Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Seminararbeit

Effiziente Programmierung: Energieeffizienz

vorgelegt von

Willy Kha

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang:	Software - System - Entwicklung
Matrikelnummer:	6824378
Betreuer:	Yevhen Alforov

Hamburg, 2018-03-26

Abstract

Diese Ausarbeitung bietet einen Einblick in die Energieeffizienz von Computern. Es beginnt mit einer Einleitung in der die Wichtigkeit der Energieeffizienz und der Leistungsvergleich beschrieben wird. Zudem werden Methoden zu Messung des Energieverbrauches vorgestellt. Danach folgen Methoden zur Energieeinsparung, wobei insbesondere das "Advanced Configuration and Power Interface" beschrieben wird. Zusätzlich werden Programmiertechniken erläutert, die die Energieeffizienz der Software verbessern können.

Contents

1 Einleitung

1.1 Motivation

Energieeffizienz wird in der heutigen Zeit immer wichtiger. Vorallem durch Umweltkatastrophen und Reaktorunfälle gelangt mehr Aufmerksamkeit auf Energieeffizientere Geräte und regenerative Energie. Vorallem Hochleistungsrechner verbrauchen sehr viel Energie. Der Leistungsstärkste Hochleistungsrechner "Sunway TaihuLight"(Stand November 2017)[1] verbraucht unter Vollast ungefähr 15 Megawatt. Dies wäre genug um ungefähr 1000 Haushalte zu versorgen. Jedoch spielt die Energieeffizienz auch für Privatpersonen eine große Rolle, da für die heutige Gesellschaft vorallem die mobile Nutzung von Computer wichtig ist. So profitieren auch geräte mit beschränkter Energier wie z.B. Smartphones und Notebooks von energieeffizienterer Hard- und Software.

1.2 TOP500 und Green500

Von 1986 bis 1992 veröffentlichte Hans Meuer jährlich eine Statistik in der die leistungsstärksten Hochleistungsrechner aufgeführt waren. Jedoch wurde die Leistung jeweils durch Herstellerangaben verglichen. Um einen besseren Vergleich zu schaffen wurde 1993 erstmals eine Liste der TOP500 veröffentlicht. Im Gegensatz zur Statistik von Meurer wird die Liste der TOP500 jedes halbe Jahr aktualisiert. Außerdem nutzt die TOP500 Liste das LINPACK Benchmark um die Leistung der Hochleistungsrechner zu ermitteln.[2] Die ermittelte Leistung wird in "Floating Point Operations Per Second"(FLOPS) angegeben. Diese Einheit gibt an wie viele Gleitkommaoperationen ein System pro Sekunde verarbeiten kann.
2007 wurde die erste Green500 Liste veröffentlicht. In dieser Liste werden die effizientesten Hochleistungsrechner aufgezählt. Dadurch wurde mehr Aufmerksamkeit bezüglich der Energieeffizienz von Hochleistungsrechnern geschaffen.

[1] `https://www.top500.org/system/178764`
[2] `https://www.top500.org/project/linpack/`

2 Messung des Energieverbrauches

Das Ziel dieses Kapitels ist es einen groben Einblick in die Energieverbrauchsmessung eines Systemes zu gewähren. Dazu werden verschiedene Messmethoden untersucht und verglichen. Um jedoch eine Grundlage für das Verständnis der Messungen zu gewähren beginnt dieses Kapitel mit einem Unterkapitel, in dem die Metriken und Grundlagen erklärt werden.

2.1 Motivation und Grundlagen

Der Energieverbrauch wird grundsätzlich immer in Watt angegeben und gemessen. Außerdem wird meistens versucht den Verbrauch von einzelnen Systemkomponenten zu messen anstatt den Verbrauch des gesamten Systems um eine detailiertere Analyse zu ermöglichen. Zudem sind kleine Energieverbrauchsveränderungen nicht direkt am Netzteil messbar. Durch den Energieverbrauch alleine kann jedoch nicht auf die Energieeffizienz eines Systemes geschlossen werden. Um die Energieeffizienz eines Systemes zu berechnen benötigt man zusätzlich die Leistung des Systems. Diese kann beispielsweise mithilfe der Einheit "FLOPS" angegeben werden.

2.2 PowerPack

Das PowerPack besteht aus einem Messgerät und der zugehörigen Software. Mit dem PowerPack ist es möglich den Energieverbrauch von Clusterknoten zu messen. Dabei kann zwischen bestimmten Komponenten unterschieden werden. Dazu wird das Messgerät zwischen dem Netzteil und den Komponenten angeschlossen. Mithilfe eines zusätzlichen Rechners und der zugehörigen Software können die Messergebnisse gesammelt werden.[1]

2.3 pmlib

Pmlib ist ein Framework zum Messen des Energieverbrauches einzelner Programmteile in Hochleistungsrechnern. Durch eine hohe Kompatibilität mit verschiedenen Messgeräten deckt pmlib eine große Reichweite an Messmöglichkeiten ab. Zusätzlich ist es möglich die P- und C-States des Hochleistungsrechners auszulesen. Diese geben Auskunft über die Leistung und den Energieverbrauch der Prozessoren. [2]

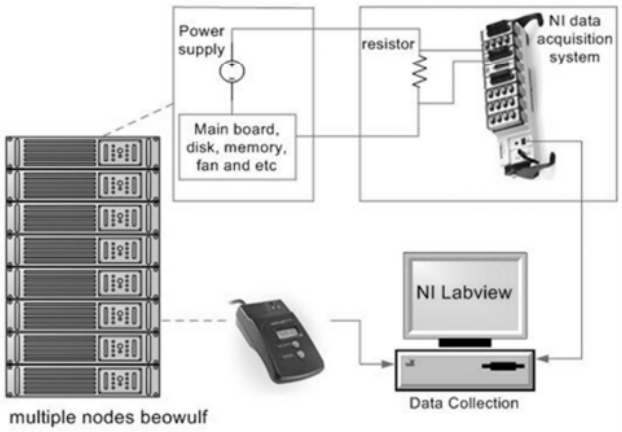

Figure 2.1: Der Aufbau einer PowerPack-Energiemessung

2.4 Powermon2

Das Powermon2 ist ein Messgerät welches zwischen Mainboard und Netzteil eingebaut wird. Mittels eines Usb-Ausgangs werden dann bis zu 3000 Messungen pro Sekunde übertragen. Das Messgerät misst die Gleichspannung, die vom Mainboard verbraucht wird und kann dabei zwischen den unterschiedlichen "Strombahnen" unterscheiden. Dadurch sind genauere Analysen über bestimmte Auswirkungen während Tests möglich.[3]

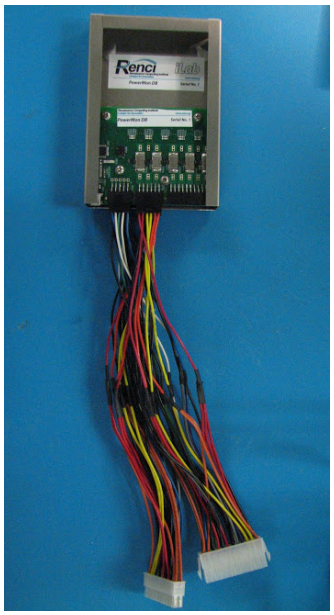

Figure 2.2: Ein Powermon Messgerät

2.5 Intel Running Average Power Limit

Intel Running Average Power Limit ist ein Feature moderner Intel Prozessoren, die die Messung des Energieverbrauch der Prozessoren erleichtert. Mithilfe von Intel Running Average Power Limit lässt sich der Energieverbrauch bestimmter Codeabschnitte leicht messen. Nach Intels Angaben lässt sich die Messung bis auf die Millisekunde genau nachvollziehen. Die Messwerte werden in bestimmten Prozessorregistern gespeichert. Da dies für Programmieranfänger schwierig sein kann empfielt es sich auf Frameworks wie z.B. jRAPL[1] zurückzugreifen. [4]

[1] `https://github.com/kliu20/jRAPL`

3 Methoden zur Energieeinsparung

Die im folgenden vorgestellten Methoden sind in modernen Betriebssystemen bereits voreingestellt vorhanden und funktionieren ohne, dass der Benutzer diese Methoden aktiv anwenden muss.

3.1 Dynamic Voltage and Frequency Scaling

Hierbei handelt es sich um eine Funktion bestimmter Computerteile. Dadurch kann die Spannung und die Frequenz der bestimmter Komponenten dynamisch vom Betriebssystem bestimmt werden. Unter anderem wird "Dynamic Voltage and Frequency Scaling" vom Prozessor, Arbeitsspeicher und von PCI-Express Geräten verwendet.
Benötigt das Betriebssystem weniger Leistung so kann es die Spannung und die Frequenz bestimmter Komponenten verringern. Dadurch verringert sich auch die Hitzeentwicklung der Komponente, wodurch weniger Energie zur Kühlung der Komponente benötigt wird. Natürlich kann das Betriebssystem auch die Spannung und die Frequenz erhöhen. [5]

3.2 Advanced Configuration and Power Interface

ACPI ist eine Schnittstelle moderner Mainboards. Es ermöglicht dem Betriebssystem die Steuerung über das Mainboard. Dadurch können komplexere Energiesparpläne, die aufgrund ihrer größe normalerweise nicht im BIOS implementiert werden können stattdessen im Betriebssystem implementiert werden. Durch das spezifieren von mehreren Systemzuständen können bestimmte Systemteile teilweise oder komplett abgeschaltet werden. [1]

[1] http://www.uefi.org/sites/default/files/resources/ACPI_6.0.pdf

3.2.1 Globale Zustände

Zustand	Beschreibung
G0 Working	Der Computer ist eingeschaltet und voll Funktionsfähig. Der Benutzer kann Programme öffnen und schließen.
G1 Sleeping	Der Computer ist in einem Stand-By Modus. Es werden keine Programme ausgeführt. In diesem Zustand können Programme pausiert werden um diese beim wechsel in den "G0 Working Zustand" wieder fortzusetzen. Das System lässt sich schnell wieder in den "G0 Working" Zustand versetzen ohne das Betriebssystem neuzustarten. Dieser Zustand lässt sich wieder in 5 unterzustände spezifizieren.
G2 Soft Off	Der Computer ist ausgeschaltet und speichert in diesem Zustand keine Programmzustände. Um von diesem Zustand in den "G0 Working" Zustand zu wechseln. In diesem Zustand kann Strom durch die einzelnen Komponenten fließen.
G3 Mechanical Off	Der Computer ist vom Stromnetz getrennt. Es fließt kein Strom durch die Komponenten.

3.2.2 Schlaf Zustand

Diese Zustände sind speziellere Zustände von "G1 Sleeping".

Zustand	Beschreibung
S1	Der Computer kann schnell wieder in den "G0 Working" Zustand wechseln. Temporäre Daten bleiben erhalten.
S2	Der Computer kann auch hier wieder schnell in den "G0 Working" Zustand wechseln. Jedoch wird in diesem Zustand Teile des Prozessors abgeschaltet, wodurch einige temporären Daten nicht erhalten bleiben.
S3	Der Computer kann auch in diesem Zustand wieder schnell in den "G0 Working" Zustand wechseln. In diesem Zustand werden zusätzlich Teile des Mainboards abgeschaltet.
S4	Der Computer spart in diesem mehr Energie braucht jedoch auch länger um wieder in den "G0 Working" Zustand zu wechseln. In diesem Zustand werden zusätzlich alle Geräte ausgeschaltet.
S5 Soft-Off	Der Computer spart mehr Energie dadurch dass es weniger Daten über den Betriebszustand speichert. Das hat zur Folge, dass die Programme, die während des Betrieb aktiv waren neu gestartet werden müssen.

3.2.3 Geräte Zustände

Diese Zustände werden genutzt um den Energieverbrauch von Geräten zu spezifizieren, die am Computer angeschlossen sind. Dabei kann es sich zum Beispiel um USB-Geräte oder Monitore handeln. Wieviel Energie in den einzelnen Zuständen gespart wird kann durch den Hersteller bestimmt werden. Viele Geräte verfügen nicht über die D1 und D2 Zustände.

Zustand	**Beschreibung**
D0	Das Gerät ist voll Funktionsfähig und benutzbar
D1	Das Gerät bietet einige Funktionen an und verbraucht weniger Energie als im Zustand D0.
D2	Das Gerät bietet weniger Funktionen als im Zustand D1 an und verbraucht demnach auch weniger Energie als im Zustand D1.
D3 hot	Das Gerät bietet keine Funktion. In diesem Zustand soll die Energieeinsparung am höchsten sein und trotzdem vom Computer erkannt werden
D3 cold	Das Gerät ist vom Computer getrennt und dadurch nicht ansprechbar. Außerdem fließt kein Strom durch das Gerät.

3.2.4 Prozessor Zustände

Diese Zustände sind von Prozessor zu Prozessor unterschiedlich. Dabei gilt jedoch, dass im Zustand P0 der Prozessor uneingeschränkt arbeiten kann. Erhöht sich die Zahl des Zustandes, so wird immer mehr Energie eingespart, wodurch die Leistung des Prozessors abnimmt. Die Anzahl an Zuständen ist dem Hersteller überlassen.

4 Energiesparendes Programmieren

Dieses Kapitel erläutert verschiedene Möglichkeiten, die beim Programmieren beachtet werden sollten um energiesparende Software zu entwickeln.

4.1 Wahl der Programmiersprache

Bei der Wahl der Programmiersprache ist zunächst festzuhalten, dass es keine optimale Programmiersprache gibt, die am wenigsten Energie verbraucht. Es muss stattdessen darauf geachtet werden was das endgültige Programm leisten soll.
In einem Versuch der "Universidade do Minho", bei dem 27 verschiedene Programmiersprachen und ihre Lösung auf 10 verschiedene Probleme untersucht wurden, zeigte sich jedoch, dass Programmiersprachen die zunächst compiliert werden müssen um ausgeführt zu werden im Durchschnitt besser abschneiden als Programmiersprachen, welche direkt ausgeführt werden können.[6]

binary-trees				
	Energy	Time	Ratio	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust $\Downarrow_2$	49.07	1263	0.039	180
(c) Fortran $\Uparrow_1$	69.82	2112	0.033	133
(c) Ada $\Downarrow_1$	95.02	2822	0.034	197
(c) Ocaml $\downarrow_1 \Uparrow_2$	100.74	3525	0.029	148
(v) Java $\uparrow_1 \Downarrow_{16}$	111.84	3306	0.034	1120
(v) Lisp $\downarrow_3 \Downarrow_3$	149.55	10570	0.014	373
(v) Racket $\downarrow_4 \Downarrow_6$	155.81	11261	0.014	467
(i) Hack $\uparrow_2 \Downarrow_9$	156.71	4497	0.035	502
(v) C# $\downarrow_1 \Downarrow_1$	189.74	10797	0.018	427
(v) F# $\downarrow_3 \Downarrow_1$	207.13	15637	0.013	432
(c) Pascal $\downarrow_3 \Uparrow_5$	214.64	16079	0.013	256
(c) Chapel $\uparrow_5 \Uparrow_4$	237.29	7265	0.033	335
(v) Erlang $\uparrow_5 \Uparrow_1$	266.14	7327	0.036	433
(c) Haskell $\uparrow_2 \Downarrow_2$	270.15	11582	0.023	494
(i) Dart $\downarrow_1 \Uparrow_1$	290.27	17197	0.017	475
(i) JavaScript $\downarrow_2 \Downarrow_4$	312.14	21349	0.015	916
(i) TypeScript $\downarrow_2 \Downarrow_2$	315.10	21686	0.015	915
(c) Go $\uparrow_3 \Uparrow_{13}$	636.71	16292	0.039	228
(i) Jruby $\uparrow_2 \Downarrow_3$	720.53	19276	0.037	1671
(i) Ruby $\Uparrow_5$	855.12	26634	0.032	482
(i) PHP $\Uparrow_3$	1,397.51	42316	0.033	786
(i) Python $\Uparrow_{15}$	1,793.46	45003	0.040	275
(i) Lua $\downarrow_1$	2,452.04	209217	0.012	1961
(i) Perl $\uparrow_1$	3,542.20	96097	0.037	2148
(c) Swift	n.e.			

fannkuch-redux				
	Energy	Time	Ratio	Mb
(c) C $\Downarrow_2$	215.92	6076	0.036	2
(c) C++ $\Uparrow_1$	219.89	6123	0.036	1
(c) Rust $\Downarrow_{11}$	238.30	6628	0.036	16
(c) Swift $\Downarrow_5$	243.81	6712	0.036	7
(c) Ada $\Downarrow_2$	264.98	7351	0.036	4
(c) Ocaml $\downarrow_1$	277.27	7895	0.035	3
(c) Chapel $\uparrow_1 \Downarrow_{18}$	285.39	7853	0.036	53
(v) Lisp $\downarrow_3 \Downarrow_{15}$	309.02	9154	0.034	43
(v) Java $\uparrow_1 \Downarrow_{13}$	311.38	8241	0.038	35
(c) Fortran $\Downarrow_1$	316.50	8665	0.037	12
(c) Go $\uparrow_2 \Uparrow_7$	318.51	8487	0.038	2
(c) Pascal $\Uparrow_{10}$	343.55	9807	0.035	2
(v) F# $\downarrow_1 \Downarrow_7$	395.03	10950	0.036	34
(v) C# $\uparrow_1 \Downarrow_5$	399.33	10840	0.037	29
(i) JavaScript $\downarrow_1 \Downarrow_2$	413.90	33663	0.012	26
(c) Haskell $\uparrow_1 \Uparrow_8$	433.68	14666	0.030	7
(i) Dart $\Downarrow_7$	487.29	38678	0.013	46
(v) Racket $\Uparrow_3$	1,941.53	43680	0.044	18
(v) Erlang $\Uparrow_3$	4,148.38	101839	0.041	18
(i) Hack $\Downarrow_6$	5,286.77	115490	0.046	119
(i) PHP	5,731.88	125975	0.046	34
(i) TypeScript $\downarrow_4 \Uparrow_4$	6,898.48	516541	0.013	26
(i) Jruby $\uparrow_1 \Downarrow_4$	7,819.03	219148	0.036	669
(i) Lua $\downarrow_3 \Uparrow_{19}$	8,277.87	635023	0.013	2
(i) Perl $\uparrow_2 \Uparrow_{12}$	11,133.49	249418	0.045	12
(i) Python $\uparrow_2 \Uparrow_{14}$	12,784.09	279544	0.046	12
(i) Ruby $\uparrow_2 \Uparrow_{17}$	14,064.98	315583	0.045	8

fasta				
	Energy	Time	Ratio	Mb
(c) Rust $\Downarrow_9$	26.15	931	0.028	16
(c) Fortran $\downarrow_6$	27.62	1661	0.017	1
(c) C $\uparrow_1 \Downarrow_1$	27.64	973	0.028	3
(c) C++ $\uparrow_1 \Downarrow_2$	34.88	1164	0.030	4
(v) Java $\uparrow_1 \Downarrow_{12}$	35.86	1249	0.029	41
(c) Swift $\Downarrow_9$	37.06	1405	0.026	31
(c) Go $\downarrow_2$	40.45	1838	0.022	4
(c) Ada $\downarrow_2 \Uparrow_3$	40.45	2765	0.015	3
(c) Ocaml $\downarrow_2 \Downarrow_{15}$	40.78	3171	0.013	201
(c) Chapel $\uparrow_5 \Downarrow_{10}$	40.88	1379	0.030	53
(v) C# $\uparrow_4 \Downarrow_5$	45.35	1549	0.029	35
(i) Dart $\Downarrow_6$	63.61	4787	0.013	49
(i) JavaScript $\Downarrow_1$	64.84	5098	0.013	30
(c) Pascal $\downarrow_1 \Uparrow_{13}$	68.63	5478	0.013	0
(i) TypeScript $\downarrow_2 \Downarrow_{10}$	82.72	6909	0.012	271
(v) F# $\uparrow_2 \Uparrow_3$	93.11	5360	0.017	27
(v) Racket $\downarrow_1 \Uparrow_5$	120.90	8255	0.015	21
(c) Haskell $\uparrow_2 \Downarrow_8$	205.52	5728	0.036	446
(v) Lisp $\Downarrow_2$	231.49	15763	0.015	75
(i) Hack $\Downarrow_3$	237.70	17203	0.014	120
(i) Lua $\Uparrow_{18}$	347.37	24617	0.014	3
(i) PHP $\downarrow_1 \Uparrow_{13}$	430.73	29508	0.015	14
(v) Erlang $\uparrow_1 \Uparrow_{12}$	477.81	27852	0.017	18
(i) Ruby $\downarrow_1 \Uparrow_2$	852.30	61216	0.014	104
(i) JRuby $\uparrow_1 \Downarrow_2$	912.93	49509	0.018	705
(i) Python $\downarrow_1 \Uparrow_{18}$	1,061.41	74111	0.014	9
(i) Perl $\uparrow_1 \Uparrow_8$	2,684.33	61463	0.044	53

Figure 4.1: Einige Testergebnisse der Universidade do Minho.
Die dünnen Pfeile zeigen die Veränderung der Platzierung an wenn nach Laufzeit sortiert werden würde.
Die dicken Pfeile zeigen die Veränderung der Platzierung an wenn nach maximalen Speicherverbrauch während der Laufzeit sortiert werden würde.

4.2 Multithreading

Um eine bessere Energieeffizienz zu erreichen sollte Multithreading benutzt werden. Das Ziel hierbei ist eine schnellere Ausführzeit zu erreichen, welches zu einem geringeren Energieverbrauch führen würde. Dabei sollte darauf geachtet werden dass die Auslastungen der Threads ähnlich sind. [7]

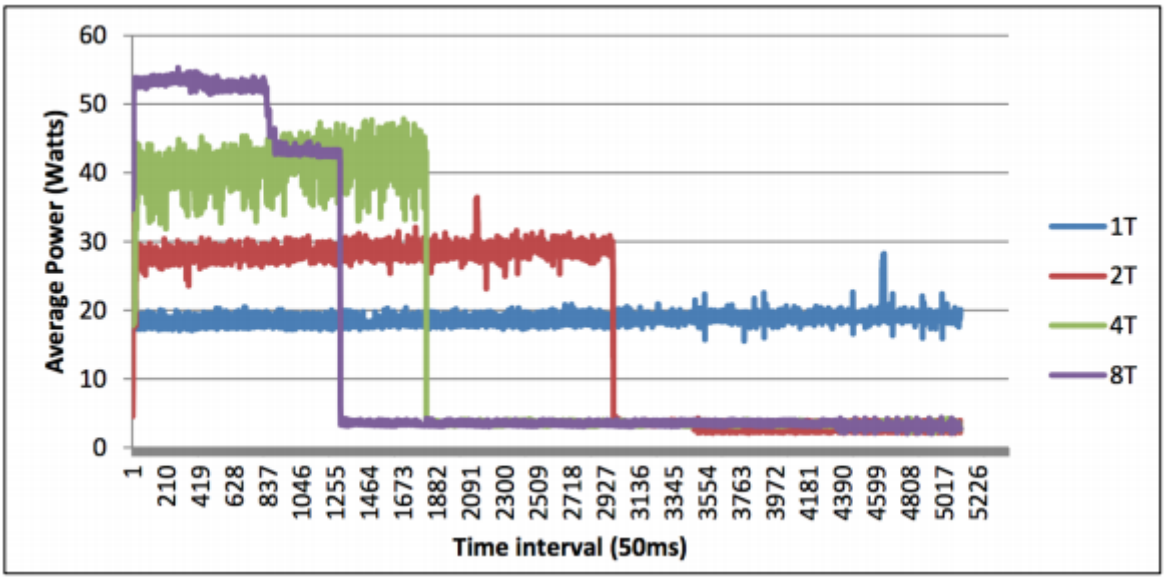

Figure 4.2: Die Auswirkung von Multithreading auf ein Programm

4.3 Wahl der Algorithmen

Bei der Wahl der Algorithmen ist es meistens die beste Entscheidung sich für den schnellsten Algorithmus zu entscheiden. Da der schnellste Algorithmus sich von Problem zu Problem unterscheidet sollte der Programmierer bekannte Lösungen seines Problems erforschen.[7]

4.4 Dateneffizienz

Durch das einsparen von unnötigen Datenbewegungen kann eine höhere Energieeffizienz erreicht werden. "Prefetching" bezeichnet das teilweise vorladen von Dateien um eine schnellere Zugriffszeit zu garantieren, falls die Datei gebraucht wird. Jedoch können hierdurch unnötige Datenzugriffe entstehen wodurch die Energieeffizienz sinkt.
Außerdem sollte beachtet werden, dass Zugriffe auf den Arbeitsspeicher deutlich mehr Energie benötigen als Zugriffe auf den Cache des Prozessors. Ein Zugriff auf den Arbeitsspeicher verbraucht demnach 40 Mal soviel Energie wie ein Zugriff auf den Cache

des Prozessors. Daraus folgt, dass die Zugriffe auf den Arbeitsspeicher so gering wie möglich gehalten werden sollten. [8]

4.5 Compiler Optimierung

Sollte eine Programmiersprache ausgewählt, welche zuerst Compiliert werden muss um ausgeführt zu werden, so könnte der genutzte Compiler eine Optimierungsoption haben. Durch diese Optimierungsoptionen könnten beispielsweise Schleifen durch "Loop unrolling" optimiert werden. Dies würde zu einer kürzeren Laufzeit und dadurch zu einer energieeffizienteren Software führen.

5 Schlussfolgerung

Die wichtiger der Energieeffizienz steigt durch die Wandlung der Gesellschaft zum "mobile computing" täglich an. Aus diesem Grund sollten sowohl Hardware als auch Softwareentwickler sich mit diesem Thema auseinandersetzen. Vor allem Software, die täglich benutzt wird sollte auf jedenfall so energieeffizient wie möglich programmiert sein. Mit ACPI und DVFS existieren bereits Industriestandards, welche eine hervorragende Basis für energieeffizientere Systeme bilden. Technologien wie Intels RAPL ermöglichen es außerdem ein einfacheres Testen der eigenen Software. Durch die Green500 entsteht ein Wettkampf um das energieeffizienteste System, welches in Zukunft hoffentlich mehr Aufmerksamkeit erlangt.

Bibliography

[1] X. Feng, "Power and energy profiling of scientific applications on distributed systems," 2008.

[2] S. Barrachina, "An integrated framework for power-performance analysis of parallel scientific workloads," *ENERGY 2013 : The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, 2013.

[3] D. Berdard, "Powermon 2: Fine-grained, integrated power measurement," *RENCI Technical Report TR-09-04*, 2009.

[4] M. Hähnel, "Measuring energy consumption for short code paths using rapl," 2012.

[5] K. Funaoka, "Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors."

[6] R. Pereira, "Energy efficiency across programming languages how does energy, time, and memory relate?," 2017.

[7] P. Larsson, "Energy-efficient software guidelines," *Intel Software Solution Group*, 2008.

[8] F. Goekkus, "Energy efficient programming," 2013.