfolgt in manchen Fällen statisch (z.B.: Timer); in anderen Fällen dynamisch (z.B.: PCI-Geräte). Außer der indizierten Zeiger-Tabelle liegt keine einheitliche Assoziation der Unterbrechungen zu den sie initiierenden Instanzen vor. Der *I-Associator* stellt eine einheitliche Assoziation wie folgt bereit:

Jedes Gerät besitzt eine systemweit einzigartige Identifikationsnummer (*Dev-ID*). Diese *Dev-ID* wird dem Betriebssystem durch den Gerätetreiber während seines Initialisierungsprozesses übergeben. Im Falle eines Gerätes, welches seinen Datentransfer anhand asynchroner Unterbrechungen durchführen kann, wird die *Dev-ID* bei Registrierung der Unterbrechungsnummer übergeben. Bei erfolgreicher Registrierung der Unterbrechungsnummer verfügt das Betriebssystem somit über eine explizite Assoziation zwischen Gerät und Unterbrechungsnummer. Diese Assoziation wird durch den *I-Assoziator* in zwei Tabellen gesichert. Eine Geräte-Unterbrechungs-Tabelle, welche für alle Geräte mit erfolgreich abgeschlossener Registrierung einer Unterbrechungsnummer diese Unterbrechungsnummer bereit hält. Eine weitere Tabelle, um zu jeder Unterbrechung das entsprechende

Gerät heraus zu finden. Der Ansatz zweier Tabellen priorisiert Leistung gegenüber sparsamer Speicherverwendung. Nehmen wir an, es stände lediglich eine Tabelle zur Verfügung, welche jeder aus N denkbaren *Dev-IDs* eine Unterbrechungsnummer zuordnet. Die Suche der Unterbrechungsnummer einer bekannten *Dev-ID* x würde nun an der x-ten Stelle der Tabelle stehen. Der Tabelle müsste lediglich x als Index übergeben werden. Die Suche wäre nach einem Schritt beendet und läge innerhalb der Komplexitätsklasse $O(1)$. Die Suche der *Dev-ID*, welche eine Unterbrechung i ausgelöst hat, ist nun weitaus komplexer. Im Worst-Case-Szenario müsste jeder der N Tabelleeinträge nach übereinstimmender Unterbrechungsnummer durchsucht werden, um die entsprechende *Dev-ID* zu ermitteln, insbesondere für Fälle, in denen der Unterbrechung i keine *Dev-ID* zugeordnet wurde. Die Suche liegt somit innerhalb der Komplexitätsklasse $O(N)$. Wird die *Dev-ID* durch x Bit dargestellt, so ergibt sich $N = 2^n$. Die Unterbrechungsbehandlung muss schnell geschehen[1]. Während der Abarbeitung können wichtigere Ereignisse aufgrund ihrer Maskierung eine zu hohe Latenz aufweisen. Innerhalb einer ISR ist somit Geschwindigkeit zu Lasten von Speicherverbrauch vorzuziehen.

Abbildung 4.1: Behandlung unterbrechungsinitiierender Systemaufrufe



Die Initiierung einer Unterbrechung i durch einen Prozess A findet in monolithischen Betriebssystemen über so genannte Systemaufrufe statt. Unterbrechungsinitiierende Systemaufrufe sind solche, welche sich um Datentransfer zwischen Prozessen und peripheren

[1] auf $32 - Bit$-Systemen ist $N \geq 4294967296$

Geräten kümmern. In UNIX-artigen Systemen wird dies durch Übergabe einer Dateireferenz signalisiert. Andere Systeme erhalten ähnliches. Anhand der Dateireferenz lässt sich über das virtuelle Dateisystem diejenige *Dev-ID* herausfinden, die eindeutig das Gerät identifiziert, welches die Daten der Datei bereitstellt. Anhand dieser Methode lässt sich herausfinden, welche Unterbrechung Prozess A voraussichtlich initiieren wird:

Der *I-Associator* überprüft bei jedem initiierten Systemaufruf, ob diesem als Parameter eine Dateireferenz eines wie oben registrierten Gerätes übergeben wird. Ist dies der Fall, so wird ein entsprechender Eintrag im PCB des Prozesses gesetzt. Zusätzlich wird die Unterbrechungsnummer im PCB gesichert. Können die entsprechenden Daten, welche durch den Systemaufruf transferiert werden, tatsächlich nur durch eine Unterbrechung bereitgestellt werden, so wird der Prozess durch den Systemaufruf in einen schlafenden Zustand überführt. Bei Abschluss des Datentransfers erhält der Scheduler die Information, dass der Prozess wieder eingegliedert werden kann. Der Scheduler teilt dem Prozess die durch ihn initiierte Unterbrechungsenergie zu. Bei Wiedereingliederung werden die restlichen Instruktionen des unterbrochenen Systemaufrufs abgearbeitet. Der PCB-Eintrag wird gelöscht. Der Datentransfer ist abgeschlossen. Können die Daten aufgrund interner Pufferung ohne Unterbrechung transferiert werden, so wird der Systemaufruf unblockiert durchgeführt. Der PCB-Eintrag kann umgehend wieder gelöscht werden. Eine schematische Darstellung der durch uns erweiterten Systemaufrufe findet sich in Abbildung 4.1.

4.4 I-Estimator

Bisherige Energieabschätzungen ordnen die während einer ISR konsumierte Energie stets dem gerade laufenden Prozess zu. Das Triggern der Energieabschätzungen findet stets durch den Scheduler statt. Zur korrekten Zuordnung der Unterbrechungsenergie muss diese zu Zeiten von Unterbrechungs-Aktivitäten geschätzt werden. Dies realisieren wir anhand von Ereigniszählern innerhalb der systeminternen Unterbrechungsbehandlung durch den *I-Estimator*. Der *I-Estimator* ordnet jeder Unterbrechung ihren Energieverbrauch zu. Dies erfolgt durch Auslösung von Energieabschätzungen bei Eintritt und Austritt der Unterbrechungsbehandlung. Der Energieverbrauch ΔE_i einer Unterbrechung i ergibt sich aus den vorliegenden prozessorspezifischen Energieverbrauchswerten (siehe Kapitel 3.1.3) des Eintrittszeitpunktes der Unterbrechung (E_{enter}) und des Austrittszeitpunktes der Unterbrechung (E_{exit}) zu

$$\Delta E_i = E_{exit} - E_{enter}. \tag{4.3}$$

Die konsumierte Energie ΔE_i muss in die Abrechnung des unterbrochenen Prozesses A_{int} sowie des Initiators A_{resp} einfließen. Dies erfolgt über die *Irq-Energy-Liste* (IEL). Tritt eine Unterbrechung i auf einem Prozessor ein, so wird die entsprechende ISR abgearbeitet. Anschließend wird der konsumierte Energiewert in der IEL gesichert und von

Prozess A_{int} abgezogen. Tritt die Unterbrechung i mehrfach vor Wiedereinlagerung des Prozesses A_{resp} durch den Scheduler ein, so wird die Messung wiederholt. Der neue gemessene Wert ΔE_i wird nun zur IEL addiert. Bei Wiedereinlagerung von A_{resp} wird der entprechende Energiewert der IEL durch den Scheduler dem verantwortlichen Prozess A_{resp} angerechnet. Würden die Energiewerte nicht gepuffert, so würde A_{resp} lediglich der Energiewert der letzten Unterbrechung zugeordnet. Dies ist nicht richtig, da die Unterbrechungsfrequenz weitaus höher sein kann als die Frequenz, mit der A_{resp} eingeplant wird. Nach Zuordnung der Energie wird der entprechende Eintrag der IEL auf Null zurückgesetzt.

Energieabschätzung und Behandlung geschachtelter Unterbrechungen des *I-Estimator* werden in den beiden folgenden Unterabschnitten erörtert.

4.4.1 Energieabschätzung

Die durch den *I-Estimator* durchgeführte Energieabschätzung wird anhand von Ereigniszählern implementiert (siehe Kapitel 3.1.1). Eine (geschachtelte) Unterbrechungsbehandlung kann während ihrer Abarbeitung den zugewiesenen Prozessor entzogen bekommen; auf keinen Fall kann die Abarbeitung jedoch zu einem anderen Prozessor migriert werden. Hierzu müsste der Scheduler in die Unterbrechungsbehandlung involviert sein. Dies ist nicht der Fall. Sei T_i die Verarbeitungszeit einer Unterbrechung i, welche durch Prozessor C abgearbeitet wird, so verbraucht die Unterbrechung i genau diejenige Energie, welche durch Prozessor C während der Zeitspanne T_i verbraucht wird. Dieser Energieverbrauch wird gemessen und der Unterbrechung zugeordnet.

4.4.2 Sporadische Unterbrechungsinitiierung

Prozesse, welche sich sporadisch unterbrechungsinitiierender Systemaufrufen bedienen, können Unterbrechungsenergie unterbrechungslastiger Prozesse zugeordnet bekommen. Dies beruht auf der Tatsache, dass die Zuordnung der Unterbrechungsenergie zu einem Prozess A nicht direkt nach jeder Unterbrechung geschieht. Die Zuordnung der Unterbrechungsenergie wird später durch den *Profiler* durchgeführt. Dieser wird durch den *Scheduler* aufgerufen.

Sei die Frequenz φ_A (bzw. φ_B) die Frequenz, mit der Unterbrechungsenergie durch den *Profiler* zu Prozess A (bzw. Prozess B) zugeordnet werden kann. Die Frequenzen mit denen die Prozesse A und B Unterbrechungen initiieren seien f_A und f_B. Ohne Beschränkung der Allgemeinheit gelte:

$$\varphi_A \approx \varphi_B. \tag{4.4}$$

Ist nun $f_A \gg f_B$, so wird beiden Prozessen in etwa die gleiche Unterbrechungsenergie zugeordnet. Dies ist nicht gerechtfertigt, da Prozess B hierbei zwangsläufig solche Ener-

gie angerechnet bekommt, die durch die Unterbrechungsbehandlung der durch Prozess A initiierten Unterbrechungen errechnet wurde.

Um zu vermeiden, dass Unterbrechungsenergie nicht falsch zugeordnet wird, sichern wir die Anzahl der Systemaufrufe, die Anzahl der pro Prozess angerechneten Unterbrechungen und die Anzahl der Unterbrechungsenergie-Zuordnungen durch den *I-Estimator*. Anhand dieser Werte und anhand des Verhältnisses aus angerechneter Unterbrechungszeit *itime* zur Summe aus Nutzerzeit *utime* und Systemzeit *stime* errechnen wir die Wahrscheinlichkeit θ mit der der Prozess tatsächlich die Unterbrechungen initiiert hat, deren Energie ihm angerechnet werden soll. Die Werte der IEL werden im Falle einer geringen Wahrscheinlichkeit θ nicht gelöscht und werden dem unterbrechungslastigen Prozess zugeordnet. Die Berechnung von θ ist für die einzelnen Unterbrechungen unterschiedlich.

4.4.3 Geschachtelte Unterbrechungen

Wird eine Unterbrechung während ihrer Bearbeitung in einem preemptiven Betriebssystem durch eine andere Unterbrechung unterbrochen, so sprechen wir von *geschachtelten Unterbrechungen*. Zur korrekten Energieverbrauchsschätzung solcher geschachtelter Unterbrechungen verwendet der *I-Estimator* eine interne zweidimensionale Tabelle (ICT, I-Counter-Tabelle). Diese enthält für jede Unterbrechungsebene e_i die Werte der zur Energieschätzung benötigten Ereigniszähler. Zu Beginn der Unterbrechungsbehandlung werden diese Zähler ausgelesen und in der Tabelle gesichert. Vor Beendigung der Unterbrechungsbehandlung werden selbige Zähler erneut ausgelesen. Die gesicherten Werte werden durch die Differenzen der neuen und alten Zählerwerte ersetzt. Anschließend wird der Energieverbrauch anhand Formel 1.3 berechnet und wie oben beschrieben zur IEL addiert. Befindet sich die Unterbrechungsbehandlung in tiefster Ebene (keine geschachtelte Unterbrechung), so muss nichts weiter getan werden. Bei höher liegenden Ebenen werden die Zählerdifferenzen zu allen tiefer liegenden Ebenen der ICT hinzuaddiert. Somit ist gewährleistet, dass jeder tiefer liegenden Ebene keine Energie von höher gelegenen Ebenen angerechnet wird. Bei konstanter Kardinalität k der Ereigniszähler liegt dieser Vorgang innerhalb der Komplexitätsklasse $O(k) \subseteq O(1)$. Im Gegensatz zur Anzahl N aller möglichen *Dev-IDs* ist k wesentlich kleiner[2].

4.5 Exakter Energiegewahrer Scheduler

Unser Exakter Energiegewahrer Scheduler ist eine Erweiterung des in 3.1.3 vorgestellten energiegewahren Scheduler. Auf die Erweiterungen des *Estimators*, des *Profilers* und des *Schedulers* gehen wir nun ein.

[2]Für unser Testsystem gilt $k < 10$

4.5.1 Estimator

Der *Estimator* ist diejenige Instanz, welche dem *Profiler* die prozessor- und prozessspezifischen Energiewerte ΔE bereitstellt. Die Energiemessung wird um die Behandlung des Energieverbrauchs asynchroner Unterbrechungen erweitert. Die durch *I-Estimator* und *I-Associator* zugewiesenen und gemessenen Energiewerte werden dem *Estimator* bereitgestellt. Während der prozessspezifischen Energieberechnung prüft der *Estimator*, ob entsprechende Einträge im PCB gesetzt wurden und rechnet gegebenfalls den entsprechenden Energiewert der IEL dem Prozess zu.

4.5.2 Profiler

Merkels *Profiler* ermittelt prozessor- und prozessspezifische Energie-Profile [Mer05]. Die prozessspezifischen Profile werden durch Auslesen prozessorspezifischer Zählerregister errechnet. Dies involviert bereits die prozessorspezifische Energiezuordnung asynchroner Unterbrechungen. Dies ist für prozessspezifische Energie-Profile nicht der Fall. Merkels prozessspezifische Energie-Profile errechnen sich aus dem so genannten „exponentiellen Schnitt“ vorangegangener Energieverbrauchsdaten [GCW95, PBB98, FM02]. Das Energie-Profil x_n der Sampling-Periode n ergibt sich aus der Superperiode T, Standard-Scheduler-Zeitscheibe τ und Energieverbrauch v_n während einer Sampling-Periode τ_n zu

$$x_n = \frac{\tau}{T} \cdot v_n + \frac{T - \tau_n}{T} \cdot x_{n-1}. \tag{4.5}$$

Der Energieverbrauch v_n berücksichtigt nicht die während der Unterbrechungsbehandlung konsumierte Energie $v_{n_{ia}}$, die durch den Prozess initiiert wurde. Zusätzlich involviert v_n diejenige Energie $v_{n_{is}}$, die während einer möglichen Unterbrechung des Prozesses verbraucht wird. Die tatsächlich konsumierte Energie v_n^* ergibt sich zu

$$v_n^* = v_n + v_{n_{ia}} - v_{n_{is}}. \tag{4.6}$$

Die Sampling-Periode τ_n muss analog hierzu um die Zeitspanne $\tau_{n_{ia}}$ erhöht und um $\tau_{n_{is}}$ erniedrigt werden. Dies führt zur Sampling-Periode τ_n^* mit

$$\tau_n^* = \tau_n + \tau_{n_{ia}} - \tau_{n_{is}}. \tag{4.7}$$

Das in Formel 4.5 ermittelte Energie-Profil führt nun zu dem unterbrechungsenergiegewahren Energie-Profil x_n^* mit

$$x_n^* = \frac{\tau}{T} \cdot v_n^* + \frac{T - \tau_n^*}{T} \cdot x_{n-1}^* \tag{4.8}$$

$$= \frac{\tau}{T} \cdot (v_n + v_{n_{ia}} - v_{n_{is}}) + \frac{T - \tau_n - \tau_{n_{ia}} + \tau_{n_{is}}}{T} \cdot x_{n-1}^* \,. \tag{4.9}$$

Um ein zur Sampling-Periode τ^* vergleichbares Energieprofil x'_n mit

$$x'_n = \frac{\tau}{T} \cdot v'_n + \frac{T - \tau'_n}{T} \cdot x'_{n-1}. \tag{4.10}$$

zu ermitteln, welches entstände, wenn der Prozess bei konstantem Energieverbrauch v_n während der Sampling-Periode τ^* ausgeführt würde, berechnen wir den Energieverbrauch v'_n zu

$$v'_n = \frac{\tau^*}{\tau} \cdot v_n. \tag{4.11}$$

Nach Einsetzen von v'_n und $\tau' = \tau^*$ in Formel 4.10 erhalten wir das probabilistische Energieprofil x'_n zu

$$\begin{aligned} x'_n &= \frac{\tau}{T} \cdot (\frac{\tau^*}{\tau} \cdot v'_n) + \frac{T - \tau^*_n}{T} \cdot x'_{n-1} \\ &= \frac{\tau^*}{T} \cdot v'_n + \frac{T - \tau^*_n}{T} \cdot x'_{n-1}. \end{aligned} \tag{4.12}$$

Der *Profiler* speichert die Energie-Profile x^*_n, x'_n und x_n im PCB eines jeden Prozesses. Dies ist für die Entscheidungen des Schedulers von enormer Bedeutung. Die Einheit der Energie-Profile ist das Watt.

4.5.3 Scheduler

Der Scheduler führt die energiegewahre Verteilung der Prozesse unter Einhaltung des thermischen Gleichgewichtes sowie der Lastverteilung aus (siehe Abbildung 4.2). Hierzu verwendet der Scheduler die Energieprofile x^*_n und x'_n eines jeden Prozesses, ein prozessorspezifisches Energielimit, welches nicht überschritten werden darf, sowie die Differenz δ_{x_n} aus

$$\delta_{x_n} = x^*_n - x'_n. \tag{4.13}$$

Liegen thermische Ungleichgewichte vor, so zieht der Scheduler Prozesse von heißen Prozessoren zu kälteren Prozessoren (siehe Abbildung 4.2). Die Temperatur eines jeden Prozessors ist, aufgrund des zugrundeliegenden thermalen Modells, durch sein prozessorspezifisches Energieprofil ersichtlich. Um die Temperatur eines heißen Prozessors zu erniedrigen muss, folglich ein *Hot-Task* zu einem anderen Prozessor migriert werden.

Verbraucht ein Prozess keine Unterbrechungsenergie , so gilt:

$$\begin{aligned} & x'_n \approx x^*_n \\ \Leftrightarrow \quad & \delta_{x_n} \approx 0. \end{aligned}$$

Überschreitet x^*_n das Energielimit, so wird der Prozess als *Hot-Task* eingestuft und migriert.

Verbraucht ein Prozess mehr Energie während der Behandlung der durch ihn initiierten Unterbrechungen als während der Abarbeitung seines Programm-Codes, so gilt:

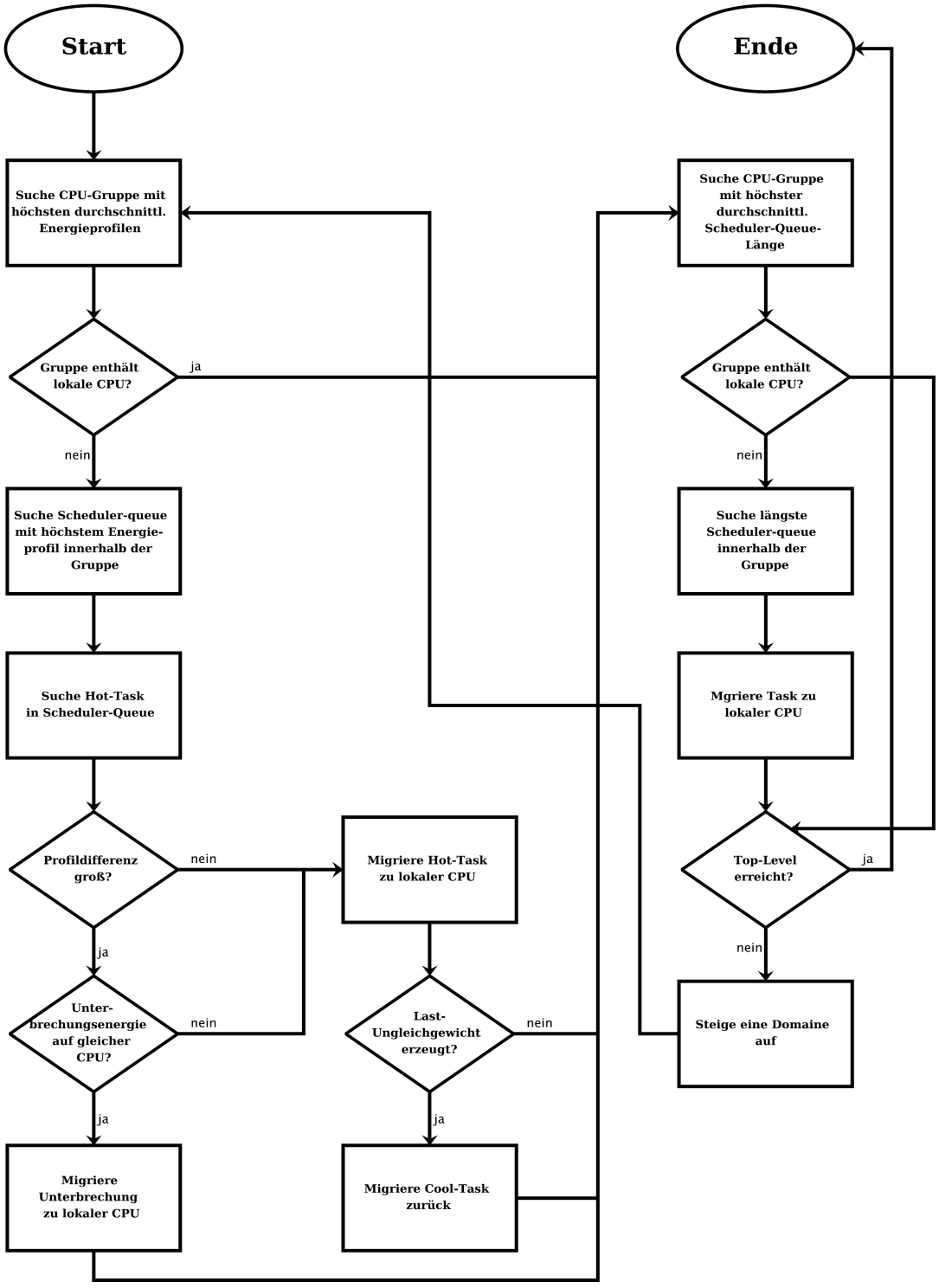
$$\begin{aligned} & x_n^* > x'_n \\ \Leftrightarrow \quad & \delta_{x_n} > 0. \end{aligned}$$

Wird die Unterbrechungsenergie auf dem selben Prozessor C verbraucht, welcher den Prozess ausführt, so wird dies dem *I-Balancer* signalisiert. Der *I-Balancer* migriert folgend die Unterbrechung zum nächsten Prozessor innerhalb des selben NUMA-Knotens. Prozessor C verbraucht nun lediglich die durch das Energieprofil x_n approximierte Energie. Ist diese unterhalb des Energielimits, so muss der Prozess nicht migriert werden.

Wird die Unterbrechungsenergie bereits auf einem entfernten Prozessor verbraucht, so approximiert x_n die Energie, die durch den Prozess auf Prozessor C verbraucht wird. Liegt diese oberhalb des Limits, so wird der Prozess migriert.

Überschreitet ein Prozessor sein Energielimit, während er lediglich einen Prozess ausführt, so führt auch dies zur Überhitzung des Prozessors. Diesem Missstand wird durch die Erweiterung von Merkels *Hot-Task-Migration* entgegengewirkt. Überschreitet das Energieprofil x_n^* das Energielimit, so wird wie oben entschieden: Treten die durch den Prozess initiierten Unterbrechungen auf dem selben Prozessor ein, welcher den Prozess durchführt, so wird die Unterbrechung zum nächsten Prozessor innerhalb des selben Knotens migriert. Werden die Unterbrechungen bereits durch einen anderen Prozessor abgearbeitet, so wird der Energieverbrauch des Prozesses durch sein Energieprofil x_n beschrieben. Der Prozess muss nur migriert werden, wenn x_n das Energielimit überschreitet.

Abbildung 4.2: Energie- und NUMA-gewahre Lastverteilung



4.6 I-Balancer

Ziel des *I-Balancer* ist die energiegewahre und NUMA-gewahre Verteilung asynchroner Unterbrechungen auf Mehrprozessorsystemen. Grundlage zur Verteilung sind die durch den *I-Associator* gelieferten Zuordnungen zwischen Prozessen und Unterbrechungen.

4.6.1 Energiegewahre Unterbrechungsverteilung

Zur energiegewahren Unterbrechungsverteilung muss die Unterbrechungsenergie ΔE_{int} in die Entscheidungen des energiegewahren Schedulers einbezogen werden. Überschreitet ein Prozessor sein Energielimit aufgrund eines hohen Betrages seiner Unterbrechungsenergie, so muss die Unterbrechung zu einem kühleren Prozessor migriert werden. Dies wird dem *I-Balancer* durch den Scheduler, wie oben beschrieben, signalisiert. Die Unterbrechung wird durch den *I-Balancer* migriert.

4.6.2 NUMA-gewahre Unterbrechungsverteilung

Findet ein unterbrechungsinitiierender Systemaufruf durch Prozess A auf Prozessor C_A statt, so ermittelt der *I-Balancer* anhand der Redirektions-Tabelle des entsprechenden IO APIC die Prozessor-Nummer id, zu welcher die Unterbrechung i gesendet wird. Ist Prozessor C_{id} Teil des NUMA-Knotens, auf welchem A ausgeführt wird, so muss nichts weiter unternommen werden. Ist dies nicht der Fall, so wird der entsprechende Eintrag der Redirektions-Tabelle durch die Prozessor-Nummer von Prozessor C_A ersetzt. Im Allgemeinen[3] wird die Unterbrechung i nun zu einem Prozessor desjenigen NUMA-Knotens gesendet, welcher den Initiator-Prozessor C_A beinhaltet.

4.6.3 Unterbrechungs-Ping-Pong

Eine anhand dieser Strategien durchgeführte Verteilung asynchroner Unterbrechungen kann zu einer Abart führen, welche wir als *Unterbrechungs-Ping-Pong* (IPP - Interrupt Ping Pong) bezeichnen. Die Migration einer Unterbrechung wird durch Beschreiben der Redirektions-Tabelle des IO-APIC durchgeführt. Die Beschreibung kann in wenigen Schritten mittels ICC-Bus durchgeführt werden. Aus diesem Gesichtspunkt ist IPP mit keinem hohen Overhead verbunden. Die Beschreibung der Redirektions-Tabelle muss im Mehrprozessorsystem synchronisiert erfolgen. Das hierfür nötige Locking der Redirektions-Tabelle sowie das Sperren der zu migrierenden Unterbrechungslinie erzeugt einen Overhead, welcher nicht vernachlässigbar ist (extrem hohe Migrationsfrequenzen führen für

[3] Würde vor Eintritt der Unterbrechung i ein Prozess B auf einem entfernten NUMA-Knoten einen Systemaufruf starten, welcher die selbe Unterbrechung i initiiert, so wird i zu diesem entfernten Knoten gesendet.

unser in Kapitel 6 evaluiertes Testsystems schlichtweg zum Einfrieren oder Absturz des Systems). Sei

$$M = \bigcup \{A_j |\ 0 \leq j \leq n-1\} \tag{4.14}$$

eine Menge aus n Prozessen, welche die gleiche Unterbrechung i implizieren und auf unterschiedlichen Knoten ausgeführt werden. Sei nun f_j die Frequenz, mit welcher Prozess $A_j \in M$ die Unterbrechung i initiiert. Ohne Beschränkung der Allgemeinheit sei

$$F = \bigcup_{k=0}^{n} \{f_j |\ A_j \in M\} \tag{4.15}$$

und $(F, \leq)$ eine geordnete Menge. So beträgt die minimale Frequenz $f_{min_{pp}}$, mit welcher die Unterbrechung i migriert wird[4]:

$$f_{min_{pp}} = f_{n-2}. \tag{4.16}$$

Hieraus ergibt sich für unsere NUMA-gewahre Unterbrechungsverteilung, dass die Migrationsfrequenz einer Unterbrechung, die durch eine Mehrzahl auf unterschiedlichen Knoten ausgeführte Unterbrechungsinitiatoren entsteht, mindestens so hoch ist wie die zweithöchste Frequenz, mit der die Unterbrechung initiiert wird. Zur hohen Migrationsfrequenz genügen somit zwei Unterbrechungsinitiierende Prozesse, welche mit hoher Frequenz Unterbrechungen initiieren.

Liegt $f_{min_{pp}}$ nun überhalb eines gewissen Schwellwertes t_{pp} (Ping-Pong-Threshold), so sprechen wir von Unterbrechungs-Ping-Pong. Der durch IPP implizierte Migrations-Overhead wird durch die *I-Balancer*-Modi *Default* und *Performance* verhindert. Welcher Modi verwendet wird, ist durch den Benutzer festzulegen. Die Modi werden im Folgenden erörtert.

4.6.4 Default Mode

Überschreitet die Migrations-Frequenz der Unterbrechung i den Schwellwert t_{pp}, so liegt IPP vor. Befindet sich der *I-Balancer* hierbei im *Default Mode*, so wird die Unterbrechung i zu demjenigen NUMA-Knoten k migriert, dessen Prozesse die meiste Unterbrechungsbearbeitungszeit der Unterbrechung i aufweisen. Der *I-Balancer* wird nun für eine benutzerdefinierte Zeitspanne T_{pp} ausgeschaltet. Nach Ablauf von T_{pp} wird der *I-Balancer* wieder eingeschaltet. Der *Default-Mode* realisiert somit die Verhinderung von IPP. Dieser erzeugt hohe Prozessleistungen, wenn sich viele bis alle der höherfrequenten Unterbrechungsinitiatoren auf dem selben NUMA-Knoten befinden. Dies kann durch explizite

[4]Dies ergibt für zwei Prozesse mit den Frequenzen $f_0 = 100\ Hz$ und $f_1 = 1000\ Hz$ eine minimale Migrationsfrequenz $f_{min_{pp}} = f_0 = 100\ Hz$

Zuordnung von Prozessen zu Prozessoren durch den Nutzer stattfinden. Initiatoren, welche sich auf anderen NUMA-Knoten befindet, werden in ihrer Leistung im *Default Mode* beschränkt. Die Abarbeitung der durch sie initiierten Unterbrechungen ist ineffizienter als die Abarbeitung von Unterbrechungen, welche durch die auf NUMA-Knoten k befindlichen Prozesse initiiert werden. Maximale Prozessleistung aller Initiatoren ermöglicht der *Performance Mode*.

4.6.5 Performance Mode

Befindet sich der *I-Balancer* während IPP im *Performance Mode*, so wird die entsprechende Unterbrechung i analog zum *Default Mode* zu demjenigen NUMA-Knoten k migriert, dessen Prozesse die meiste Unterbrechungsbearbeitungszeit der Unterbrechung i aufweisen. Initiatoren der Unterbrechung i, welche sich auf anderen NUMA-Knoten befinden werden im Gegensatz zum *Default Mode* zusätzlich zum Knoten k migriert. Dies impliziert die höchste Prozessleistung aller Initiatoren. Der *I-Balancer* wird nun für die Zeitspanne T_{pp} ausgeschaltet. Während dieser Zeitspanne wird die Migration der durch den *I-Balancer* migrierten Prozesse zu Prozessoren anderer Knoten unterbunden. Nach Ablauf von T_{pp} wird der *I-Balancer* wieder eingeschaltet. Die durch den *I-Balancer* initiierte Unterbindung von Prozessmigrationen wird aufgehoben.

4.7 Bottom Halves

Wie bereits in Kapitel 2.2.1 gezeigt wurde, involvieren asynchrone Unterbrechungen in vielen Fällen Auslösungen unkritischer Operationen (*Bottom Halves*), welche außerhalb der ISRs behandelt werden. Diese *Bottom Halves* müssen zusätzlich durch *I-Estimator* und *I-Associator* betrachtet werden. Im *I-Associator* müssen während einer ISR, welche durch eine Unterbrechung i initiiert wird, die von ihr aktivierten *BHs* der Unterbrechung i zugeordnet werden. Dies geschieht analog zu 4.3. Die Energie, welche durch die nun zugeordneten *BHs* verbraucht wird, muss durch den *I-Estimator* gemessen und dem entsprechenden Wert der IEL angerechnet werden. Dies geschieht analog zu Kapitel 4.4.

Aufgrund der von uns durchgeführten Tests des im folgenden Kapitel beschriebenen Testsystems hat sich die explizite Betrachtung von *BHs* als nicht praktikabel erwiesen. Dies ist durch eine Eigenart des 2.6er Linux-Kernels, welcher durch uns erweitert wurde, begründet. Nach Eintritt einer Unterbrechung wird an die Adresse der zugehörigen Kernel-ISR gesprungen. Dort werden die treiberspezifischen ISRs herausgefunden und behandelt. Während der Abarbeitung der treiberspezifischen ISRs werden optional vorhandene *BHs* aktiviert. Diese *BHs* wiederum werden fast alle noch vor Rücksprung aus der Kernel-ISR behandelt. Anhand von Messungen haben wir beobachtet, dass *BHs* außerhalb der Kernel-ISR fast gar nicht aufgerufen werden. Die Messung der durch *BHs*

verbrauchten Energie wird durch unser Modell somit schon erbracht. Weiterhin wird der durch den *I-Estimator* implizierte Overhead der Zählerauslesung bei expliziter Betrachtung aller *BHs* jenseits von 100% liegen.

Kapitel 5

Implementierung

Zur Implementierung unseres Designs wählten wir einen Linux-Kern, welchen wir mit Energieabrechnung sowie energie- und NUMA-gewahrer Verteilung asynchroner Ereignisse ausstatteten. Zuvor erweiterten wir das System mit Merkels energiegewahren Scheduler [Mer05]. Der Umfang unserer Implementierung liegt bei ungefähr 5000 Code-Zeilen, die wir modifiziert haben. Im Folgenden wird die Implementierung der einzelnen Komponenten unseres Systems erläutert.

5.1 I-Associator

Der *I-Associator* dient zur Bereitstellung von Assoziationen zwischen Unterbrechungen und den sie auslösenden Geräten. Aufgrund dieser Assoziationen lassen sich Unterbrechungen den sie initiierenden Prozessen zuordnen.

Es ist zu beachten, dass nicht jede Unterbrechung einem Prozess zugeordnet werden kann. Timer, welche zum Triggern des Schedulers oder anderen notwendigen periodischen Kernelaktivitäten verwendet werden, können keinem bestimmten Prozess zugeordnet werden. Folglich werden sie in unserer Implementierung nicht berücksichtigt. Durch solche Unterbrechungen implizierter Energieverbrauch kann protokolliert und anteilig auf alle Prozesse verteilt werden. Der hierdurch entstehende Overhead ist, wie unsere Messungen gezeigt haben, nicht gerechtfertigt. Der Energieverbrauch solcher Unterbrechungen ist vernachlässigbar. Der *I-Associator* assoziiert ausschließlich Geräte, welche durch das virtuelle Dateisystem einer Datei zugeordnet werden können.

In Linux können Unterbrechungen statisch durch Magische Nummern oder dynamisch registriert werden. Im letzteren Falle geschieht diese Registrierung durch den Treiber des Gerätes, welcher anhand von *request_irq()* eine Unterbrechungsnummer registriert. Hierzu übergibt er als Parameter eine Unterbrechungsnummer, die Adresse seiner ISR und weitere gerätespezifische Daten. Bei erfolgreicher Registrierung wird dem Treiber dies

anhand des Rückgabewertes der Funktion *request_irq* mitgeteilt. Bei Eintreten der registrierten Unterbrechung wird innerhalb der systemweiten ISR die Treiber-ISR aufgerufen. Um durch unsere Implementierung nicht zu sehr in treiberspezifische Details einzugreifen, haben wir die Assoziationen des *I-Associators* durch zwei statische Tabellen realisiert.

Die Assoziationen werden im Linux-Kernel an folgenden Stellen benötigt:

1. Systemaufrufe und

2. Unterbrechungsbehandlung.

Zur Bereitstellung der Assoziationen während der Abarbeitung der Systemaufrufe bietet der *I-Associator* eine Liste, welche zu jedem registrierten Gerät die entsprechende Unterbrechungsnummer bereitstellt. Anhand des übergebenen Dateideskriptors wird durch das virtuelle Dateisystem herausgefunden, welches Gerät die referenzierte Datei verwaltet. Das Gerät wird durch seine eindeutige Identifikationsnummer, welche aus Major- und Minor-Nummer besteht, identifiziert. Bei Ein- und Austritt des Systemaufrufs wird geprüft, ob die Identifikationsnummer registriert ist. Ist dies der Fall, so wird die entsprechende Unterbrechungsnummer bereitgestellt. Diese Überprüfung findet aus Effizienzgründen nur innerhalb solcher Systemaufrufe statt, welche beim Datentransfer des Gerätes häufig einbezogen werden können und eine hohe Unterbrechungsdichte aufweisen können. Dies haben wir anhand von *strace*, einem Linux-Tool, und dem Proc-Dateisystem */proc/interrupts* ermittelt. Für Blockgeräte sind dies die Systemaufrufe *read* und *write*. Netzwerkkarten verwenden zum Datentransfer hauptsächlich *send*, *recv* und *sendfile*. Der Energieverbrauch von Character-Geräten ist vernachlässigbar. Dennoch können diese anhand ihrer *Dev-ID* eine Assoziation im *I-Associator* erhalten und analog zu Blockgeräten in die Energieabrechnung involviert werden. Wird zu Beginn des Systemaufrufes eine Assoziation gefunden, so wird unter anderem die entsprechende Unterbrechungsnummer im PCB des aufrufenden Prozesses *current* gesichert. Bei Beendigung des Systemaufrufes werden die entsprechenden PCB-Werte wieder gelöscht.

Wird in Linux eine ISR abgearbeitet, so wird die verantwortliche Unterbrechung auf allen Prozessoren maskiert. Eine zeitaufwändige ISR verschlechtert somit die Unterbrechungslatenz sowie die Latenz des restlichen Systems. Eine ISR kann nur durch nicht maskierte Unterbrechungen unterbrochen werden. Während ihrer Abarbeitung kann kein Nutzer-Programm abgearbeitet werden. Folglich müssen ISRs schnell abgearbeitet werden. Um dies sicherzustellen, stellt der *I-Associator* eine Bitmaske mit zugehöriger Tabelle bereit. Assoziiert der *I-Associator* eine Unterbrechung x zu einem Gerät, so wird das x-te Bit der Bitmaske gesetzt. In der Tabelle findet sich die Geräte-Identifikationsnummer an x-ter Stelle. Um herauszufinden, ob eine Unterbrechung assoziiert wurde, muss lediglich die Bitmaske überprüft werden.

Zur Bereitstellung von Information bezüglich prozessspezifischer Unterbrechungsenergie stellt der *I-Associator* die Datenstrukturen *hirq_sys_request* und *irq_sub_energy* bereit, welche den PCBs der Prozesse hinzugefügt wurden. In Linux werden PCBs durch die Datenstruktur *task_struct* implementiert.

Die Struktur *irq_sys_request* dient zur Sicherung von Informationen bezüglich eines unterbrechungsinitiierenden Systemaufrufes und zu Debugging-Zwecken. Ein Auszug dieser Struktur ist in Abbildung 5.1 gegeben.

```
struct irq_sys_request {
        int                     is_set;
        int                     irq;
        int                     sys_nr;
        int                     count;
        int                     handled;
        unsigned long long      sum;
        int                     nrirqs;
        unsigned long long      itime;
};
```

Abbildung 5.1: Datenstruktur *irq_sys_request*

Befindet sich ein Prozess innerhalb der Abarbeitung eines unterbrechungsinitiierenden Systemaufrufes, so wird *is_set* gesetzt. Nach Abschluss des Systemaufrufs wird dieser Wert wieder gelöscht. Befindet sich ein Prozess innerhalb eines unterbrechungsinitiierenden Systemaufrufes, so kann dies anhand des Wertes von *is_set* herausgefunden werden. Diese Überprüfung ist notwendig, um herauszufinden, ob einem Prozess die aufgetretene Unterbrechungsenergie zuzuordnen ist. Ist der Wert gesetzt, so wird dem Prozess die aufgetretene Unterbrechungsenergie angerechnet. Weiterhin werden die Anzahl aller tatsächlich angerechneter Unterbrechungen (*nirqs*), der gesamte Betrag konsumierter Unterbrechungsenergie (*sum*), Unterbrechungsnummer (*irq*), Anzahl der Zyklen, in welcher sich der Prozess in der Unterbrechungsbehandlung befand (*itime*), Nummer des aktuellen Systemaufrufs (*sys_nr*), Anzahl unterbrechungsinitiierender Systemaufrufe (*count*), Anzahl der Unterbrechungsenergie-Zuordnungen durch den *Estimator* (*handled*) und viele weitere zu Testzwecken benötigte Daten gesichert.

Wurde ein Prozess unterbrochen, so wird dies durch das Feld *is_set* der Struktur *irq_sub_energy* protokolliert (siehe Abbildung 5.2). Weiterhin wird die Anzahl (*count*) der Prozessunterbrechungen, der dem Prozess noch nicht abgerechnete Energiebetrag (*energy*), die Summe der dem Prozess abgerechneten Energie (*sum*) sowie die Anzahl der Zyklen (*itime*), welche dem Prozess ungerechtfertigt während seiner Unterbrechung angerechnet wurden, gesichert.

```
struct irq_sub_energy {
         atomic_t                is_set;
         int                     count;
         atomic_t                energy;
         unsigned long long      sum;
         atomic_t                itime;
};
```

Abbildung 5.2: Datenstruktur *irq_sys_request*

Die Felder dieser Strukturen werden von verschiedenen Instanzen unserer Implementierung ausgelesen oder beschrieben. Je nachdem, welche Instanzen um diese Daten konkurrieren, erfolgt der Zugriff atomar, durch das Setzen von Locks oder durch Sperren von Unterbrechungen. Atomarer Zugriff wird in Linux durch den Datentyp *atomic_t* bereitgestellt. Zum Locking verwendeten wir bereits vorhandene Spinlocks. Das Sperren von Unterbrechungen kann global für eine Unterbrechungsnummer auf allen Prozessoren oder lokal für alle Unterbrechungen eines Prozessors stattfinden.

5.2 I-Estimator

Der *I-Estimator* rechnet jede Energie, die während einer ISR verbraucht wird, der verursachenden Unterbrechung (sofern diese durch den *I-Associator* assoziiert wurde) zu. Dies geschieht wie in Kapitel 4 beschrieben.

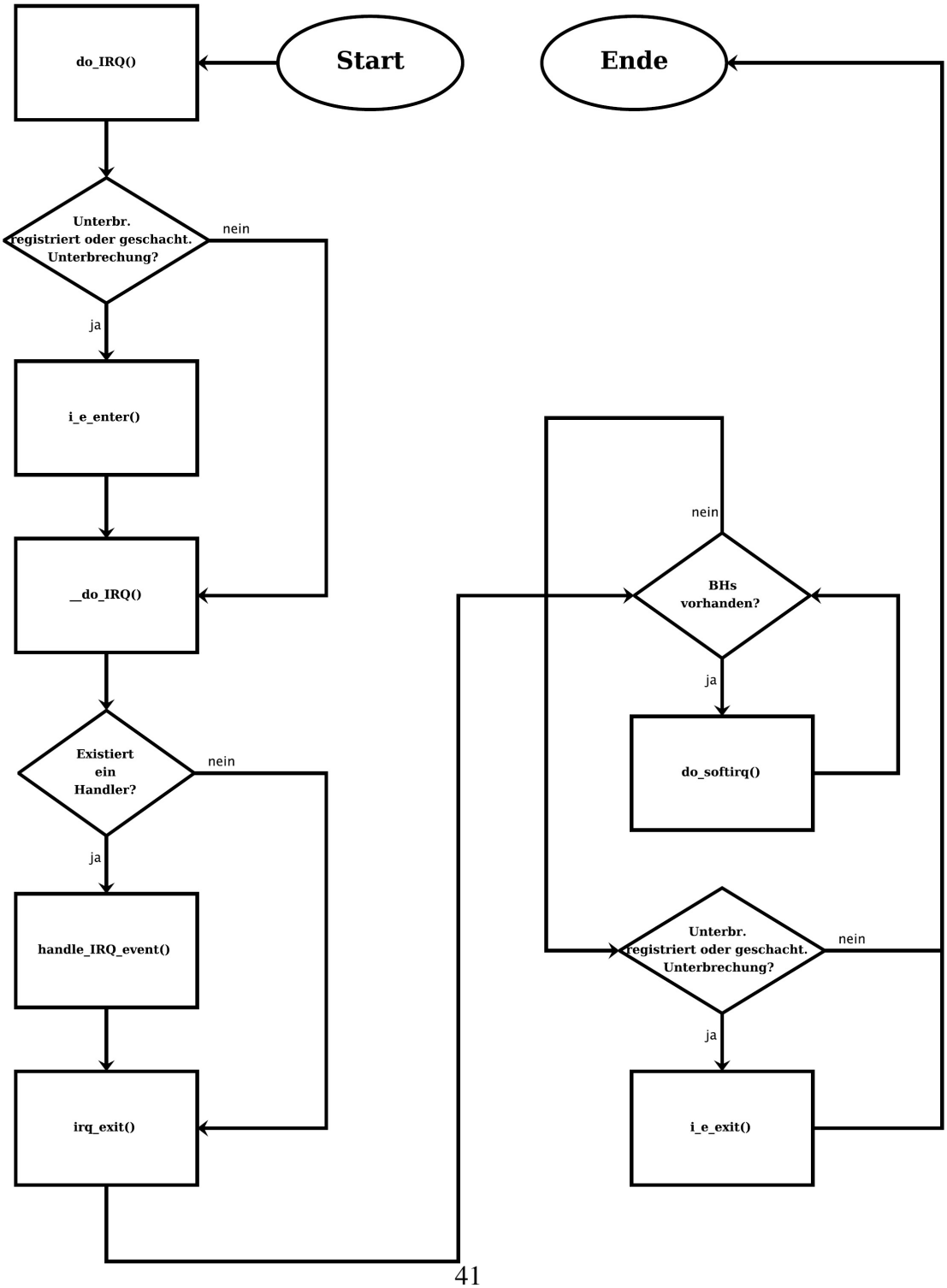
Die Abarbeitung einer ISR in Linux ist in Abbildung 5.3 ersichtlich. Da fast alle *Bottom Halves* innerhalb der Funktion *do_exit* abgearbeitet werden, repräsentiert die Energiemessung

$$\Delta E = E_{ISR_{exit}} - E_{ISR_{enter}} \tag{5.1}$$

die durch die ISR und der durch sie aktivierten *Bottom Halves* konsumierte Energie. Der Energiebetrag ΔE wird anhand Formel 3.1 durch Auslesen der Ereigniszähler durch die Funktionen *i_e_enter* und *i_e_exit* ermittelt. Die Berechnung findet in *i_e_exit* statt.

Zusätzlich misst der *I-Estimator* die Anzahl der Zyklen, welche während der Unterbrechungs-Abarbeitung verstreichen. Diese dienen dem *Profiler* zur Berechnung der Sampling-Periode τ_n^*. Zur Bereitstellung der Unterbrechungszeit haben wir analog zu Nutzer- und Systemzeit (*utime*, *stime*) die Unterbrechungszeit (*itime*) zur Datenstruktur *task_struct* (PCB) hinzugefügt.

Abbildung 5.3: Linux ISR-Abarbeitung



5.2.1 Bottom Halves

Der *I-Estimator* misst die Unterbrechungsenergie, die während *do_IRQ* konsumiert wird. Dies involviert die Messung der durch die Unterbrechung initiierten Bottom Halves. Während dieser Bottom Halves ist der Entzug des Prozessors an vielen Stellen möglich. Dies kann dazu führen, dass die Abarbeitung der Bottom Halves erst zum nächsten Timer-Tick beendet wird. Die gemessene Energie entspricht somit im schlechtesten Fall der Energie, die während eines kompletten Ticks verbraucht wird. Liegt ein solcher Fall vor, so entspricht die Zählerdifferenz des Time-Stamp-Counter in etwa den Zyklen, die während eines Ticks verstreichen. In diesem Fall wird die Unterbrechungsenergie auf einen von uns festgelegten, für die Unterbrechungsbehandlung typischen, Wert gesetzt (ca. $400\ nJ$).

5.3 Scheduler

Führt ein Prozess einen unterbrechungsinitiierenden Systemaufruf aus, so wird er in eine Warteschlange gesetzt. Nach Abarbeitung der Unterbrechung wird dem Scheduler signalisiert, dass der wartende Prozess wieder eingegliedert werden kann. Vor dieser Wiedereingliederung findet die energiegewahre Prozessverteilung (siehe Kapitel 4.5) sowie das Triggern des *I-Balancer* statt.

Als Eingabe erhält der Scheduler die prozessspezifischen Energieprofile x_n und x_n^* sowie die Unterbrechungszeit *itime* (siehe Kapitel 4.5). Wird ein Prozess in eine Warteschlange gesetzt, so wird dieser Prozess nicht mehr über die pro Prozessor implementierte Warteschlange *runqueue* referenziert. Der *I-Balancer* benötigt allerdings zur Grundlage der Unterbrechungsverteilung Informationen bezüglich der Unterbrechungszeiten wartender Prozesse. Um dies sicherzustellen, protokollieren wir die Unterbrechungszeiten schlafender Prozesse pro Prozessor innerhalb des Schedulers.

Nach Abarbeitung der Prozessverteilung wird der *I-Balancer* aufgerufen. Soll aufgrund der Überschreitung des Energielimits eine Unterbrechung migriert werden, so wird dies dem *I-Balancer* durch Anhängen des Migrationswunsches an eine Liste mitgeteilt.

5.4 I-Balancer

In Linux erhält jeder Prozessor einen dedizierten Scheduler. Jeder Scheduler verfügt über eine Warteschlange *rq* des Datentyps *struct rq*. Diese Warteschlange verfügt über Zeiger zu allen auf dem Prozessor befindlichen PCBs, welche lauffähige Prozesse repräsentieren. Aus Locking-Gründen haben wir den I-Balancer innerhalb des Schedulers implementiert. Das Sperren der *runqueues* sowie das Sperren einzelner Prozesse außerhalb des Schedulers hat sich als schwierig und fehleranfällig erwiesen. Zur Überprüfung der Unterbrechungszeit aller Prozesse eines Prozessors sowie zur Migration von Prozessen greifen

wir auf die Prozesse der Warteschlangen zu. Um diesen Zugriff zu synchronisieren, halten wir die Spinlocks der Warteschlangen. Weiterhin lesen und beschreiben wir Daten des *I-Associators* innerhalb des Schedulers. Diese Daten sind Teil des PCB. Ihr Zugriff erfolgt über Referenzen zu Prozessen. Auch dies kann ohne Halten der Spinlocks nicht geschehen. Andernfalls riskieren wir einen inkonsistenten Zustand des Systems.

Findet bei der Abarbeitung des Schedulers durch einen Prozessor ein Prozesswechsel statt, so wird der *I-Balancer* aufgerufen, sofern dieser nicht bereits auf einem anderen Prozessor abgearbeitet wird. Der *I-Balancer* wird somit serialisiert und periodisch aufgerufen.

Als Eingabewerte benötigt der *I-Balancer* Informationen über die zugrundeliegende NUMA-Architektur, prozessspezifische Unterbrechungsinitiierungen und aktuelle Unterbrechungsverteilung. NUMA-Architektur und aktuelle Unterbrechungsverteilung werden während der Initialisierung ausgelesen.

Wird der *I-Balancer* aufgerufen, so prüft er zuerst, ob Migrationswünsche des Schedulers vorliegen. Diese befinden sich in einer doppelt verketteten Liste und werden direkt bedient. Die Priorität der energiegewahren Unterbrechungsverteilung übersteigt die Priorität der NUMA-gewahren Unterbrechungsverteilung. Aus diesem Grund erfolgt die Migration auch während IPP (Unterbrechungs-Ping-Pong). Der *I-Balancer* erhält als Parameter eine Referenz auf den aktuell ausgeführten Prozess. Handelt es sich um einen unterbrechungsinitiierenden Prozess, so wird geprüft, ob die Unterbrechung zum selben NUMA-Knoten, welcher den aktuellen Prozess abarbeitet, gesendet wird. Ist dies nicht der Fall, so wird die Unterbrechung zum entsprechenden NUMA-Knoten migriert. Bei zu hoher Migrationsfrequenz wird der IPP-Modus aktiviert.

5.4.1 IPP

Liegt IPP vor, so wird wie in Kapitel 4.6 beschrieben zwischen *Default-* und *Performance-Mode* unterschieden.

Default Mode

Im Default-Mode prüft der *I-Balancer* alle Warteschlangen auf Prozesse, welche Unterbrechungsenergie verbrauchen. Die Unterbrechungszeiten (*itimes*) dieser Prozesse werden gesichert und um die Unterbrechungszeiten der schlafenden Prozesse, welche zuletzt durch den lokalen Prozessor abgearbeitet wurden, ergänzt. Hierzu sichern wir pro Prozessor die Unterbrechungszeiten der Prozesse, welche aus einer lokalen Warteschlange entfernt werden. Anhand der zugrundeliegenden NUMA-Architektur wird aufgrund der Summe der Unterbrechungszeiten derjenige Knoten ermittelt, welcher die längste Zeit in der Unterbrechungsbehandlung der für IPP verantwortlichen Unterbrechung verbringt. Dieser Knoten erhält die Unterbrechung.

Performance Mode

Der NUMA-Knoten k, welcher die Unterbrechung erhalten soll, wird analog zum Default-Mode ermittelt. Im Performance-Mode werden alle Prozesse, welche eine hohe Energieprofildifferenz δ_{x_n} aufweisen, zu diesem Knoten migriert. Aufgrund von Locking-Problemen, schwer referenzierbaren schlafenden Prozessen und aus Effizienzgründen erfolgt diese Migration allerdings erst bei Wiedereingliederung der betreffenden Prozesse durch den Scheduler. Dieser prüft bei Wiedereingliederung aller Prozesse, ob IPP und Performance-Mode vorliegen und migriert die Prozesse zum Knoten k.

5.5 Proc-Dateisystem – Schnittstelle

Zum Export und Import von Daten aus und zum Betriebssystem-Kern stellt Linux das Proc-Dateisystem als Schnittstelle bereit. Wir haben uns zur Auswertung und zum Debugging unserer Implementierung an mehreren Stellen dieser Schnittstelle bedient. Dies wird im Folgenden beschrieben.

5.5.1 Energieabrechnung

Zur Ein- und Abschaltung der Einbeziehung von Unterbrechungsenergieen in den energiegewahren Scheduler haben wir die Datei */proc/eeas* zum Proc-Dateisystem hinzugefügt. Zur Überprüfung unserer Energieprofile sowie unserer in Kapitel 4 beschriebenen Datenstrukturen haben wir Merkels prozessspezifische Datei */proc/<PID>/energy* erweitert. Anhand dieser Datei können wir die für uns relevanten Werte auslesen.

5.5.2 Unterbrechungsverteilung

Prozessorspezifische Unterbrechungs-Kardinalitäten werden in Linux durch Auslesen der Datei */proc/interrupts* bereitgestellt. Zur Überprüfung unserer Unterbrechungsverteilung haben wir die Datei */proc/ibal* hinzugefügt. Das Auslesen dieser Datei gibt uns Informationen über die Häufigkeit von IPP, die Migrationsreihenfolge der Unterbrechungen sowie einige Informationen über die Aufrufzeitpunkte des *I-Balancer*. Diese Informationen werden für jede durch den *I-Associator* registrierte Unterbrechungsnummer ausgegeben.

5.5.3 Drosselung

Um die Drosselungen der einzelnen Prozessoren zu messen haben wir zum Proc-Dateisystem die Datei */proc/throttle* hinzugefügt. Das Auslesen dieser Datei liefert den prozen-

tualen Anteil der Drosselungen eines jeden Prozessors.

5.5.4 Overhead

Zur Messung des unterbrechungsspezifischen Overheads, welcher durch den *I-Estimator* verursacht wird, haben wir für jede Unterbrechung die Datei */proc/<IRQ>/energy* erstellt. Um den Overhead der Unterbrechungsnummer *<IRQ>* auf Prozessor c zu ermitteln beschreiben wir */proc/<IRQ>/energy* mit dem Wert c. Durch späteres Auslesen dieser Datei bringen wir in Erfahrung, wieviel Zyklen zur Bestimmung der Unterbrechungsenergie und für die gesamte Unterbrechungsbehandlung benötigt wurden.

Kapitel 6

Evaluierung

Die Evaluierung unserer Implementierung gliedert sich in Energieabrechnung, Energieverteilung und Unterbrechungsverteilung. Zuvor gehen wir genauer auf unser Test-System ein. In Abschnitt 6.2 findet sich die Evaluierung der Korrektheit unserer Unterbrechungsenergie-Zuordnung. Im folgenden Abschnitt 6.3 findet sich die Überprüfung der Energieverteilung nach Einbeziehung der Unterbrechungsenergie. Abschnitt 6.4 dient zur Überprüfung unserer Unterbrechungsverteilung. Der letzte Abschnitt 6.5 befasst sich mit dem durch unsere Implementierung entstandenen Overhead.

6.1 Test-System

Zur Evaluation unseres modifizierten Linux-Kernels verwendeten wir einen IBM x445 8-fach Pentium Xeon Multiprozessor. Jeder dieser Prozessoren verfügt über eine maximale Taktfrequenz von $2{,}2\ GHz$ und unterstützt HyperThreading. Die Prozessoren verteilen sich gleichmäßig über zwei NUMA-Knoten.

Zur Evaluierung haben wir vier Geräte im *I-Associator* registriert. Hierzu gehören eine 32-Bit Intel Pro/1000 Gigabit Netzwerkkarte (PCI), ein Matsushita SR-8177 DVD-Rom-Laufwerk (ATA) und zwei interne IBM MAS3367NC 35,4 GB Festplatten (SCSI). Die beiden Festplatten teilen sich eine Unterbrechungsnummer. Die Netzwerkkarte sowie das DVD-Rom-Laufwerk erhalten separate Unterbrechungsnummern. Alle weiteren vorhandenen Geräte initiieren eine vernachlässigbare Anzahl an Unterbrechungen und verbrauchen kaum Unterbrechungsenergie.

Der von uns gewählte Testrechner unterstützt HyperThreading. Somit involviert aus architektonischer Sicht jeder physische Prozessor zwei logische Prozessoren. Die von uns verwendeten Gewichte zur Energieberechnung basieren aufgrund fehlender exklusiver Zählerregister pro logischem Prozessor auf einem Testsystem mit deaktivierter SMT-Unterstützung. Von den acht von uns verwendeten Zählern sind lediglich drei exklusiv

für die logischen Prozessoren nutzbar. Dies macht eine exakte Energieberechnung pro logischen Prozessor unmöglich. Dennoch haben wir unsere Implementierung SMT-fähig gestaltet. Um einen wie in Merkels *Estimator* durch Multiplexen partitionierter Zähler entstehenden Overhead zu umgehen, behandeln wir solche Register wie geteilte Register [Mer05]. Alle nicht exklusiv nutzbaren Register werden bei der Energieberechnung im SMT-Fall halbiert. Dies führt zu ungenauen Ergebnissen. Aufgrund fehlender Zählerregister ist dies unvermeidbar. Das Vorhandensein expliziter Zähler pro logischen Prozessor wird allerdings für unsere Arbeit zu keinen neuen Erkenntnissen führen. Wir können somit auf die SMT-Unterstützung ohne nachteilige Auswirkungen auf unsere Arbeit verzichten.

Eine weitere Unannehmlichkeit stellte ein Fehler unseres zugrundeliegenden IO-APIC dar. Die Unterbrechungen können trotz Beschreibung der Redirektions-Tabelle lediglich zu vier der acht verfügbaren Prozessoren gesendet werden. Der Unterbrechungsverteilung stehen somit nur die Hälfte der Prozessoren zur Verfügung.

6.2 Energieabrechnung

Zur Energiemessung unserer Xeon-Prozessoren haben wir die durch Kellner gemessenen Ereigniszähler und Energiegewichte (siehe Tabelle 6.1) verwendet [Kel03].

Ereignis	Gewicht
Time Stamp Counter	6,17 nJ
Unhalted Cycles	7,12 nJ
μOp Queue Writes	4,75 nJ
Retired Branches	0,56 nJ
Mispredicted Branches	340,46 nJ
Mem Retired	1,73 nJ
Mob Load Replay	29,96 nJ
ld Miss 1L Retired	13,55 nJ

Tabelle 6.1: Energiegewichte Pentium 4

Zum Test der Korrektheit und Zuverlässigkeit unserer Energieabrechnung und der Zuteilung der Unterbrechungsenergie haben wir zwei unterbrechungslastige Testprogramme verwendet. Zur Netzwerkbelastung ist dies *netperf*, zum Datentransfer mit Massenspeicher haben wir *cat* verwendet. Zuerst haben wir die Energieprofile (Tabelle 6.2), den Energieverbrauch (Tabelle 6.3) und die Zeitverteilung (Tabelle 6.4) der Testprogramme gemessen. Hierzu starteten wir die Programme jeweils einzeln bei ausgeschaltetem *I-Balancer*. Mittels *taskset* haben wir die Testprogramme zu verschiedenen Prozessoren

delegiert. Nach der Laufzeit t haben wir durch Auslesen von */proc/<PID>/energy* die für uns relevanten Daten ausgelesen und das Programm beendet.

Die erste Spalte der der Tabellen 6.2 bis 6.4 liefert den Namen der ausgeführten Applikation. Im Falle von *netperf* haben wir in Klammern den Durchsatz in $\mathrm{Mbits/s}$ zugefügt. Die beiden folgenden Spalten enthalten die Nummer des Prozessors ($\mathrm{C_{App}}$), welcher die Applikation ausführt und die Nummer des Prozessors ($\mathrm{C_{Int}}$), welcher die Unterbrechungsbehandlung abarbeitet. Die vierte Spalte repräsentiert das Zeitintervall, nach dem wir die Werte gemessen und die Applikation beendet haben. Tabelle 6.2 gibt die Energieprofile x_n, x_n^* und x_n' sowie die Profildifferenz δ_{x_n} in der SI-Einheit Watt an (siehe Kapitel 4). Die letzte Spalte gibt die Anzahl der Unterbrechungen an, die der Applikation angerechnet wurden.

Tabelle 6.3 gibt die Energieverbrauchswerte der Applikationen in mJ an. Aufgeführt ist die Gesamtenergie ($\mathrm{E_{ges}}$), welche die Applikation während der Zeit t verbraucht hat, die Unterbrechungsenergie ($\mathrm{E_{add}}$), die der Applikation angerechnet wurde, sowie die Unterbrechungsenergie ($\mathrm{E_{sub}}$), die der Applikation ungerechtfertigt angerechnet wurde, während sie selbst unterbrochen wurde. Die letzte Spalte enthält den prozentualen Anteil angerechneter Unterbrechungsenergie relativ zur Gesamtenergie.

Tabelle 6.4 enthält die Zeitspannen, in welchen die Applikation im Nutzerraum (utime), im Kernelraum (stime) sowie während der Unterbrechungsbearbeitung (itime) Energie verbrauchte. Die Zeitspannen sind in *ticks* angegeben. Ein *tick* entspricht der Periode, in welcher der Scheduler durch den Timer aufgerufen wird. Für unseren Fall ist dies eine Millisekunde.

Applikation	$\mathrm{C_{App}}$	$\mathrm{C_{Int}}$	t	$\mathrm{x_n}$	$\mathrm{x_n'}$	$\mathrm{x_n^*}$	$\delta_{\mathrm{x_n}}$	#Int
netperf (921)	0	4	10s	34,3	34,3	33,0	-1,3	77000
netperf (915)	2	4	10s	34,3	34,3	33,0	-1,3	76000
netperf (941)	4	4	10s	35,0	35,0	34,8	-0,2	80000
netperf (941)	6	4	10s	34,2	34,2	33,9	-0,3	80000
cat cdrom	0	4	50s	36,1	36,1	35,4	-0,7	32000
cat cdrom	2	4	50s	36,2	36,2	35,6	-0,6	32500
cat cdrom	4	4	50s	30,1	30,1	30,8	-0,7	32000
cat cdrom	6	4	50s	38,6	38,6	36,0	-1,6	32200
cat sda	0	4	30s	34,6	34,6	34,1	-0,5	3700
cat sda	2	4	30s	34,0	34,0	33,8	-0,8	5800
cat sda	4	4	30s	35,6	35,6	35,2	-0,4	4800
cat sda	6	4	30s	35,5	35,5	35,0	-0,5	5300

Tabelle 6.2: Auswertung der Energieabrechnung - Energieprofile

Programm	C_{Progr}	C_{Int}	t	E_{ges}	E_{add}	E_{sub}	$\%E_{add}$
netperf (902)	0	4	10 s	395	182	0	46,1
netperf (902)	2	4	10 s	403	186	0	46,1
netperf (902)	4	4	10 s	231	104	0	45,0
netperf (902)	6	4	10 s	227	113	0	49,8
cat cdrom	0	4	50 s	52	28	0	53,8
cat cdrom	2	4	50 s	52	29	0	55,9
cat cdrom	4	4	50 s	161	29	2	18,0
cat cdrom	6	4	50 s	44	29	0	65,9
cat sda	0	4	30 s	793	8	0	1
cat sda	2	4	30 s	622	13	0	2
cat sda	4	4	30 s	703	5	0	1
cat sda	6	4	30 s	695	6	0	1

Tabelle 6.3: Auswertung der Energieabrechnung - Energieverbrauch

Programm	C_{Progr}	C_{Int}	t	$utime$	$stime$	$itime$
netperf (941)	0	4	10 s	37	6151	5013
netperf (941)	2	4	10 s	31	6307	5146
netperf (941)	4	4	10 s	38	4973	1396
netperf (941)	6	4	10 s	29	3284	2946
cat cdrom	0	4	50 s	21	625	694
cat cdrom	2	4	50 s	31	602	713
cat cdrom	4	4	50 s	32	3060	657
cat cdrom	6	4	50 s	25	351	719
cat sda	4	4	30 s	1284	22007	226
cat sda	4	6	30 s	995	17035	335
cat sda	4	2	30 s	1330	18842	130
cat sda	4	2	30 s	1374	18573	169

Tabelle 6.4: Auswertung der Energieabrechnung - Zeitverteilung

Die durch unsere Testprogramme initiierten Unterbrechungen haben wir stets zum vierten Prozessor delegiert. Dieser Prozessor erhält somit die von uns initiierten Unterbrechungen und den Timer. Weitere Unterbrechungen finden auf diesem Prozessor nicht statt. Die Testprogramme verfügen über unterschiedliche Charakteristiken. Die höchsten Unterbrechungsdichten finden sich durch Inanspruchnahme der Netzwerkarte durch *netperf* (bis zu 8000 Unterbrechungen pro Sekunde). Inanspruchnahme des CD-Laufwerks (640 Unterbrechungen pro Sekunde) oder der Festplatten (bis zu 330 Unterbrechungen pro Sekunde) durch *cat* liefern weitaus geringere Unterbrechungsdichten.

Unsere Messungen zeigen die direkte Kopplung von Unterbrechungsenergie und Speicherhierarchie. Trifft eine Unterbrechung auf einem, aus Sicht des Initiatorprozesses entfernten Prozessor ein, so wird für die Abarbeitung der Unterbrechung ein höherer Energiebetrag benötigt. Die Daten müssen zum Initiatorprozessor (Prozessor, der den Initiatorprozess abarbeitet) transferiert werden. Je mehr Ebenen der Speicherhierarchie zwischen den beiden Prozessoren liegen, desto höher sind die Energiekosten dieses Datentransfers. Tabelle 6.5 zeigt den prozentualen Anstieg der Unterbrechungsenergie pro Unterbrechung für die Fälle

1. Unterbrechung trifft auf einem Prozessor des selben Knotens ein (3. Spalte) und
2. Unterbrechung trifft auf einem anderen Knoten ein (4. Spalte).

Gerät	E_{int}	CPU	NUMA
Netzwerkkarte	1130 nJ	10 %	57 %
CD-Rom-Laufwerk	890 nJ	0 %	1 %
Festplatte	875 nJ	34 %	157 %

Tabelle 6.5: Unterbrechungsenergie in Abhängigkeit des Initiatorprozessors

Der prozentuale Anstieg bezieht sich stets auf diejenige Unterbrechungsenergie E_{int}, die durch einen Prozess angerechnet wurde, der durch den gleichen Prozessor wie die Unterbrechung selbst abgearbeitet wird. Diese unterschiedliche Unterbrechungsenergie impliziert die in Tabelle 6.2 ersichtlichen unterschiedlichen Profildifferenzen bei unterschiedlicher Platzierung der Initiatorprozesse.

Das CD-Rom-Laufwerk bildet nach der Studie unserer Messungen eine Ausnahme der direkten Kopplung zwischen Unterbrechungsenergie und Speicherhierarchie. Dies ist durch die Art und Weise der zugrundeliegenden Unterbrechungsbehandlung des für unser CD-Rom-Laufwerk zuständigen IDE-Treibers zu erklären. Nach Abarbeitung der Treiber-ISR werden *Bottom Halves* aufgerufen. Im Gegensatz zu den beiden anderen Treibern können diese den Prozessor freiwillig abgeben. Dies führt dazu, dass die Abarbeitung der *Bottom Halves* erst nach dem nächsten Timer-Tick beendet werden kann. Folglich liegt

die von uns gemessene Unterbrechungsenergie im Bereich der Energie, die während einer Scheduler-Zeitscheibe verbraucht wird. Dies ist nicht richtig. Präventiv überprüfen wir zu jeder Berechnung der Unterbrechungsenergie, wie viele Prozessor-Zyklen während der Unterbrechungsbehandlung verstrichen sind. Dies birgt keinen zusätzlichen Overhead, da der Time-Stamp-Counter, welcher die Anzahl verstrichener Zyklen repräsentiert, bereits zur Energiemessung von uns verwendet wird (siehe Tabelle 6.1). Tritt ein solcher Fall ein, so setzen wir die Zählerdifferenzen auf ihre Maximalwerte. Die Maximalwerte haben wir auf diejenigen Werte festgesetzt, welche maximal durch Unterbrechungen auftreten, die den Prozessor nicht während der BH-Behandlung freiwillig abgeben. Auch nach sorgfältiger Prüfung der Implementierungs-Details des IDE-Treibers haben wir weder den Grund für die freiwillige Abgabe des Prozessors noch den Grund für den hohen Energiebedarf von *cat*, welches auf dem selben Prozessor ausgeführt wird, wie die durch es initiierten CD-Rom-Unterbrechungen, finden können.

Unsere Messergebnisse zeigen weiterhin, dass die Energiedichten der Unterbrechungsbehandlung unserer Test-Applikationen geringer als die Energiedichten außerhalb der durch sie initiierten Unterbrechungsbehandlung sind. Dies zeigt sich durch die gering negativen Profildifferenzen in Tabelle 6.2. Folglich errechnen energiegewahre Scheduler für Prozesse, deren Unterbrechungen auf entfernten Prozessoren abgearbeitet werden, keine Energieprofile, deren Beträge größer sind als die von unserem Scheduler errechneten Profilbeträge. Dies führt dazu, dass die durch Prozesse initiierten Unterbrechungen nicht dazu führen, dass diese Prozesse ungerechtfertigt als *Hot-Tasks* eingeordnet werden. Das Gegenteil ist der Fall: Verbrauch ein Prozess A außerhalb seiner Unterbrechungsbehandlung viel Energie und während seiner durch ihn initiierten Unterbrechungsbehandlung wenig Energie, so wird er, wenn die Unterbrechungen auf dem selben Prozessor wie der Prozess abgearbeitet werden, vom herkömmlichen energiegewahren Scheduler als *Hot-Task* eingestuft und gegebenfalls zu einem anderen Prozessor migriert. Je mehr Zeit Prozess A hierbei für seine Unterbrechungsbehandlung benötigt, desto ungerechtfertigter ist diese Entscheidung. Wir erläutern dies an folgendem Beispiel.

Prozess A verbraucht $x_n = 70\ \mathrm{Watt}$ außerhalb der Unterbrechungsbehandlung während einer Periode

$$\tau_A = \mathrm{stime} + \mathrm{utime}.$$

Der Verbrauch während der Unterbrechungsbehandlung liegt analog zu unseren Messungen bei etwa $x_{int} = 30\ \mathrm{Watt}$ während der Periode

$$\tau_{int} = \mathrm{itime}.$$

Sei nun

$$\tau_A \approx \tau_{int},$$

so liegt der tatsächliche Energieverbrauch bei $50\ \mathrm{Watt}$. Dies ist dem herkömmlichen energiegewahren Scheduler nicht bekannt. Folglich migriert er einen Prozess aufgrund einer zu hohen Profilberechnung.

Wird ein Prozess hingegen kontinuierlich durch Unterbrechungen, die durch einen anderen Prozess initiiert werden, unterbrochen, so errechnet der energiegewahre Scheduler ein zu niedriges Energieprofil. Wird ein Prozess A, der $70\ \mathrm{Watt}$ benötigt, durch nicht durch ihn initiierte Unterbrechungen unterbrochen, welche $30\ \mathrm{Watt}$ benötigen, so misst der energiegewahre Scheduler für ihn ein zu geringeres Energieprofil. Wird der Prozess immer zu 50 Prozent seiner Verarbeitungszeit unterbrochen, so ergibt sich für ihn ein ungerechtfertigtes Energieprofil von $50\ \mathrm{Watt}$. Prozess A kann nun ungerechtfertigt als *Cool-Task* eingestuft werden, obwohl er als *Hot-Task* eingestuft werden muss. Verursacht ein solcher Prozess einen Temperaturanstieg auf dem ihn abarbeitenden Prozessor, so kann der energiegewahre Scheduler die Temperatur nicht herabsenken, da er dem Prozess ein zu niedriges Profil zuordnet. Auf Grundlage dieses niedrigen Energieprofils erfolgt keine Einstufung als *Hot-Task*. Der Prozess wird nicht migriert. Die Auswirkungen dieses Sachverhaltes sind in Abbildung 6.1 für den Prozess *cpu_hock* ersichtlich. *cpu_hock* benötigt zur Abarbeitung ca. $56\ \mathrm{Watt}$. Er initiiert keine Unterbrechungen und lastet seinen Prozessor zu 100 Prozent durch Fließkomma-Arithmetik aus. Je öfter *cpu_hock* unterbrochen wird, desto geringer ist sein durch den energiegewahren Scheduler errechnetes Energieprofil. Zur Verdeutlichung haben wir die Energieprofile x'_n und x^*_n gegen die Unterbrechungsdichte, mit welcher der Prozess unterbrochen wird, aufgetragen. Die Unterbrechungen haben wir durch *netperf* durch einen entfernten Prozessor initiiert. Zur dynamischen Veränderung der Unterbrechungsdichten δ_{int} haben wir den zu unserer Instanz von *netperf* zugeordneten *netserver* mit SYN-Flooding beschäftigt.

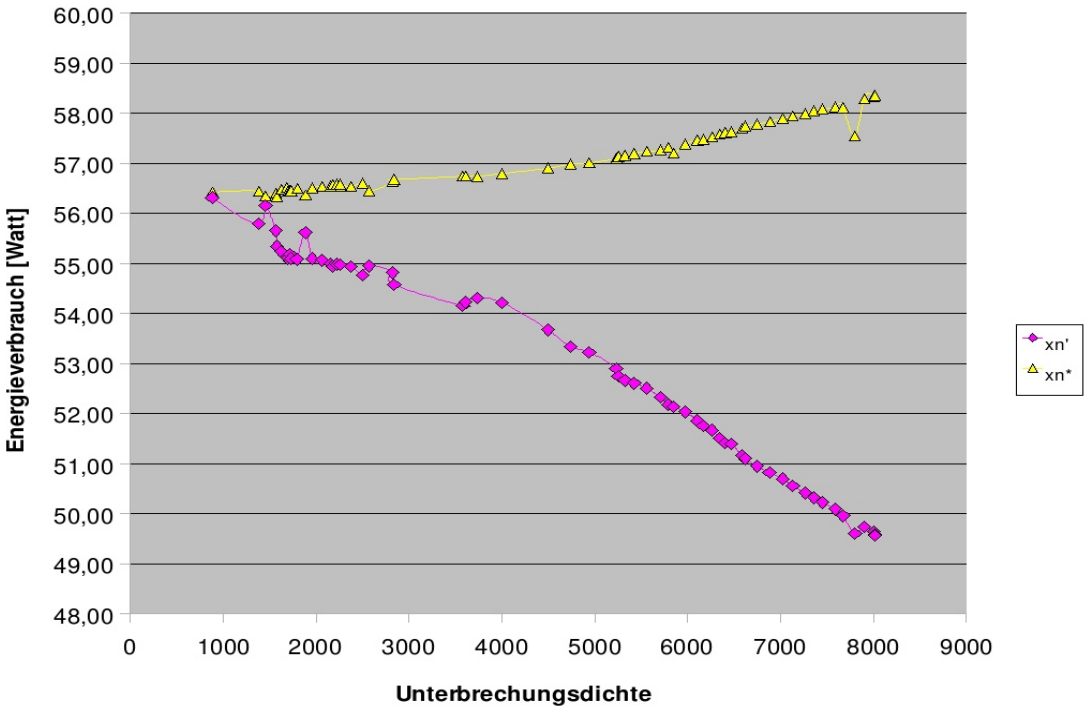
Im Gegensatz zur Profilberechnung kann der Anteil der Unterbrechungsenergie über die Hälfte der Gesamtenergie betragen (siehe Tabelle 6.3). Dies gilt insbesondere für *netperf*. Aufgrund des hohen Anteils an Unterbrechungsenergie (ca $50\ \%$), der hohen initiierten Unterbrechungsdichte $(8000/s)$ sowie der hohen Prozessorauslastung (bis $50\ \%$) beschränken wir die weitere Evaluierung auf *netperf*. Die Netzwerk-Karte liegt mit einer maximalen Übertragungsrate von $1000\ \mathrm{Mbit/s}$ weit über den Übertragungsraten der anderen Geräte.

Zum letztendlichen Beweis der korrekten Zuordnung der Unterbrechungsenergie haben wir die *netperf* parallel auf den verschiedenen Prozessoren gestartet. Als Maß der korrekten Energiezuordnung nehmen wir den Quotienten q aus der Anzahl zugeordneter Unterbrechungen pro Sekunde $\#\mathrm{Int}$ und des resultierenden Durchsatzes u zu

$$q = \frac{\#\mathrm{Int}}{u}. \tag{6.1}$$

Wir haben acht Instanzen von *netperf* parallel ausgeführt. Diese teilten sich den Gesamt-Durchsatz der Netzwerkkarte. Die einzelnen Instanzen erhielten hierbei zwischen 85 und

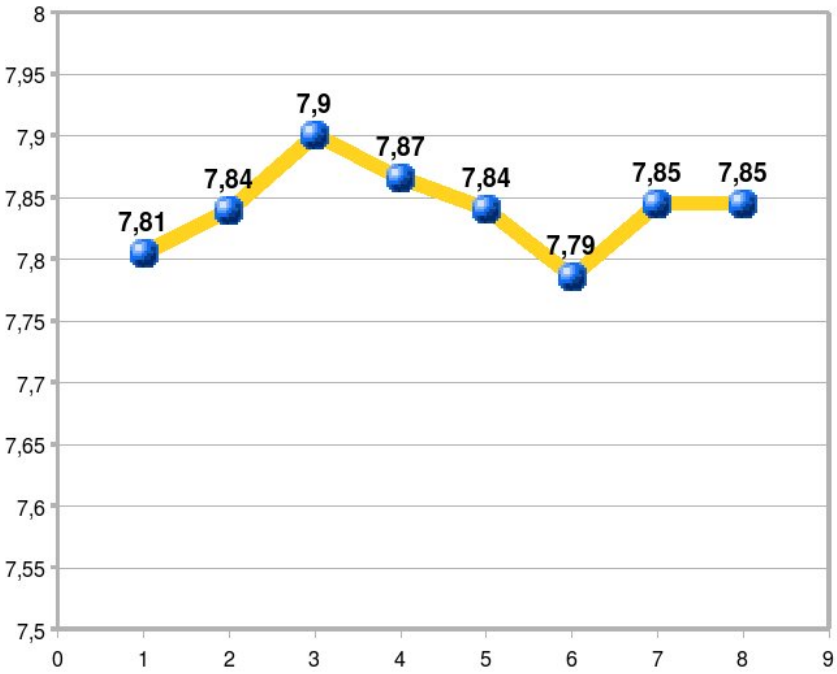
Abbildung 6.1: Energieprofile von *cpu_hock*



155 Mbits/s. Die unterschiedlichen Durchsätze resultieren darauf, dass die verschiedenen Instanzen nicht zur exakt gleichen Zeit gestartet werden. Diejenige Instanz von *netperf*, welche durch unser Skript zuerst gestartet wurde, erhielt folglich für eine kurze Zeitspanne den kompletten Durchsatz der Netzwerkkarte. Nach Start der zweiten Instanz teilten sich für eine kurze Zeitdauer zwei Instanzen den gesamten zur Verfügung stehenden Durchsatz. Dies setzt sich analog für alle weiteren Instanzen fort. Weiterhin erhalten die Instanzen von *netperf* einen tendenziell höheren Durchsatz, wenn sie auf dem selben Knoten ausgeführt werden, der die Unterbrechung behandelt.

In Abbildung 6.2 sehen wir, dass der Quotient q für die Applikationen in etwa gleich ist. Die x-Achse repräsentiert die einzelnen Instanzen von *netperf*, die y-Achse zeigt den Quotienten q an. Bei serieller Ausführung von *netperf* haben wir für Durchsätze um 100 Mbits/s gleichwertige Quotienten erhalten. Wir haben bei unseren seriellen Tests stets eine konstanten Quotienten q gemessen. Wir postulieren somit eine erfolgreiche Zuordnung der Unterbrechungsenergie im Falle mehrerer um die Unterbrechungsenergie

Abbildung 6.2: Quotient q für parallele Ausführung von *netperf*



konkurrierender Prozesse.

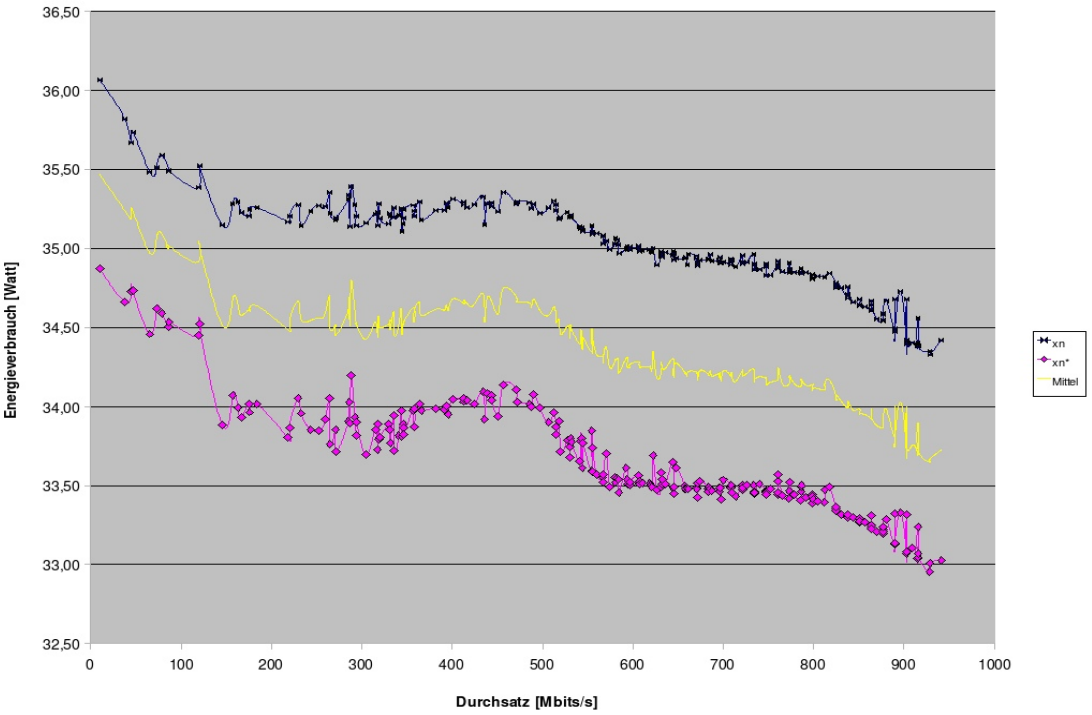
Die Abbildungen 6.3 und 6.4 zeigen die Abhängigkeit der Energieprofile von *netperf* zum Durchsatz sowie zur Speicherlokalität. Um den Durchsatz des *netserver*s zu manipulieren haben wir das entsprechende System inkrementell mit SYN-Flooding beschäftigt. In Abbildung 6.3 ist ersichtlich, dass die Profildifferenz bei Ausführung von *netperf* auf einem Knoten, welcher keine Unterbrechungen abarbeitet, konstant ist. Sie liegt bei ca. $-1,3\ \mathrm{Watt}$. Tendenziell nimmt der Energieverbrauch bei höherem Durchsatz ab.

Die folgende Abbildung 6.4 zeigt die Energieprofile von *netperf*, welches durch die Prozessoren 4 und 6 abgearbeitet wird. Diese sind Teil des Knotens, welcher die Unterbrechungsbehandlung durchführt. In beiden Fällen zeigt sich ein minimaler Energieverbrauch bei einem Durchsatz von ca $500\ \mathrm{MBit/s}$. Bei höherem Durchsatz steigen die Energieprofile bei gleichbleibender Profildifferenz. Die Profildifferenz beträgt ca. $-0,3\ \mathrm{Watt}$.

6.2.1 Sporadische Unterbrechungsinitiierung

Falsche Energiezuordnung aufgrund sporadischer Unterbrechungsinitiierung haben wir lediglich für Unterbrechungsenergie messen können, die durch die Unterbrechungen der

Abbildung 6.3: Energieprofile *netperf* – Unterbrechungsknoten ungleich Initiatorknoten

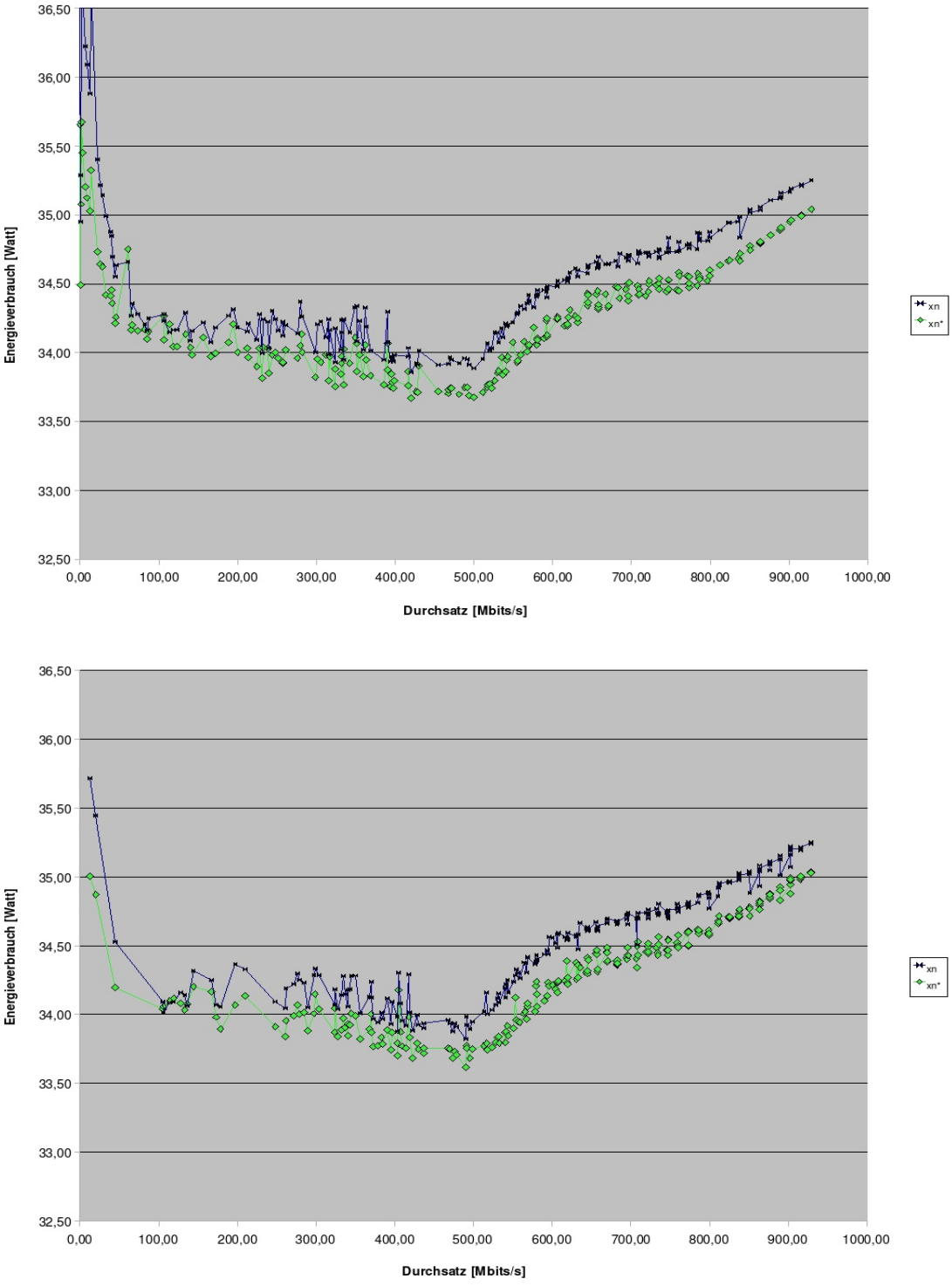


Festplatten entstanden sind. Haben wir mittels *cat* Daten der Festplatten ausgelesen, so wurden Teile dieser Energie Prozessen zugeordnet, welche geringe Nutzer- und Systemzeiten aufweisen. Als Beispiel sei hier *ssh* genannt. Wir haben uns mit diesem Programm auf den Testrechner eingeloggt. *ssh* konkurriert hierbei mit *cat* um die Unterbrechungsenergie der Festplatte, da es periodisch Daten auf dieser sichert (z.B.: *history*). Die Wahrscheinlichkeit θ lässt sich in diesem Falle durch

$$\theta = \frac{\text{stime} + \text{utime}}{\text{stime} + \text{utime} + \text{itime}} \tag{6.2}$$

beschreiben. Liegt dieser Quotient im Bereich von Null, so wird die Unterbrechungsenergie nicht zugeordnet. Da wir uns im Laufe der Evaluierung dazu entschieden haben unseren Fokus auf die Unterbrechungsenergie der Netzwerkkarte zu setzen, haben wir aus Effizienzgründen θ nicht mehr berechnet. Falsche Energiezuordnungen der Unterbrechungsenergie, die durch die Netzwerkkarte initiiert wurde, ist nicht aufgetreten.

Abbildung 6.4: Energieprofile *netperf* – Unterbrechungsknoten gleich Initiatorknoten



6.3 Energieverteilung

Zur Evaluierung der Energieverteilung haben wir das unterbrechungslastige Testprogramm *netperf* und die beiden rechenlastigen Testprogramme *cpu_hock* und *loopjmp* verwendet. Ihre gemittelten Energieprofile sind aus Tabelle 6.6 ersichtlich. Für jeden Testlauf haben wir sechs *cpu_hock*-Instanzen, drei *loopjmp*-Instanzen und eine Instanz von *netperf* parallel gestartet. Ihre Ausführungszeit betrug jeweils eine viertel Stunde.

Programm	Energieprofil
netperf	35 Watt
cpu_hock	55 Watt
loopjmp	67 Watt

Tabelle 6.6: Energieprofile

Anhand des thermalen Modells können wir aus dem Energieverbrauch eines Prozessors seine Temperatur ermitteln. Umgekehrt können wir aus der Prozessortemperatur den Energieverbrauch berechnen. Aufgrund ungleichmäßiger sowie überproportionierter Kühlleistung des Testsystems liegt die maximale durch rechenlastige Applikationen herbeigeführte Prozessortemperatur zwischen $34\ °C$ und $44\ °C$ (siehe Tabelle 6.7).

CPU	0	1	2	3	4	5	6	7
Temperatur	$43\ °C$	$36\ °C$	$37\ °C$	$44\ °C$	$41\ °C$	$34\ °C$	$37\ °C$	$40\ °C$

Tabelle 6.7: Maximale Prozessortemperaturen

Um ein System zu simulieren, welches nahe an seiner Überhitzungsgrenze arbeitet, haben wir, anhand des vorliegenden thermalen Modells, die Energielimits auf Beträge gesetzt, welche eine Maximaltemperatur von $38\ °C$ approximieren. Wird diese Temperatur überschritten, so wird gedrosselt. Den Anteil solcher Drosselungen haben wir aus */proc/throttle* entnommen. Die Prozessortemperaturen haben wir periodisch ausgelesen. In Tabelle 6.8 haben wir die errechneten Energiewerte der verschiedenen Prozessoren zu einer Maximaltemperatur von $38\ °C$ aufgeführt. Wir beziehen uns hierbei auf das durch Merkel veröffentlichte thermale Modell des Testsystems [Mer05].

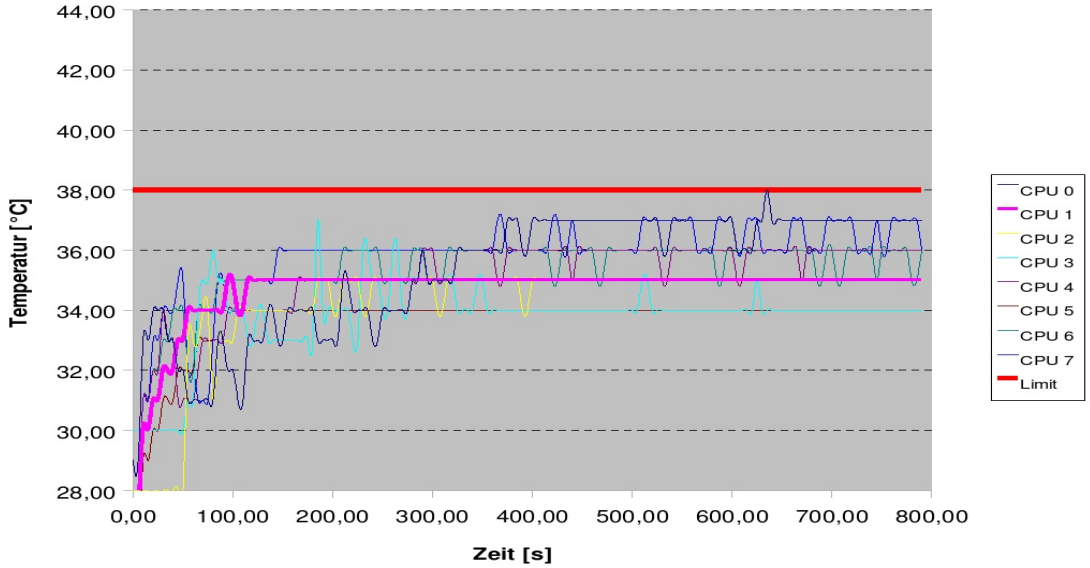
Um die Einhaltung des thermalen Gleichgewichts unserer Implementierung zu testen, haben wir den oben beschriebenen Testlauf auf Systemen mit nicht-energiegewahren Scheduler (NEAS – No Energy Aware Scheduler), energiegewahren Scheduler (EAS – Energy Aware Scheduler) und unserem unterbrechungsenergiegewahren Scheduler (EEAS – Enhanced Energy Aware Scheduler) ausgeführt. Die Energielimits haben wir so

CPU	0	1	2	3	4	5	6	7
Limit (38 $°C$)	34 W	56 W	66 W	33 W	35 W	49 W	76 W	40 W

Tabelle 6.8: Temperaturbezogene Energielimits

gewählt, dass die Maximaltemperatur von $38\ °C$ nicht übertroffen wird. Nach Starten der Testläufe haben wir in zeitlichen Perioden von 5 Sekunden die Temperaturen der Prozessoren ausgelesen. Für den Linux-Kern ohne energiegewahren Scheduler ergeben sich die in Abbildung 6.5 ersichtlichen Prozessortemperaturen. Die Prozessortemperaturen der Linux-Kerne mit EAS und EEAS sind in Abbildung 6.6 ersichtlich.

Abbildung 6.5: Thermales Gleichgewicht – Nicht Energiegewahr



Die Maximaltemperatur wird durch den Kern mit NEA-Scheduler durch keinen der Prozessoren überschritten. Die Kerne mit EAS- und EEA-Scheduler führen lediglich zu kurzen Überschreitungen der Maximaltemperaturen für den ersten Prozessor. Die Maximale Überschreitung liegt für den EA-Scheduler bei $2\ °C$ und für den EEA-Scheduler bei $1\ °C$. Dies sind keine erheblichen Überschreitungen zumal sie stets von kurzer Dauer sind.

Der Vorteil des EA- sowie des EEA-Schedulers liegen in der Art und Weise, wie die Überschreitung der Maximaltemperatur unterbunden wird. Die energiegewahren Schedu-

ler können die verbrauchte Energie über die verschiedenen Prozessoren verteilen. Dies minimiert die Anzahl der Prozessor-Drosselungen. Aus Tabelle 6.9 ist ersichtlich, dass die Überschreitung der Maximaltemperatur durch unseren EEA-Scheduler die wenigsten Drosselungen benötigt. Es folgen der EA-Scheduler und der NEA-Scheduler auf den Plätzen zwei und drei. Der nichtenergiegewahre Scheduler kann die Prozesse nicht anhand ihrer Energiecharakteristiken einordnen. Aus energiegewahrer Sicht ist sein Scheduling zufällig. Der NEA-Scheduler kann die beiden Prozesse *cpu_hock* und *loopjmp* nicht energetisch unterscheiden. Beide lasten ihre Prozessoren voll aus. Dies kann dazu führen, dass *loopjmp* zu einem Prozessor mit niedrigem maximalen Energielimit (Prozessor 0, 3, 5, 7) delegiert wird. Der Energieverbrauch von *loopjmp* übersteigt diese maximalen Energielimits stetig. Dies führt zur Drosselung des jeweiligen Prozessors. Die energiegewahren Scheduler beziehen die Energieprofile der Prozessoren ein. *loopjmp* wird als *Hot-Task* eingestuft und zu einem Prozessor mit höherem Energielimit (Prozessor 2, 6) migriert. Drosselungen werden hierdurch vermindert.

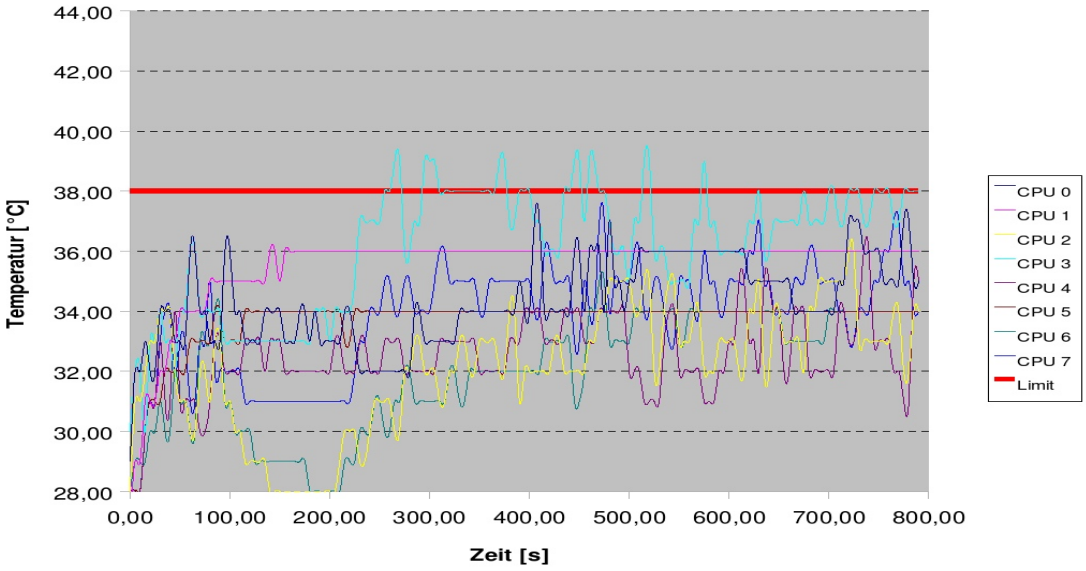
CPU	NEAS	EAS	EEAS
0	62 %	43 %	19 %
1	21 %	2 %	0 %
2	8 %	0 %	0 %
3	43 %	41 %	49 %
4	49 %	53 %	46 %
5	15 %	24 %	19 %
6	0 %	0 %	0 %
7	37 %	17 %	48 %
$\sum_{i=0}^{7}$	27 %	20 %	19 %

Tabelle 6.9: Prozessorspezifische prozentuale Drosselungen

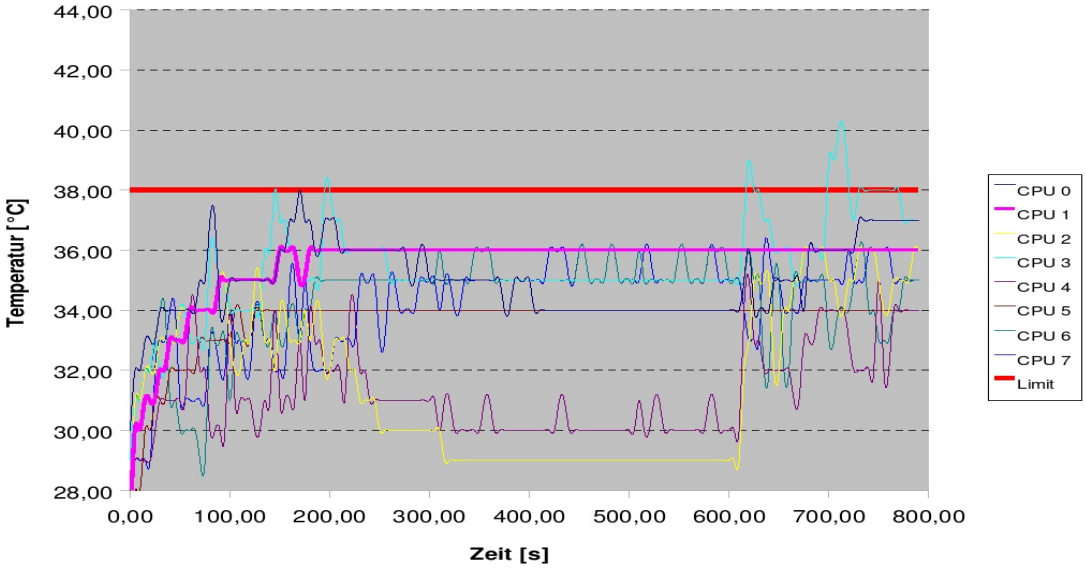
Der EEA-Scheduler verursacht eine wesentliche Minimierung der Drosselung von Prozessor 0. Dieser wird um 24 % weniger gedrosselt als dies der Fall für den EA-Scheduler ist. Dies beruht darauf, dass die durch *netperf* initiierten Unterbrechungen zu diesem Prozessor delegiert werden. Befindet sich einer der rechenlastigen Prozesse auf diesem Prozessor, so errechnet der EA-Scheduler für diesen Prozess ein geringeres Energieprofil. Hieraus resultierend ist die Einstufung dieses Prozesses als *Hot-Task* sowie seine Migration unwahrscheinlicher. Folglich wird der Prozessor häufiger gedrosselt. Der EEA-Scheduler berechnet die realen Energieprofile unter Berücksichtigung der Unterbrechungsenergie. Die unterbrechungslastigen Prozesse werden sehr wahrscheinlich als *Hot-Tasks* eingestuft und migriert. Die minimale Drosselung des Prozessors 0 ermöglicht einen höheren Durchsatz von *netperf*. Dieser liegt im Falle des EEA-Schedulers bei

$816{,}90\ \mathrm{Mbit/s}$. Im Falle des EA-Schedulers und einer Drosselung von $43\ \%$ liegt der erzielte Durchsatz lediglich bei $290{,}70\ \mathrm{Mbit/s}$. Die häufigere Drosselung des Prozessors 7 durch den EEA-Scheduler ist unter anderem die Folge der geringeren Drosselung von Prozessor 0.

Abbildung 6.6: Thermales Gleichgewicht – Energiegewahr



(a) EAS



(b) EEAS

6.4 Unterbrechungsverteilung

Zur Evaluierung unserer Unterbrechungsverteilung haben wir ausschließlich die Unterbrechungen der Netzwerkkarte herangezogen. Die explizite Betrachtung von Unterbrechungen, die durch die Festplatten oder das DVD-Rom-Laufwerk initiiert werden, erbringen keinen zusätzlichen Informationsgewinn. Ihre Verteilung verläuft analog zur Netzwerkkarte. Als Initiator der Unterbrechungen der Netzwerkkarte haben wir wie bisher *netperf* verwendet.

Unser Testszenario besteht in der expliziten Zuweisung verschiedener Instanzen von *netperf* zu verschiedenen Prozessoren mittels *taskset*. Dies haben wir separat für die beiden Modi *Default Mode* und *Performance Mode* durchgeführt. Es folgen die Auswertungen der Unterbrechungsverteilung beider Modi sowie die Erörterung des Leistungsgewinnes, der durch unsere Unterbrechungsverteilung erzielt wird.

6.4.1 Default Mode

Im *Default Mode* haben wir t_{pp} auf eine Sekunde festgelegt. Tritt IPP ein, so wird der *I-Balancer* für eine Sekunde abgeschaltet. Während dieser Zeitspanne können nur solche Unterbrechungs-Migrationen stattfinden, die durch den EEA-Scheduler initiiert werden (siehe Abschnitt 4.6).

Zum Test des *Default Mode* haben wir jeweils $2n$ Instanzen von *netperf* über einen Zeitraum von 100 Sekunden gestartet. Jeweils n dieser Instanzen wurden mittels *taskset* zum selben NUMA-Knoten delegiert. In Tabelle 6.10 findet sich die Auswertung dieses Testlaufs. Die Spalten 2 bis 5 geben die Anzahl Q_i der Unterbrechungen an, die durch Prozessor i abgearbeitet wurden. Die folgende Spalte gibt die Summe der Unterbrechungen auf allen Prozessoren während des Testlaufs an. Die zweitletzte Spalte gibt die Häufigkeit von IPP an. Die letzte Spalte gibt die Summe des Durchsatzes der $2n$ Instanzen von *netperf* an.

n	Q_0	Q_2	Q_4	Q_6	$\sum Q_i$	#IPP	$\sum$ Durchsatz
1	199996	202824	200543	196937	800300	94	398,65
2	203609	202480	197058	198343	801490	93	398,61
3	203609	202480	197058	198343	801490	93	391,46
4	208549	211591	191941	192416	804497	91	405,40

Tabelle 6.10: Unterbrechungsverteilung – *Default Mode*

Die in etwa gleichen Unterbrechungsdichten δ der einzelnen Prozessoren bleiben bei steigender Anzahl von Prozessen konstant ($\delta \approx 2000$). Folglich haben wir durch den

Default Mode eine *Round Robin*-Verteilung realisiert. Jeder Prozessor, dem Unterbrechungen zugeordnet werden können, erhält gleich viele Unterbrechungen. Weiterhin sind die Häufigkeiten von IPP sowie die Summen der Durchsätze in etwa konstant. Die geringe Zahl des Durchsatzes resultiert aus den Rahmenbedingungen des Netzwerkes zur Testzeit. Diese ließen einen Maximaldurchsatz von ca. $425\ \mathrm{Mbit/s}$ zu. Der Durchsatzverlust von ca. $20\ \mathrm{Mbit/s}$ ist auf den schlechteren Durchsatz zurückzuführen, den eine Instanz von *netperf* erzielt, wenn die durch ihn initiierten Unterbrechungen auf einem entfernten NUMA-Knoten eintreffen. Wir erläutern dies in Unterabschnitt 6.4.3. Weiterhin verringert sich der Durchsatz in Folge des Overheads, der durch die Zuteilung der entsprechenden Netzwerk-Pakete zu den einzelnen Instanzen von *netperf* entsteht.

6.4.2 Performance Mode

Im *Default Mode* haben wir t_{pp} auf zehn Sekunden festgelegt. Wir begründen dies durch den höheren Overhead, der durch die Migration der Prozesse zu demjenigen Knoten, welcher die Unterbrechungen während IPP erhalten wird, entsteht. Je geringer t_{pp}, desto höher die Kardinalität der Migrationen. Die Resultate des Testlaufs, den wir analog zum Testlauf im *Default Mode* gestaltet haben, sind in Tabelle 6.11 ersichtlich.

n	Q_0	Q_2	Q_4	Q_6	$\sum Q_i$	#IPP	$\sum$ Durchsatz
1	25	161843	626731	12409	801008	4	941,49
2	416978	55934	63296	265250	801458	5	943,79
3	511805	59	34520	255724	802108	7	942,10
4	634413	141	1850	166055	802459	4	942,36

Tabelle 6.11: Unterbrechungsverteilung – *Performance Mode*

Es zeigt sich, dass die Summe des Durchsatzes der $2n$ Instanzen von *netperf* dem maximalen Durchsatz entspricht, den wir auf unserem Testsystem erreichen können. Dieser liegt im höchsten Falle bei ca $944\ \mathrm{Mbit/s}$. Hieraus schließen wir, dass der Overhead der durch die Migration derjenigen Instanzen, welche durch Prozessoren des Knotens ausgeführt werden, dem die Unterbrechungen nicht zugeteilt wurden, keine erkennbare Minderung des Gesamtdurchsatzes impliziert. Die Unterbrechungsverteilung ähnelt, im Gegensatz zum *Performance Mode*, nicht einer *Round-Robin*-Verteilung. Erkennt der *I-Balancer* IPP, so wird die Unterbrechung zu demjenigen Knoten migriert, dessen zugeteilte Instanzen von *netperf* mehr Zeit in der durch sie initiierten Unterbrechungsbehandlung verbringen als die Instanzen von *netperf* des anderen Knotens. Die Häufigkeit von IPP liegt in etwa bei fünf. Dies resultiert daraus, dass die Instanzen von *netperf* während IPP zu ein und demselben Knoten migriert werden. Nach Ablauf von t_{pp} werden die zuvor migrierten Instanzen von *netperf* zur Einhaltung der Lastenverteilung durch den Scheduler

zurück migriert. Die durch den *I-Balancer* durchgeführte Migrationen führen immer dann zwingend zu einer ungleichen Lastenverteilung, wenn mehr als vier Instanzen von *netperf* existieren. Dies impliziert für zwei vorhandenen NUMA-Knoten, denen jeweils vier Prozessoren zugeordnet sind, mindestens einen Prozessor, der zwei Instanzen von *netperf* ausführt. Da die Prozessoren des anderen NUMA-Knotens während unserer Testläufe kaum Prozesse abarbeiten werden einige der durch den *I-Balancer* migrierten Instanzen von *netperf* durch den Scheduler zurück migriert. Durch den *Performance Mode* haben wir den höchstmöglichen Durchsatz erzielt. Der *Default Mode* garantiert eine gleichmäßigere Verteilung der Unterbrechungen. Soll ein System den maximalen Durchsatz seiner unterbrechungsinitiierenden Prozesse ermöglichen, so ist der *Performance Mode* vorzuziehen. Soll eine gleichmäßige Verteilung der Unterbrechungen realisiert werden, so muss die Wahl auf den *Default Mode* fallen.

6.4.3 Leistungssteigerung

Aufgrund der Speicherhierarchie ist es günstiger, wenn Unterbrechungen durch Prozessoren desjenigen NUMA-Knotens abgearbeitet werden, dessen Prozessoren die Unterbrechungsinitiatoren abarbeiten. Um dies zu belegen haben wir jeweils eine Instanz von *netperf* auf verschiedenen Prozessoren gestartet, während die durch *netperf* initiierten Unterbrechungen durch den ersten Prozessor des zweiten Knotens (Prozessor 4) abgearbeitet werden. Das Ergebnisse dieser Test haben wir in Tabelle 6.12 festgehalten. Die erste beiden Spalten geben den Prozessor $C_{netperf}$ und den NUMA-Knoten $\mathrm{NUMA}_{netperf}$ an, zu dem unsere Instanz von *netperf* delegiert wurde. Die folgenden beiden Spalten geben den Prozessor C_{int} sowie den NUMA-Knoten NUMA_{int} an, durch den die durch *netperf* initiierte Unterbrechungen abgearbeitet werden. Die letzte Spalte gibt den durch *netperf* erzielten Durchsatz in $\mathrm{Mbit/s}$ an.

$C_{netperf}$	$\mathrm{NUMA}_{netperf}$	C_{int}	NUMA_{int}	Durchsatz
0	0	4	1	888.63
1	0	4	1	832.96
2	0	4	1	834.60
3	0	4	1	833.51
4	1	4	1	941.25
5	1	4	1	941.24
6	1	4	1	941.24
7	1	4	1	941.25

Tabelle 6.12: Durchsatz – *netperf*

Der Test zeigt einen in etwa $10\ \%$ höheren Durchsatz derjenigen Instanzen von *net-*

perf, die durch den selben Prozessor abgearbeitet werden wie die durch die Instanzen initiierten Unterbrechungen. Die Migration unterbrechungsinitiierender Prozesse zu den Prozessoren, die die Unterbrechungen abarbeiten, ergibt einen Leistungsgewinn. Dieser Leistungsgewinn wird durch unsere Unterbrechungsverteilung gewährleistet.

6.5 Overhead

Zur Bestimmung des Overheads, der durch die Energieabschätzungen des *I-Estimators* entstehen, haben wir die Anzahl der Zyklen, die während der Energiemessung des *I-Estimators* verstreichen, mit der Anzahl der Zyklen verglichen, die während der „restlichen“ Unterbrechungsbehandlung verstreichen. Hierzu haben wir für jede registrierte Unterbrechung die Datei */proc/irq/<IRQ>/energy* ausgelesen. Die Durchschnittswerte dieser Messungen sind in Tabelle 6.13 gegeben. Die erste Zeile der Tabelle benennt das Programm, welches die Unterbrechungen initiierte. Die folgenden Spalten enthalten die Unterbrechungsnummer i, den Overhead c_O in Zyklen, die gesamten verstrichenen Zyklen während der Unterbrechungsbehandlung c_{O+R}, die Anzahl der Zyklen der Unterbrechungsbehandlung ohne Energiemessung $c_R = c_{O+R} - c_O$ und den Anteil $p_O = \frac{c_O}{c_{O+R}}$ des Overheads der Energiemessung in Prozent.

Programm	i	c_O	c_{O+R}	c_R	p_O
cat cdrom	14	5187	3019323	3014136	0
cat sda	18	6678	158043	151365	4
netperf(4)	16	7573	108844	101271	7
netperf(0)	16	6336	259271	253385	2
ping	16	6761	145397	138636	5

Tabelle 6.13: Overhead Energiemessung

Der Overhead liegt in allen Fällen unterhalb von $10\ \%$. Die überdurchschnittlich hohe Zyklenzahl der Unterbrechungsbehandlung des DVD-Rom-Laufwerks resultiert aus dem bereits erwähnten Phänomen, dass seine *Bottom Halves* unterbrochen werden. Der Overhead liegt für alle Unterbrechungen im Bereich von 5000 bis 7500 Zyklen. Es zeigt sich weiterhin, dass die Unterbrechungsbehandlung für Netzwerkunterbrechungen in Abhängigkeit zur Speicherlokalität steht. Die Netzwerkunterbrechung wurde zum zweiten NUMA-Knoten delegiert. *netperf(0)* wurde auf dem ersten Knoten ausgeführt. *netperf(4)* wurde auf dem zweiten Knoten ausgeführt. Für *netperf(4)* ergibt sich eine $2{,}5$-fache Zyklenzahl während der Unterbrechungsbehandlung. *ping* haben wird keinen festen Prozessor zugeordnet. Während der Ausführung befand sich *ping* häufiger auf dem zweiten

Knoten. Die Zyklenzahl seiner Unterbrechungsbehandlung liegt somit zwischen den Zyklenzahlen von *netperf(0)* und *netperf(4)*.

Kapitel 7

Schlussfolgerung

7.1 Ergebnis

Wir haben durch diese Arbeit den Beweis dafür erbracht, dass die explizite Berechnung und Zuordnung der durch die Unterbrechungsbehandlung konsumierten Energie zu den Prozessen, welche diese Unterbrechungen initiieren, die Entscheidungen eines energiegewahren Schedulers beeinflusst und verbessert. Wird die Unterbrechungsenergie nicht den sie initiierenden Prozessen zugeordnet, so führt dies zu falschen Entscheidungen des Schedulers. Unterscheidet sich der Energieverbrauch der Unterbrechungsbehandlung vom Energieverbrauch eines Prozesses, der diesen Energieverbrauch durch Unterbrechungsinitiierungen verantwortet, so errechnen wir aus beiden Verbrauchswerten den tatsächlichen Energiebedarf des Prozesses. Dieser Energieverbrauch spiegelt den tatsächlich durch den Prozess benötigten Energiebetrag wieder. Dies kann dazu führen, dass ein Prozess, welcher ohne diese Berücksichtigung als *Hot Task* eingestuft wurde durch Involvierung der Unterbrechungsenergie in seine Verbrauchsberechnung als *Cool Task* eingestuft wird. Die Veränderung der Einstufung vom *Hot Task* zum *Cool Task* ist ebenso möglich. Der durch uns erweiterte unterbrechungsenergiegewahre Scheduler verfügt über dieses Wissen. Die hierdurch getroffenen Entscheidungen verbessern die Einhaltung des thermalen Gleichgewichts. Hierdurch werden die Drosselungs-Kardinalitäten von Prozessoren, die ohne unseren Scheduler ihre Maximaltemperatur überschreiten, verringert.

Weiterhin haben wir durch unsere Arbeit eine dynamische und prozessspezifische Unterbrechungsverteilung realisiert. In NUMA-Multiprozessorsystemen ist lokaler Speicherzugriff effizienter als der Zugriff auf Speicher entfernter Knoten. Dies gilt auch für die Unterbrechungsbehandlung. Treten die durch einen Prozess initiierte Unterbrechungen auf einem entfernten Knoten an, so impliziert dies ineffiziente Inter-Node-Zugriffe. Dies ist aus energetischer sowie aus leistungsbezogener Sicht ineffizient. Inter-Node-Zugriffe verbrauchen mehr Energie als lokale Zugriffe. Zusätzlich verschlechtern sie den

Programm-Durchsatz. Durch unsere Arbeit haben wir Assoziationen zwischen Unterbrechungen und den sie initiierenden Prozessen geschaffen. Anhand dieser Information haben wir durch Prozess- und Unterbrechungs-Migrationen die Zahl von Inter-Node-Zugriffen minimiert. Somit haben wir den Energieverbrauch des Gesamtsystems verringert und den Durchsatz des Gesamtsystems gesteigert.

7.2 Zusammenfassung

Es besteht ein direkter Zusammenhang zwischen ausgeführten Instruktionen, Energieverbrauch und Prozessortemperatur. Übersteigt die implizierte Temperatur einen prozessorspezifischen Maximalwert, so muss der Prozessor gedrosselt werden. Um die Anzahl solcher Drosselungen zu minimieren haben wir bereits entwickelte Mechanismen verwendet, die Prozesse aufgrund ihrer Energiecharakteristik zu den verschiedenen Prozessoren eines Multiprozessorsystems migrieren. Weißt ein Prozess einen hohen Energieverbrauch auf, so wird er als *Hot Task* eingestuft. Verbraucht er wenig Energie, so wird er als *Cool Task* eingestuft. Anhand dieser Prozesseinteilung können die Prozessortemperaturen durch Prozess-Migration ausbalanciert werden.

Bisherige Ansätze solcher energiegewahren Scheduler ordnen den Prozessen jegliche Energie zu, die auf den sie ausführenden Prozessoren zwischen Prozesseinplanung und Prozessausplanung verbraucht wird. Treten während solcher Zeitscheiben Unterbrechungen auf diesen Prozessoren auf, so wird die Energie dem gerade unterbrochenen Prozess zugeordnet. Dies ist in mehrfädigen Systemen nicht immer richtig und kann zur falschen Prozesseinstufung führen. Um diesem Sachverhalt entgegen zu wirken haben wir die Schnittstelle *Systemaufrufe* erweitert und eine neue Schnittstelle zwischen Unterbrechungen und ihren Initiatoren geschaffen. Versucht ein Prozess auf ein externes Gerät zuzugreifen, so tut er dies mittels Systemaufrufen. Initiiert ein solcher Systemaufruf eine Unterbrechung, so markieren wir den Prozess mit der entsprechenden Unterbrechungsnummer. Die Unterbrechungsnummer ermitteln wir anhand des Dateideskriptors, der dem Systemaufruf übergeben wird. Findet die Unterbrechung statt, so messen wir ihren Energieverbrauch und die Unterbrechungszeit, die sie zur Abarbeitung benötigt. Zur nächsten Einplanung des Initiator-Prozesses wird ihm die Unterbrechungszeit sowie die Unterbrechungsenergie angerechnet. Weiterhin wird die Unterbrechungsenergie und die Unterbrechungszeit von den unterbrochenen Prozessen abgezogen.

Unsere Assoziationen zwischen Unterbrechungen und den sie initiierenden Prozessen haben wir zur Implementierung einer prozessspezifischen Unterbrechungsverteilung herangezogen. Initiiert ein Prozess eine Unterbrechung, so wird diese zu dem NUMA-Knoten des Prozessors migriert, der den Prozess abarbeitet. Überschreitet die Migrationsfrequenz der Unterbrechungen einen Schwellwert, so sprechen wir von Unterbrechungs-Ping-Pong (IPP). Um diesen zu verhindern stellen wir zwei Strategien zur Verfügung:

Default Mode und *Performance Mode*. Durch beide Modi werden die Unterbrechungen zu demjenigen NUMA-Knoten gesendet, dessen Prozesse die meiste Unterbrechungszeit initiieren. Im *Default Mode* wird die Unterbrechungsmigration anschließend für eine festgelegte Zeitspanne t_{pp} ausgeschaltet. Im *Performance Mode* erfolgt nach der Abschaltung der Unterbrechungsmigration die Prozessmigration aller Prozesse, die durch Prozessoren außerhalb des dedizierten Knotens abgearbeitet werden. Diese Prozesse werden zum dedizierten Knoten migriert. Während der Zeitspanne t_{pp} werden die betroffenen Prozesse durch den Scheduler zu keinem anderen NUMA-Knoten migriert. Wird die Unterbrechungsmigration nach der festgelegten Zeitspanne wieder aktiviert, so kann der Scheduler die betroffenen Prozesse wieder zu anderen Knoten migrieren. Durch Aktivierung des *Default Mode* erreichen wir die gleichmäßige Verteilung der Unterbrechungen zu allen Prozessoren. Im *Performance Mode* werden die Durchsätze aller unterbrechungsinitiierender Aktivitäten maximiert.

7.3 Zukünftige Arbeit

Unsere Arbeit realisiert die korrekte Abrechnung jeglicher durch die Prozessoren direkt verbrauchten Energie. Die Abrechnung der Energie, die durch externe Geräte verursacht wird, können wir durch unsere Methoden nicht messen. In zukünftigen Arbeiten kann eine Verknüpfung zwischen der Assoziation von Prozessen zu Unterbrechungen und der Energie, die unterbrechungsausstoßend Geräte verbrauchen, geschaffen werden. Diese Energie kann folgend den Prozessen, die die Unterbrechungen initiieren, angerechnet werden.

Die Zuordnung von Unterbrechungen zu den sie initiierenden Prozessen kann zur Abwehr von Netzwerk-Attacken verwendet werden. Insbesondere können unsere Assoziationen dazu verwendet werden DoS-Angriffe (Denial of Service) abzuschwächen. Besteht ein solcher Angriff aus höherfrequenten *ICMP-ECHO-REQUEST*-Anfragen, so kann das Netzwerk durch höherfrequentes Senden von *ICMP-ECHO-REPLYs* zum Stillstand gebracht werden. Erfolgt der Datentransfer der Netzwerkkarte durch Unterbrechungen, so wird der Prozessor, welcher die Unterbrechungen erhält, für jeden dieser Requests unterbrochen. Dies kann zum Zusammenbruch des Gesamtsystems führen. Für solche Request-Anfragen werden keine Nutzer-Prozesse involviert. Anhand unserer Assoziation zwischen Unterbrechungsinitiatoren und Unterbrechungen kann herausgefunden werden, dass die hohe Unterbrechungsfrequenz nicht durch die Prozesse des Systems initiiert werden. Anhand dieser Information können *ICMP-ECHO-REPLYs* abgeschaltet werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Arn07] Barry Arndt. Linux networking scalability on high-performance scalable servers. Technical report, IBM Systems and Technology Group, April 2007.

[BC05] Daniel Bovet and Marco Cesati. *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005.

[Bel00] Frank Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the Ninth ACM SIGOPS European Workshop 2000*, September 2000.

[BTM00] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the* 27^{th} *International Symposium on Computer Architecture (ISCA'00)*, June 2000.

[BWWK03] Frank Bellosa, Andreas Weissel, Martin Waitz, and Simon Kellner. Event-driven energy accounting for dynamic thermal management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'03)*, September 2003.

[Dat06] Intel Datasheet. Intel pentium 4 processor 6x1 sequence - on 65 nm process in the 775-land lga package supporting hyper-threading technology and intel 64 architecture, 2006.

[Dat07] Intel Datasheet. Intel pentium d processor, intel pentium processor extreme edition, intel pentium 4 processor and intelcore2 duo extreme processor thermal and mechanical design guidelines (tmdg) — for the intel pentium d processor 800 and 900 sequences, intel pentium processor extreme edition 840 , 955 , 965 , and intel pentium 4 processor 6x1 sequence and intel core2 duo extreme processor x6800 -, 2007.

[FM02] Krisztián Flautner and Trevor Mudge. Vertigo: Automatic performance-setting for linux. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI'02)*, December 2002.

[GCW95] K. Govil, E. Chan, and H. Wassermann. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking (MOBICOM'95)*, March 1995.

[GSI+02] S. Gurumurthi, A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. John. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA'02)*, February 2002.

[IB008] Irqbalance - handholding your interrupts for power and performance, 2008. Online erhältlich unter http://irqbalance.org/ - Januar 2008.

[IM03] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the* 36^{th} *Annual IEEE/ACM International Symposium on Microarchitecture (MIRCO'03)*, pages 93–104, Washington, DC, USA, 2003. IEEE Computer Society.

[Int] *IA-32 Intel Architecture Software Developer's Manual*.

[Int96] 82093aa i/o advanced programmable interrupt controller (ioapic), May 1996. Intel Corporation.

[Int03] Event counters on intel pentium 4 processors, including hyperthreading technology enabled processors, 2003. Intel Corporation.

[Kel03] Simon Kellner. Event-driven temperature-control in operating systems. Department of Computer Science, student thesis SA-I4-2003-02, April 2003.

[Lov05] Robert Love. *Linux-Kernel-Handbuch*. Addison-Wesley, 2005.

[MB06] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. In *First ACM SIGOPS EuroSys Conference*, Leuven, Belgium, April 18–21 2006.

[Mer05] Andreas Merkel. Balancing power consumption in multiprocessor systems. Master's thesis, Universität Karlsruhe, System Architecture Group, September 2005.

[PBB98] Trevor Pering, Tom Burd, and Robert Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the 1998 International Symposium on Low-Power Electronics and Design (ISLPED'98)*, pages 76–81, New York, NY, USA, June 1998. ACM Press.

[Rec02] *Technisches Schreiben*. Hanser Verlag, 2002.

[Tan02] Andrew S. Tanenbaum. *Moderne Betriebssysteme*. Prentice Hall/Pearson Studium, 2 edition, 2002.

[Tan03] Andrew S. Tanenbaum. *Verteilte Systeme*. Prentice Hall/Pearson Studium, 1 edition, 2003.

[Tip94] Paul A. Tipler. *Physik*. Spektrum Verlag, 1994.