Betriebssystemmechanismen für eine taktfrequenzgesteuerte Energieverwaltung

Studienarbeit im Fach Informatik

vorgelegt von

Martin Alt

geboren am 12. Mai 1978 in Nürnberg

Angefertigt am Institut für Informatik (IV)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer:	*Prof. Dr. rer. nat. Fridolin Hofmann* *Dr. Ing. Frank Bellosa*
Beginn der Arbeit:	14. April 2000
Abgabe der Arbeit:	31. Juli 2000

Ich versichere, dass ich meine Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 30. Juli 2000 ______________________________

Inhaltsverzeichnis

1. Einleitung

Auf Grund der gestiegenen Anforderungen an die Rechenleistung werden immer schnellere Prozessoren in mobile, batteriebetriebene Systeme eingebaut. Der Energieverbrauch eines Prozessors steigt jedoch mit dessen Geschwindigkeit an. Deshalb können bislang nur Prozessoren mit begrenzter Leistungsfähigkeit, aber auch begrenztem Energieverbrauch in solchen Systemen eingesetzt werden, um eine ausreichend lange Batterielebensdauer zu gewährleisten.

Andererseits wird die volle Rechenleistung des Prozessors nur selten benötigt. Statt dessen steht der Prozessor – zumindest im interaktiven Betrieb – häufig lange Perioden leer und es werden nur sporadisch hohe Anforderungen an die Rechenkapazität gestellt.

Deshalb bietet sich die Möglichkeit, die Batterielebensdauer eines mobilen Systems zu erhöhen, indem die Taktfrequenz des Prozessors – und damit dessen Leistungsaufnahme – in Perioden mit geringer Auslastung gedrosselt wird. Zu diesem Zweck muss das Betriebssystem Mechanismen der Energieverwaltung zur Verfügung stellen, die dafür sorgen, dass der Prozessor weder zu langsam läuft, noch zu viel Energie verbraucht.

In verschiedenen Arbeiten ([WWDS94, GCW95, Mar99]) wurden Ansätze zur Energieverwaltung verfolgt, deren Ziel es ist, die mögliche Anzahl von Rechenoperationen pro Batterieentladung zu maximieren. Zu diesem Zweck wird die Geschwindigkeit des Prozessors gedrosselt, sobald die anfallende Rechenlast sinkt und erhöht wenn der Rechenzeitbedarf steigt. Dementsprechend hängen Geschwindigkeit, Leistungsaufnahme und Batterielebensdauer nur von der Auslastung des Systems ab.

In dieser Arbeit soll ein Ansatz verfolgt werden, bei dem die Geschwindigkeit statt dessen durch die Angabe eines Verbrauchslimits bzw. einer Batterielebensdauer durch den Benutzer bestimmt wird. Zusätzlich soll die Geschwindigkeit nicht für den ganzen Prozessor festgelegt werden, sondern prozessbasiert (vgl. [Bel99]). Der Prozesskontext muss dementsprechend um einen Eintrag für die Geschwindigkeit erweitert werden und die Taktfrequenz des Prozessors muss bei jedem Kontextwechsel verändert werden. Dadurch können verschiedenen Prozessen unterschiedliche Geschwindigkeiten zugeordnet werden, um so beispielsweise Echtzeitanforderungen erfüllen zu können oder um interaktive Prozesse zu bevorzugen.

Ziel dieser Arbeit ist es, eine Strategie zur Geschwindigkeitsbestimmung zu implementieren, die zum einen für die Einhaltung eines vorgegebenen Energieverbrauchs sorgt, zum anderen interaktive Prozesse bevorzugt. Im praktischen Teil der Arbeit soll ein Linux-System um die entsprechenden Komponenten der Energieverwaltung erweitert werden.

Um einen vorgegeben Verbrauch möglichst genau einzuhalten, ist es nötig, die Auswirkungen einer Geschwindigkeitsänderung gut abschätzen zu können. Im folgenden Kapitel wird deshalb zunächst ein Überblick über die durch Taktfrequenzdrosselung möglichen Einsparungen gegeben. Außerdem wird untersucht, von welchen anderen Faktoren der Energieverbrauch eines Prozessors abhängt.

In Kapitel drei wird eine Strategie zur Festlegung der Geschwindigkeit entwickelt, die die oben genannten Anforderungen erfüllt. Außerdem werden einige allgemeine Überlegungen zur Implementierung angestellt, insbesondere was die Bestimmung des Energieverbrauchs einzelner Prozesse betrifft.

In Kapitel vier wird schließlich die Implementierung der genannten Strategie beschrieben. Abschließend werden die Ergebnisse der Implementierung kurz dargestellt.

2. Energieverwaltung im Betriebssystem

2.1. Ziele

Das Ziel der Energieverwaltung ist es, die Batterielebensdauer eines mobilen Systems zu erhöhen, ohne dabei die Dienstgüte wesentlich zu verringern. Zu diesem Zweck wird die Taktfrequenz des Prozessors verringert sobald die Anforderungen an die Rechenleistung dies zulassen. Durch die Senkung der Taktfrequenz wiederum sinkt auch die Leistungsaufnahme des Prozessors und somit steigt die Batterielebensdauer.

Die Drosselung der Taktfrequenz hat zwei entscheidende Auswirkungen: Zum einen wird die Rechenleistung vermindert, zum anderen sinkt die Verlustleistung. Es gilt deshalb einen Kompromiss zu finden, zwischen den Anforderungen an die Rechenleistung einerseits und den Anforderungen an die Batterielebensdauer andererseits. Dabei können unterschiedliche Ansätze verfolgt werden. Zum einen kann versucht werden, die „optimale" Geschwindigkeit automatisch zu finden. Beispielsweise kann die Taktfrequenz bei hoher Rechenlast angehoben und bei niedriger gesenkt werden. Auf der anderen Seite gibt es die Möglichkeit Benutzer bzw. Anwendung selbst über das Verhältnis zwischen Rechen- und Verlustleistung entscheiden zu lassen, z. B. durch Angabe eines maximalen Energieverbrauchs.

In jedem Fall ist es jedoch ein Ziel, durch die Senkung der Geschwindigkeit eine Verringerung der Leistungsaufnahme des Prozessors zu erreichen. Wie die Verlustleistung mit der Geschwindigkeit zusammenhängt wird in den folgenden Abschnitten noch genauer dargestellt. Generell gilt es jedoch zu beachten, dass die Senkung der Geschwindigkeit nicht nur die Leistungsaufnahme verringert, sondern auch die Rechenzeit erhöht! Da die verbrauchte Energie sich aus Verlustleistung mal Rechenzeit zusammensetzt, verringert sich die benötigte Energie nur dann, wenn die Verlustleistung stärker sinkt als die Rechenzeit steigt. Anderenfalls heben sich beide Effekte auf, oder die benötigte Energie erhöht sich sogar.

Zur Bewertung des Energieverbrauchs bei verschiedenen Geschwindigkeiten wurde deshalb in [WWDS94] als Maß für das Verhältnis zwischen Energieverbrauch und Rechenleistung „Million Instructions per Joule" (MIPJ) vorgeschlagen, also Rechenleistung ([MIPS]) durch Verlustleistung ([W]). So lange die MIPJ-Rate unverändert bleibt, sind verschiedene Geschwindigkeiten – vom Energieverbrauch her – gleichwertig. Ansonsten ist der Geschwindigkeit mit der größten MIPJ-Rate der Vorzug zu geben.

Das Ziel einer Energieverwaltung ist es also nicht allein, die Batterielebensdauer zu erhöhen. Denn sinken die MIPJ bei einer Drosselung der Taktfrequenz, so mag sich zwar die Batterielebensdauer erhöhen, die bei einer Batterieentladung insgesamt geleistete Rechenarbeit sinkt jedoch. Statt dessen sollte versucht werden, die Anzahl der möglichen Rechenoperationen pro Batterieentladung zu maximieren.

2.2. Effizienzsteigerung durch Taktfrequenzreduzierung

Einige modernere Prozessoren (StrongARM1100 [Int99], PowerPC860 [Mot98], Intel Mobile Pentium III [Int00b, Int00a], AMD Èlan SC410 [AMDb]) die für den Einbau in batteriebetriebene Systeme konzipiert wurden, bieten die Möglichkeit die Taktfrequenz zu senken um so Energie zu sparen. Im folgenden Abschnitt soll dargestellt werden, welche Auswirkungen eine Taktfrequenzänderung auf die Verlustleistung eines Prozessors hat.

Als Testsystem wurde ein mit dem AMD Èlan SC410 Prozessor ausgestatteter PC verwendet, der unter Linux 2.2.12 läuft. Die Taktfrequenz des Élan Prozessors kann von maximal 100 MHz auf 66, 33, 16, 8, 4, 2 oder 1 MHz gesenkt werden. Bei dem in dieser Arbeit eingesetzten System ist die Taktfrequenz jedoch, wegen der hohen Wärmeentwicklung bei 100 MHz, auf 66 MHz begrenzt.

Um Rechenleistung und elektrische Verlustleistung des Prozessors bei Ausführung verschiedener Maschinenbefehle zu messen, wurden Microbenchmarks aus synthetischem Assembler-Code eingesetzt. Dabei wurde darauf geachtet, dass die jeweiligen Benchmarkprogramme klein genug sind, um vollständig in den 8 kB großen Cache des Prozessors geladen werden zu können. Dadurch wird die Anzahl der Hauptspeicherzugriffe minimiert. Weitere Einzelheiten zum AMD Èlan SC410 sind in Abschnitt 4.1 angegeben, die Durchführung der Messungen wird in Anhang A ausführlich beschrieben.

Lässt man alle anderen Parameter, wie z. B. die Versorgungsspannung unverändert, und misst die Leistungsaufnahme eines Prozessors bei verschiedenen Geschwindigkeiten, so kann man einen annähernd linearen Zusammenhang feststellen. Das Ergebnis einer solchen Messung ist für den AMD Èlan SC410 in der Abb. 2.1 wiedergegeben.

Gemessen wurde der Energieverbrauch von Ganzzahladdition („add"), „mov" Befehl (kopiert den Inhalt eines Register in ein anderes Register), logischem Und („and") und Negation („not"). Die Werte wurden bei Taktfrequenzen von 1, 2, 4, 8, 16, 33 und 66 MHz gemessen (die Verbindung der Messwerte in der Abbildung dient lediglich der Darstellung).

Eine Verringerung der benötigten Energie pro Rechenoperation ist somit nicht möglich, da ja auch die Rechenleistung linear mit der Taktfrequenz sinkt. Deshalb hätte eine Halbierung der Taktfrequenz zwar eine Halbierung der Verlustleistung zu Folge, zur Bewältigung der gleichen Rechenarbeit wäre aber auch doppelt so viel Zeit nötig, so dass sich beide Effekte aufheben.

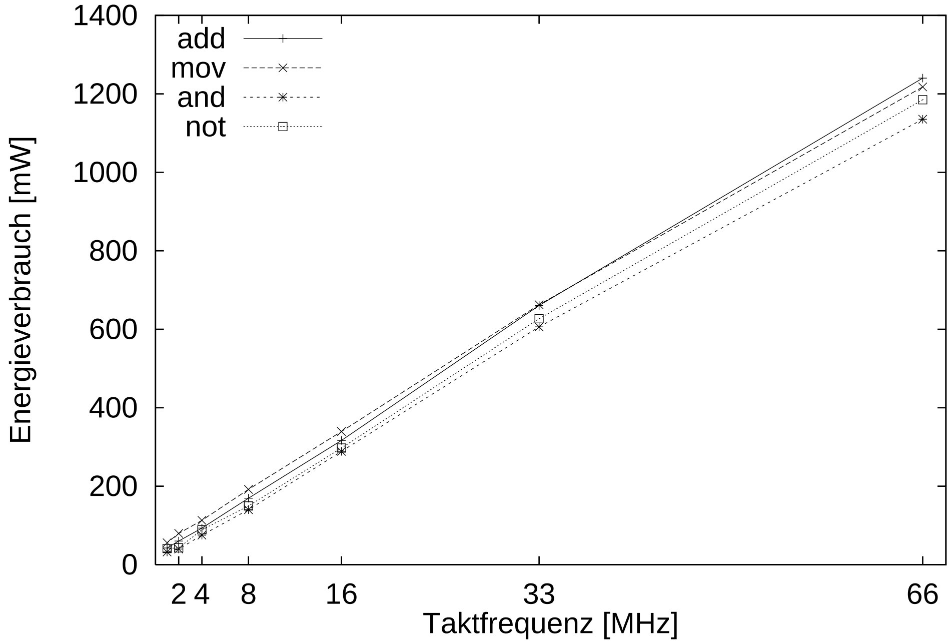

Abbildung 2.1.: Leistungsaufnahme des AMD Èlan SC410 bei verschiedenen Taktfrequenzen.

Auch die MIPJ-Rate bleibt somit weitgehend konstant. Deshalb scheint sich aus einer Geschwindigkeitsreduzierung zunächst keine höhere Anzahl an Rechenoperationen pro Batterieentladung zu ergeben. Es existieren jedoch verschiedene Faktoren, die eine Erhöhung der Rechenkapazität pro Batterieentladung dennoch zulassen.

Zum einen steigt die Rechenleistung der CPU, bedingt durch die Speicherzugriffszeit nicht linear mit der Taktfrequenz. Vielmehr ist bei hoher Geschwindigkeit ein Abfall der Rechenleistung zu beobachten. Der Zusammenhang zwischen Rechenleistung und Taktfrequenz wird im Abschnitt 2.2.2 eingehend untersucht.

Außerdem hat eine geringere Verlustleistung des Prozessors eine nicht unerhebliche Auswirkung auf die Batteriekapazität. Diese Effekte werden im Abschnitt 2.2.3 ausführlich diskutiert.

Des weiteren ist es möglich, die Spannung mit der Taktfrequenz zu verringern, wodurch der Energieverbrauch überproportional zur Rechenleistung sinkt. Diese Möglichkeit wird in Abschnitt 2.2.4 näher betrachtet.

Zunächst soll jedoch die Auswirkung des Stromverbrauchs des Gesamtsystems, sowie konstanter Anteile im Verbrauch des Prozessors näher betrachtet werden.

2.2.1. Auswirkungen des Energieverbrauchs des Gesamtsystems

Bisher wurde nur der Energieverbrauch der CPU, und auch hier nur der variable Anteil dargestellt. Der Energiebedarf anderer Teile des Systems, wie z. B. der Grafikkarte, hängt jedoch nur in geringem Maße oder gar nicht von der Taktfrequenz des Prozessors ab. Berücksichtigt man auch den Stromverbrauch dieser Komponenten, so ergibt sich die Verlustleistung des Gesamtsystems aus $P = P_c + f \cdot P_d$, wobei P_c der konstante Anteil ist und P_d die Steigung mit der Taktfrequenz angibt.

Selbstverständlich bleibt dieser konstante Anteil nicht ohne Auswirkung auf die MIPJ-Rate. Während die Verminderung der Rechenleistung bei rein taktfrequenzabhängigem Energieverbrauch durch die entsprechende Verminderung des Stromverbrauchs aufgehoben wurde, sinkt dieser nun nicht mehr im gleichen Maße wie die Rechenleistung. Deshalb sinkt auch die MIPJ-Rate – die selbe Rechenarbeit kostet also bei geringerer Taktfrequenz mehr Energie als vorher. In Abb. 2.2 ist die MIPJ-Rate für den AMD Èlan SC410 bei verschiedenen Taktfrequenzen dargestellt. Aufgeführt sind die MIPJ für Integer-Addition („add"), Ausführung einer leeren Schleife („jmp") und Cache-Zugriffe („cache"). Dabei wurde nebst dem Energieverbrauch des Prozessors auch der von Hauptspeicher, Flashdisk, Graphikkarte und Netzwerkadapter berücksichtigt. Deshalb ist bei geringen Taktfrequenzen ein deutlicher Abstieg zu erkennen, der u. a. vom konstanten Anteil P_c des Gesamtsystems herrührt. Unter diesen Voraussetzungen ist also eine Taktfrequenzsenkung nicht rentabel.

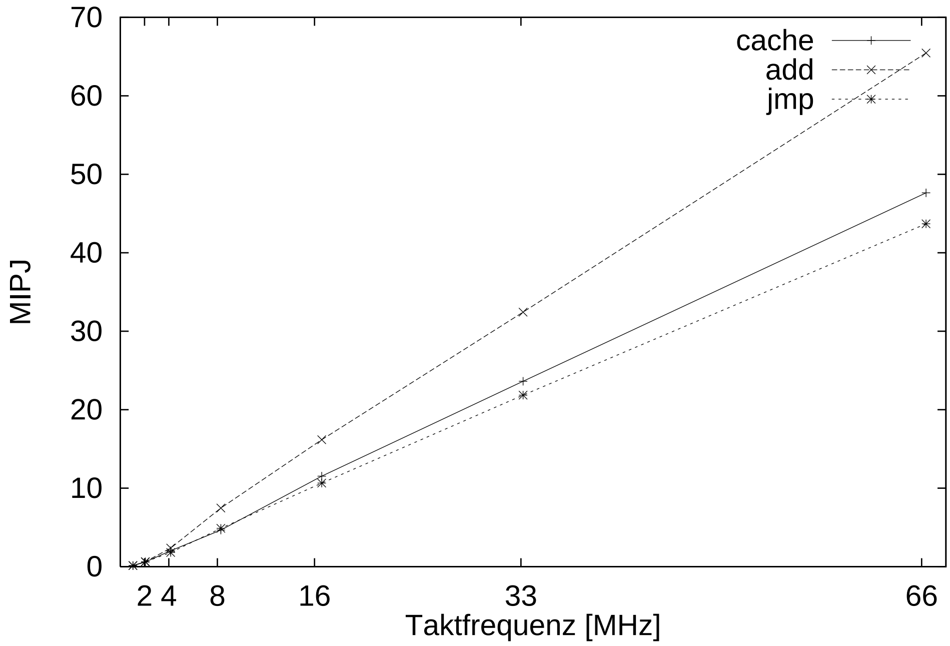

Abbildung 2.2.: MIPJ Rate des AMD Èlan SC410 bei Berücksichtigung des Gesamtenergieverbrauchs

Allerdings wurde bei dieser Betrachtungsweise implizit die Annahme getätigt, dass das System stets voll ausgelastet ist und nur so lange Strom verbraucht, wie gerechnet wird. Zumindest im interaktiven Betrieb ist diese Annahme natürlich unrealistisch, da dort ein großer Teil der Zeit damit verbracht wird, auf Eingaben des Benutzers zu warten. Auch wenn man voraussetzt, dass der Prozessor selbst keinen (oder nur verschwindend wenig) Strom verbraucht, so lange keine Berechnungen durchgeführt werden, so hat das Gesamtsystem doch weiterhin eine gewisse Verlustleistung. Diese wird sich nur wenig von der oben genannten Konstante P_c unterscheiden .

Legt man also eine feste Betriebsdauer T zu Grunde, bleibt der durch P_c verursachte Energieverbrauch auch bei einer Taktfrequenzreduzierung gleich (nämlich $P_c \cdot T$). Es ändert sich nur der variable Anteil und die maximal mögliche Anzahl von Befehlen die innerhalb dieser Zeit ausgeführt werden kann. Der variable Anteil am Energieverbrauch beträgt $f \cdot P_d \cdot t(f)$, wobei $t(f)$ die Rechenzeit bei Taktfrequenz f ist. Geht man davon aus, dass die innerhalb T anfallende Rechenarbeit konstant ist und sich die Rechenleistung proportional zur Taktfrequenz verhält, so ist $t(nf) = t(f)/n$. Deshalb bleibt der variable Energieverbrauch in beiden Fällen gleich: $f \cdot P_d \cdot t(f) = nf \cdot P_d \cdot t(f)/n$. Somit macht es bei fester Betriebsdauer – vom Energieverbrauch her – keinen Unterschied mit welcher Taktfrequenz das System läuft, vorausgesetzt die geleistete Rechenarbeit bleibt gleich.

Somit ist es stark von der Systemauslastung abhängig, wie stark das Verhältnis zwischen variablem und konstantem Anteil die Batterielebensdauer beeinflusst. Um eine Energieeinsparung zu erreichen muss die MIPJ-Rate des Prozessors bei Taktfrequenzsenkung jedoch allgemein um so stärker steigen, je höher die Verlustleistung des restlichen Systems ist.

Da sich eine Geschwindigkeitssenkung also nur lohnt, wenn der Verbrauch des Gesamtsystems relativ gering ist, erscheint eine systemweite Energieverwaltung sinnvoll, wie sie beispielsweise in [IMT99] beschrieben wird. Denn wenn nicht nur die Leistungsaufnahme des Prozessors gesenkt wird, sondern auch die aller anderen Komponenten, so sind wesentlich höhere Einsparungen zu erwarten.

2.2.2. Zusammenhang von Taktfrequenz und Rechenleistung

Bisher wurde von einem direkt proportionalem Zusammenhang zwischen Rechenleistung (ausgedrückt etwa in MIPS) und Taktfrequenz (in MHz) ausgegangen. Dem lag die Annahme zu Grunde, das die mittlere Anzahl von Taktzyklen pro Befehl („Cycles per Instruction", CPI) unabhängig von der Taktfrequenz ist und die Ausführungszeit deshalb proportional zur Geschwindigkeit. Tatsächlich bleiben die CPI bei verschiedenen Geschwindigkeiten jedoch nicht konstant.

In der Abbildung 2.3 sind die für die Ausführung verschiedener Befehle (Ganzzahl-Addition („add"), unbedingter Sprung („jmp") und Cache-Zugriff („cache")) benötigten Taktzyklen bei verschiedenen Geschwindigkeiten aufgetragen.

Bei allen Befehl ist ein deutlicher Anstieg der benötigten Taktzyklen bei niedriger Taktfrequenz zu erkennen. Dieser Anstieg ist auf verschiedene Faktoren zurückzuführen.

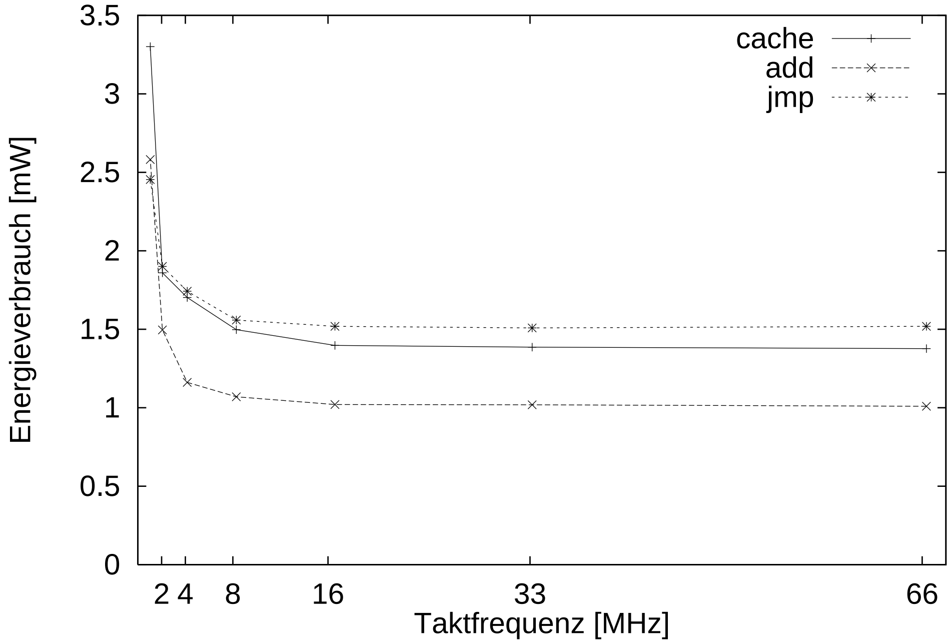

Abbildung 2.3.: Mittlere Anzahl von Taktzyklen pro Befehl.

So entsteht beispielsweise durch Interrupt-Behandlung und Kontextwechsel eine gewisse betriebssystembedingte Rechenlast. Diese ist, da die Zeitscheibengröße und die Häufigkeit der Interrupts (z. B. Zeitgeber-Interrupts) nicht mit der Taktfrequenz abgesenkt wird, weitgehend konstant. Während sie jedoch bei hoher Taktfrequenz im Vergleich zur Rechenkapazität sehr gering ausfällt, schlägt sie bei sehr niedriger Taktfrequenz deutlich zu Buche. Dadurch steht auf Benutzerebene weniger Rechenzeit zur Verfügung. Somit ändert sich zwar nicht wirklich die Anzahl der pro Befehl benötigten Taktzyklen, die *mittlere* Anzahl der Taktzyklen pro Befehl in der Anwendung erhöht sich dennoch.

Außerdem kommt beim AMD Èlan SC410 die Tatsache hinzu, dass nur ein 8 kB großer Cache existiert, der sowohl als Daten- als auch als Befehlscache dient. Die gemessenen Benchmarks sind so ausgelegt, dass sie vollständig in den Cache passen und somit möglichst wenig Speicherzugriffe benötigen. Sobald der Prozess jedoch durch das Betriebssystem unterbrochen wird, werden Teile des Prozesses aus dem Cache verdrängt und müssen später wieder eingelagert werden. Somit wird der entstehende Overhead auch von der Speicherzugriffszeit beeinflusst.

Die mittlere Anzahl von Taktzyklen für einen Hauptspeicherzugriff (ohne Cache) verändert sich ebenfalls mit der Taktfrequenz. In Abbildung 2.4 sind die CPI für Hauptspeicherzugriffe (Lesezugriff) aufgetragen. Hier ist sowohl bei niedriger als auch bei hoher Geschwindigkeit ein deutlicher Anstieg der benötigten Taktzyklen erkennbar.

Der Anstieg bei hoher Geschwindigkeit ist auf die – relativ zur CPU-Taktfrequenz – geringer werdende Speicherzugriffsgeschwindigkeit zurückzuführen. Die in Tabelle 2.1

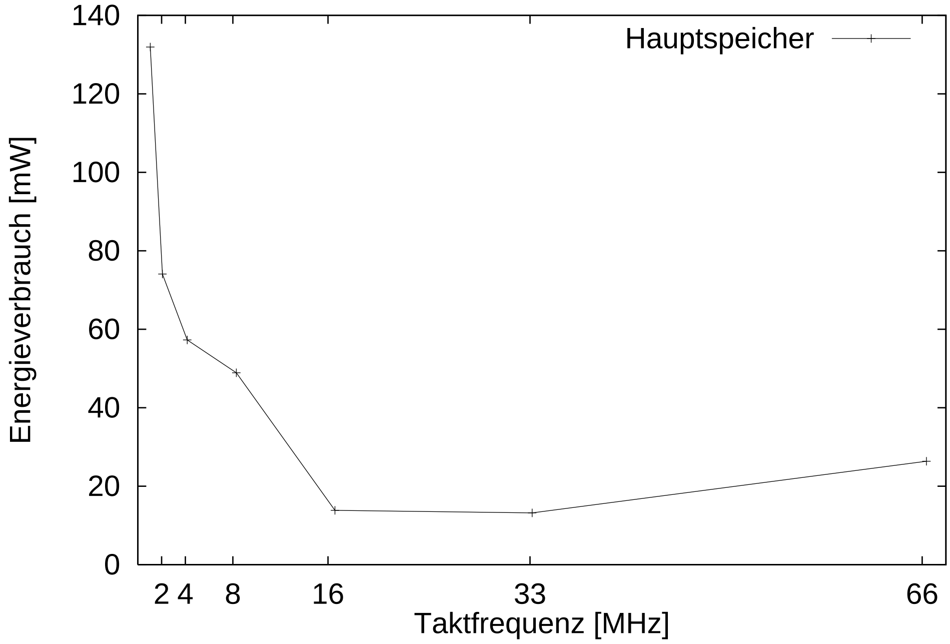

Abbildung 2.4.: Mittlere Anzahl von Taktzyklen pro Speicherzugriff

aufgeführten Werte für Prozessor- und Speichertakt sind dem Datenblatt für den AMD Èlan SC410 ([AMDa]) entnommen. Wie man sieht beträgt der Speichertakt unter 33 MHz Prozessortakt stets das doppelte der CPU-Frequenz. Bei 66 und 100 MHz allerdings bleibt der Speicher zurück, wodurch es zu entsprechenden Verzögerungen bei Speicherzugriffen kommt.

Prozessortakt	[MHz]	1	2	4	8	16	33	66	100
Speichertakt	[MHz]	2	4	8	16	33	66	66	66

Tabelle 2.1.: Speichertakt bei unterschiedlichem Prozessortakt.

Dadurch werden die CPI bei 33 MHz minimal und bleiben auch bei 16 MHz weitgehend gleich. Bei 8 MHz hingegen ist ein spontaner Anstieg der benötigten Taktzyklen zu erkennen, der durch die Art der Speicheradressierung hervorgerufen wird: Der im AMD Èlan SC410 integrierte DRAM-Controller adressiert den Speicher bei Taktfrequenzen über 8 MHz im Seitenmodus (vgl. [AMDb]). Dadurch müssen nicht für jeden Speicherzugriff Zeilen- und Spaltenadresse übertragen werden, sondern die Zeilenadresse wird für mehrere Zugriffe auf die selbe Zeile nur einmal übertragen. Da stets eine ganze Cache-Zeile gelesen wird, finden immer mehrere Speicherzugriffe hintereinander statt, so dass eine erneute Zeilenadressierung für jeden Speicherzugriff beträchtliche Geschwindigkeitseinbußen zur Folge hat. Deshalb werden für einen Speicherzugriff bei 8 MHz wesentlich mehr Taktzyklen als bei 16 MHz benötigt.

Der weitere Anstieg der CPI unter 8 MHz dürfte unter anderem auf Speicherauffrischungszyklen zurückzuführen sein. Der DRAM Speicher muss alle 15,6μs aufgefrischt werden. Bei einer Geschwindigkeit von 66 MHz entspricht das einem Auffrischungs-Intervall von ca. 1030 Takten, bei einer Taktfrequenz von 1 MHz erfolgt ein Auffrischungszyklus schon alle 15,6 Takte.

Somit sinkt die Rechenleistung im Verhältnis zur Taktfrequenz also sowohl bei hohen als auch bei sehr niedrigen Taktfrequenzen. So lange die Geschwindigkeit nicht zu sehr gesenkt wird, kann eine Reduzierung der Taktfrequenz also durchaus zu einer Energieeinsparung führen. Dieser Effekt wird vor allem bei sehr schnellen Rechnern, deren Taktfrequenz weit über der Geschwindigkeit des Speichers liegt, eine große Rolle spielen. Es bleibt die Frage, in welcher Größenordnung die erwartete Effizienzsteigerung liegt, und welche Geschwindigkeit – im Sinne des Energiebedarfs – optimal ist.

„Optimale" Geschwindigkeit bei Speicherzugriffen

Die speicherbedingte Verminderung der Rechenleistung bei hohen Taktfrequenzen hängt von einer Reihe von Faktoren ab: Zum einen die relative Häufigkeit von Speicherzugriffen, die selbst wiederum von Programmstruktur und Cache-Größe (bzw. Fehlzugriffsrate) abhängig ist. Zum anderen spielt natürlich die Speicherzugriffsgeschwindigkeit, bzw. die Differenz zwischen Prozessor- und Speichertakt eine wesentliche Rolle.

Während Cache-Größe und Speichergeschwindigkeit leicht anzugeben sind, kann man die Effekte von Programmstruktur und anderen Faktoren, die die Häufigkeit von Cache-Misses und Speicherzugriffen bestimmen, nur sehr schlecht abschätzen. Sehr allgemein kann man – in Anlehnung an das Ahmdahlsche Gesetz ([HP94]) – folgenden Zusammenhang feststellen ([Mar99]): Der Speedup S bei der Erhöhung der Geschwindigkeit auf Taktfrequenz f gegenüber Taktfrequenz f_1 beträgt

$$S = \frac{1}{m + \frac{f_1(1-m)}{f}}$$

wobei m der Anteil der Ausführungszeit bei Taktfrequenz f_1 ist, der von Speicherzugriffen herrührt. Voraussetzung ist außerdem, dass die Speicherzugriffsgeschwindigkeit bei beiden Taktfrequenzen gleich bleibt.

Somit gilt beispielsweise für die Erhöhung der Geschwindigkeit von 33 auf 66 MHz beim AMD Èlan SC410 $S = 1/\left(m + \frac{(1-m)}{2}\right)$. Werden im Mittel 1,33 Speicherzugriffe und 2 Taktzyklen durchschnittliche Ausführungszeit pro Befehl, 8 Taktzyklen pro Hauptspeicherzugriff und eine Cache-Miss Rate von 10% angenommen, so sind pro Befehl ca. 3,06 Taktzyklen nötig (bei 33 MHz). Davon entstehen 1,06 Taktzyklen durch Hauptspeicherzugriffe. Damit ergibt sich für m etwa 0,346, und der Speedup bei einer Verdoppelung der Geschwindigkeit auf 66 MHz läge bei $S \approx 1,486$, statt $S = 2$ bei „idealem Speicher".

Der Zusammenhang zwischen dem Anteil der Speicherzugriffe an der Rechenzeit m und dem Speedup S bei Verdoppelung der Taktfrequenz (also idealer Speedup $S = 2$) ist in Abbildung 2.5 dargestellt.

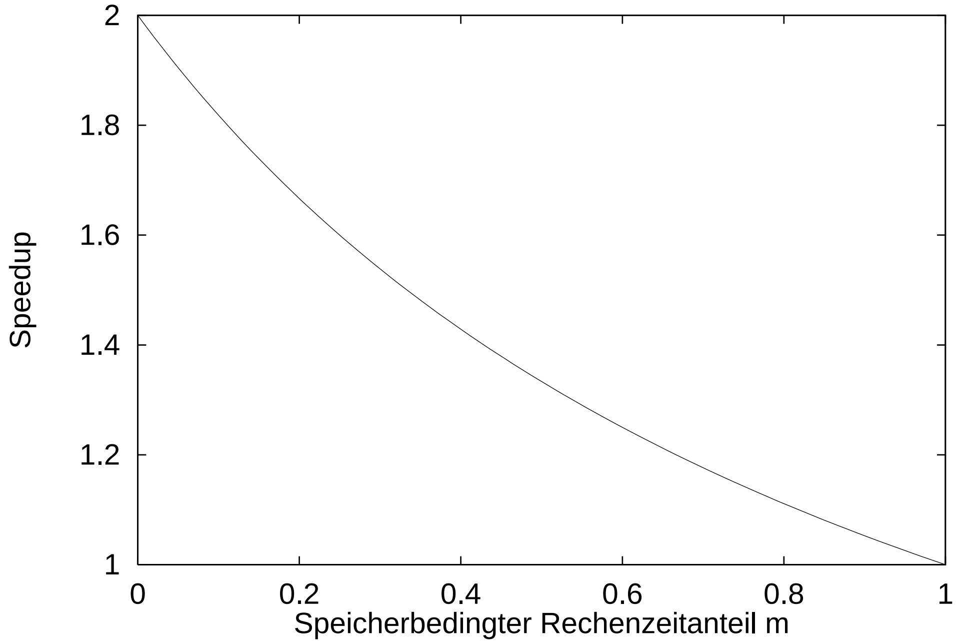

Abbildung 2.5.: Speedup bei Verdoppelung der Taktfrequenz in Abhängigkeit des speicherbedingten Anteils der Rechenzeit m.

Hängt die Verlustleistung des Prozessors nur von der Geschwindigkeit ab, ohne oder mit nur geringem taktfrequenzunabhängigem Anteil (vgl. Abschnitt 2.2.1), so erscheint es durchaus sinnvoll die Taktfrequenz so weit zu senken, bis das Verhältnis zwischen Prozessor- und Speichertakt nicht mehr verbessert werden kann. Ob eine solche Drosselung der Geschwindigkeit aus Sicht der Anwendung noch annehmbar ist sei vorerst dahingestellt, diese Frage wird in Abschnitt 2.4 ausführlich diskutiert.

Betrachtet man den Energiebedarf des Gesamtsystems, inklusive der geschwindigkeitsunabhängigen Anteile, so darf die Geschwindigkeit natürlich nicht zu weit gesenkt werden. Schließlich hat nun eine Verlangsamung des Rechners zwei entgegengesetzte wirkende Folgen: Auf der einen Seite eine Verminderung der Verlustleistung, auf der anderen Seite die Erhöhung der Bearbeitungsdauer. Da die MIPS-Rate proportional zum Speedup steigt gilt für die MIPJ-Rate $MIPJ = MIPS_{f_1} \cdot S/P$, wobei S der Speedup, P die Verlustleistung und $MIPS_{f_1}$ die Rechenleistung bei der Ausgangsgeschwindigkeit f_1 ist. Die theoretisch optimale Geschwindigkeit findet man durch Ableiten und Null setzen. Für $P = P_c + f \cdot P_d$ gilt:

$$f_{opt} = \sqrt{f_1 \frac{(1-m)}{m} \frac{P_c}{P_d}}$$

Wie zu erwarten hängt die optimale Geschwindigkeit nun nicht nur von der Speichergeschwindigkeit und dem Anteil der Speicherzugriffe ab, sondern auch vom Verhältnis zwischen konstantem und variablem Verbrauchsanteil. Da P_c jedoch meist erheblich größer ist als P_d wird die optimale Geschwindigkeit im allgemeinen so hoch sein, dass sich das Senken der Taktfrequenz – allein auf Grund der Speicherlatenz – nur selten lohnt.

Für das oben betrachtete Beispiel läge die optimale Geschwindigkeit bei ca 84 MHz, also oberhalb der maximal möglichen Taktfrequenz (für $P_c = 2016$mW und $P_d = 18$mW – vgl. Anhang A und Abschnitt 2.3). Tatsächlich müsste der Anteil m mehr als 46% betragen, um eine Senkung der Taktfrequenz auf 33 MHZ zu rechtfertigen.

Obwohl der Unterschied zwischen Speicher- und Prozessortakt in diesem Beispiel nicht so groß war, wie es bei sehr schnellen Prozessoren oft der Fall ist, werden die Auswirkungen der Speicherzugriffe allein eine Verlangsamung des Prozessors nur selten rechtfertigen, vor allem dann nicht, wenn große Pufferspeicher vorhanden sind. Bei der Wahl der Taktfrequenz sollten diese Effekte dennoch berücksichtigt werden. Immerhin ist ein gewisses Einsparungspotential vorhanden, das zusammen mit den in den folgenden Abschnitten beschriebenen Auswirkungen eine Geschwindigkeitsreduzierung durchaus legitimieren kann.

2.2.3. Auswirkungen auf die Batterielebensdauer

In den vorherigen Abschnitten wurde das eigentliche Ziel, die Maximierung der Rechenoperationen pro Batterieentladung, gleichgesetzt mit der Minimierung des Energieverbrauchs pro Rechenoperation, also der MIPJ-Rate. Dabei wurde stillschweigend vorausgesetzt, dass die Batteriekapazität C konstant bleibt. Diese gibt an, wie viel Energie (meist in Wh, also 3600 Joule) eine Batterie speichern kann. Die Lebensdauer T einer idealen Batterie beträgt bei zeitlich konstantem Stromfluss I und Spannung U, bzw. bei konstanter Last P somit $T = \frac{C}{U \cdot I} = \frac{C}{P}$.

Die Anzahl der Befehle pro Batterieentladung („Computations per Discharge", CPD) ergibt sich aus $CPD = MIPS \cdot T = MIPS \cdot \frac{C}{P} = C \cdot MIPJ$, vorausgesetzt P und C wurden in Joule statt in Wh angegeben. Somit genügt es bei lastunabhängigem C die MIPJ zu betrachten.

Tatsächlich hängt die Batteriekapazität jedoch von der Entladegeschwindigkeit, also von der Höhe des Entladestroms bzw. der Leistungsaufnahme des angeschlossenen Prozessors ab, und ist keineswegs konstant ([MS96]). Im folgenden sollen deshalb die Eigenschaften nicht idealer Batterien, sowie deren Auswirkungen auf die Wahl der optimalen Geschwindigkeit kurz dargestellt werden (wobei die Angaben im wesentlichen auf den in [Mar99] aufgeführten Ergebnissen beruhen).

Bei nicht idealen Batterien wird die Batteriekapazität folgendermaßen durch den Entladestrom beeinflusst:

$$C = \frac{K}{I^\alpha}$$

Dabei ist K eine durch Bauweise und chemische Zusammensetzung der Batterie bestimmte Konstante. α ist ein (ebenfalls batterieabhängiger) Parameter, der angibt wie stark die Kapazität durch die Höhe des Entladestroms beeinflusst wird. Für ideale Batterien wäre $\alpha = 0$, tatsächlich variiert α zwischen $0,2$ und $0,7$.

In der Abbildung 2.6 ist die Kapazität in Abhängigkeit des Entladestroms für verschiedene Werte von α angegeben. Der Entladestrom wurde als Vielfaches der Konstante K angegeben, und die Kapazität relativ zu der bei einem Entladestrom von $1K$.

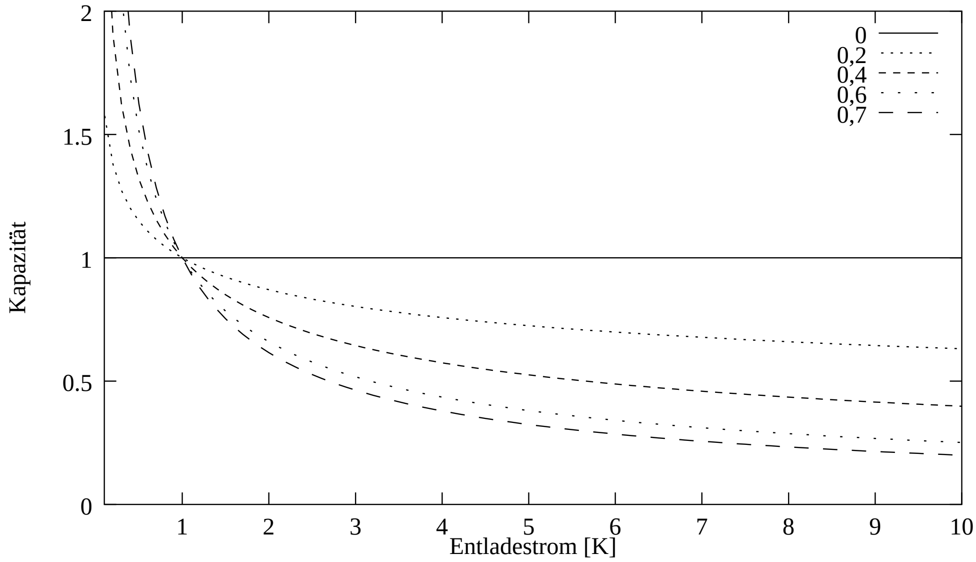

Abbildung 2.6.: Abhängigkeit der Batteriekapazität vom Entladestrom für verschiedene α.

Wie man sieht kann die Verminderung des Stromflusses bzw. des Energieverbrauchs die Kapazität zum Teil beträchtlich erhöhen. Allerdings liegt diesem Modell ein zeitlich konstanter Entladestrom zu Grunde. Diese Voraussetzung wird ein Prozessor bzw. allgemein ein mobiles System jedoch kaum je erfüllen. Vielmehr wird der Energieverbrauch dort, je nachdem ob der Prozessor arbeitet oder nicht, stark schwanken. Somit stellt sich die Frage, welche Auswirkungen ein solch schwankender Verbrauch auf die Kapazität hat.

Simulationen mit einem detaillierten Batteriemodell (eine ausführliche Darstellung der Ergebnisse findet man in [MS99]) haben ergeben, dass die Kapazität weniger vom mittleren Stromverbrauch abhängt, wie man vielleicht intuitiv erwartet hätte. Vielmehr scheint der maximale Stromverbrauch die Batteriekapazität wesentlich stärker zu beeinflussen. Allerdings ist das nur dann der Fall, wenn die Schwankungen einen hinreichend großen zeitlichen Abstand haben. Ändert sich der Stromverbrauch häufiger als etwa einmal pro Sekunde, so hängt die Kapazität wiederum stärker vom mittleren Stromverbrauch ab.

Untersucht man also die Batterielebensdauer eines mobilen Systems, so kann man bei der Abschätzung der Batteriekapazität von einer konstanten Last, und somit von konstantem Stromverbrauch ausgehen. Schließlich beeinflussen auch längere Leerzeiten mit geringem Stromverbrauch die Kapazität nicht so sehr, wie die Stromverbrauchsspitzen. Kurze Schwankungen mit einer Änderungsfrequenz über 1 Hz kann man dabei herausmitteln.

Die so gewonnene Schätzung für die Batteriekapazität wird zwar etwas schlechter ausfallen, als die tatsächliche, als Worst-Case Abschätzung mag sie jedoch genügen.

Rechenoperationen pro Batterieentladung und Work-Ratio

Nachdem die Batteriekapazität stark von der Verlustleistung abhängt, genügt es offensichtlich nicht mehr, allein die MIPJ-Rate als Maß für die Verlängerung der Batterielebensdauer bei verschiedenen Geschwindigkeiten zu betrachten. Statt dessen erscheint es zweckmäßig die Zahl der möglichen Befehle pro Batterieentladung („Computations Per Discharge", CPD) zu betrachten. Diese ergeben sich aus der Rechenleistung R und der Batterielebensdauer T: $CPD = R \cdot T$.

Die Batterielebensdauer lässt sich bei konstantem Stromfluss I und konstanter Spannung U, unter Berücksichtigung der Formel für die Batteriekapazität ($C = K/I^{\alpha}$) errechnen aus

$$T = \frac{C}{I \cdot U} = \frac{K}{I^{\alpha} \cdot I \cdot U} = \frac{K}{I^{1+\alpha} \cdot U}$$

Somit gilt $CPD = R \cdot K/(I^{1+\alpha} \cdot U)$.

Zur Berechnung der CPD ist es also nötig sowohl Energieverbrauch und Rechenleistung als auch die Batterieparameter K und α zu kennen. Martin und Siewiorek schlagen deshalb in [MS96] als Maß für die Effizienzsteigerung das Verhältnis zwischen den CPD bei n-facher Geschwindigkeit (W_n) gegenüber der Anzahl bei einfacher Geschwindigkeit (W_1) vor, die so genannte „Work Ratio" W_n/W_1. Bei einer Work Ratio größer eins ist es sinnvoll die Geschwindigkeit zu erhöhen, anderenfalls nicht.

Wenn man für den Stromverbrauch einen taktfrequenzabhängigen Anteil $f \cdot I_d$ (Taktfrequenz f) sowie einen unabhängigen Anteil I_c annimmt, also $I = I_c + f \cdot I_d$, so ergibt sich, unter Berücksichtigung der obigen Formel für die Batterielebensdauer bei konstantem Stromverbrauch, für die Batterielebensdauer T:

$$T = \frac{K}{(I_c + f \cdot I_d)^{1+\alpha} \cdot U} = \frac{K'}{(I_c + f \cdot I_d)^{1+\alpha}}$$

wobei $K' = K/U$ eine Konstante ist (vorausgesetzt U ist konstant). Die geleistete Rechenarbeit W beträgt $W = R \cdot T$ (R ist die Rechenleistung). Bei einem linearen Zusammenhang zwischen Rechenleistung und Taktfrequenz, also $R = f \cdot R_d$, ergibt sich die geleistete Arbeit W_n bei einer Erhöhung der Taktfrequenz von f_1 auf $n \cdot f_1$:

$$W_n = nf_1 R_d \cdot T = \frac{nf_1 R_d \cdot K'}{(I_c + nf_1 I_d)^{1+\alpha}}$$

Für die Work Ratio ergibt sich folgender Zusammenhang:

$$\frac{W_n}{W_1} = \frac{nf_1 R_d K'/(I_c + nf_1 I_d)^{1+\alpha}}{f_1 R_d K'/(I_c + f_1 I_d)^{1+\alpha}} = n \cdot \left(\frac{I_c + f_1 I_d}{I_c + nf_1 I_d}\right)^{1+\alpha}$$

Setzt man $\rho = I_c/(I_c + fI_d)$, also der Anteil des taktfrequenzunabhängigen Stroms am Gesamtstrom, so ergibt sich (siehe [MS96])

$$\frac{W_n}{W_1} = n \cdot \left(\frac{I_c/\rho}{(I_c/\rho) + (n-1)(I_c/\rho)(1-\rho)}\right)^{1+\alpha} = n \cdot \left(\frac{1}{\rho + n(1-\rho)}\right)^{1+\alpha}$$

Somit hängt die Work Ratio bei unterschiedlicher Geschwindigkeit vom Batterieparameter α und von den Verhältnissen zwischen konstantem und gesamtem Stromverbrauch sowie zwischen alter und neuer Taktfrequenz ab. Es ist nicht mehr nötig die Rechenleistung genau anzugeben, auch K wird nicht mehr benötigt.

Durch ableiten nach n und Null setzen ergibt sich für die optimale Geschwindigkeit $n_{opt} = \rho/(\alpha(1-\rho))$. Als Beispiel soll nun wiederum der AMD Èlan SC410 betrachtet werden, wobei $\alpha = 0.5$ angenommen wird. Der konstante Stromverbrauch des Gesamtsystems beträgt $I_c = 611$mA, der dynamische $I_d = 5$mA. Für 66 MHz ergibt sich $\rho = 0.565$. Die theoretisch optimale Taktfrequenz läge bei 171 MHz, also jenseits der maximalen Geschwindigkeit.Eine Taktfrequenzreduzierung ist deshalb nicht sinnvoll.

Zwei Punkte sind noch anzumerken. Zunächst einmal geht nur das Verhältnis zwischen den beiden *Taktfrequenzen* ein, nicht zwischen den *Rechenleistungen* – die Rechenleistung muss aber, wie bereits erläutert, nicht proportional zur Taktfrequenz sein. Dieses Problem lässt sich beseitigen, wenn das n außerhalb der Klammer durch das Leistungsverhältnis $\tilde{n}$ der Rechenleistung bei Taktfrequenz nf_1 gegenüber Taktfrequenz f_1 ersetzt wird. Damit ergibt sich

$$\frac{W_n}{W_1} = \tilde{n} \cdot \left(\frac{1}{\rho + n(1-\rho)}\right)^{1+\alpha}$$

Außerdem sollte man bei Verwendung der Work Ratio stets im Gedächtnis behalten, dass nur die bei einer Vollauslastung theoretisch möglichen Rechenoperationen pro Batterieentladung betrachtet werden. Steht das System zeitweise leer, so ändern sich die Werte, wie in Abschnitt 2.2.1 erläutert.

2.2.4. Spannungssenkung

Eine viel versprechende Möglichkeit, um durch Taktfrequenzreduzierung Energie zu sparen besteht darin, die Spannung zusammen mit der Taktrate zu senken. Der AMD Èlan SC410 kann mit zwei unterschiedlichen Spannungen betrieben werden (wobei die Spannung nicht dynamisch angepasst werden kann). In der Tabelle 2.2 ist die Verlustleistung des Prozessors ([AMDa]) bei verschiedenen Versorgungsspannungen und Geschwindigkeiten angegeben.

Wie man sieht, ist eine deutliche Senkung der Verlustleistung möglich, ohne die Verarbeitungsgeschwindigkeit zu senken. Allerdings hängt die maximale Geschwindigkeit eines Prozessors von der Versorgungsspannung ab, so dass einer Senkung der Spannung eine Senkung der Geschwindigkeit vorausgehen muss.

Geschwindigkeit	100 MHz	66 MHz	33 MHz	4 MHz
Verlustleistung bei 3,3 V	1818 mW	1222 mW	703 mW	192 mW
Verlustleistung bei 2,7 V	–	753 mW	469 mW	115 mW

Tabelle 2.2.: Verlustleistung des AMD Èlan SC410 bei unterschiedlichen Geschwindigkeiten und Spannungen

Während es zwar bereits eine Reihe von Prozessoren gibt, die mit zwei oder drei unterschiedlichen Spannungen betrieben werden können, wurde bislang die Möglichkeit einer dynamischen Anpassung der Spannung an die Taktfrequenz lediglich in Forschungsprojekten realisiert ([PB98]). Somit kann man die tatsächlich möglichen Einsparungen nur schätzen, an Hand der für verschiedene statische Spannungen angegebenen Daten.

Allgemein lässt sich die Verlustleistung einer integrierten CMOS Schaltung errechnen aus $P = fCU^2$, wobei f die Taktfrequenz, U die Versorgungsspannung und C eine Konstante bezeichnen. Somit kann man davon ausgehen, dass der Energieverbrauch linear in der Taktfrequenz und quadratisch in der Spannung ist.

Wie bereist erwähnt bedingt eine Spannungssenkung auch eine Taktfrequenzverminderung. Hier kann man von einem linearen Zusammenhang zwischen Spannung und Taktfrequenz ausgehen (vgl. [WWDS94]), also $U = f \cdot C_u$. Beispielsweise benötigt der Motorola 6805 Mikrocontroller 5 V bei 6 MHz, 3,3 V bei 4,5 MHz und 2,2 V bei 3 MHz.

Somit ergibt sich die Verlustleistung in Abhängigkeit von der Spannung aus $P = fC(C_u f)^2 = f^3 C'$. Geht man in erster Näherung von einem linearen Zusammenhang zwischen der Rechenleistung R und der Taktfrequenz aus, d. h. $R = f \cdot C_R$, so gilt für das Verhältnis zwischen Energieverbrauch und Rechenleistung (MIPJ-Rate):

$$\frac{R}{P} = \frac{f \cdot C_R}{f^3 \cdot C'} = \frac{\hat{C}}{f^2}$$

wobei $\hat{C} = C_R / C'$ eine Konstante ist. Es sind also Einsparungen möglich, die sich quadratisch zur Taktfrequenz verhalten, d. h. R/P wird beispielsweise vervierfacht, wenn f halbiert wird.

Dieses Ergebnis ist jedoch in mehrfacher Hinsicht zu optimistisch. Schließlich enthält auch der Prozessor selbst Anteile, die von der Taktfrequenz weitgehend unabhängig sind, so z. B. die Echtzeit-Uhr oder verschiedene Intervallzähler mit fester Frequenz (vgl. 4.2.2). Außerdem ist auch das Senken der Spannung nicht für alle Teile des Prozessors möglich. Vielmehr sind häufig zwei verschiedene Versorgungsspannungen nötig, so z. B. für den SA1100 ([Int99]) der mit einer Versorgungsspannung von 1,5 V für den Prozessorkern und 3,3 V für Peripheriekontakte arbeitet. Während die Spannung des Prozessorkerns gesenkt werden kann, wird das für die Spannung, mit der der Prozessor mit dem restlichen System kommuniziert nicht der Fall sein.

Dadurch ergibt sich für die Verlustleistung ein konstanter Anteil, ein von f abhängiger (also Taktfrequenz variabel, Spannung konstant), ein von f^2 abhängiger (Geschwindigkeit konstant, Spannung variabel) und schließlich ein durch f^3 bestimmter Teil (Spannung und Taktfrequenz variabel). Insgesamt ergibt sich also $P = a_0 + a_1 f + a_2 f^2 + a_3 f^3$ ([Mar99]).

Zusätzlich tritt ein weiteres Problem auf: Da die maximale Geschwindigkeit von der Spannung abhängt, kann die Taktfrequenz zwar ohne weiteres verringert werden, um sie zu erhöhen muss jedoch zunächst die Spannung auf den richtigen Pegel gebracht werden. Dabei muss die Zeit beachtet werden die für einen Spannungswechsel benötigt wird. Man kann davon ausgehen, dass die Wartezeit im Bereich einiger zehn Mikrosekunden liegen wird ([WWDS94]). Somit können Geschwindigkeit und Spannung zwar gleichzeitig gesenkt werden, für eine Erhöhung ist jedoch eine kurze Pause nötig.

2.2.5. „Optimale" Geschwindigkeit

Da die – im Sinne der Verlustleistung – optimale Geschwindigkeit stark von der verwendeten Hardware (Prozessor und andere Systemkomponenten, sowie Batterien) aber auch von der verwendeten Software abhängt, ist es natürlich nicht möglich eine allgemein gültige optimale Taktfrequenz zu finden. Es lassen sich lediglich einige allgemeine Schlüsse ziehen.

Zunächst einmal sind die Einsparungen bei einer alleinigen Taktfrequenzsenkung so gering, dass es sich im Allgemeinen nicht lohnen wird, die Geschwindigkeit zu drosseln ohne die Spannung ebenfalls zu senken. Dann allerdings sind durchaus Einsparungen möglich.

Andererseits wirkt der so erreichten Verringerung der MIPJ-Rate der konstante Verbrauch des restlichen Systems entgegen, der durch die Verlängerung der Rechendauer an Einfluss gewinnt. Deshalb ist es wichtig, den Energieverbrauch der verschiedenen Komponenten nicht isoliert zu betrachten, sondern die verschiedenen Maßnahmen zu koordinieren. Dadurch kann der konstante Anteil im Energieverbrauch gesenkt werden.

Insgesamt wird sich ein Zusammenhang zwischen der möglichen Anzahl von Rechenoperationen pro Batterieentladung und der Geschwindigkeit ergeben, der dem in der Abbildung 2.7 gezeigten ähnelt: Beginnt man mit der langsamsten Geschwindigkeit, so hat eine Taktfrequenzsteigerung zunächst auch eine CPD-Steigerung zur Folge (im wesentlichen bedingt durch den taktfrequenzunabhängigen Verbrauch des Gesamtsystems). Bei einer gewissen „optimalen" Geschwindigkeit f_{opt} wird ein Maximum der CPD erreicht. Eine darüber hinausgehende Taktfrequenzerhöhung hat eine Verringerung der CPD zur Folge (bedingt durch den hohen taktfrequenzabhängigen Verbrauch des Prozessors).

Aus Sicht des Energieverbrauchs sollte der Prozessor also mit dieser Taktfrequenz f_{opt} betrieben werden.

Neben dem optimalen Energieverbrauch müssen natürlich auch die Anforderungen der Anwendung betrachtet werden. Hier muss ein Kompromiss zwischen hoher Rechenleistung einerseits und niedrigem Energieverbrauch andererseits gefunden werden. Dieses Problem ist Gegenstand des Abschnitts 2.4.

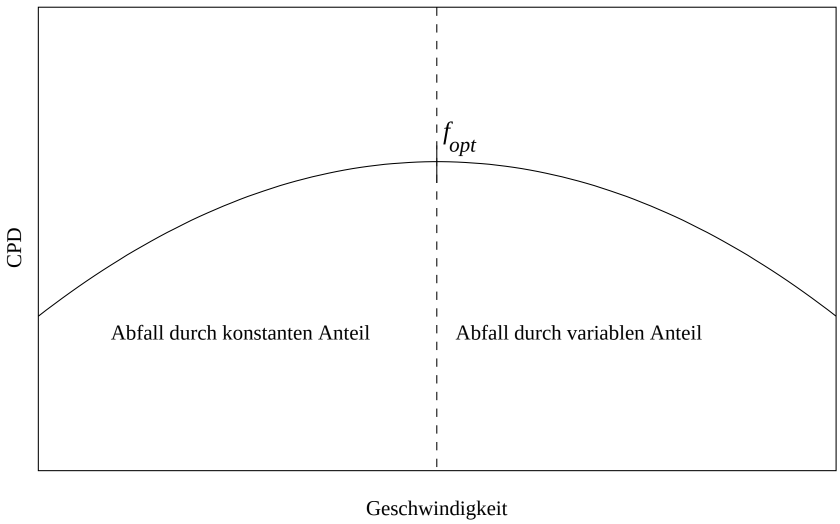

Abbildung 2.7.: Zusammenhang zwischen CPD und Geschwindigkeit

2.3. Energieverbrauch des AMD Èlan SC410

In diesem Abschnitt soll ein Überblick über den Stromverbrauch des im praktischen Teil der Arbeit verwendeten AMD Èlan SC410 Prozessors gegeben werden. Wie bereits erwähnt, bietet dieser die Möglichkeit, die Taktfrequenz in einem Bereich von 1 bis 66 MHz stufenweise anzupassen (siehe 4.1.1). Außerdem kann der Élan mit zwei unterschiedlichen Versorgungsspannungen betrieben werden (wobei die Spannung nicht dynamisch verändert werden kann).

In Abb. 2.8 ist die Verlustleistung in Abhängigkeit von der Taktfrequenz für verschiedene Befehle angegeben. Zur Messung wurde eine Schleife ausgeführt, innerhalb der die entsprechende Instruktion 1000 oder 2000 mal (je nach Befehlslänge) ausgeführt werden, wobei als Operanden stets die 32-bit Register des Prozessors verwendet werden. Die gesamte Schleife kann in den Cache geladen werden, sodass die Anzahl der Hauptspeicherzugriffe möglichst gering wird. Weitere Einzelheiten zur Durchführung der Messungen sind in Anhang A angegeben.

Die folgenden Befehle sind in der Abbildung angegeben: Ganzzahladdition („add"), „move" Befehl (kopiert den Inhalt eines Register in ein anderes Register), logisches Und („and"), Negation („not"), Vertauschung der Bit-Anordnung („bswap") und unbedingter Sprung („jmp").

Wie bereits erwähnt, besteht ein linearer Zusammenhang zwischen Taktfrequenz und Energieverbrauch. Der Energieverbrauch $P(f)$ des Prozessors lässt sich somit durch eine Ausgleichsgerade $P(f) = P_c + f \cdot P_d$ annähern. In Tabelle 2.3 ist die Steigung P_d für einige Befehle angegeben, außerdem die Standardabweichung σ. Wie man sieht beträgt die Abweichung lediglich einige Milliwatt, so dass der Energieverbrauch sehr gut linear

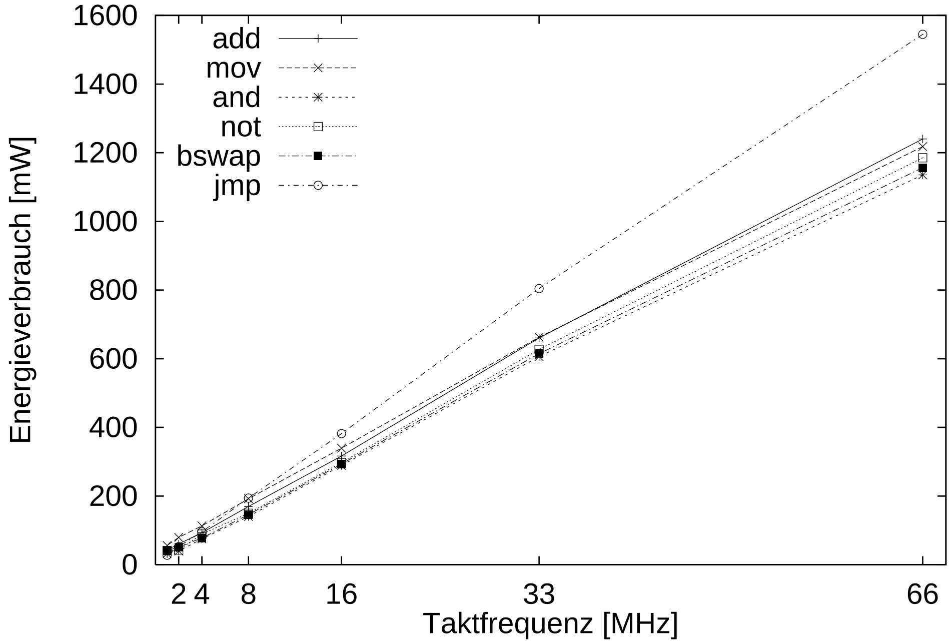

Abbildung 2.8.: Leistungsaufnahme des AMD Èlan SC410 bei verschiedenen Taktfrequenzen.

angenähert werden kann. Allerdings schwankt die Steigung bei unterschiedlichen Befehlen, wodurch sich bei 66MHz Taktfrequenz Unterschiede von einigen 100mW ergeben.

Name	Steigung	σ
add	18,51 mW/MHz	11,41 mW
mov	17,87 mW/MHz	11,12 mW
not	17,79 mW/MHz	12,04 mW
and	17,16 mW/MHz	12,74 mW
bswap	17,36 mW/MHz	12,44 mW
jmp	23,38 mW/MHz	11,10 mW

Tabelle 2.3.: Steigung des Energieverbrauchs und Standardabweichung σ zwischen Messdaten und Ausgleichsgerade.

Insbesondere hat der unbedingte Sprungbefehl („jmp”) eine wesentlich höhere Leistungsaufnahme, als die arithmetisch-logischen Befehle. Es ist anzunehmen, dass dieser hohe Verbrauch damit zusammenhängt, dass die Pipelines des Prozessors bei jedem Sprungbefehl geleert werden müssen.

Die Verlustleistung hängt somit sowohl von der Taktfrequenz, als auch vom ausgeführten Befehl ab. Allerdings ist auch die Verlustleistung bei gleichem Befehl und gleichbleibender

Geschwindigkeit nicht konstant, sondern hängt vom Inhalt der Operandenregister ab. In Tabelle 2.4 ist die Verlustleistung für „bswap" und „mov" Instruktion und unterschiedliche Registerwerte angegeben. Beim mov Befehl haben Quell- und Zielregister den gleichen Wert, bei bswap wird der Registerinhalt am Anfang auf den angegebenen Wert gesetzt und danach stets mit dem Resultat der letzten Ausführung ausgeführt (die Reihenfolge der Bits wird also jedesmal umgekehrt).

Registerinhalt	mov	bswap
`0xffffffff`	1057 mW	1101 mW
`0x0fffffff`	1072 mW	1121 mW
`0x00ffffff`	1083 mW	1135 mW
`0x000fffff`	1098 mW	1142 mW
`0x0000ffff`	1112 mW	1159 mW
`0x00000fff`	1133 mW	1168 mW
`0x000000ff`	1149 mW	1180 mW
`0x0000000f`	1166 mW	1192 mW
`0x00000000`	1183 mW	1203 mW
`0xf0f0f0f0`	1122 mW	1155 mW
`0x0f0f0f0f`	1120 mW	1153 mW

Tabelle 2.4.: Verlustleistung in Abhängigkeit des Registerinhaltes

Wie man sieht, hängt der Energieverbrauch im wesentlichen davon ab, wie viele Bits des Registers gesetzt sind und wie viele nicht. So unterscheidet sich die Leistungsaufnahme bei `0x0000ffff`, `0xf0f0f0f0` und `0x0f0f0f0f` um maximal 10 mW bei move und 6 6 mW bei bswap, also unter unter 0,9 bzw. 0,6%. Der maximale Unterschied bei unterschiedlicher Anzahl gesetzter Bits beträgt hingegen 126 bzw. 102 mW, also immerhin 10,7 und 8,5%.

Allerdings scheint auch die Position der gesetzten oder gelöschten Bits eine Rolle zu spielen: In Tabelle 2.5 ist wiederum die Verlustleistung für den move Befehl eingetragen, wobei die Anzahl der gesetzten Bits stets gleich bleibt, lediglich die Position wird verändert. Offensichtlich verursacht das setzen eines der letzten drei Halb-Bytes einen höheren Stromverbrauch, als es bei anderen Positionen der Fall ist.

Neben dem Registerinhalt spielt auch die Art der Operanden eine Rolle: Bisher waren Quell- und Zieloperand Mehrzweckregister des Prozessors. Für die meisten Befehle des AMD Èlan SC410 existieren jedoch Varianten, die auf direkt angegebene Werte („immediate"), Register oder Werte im Speicher angewendet werden können. In Tabelle 2.6 ist die Leistungsaufnahme bei Ausführung der verschiedenen Varianten des move Befehls aufgeführt (bei konstanter Taktfrequenz von 66 MHz).

Die Werte für den Energieverbrauch unterscheiden sich zum Teil beträchtlich, was hauptsächlich auf die Zahl der Speicherzugriffe zurückzuführen sein dürfte. Der move Befehl für zwei Register ist zwei Byte lang (vgl. [Int97]), somit müssen nur zwei Bytes aus dem Speicher gelesen werden. Die Version „Immediate"-Register ist dagegen insgesamt fünf Byte

Registerinhalt	Verlustleistung
`0x0000000f`	1172 mW
`0x000000f0`	1178 mW
`0x00000f00`	1179 mW
`0x0000f000`	1179 mW
`0x000f0000`	1186 mW
`0x00f00000`	1184 mW
`0x0f000000`	1184 mW
`0xf0000000`	1185 mW

Tabelle 2.5.: Verlustleistung in Abhängigkeit des Inhalts des Quell-Registers beim „mov" Befehl

Quelle	Ziel	P
Register	Register	1218 mW
Register	Speicher	1555 mW
Immediate	Register	1249 mW
Immediate	Speicher	1525 mW
Speicher	Register	1483 mW

Tabelle 2.6.: Leistungsaufnahme P in Abhängigkeit von Quell- und Zieloperand des move Befehls bei 66 MHz Taktfrequenz.

lang, da nicht nur der Befehl selbst, sondern auch der Quell-Operand aus dem Speicher gelesen werden muss. Der Anstieg der Verlustleistung ist auf die mit den zusätzlichen Speicherzugriffen verbundenen Energiekosten zurückzuführen.

Der hohe Energieverbrauch bei Schreibzugriffen auf den Speicher dürfte daher kommen, dass alle anderen Speicherzugriffe durch den Cache bedient werden können (wie bereits erwähnt sind die Benchmarks klein genug, um vollständig im Cache gehalten zu werden), wohingegen die Schreibzugriffe direkt an den Speicher gehen, da es sich um einen „write-through" Cache handelt (vgl. [AMDb, AMD96]).

Bei der Speicher-Register Variante des Befehls hingegen können die nötigen Speicherzugriffe wiederum durch den Cache bedient werden.

In Abbildung 2.9 ist der Energieverbrauch der move Instruktionen nochmals für verschiedene Geschwindigkeiten dargestellt. Dabei steht „r" für Register, „i" für Immediate und „m" für Hauptspeicheroperanden. Zusätzlich ist der Energiebedarf für den mov Befehl aufgetragen („mov m m"), wenn sowohl Quelle als auch Ziel ungepufferte Speicherzugriffe erfordern und außerdem die ausgeführte Schleife so groß ist, dass sie nicht vollständig in den Cache passt, d. h. dass jeder Befehl aus dem Speicher geholt werden muss. Wie man sieht, steigt der Energieverbrauch mit der Anzahl der Speicherzugriffe rapide an.

Insgesamt hängt die Verlustleistung des Prozessors bei konstanter Geschwindigkeit also in gewissem Umfang von der Art des Befehls und dem Inhalt der Operandenregister ab

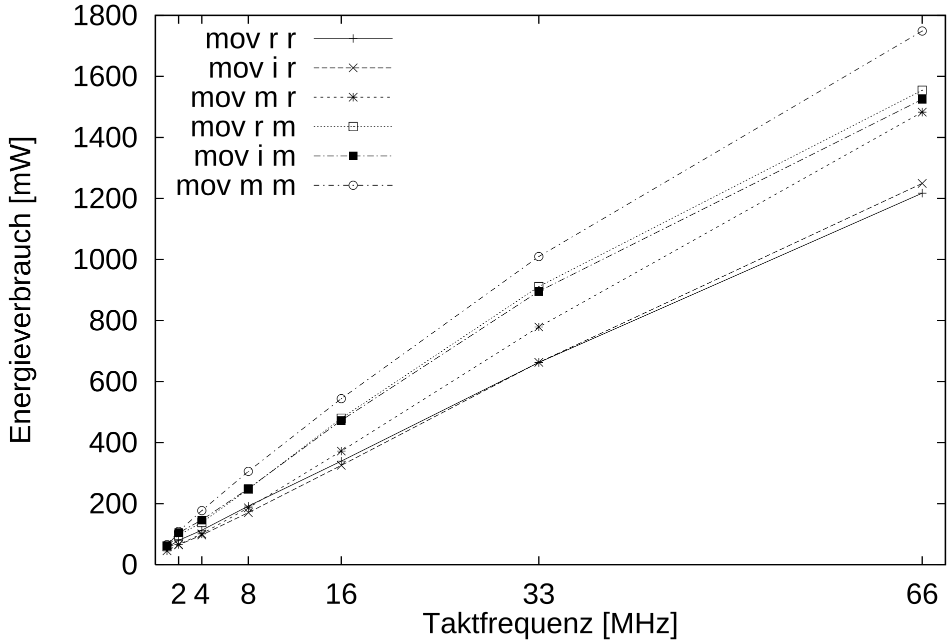

Abbildung 2.9.: Verlustleistung der verschiedenen Varianten des move Befehls.

(Unterschiede von unter 150mW), wesentlich mehr wird die Verlustleistung jedoch durch Speicherzugriffe bestimmt (bis zu 600mW Unterschied).

Kombination verschiedener Befehle

Bislang wurde nur die Verlustleistung einzelner Befehle betrachtet. Nun soll kurz dargestellt werden, wie sich der Energieverbrauch verhält, wenn verschiedene Befehle gemischt werden. Als Beispiel soll eine Schleife untersucht werden, bei der abwechselnd der „imul" (Ganzzahlmultiplikation) und der „not" Befehl auf unterschiedlichen Registern ausgeführt werden.

Bei einer Geschwindigkeit von 66 MHz beträgt die Verlustleistung bei Ausführung des „not" Befehls 1185 mW, bei „imul" sind es nur 1077 mW, also ein Unterschied von über 100 mW. Werden beide Befehle abwechselnd ausgeführt, so ergibt sich eine Leistungsaufnahme von 1091 mW.

Somit scheint die „imul" Instruktion die Leistungsaufnahme wesentlich stärker zu beeinflussen, als die Negation. Da eine Multiplikation jedoch durchschnittlich etwa 26 Taktzyklen, eine einfache Negation hingegen nur 3 Taktzyklen benötigt, war dies nicht anders zu erwarten.

Deshalb bietet sich die Annahme an, dass sich die Leistungsaufnahme bei Mischung beider Befehle aus dem mit den CPI gewichteten Mittel der einzelnen Verlustleistungen ergibt, also $P = (CPI_{imul} \cdot P_{imul} + CPI_{not} \cdot P_{not})/(CPI_{imul} + CPI_{not})$. Damit ergibt sich $P = (26 \cdot 1077\text{mW} + 3 \cdot 1185\text{mW})/(26 + 3) \approx 1088\text{mW}$, also eine Abweichung vom lediglich 3 mW gegenüber dem gemessenen Wert.

Misst man die Ausführungszeiten wenn beide Befehle abwechselnd ausgeführt werden, so ergibt sich folgendes: Die mittlere Anzahl von Takten pro Befehl für beide Befehle zusammen resultiert nicht aus dem Mittelwert (14,5 CPI) sondern liegt weit darüber, bei 23 CPI. Das bedeutet, dass die Ausführungszeit wesentlich mehr von der Multiplikation als von der Negation abhängt. Aus der Lösung der Gleichung $CPI_{imul} + a \cdot CPI_{not}/(1 + a) = CPI_{gesamt}$ ergibt sich $CPI_{gesamt} = (CPI_{imul} + 0,15 CPI_{not})/1,15$, d. h. der „not" Befehl hat einen Anteil von nur 13% an CPI_{gesamt}. Überträgt man dieses Verhältnis auch auf die Verlustleistung, so ergibt sich $P = (1077\text{mW} + 0,15 \cdot 1185\text{mW})/1,15 = 1091\text{mW}$. Dieser Wert stimmt genau mit dem gemessenen Wert überein.

Somit scheint es durchaus möglich die Verlustleistung des Prozesses relativ genau zu bestimmen, wenn man die Verlustleistung der einzelnen Befehle, deren CPI und natürlich die Anzahl der Befehle im Programm kennt. Allerdings spielt – wie oben erläutert – auch die Zahl der Speicherzugriffe eine erhebliche Rolle. Um die Leistungsaufnahme zu bestimmen muss somit auch diese Zahl bekannt sein.

2.3.1. Spannungsänderungen

Der AMD Èlan SC410 kann entweder mit 3,3 V oder mit 2,7 V Versorgungsspannung betrieben werden, wobei die Spannung jedoch nicht dynamisch gesenkt werden kann. Da eine dynamische Spannungssenkung jedoch eine sehr vielversprechende Möglichkeit der Energieeinsparung wäre, soll nun eine Abschätzung der bei dynamischer Spannungsänderung theoretisch möglichen Einsparungen gegeben werden.

Leider sind die im Datenblatt aufgeführten Werte für die Leistungsaufnahme bei verschiedenen Spannungen und Geschwindigkeiten nur sehr unzureichend. Zudem beziehen sich die Angaben auf den Élan SC400, der zusätzlich einen eingebauten LCD-Controller hat, der den Energieverbrauch ebenfalls beeinflusst. Deshalb werden im weiteren nur die bei konstanter Versorgungsspannung von 3,3 V gemessenen Werte verwendet.

Wie im Abschnitt 2.2.4 erläutert, wird die Versorgungsspannung in der Regel quadratisch in die Verlustleistung eingehen. Da alle Komponenten des AMD Èlan SC410 mit einer einheitlichen Spannung versorgt werden, ergibt sich folgender Zusammenhang zwischen Versorgungsspannung U und Verlustleistung P:

$$P = C_1 U^2 + C_2 f U^2$$

wobei C_1 und C_2 Konstanten sind. Wie schon bei konstanter Spannung, wird zu erwarten sein, dass beide Konstanten vom gerade ausgeführten Befehl, insbesondere von der Anzahl der Speicherzugriffe abhängen.

In Tabelle 2.7 sind Werte für C_2 bei verschiedenen Befehlen angegeben. Die Werte wurden aus den Parametern der Ausgleichsgeraden für Taktfrequenzänderung und konstante

Name	C_2
add	1,70
mov	1,64
not	1,63
and	1,58
bswap	1,59
jmp	2,15

Tabelle 2.7.: Steigung des Energieverbrauchs bei variabler Spannung (C_2).

Spannung (Tabelle 2.3) ermittelt, indem $C_2U^2 = C_2 \cdot 3,3^2 = P_d$ gesetzt wurde. C_1 ergibt sich aus dem im Datenblatt angegebenen Verbrauch im Standby Betrieb ($P_C = 63$mW), wobei $C_1U^2 = C_1 \cdot 3,3^2 = P_C$ gesetzt wird. Damit ergibt sich $C_1 = 5,79$.

Wie bereits erwähnt, kann der AMD Èlan SC410 entweder mit 3,3 V oder mit 2,7 V Versorgungsspannung betrieben werden. Bei 3,3 V beträgt die maximale Taktfrequenz 100 MHz, bei 2,7 V 66 MHz. Geht man davon aus, dass die Spannung linear mit der Taktfrequenz abgesenkt werden kann, so ergeben sich für die verschiedenen Geschwindigkeitsstufen die in Tabelle 2.8 angegebenen Werte für die Versorgungsspannung.

Taktfrequenz	Spannung
66 MHz	2,7 V
33 MHz	2,25 V
16 MHz	2,03 V
8 MHz	1,92 V
4 MHz	1,87 V
2 MHz	1,84 V
1 MHz	1,825 V

Tabelle 2.8.: Spannung in Abhängigkeit der Taktfrequenz.

Die daraus resultierende Verlustleistung ist in Abb. 2.10 für Addition und logisches Und wiedergegeben. Zum Vergleich wurde auch der Verlauf der Leistungsaufnahme bei konstanter Spannung eingetragen.

2.3.2. Work-Ratio

In Tabelle 2.9 ist die aus der gemessenen Rechen- und elektrischen Verlustleistung resultierende Work-Ratio am Beispiel des Additionsbefehls wiedergegeben.

Dabei steht „U konst." für konstante Spannung, „U var." für variable Spannung, unter Verwendung des oben angegebenen Zusammenhangs zwischen Spannung und Leistungsaufnahme. Außerdem wurde die Work-Ratio sowohl unter Berücksichtigung des Energieverbrauchs des Gesamtsystems (mit einer konstanten Verlustleistung von 2016 mW) als

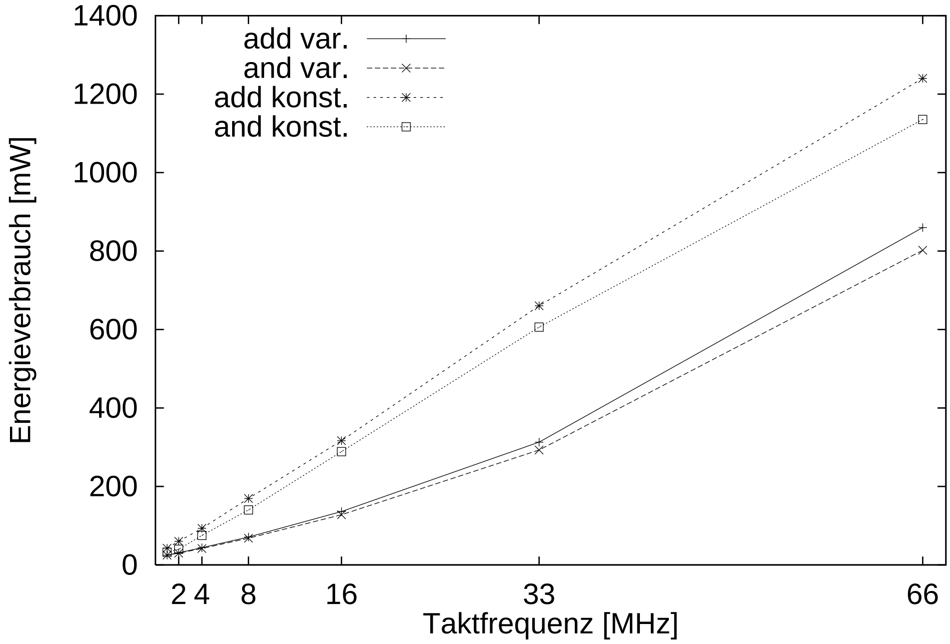

Abbildung 2.10.: Verlustleistung bei variabler und konstanter Spannung.

Geschwindigkeit	U konst.		U var.	
	Prozessor	Gesamtsystem	Prozessor	Gesamtsystem
66 MHz	1,04	82,04	1,12	60,91
33 MHz	1,33	54,54	2,30	49,90
16 MHz	1,99	33,43	3,84	32,16
8 MHz	2,43	17,58	4,72	17,35
4 MHz	2,75	8,55	4,27	8,47
2 MHz	2,05	3,39	2,82	3,38
1 MHz	1	1	1	1

Tabelle 2.9.: Work Ratio für Ganzzahladdition bei konstanter und variabler Spannung.

auch mit dem wesentlich geringeren konstantem Verbrauch des Prozessors (ca. 63 mW) angegeben. Als Batterieparameter wurde $\alpha = 0,5$ gewählt.

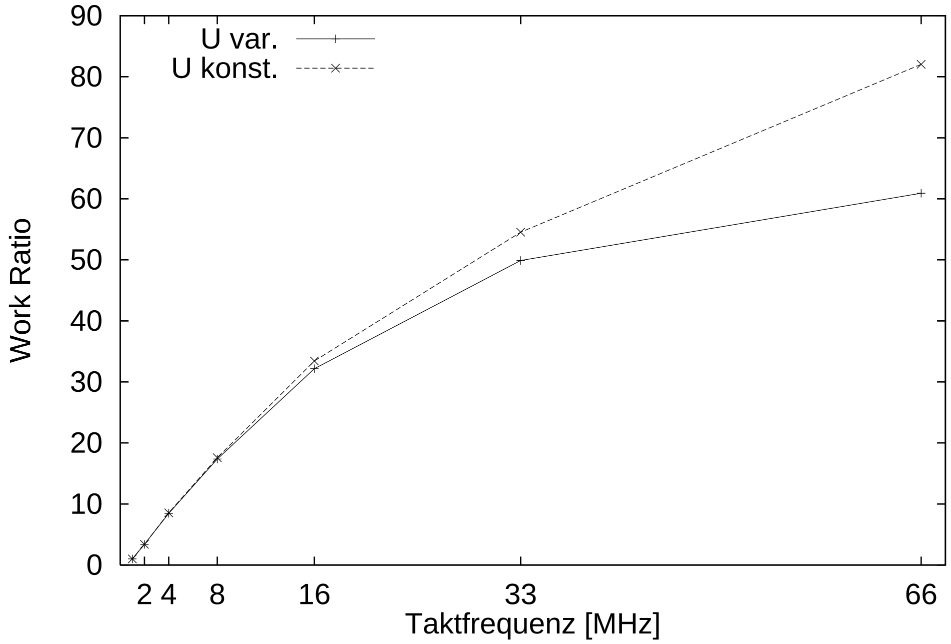

Abbildung 2.11.: Work Ratio bei Betrachtung des Stromverbrauchs des Gesamtsystems.

Die Work Ratio bei Berücksichtigung des Gesamtsystems ist in Abb. 2.11 angegeben, Abb. 2.12 zeigt das Verhalten wenn nur der Prozessor betrachtet wird. Wie – wegen der hohen Verlustleistung des restlichen Systems – zu erwarten, ergibt sich die maximale Work Ratio im ersteren Fall bei maximaler Geschwindigkeit. Die optimale Geschwindigkeit ist also höher als die maximale Taktfrequenz. Nur wenn der Verbrauch des restlichen Systems außer Acht gelassen wird, reichen die Einflüsse von Batteriekapazität und Speicherzugriffsgeschwindigkeit aus, um eine Taktfrequenzsenkung zu rechtfertigen.

2.4. Mögliche Strategien zur Energieverwaltung

Ist die theoretisch optimale Taktfrequenz nicht größer oder gleich der maximal möglichen, so hat eine Erhöhung der CPD im Allgemeinen eine Senkung der Taktfrequenz auf einen Wert zwischen der maximalen und der optimalen Geschwindigkeit zur Folge.

Bei der Wahl der Taktfrequenz muss also stets ein Kompromiss getroffen werden, zwischen einer hohen Dienstgüte (verbunden mit geringer Batterielebensdauer) einerseits, und einer effizienten Energienutzung (verbunden mit geringer Geschwindigkeit) andererseits. Bei der Suche nach einem solchen Kompromiss sind unterschiedliche Ansätze denkbar.

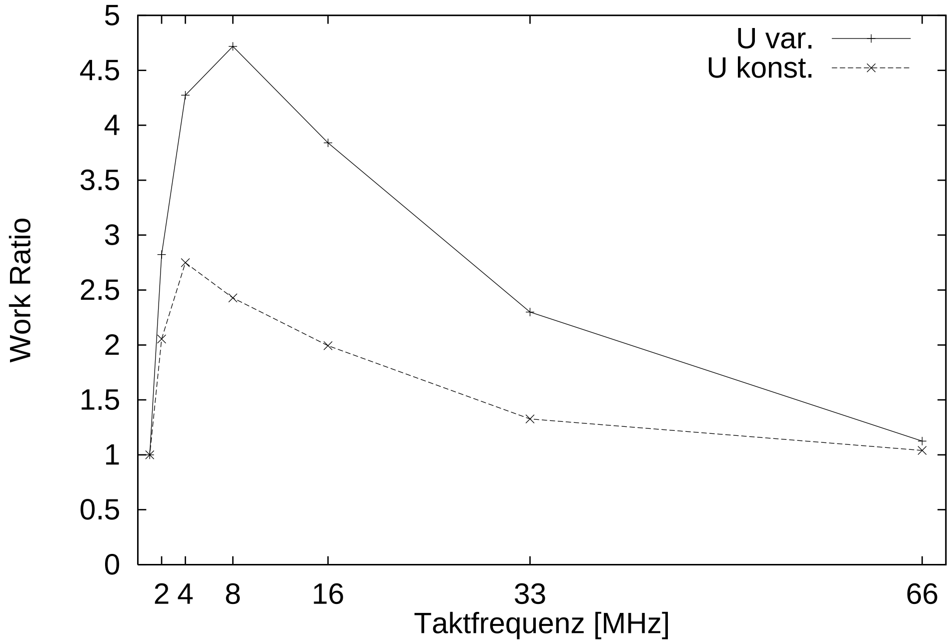

Abbildung 2.12.: Work Ratio bei Betrachtung des Prozessorverbrauchs.

Eine in verschiedenen Arbeiten ([WWDS94, GCW95]) beschriebene Methode bemüht sich, die Auslastung des Prozessors auf möglichst hohem Niveau zu halten. Dem liegt die Überlegung zu Grunde, dass die Rechenleistung so lange ausreichend ist, wie alle Aufträge bearbeitet werden können. D. h. so lange, wie die Last (Ankunfts- durch Bedienrate) kleiner oder gleich 100% ist.

Andererseits bringt eine Senkung der Last auf niedrigere Werte keine Verbesserung der Dienstgüte, sodass die Bedienrate (und damit die Geschwindigkeit) soweit gesenkt werden kann, bis eine Vollauslastung erreicht wird. So lange die Geschwindigkeit über der optimalen liegt, wird dadurch eine Erhöhung der CPD erreicht.

Die in [GCW95] angegebenen Algorithmen sind dementsprechend alle nach dem gleichen Prinzip aufgebaut. Die Taktfrequenz wird jeweils nach einem kurzen Zeitintervall neu bestimmt. Dabei wird zunächst gezählt wie viele Taktzyklen des vergangenen Zeitfensters im idle-loop verbracht wurden. Anschließend wird auf dieser Grundlage die Last für die nächste Periode vorhergesagt und die Geschwindigkeit entsprechend verringert (bei Last kleiner eins) bzw. erhöht (bei Last größer eins). Unterschiedlich ist bei diesen Algorithmen lediglich die Methode zur Lastvorhersage für das nächste Intervall.

Diese Vorgehensweise birgt allerdings eine Reihe von Problemen: Zunächst einmal hängt die Leistungsaufnahme innerhalb eines Intervalls allein von der anfallenden Rechenlast ab. Es ist nicht möglich den Stromverbrauch in irgendeiner Weise zu regulieren, z. B. durch Angabe eines Verbrauchs, der im Mittel eingehalten werden soll. Ein solcher Mechanismus wäre jedoch u. a. dann sinnvoll, wenn die Energiesparmaßnahmen mehrerer

Komponenten des Systems koordiniert werden sollen (vgl. 2.2.1). In diesem Falle wäre es nützlich für jede Komponente einen Energieverbrauch vorgeben zu können, der eingehalten werden soll.

Außerdem ist es bei solchen Strategien nicht möglich, verschiedene Prozesse unterschiedlich schnell auszuführen – die Geschwindigkeit wird schließlich für den gesamten Prozessor bestimmt, unabhängig davon, welche Prozesse ausgeführt werden. Will man jedoch beispielsweise interaktive Prozesse mit sporadischem aber hohem Rechenzeitbedarf bevorzugen, so wäre es wünschenswert diese schnell, langlaufende Hintergrundprozesse dafür entsprechend langsamer ablaufen zu lassen.

Auch für eventuelle Echtzeitanforderungen wäre eine prozessbasierte Geschwindigkeitsfestlegung unumgänglich (siehe [Bel99]). Nur dann können Prozesse mit kurzfristiger Zielzeit schnell genug laufen um diese Zielzeit einzuhalten, wohingegen unkritische Hintergrundprozesse entsprechend verlangsamt werden können.

Ein weiteres Manko der oben geschilderten Vorgehensweise ist, dass die Geschwindigkeit des Prozessors lediglich von der Zahl der idle-loops abhängt. Schließlich können durchaus Prozesse existieren die unkritische Aufgaben erledigen, die etwas verzögert werden können. In diesem Falle würde der Prozessor jedoch stets mit voller Geschwindigkeit laufen.

Auch verbrauchen unterschiedliche Prozesse unterschiedlich viel Strom (vgl. 3.2). Somit hat eine Taktfrequenzdrosselung bei verschiedenen Prozessen unterschiedlich starke Einsparungen zur Folge, und man könnte daran denken sehr verbrauchsintensive Prozesse langsamer ablaufen zu lassen als andere.

Zudem gibt es Fälle, wo die Auslastung von 100% nicht unbedingt den besten Kompromiss darstellt: Es wäre beispielsweise möglich, dass man weniger daran interessiert ist, die beste CPD-Rate zu erreichen, sondern daran, dass das System noch bis zu einem bestimmten Zeitpunkt in der Zukunft arbeitet. Ist die gefordert Lebensdauer länger als die bei 100% Auslastung erreichte, so muss der Prozessor weiter verlangsamt werden, anderenfalls kann er beschleunigt werden.

Deshalb soll in dieser Arbeit ein Ansatz verfolgt werden, bei dem ein von der Anwendung vorgegebenes Energieverbrauchslimit eingehalten wird. Außerdem wird die Geschwindigkeit nicht global, sondern für jeden Prozess einzeln gesetzt, wie im folgenden Kapitel dargestellt.

3. Taktfrequenzgesteuerte Energieverwaltung im Scheduler

Im folgenden soll eine Strategie zur Taktfrequenzsteuerung dargestellt werden, die die Geschwindigkeit des Prozessors nicht periodisch, auf Grund der globalen Leistungsaufnahme, sondern für jeden Prozess einzeln bestimmt. Dabei wird jedem Prozess eine Geschwindigkeit zugeordnet, und in seinem Prozesskontext vermerkt. Kommt der Prozess zur Ausführung wird die CPU-Geschwindigkeit entsprechend gesetzt.

Die Ausführungen in den folgenden Abschnitten beschränken sich auf ein System, bei dem die Prozesse nach Round Robin ([Tan92]) oder einer ähnlichen Strategie mit Zeitscheiben abgearbeitet werden. Außerdem wird angenommen, dass die Taktfrequenz nicht kontinuierlich verändert werden kann, sondern nur stufenweise. Des weiteren bringt die Geschwindigkeitsumschaltung weder besondere Energiekosten noch zeitliche Verzögerungen mit sich.

Letzteres wird so natürlich nicht erfüllt werden können, da das Umschalten der Taktfrequenz – insbesondere bei Spannungsänderungen – nicht ohne eine gewisse Verzögerung möglich ist. Diese wird wohl in der Größenordnung von einigen zehn Mikrosekunden liegen (vgl. [WWDS94]). Wenn Prozess- bzw. Taktfrequenzwechsel nicht zu häufig erfolgen, sind die Verzögerungen im Verhältnis zur Gesamtrechenleistung jedoch so gering, dass ihre Auswirkungen durchaus vernachlässigbar sind.

3.1. Ziele und allgemeine Überlegungen

Wie bereits erwähnt, ist es das Ziel eine Strategie zu implementieren bei der das Verhältnis zwischen Dienstgüte und Verlustleistung durch Angabe einer gewünschten Batterielebensdauer, bzw. einer durchschnittlichen Leistungsaufnahme bestimmt wird.

Beide Ansätze sind eng miteinander verknüpft, da die Batterielebensdauer im wesentlichen von der mittleren Verlustleistung des Systems abhängt. Eine erste Näherung für die Batterielebensdauer erhält man, indem man Batteriekapazität durch Verlustleistung teilt, bzw. ergibt sich die Verlustleistung umgekehrt aus Kapazität durch geforderte Lebensdauer.

Aus den unter 2.2.3 genannten Gründen ist die Bestimmung des zu einer Batterielebensdauer gehörigen Verbrauchs jedoch nicht ganz so einfach. Statt dessen ist ein detailliertes

Batteriemodell oder eine Art Batterieanzeige nötig, um die Restlebensdauer zu bestimmen. An Hand der so ermittelten Werte muss dann die Verlustleistung des Prozessors periodisch angepasst werden, sodass die erwartete verbleibende Batterielebensdauer mit der geforderten übereinstimmt. Diese Aufgabe kann jedoch von einem auf Benutzerebene laufenden Hintergrundprozess erledigt werden. Aus Sicht des Betriebssystems genügt es also in beiden Fällen, Mechanismen zur Verfügung zu stellen mit denen die Verlustleistung des Prozessors eingestellt werden kann.

Wie bereits erwähnt, soll außerdem der Prozesskontext erweitert und jedem Prozess eine eigene Geschwindigkeit zugeordnet werden. Somit hat der Scheduler nun nicht mehr nur zu entscheiden wann ein Prozess ausgeführt wird, sondern auch wie schnell. Wenn Geschwindigkeitsumschaltungen keine, oder nur geringe Kosten verursachen, so kann man die Frage der Geschwindigkeit zunächst getrennt von Fragen der Reihenfolge betrachten. Die beiden Probleme sind voneinander unabhängig, sodass bestehende Scheduling-Strategien einfach um eine Komponente zur Geschwindigkeitsbestimmung erweitert werden können.

Allerdings ist immer dann Vorsicht geboten, wenn bei einer Scheduling-Strategie Entscheidungen auf Grund der Rechenzeit getroffen werden, wie z. B. bei MLFB (siehe [Tan92]). Da die Bearbeitungsgeschwindigkeit des Prozessors verändert werden kann, ist es möglich, dass zwei Prozesse zwar die selbe Rechenzeit bekommen haben, auf Grund unterschiedlicher Geschwindigkeiten jedoch keineswegs die selbe Anzahl an Befehlen ausführen konnten. Bei variabler Geschwindigkeit ist die Rechenzeit somit als Maß für den Rechenaufwand ungeeignet. Das Problem kann jedoch leicht behoben werden, wenn man statt der Rechenzeit auf die Anzahl der verbrauchten Taktzyklen zurückgreift.[1] Alternativ könnte man die Rechenzeit für jede Taktfrequenz unterschiedlich gewichten.

Ähnliches gilt auch für die Zeitscheibengröße: Je nach Geschwindigkeit kann innerhalb einer Zeitscheibe unterschiedlich viel Rechenarbeit geleistet werden. Ob die Zeitscheibenlänge deshalb mit der Geschwindigkeit erhöht oder gesenkt werden sollte ist fraglich. Beispielsweise würde bei einer Taktfrequenzreduzierung von 66 auf 1 MHz die Standard-Linux-Zeitscheibe von 200ms auf 13,2 Sek. erhöht! Für kürzere Prozesse würde der Round-Robin Scheduler damit zum FIFO-Scheduler degradiert. Sinnvoller ist es wohl, diesen Aspekt statt dessen bei der Wahl der Geschwindigkeit zu berücksichtigen, und darauf zu achten dass Prozesse durch Zeitscheiben mit langsamer Geschwindigkeit nicht unbeabsichtigt zurückgestuft werden.

Da die Ausführungsreihenfolge also – abgesehen von den gerade erwähnten Problemen – unabhängig von der Geschwindigkeit ist, wird im folgenden lediglich die Frage betrachtet, wie man die Taktfrequenz bestimmt mit der ein Prozess ausgeführt werden soll.

Für die Wahl der Geschwindigkeit eines Prozesse sind verschiedene Entscheidungskriterien denkbar. So könnte die Geschwindigkeit beispielsweise durch die Vorgabe einer Priorität durch den Benutzer bestimmt werden. Im Sinne einer fairen Energiezuteilung

[1] Tatsächlich ist dieses Verfahren jedoch nicht ganz korrekt, da die mittlere Anzahl der Taktzyklen pro Befehl selbst von der Geschwindigkeit abhängt (vgl. 2.2). Meist dürfte der Fehler jedoch so klein sein, dass die Anzahl der Taktzyklen als Maß für den Rechenaufwand genügt.

könnte man auch daran denken die Taktfrequenz vom bisherigen Energieverbrauch eines Prozesses abhängig zu machen.

In dieser Arbeit wurde die Geschwindigkeit an Hand der Rechenzeit bestimmt, um interaktive Prozesse zu bevorzugen (siehe 3.3).

3.2. Bestimmung des Energieverbrauchs eines Prozesses

Um einen vorgegebenen Verbrauch möglichst genau einzuhalten, ist es nötig den Energieverbrauch des Prozessors bzw. der einzelnen Prozesse möglichst genau abschätzen zu können. Wie in Abschnitt 2.3 dargestellt, hängt der Energieverbrauch des Prozessors hauptsächlich von Taktfrequenz und Spannung, aber auch von der Art der ausgeführten Befehle, dem Inhalt der Operandenregister und von der Anzahl der Speicherzugriffe ab.

Somit ist anzunehmen, dass auch verschiedene Prozesse einen sehr unterschiedlichen Stromverbrauch aufweisen, bestimmt durch die Anzahl der Speicherzugriffe und die Art der ausgeführten Befehle. In Tabelle 3.1 ist der Energieverbrauch des Prozessors bei Ausführung verschiedener Prozesse angegeben.

Geschwindigkeit	find	gcc	gzip
66 MHz	1569 mW	1580 mW	1554 mW
33 MHz	932 mW	944 mW	941 mW
16 MHz	500 mW	503 mW	506 mW
8 MHz	252 mW	251 mW	251 mW
4 MHz	160 mW	159 mW	156 mW
2 MHz	106 mW	104 mW	100 mW
1 MHz	68 mW	65 mW	069 mW

Tabelle 3.1.: Verlustleistung bei Ausführung verschiedener Prozesse

Bei „find” handelt es sich um das normale Linux find Programm, das alle Dateien auf der lokalen Platte des Rechners ausgibt, „gcc” ist der GNU C-Compiler, wobei ein relativ kurzes C Programm übersetzt wird. „gzip” ist das Linux GNU Zip Kompressionsprogramm, komprimiert wird eine 2 MB große Textdatei.

Wie man sieht unterscheidet sich der Energieverbrauch der verschiedenen Prozesse eher geringfügig, um maximal 30 mW bei 66 MHz. Die Verlustleistung ist wesentlich höher, als die bei Ausführung der unter 2.3 angegebenen Befehle. Somit scheint der Energieverbrauch der Prozesse hauptsächlich von den Speicherzugriffen beeinflusst zu werden, so dass die Anzahl der Speicherzugriffe evtl. ein Maß für den Energieverbrauch eines Prozesses wäre.

Noch genauer ließe sich der Energieverbrauch jedoch bestimmen, wenn zusätzlich Anzahl und Art der ausgeführten Befehle bekannt wäre. Mit Hilfe einer Tabelle für CPI und

Verlustleistung der einzelnen Befehl ließe sich dann der Energieverbrauch des Prozesses bestimmen, wie unter 2.3 beschrieben.

Um die Anzahl der ausgeführten Befehle zu bestimmen, könnten die in vielen neueren Prozessoren vorhandenen Ereigniszähler (siehe z. B. [Int97]) verwendet werden.

Da der AMD Èlan SC410 jedoch keine Möglichkeiten zur Verfügung stellt, um die Anzahl der Speicherzugriffe oder Befehlsausführungen eines Prozesses zu bestimmen, bleibt in diesem Fall lediglich die Rechenzeit als Maß für den Energieverbrauch. Die Rechenzeit wird dann mit dem vorher gemessenen Wert für die Verlustleistung multipliziert.

3.3. Prozessbasierte Strategie zur Energieverwaltung

Die Wahl der Geschwindigkeit lässt sich in zwei Probleme teilen: Zum einen muss geklärt werden, wie viel Energie innerhalb eines bestimmten Zeitintervalls verbraucht werden darf, zum anderen wie diese Energie auf die Prozesse „verteilt" werden soll.

Die erste Frage mag zunächst überflüssig erscheinen, da das Ziel der Geschwindigkeitsanpassung ja gerade die Einhaltung eines *vorgegebenen* Verbrauchs ist. Der Verbrauch des Prozessors hängt jedoch nicht nur von der Geschwindigkeit ab, sondern auch von der Auslastung. Wenn der Prozessor leersteht wird der Stromverbrauch sehr gering, so dass die mittlere Verlustleistung sinkt. Zum Ausgleich kann der Prozessor später mit höherer Geschwindigkeit und höherer Verlustleistung betrieben werden, so lange bis der mittlere Verbrauch sich der Vorgabe wieder weitgehend angenähert hat. Aus dem vorgegebenem „Soll-Verbrauch" und dem tatsächlichem „Ist-Verbrauch" muss also periodisch eine neue Vorgabe berechnet werden.

Die Regelung des Verbrauchs und die Verteilung auf die verschiedenen Prozesse werden in den nächsten Abschnitten näher betrachtet.

3.3.1. Regelung des Energieverbrauchs

Da die Verlustleistung eines Prozessors auch bei gleichbleibender Geschwindigkeit nicht völlig konstant ist, muss die Geschwindigkeit periodisch angepasst werden um die vorgegebene Verlustleistung einzuhalten. Aus dem Unterschied zwischen tatsächlichem und geplantem Verbrauch muss ein neuer Zielverbrauch errechnet werden, mit dem dann wiederum die Geschwindigkeiten der einzelnen Prozesse festgelegt werden können (siehe Abschnitt 3.3.2).

Durch eine Regelung des Zielverbrauchs werden noch zwei weitere Probleme beseitigt: Da die Taktfrequenz nur stufenweise angepasst werden kann, kann die Geschwindigkeit der Prozesse nur selten so gewählt werden, dass eine Vorgabe genau eingehalten wird. Die dadurch entstehenden Abweichungen zwischen Soll- und Ist-Verbrauch werden durch den Regler mittelfristig ausgeglichen.

Außerdem verbraucht ein Prozessor im Leerlauf nur sehr wenig Strom. Dadurch sinkt die mittlere Verlustleistung, so dass der Prozessor entsprechend schneller betrieben werden kann, wenn wieder Rechenlast vorhanden ist. Auch der Ausgleich dieser Leerlaufperioden kann durch den Regler erfolgen.

Ein einfacher Proportional-Regler könnte so aussehen, dass sich der Zielverbrauch Z_t zum Zeitpunkt t aus dem vorgegebenen Soll-Verbrauch P_S und der tatsächlichen mittleren Leistungsaufnahme $P_M(t)$ folgendermaßen zusammensetzt: $Z_t = P_S + c \cdot (P_S - P_M(t))$. Die Konstante c steuert, wie schnell die Abweichungen vom Zielverbrauch ausgeglichen werden.

Dieser Ansatz bringt jedoch gewisse Probleme mit sich. Die mittlere Verlustleistung $P_M(t)$ zum Zeitpunkt t ergibt sich aus dem Integral (bzw. am Rechner aus der Summe) der Verlustleistung über die Zeit, dividiert durch t, also

$$P_M(t) = \frac{\int_0^t P_I(\tau)\,\mathrm{d}\tau}{t}$$

$P_I(\tau)$ bezeichnet die aktuelle Verlustleistung des Prozessors zum Zeitpunkt τ. Das Problem besteht nun darin, dass auch große Abweichungen der aktuellen Verlustleistung P_I vom Soll-Wert bei großem t nur sehr geringe Änderungen an der mittleren Verlustleistung P_M verursachen. Dadurch wird die Regelung bei großem t sehr ungenau, bzw. sehr schwerfällig, unabhängig davon, wie groß man c am Anfang wählt. Auch bei einer zeitabhängige „Konstante" $c(t)$ treten Probleme auf, da die Abweichungen vom Mittelwert irgendwann so klein werden, dass sie auf dem Rechner evtl. nicht mit hinreichender Genauigkeit dargestellt werden können.

Deshalb wurde in dieser Arbeit der folgende Regelalgorithmus implementiert: Zunächst wird die Abweichung D_t zwischen geplanter und tatsächlicher Energieaufnahme (in J) zum Zeitpunkt t berechnet:

$$D_t = t \cdot P_S - \int_0^t P_I(\tau)\,\mathrm{d}\tau$$

Das Integral wird durch die Summe über die verbrauchten Zeitscheiben aller Prozesse, multipliziert mit der jeweiligen Verlustleistung angenähert. Der neue Zielverbrauch zum Zeitpunkt t wird aus

$$Z_t = P_s + \frac{D_t}{t_R}$$

errechnet, wobei t_R die Dauer zwischen zwei Aufrufen der Regelungsroutine ist. Wenn Z_t genau eingehalten wird, so ist der Fehler zum Zeitpunkt $t + t_R$ offensichtlich ausgeglichen und Soll- und Ist-Verbrauch stimmen überein. Tatsächlich wird das natürlich nicht der Fall sein, da es aus den oben genannten Gründen zumeist nicht möglich ist, einen vorgegebenen Verbrauch genau einzuhalten.

An Stelle der Dauer eines Regelungsintervalls könnte man für t_R auch höhere Werte wählen, so dass der Zielverbrauch weniger stark von der Abweichung beeinflusst wird und schwächer schwankt. Ein niedrigerer Wert sollte allerdings nicht verwendet werden.

In diesem Fall würde der Fehler häufig bereits wieder in die Gegenrichtung ausschwingen, bevor der Regler erneut aufgerufen wird (vorausgesetzt Z_t wird eingehalten).

Setzte man an Stelle von t_R die Gesamtlaufzeit t ein, so ergibt sich wieder der oben beschriebene Proportionalregler.

3.3.2. Bestimmung der Geschwindigkeit

Nachdem ein kurzfristiger Zielverbrauch Z (in mW) aufgestellt wurde muss entschieden werden, wie schnell jeder einzelne Prozess laufen darf. Die Geschwindigkeiten müssen dann so bestimmt werden, dass die mittlere Verlustleistung P des Prozessors kleiner als Z ist.

Die Wahl der Geschwindigkeit f_i des Prozesses i kann sich, wie bereits erwähnt, auf verschiedene Merkmale stützen. Allgemein wird man jedem Prozess eine Bewertung b_i zuordnen können, die dann bei der Wahl der Geschwindigkeit berücksichtigt wird, indem Prozessen mit hoher Bewertung eine hohe Geschwindigkeit und solchen mit niedriger Bewertung eine geringere Taktfrequenz zugeordnet wird.

Die Geschwindigkeit eines Prozesses hängt jedoch nicht nur von dieser Bewertung ab, sondern auch von allen anderen Prozessen und deren Bewertung. Denn wird ein Prozess beschleunigt, so muss gleichzeitig ein anderer verlangsamt werden, da ja insgesamt eine vorgegebene Verlustleistung eingehalten werden muss. Die Geschwindigkeit der Prozesse muss deshalb nicht nur dann neu berechnet werden, wenn Z neu bestimmt wurde, sondern auch dann, wenn sich die Anzahl der lauffähigen Prozesse im System oder deren Bewertung ändert.

Um die Taktfrequenzen so zu bestimmen, dass die Vorgabe für die mittlere Verlustleistung eingehalten wird, ist es natürlich nötig die Leistungsaufnahme abschätzen zu können. In einem Round-Robin basierten System lässt sich die mittlere Verlustleistung P folgendermaßen bestimmen:

$$P = \frac{\sum_{i=1}^{N} t_i \cdot p_i(f_i)}{\sum_{i=1}^{N} t_i}$$

Dabei ist t_i die Zeitscheibenlänge, f_i die Geschwindigkeit und $p_i(f)$ die Verlustleistung von Prozess i bei Geschwindigkeit f und N die Anzahl der lauffähigen Prozesse.

Nun lassen sich die Geschwindigkeiten der Prozesse über ein Gleichungssystem bestimmen. Wenn man beispielsweise fordert, dass sich die Geschwindigkeit direkt proportional zu b_i verhält, also $f_i = g \cdot b_i$, so ergibt sich die gesuchte Geschwindigkeitsverteilung (d. h. der gesuchte Parameter g) aus

$$Z = \frac{\sum_{i=1}^{N} t_i p_i(b_i \cdot g)}{\sum_{i=1}^{N} t_i}$$

Dieses Vorgehen hat jedoch verschiedene Nachteile: Zum einen müssen die $p_i(f)$ als Funktion angegeben werden. Das wird nicht immer möglich sein, da die Verlustleistung des Prozessors ja nur bei den verschiedenen tatsächlich möglichen Geschwindigkeitsstufen bestimmt werden kann. Aber auch wenn der Energieverbrauch als Funktion dargestellt werden kann, handelt es sich bei den $p_i(f)$ in der Regel um Polynome dritten Grades (siehe 2.2.4), und die Lösung der Gleichung ist entsprechend aufwändig. Zum anderen ergeben sich aus der Gleichung Geschwindigkeiten die in der Regel nicht mit den möglichen Geschwindigkeitsstufen übereinstimmen werden. Statt mit der errechneten Taktfrequenz können die Prozesse nur mit der nächstkleineren Geschwindigkeitsstufe bearbeitet werden.

Da die Geschwindigkeit ohnehin nur stufenweise verändert werden kann liegt es nahe, die Geschwindigkeit nicht für jeden Prozess einzeln zu bestimmen, sondern die Prozesse in Klassen einzuteilen und alle Prozesse einer Klasse mit der gleichen Taktfrequenz zu bearbeiten.

Im praktischen Teil dieser Studienarbeit wurde eine Strategie mit zwei Klassen implementiert. Eine für kurze, interaktive Prozesse die schnell laufen sollen, und eine für langlaufende Hintergrundprozesse, die entsprechend langsamer ausgeführt werden. Die Verwendung von nur zwei Klassen hat den Vorteil, dass das Zuordnen der Geschwindigkeiten stark vereinfacht werden kann. Außerdem ist auch das einordnen der Prozesse wesentlich einfacher als es bei mehreren Klassen der Fall wäre. Des weiteren stellt der AMD Èlan SC410 ohnehin nur sechs verschiedene Geschwindigkeiten zur Verfügung (siehe Abschnitt 4.1.1), so dass bei einer höheren Klassenanzahl verschiedene Klassen häufig trotzdem die gleiche Geschwindigkeit hätten.

Zusätzlich wird der unterschiedliche Verbrauch verschiedener Prozesse nicht weiter berücksichtigt, sondern zur Vorhersage der Verlustleistung wird ein für alle Prozesse gleicher Verbrauch $p(f)$ angenommen. Dadurch wird die Bestimmung der Geschwindigkeiten weiter vereinfacht, der so verursachte Fehler wird durch die periodische Regelung des Vorgabewertes ausgeglichen.

Die Geschwindigkeiten f_s und f_l für schnelle und langsame Klasse werden dann folgendermaßen bestimmt: Zunächst wird f_s auf die schnellste mögliche Geschwindigkeit (f_{max}) und f_l auf die langsamste Geschwindigkeit (f_{min}) gesetzt. Die Verlustleistung ergibt sich damit, bei n_l Prozessen in der langsamen und n_s Prozessen in der schnellen Klasse, aus

$$P = \frac{n_l \cdot p(f_{min}) + n_s \cdot p(f_{max})}{n_l + n_s}$$

Ist $P < Z$, so kann f_l erhöht werden bis $P > Z$ ist. Die letzte Geschwindigkeit bei der P noch im erlaubten Bereich war, ist die optimale. Analog wird bei $P > Z$ f_s so lange gesenkt bis P einen zulässigen Wert erreicht.

Neben der Geschwindigkeit mit der Prozesse aus einer Klasse ausgeführt werden, muss die Zuordnung der Prozesse bestimmt werden. Hier bietet es sich an, Schwellwerte zu benutzen und Prozesse einer Klasse zuzuordnen, wenn beispielsweise ihr bisheriger Energieverbrauch oder die verbrauchte Rechenzeit zwischen zwei Schwellwerten liegt.

Dabei sollte natürlich nicht die gesamte Rechenzeit eines Prozesses berücksichtigt werden: Auch ein interaktiver Prozess wird, wenn er lange genug existiert, irgendwann jeden Schwellwert überschreiten. Statt dessen sollte der Anteil der Rechenzeit an der Gesamtzeit im System berücksichtigt werden. Aber auch hier ist Vorsicht geboten: Wenn ein Prozess zunächst eine Eingabe vom Benutzer verlangt und lange warten muss, so kann er später entsprechend lange mit hoher Geschwindigkeit arbeiten, obwohl es sich eigentlich nicht mehr um einen interaktiven Prozess handelt. Somit sollte immer nur das Verhalten innerhalb einer kurzen Zeitspanne berücksichtigt werden, nicht seit der Ankunft im System.

Im Rahmen dieser Arbeit wurde die Klassenzugehörigkeit durch folgende einfache Heuristik bestimmt: Jeder Prozess, der lauffähig wird, wird der schnellen Klasse zugeordnet. Jeder Prozess der länger als drei Zeitscheiben rechnet wird der langsamen Klasse zugeordnet. Dadurch wird erreicht, dass Prozesse die sich häufig blockieren bevorzugt behandelt werden.

4. Beschreibung der Implementierung

4.1. Plattform

Im Rahmen dieser Arbeit wurde das Betriebssystem Linux um eine Komponente zur Energieverwaltung erweitert. Die Umsetzung erfolgte auf dem DIMM-PC/486-I ([JUM97]), einem AMD Èlan SC410 Prozessor mit 16 MB Hauptspeicher sowie 16MB Flash-Harddisk (JUMPtec Chipdisk-IDE). Zusätzlich wurde eine Graphikkarte (JMUPtec DIMM-PC/VGA1) und ein Netzwerkadapter (JUMPtec DIMM-PC/ETH1) verwendet.

Außerdem wurde „White-Dwarf-Linux", eine auf Version 2.2.12 basierende, an den DIMM-PC angepasste Linux Version verwendet.

In den beiden folgenden Abschnitten wird ein kurzer Einblick in die Möglichkeiten der Taktfrequenzsteuerung beim AMD Èlan SC410 gegeben und der Linux Scheduler wird grob skizziert.

4.1.1. Taktfrequenzsteuerung beim AMD Èlan SC410

Der AMD Èlan SC410 ist mit einer sogenannten „Power Management Unit" (PMU) ausgestattet, die es erlaubt, die Taktfrequenz der momentanen Systemlast anzupassen. Zu diesem Zweck kann die PMU so programmiert werden, dass sie die Geschwindigkeit automatisch verringert, wenn der Prozessor längere Zeit inaktiv war, bzw. erhöht, wenn eine hohe Rechenlast vorhanden ist.

Alternativ kann man die Geschwindigkeit auch durch Register der PMU steuern. In dieser Arbeit wurde der letztere Ansatz gewählt, da komplexere Geschwindigkeitsanpassungsstrategien mit den Mitteln der PMU nicht realisierbar sind. Außerdem wäre eine Prozessbasierte Geschwindigkeitsänderung nicht möglich.

Die PMU bietet sieben verschiedene „PMU-Modes" zur Geschwindigkeitsanpassung. Den Modi „Hyper- ", „High-" und „Low-Speed Mode" ist jeweils eine bestimmte Taktfrequenz zugeordnet. Während der Hyperspeedmode stets mit 66 MHz arbeitet, kann die Geschwindigkeit für den High-Speed Mode auf 33, 16 oder 8 MHz und für den Low-Speed Mode auf 8, 4, 2 oder 1 MHz gesetzt werden. Die verbleibenden vier PMU-Zustände sind unterschiedliche „Schlaf-Zustände" der CPU und sind für die Taktfrequenzsteuerung uninteressant. Insgesamt sind somit sieben verschiedene Geschwindigkeiten möglich.

Alle benötigten Taktfrequenzen werden mit Hilfe eines einzigen 32 kHz Eingangstaktes erzeugt. Durch mehrerer „Phase Locked Loops" (PLLs) wird ein 66 MHz Signal gewonnen. Dieses wiederum wird durch Frequenzteiler in 1, 2, 4, 8, 16 und 33 MHz Signale aufgeteilt.

Da die verschiedenen Taktfrequenzsignale gleichzeitig erzeugt werden und immer mitlaufen, kann die CPU-Taktfrequenz sehr schnell umgeschaltet werden. Wenn nötig kann mit jedem Takt des 32 kHz Quarz eine Geschwindigkeitsänderung erfolgen.

Die 66 MHz Geschwindigkeit im „Hyper-Speed Mode" wird durch einen zusätzlichen PLL im CPU Kern aus dem 33 MHz Takt erzeugt. Dadurch entsteht beim Wechsel in diesen Modus eine Verzögerung von einer Millisekunde.

CPU 1x clock	33 (66)	16	8	4	2	1
DRAM [MHz]	66	33	16	8	4	2
ISA [MHz]	8	8	8	4	2	1
DMA [MHz]	16	16	8	4	2	1
ROM [MHz]	33	16	7	4	2	1

Tabelle 4.1.: Taktfrequenzen beim AMD Èlan SC410 (Quelle: [AMDa])

Die Taktraten anderer Komponenten des SC410 werden bei Geschwindigkeitsänderungen ebenfalls angepasst, wie der Tabelle 4.1 (vgl. [AMDa]) zu entnehmen ist.

Insbesondere ist zu beachten, dass der Speichertakt so angepasst wird, dass er mit doppelter CPU-Frequenz läuft. Lediglich bei einer Geschwindigkeit von 66 MHz kann der Speichertakt nicht weiter erhöht werden, so dass CPU und Speicher gleich schnell getaktet sind, was zu einer entsprechenden Verzögerung bei Speicherzugriffen führt (siehe Abschnitt 2.2.2).

4.1.2. Aufbau des Linux Schedulers

Der Linux Scheduler ist im wesentlichen ein Round-Robin Scheduler, der um Prioritäten und Scheduling Klassen erweitert wurde. Ein Prozess kann einer der Scheduling Klassen `SCHED_FIFO`, `SCHED_RR` oder `SCHED_OTHER` angehören.

Zunächst werden die Prozesse der Klassen `SCHED_FIFO` und `SCHED_RR` abgearbeitet. Die Bearbeitungreihenfolge wird durch die Prioritäten bestimmt (höchste Priorität zuerst). Bei Prozessen mit gleicher Priorität entscheidet die Position in der Bereit-Warteschlange, wobei neue Prozesse stets *an den Anfang* der Warteschlange gestellt werden.

Prozesse der Klasse `SCHED_FIFO` werden so lange bearbeitet, bis sie das System verlassen, oder von einem neu ankommenden Prozess höherer oder gleicher Priorität[1] verdrängt werden. Prozesse der Klasse `SCHED_RR` werden nach dem Ende ihrer Zeitscheibe verdrängt und am Ende der Warteschlange eingereiht.

[1] Da neu ankommende Prozesse an den Anfang der Warteschlange gestellt werden, kann ein laufender Prozess auch von einem Prozess gleicher Priorität verdrängt werden.

Sind keine Prozesse aus den beiden oben genannten Klassen im System, werden diejenigen der Klasse `SCHED_OTHER` nach Round Robin abgearbeitet. Die Ausführungsreihenfolge innerhalb einer Round-Robin Runde wird wiederum durch Priorität und Position in der Bereit-Warteschlange bestimmt.

Da ein neu ankommender Prozess am Anfang der Warteschlange eingereiht wird, wird er sofort bearbeitet, vorausgesetzt es existiert kein Prozess höherer Priorität. Erst wenn der Prozess das System wieder verlässt oder seine Zeitscheibe aufgebraucht hat darf der vorherige Prozess weiterarbeiten, bis er den Rest seiner eigenen Zeitscheibe verbraucht hat.

Wenn alle Prozesse ihre Zeitscheibe aufgebraucht haben werden diese neu berechnet. Dabei gibt die Priorität des Prozesses die Länge seiner Zeitscheibe in 10ms Einheiten („jiffies") an.

Außerdem wird auch die Zeitscheibengröße der Prozesse angepasst, die momentan nicht lauffähig sind. Dabei wird die Zeitscheibengröße schrittweise erhöht bis hin zum doppeltem der normalen Zeitscheibe: Die Zeitscheibe Z_t in der t-ten Runde nachdem der Prozess den Zustand Bereit verlassen hat ergibt sich aus $Z_t = \frac{Z_{t-1}}{2} + T$, wobei T die normale Zeitscheibenlänge des Prozesses, also Priorität·10ms, angibt. Diese rekursive Berechnung entspricht folgendem Zusammenhang: $Z_t = Z_0 \cdot 2^{-t} + T \cdot (2 - 2^{-t+1})$, wobei Z_0 den Rest der Zeitscheibe des Prozess bei Aufgabe der CPU bezeichnet. Für große t nähert sich die Zeitscheibenlänge also $2T$ an. Dadurch bekommen Prozesse, die lange Zeit inaktiv waren einen gewissen Vorteil vor Prozessen, die lange Zeit rechnen.

Das Anwachsen der Zeitscheibenlänge ist in Abbildung 4.1 wiedergegeben (mit $Z_0 = 0,5T$).

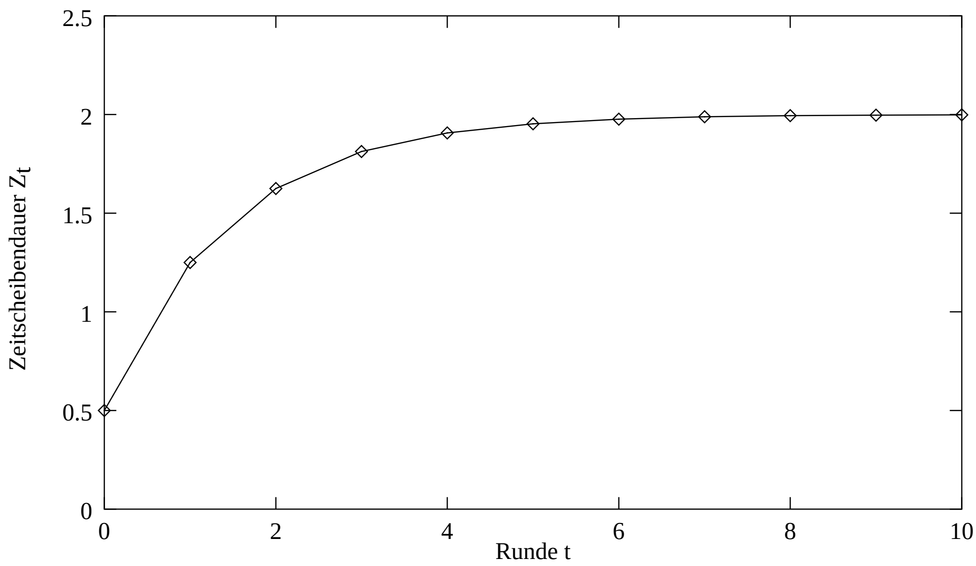

Abbildung 4.1.: Zeitscheibengröße nach t Runden.

4.2. Änderungen am Linux Kern

4.2.1. Überblick

Die im Rahmen dieser Arbeit implementierte Energieverwaltung bietet die Möglichkeit die Geschwindigkeit des Prozessors entweder durch die Angabe einer bestimmten Geschwindigkeit oder durch Vorgabe eines Energieverbrauchslimit zu bestimmen. Wird der Verbrauch vorgegeben, so wird die Geschwindigkeit der Prozesse automatisch bestimmt. Dabei wird das unter 3.3.2 beschriebene Verfahren mit zwei Geschwindigkeitsklassen verwendet. Zusätzlich besteht die Möglichkeit einzelnen Prozessen eine feste Geschwindigkeit, oder eine feste Geschwindigkeitsklasse zuzuordnen.

Der Energieverbrauch der Prozesse wird über deren Rechenzeit bestimmt (siehe 4.2.2), da der AMD Èlan SC410 keine anderen Möglichkeiten zur Verfügung stellt. Der Energieverbrauch ergibt sich aus der Rechenzeit, multipliziert mit einem prozess- und geschwindigkeitsabhängigem Faktor für die Verlustleistung.

Der Prozesskontext wurde um einen Eintrag für die Geschwindigkeit erweitert. Außerdem wurden Einträge für die verbrauchte Energie sowie für die Verlustleistung bei unterschiedlichen Geschwindigkeiten hinzugefügt.

4.2.2. Energieverbrauchsbestimmung

Wie bereits erwähnt, benötigt man für eine möglichst genaue Abschätzung des Stromverbrauchs eines Prozessors bzw. eines Prozesses Informationen über die Aktivitäten des Prozessors, insbesondere über die Anzahl der Speicherzugriffe. Leider fehlen dem AMD Èlan SC410 entsprechende Möglichkeiten.

Das einzige zur Verfügung stehende Maß für den Energieverbrauch eines Prozesses ist deshalb seine Rechenzeit, die mit einem festen (vom Benutzer vorgegebenen) Wert für die Verlustleistung multipliziert wird.

Zur Bestimmung der Rechenzeit benötigt man eine möglichst genaue Uhr. Die Echtzeit-Uhr des AMD Èlan SC410 ist für diese Zwecke ungeeignet, da sie mit einer Taktfrequenz von 32 kHz arbeitet und somit nur eine Auflösung von mehr als 31μs bietet. Eine genauere Zeitauflösung kann durch Nutzung eines „Programmable Intervall Timer" (PIT) erreicht werden. Der PIT besteht aus einem 16 Bit breiten Zähler, der zunächst auf einen beliebigen Ausgangswert gesetzt wird. Dieser Wert wird dann mit einem Takt von 1,1892 MHz heruntergezählt, bis der Zählerstand 0 erreicht. Dann wird ein Interrupt ausgelöst, der Zähler wird wieder auf den Ausgangswert gesetzt und das ganze beginnt von neuem.

Zusätzlich kann der aktuelle Zählerstand jederzeit ausgelesen werden. Durch das Zählen der Interrupts und das Auslesen des Zählerstands ist es theoretisch möglich, Zeitmessungen mit einer Auflösung von unter $1\,\mu$s zu erreichen. Allerdings sind zum Auslesen des Zählerregisters mehrere Buszugriffe nötig, so dass das Auslesen relativ viel Zeit in

Anspruch nimmt. Zusätzlich variiert die Geschwindigkeit mit der CPU bzw. Bus Taktfrequenz. In Tabelle 4.2 sind die Zeiten angegeben die bei unterschiedlichen Geschwindigkeiten zum Auslesen benötigt werden. Eine genauere Bestimmung der Rechenzeit ist mit dem AMD Èlan SC410 jedoch nicht möglich.

Taktfrequenz	Zugriffszeit
66 MHz	10 μs
33 MHz	15 μs
16 MHz	23 μs
8 MHz	26 μs
4 MHz	29 μs
2 MHz	31 μs
1 MHz	32 μs

Tabelle 4.2.: Benötigte Zeit zum Auslesen des PIT-Registers.

Die Bestimmung des Energieverbrauchs wird in der Funktion `pmu_update_balance()` durchgeführt. Diese muss vor jedem Kontextwechsel und vor jedem Geschwindigkeitswechsel aufgerufen werden, da der Energieverbrauch immer an Hand der aktuellen Geschwindigkeit berechnet und dem aktuellen Prozess zugeordnet wird.

Zunächst wird mit dem oben erläuterten Verfahren ermittelt, wie viel Zeit seit dem letzten Aufruf der Funktion vergangen ist, dieser Wert wird dann mit der Verlustleistung des aktuellen Prozesses multipliziert. Ist für den aktuellen Prozess keine Verlustleistung angegeben, so wird ein Standardwert angenommen. Der so ermittelte Wert für die Energieaufnahme wird zu der globalen und zur prozesslokalen Variable für den Energieverbrauch hinzu addiert. Ist kein Prozess aktiv, so wird die verbrauchte Energie dem idle-Prozess zugeschrieben.

Wenn die Taktfrequenzsteuerung aktiv ist, wird außerdem an Hand des gesetzten Limits für die Verlustleistung berechnet, wie hoch die Energieaufnahme sein sollte und die Differenz zwischen Soll und Ist-Wert wird in der Variable `pmu_balance` vermerkt.

4.2.3. Änderungen am Scheduler

Nachdem festgelegt wurde, welcher Prozess als nächstes ausgeführt wird, muss die Geschwindigkeit des Prozesses evtl. neu bestimmt werden und die Geschwindigkeit der CPU entsprechend gesetzt werden.

Zunächst wird der Energieverbrauch des gerade unterbrochenen oder beendeten Prozesses in dessen Kontext vermerkt (siehe 4.2.2). Anschließend muss die Klassenzugehörigkeit des Prozesses überprüft werden. Befindet sich der Prozess in der schnellen Klasse, so wird geprüft, wie viele Zeitscheiben der Prozess noch rechnen darf bevor er in die langsamere Klasse eingereiht wird. Hat er alle Zeitscheiben in der schnellen Klasse aufgebraucht wird er der langsamen zugeordnet. Befindet sich der Prozess umgekehrt in der langsamen

Klasse und hat den lauffähigen Zustand verlassen (weil er z. B. auf eine E-/A-Operation wartet), so wird die Anzahl der Zeitscheiben in der schnellen Klasse wieder erhöht.

Die Regelung des Energieverbrauchs (vgl. 3.3.1) und die Bestimmung der Geschwindigkeit (siehe 3.3.2) werden in der Funktion `pmu_recalculate_speed` durchgeführt. Diese Funktion muss aufgerufen werden, wenn entweder eine Neuberechnung der Geschwindigkeit durchgeführt werden muss (weil sich die Anzahl der Prozesse im System oder deren Klassenzugehörigkeit geändert hat) oder der zulässige Energieverbrauch neu bestimmt werden muss. Dazu wird geprüft, ob der gerade unterbrochene Prozess in eine andere Klasse eingereiht wurde oder nicht mehr lauffähig ist, oder ob der nächste auszuführende Prozess gerade erst lauffähig geworden ist. Wenn eine der Bedingungen zutrifft müssen die Geschwindigkeiten neu berechnet werden. Die Regelung des Energieverbrauchs wird bei Neuberechnung der Geschwindigkeit und spätestens nach einer Sekunde durchgeführt.

Zuletzt muss die Geschwindigkeit des Prozessors durch Aufruf der Funktion `pmu_do_speed_change` auf die Taktfrequenz des Prozesses gesetzt werden. Danach kann der eigentliche Kontextwechsel wie im normalen Linux ablaufen.

Bestimmung der Geschwindigkeiten

Zur Bestimmung der Taktfrequenzen wird zunächst angenommen, dass die Prozesse der langsamen Klasse mit minimaler, die der schnellen mit maximaler Geschwindigkeit ausgeführt werden. Die daraus resultierende mittlere Verlustleistung wird abgeschätzt durch

$$P = \frac{n_l p(f_{min}) + n_s p(f_{max})}{n_l + n_s}$$

wobei n_l die Anzahl der Prozesse in der langsamen und n_s die der schnellen Klasse ist und $p(f_{min})$ die Verlustleistung bei minimaler, $p(f_{max})$ die Verlustleistung bei maximaler Geschwindigkeit ist. Statt der im Prozesskontext vermerkten Verlustleistung der einzelnen Prozesse wird die Standard-Verlustleistung verwendet, um die Berechnung zu beschleunigen.

Liegt die so errechnete Abschätzung über dem zulässigen Wert, so wird die Geschwindigkeit der schnellen Klasse gesenkt, anderenfalls die der langsamen Klasse erhöht, so lange bis ein passender Verbrauchswert erreicht wird.

4.2.4. Schnittstelle

Zur Bestimmung der Geschwindigkeit bzw. der mittleren Verlustleistung werden folgende Betriebssystemaufrufe zur Verfügung gestellt:

`int set_speed(int speed, pid_t pid)`
: Dient zum setzen einer festen Geschwindigkeit. Die Geschwindigkeit kann für den ganzen Prozessor gesetzt werden oder für einzelne Prozesse. Hat ein Prozess keine

eigene Geschwindigkeit in seinem Kontext vermerkt, so wird er mit einer Standardgeschwindigkeit ausgeführt oder die Geschwindigkeit wird, ist die automatische Taktfrequenzregelung aktiv, an Hand seiner Klassenzugehörigkeit bestimmt.

- `speed` gibt die neue Geschwindigkeit an. Werte von 0 bis 6 bezeichnen die verschiedenen Geschwindigkeitsstufen von 1 bis 66 MHz. Mit 7 und 8 wird der Prozess der langsamen bzw. schnellen Geschwindigkeitsklasse zugewiesen, ein Wert von -1 gibt an, dass der Prozess mit der Standardgeschwindigkeit ausgeführt werden soll.
- `pid` gibt an, für welche Prozesse die gesetzte Geschwindigkeit gilt. Es kann entweder eine gültige PID angegeben werden, 0 für den aktuellen Prozess oder -1 um die Standardgeschwindigkeit zu verändern.

`int pmu_set_schedpar(int dosched, int newlimit, int reset)`

Bestimmt ob die Geschwindigkeit automatisch, durch ein Verbrauchslimit, bestimmt wird und setzt gegebenenfalls den gewünschten Energieverbrauch. Wird die automatische Taktfrequenzbestimmung abgeschaltet, so wird die Geschwindigkeit nur durch `set_speed` Aufrufe bestimmt. Anderenfalls wird die Geschwindigkeit aller Prozesse ohne eine fest zugeordnete Taktfrequenz an Hand der Verlustleistung berechnet, wie unter 3.3.2 erläutert.

- `dosched` gibt an ob die Geschwindigkeit automatisch bestimmt werden soll oder nicht. Mit 1 wird die Taktfrequenzregelung eingeschaltet, mit 0 ausgeschaltet.
- `newlimit` gibt das Limit für die mittlere Verlustleistung an.
- `reset` bestimmt, ob die bisher verbrauchte Energie bei der Berechnung der mittleren Verlustleistung berücksichtigt werden soll, oder nicht. Mit 1 wird die Energiebilanz auf 0 gesetzt und nur die Energieaufnahme seit dem Aufruf von `pmu_set_schedpar` wird berücksichtigt. Wird der Parameter auf 0 gesetzt, so werden auch die vorherigen Werte berücksichtigt.

`int sys_set_current_consumption(pmu_current_t *curr, pid_t pid)`

Speichert die mittlere Verlustleistung eines Prozesses in dessen Prozesskontrollblock. Da die Abschätzung der Energieaufnahme eines Prozesses, wie bereits erwähnt, nur an Hand der Rechenzeit erfolgen kann, wird ein vorgegebener Mittelwert für die Verlustleistung benötigt. Zur Errechnung des Energieverbrauchs wird dieser Wert mit der Rechenzeit multipliziert.

- `curr` ist ein Zeiger auf eine Struktur, die die Verlustleistung für die unterschiedlichen Geschwindigkeiten enthält. Der Typ `pmu_current_t` wird in `pmu.h` definiert und enthält neben einem Feld für die Verlustleistungen eine Maskenvariable mit der angegeben werden kann, welche Einträge verändert werden sollen.

- `pid` gibt an, für welchen Prozess die Verbrauchswerte gesetzt werden sollen. Es kann entweder eine gültige PID angegeben werden, 0 für den aktuellen Prozess oder -1 um eine Standard-Verlustleistung zu setzen, die immer dann verwendet wird wenn einem Prozess keine speziellen Werte zugeordnet wurden.

`pmu_acct_ctrl(int arg)`
: Die Funktion dient dazu die Zeit- bzw. Energieverbrauchsmessung zu Debug-Zwecken auszuschalten bzw. alle Werte zurückzusetzen.

Außerdem werden folgende Dateien ins `proc` Dateisystem eingebunden:

`/proc/pmu`
: Über diese Datei können Informationen über die Energieverwaltung gelesen werden. Im einzelnen werden aktueller PMU-Mode und Geschwindigkeit, die Zeit die in den verschiedenen Geschwindigkeitsstufen verbracht wurde (gegliedert in Rechen- und Leerzeit), das eingestellte Limit für die mittlere Verlustleistung, sowie die tatsächliche mittlere Verlustleistung ausgegeben.

`/proc/pmu_consumption`
: Ein Lesezugriff liefert die Werte für die Standardverlustleistung zurück, die für die Berechnung des Energieverbrauchs der Prozesse verwendet wird, denen keine spezielle Verlustleistung zugeordnet wurde.
Mit einem Schreibzugriff lassen sich die Werte verändern.

4.3. Ergebnisse

4.3.1. Regelung

In der Abbildung 4.2 ist die Verlustleistung des Prozessors bei Ausführung des Linux find Programms über 60 Sekunden (100 Messwerte) hinweg aufgetragen, außerdem ist der Mittelwert angegeben. Als Verbrauchslimit wurden 1200 mW angegeben.

Man sieht deutlich, wie die Verlustleistung zwischen zwei Geschwindigkeiten mit 900 und 1600 mW Verlustleistung schwankt. Die mittlere Leistungsaufnahme nähert sich jedoch den eingestellten 1200 mW.

In Tabelle 4.3 ist die mittlere Verlustleistung für weitere Vorgabewerte und Prozesse angegeben. Bei allen Messungen wurden die gleichen Standardwerte für die Leistungsaufnahme des Prozessors eingestellt. Diese sind in Tabelle 4.4 angegeben.

Wie man sieht, wird der vorgegebene Verbrauch meist relativ genau eingehalten. Obwohl die Abschätzung des Energieverbrauchs der Prozesse allein auf der Rechenzeit basiert und außerdem für alle Prozesse eine einheitliche Verlustleistung angegeben wurde, liegen die Abweichungen doch immer unter 20 mW.

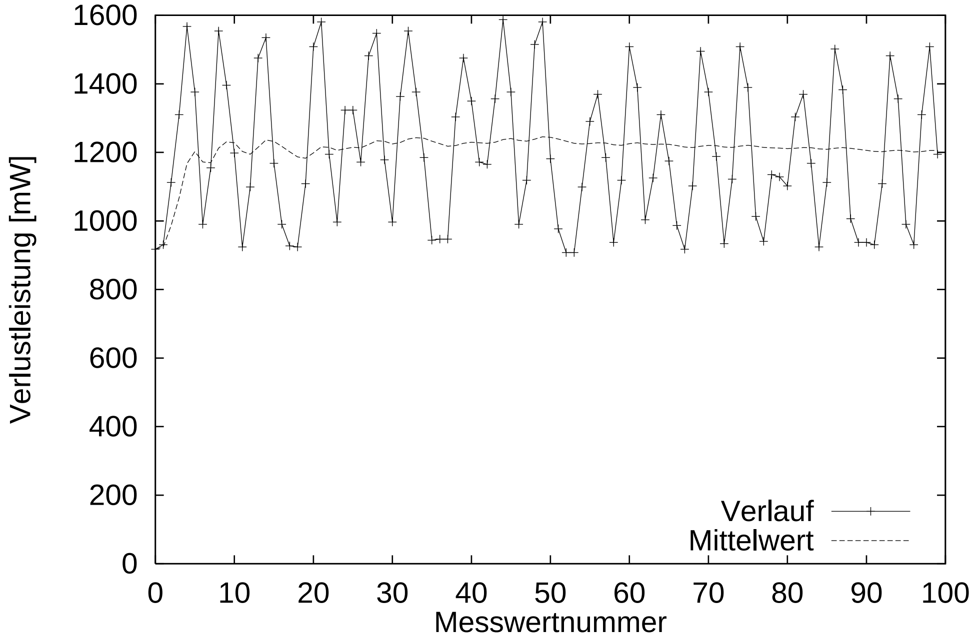

Abbildung 4.2.: Verlauf der Verlustleistung des find Prozesses bei Begrenzung auf 1200 mW

Zielverbrauch	find	gcc	gzip
1400 mW	1393,62 mW	1411,61 mW	1400,26 mW
1200 mW	1205,39 mW	1203,05 mW	1196,51 mW
1000 mW	989,18 mW	986,70 mW	990,69 mW
800 mW	807,87 mW	809,36 mW	806,09 mW
600 mW	592,88 mW	595,52 mW	589,22 mW
400 mW	406,36 mW	405,64 mW	395,11 mW
200 mW	196,42 mW	194,37 mW	181,50 mW
100 mW	97,68 mW	113,16 mW	93,42 mW

Tabelle 4.3.: mittlere Verlustleistung verschiedener Prozesse bei unterschiedlichem Zielverbrauch.

Geschwindigkeit	Verlustleistung
66 MHz	1575 mW
33 MHz	938 mW
16 MHz	501 mW
8 MHz	251 mW
4 MHz	159 mW
2 MHz	105 mW
1 MHz	66 mW

Tabelle 4.4.: Verlustleistung des Prozessors bei unterschiedlichen Geschwindigkeiten.

4.3.2. Ausführungszeit bei konstanter Spannung

In Tabelle 4.5 ist die Ausführungszeit verschiedener Programme bei unterschiedlichen Energievorgaben angegeben. Vor allem bei niedrigen Energieverbrauchsvorgaben ist ein überproportionaler Anstieg der Rechendauer zu erkennen. Dieser Anstieg dürfte der – bei niedrigen Taktraten stark sinkenden – Speicherzugriffsgeschwindigkeit (siehe 2.2.2) zuzuschreiben sein.

	find		gcc		gzip	
Zielverbrauch	Zeit	Energie	Zeit	Energie	Zeit	Energie
1400 mW	4,65 s	6510 mJ	5,48 s	7672 mJ	9,85 s	13790 mJ
1200 mW	5,07 s	6084 mJ	5,92 s	7104 mJ	11,29 s	13548 mJ
1000 mW	5,94 s	5940 mJ	6,66 s	6660 mJ	13,21 s	13210 mJ
800 mW	7,18 s	5744 mJ	8,22 s	6576 mJ	16,03 s	12824 mJ
600 mW	10,51 s	6306 mJ	11,69 s	7014 mJ	22,75 s	13650 mJ
400 mW	17,33 s	6932 mJ	19,89 s	7956 mJ	38,26 s	15304 mJ
200 mW	64,02 s	12804 mJ	78,55 s	15710 mJ	131,75 s	26350 mJ
100 mW	225,89 s	22589 mJ	306,21 s	30621 mJ	483,50 s	48350 mJ

Tabelle 4.5.: Rechenzeit und Energieaufnahme verschiedener Prozesse bei unterschiedlichem Zielverbrauch und konstanter Spannung.

Zusätzlich ist in der Tabelle der gesamte Energieverbrauch des Prozesses in Millijoule angegeben. Wie man sieht, sinken die Joule mit abnehmendem Zielverbrauch, bis zu einer Vorgabe von 800 mW. Eine weitere Verringerung des Zielverbrauchs ist unrentabel, da die Gesamtenergie des Prozesses wieder stark ansteigt.

Allerdings fallen die ermittelten Werte insofern zu optimistisch aus, als ausschließlich der Verbrauch der CPU berücksichtigt wurde. Wird auch der Verbrauch des Gesamtsystems berücksichtigt, so sind durch Taktfrequenzsenkung keine Einsparungen zu erreichen.

	find		gcc		gzip	
Zielverbrauch	Zeit	Energie	Zeit	Energie	Zeit	Energie
1000 mW	5,63 s	5630 mJ	5,03 s	5030 mJ	9,24 s	9240 mJ
800 mW	6,26 s	5008 mJ	5,76 s	4608 mJ	10,68 s	8544 mJ
600 mW	6,75 s	4050 mJ	6,56 s	3936 mJ	12,43 s	7458 mJ
400 mW	8,31 s	3324 mJ	7,82 s	3128 mJ	15,53 s	6212 mJ
200 mW	16,78 s	3356 mJ	16,32 s	3264 mJ	30,27 s	6054 mJ
100 mW	76,35 s	7635 mJ	66,47 s	6647 mJ	109,66 s	10966 mJ

Tabelle 4.6.: Rechenzeit und Energieaufnahme verschiedener Prozesse bei unterschiedlichem Zielverbrauch und variabler Spannung.

4.3.3. Ausführungszeit bei variabler Spannung

Die Tabelle 4.6 ist genauso aufgebaut wie die Tabelle aus dem letzten Abschnitt. Nun wurde jedoch angenommen, dass sich die Spannung mit der Taktfrequenz absenken lässt. Die daraus resultierenden Werte für die Verlustleistung bei verschiedenen Taktfrequenzen sind in Tabelle 4.7 dargestellt. Die Werte wurden durch das unter 2.3.1 angegebene Verfahren ermittelt.

Geschwindigkeit	Spannung	Leistungsaufnahme
66 MHz	2,7 V	1070 mW
33 MHz	2,25 V	456 mW
16 MHz	2,03 V	215 mW
8 MHz	1,92 V	111 mW
4 MHz	1,87 V	78 mW
2 MHz	1,84 V	60 mW
1 MHz	1,825 V	48 mW

Tabelle 4.7.: Verlustleistung bei variabler Spannung

Wie man sieht ergibt sich bei einer Spannungssenkung ein deutlich niedrigerer Energieverbrauch. Lediglich eine Drosselung der Leistungsaufnahme auf 100 mW ist nicht mehr rentabel.

Dennoch ist bei Berücksichtigung des Energieverbrauchs des Gesamtsystems auch hier keine Energieeinsparung zu erreichen, so lange der Verbrauch der anderen Komponenten (2016 mW) nicht ebenfalls gesenkt wird. Allerdings wurden die Auswirkungen auf die Batteriekapazität nicht berücksichtigt.

4.3.4. Nebenwirkungen und Probleme bei Taktfrequenzänderungen

Bei der Senkung der Taktfrequenz treten immer dann Probleme auf, wenn Teile der Software sich auf eine konstante Ausführungszeit von Befehlen verlassen.

So existiert im Linux-System beispielsweise die Funktion `nanosleep`, die die Ausführung des aktuellen Prozesses für eine bestimmte Anzahl von Nanosekunden unterbricht. Wird durch den Prozessor keine andere Möglichkeit zur Verfügung gestellt, so wird die Anzahl der verstrichenen Nanosekunden durch die Anzahl von Durchläufen einer leeren Schleife bestimmt. Beim Systemstart wird ermittelt, wie lange ein Schleifendurchlauf durchschnittlich benötigt, und das Resultat wird dann zur Zeitbestimmung verwendet. Da die Anzahl dieser Schleifendurchläufe pro Nanosekunde jedoch direkt von der Taktfrequenz abhängig ist, wird ein Aufruf von `nanosleep` bei niedrigeren Taktfrequenzen eine wesentlich höhere Zeit lang warten, als angegeben.

Auch bei Timeouts entstehen Probleme: Bei Zugriffen auf ein entferntes Dateisystem über NFS wird der Zugriff beispielsweise abgebrochen, wenn innerhalb einer bestimmten Zeitspanne keine Antwort vom Server erhalten wurde. Das führte im vorliegenden System dazu, dass bei Geschwindigkeiten unter 16 MHz *alle* NFS-Zugriffe abgebrochen werden, da der Timeout abläuft, bevor die Antwort des Servers empfangen bzw. bearbeitet wurde.

Allgemein gibt es also immer dann Probleme, wenn Entscheidungen auf Grund vorher bestimmter Ausführungszeiten getroffen werden.

5. Zusammenfassung

Ziel dieser Arbeit war es, eine Energieverwaltung zu implementieren, bei der die Geschwindigkeit durch die Angabe eines Verbrauchslimits bzw. einer Batterielebensdauer durch den Benutzer bestimmt wird. Außerdem sollte die Geschwindigkeit prozessbasiert festgelegt werden.

Wie in Kapitel zwei erläutert, hängt der Energieverbrauch eines Prozessors und die Effizienzsteigerung bei Taktfrequenzsenkung von einer Vielzahl unterschiedlicher Faktoren ab. Zum einen spielt die verwendete Hardware eine Rolle, insbesondere die Art der verwendeten Batterien, der Energieverbrauch des Gesamtsystems und vor allem die Möglichkeit auch die Spannung abzusenken.

Andererseits wird der Energieverbrauch auch von der ausgeführten Software beeinflusst. Eine gewisse Rolle spielen die Art des ausgeführten Befehls und sogar der Inhalt der Operandenregister. Als besonders wichtige Größe erscheint jedoch die Zahl der Hauptspeicherzugriffe: Zum einen wird die Ausführungsgeschwindigkeit im hohen Maße von der Anzahl der Speicherzugriffe bestimmt, zum anderen haben die Messungen am AMD Èlan SC410 ergeben, dass auch die Verlustleistung von der Zahl der Speicherzugriffe abhängt. Somit scheint vor allem die Verlangsamung von Prozessen mit hohen Speicherzugriffsraten ein großes Einsparungspotential zu bieten.

Insgesamt hat sich jedoch ergeben, dass eine Taktfrequenzsenkung häufig nur dann zu einer Erhöhung der möglichen Operationen pro Batterieentladung führt, wenn auch die Spannung gesenkt werden kann. Ziel dieser Arbeit war es jedoch weniger eine möglichst hohe Anzahl von Operationen pro Batterieentladung zu erreichen, es ging vielmehr darum, ein vorgegebenes Verbrauchslimit bzw. eine vorgegebene Batterielebensdauer einzuhalten.

In Kapitel drei wurde gezeigt, dass es auch zur Einhaltung einer vorgegebenen Batterielebensdauer genügt, Betriebssystemmechanismen zur Einhaltung eines Verbrauchslimits zur Verfügung zu stellen. Außerdem wurde ein Verfahren beschrieben, das die Berechnung der Geschwindigkeiten auf zwei Komponenten aufteilt: Zum einen die Regelung des Energieverbrauchs, d. h. die Berechnung eines kurzfristigen Zielverbrauchs aus Soll- und Ist-Wert der verbrauchten Energie. Zum anderen die Verteilung der vorhanden Energie auf die verschiedenen Prozesse.

Die Aufgabe, jedem Prozess eine Geschwindigkeit so zuzuordnen, dass sich im Mittel nach kurzer Zeit der vorgegebene Energieverbrauch ergibt, ist äußerst komplex. Da diese Aufgabe jedoch sehr häufig gelöst werden muss (immer dann, wenn sich der Zielverbrauch oder die Anzahl der lauffähigen Prozesse ändert), wurde in dieser Arbeit eine Strategie implementiert, die alle Prozesse in zwei Klassen aufteilt. Eine Klasse, deren Prozesse möglichst schnell ausgeführt wird, für kurze interaktive Prozesse und eine Klasse deren Prozesse entsprechend langsamer ausgeführt werden, für langlaufende Hintergrundprozesse.

Zur Regelung der Verlustleistung werden natürlich Kenntnisse über den Energieverbrauch der Prozesse benötigt. Die in Kapitel drei dargestellten Messungen haben jedoch gezeigt, dass sich der Energieverbrauch verschiedener Prozesse – zumindest beim AMD Èlan SC410 – nur in geringem Maße unterscheidet. Dadurch ergibt auch eine Regelung, die sich nur auf die Rechenzeit als Maß für den Energieverbrauch stützt, ein relativ gutes Ergebnis, wie in Kapitel vier dargestellt.

Allerdings hat sich gezeigt, dass eine Senkung des Energieverbrauchs immer mit einer noch stärkeren Verminderung der Rechenleistung verbunden ist, so dass die (bei Vollauslastung) pro Batterieentladung geleistete Rechenarbeit nicht steigt, sondern sinkt. Die in Kapitel vier angeführten Berechnungen mit dem Energieverbrauchsmodell bei variabler Spannung (siehe Abschnitt 2.3) deuten darauf hin, dass auch im Falle einer dynamischen Anpassung der Versorgungsspannung an die Taktfrequenz echte Einsparungen nur dann möglich sind, wenn auch der Energieverbrauch anderer Komponenten des Systems reduziert werden kann.

A. Durchführung der Messungen

Zur Messung der Verlustleistung des Prozessors wurde ein Conrad M3860-M Digital-Multimeter verwendet. Gemessen wurde der Stromfluss an der Stromversorgung der Hauptplatine. An Hand der Werte aus dem Datenblatt des Prozessors und der gemessenen Werte wurde der Stromverbrauch der anderen Komponenten auf 611 mA bzw. 2016 mW festgelegt. Die gemessenen Werte wurden über die serielle Schnittstelle an einem anderen Rechner eingelesen und – wenn nicht anders angegeben – der Mittelwert über 25 Messwerte gebildet.

Zur Messung des Energieverbrauchs einzelner Maschinenbefehle wurde jeweils ein Benchmark-Programm erstellt, das den entsprechenden Befehl in einer Schleife ausführt. Innerhalb einer Schleife wurden jeweils zwischen 1024 und 4096 Befehle ausgeführt. Die Länge einer Schleife wurde immer so gewählt, dass die gesamte Schleife in den 8kB Cache des AMD Èlan SC410 passt.

Vor Ausführung der Schleife wurden die in der Kommandozeile angegebenen Parameter in die Operandenregister geladen.

Als Beispiel ist das zur Messung des move-Befehls verwendete Programm im folgenden wiedergegeben. Die `asm` Anweisung bindet Assembler-Code in das C-Programm ein.

```
int main(int argc, char *argv[]) {
  unsigned long int wert = 0;
  if (argc > 1)   {
    wert = strtoul(argv[1], NULL, 0);
  }

__asm__ __volatile__ ("": :"b" (wert)); /* Register ebx laden */
  while(1) {
    __asm__ __volatile__ (
        "movl %%ebx, %%eax\n\t"
        "movl %%ebx, %%eax\n\t"
        (...)
        "movl %%ebx, %%eax\n\t" : : : "%eax");
  }
  return 0;
}
```

Zur Messung der Ausführungszeit wurde zusätzlich vor der Schleife die Uhrzeit bestimmt und nach jedem Schleifendurchlauf eine Variable hochgezählt. Nach einer bestimmten Anzahl an Schleifendurchläufen wurde erneut die Uhrzeit bestimmt um die Ausführungsdauer zu bestimmen.

Literaturverzeichnis

[AMDa] AMD: *AMD Élan SC400 and Élan SC410 Micorcontrollers Data sheet.* Order Number 21028.

[AMDb] AMD: *AMD Élan SC400 and Élan SC410 Micorcontrollers User's Manual.* Order Number 21030.

[AMD96] AMD: *AMD Élan SC400 Microcontroller Register Set Reference*, Dezember 1996. Order Number 21032.

[Bel99] Bellosa, F.: *EndurIX: OS-Directed Throttling of Processor Activity for Dynamic Power Management.* Technischer BerichtTR-I4-99-3, Universität Erlangen-Nürnberg, IMMD 4, Juni 1999.

[GCW95] Govil, K., E. Chan und H. Wassermann: *Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU.* In: *Proceedings of the 1st Conference on Mobile Computing and Networking MOBICOM'95*, März 1995.

[HP94] Hennessy, J. L. und D. A. Patterson: *Rechnerarchitektur.* Lehrbuch Informatik. Vieweg, Braunschweig/Wiesbaden, 1. Auflage, 1994.

[IMT99] Intel, Microsoft und Toshiba: *Advanced Configuration and Power Interface Specification 1.0b*, Februar 1999.

[Int97] Intel: *Intel Architecture Software Developer's Manual – Volume 2: Instruction Set Reference*, 1997. Order Number 24319101.

[Int99] Intel: *Intel StrongARM SA-1100 Microprocessor Developer's Manual*, April 1999.

[Int00a] Intel: *Intel SpeedStep Technology*, Januar 2000.

[Int00b] Intel: *Mobile Pentium(R) III Processor Datasheet*, 2000. Order Number 24548302.

[JUM97] JUMPtec: *DIMM-PC/486-I Technical Manual Rev.1.0*, 1997.

[LF93] Liebig, H. und T. Flik: *Rechnerorganisation.* Springer-Verlag, Berlin Heidelberg New York, 2. Auflage, 1993.

[Mar99] Martin, Thomas L.: *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing.* Doktorarbeit, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1999.

[Mot98] Motorola: *MPC860 PowerQUICC User's Manual Rev.1*, September 1998.

[MS96] Martin, T. und D. Siewiorek: *A Power Metric for Mobile Systems.* In: *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'96*, 1996.

[MS99] Martin, T. und D. Siewiorek: *The impact of battery capacity and memory bandwidth on CPU speed-setting: a case study.* In: *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'99*, August 1999.

[PB98] Pering, T. und R. Broderson: *Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System.* In: *Proceedings of the International Symposium on Computer Architecture ISCA'98*, Juni 1998.

[Tan92] Tanenbaum, Andrew S.: *Modern Operating Systems.* Prentice-Hall International, London, 1992.

[Tra00] Transmeta: *The Technology behind Crusoe Processors*, Januar 2000.

[WWDS94] Weiser, M., B. Welch, A. Demers und S. Shenker: *Scheduling for Reduced CPU Energy.* In: *Proceedings of the First Symposium on Operating System Design and Implementation OSDI'94*, November 1994.