

Oliver Arnold

Dynamisches Task-Scheduling in heterogenen
Mehrprozessorsystemen

Beiträge aus der Informationstechnik

Mobile Nachrichtenübertragung

Nr. 65

Oliver Arnold

**Dynamisches Task-Scheduling in heterogenen
Mehrprozessorsystemen**

 VOGT

Dresden 2014

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Bibliographic Information published by the Deutsche Bibliothek

The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the internet at <http://dnb.ddb.de>.

Zugl.: Dresden, Techn. Univ., Diss., 2014

Die vorliegende Arbeit stimmt mit dem Original der Dissertation „Dynamisches Task-Scheduling in heterogenen Mehrprozessorsystemen“ von Oliver Arnold überein.

© Jörg Vogt Verlag 2014

Alle Rechte vorbehalten. All rights reserved.

Gesetzt vom Autor

ISBN 978-3-938860-77-9

Jörg Vogt Verlag
Niederwaldstr. 36
01277 Dresden
Germany

Phone: +49-(0)351-31403921

Telefax: +49-(0)351-31403918

e-mail: info@vogtverlag.de

Internet : www.vogtverlag.de

Technische Universität Dresden

Dynamisches Task-Scheduling in heterogenen Mehrprozessorsystemen

Oliver Arnold

von der Fakultät Elektrotechnik und Informationstechnik
der Technischen Universität Dresden

zur Erlangung des akademischen Grades eines

Doktoringenieurs

(Dr.-Ing.)

genehmigte Dissertation

Vorsitzender:	Prof. Dr.-Ing. habil. Leon Urbas
Gutachter:	Prof. Dr.-Ing. Dr. h.c. Gerhard Fettweis Prof. Dr.-Ing. Jürgen Teich
Tag der Einreichung:	10.10.2013
Tag der Verteidigung:	17.01.2014

Danksagung

Diese Arbeit entstand während meiner Zeit am Vodafone Stiftungslehrstuhl für Mobile Nachrichtensysteme der Technischen Universität Dresden. Ich möchte mich bei allen Mitgliedern der Hardware-Arbeitsgruppe bedanken, mit denen ich zusammengearbeitet habe und die immer zu einer erfrischenden Diskussion bereit waren. Für die fachlichen Anregungen, Betreuung und Unterstützung für diese Arbeit, möchte ich meinem Doktorvater Professor Gerhard Fettweis herzlich danken. Des Weiteren danke ich Professor Jürgen Teich von der Friedrich-Alexander-Universität Erlangen-Nürnberg sehr herzlich für die Begutachtung meiner Dissertation.

Ein großer Dank gilt meiner Familie für ihre Unterstützung meiner Arbeit und Tätigkeit an der Technischen Universität Dresden.

Des Weiteren bedanke ich mich beim Tomahawk2-Team, insbesondere bei Benedikt Nöthen, Steffen Kunze, Esther Pérez-Adeva, Emil Matúš, Tobias Seifert, Erik Fischer, Holger Eisenreich und Sebastian Höppner. Ihr unermüdliches Engagement haben dieses Projekt erst möglich gemacht.

Abstract

Simultaneously improving both, computational power and energy efficiency, represents one of the major challenges in the development of future hardware systems. In this regard, the systems' manufacturing process as well as the hardware architecture must be examined. Concerning architecture design, the combination of different types of processing elements (PEs) in a single chip, is a recent trend. A smart management of these PEs allows an increase in computational power as well as energy efficiency.

In this work a dedicated scheduling unit called CoreManager is proposed to control heterogeneous Multiprocessor Systems-on-Chips (MPSoCs). This unit is responsible for dynamically distributing atomic tasks on different PEs as well as managing the inherent data transfers. For this purpose, a runtime analysis of the data dependencies is firstly performed. Based on this analysis, a schedule is created to allocate the PEs and explicitly reserve and administrate the local memories. The results of the dynamic data dependency analysis are additionally reused in order to increase data locality. In particular, required data are kept in the local memories, thus reducing the number of transfers from the global memories.

The CoreManager has been profiled in order to expose the most time consuming components. Dynamic data dependency check was found to be the limiting factor regarding system scalability. An extension of the instruction set architecture of the CoreManager has been developed and integrated to solve this issue. This extension allows speeding-up the data dependency check process and other CoreManager components, thus increasing performance while reducing energy consumption.

A battery-aware mode of operation of the CoreManager is introduced, which allows extending the lifetime of the system. Furthermore, a failure-aware dynamic task scheduling approach for unreliable heterogeneous MPSoCs was integrated in the CoreManager. It enables a detection and isolation of erroneous PEs, connections and memories.

By applying these approaches the efficient management of heterogeneous MPSoCs is enabled.

Zusammenfassung

Eine Erhöhung der Rechenleistung bei einer gleichzeitigen Verbesserung der Energieeffizienz ist eine der Hauptaufgaben bei der Entwicklung zukünftiger Hardwaresysteme. Hierbei muss sowohl der Herstellungsprozess als auch die Hardware-Architektur betrachtet werden. Ein vielversprechender Ansatz für eine Verbesserung der Hardware-Architektur besteht darin, mehrere heterogene Prozessoren auf einem Chip zu integrieren. Hierbei kann, bei einer intelligenten Verteilung der Aufgaben, sowohl die Rechenleistung als auch die Energieeffizienz erhöht werden.

In dieser Arbeit wird ein dedizierter Task-Scheduler, genannt CoreManager, konzipiert und entworfen. Dessen Aufgabe ist es, atomare Tasks dynamisch in einer heterogenen Plattform auf Prozessorelemente (PE) zu verteilen. Die Datenabhängigkeiten der Tasks werden zur Laufzeit analysiert. Anhand dieser Abhängigkeiten wird ein Schedule aufgebaut. Prozessorelemente werden daraufhin allokiert und der lokale Speicher explizit reserviert und verwaltet. Ein Management der Datentransfers erlaubt eine Erhöhung der Datenlokalität. Dies wird durch eine Wiederverwendung der Resultate der dynamischen Datenabhängigkeitsanalyse erreicht.

Eine Analyse der Abarbeitung des CoreManagers hat ergeben, dass die dynamische Datenabhängigkeitsanalyse der zeitaufwendigste Teil der Abarbeitung ist. Um die Skalierung des CoreManagers zu verbessern, wurde dieser um einen anwendungsspezifischen Befehlsatz erweitert. Die weiteren Komponenten des CoreManagers wurden ebenfalls beschleunigt. Bei einer batteriebetriebenen Abarbeitung erlaubt es eine Anpassung der Verfahren im CoreManager, die Systemlaufzeit zu verlängern. Des Weiteren erfolgte eine Anpassung des CoreManagers, um eine fehlerfreie Abarbeitung von Tasks sicherzustellen. Hierbei werden fehlerhafte Prozessoren, Verbindungen und Speicher erkannt, markiert und gegebenenfalls isoliert.

Unter Verwendung dieser Ansätze und Verfahren ist ein effizientes Management von heterogenen MPSoCs möglich.

Inhaltsverzeichnis

Inhaltsverzeichnis	XIII
Abbildungsverzeichnis	XVII
Tabellenverzeichnis	XX
Quelltextverzeichnis	XXII
Abkürzungsverzeichnis	XXIII
Symbolverzeichnis	XXVII
1 Einleitung	1
1.1 Motivation	1
1.2 Ziele der Arbeit	2
1.3 Gliederung der Arbeit	3
2 Grundlagen	5
2.1 Rechnermodell	5
2.1.1 Taskgraph	5
2.1.2 Atomarer Task	6
2.1.3 Kontrollfluss	7
2.2 TaskC Programmiermodell	9
2.3 TaskC Hardwarearchitektur	11
2.4 CoreManager	12
3 Stand der Technik	15
3.1 Heterogene MPSoCs	15
3.1.1 Architekturen für Applikationsprozessoren für mobile Handhelds . .	16
3.1.2 Architekturen für Multimedia Anwendungen	16
3.1.3 Architekturen für die digitale Signalverarbeitung	17
3.2 Programmiermodelle	18
3.3 Dynamische Scheduling-Einheiten	22
3.4 Dynamische Scheduling-Einheiten mit Anpassung des Befehlssatzes	24

3.5	Erhöhung der Datenlokalität	26
3.6	Resiliente Abarbeitung auf Systemebene	27
3.7	Erhöhung der Laufzeit einer Batterie durch Maßnahmen auf Systemebene	28
4	Simulationsmethodik	30
4.1	Simulationsumgebungen und dazugehörige Werkzeuge	31
4.1.1	oaTLM	31
4.1.2	Synopsys CoMET	33
4.1.3	Tensilica Xplorer und XTSC	34
4.2	Universelle Generatoren	37
4.2.1	TGFFGenerator	37
4.2.2	SDF3Generator	38
4.2.3	StaticScheduleGenerator	38
4.3	Analysewerkzeuge	39
4.3.1	TaskVisualizer	39
4.3.2	PowerProfiler	40
4.3.3	DebugChecker	43
5	Software CoreManager	45
5.1	Einführung	45
5.2	Zyklenakkurate Simulationsumgebung	46
5.2.1	Systemmodell	46
5.2.2	CoreManager Ablauf	47
5.2.3	CoreManager Komponenten und Schnittstellen	50
5.3	Erweitertes Taskmanagement	53
5.3.1	Datenlokalität	53
5.3.2	Proaktives Laden von unabhängigen Tasks	54
5.3.3	Erweiterte Spezifikation von Daten	55
5.3.4	Erweiterte Annotation von Datenabhängigkeiten	56
5.3.5	Weiche Echtzeitverarbeitung	57
5.3.6	Leistungsverbrauchs- und Energiemanagement	58
5.4	CoreManager ARM926: Ergebnisse auf Systemebene	59
5.5	CoreManager ARM926: Ergebnisse auf Komponentenebene	66
5.6	Transaktionsbasierte Simulationsumgebung	68
5.7	Ergebnisse der transaktionsbasierten Simulationsumgebung	69
5.8	Zusammenfassung	70
6	Anpassung an fehlerbehaftete Systeme	72
6.1	Resilientes Systemmodell	72
6.2	Task Management für fehlerhafte MPSoCs	73
6.3	CoreManager ARM926: resiliente Abarbeitung	74

7	Batterieabhängiges Taskmanagement	77
7.1	Erweitertes Systemmodell	77
7.2	CoreManager-Adaption	81
7.3	CoreManager ARM926: Anpassungen an eine batteriebetriebene Abarbeitung	82
8	CoreManager mit Befehlssatzerweiterung	84
8.1	Systemmodell für die Befehlssatzerweiterung	84
8.2	Befehlssatzerweiterung	87
8.3	CoreManager mit Befehlssatzerweiterung: Systemebene	92
8.4	CoreManager mit Befehlssatzerweiterung: Komponentenebene	97
8.5	CoreManager mit Befehlssatzerweiterung: Flächenbedarf	102
8.6	CoreManager mit Befehlssatzerweiterung: Leistungs- und Energieverbrauch	104
9	Heterogener Tomahawk2 MPSoC	108
9.1	Tomahawk2 Systemebene	109
9.2	Messergebnisse: CoreManager mit Befehlssatzerweiterung	110
9.3	Vergleich mit anderen Architekturen	111
10	Analytische Betrachtungen und Modelle	114
10.1	Abschätzung der Tasklaufzeit für eine Skalierung auf 1000 Prozessorelemente	114
10.2	Modellierung der Abarbeitungszeit der Komponenten des CoreManagers . .	116
10.3	Modellierung des CoreManagers für Untersuchungen auf einer abstrakteren Ebene	119
11	Zusammenfassung und Ausblick	125
11.1	Zusammenfassung	125
11.2	Ausblick	126
	Literaturverzeichnis	126
A	Applikationen	139
A.1	Bildverarbeitung	139
A.1.1	Sepia	139
A.1.2	Kantendetektion (Sobel)	140
A.1.3	JPEG-Dekodierung	140
A.2	Signalverarbeitung	141
A.2.1	Filter mit endlicher Impulsantwort (FIR)	141
A.2.2	Schnelle Fourier-Transformation (FFT)	141
A.2.3	GSM-Bitübertragungsschicht	142
A.2.4	UMTS-Bitübertragungsschicht	142
A.2.5	LTE-Bitübertragungsschicht	143
A.3	Synthetische Applikationen	144
A.3.1	Variable Anzahl von Datentransfers	144
A.3.2	Variable Taskausführungszeit	145

INHALTSVERZEICHNIS

A.3.3	Task Graphs For Free	145
A.4	Weitere Applikationen	145
A.4.1	Advanced Encryption Standard	145
A.4.2	Mandelbrot-Menge	146
A.4.3	Sortierung	146
B	Weitere Abbildungen	148
C	Weitere Tabellen	152
D	Weitere Quelltexte	155

Abbildungsverzeichnis

2.1	Taskgraph, bestehend aus Tasks mit Eingangs- und Ausgangsdaten	6
2.2	a) Atomarer Task und b) Abarbeitung eines Tasks inklusive der Transfers der Eingangs- und Ausgangsdaten	7
2.3	Kontrollfluss, Taskgraphen und deren Abarbeitung	8
2.4	TaskC Hardwaregrundstruktur	11
2.5	Abarbeitung auf Systemebene	13
2.6	Ein- und zweidimensionale Datenabhängigkeitsanalyse	14
4.1	Übersicht der verwendeten Simulationsumgebungen und Werkzeuge	31
4.2	Transaktionsbasiertes Simulationsumgebung: oaTLM	32
4.3	Synopsys CoMET Simulationsmodell	35
4.4	Entwicklung von neuen Befehlen mit dem Tensilica Xtensa Xplorer	36
4.5	Tensilica XTSC Simulationsmodell	37
4.6	StaticScheduleGenerator	39
4.7	TaskVisualizer: Bildschirmabbild	40
4.8	PowerProfiler	41
5.1	Architektur des zyklenakkuraten Simulationsmodells	47
5.2	Komponenten des Software CoreManagers	49
5.3	PE-Allokation	52
5.4	Erhöhung der Datenlokalität bei abhängigen Tasks	55
5.5	CoreManager ARM926: Skalierung der Applikation bei einer Änderung der Taktfrequenz und der dynamischen Datenabhängigkeitsanalyse (DDAA) des CoreManagers (CM) über einer Veränderung der Laufzeit der Tasks mit jeweils einem Eingangs- und einem Ausgangsdatentransfer mit 10 PEs und einer Taskfenster-Größe von 20	62
5.6	Software CoreManager ARM926: Skalierung der Applikation bei einer Änderung der Anzahl der Transfers	62
5.7	CoreManager ARM926: Beschleunigung bei einer Variation der Anzahl der PEs, normiert auf ein PE	63
5.8	Beschleunigung bei einer Variation der Taktfrequenz des Verbindungsnetzwerkes, normiert auf 25 MHz	63

5.9	TaskVisualizer Ausschnitt bei der Abarbeitung von vier Applikation unterschiedlicher Priorität	65
5.10	Vergleich der Laufzeit der Applikation bei einer Verwendung eines statischen und eines dynamischen Schedules, normiert auf eine statische Variante . . .	65
5.11	Software CoreManager ARM926: Bearbeitungszeit aller im CoreManager vorhandenen Komponenten	67
5.12	Software CoreManager ARM926: Bearbeitungszeit der dynamischen Datenabhängigkeitsanalyse bei einer Änderung der Anzahl der Transfers und der Größe des Taskfensters	67
5.13	Ergebnisse der transaktionsbasierten Simulationsumgebung: Beschleunigung und relative Fehler	70
6.1	Resilientes Systemmodell	73
6.2	CoreManager Fehlerstatus und Übergänge je PE	74
6.3	Veränderung der Applikationslaufzeit durch das Einbringen verschiedener Fehlerarten und -wahrscheinlichkeiten. Für jede Applikation normiert auf eine fehlerfreie Abarbeitung	76
7.1	Um ein Batteriemodell erweitertes Systemmodell	78
8.1	Systemmodell für die Befehlssatzerweiterungen	85
8.2	a) CoreManager Modul und b) PE Modul	86
8.3	Entwicklungsgeschichte der Beschleunigung der dynamischen Datenabhängigkeitsanalyse	89
8.4	Befehlssatzerweiterung für einen flexiblen Schedulingansatz	90
8.5	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Tasklaufzeit, Vergleich der Prozessoren CM-LX4, CM-FLIX und CM-TIE	94
8.6	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Tasklaufzeit, Vergleich der Implementierungen mit Polling und Interrupts auf dem Prozessor CM-TIE	94
8.7	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Tasklaufzeit bei einer Änderung des Anzahl der Transfers und ohne/mit DDAA, Abarbeitung auf einem CM-TIE Prozessor	96
8.8	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Größe des Taskfensters	96
8.9	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Tasklaufzeit bei einer Änderung der Größe der Transfers mit 75 % Taskabhängigkeiten. Abarbeitung auf einem CM-TIE Prozessor für sieben PEs	97
8.10	CoreManager mit Instruktionssatzerweiterung: Bearbeitungszeit der Komponenten im CoreManager für verschiedene Prozessoren und eine unterschiedliche Anzahl von Datentransfers	99
8.11	CoreManager mit Instruktionssatzerweiterung: Bearbeitungszeit der dynamischen Datenabhängigkeitsanalyse	99

8.12	CoreManager mit Instruktionssatzerweiterung: Bearbeitungszeit der dynamischen PE-Allokation	100
8.13	CoreManager mit Instruktionssatzerweiterung: Bearbeitungszeit des dynamischen Scheduling	100
9.1	Tomahawk2 Die-Photo	109
9.2	ATE-Produkt	113
10.1	Veranschaulichung der Modellierung des CoreManagers	120
A.1	Parallelisierung der JPEG Applikation	141
A.2	GSM-Bitübertragungsschicht für den Downlink: Sender-, Empfängerseite und AWGN Kanal	143
A.3	UMTS-Bitübertragungsschicht für den Downlink auf Senderseite	143
A.4	LTE-Bitübertragungsschicht für den Downlink auf Senderseite	144
A.5	Parallelisierung der Sortierungsapplikation	147
B.1	CoreManager ARM926: Vergleich der Skalierung bei der Verwendung von fünf und zehn PEs (CoreManager Taktfrequenz: 400 MHz)	148
B.2	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Tasklaufzeit, Vergleich der Implementierungen mit Polling und Interrupts auf dem Prozessor CM-FLIX	149
B.3	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Tasklaufzeit, mit Polling (APP: niedrig, PE: hoch)	149
B.4	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Tasklaufzeit (APP: niedrig, PE: Interrupts)	150
B.5	CoreManager mit Instruktionssatzerweiterung: Skalierung über die Tasklaufzeit bei einer Änderung des Größe der Transfers mit 50 % Taskabhängigkeiten. Abarbeitung auf einem CM-TIE Prozessor für sieben PE	150
B.6	CoreManager mit Instruktionssatzerweiterung: Bearbeitungszeit der Komponenten im CoreManager bei einer Tasklaufzeit von hunderttausend Takten	151

Tabellenverzeichnis

5.1	Zyklenakkurates Systemmodell: Parameterspezifikation	48
5.2	Vergleich der Skalierung auf 80 % der vorhandenen PEs des Nexus (Taktfrequenz unbekannt) [MJ10] und des ARM926 CoreManagers (Taktfrequenz 400 MHz)	61
5.3	Relative Laufzeitänderung in Prozent bei einer Verwendung von Datenlokalität bei einer Taktfrequenz des Verbindungsnetzwerks (VBN) von 100 und 400 MHz	64
7.1	Leistungsverbräuche der einzelnen Komponenten für ein System mit Batterie	79
7.2	Batteriebewusste Abarbeitung mit niedriger und hoher Grundlast. Prozentuale Änderung im Vergleich zu einer Abarbeitung ohne Berücksichtigung der Batterie	83
8.1	Befehlssatzerweiterung: Übersicht aller neuen Befehle, Befehlsargumente und -beschreibung	91
8.2	Bearbeitungszeit der dynamischen Allokation von lokalem Speicher (relative Zunahme im Vergleich zu der Bearbeitungszeit des CM-TIE Prozessors) . .	101
8.3	Bearbeitungszeit für das Auflösen der Datenabhängigkeiten (in Takten) . .	101
8.4	Benötigte Fläche des CoreManagers für verschiedene Prozessorkonfigurationen und Frequenzen, in mm ² (65 nm Prozess für niedrige Leistungsaufnahme, Typischer Fall, 25 °C, 1,25 V)	103
8.5	CoreManager CM-TIE: Benötigte Fläche von ausgewählten Befehlen (Abschätzung mit Tensilica Xplorer für eine 65 nm Bibliothek mit niedriger Leistungsaufnahme)	103
8.6	CoreManager mit Befehlssatzerweiterung: Leistungs- (Prozessor, Speicher und Sonstige) und Energieverbrauch (Prozessor, je Transfer-Vergleich bzw. je Task) für 65 nm TSMC Prozess für niedrige Leistungsaufnahme (schlimmster Fall mit 125 °C, 1,08 V)	106
9.1	Messergebnisse des CoreManagers mit Befehlssatzerweiterung, Fläche in mm ²	110
9.2	Messergebnisse des CoreManagers mit Befehlssatzerweiterung, Leistungsverbrauch in mW	111
9.3	ATE-Produkt: Parameter	112

10.1	Abschätzung der benötigten Tasklaufzeit bei 90 % Auslastung von 1000 Prozessorelementen	116
10.2	CoreManager Modellierung: Parameter für CM-TIE für unterschiedliche Tasklaufzeiten (TL) und Anzahl von Transfers	122
10.3	CoreManager Modellierung: Modellierte Werte und relativer Fehler für die dynamische Datenabhängigkeitsanalyse im Vergleich zu Tabelle 10.2	123
10.4	CoreManager Modellierung: benötigte Tasklaufzeit (TL) für eine Skalierung auf 1000 PEs mit einer Auslastung von 90 % auf dem Prozessor CM-TIE .	124
C.1	TGFF Parameter	152
C.2	Tensilica LX4 Parameter	153
C.3	Relative Abweichung bei einer Änderung der Tasklaufzeit zu 50 %, 100 % und 150 % für die Prozessoren CM-TIE, CM-FLIX und CM-LX4, mit und ohne dynamische Datenabhängigkeitsanalyse (DDAA)	153
C.5	Profile für eine weitere Last für Systeme mit Batterie.	154
C.4	Benötigte Fläche des CoreManager für verschiedene Prozessorkonfigurationen und Frequenzen, in mm ² (65 nm Prozess für niedrige Leistungsaufnahme, Worst Case, 125 °C, 1,08 V)	154

Quellcodeverzeichnis

- 2.1 TaskC Programmiermodell: Kontrollcode 10
- 2.2 Taskdefinition 10
- 5.1 Explizite Datenabhängigkeiten 57
- D.1 Tomahawk2: CoreManager Debug Opcodes, Teil 1 156
- D.2 Tomahawk2: CoreManager Debug Opcodes, Teil 2 157

Abkürzungsverzeichnis

Abb.	Abbildung
AES	Advanced Encryption Standard
ADPLL	All Digital Phase-Locked Loop
APP	Applikationsprozessor
ASAP	As-Soon-As-Possible
ASIC	Application-Specific Integrated Circuit
ASIP	Application Specific Integrated Processor
ATE	Produkt aus Fläche, Zeit und Energie
AWGN	Additive white Gaussian noise
bzgl.	bezüglich
bzw.	beziehungsweise
ca.	circa
Cell BE	Cell-Broadband-Engine
CM	CoreManager
CMOS	Complementary Metal Oxide Semiconductor
CUDA	CompUte Driver Architecture
DAG	gerichteter azyklischer Graph
DDAA	Dynamische Datenabhängigkeitsanalyse
DFG	Datenflussgraph

ABKÜRZUNGSVERZEICHNIS

d. h.	das heißt
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
DSM	Distributed Shared Memory
DSP	Digitaler Signalprozessor
EIB	Element Interconnect Bus
FEC	Forward Error Correction
FFT	Schnelle Fourier-Transformation
FIFO	First In-First Out
FIR	Finite Impulse Response
FLIX	Flexible Length Instruction Extensions
FPGA	Field Programmable Gate Array
FSM	Finite State Machines
GPP	General Purpose Processor
GSM	Global System for Mobile Communication
GTU	globale Task-Unit
JPEG	Joint Photographic Experts Group
KPN	Kahn Prozessnetzwerke
LTE	Long Term Evolution
LTU	lokale Task-Unit
MeP	Media embedded Processor
MFC	Memory Flow Controller
MHz	Megahertz
MISP	Multiple Instructions Stream Prozessor
MPI	Message Passing Interface
MPSoC	Multi-Processor System-on-Chip
NoC	Network-on-Chip
NUMA	Non-Uniform Memory Access
oDL	Ohne Datenlokalität
OmpSs	OpenMP Superscalar
OMT	Object Mapping Tabellen
OpenCL	Open Compute Language
OpenMP	Open Multi-Processing

PE	Prozessorelement
PPE	Power Processing Element
QM	Queue Manager
RISC	Reduced Instruction Set Computer
RO	Read-Only
RTL	Register Transfer Level
RW	Read-Write
SD	Sphere Detection
SDF	Synchrone Datenflussgraphen
SDR	Software Defined Radio
SIMD	Single Instruction Multiple Data
SMP	symmetrisches Multiprozessorsystem
SMPS	SMP Superscalar
SoC	System-on-Chip
SPE	Synergistic Processing Elemente
SPMD	Single Program Multiple Date
SPU	Synergistic Processing Unit
StarSs	STAR superscalar
Szen.	Szenario
TGFF	Task Graphs For Free
TIE	Tensilica Instruction Extension
TL	Tasklaufzeit
TRS	Task-Reservation Station
TSMC	Taiwan Semiconductor Manufacturing Company
TUD	Technische Universität Dresden
TU Dresden	Technische Universität Dresden
UART	Universal Asynchronous Receiver Transmitter
UMC	United Microelectronics Corporation
UMTS	Universal Mobile Telecommunication System
VCN	Verbindungsnetzwerks
VCD	Value Change Dump
VDSP	Vektor-DSP
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
Vrgl.	Vergleich

ABKÜRZUNGSVERZEICHNIS

WFI	Wait-for-Interrupt
XML	Extensible Markup Language
XPG	Xtensa Processor Generator
XTSC	XTensa SystemC
z. B.	zum Beispiel

Symbolverzeichnis

α	Batterie-Parameter: Batteriekapazität
β	Batterie-Parameter: Nichtlinearität
c_{Konst}	Konstanter Mehraufwand
I_k	Konstante Last
n_{IN}	Anzahl der Eingangsdatentransfers
n_{OUT}	Anzahl der Ausgangsdatentransfers
n_{TiS}	Anzahl der Tasks im System
$n_{Transfers}$	Anzahl der Eingangs- und Ausgangsdatentransfers
n_{ws}	Größe des Taskfensters
t_{Dep}	Zeit: Dynamische Datenabhängigkeitsanalyse
$t_{DMAC_Allokation}$	Zeit: DMAC Allokation
t_{DMAC_Config}	Zeit: DMAC Konfiguration
$t_{Interrupt}$	Zeit: Interrupt Routine
$t_{Mehrbelastung}$	Zeit: Mehrbelastung
$t_{Nachfolger_Bearbeiten}$	Zeit: Nachfolger-Tasks bearbeiten
$t_{PE_Allokation}$	Zeit: PE Allokation
$t_{Speicher_Allokation}$	Zeit: Allokation von lokalem Speicher
t_{Task_END}	Zeit: Bearbeitung eines Tasks nach dessen Start
t_{Task_START}	Zeit: Bearbeitung eines Tasks vor dessen Start
t_{Task_ZEIT}	Zeit: Bearbeitung eines Tasks
$t_{TaskBeschreibung}$	Zeit: Transfer der Taskbeschreibung
$t_{Tasklaufzeit}$	Tasklaufzeit
V_{Mode}	Spannung für die Aktivierung der CoreManager-Verfahren
V_{Tot}	Totspannung einer Batterie

Kapitel 1

Einleitung

1.1 Motivation

Zukünftige technische Systeme verlangen eine immer höhere Rechenleistung bei gleichzeitiger Reduzierung des Energieverbrauches. Dies kann einerseits durch eine Verbesserung in der Herstellung, z. B. durch eine reduzierte Strukturbreite der Fertigungstechnologie und durch spezielle Prozesse für niedrige Energieverbräuche, realisiert werden. Andererseits können Verbesserungen durch Anpassung der Komponenten, der Architektur und der Laufzeitumgebung des Systems erzielt werden.

Ein Ansatzpunkt, um gleichzeitig sowohl eine Erhöhung der Rechenleistung als auch die Reduktion des Energieverbrauches zu erreichen, ist die Veränderung der Chiparchitektur. Hierbei werden anstatt eines Prozessors mehrere auf dem Chip integriert. Dieser Schritt folgt logisch aus der Tatsache, dass die dynamische Verlustleistung eines Prozessors quadratisch mit der Versorgungsspannung, aber nur linear mit der Taktfrequenz, anwächst. Eine Verteilung der Rechenleistung auf mehrere Prozessoren erlaubt demzufolge eine Reduzierung der Taktfrequenz bei gleichzeitiger Senkung der Versorgungsspannung. Eine Reduzierung der Leistungsaufnahme ist somit möglich. Der Trend, mehrere Prozessoren auf einem Chip zu integrieren, wird sich in zukünftigen Systemen weiter fortsetzen [ITR08]. Im Zeitraum zwischen 2010 und 2022 wird erwartet, dass sich die Anzahl der Prozessoren auf einem Chip, um ungefähr den Faktor 20 erhöht. Hinsichtlich der Leistungsaufnahme von portablen Geräten, wird eine Verdreifachung im selben Zeitraum prognostiziert. Des Weiteren steigen die Anforderungen an die Wiederverwendung von Komponenten eines Chips. Daher wird auch eine Zunahme der Wiederverwendbarkeit der Bauelemente von 50 Prozent im Jahr 2010 auf 94 Prozent im Jahr 2022 erwartet. Allerdings wirft der praktische Übergang, bezüglich der Verwendung von einem Prozessor hin zu einem Mehrprozessorensystem auf einem Chip (MPSoC), viele Fragen auf, so z. B.: Wie können mehrere Prozessoren auf einem Chip integriert werden? Müssen die Prozessoren homogen sein? Wie kann eine Synchronisation und ein Datenaustausch zwischen den Prozessoren realisiert werden? Kann man existierende Prozessoren wiederverwenden oder ist eine Anpassung dieser notwendig? Wie kann die vorhandene Rechenleistung der Applikation so zur Verfügung gestellt werden,

dass keine Kenntnis der Hardware benötigt wird? Wie skalieren sowohl die Hardware als auch die Software? Wie kann die Leistungsaufnahme reduziert und reguliert werden? Diese Liste von Fragen kann beliebig erweitert werden. Sie zeigt gleichzeitig die Komplexität der Aufgabe und deren inhärenten Problemstellungen, welchen sich die Forschung nun stellen muss. Gerade und insbesondere hinsichtlich der Weiterentwicklung bestehender Systeme hin zu zukunftsweisenden.

Die Laufzeitumgebung des Systems bestimmt die Abarbeitung der Applikation. Hierzu gehören Verfahren, die Daten und Aufgaben nach zeitlichen Vorgaben an die Prozessoren zu verteilen. Eine Verbesserung der Verfahren ist z. B. möglich, indem die Datenlokalität erhöht wird. Hierbei kann, ohne einen Eingriff in die Applikation oder die Hardware, eine Steigerung der Leistungsfähigkeit des Systems erreicht werden. Der in einigen Systemkonfigurationen auftretende Flaschenhals beim Zugriff auf den Speicher, welcher durch das Memory Wall beschrieben wird [ABC⁺06], verringert sich.

Auf der Ebene der Applikation für Terminals ist es möglich, Modem- und Nutzeranwendungen auf einem Chip zu vereinen und parallel abzuarbeiten. Es wird beiden Seiten ein kontrollierter Zugriff auf alle Ressourcen des Systems gewährt. Im EU Projekt *ICT-eMuCo* [EMU10] wurde dieser Ansatz untersucht. Resultat war, dass sowohl Nutzer-Applikationen wie z. B. die JPEG-Dekodierung als auch die Verarbeitung von Basisband-Algorithmen in gleichzeitiger Ausführung ermöglicht wird. Hierbei kann eine Abarbeitung mit einer Priorisierung bestimmter Tasks verwendet werden. Die Anforderung an die Laufzeitumgebung steigt in diesem Fall, da sich die Parallelitätsprofile [RLAF09] der Applikation unterscheiden und deren Startzeitpunkte ungewiss sind. Dadurch wird für diese eine dynamische Zuordnung von Ressourcen notwendig, um eine effiziente Nutzung des Systems zu ermöglichen. Des Weiteren kann für bestimmte Applikationen keine Vorhersage bezüglich der Laufzeit eines Task gemacht werden. Für batteriebetriebene Geräte ist eine Adaption notwendig, um deren Laufzeit zu verlängern.

Für den zellularen Mobilfunk werden Basisstationen benötigt, deren Betrieb einen hohen Energiebedarf aufweist [FZ08]. Zudem ist der relative Anteil des Energieverbrauches für die Basisbandverarbeitung größer, je kleiner die Basisstation ist [ARFB10]. Die zu erwartende Steigerung in der Nutzung von kleinen Basisstationen für Femto- und Mikro-zellen, wird eine effiziente Abarbeitung des Basisbandes unumgänglich werden lassen. In Basisstationen führt ein dynamisches Management der Prozessoren und Speicherinhalte zu einer Reduzierung der Leistungsaufnahme. Innerhalb des Projektes *Cool BaseStations* wurde dieser Ansatz verfolgt und erfolgreich umgesetzt [COO11]. Das Resultat ist eine adaptive Anpassung des Energieverbrauches an die Last der Basisstation. Die Energieeffizienz wird hierbei gesteigert.

1.2 Ziele der Arbeit

Die Ziele dieser Arbeit sind:

- Konzept und Entwurf eines dedizierten dynamischen Task-Schedulers für heterogene Mehrprozessor-Architekturen; Analyse des Verhaltens und Bestimmung der Limitie-

rung wesentlicher Systemparameter wie z. B. der Skalierung über die Anzahl der Prozessoren

- Entwicklung einer modularen Simulationsplattform und dazugehörige Werkzeuge für experimentelle Analysen des taskbasierten Schedulers
- Konzept und Entwurf einer funktional korrekten, abstrakten Simulationsumgebung, um Applikationen mit einem taskbasierten Programmiermodell zu entwerfen
- Entwurf von Entwicklungswerkzeugen für die Bearbeitung des Quellcodes, der Analyse der Abarbeitung und für die Visualisierung des Verhaltens von Applikationen
- Analyse wesentlicher Komponenten eines dynamischen Task-Schedulers und Beschleunigung dieser unter Verwendung eines anwendungsspezifischen Befehlssatzes
- Konzept, Entwurf und Untersuchung eines Task-Schedulers mit verteilter Architektur
- Konzept und Entwurf einer universellen Debug-Umgebung sowohl für Simulationen, für FPGA Prototypen als auch für Chip-Implementierungen
- Entwicklung und Untersuchung eines Task-Scheduling Verfahrens, welches den Batteriezustand mit einbezieht
- Konzept, Entwurf und Untersuchung eines Task-Schedulers, welcher ein statisches Schedule abarbeiten kann
- Untersuchung und Entwurf eines ausfallsicheren Task-Schedulingverfahrens hinsichtlich Fehlern in der Abarbeitung von Tasks; des Weiteren ein Ansatz um dynamisch fehlerhafte Prozessoren und Speicher zu erkennen, zu markieren und gegebenenfalls zu isolieren
- Beschreibung des Zeitverhaltens wesentlicher Komponenten des Task-Schedulers mit analytischen Methoden
- Konzept und Entwurf eines abstrakten Modells des Task-Schedulers

1.3 Gliederung der Arbeit

Diese Arbeit untergliedert sich wie folgt: Die zum Verständnis benötigten Grundlagen der verwendeten Hardware-Architektur, das TaskC Programmiermodell und die Task-Schedulingeinheit, genannt CoreManager, werden im zweiten Kapitel vorgestellt. Im darauf folgenden Kapitel befindet sich ein Vergleich der hier gewählten Ansätze mit dem Stand der Technik. Im vierten Kapitel werden die verwendeten Entwicklungswerkzeuge vorgestellt, von den ein Großteil neu geschrieben wurde, und neben ihren Einsatzzwecken auch ihre Arbeitsweisen erläutert. Im fünften Kapitel wird die Softwareimplementierung des Task-Schedulers, genannt CoreManager, auf der Synopsys CoMET-Plattform eingeführt

und analysiert. Hierbei werden mehrere Anpassungen vorgenommen, um sowohl die Flexibilität als auch die Leistungsfähigkeit zu erhöhen. Das sechste Kapitel beschäftigt sich mit fehlerhaften Systemen. Hierbei wird ein Ansatz auf Systemebene vorgestellt, welcher zur Laufzeit Fehler erkennen und korrigieren kann. Fehlerhafte Module werden isoliert. Im siebten Kapitel wird eine Adaption des CoreManager-Konzeptes an ein batteriebetriebenes System vorgenommen. Dafür wird unter anderem ein Batteriemodell in das System eingeführt. Dies erlaubt eine gezielte Änderung des Scheduling und dessen Anpassung an den aktuellen Zustand der Batterie. Im folgenden Kapitel wird die Tensilica XTSC Umgebung verwendet um den Befehlssatz eines Prozessors für das Task-Management anzupassen und hierbei zu beschleunigen. Im darauf folgenden Kapitel wird die Chip-Implementierung des heterogenen Tomahawk2 MPSoCs vorgestellt. Der Fokus liegt hierbei auf dem CoreManager. Für diesen werden Messwerte dargelegt und mit anderen Ansätzen verglichen. In Kapitel zehn wird eine analytische Betrachtung des Systems und eine Modellierung des CoreManagers vorgestellt. Das abschließende Kapitel fasst die Arbeit zusammen und zeigt offene Punkte für weitere Forschungsmöglichkeiten auf.

Relevante eigene Publikationen

- ***Leistungsverbrauch von Makro- und Femtobasisstationen***

Arnold, Oliver ; Richter, Fred ; Fettweis, Gerhard ; Blume, Oliver: Power consumption modeling of different base station types in heterogeneous cellular networks. In: *IEEE Future Network and Mobile Summit*, 2010

- ***Parallelitätsprofile von Applikationen und deren Anwendung***

Ristau, Bastian ; Limberg, Torsten ; Arnold, Oliver ; Fettweis, Gerhard: Dimensioning heterogeneous MPSoCs via parallelism analysis. In *Design, Automation & Test in Europe Conference & Exhibition, DATE'09.*, 2009

- ***CoreManager und TaskC Programmiermodell***

Arnold, Oliver ; Matus, Emil ; Noethen, Benedikt ; Winter, Markus ; Limberg, Torsten ; Fettweis, Gerhard: Tomahawk: Parallelism and heterogeneity in communications signal processing MPSoCs. In: *ACM Transactions on Embedded Computing Systems (TECS) - Special Issue on Design Challenges for Many-Core Processors*, 2014