TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Angewandte Mechanik

Methoden zur parallelen Berechnung von nicht-glatten dynamischen Systemen

Dipl.-Ing. Univ. Patrick Jan Clauberg

Vollständiger Abdruck der von der Fakultät für Maschinenwesen
der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. dr. ir. Daniel J. Rixen
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr.-Ing. habil. Heinz Ulbrich
	2. Univ.-Prof. Dr.-Ing. Horst Baier

Die Dissertation wurde am 14.08.2012 bei der Technischen Universität München
eingereicht und durch die Fakultät für Maschinenwesen
am 20.12.2012 angenommen.

Zusammenfassung

Simulationsmethoden nehmen im heutigen Produktentwicklungsprozess einen integralen Bestandteil ein. Je später Simulationen im Produktentwicklungsprozess eingesetzt werden, desto detailliertere Simulationsmodelle müssen gewählt werden, die zu höheren Rechenzeiten führen. Um weiterhin Simulationsmethoden effizient einsetzen zu können, müssen die damit verbundenen Rechenzeiten reduziert werden. Eine der wesentlichen Möglichkeiten dazu bieten Parallelisierungsmethoden, da nahezu jeder moderne Rechner über mehr als einen Rechenkern verfügt. In dieser Arbeit werden die fein-, mittel- und grobstrukturierte Ebene der Parallelisierung hinsichtlich ihres Potentials zur parallelen Berechnung von nicht-glatten Mehrkörpersystemen untersucht. Auf der mittelstrukturierten Ebene werden zwei neue Methoden zur parallelen Auswertung der Bewegungsgleichungen des Mehrkörpersystems vorgestellt. Hinsichtlich der numerischen Integration von nicht-glatten dynamischen Systemen werden einerseits Methoden zur Parallelisierung und andererseits zwei Time-Stepping Verfahren mit inexakten Jacobi-Matrizen gezeigt. Das enorme Potential der entwickelten Methoden wird an akademischen und komplexen industriellen Beispielen nachgewiesen.

Abstract

Simulation methods are an important part of today's product development process. At later stages of the product development process, it is necessary to utilize more complex simulation models which lead to increased computational times. In order to realize efficient simulation processes, computational times have to be reduced. Since nearly every personal computer provides more than one core, parallelization methods are one of the most promising ways. In this thesis, the finely-, middle- and coarsely-structured level of parallelization methods are investigated regarding their potential for non-smooth multibody dynamics. Concerning the middle-structured level, two new internal parallelization methods for the parallel evaluation of the equations of motion are presented. On the coarsely-structured level, two Time-Stepping schemes with inexact Jacobian matrices and parallelization methods within the numerical integration schemes are proposed. The huge potential of the developed methods is presented by academic and complex industrial examples.

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Angewandte Mechanik der Technischen Universität München unter der Leitung meines Doktorvaters Univ.-Prof. Dr.-Ing. habil. Heinz Ulbrich. Ihm gebührt mein besonderer Dank für seine stete fachliche und menschliche Unterstützung sowie für das entgegengebrachte Vertrauen. Seine Unterstützung und die vielfältigen Möglichkeiten am Lehrstuhl haben maßgeblich zum Gelingen meiner Arbeit beigetragen.

Herrn Univ.-Prof. Dr.-Ing. Horst Baier danke ich für sein Interesse an meiner Arbeit und die Übernahme meines Zweitgutachtens. Herrn Univ.-Prof. dr. ir. Daniel J. Rixen danke ich für die Übernahme des Vorsitzes meiner Prüfungskommission.

Allen Mitarbeitern des Lehrstuhls gilt mein Dank für die große Hilfsbereitschaft und das hervorragende Arbeitsklima. Insbesondere möchte ich mich bei Markus Friedrich, Benedikt Huber, Robert Huber, Johannes Mayet und Corinna Scheffel für die kritische Durchsicht meiner Arbeit und viele interessante Diskussionen bedanken.

Nicht zuletzt gebührt mein besonderer Dank meinem Vater Karl-Heinz Clauberg für seine immerwährende Unterstützung bei meinem bisherigen Werdegang.

Garching, im Januar 2013 *Jan Clauberg*

Inhaltsverzeichnis

1 Einleitung

Simulationsmethoden nehmen heutzutage bereits einen ausgesprochen wichtigen Platz im Produktentstehungsprozess ein, dessen Trend eindeutig in Richtung Virtualisierung zeigt. Beispielsweise werden in der Automobilindustrie zunehmend mehr Hardware-Prototypenstufen durch rein virtuelle Prototypenstufen ersetzt.

Dieser Trend wird maßgeblich durch den Kostendruck erzeugt, der in nahezu jedem Entwicklungsbereich herrscht. Die Konstruktion und der Aufbau von Hardware Prüfständen sind äußerst zeit- und kostenaufwendig. Folglich ist es nicht verwunderlich, dass Prüfstände in den ersten Entwicklungsphasen zunehmend durch Simulationen ersetzt werden. Aber nicht nur die Konstruktion und der Aufbau der Prüfstände ist zeit- und kostenaufwendig, sondern auch jeder einzelne Versuch. Simulationsmethoden ermöglichen es, viele verschiedene Konfigurationen technischer Systeme zu berechnen, ohne dass für jede Konfiguration ein realer Versuch durchgeführt werden muss. Neben der Zeit- und Kostenersparnis erlauben Simulationen zudem virtuelle Sensoren an Stellen des technischen Systems anzubringen, an denen im realen System keine Sensoren angebracht werden können. Allerdings darf trotz aller Vorteile der Simulationsmethoden nie vergessen werden, dass Simulationen nur dann sinnvoll eingesetzt werden können, wenn sie ausführlich verifiziert worden sind.

Unter den Simulationsmethoden nimmt die Simulation von Mehrkörpersystemen eine zentrale Rolle ein. Sie dient als Simulationsbasis in vielen Anwendungen, begonnen bei HIL-Prüfständen[1] bis hin zu Fahrdynamikuntersuchungen. Die nicht-glatte Mehrkörpertheorie bietet dabei Methoden um auch sehr große Mehrkörpersysteme mit einer hohen Anzahl an ein- und zweiseitigen Bindungen effizient numerisch lösen zu können. Sie erlaubt es zudem Haft-Gleit-Übergänge zu modellieren und Stöße physikalisch motiviert zu berücksichtigen, indem die entsprechenden Gesetze nicht wie in der glatten Mehrkörperdynamik üblich regularisiert werden, sondern als mengenwertige Nebenbedingungen formuliert werden. Zur Lösung dieser Nebenbedingungen wurden vor allem in den letzten 10 Jahren sehr effiziente und schnelle Algorithmen entwickelt [59, 114].

Seit einigen Jahren wird jedoch offensichtlich, dass die Größe und die Komplexität von Simulationsmodellen schneller anwachsen als die Rechengeschwindigkeit der Simulationsrechner. Simulationen können jedoch nur dann weiterhin zu Zwecken wie der Parametervariation oder der numerischen Optimierung genutzt werden, wenn die Rechenzeiten niedrig sind.

Neben einer Vielzahl weiterer Methoden, um die Rechenzeit von Simulationen zu reduzieren, nimmt die Methode der Parallelisierung heutzutage eine zentrale Rolle ein.

1 HIL = Hardware in the Loop

Seit einiger Zeit stagniert die Taktfrequenz der Prozessoren von Arbeitsplatzrechnern, jedoch verfügt mittlerweile nahezu jeder Arbeitsplatzrechner über mehr als einen Rechenkern. Folglich müssen Methoden entwickelt und umgesetzt werden, um dieses Potential zu nutzen. In vielen anderen Simulationsbereichen, wie den Finite-Elemente Methoden oder der CFD-Simulation, haben parallele Methoden bereits vor Jahren Einzug gehalten. In der Mehrkörpertheorie wird das enorme Potential von modernen Rechnern jedoch noch nicht zufriedenstellend ausgenutzt.

1.1 Ziel und Aufbau der Arbeit

Ziel dieser Arbeit ist es, Methoden vorzustellen, die das enorme Potential von modernen Mehrkernrechnern in der Mehrkörpersimulation ausnutzen. Dabei sollen neben den theoretischen Überlegungen vor allem auch die Anwendung und die experimentelle Untersuchung der Methoden im Mittelpunkt stehen.

Deshalb ziehen sich die Anwendungsbeispiele, die in Kapitel 6 ausführlich dargestellt werden, als eine Art roter Faden durch die gesamte Arbeit. Die Effizienz und die Eigenschaften aller Methoden werden an diesen Beispielen gezeigt. Zu den Beispielen gehören sowohl typische akademische Beispiele als auch industrielle Anwendungen, sodass das reale Spektrum der Mehrkörpersimulation gut repräsentiert wird. Die genutzte Rechnerarchitektur und die verwendeten Systembibliotheken werden in Abschnitt 1.2 genannt.

In Kapitel 2 wird ein Überblick über die nicht-glatte Mehrkörpertheorie gegeben, die als Übermenge ebenfalls die Theorie der glatten Mehrkörpersysteme beinhaltet. Dabei wird auf die Herleitung der Bewegungsgleichungen und auf die Formulierung der mengenwertigen Nebenbedingungen eingegangen. Abgeschlossen wird das Kapitel mit einer Einführung in die softwareseitige Umsetzung der Struktur der Mehrkörperdynamik in eine Simulationsumgebung.

Kapitel 3 beschäftigt sich allgemein mit der Thematik der Parallelisierung. In diesem Zusammenhang wird darauf eingegangen, welche parallelen Hardware-Architekturen für die parallele Berechnung von Mehrkörpersystemen in Frage kommen, und welche Methoden für die Parallelisierung geeignet sind. Einen zentralen Bestandteil dieses Kapitels nimmt die Beschreibung der Herausforderungen der Parallelisierung ein, die bei der parallelen Berechnung von Mehrkörpersystemen auftreten. Ebenso werden Begriffe, Definitionen und die Eingliederungen der Parallelisierungsmethoden eingeführt, die in der vorliegenden Arbeit verwendet werden. Es wird experimentell untersucht, ob die Parallelisierung auf der Ebene der linearen Algebra sinnvoll ist. Das Kapitel schließt mit der Thematik der Lastverteilung von parallelen Programmen.

In der vorliegenden Arbeit werden parallele Methoden vorgestellt, die sich in zwei Klassen einteilen lassen. In die systeminterne Parallelisierung und die Parallelisierung auf Integratorebene.

Kapitel 4 behandelt die systeminterne Parallelisierung. Es werden zwei neue Methoden zur systeminternen Parallelisierung vorgestellt: Die *Grenzzeit* Parallelisierungsmethode und die *semi-dynamische* Parallelisierungsmethode. Beide neuen Methoden werden auf die Beispiele aus Kapitel 6 angewandt und die Ergebnisse interpretiert. Abschließend erfolgt eine Gegenüberstellung der genannten Methoden zur systeminternen Parallelisierung.

Kapitel 5 behandelt die effiziente numerische Integration von nicht-glatten Mehrkörpersystemen. Dazu untergliedert sich das Kapitel in einen ersten Teil, der sich mit der impliziten numerischen Integration beschäftigt und einen zweiten Teil, der sich mit Parallelisierungsmethoden auf Integratorebene beschäftigt.
Für die effiziente numerische Integration von nicht-glatten Mehrkörpersystemen werden Verfahren mit exakter und inexakter JACOBI-Matrix behandelt. Für die Integration mit inexakten JACOBI-Matrizen werden ein linear-implizites Verfahren und ein semi-voll-implizites Verfahren vorgeschlagen. Die Effizienz der Verfahren wird ebenfalls an Beispielen aus Kapitel 6 gezeigt.
Zur Ordnungserhöhung der Verfahren werden Extrapolationsmethoden herangezogen, die sich hervorragend zur parallelen Berechnung eignen. Zusätzlich zur parallelen Berechnung der Zwischenschritte der Extrapolationsmethoden wird gezeigt, wie auch die Speicherung der Simulationsergebnisse parallel zur Integration ausgeführt werden kann. Neben den genannten Parallelisierungsmethoden wird dargestellt, wie sich die numerische Berechnung der JACOBI-Matrizen für die implizite Integration parallelisieren lässt. Alle Parallelisierungs- und Integrationsmethoden werden anhand von Beispielen untersucht und diskutiert.

In Kapitel 7 wird eine Zusammenfassung der wichtigsten Aspekte der Arbeit gegeben. Die Zusammenfassung schließt mit dem Vorschlag von Ansatzpunkten für weitere Arbeiten zur Parallelisierung von Mehrkörpersimulationen.

1.2 Software und Hardware

In diesem Abschnitt werden die Software und Hardware beschrieben, die in dieser Arbeit verwendet werden. Abschnitt 1.2.1 geht dabei sowohl auf die verwendete Mehrkörpersimulationsumgebung ein als auch auf das zugrunde liegende Betriebssystem und seine Bibliotheken. Abschnitt 1.2.2 gibt einen Überblick über die verwendete Hardware.

1.2.1 Simulationsumgebung

Mehrkörpersimulationsumgebung

Alle Methoden und Algorithmen wurden in die Mehrkörpersimulationsumgebung MBSIM [13] implementiert.

MBSim ist eine Modellierungs- und Simulationsumgebung für nicht-glatte dynamische Systeme, die am Lehrstuhl für Angewandte Mechanik der Technischen Universität München entwickelt wird. Die Entwicklung geht im Wesentlichen auf Förg [59], Zander [151], Schindler [126], Friedrich [65], Schneider [131] und Huber [82] zurück.

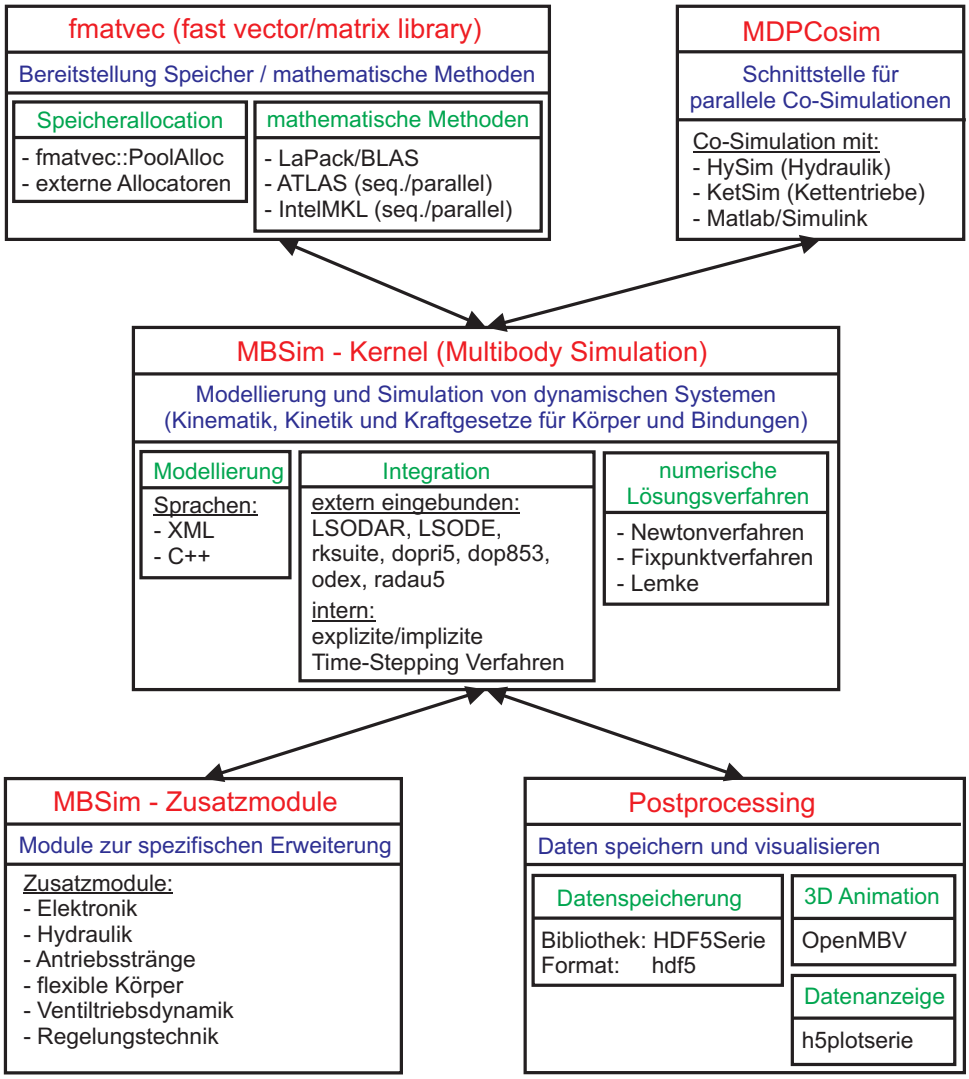


Abbildung 1.1: Einbettung von MBSim in die gesamte Simulationsumgebung.

MBSim [13] ist in der objektorientierten Programmiersprache C++ geschrieben und verfolgt wie auch die Umgebungen Siconos [20] und daVinci Code [27] eine Auftrennung aller Objekte in massebehaftete Körper (objects) und masselose Verbindungen (links). Abbildung 1.1 zeigt die gesamte Modellierungs- und Simulationsumgebung, in die MBSim als zentraler Bestandteil eingebettet ist.

fMatVec - fast matrix/vector library Die mathematische Basis stellt die FMATVEC Bibliothek [3] dar, die sowohl für die Bereitstellung von Speicherplatz für Vektoren und Matrizen als auch für die Bereitstellung von mathematischen Methoden der linearen Algebra zuständig ist. Zur Speicherallokation kann auf eigene Routinen (fmatvec::PoolAlloc) oder auf beliebige STL-Allocatoren[2] zurückgegriffen werden. Für die effiziente Bereitstellung mathematischer Methoden kann FMATVEC auf LAPACK/BLAS [2,11], ATLAS [1] und die INTELMKL [10] zugreifen. Sowohl ATLAS als auch die INTELMKL stellen sequentielle und parallele Algorithmen zur Verfügung. Ein Vergleich der verschiedenen Matrix/Vektor-Bibliotheken wird in Abschnitt 3.5 vorgestellt.

MBSim - Kernel und Zusatzmodule MBSIM entstand als Modellierungs- und Simulationsumgebung für starre nicht-glatte Mehrkörpersysteme. Im Laufe der Entwicklung wurde MBSIM um Module für flexible Körper, Elektronik, Hydraulik, Antriebsstränge, Ventiltriebsdynamik und Regelungstechnik erweitert.

Postprocessing Simulationsergebnisse werden in dem hierarchischen Datenformat HDF5 [5] gespeichert. Dazu stellt die HDF5SERIE Bibliothek [6] die nötigen Interfaces zu der HDF5 Bibliothek zur Verfügung. Die Simulationsergebnisse können in H5PLOTSERIE dargestellt bzw. in OPENMBV [15] visualisiert werden.

MDPCosim - parallele Co-Simulation Zusätzlich bietet MDPCOSIM [65] die Möglichkeit paralleler Co-Simulationen zwischen MBSim und weiteren spezifischen Simulationsumgebungen. Dazu gehören HYSIM [115] für hydraulische Systeme, KETSIM [88] für die Dynamik von Kettentrieben und MATLAB/SIMULINK [12].

Betriebssystem und Bibliotheken

Als Betriebssystem wird OpenSuse Linux 11.4 [17] verwendet. Alle Methoden sind in der objektorientierten Programmiersprache C++ umgesetzt. Als Compiler kommt GNU GCC [4] in der Version 4.5.1 zur Anwendung.

Alle parallelen Methoden sind auf Basis von OPENMP [16, 80] in der Version 3.1 implementiert. OPENMP stellt Compiler Direktiven zur Parallelisierung von C, C++ und Fortran Quellcode zur Verfügung.

1.2.2 Rechnerarchitektur

Als Hardware kommt eine herkömmliche Workstation zur Anwendung (shared-memory Hardwarearchitektur), die über zwei Intel Xeon X5650 (2.67 GHz) Prozessoren mit je 6 Kernen verfügt. Für die insgesamt 12 echten Kerne stehen 12 GB Arbeitsspeicher zur Verfügung (siehe Tabelle 1.1).

2 STL = [C++] Standard Template Library

Tabelle 1.1: Verwendete Hardware.

Prozessoren	2x Intel Xeon X5650 mit 2.67 GHz
Arbeitsspeicher	12 GB - 6x 2 GB 1333 MHz DDR3 ECC RDIMM
Festplatte	Western Digital Caviar Green 500 GB

Um die Rechenzeiten von sequentiellen und parallelen Simulationen vergleichen zu können, wurde die *Intel Turbo Boost* Technologie des Prozessors deaktiviert. Die *Intel Turbo Boost* Technologie ermöglicht es, dass ein einzelner Rechenkern der CPU hochgetaktet wird, wenn die restlichen Rechenkerne unbelastet sind.
Dies würde jedoch dazu führen, dass die sequentielle Simulation mit einer höheren Taktfrequenz abgearbeitet wird als die parallele Version, was die Aussagekraft des Vergleichs der Simulationen bzw. die Berechnung der Beschleunigungen beeinträchtigen würde.

2 Nicht-glatte Mehrkörperdynamik

Dieses Kapitel gibt einen Überblick über die Dynamik von nicht-glatten Mehrkörpersystemen. Es wird sowohl auf die Herleitung der Bewegungsgleichungen und auf die Formulierung der Nebenbedingungen eingegangen, als auch auf mögliche Lösungsverfahren.

Das wesentliche Unterscheidungsmerkmal zwischen der Theorie glatter und nichtglatter Mehrkörperdynamik kann in den Geschwindigkeitsverläufen der Systeme gesehen werden. In der nicht-glatten Mehrkörperdynamik können - im Gegensatz zur glatten Mehrkörperdynamik - Sprünge in den Geschwindigkeiten auftreten, die unter anderem von Stößen und schaltenden Reglern verursacht werden. Zur Lösung nicht-glatter dynamischer Systeme sind spezielle numerische Verfahren notwendig, die in Abschnitt 2.3 und in Kapitel 5 näher dargestellt werden.

Zum Verständnis der in dieser Arbeit dargestellten Parallelisierungs- und Integrationsmethoden ist sowohl ein Überblick über die Gleichungen der Mehrkörpertheorie als auch der softwareseitigen Implementierung notwendig. Dazu werden in Abschnitt 2.2 ausgehend von den Bewegungsgleichungen glatter Mehrkörpersysteme, die Bewegungsgleichungen von nicht-glatten Mehrkörpersystemen hergeleitet. Anschließend werden in Abschnitt 2.3 die wichtigsten mengenwertigen Kraftgesetze den zugehörigen regularisierten einwertigen Kraftgesetzen gegenübergestellt. In Abschnitt 2.4 wird die Formulierung der Kontaktkinematik in der Mehrkörperdynamik gezeigt und mögliche Lösungsverfahren genannt. Ein Überblick über die softwareseitige Implementierung der Bewegungsgleichungen wird in Abschnitt 2.5 gegeben, die sich an der Mehrkörpersimulationsumgebung MBSim [13] orientiert. Im letzten Abschnitt 2.6 des Kapitels werden weitere physikalische Bereiche gezeigt, die mit der gleichen Gleichungsstruktur wie die der Mehrkörperdynamik abgedeckt werden können. Für detailliertere Informationen wird auf die Dissertation von Förg [59] verwiesen.

2.1 Literaturüberblick

Im Folgenden wird ein Überblick über grundlegende Veröffentlichungen in der starren und flexiblen nicht-glatten Mehrkörperdynamik gegeben. Dabei werden im ersten Abschnitt die Werke aufgezeigt, die sich im Wesentlichen mit der Theorie der nichtglatten Mehrkörpersysteme beschäftigen und im zweiten Abschnitt werden industrielle Anwendungsbeispiele aufgezeigt, die am Lehrstuhl für Angewandte Mechanik bearbeitet wurden.

Zur Theorie der nicht-glatten Mehrkörpersysteme sei vor allem auf die Werke von Brogliato [32], Brogliato et al. [33], Acary und Brogliato [19], Leine

und van de Wouw [99], Moreau [105], Stewart [137], Schiehlen [125] und Thomsen und True [141] hingewiesen. Pfeiffer [114, 117] und Glocker [70] präsentieren in ihren Werken eine ausführliche Herleitung der nicht-glatten Mehrkörpertheorie, die an einer Vielzahl von akademischen und industriellen Beispielen angewendet wird. Ulbrich zeigt in [145,146] die Übertragung der nicht-glatten Mechanik auf die Maschinendynamik, insbesondere auf die Simulation von variablen Ventiltrieben. Förg [59], Stiegelmeyr [138] und Studer [139] beschäftigen sich in ihren Dissertationen mit effizienten numerischen Verfahren zur Integration der Bewegungsgleichungen und zur Lösung der mengenwertigen Nebenbedingungen.
Im Bereich der flexiblen Mehrkörpertheorie können die Werke von Bremer und Pfeiffer [31] und Shabana [133] herangezogen werden. Funk [66] und Zander [151] beschäftigen sich in ihren Dissertationen mit der hybriden nicht-glatten Mehrkörpertheorie.

Geier [68], Schindler [126] und Cebulla et al. [35] wenden die hybride nicht-glatte Mehrkörpertheorie auf die Simulation von CVT-Getrieben (Continuously Variable Transmission) an. Engelhardt [50], Huber und Ulbrich [84] und Schneider et al. [132] zeigen die Vorteile der nicht-glatten Mehrkörpertheorie zur Simulation von Ventiltrieben. Huber et al. [83] modellieren die Kontakte zwischen den Windungen elastischer Ventilfedern mittels nicht-glatter Mechanik. Ginzinger [69] wendet die Theorie zur Simulation und Regelung von Anstreifvorgängen von Rotoren an.

2.2 Bewegungsgleichungen

Mehrkörpersysteme können in massebehaftete Körper und masselose Verbindungen eingeteilt werden. Die Bewegungsgleichungen von ein- und zweiseitig gebundenen mechanischen Systemen können grundlegend auf zwei Wegen hergeleitet werden. Analytische Methoden wie Lagrange I/II gehen von Energiebetrachtungen aus [114], wohingegen synthetische Methoden wie der Newton-Euler-Formalismus [145] von Impuls- und Drallsatz ausgehen. Beide Wege führen zu identischen Bewegungsgleichungen, wobei der Aufwand für die Herleitung unterschiedlich sein kann.

Jedes beliebige glatte mechanische System mit n_i Körpern kann mit den Gleichungen (2.1) beschrieben werden.

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{u}} = \boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t) \tag{2.1a}$$

$$\dot{\boldsymbol{q}} = \boldsymbol{T}(\boldsymbol{q})\boldsymbol{u} \tag{2.1b}$$

Der generalisierte Systemzustand $\boldsymbol{z}$ ist durch die n_i Bewegungsgleichungen der Einzelkörper festgelegt. Die Bewegung des Systems wird dabei über die generalisierten

Lagen $\boldsymbol{q}$ und verallgemeinerten Geschwindigkeiten $\boldsymbol{u}$ (2.2) beschrieben.

$$\boldsymbol{q} = \begin{pmatrix} \boldsymbol{q}^{(1)} \\ \vdots \\ \boldsymbol{q}^{(n_i)} \end{pmatrix} \in \mathbb{R}^{n_q} \qquad \boldsymbol{u} = \begin{pmatrix} \boldsymbol{u}^{(1)} \\ \vdots \\ \boldsymbol{u}^{(n_i)} \end{pmatrix} \in \mathbb{R}^{n_u} \tag{2.2}$$

Somit besitzt der Systemzustand

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{q} \\ \boldsymbol{u} \end{pmatrix} \in \mathbb{R}^{n_z}$$

$n_z = n_q + n_u$ Zustandsgrößen.

Der Zusammenhang zwischen den zeitlichen Ableitungen der generalisierten Lagen $\dot{\boldsymbol{q}}$ und den verallgemeinerten Geschwindigkeiten $\boldsymbol{u}$ wird mit der Abbildungsmatrix

$$\boldsymbol{T}(\boldsymbol{q}) \in \mathbb{R}^{n_q \times n_u} \tag{2.3}$$

beschrieben. Der Vorteil der Einführung der Abbildungsmatrix $\boldsymbol{T}(\boldsymbol{q})$ liegt darin, dass die Winkelgeschwindigkeiten der Körper im Allgemeinen nicht integrierbar sind [59] und es in der Regel von Vorteil ist, die rotatorischen räumlichen Lagen in Parametrisierungen wie den Kardan-, Euler- oder Resalwinkeln [114] darzustellen, jedoch die Winkelgeschwindigkeiten im körperfesten Koordinatensystem des einzelnen Körpers zu beschreiben. Dies hat den Vorteil, dass der Trägheitstensor eines Starrkörpers im körperfesten Koordinatensystem konstant ist.

Die Projektion der Massenmatrizen $\boldsymbol{M}_i$, der Vektoren der rechten Seite $\boldsymbol{h}_i$ und der Kraftrichtungsmatrizen $\boldsymbol{W}_i$ aller Elemente des Mehrkörpersystems in die Richtung der generalisierten Koordinaten erfolgt mit den Jacobi-Matrizen $\boldsymbol{J}_i$.

Der Vektor

$$\boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t) = \sum_{i=1}^{n_i} \boldsymbol{J}_i^T \boldsymbol{h}_i \tag{2.4}$$

beinhaltet alle glatten internen, externen und gyroskopischen Kräfte. An dieser Stelle sei explizit darauf hingewiesen, dass in dieser Beschreibungsform folglich auch alle regularisierten ein- und zweiseitigen Bindungen im Vektor $\boldsymbol{h}$ enthalten sind.

Die positiv definite symmetrische Matrix

$$\boldsymbol{M}(\boldsymbol{q}) = \sum_{i=1}^{n_i} \boldsymbol{J}_i^T \boldsymbol{M}_i \boldsymbol{J}_i \tag{2.5}$$

stellt die von den generalisierten Lagen $\boldsymbol{q}$ abhängige Massenmatrix dar.

Die Darstellung (2.1) setzt Beschleunigungen und Kräfte in Beziehung zueinander und ist somit nicht dafür geeignet, Systeme mit Stößen zu beschreiben. Aus diesem Grund wird zur Beschreibung von ein- und zweiseitig gebundenen diskontinuierlichen

Systemen auf die *Maßdifferentialgleichung* [105, 114] übergegangen (2.6).

$$\boldsymbol{M}(\boldsymbol{q})d\boldsymbol{u} = \boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t)dt + \boldsymbol{W}(\boldsymbol{q})d\boldsymbol{\Lambda} \tag{2.6a}$$

$$\dot{\boldsymbol{q}} = \boldsymbol{T}(\boldsymbol{q})\boldsymbol{u} \tag{2.6b}$$

$$(\boldsymbol{\Lambda},\boldsymbol{q},\boldsymbol{u},t) \in \mathcal{N}$$

Die Menge aller Stoßzeitpunkte t_k wird als Menge $\mathcal{M}_\mathcal{S}$ definiert. Der Vektor $d\boldsymbol{\Lambda}$ beschreibt die Impulse der mengenwertigen Kraftgesetze (Menge $\mathcal{N}$), die durch die Kraftrichtungsmatrix

$$\boldsymbol{W}(\boldsymbol{q}) = \sum_{i=1}^{n_i} \boldsymbol{J}_i^T \boldsymbol{W}_i \tag{2.7}$$

in die Richtung der verallgemeinerten Geschwindigkeiten projiziert werden. Das Maß für die Beschleunigungen

$$d\boldsymbol{u} = \dot{\boldsymbol{u}}dt + (\boldsymbol{u}^+ - \boldsymbol{u}^-)d\eta \tag{2.8}$$

wird in einen LEBESGUE-integrierbaren Anteil $\dot{\boldsymbol{u}}dt$ für den kontinuierlichen Teil der Bewegung und einen diskreten Anteil $(\boldsymbol{u}^+ - \boldsymbol{u}^-)d\eta$ für alle Diskontinuitätszeitpunkte $t_k \in \mathcal{M}_\mathcal{S}$ aufgeteilt. Während den Diskontinuitätszeitpunkten $t_k \in \mathcal{M}_\mathcal{S}$ besteht der zweite Anteil der Gleichung (2.8) aus der Differenz des linken und rechten Geschwindigkeitsgrenzwertes $(\boldsymbol{u}^+ - \boldsymbol{u}^-)$, die mit der Summe der Delta-Distributionen bezüglich der Diskontinuitätszeitpunkte $t_k \in \mathcal{M}_\mathcal{S}$ gewichtet wird [114]:

$$d\eta = \sum_{t_k} d\delta_k \,, \tag{2.9a}$$

$$\int_{\{t_k\}} (\boldsymbol{u}^+ - \boldsymbol{u}^-)d\delta_k = \begin{cases} \boldsymbol{u}^+ - \boldsymbol{u}^- & \text{für } t = t_k \\ 0 & \text{für } t \neq t_k \end{cases}. \tag{2.9b}$$

Analog zu dem Maß für die Beschleunigungen $d\boldsymbol{u}$ wird auch das Maß für die Impulse

$$d\boldsymbol{\Lambda} = \boldsymbol{\lambda}dt + \boldsymbol{\Lambda}d\eta \tag{2.10}$$

in einen LEBESGUE-integrierbaren Anteil $\boldsymbol{\lambda}dt$ und einen diskreten Anteil $\boldsymbol{\Lambda}d\eta$ aufgeteilt.

Nun kann Gleichung (2.6) unter Berücksichtigung von Gleichung (2.9) integriert werden und stellt in dieser Form die Grundlage der nicht-glatten Mehrkörperdynamik dar.

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{u}} = \boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t) + \boldsymbol{W}(\boldsymbol{q})\boldsymbol{\lambda} \qquad \forall t \notin \mathcal{M}_\mathcal{S} \tag{2.11a}$$

$$\boldsymbol{M}_k \left(\boldsymbol{u}_k^+ - \boldsymbol{u}_k^-\right) = \boldsymbol{W}_k \boldsymbol{\Lambda}_k \qquad \forall t \in \mathcal{M}_\mathcal{S} \tag{2.11b}$$

$$\dot{\boldsymbol{q}} = \boldsymbol{T}(\boldsymbol{q})\boldsymbol{u}$$

$$(\boldsymbol{\Lambda},\boldsymbol{\lambda},\boldsymbol{q},\boldsymbol{u},t) \in \mathcal{N}$$

Gleichung (2.11a) gilt während glatten Phasen, wohingegen Gleichung (2.11b) während Diskontinuitätszeitpunkten $t_k \in \mathcal{M}_S$ gültig ist. Kapitel 5 beschäftigt sich mit numerischen Lösungsmöglichkeiten.

Für die Bestimmung der unbekannten Größen $\boldsymbol{\Lambda}$ und $\boldsymbol{\lambda}$ sind zusätzliche Nebenbedingungen notwendig, auf die im nächsten Abschnitt eingegangen wird.

2.3 Kraftgesetze

Kraftgesetze können in der Mechanik grundsätzlich in *einwertige* und *mengenwertige* Kraftgesetze eingeteilt werden. Einwertige Kraftgesetze sind explizit vom Systemzustand ($\boldsymbol{q}$,$\boldsymbol{u}$,t) abhängig und können somit direkt ausgewertet werden. Mengenwertige Kraftgesetze können nur als mengenwertige Nebenbedingungen angegeben werden und benötigen somit spezielle numerische Verfahren für ihre Lösung. Es haben sich im Wesentlichen zwei mathematische Formulierungen als geeignet zur Beschreibung der mengenwertigen Kraftgesetze erwiesen: Einerseits die Formulierung als *Lineare Komplementaritätsprobleme* und zum anderen die Formulierung mittels Projektionsfunktionen [130], einer mathematischen Methodik aus der konvexen Analysis. Die zweite Methode wird am Ende dieses Abschnitts näher erläutert.

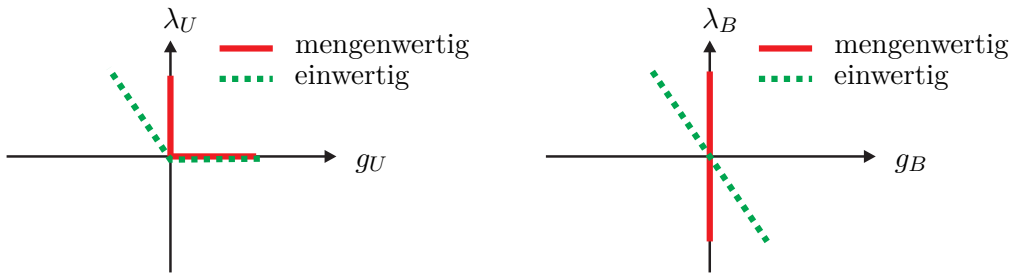
Der Grenzfall $c \to \infty$ beschreibt in der Mechanik den Übergang von einwertigen zu mengenwertigen Kraftgesetzen. Für die Verwendung von regularisierten ein- und

Typ	**Charakteristika**
einwertig	- explizit auswertbar - physikalisch motiviert - steife Differentialgleichungen - oft unbekannte Parameter (c,d)
mengenwertig	- kein Eindringen der Kontaktpartner - COULOMB-Reibung modellierbar - spezielle Verfahren zur Lösung notwendig

Tabelle 2.1: Charakteristika einwertiger und mengenwertiger Kraftgesetze.

zweiseitigen Bindungen ist die Bestimmung der Kontaktsteifigkeit c und der Kontaktdämpfung d notwendig, die beispielsweise nach der HERTZSCHEN Pressung berechnet werden können. Dies führt einerseits dazu, dass die Parameter c und d physikalisch motiviert bestimmt werden können, jedoch entstehen in der Regel steife Differentialgleichungen, die während der numerischen Integration spezielle zeitaufwendige Verfahren oder niedrigere Zeitschrittweiten erfordern (siehe Kapitel 5).

Vor allem bei der Simulation von großen Systemen mit einer hohen Anzahl an ein- und zweiseitigen Bindungen wird daher die Verwendung von mengenwertigen Kraftgesetzen bevorzugt. Durch die Verwendung von mengenwertigen Kraftgesetzen können steife Differentialgleichungen vermieden werden und zusätzlich ist es möglich



(a) Einseitige Bindung. (b) Zweiseitige Bindung.

Abbildung 2.1: Einseitige und zweiseitige Bindungen.

Stöße, Haft-Gleit-Übergänge oder auch schaltende Regler zu modellieren, ohne die Kraftgesetze zu regularisieren.

Den Vorteilen gegenüber steht jedoch, dass zur Lösung mengenwertiger Kraftgesetze spezielle numerische Verfahren benötigt werden. Tabelle 2.1 fasst die Charakteristika einwertiger und mengenwertiger Kraftgesetze kompakt zusammen. Ein ausführlicher Vergleich der beiden Klassen kann in [62] gefunden werden, worin Förg et al. einwertige und mengenwertige Formulierungen anhand eines Ventiltriebs vergleichen.

In der Mehrkörperdynamik spielen im Wesentlichen drei Arten von Kraftgesetzen eine Rolle: Zweiseitige Bindungen, einseitige Bindungen (Kontakte) und Reibgesetze. Vorerst werden nur glatte Bewegungsabschnitte betrachtet, also Abschnitte in denen keine Stöße auftreten. Eine zweiseitige mengenwertige Bindung (Abbildung 2.1(b)) kann in der Form

$$g_B = 0, \qquad \lambda_B \in \mathbb{R} \tag{2.12}$$

dargestellt werden. Der mengenwertige Charakter des Kraftgesetzes besteht darin, dass für $g_B = 0$ eine unendliche Anzahl möglicher Bindungskräfte λ_B existiert. Zur Lösung des Kraftgesetzes sind die Bewegungsgleichungen des Mehrkörpersystems notwendig.
Einseitige mengenwertige Bindungen, auch Kontakte genannt (Abbildung 2.1(a)), können mittels der Signorini-Fichera Bedingung

$$g_U \geq 0, \quad \lambda_U \geq 0, \quad g_U \, \lambda_U = 0 \tag{2.13}$$

beschrieben werden. In den Gleichungen (2.12) und (2.13) stellt $g_{B,U}$ den Normalabstand der beiden Kontaktpartner und $\boldsymbol{\lambda}_{B,U}$ die zugehörige Bindungskraft dar.

Coulomb Reibung (Abbildung 2.2) kann mathematisch nach Gleichung (2.14) dargestellt werden, worin $\dot{\boldsymbol{g}}_T \in \mathbb{R}^2$ die Tangentialgeschwindigkeiten, μ den Haftbeiwert, λ_N die Normalkraft und $\boldsymbol{\lambda}_T \in \mathbb{R}^2$ die Tangentialkräfte beschreiben.

$$\begin{aligned} \dot{\boldsymbol{g}}_T = \boldsymbol{0} \quad &\Rightarrow \quad |\boldsymbol{\lambda}_T| \;\; \leq \mu \, |\lambda_N| \\ \dot{\boldsymbol{g}}_T \neq \boldsymbol{0} \quad &\Rightarrow \quad \boldsymbol{\lambda}_T \;\; = -\tfrac{\dot{\boldsymbol{g}}_T}{|\dot{\boldsymbol{g}}_T|} \, \mu \, |\lambda_N| \end{aligned} \tag{2.14}$$

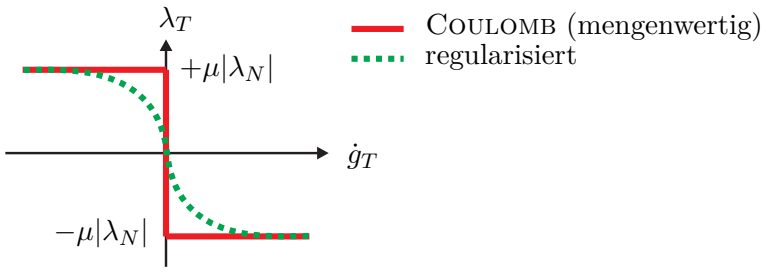


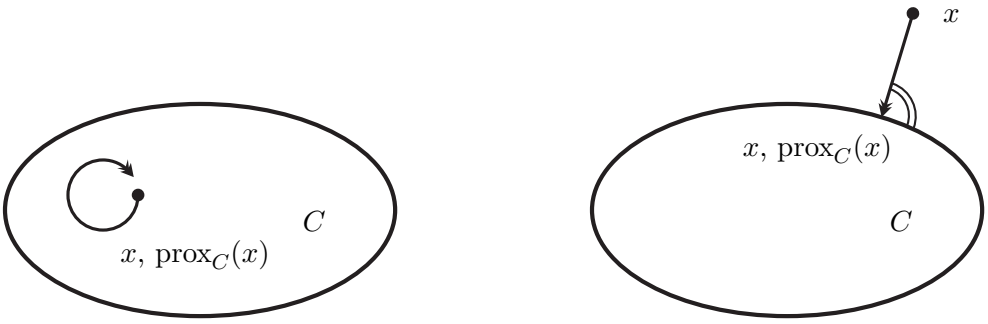
Abbildung 2.2: Ebene COULOMB Reibung und ebene regularisierte Reibung.

Stöße beeinflussen die Geschwindigkeiten aller Körper nach dem Stoßvorgang. Aus diesem Grund müssen Stoßgesetze auf Geschwindigkeitsebene definiert werden. Dazu wird in den Gleichungen (2.12), (2.13) (insofern der Kontakt geschlossen ist) und (2.14) g durch $\dot{g}^+$ und λ durch Λ ersetzt. Es wird im Folgenden von vollplastischen Stößen ($\varepsilon = 0$) ausgegangen.

Die Kraftgesetze (2.12), (2.13) und (2.14) mögen aus mathematischer Sicht die Problemstellung eindeutig beschreiben, jedoch lassen sie sich in dieser Form nicht effizient numerisch lösen. Deshalb werden die Gesetze zu sogenannten Projektionsfunktionen umgeschrieben, einer Methodik aus der konvexen Analysis [59, 63]. Die Projektionsfunktionen sind nach Gleichung (2.15) definiert [122, 123] und können wie in Abbildung 2.3 anschaulich dargestellt werden.

$$\mathbf{prox}_C(\boldsymbol{x}) = \arg\min_{\boldsymbol{x}^* \in C} \|\boldsymbol{x} - \boldsymbol{x}^*\| \tag{2.15}$$

Unter Verwendung der euklidischen Norm $||\cdot||$ kann Gleichung (2.15) als diejenige Abbildungsfunktion betrachtet werden, die einen gegebenen Punkt an den nächstgelegenen Punkt innerhalb der konvexen Menge C projiziert. Entsprechend wird ein Punkt, der sich bereits in der Menge C befindet, auf sich selbst abgebildet (Abbildung 2.3(a)) und ein Punkt außerhalb der Menge C wird auf dem kürzesten Weg auf den Rand der Menge C projiziert (Abbildung 2.3(b)).



(a) Argument innerhalb. (b) Argument außerhalb.

Abbildung 2.3: Grafische Darstellung der Projektionsfunktionen [59].

Mit Gleichung (2.15) können die Gleichungen (2.12), (2.13) und (2.14) zu Gleichung (2.16) umgeschrieben werden.

$$\lambda_B = \mathrm{prox}_{C_B}(\lambda_B - r_B\, g_B)\ , \qquad \Lambda_B = \mathrm{prox}_{C_B}(\Lambda_B - r_B\, \dot{g}_B^+) \tag{2.16a}$$

$$\lambda_U = \mathrm{prox}_{C_U}(\lambda_U - r_U\, g_U)\ , \qquad \Lambda_U = \mathrm{prox}_{C_U}(\Lambda_U - r_U \dot{g}_U^+) \tag{2.16b}$$

$$\boldsymbol{\lambda}_T = \mathbf{prox}_{C_T(\lambda_N)}(\boldsymbol{\lambda}_T - r_T\, \dot{\boldsymbol{g}}_T)\ , \qquad \boldsymbol{\Lambda}_T = \mathbf{prox}_{C_T(\Lambda_N)}(\boldsymbol{\Lambda}_T - r_T\, \dot{\boldsymbol{g}}_T^+) \tag{2.16c}$$

$$\text{mit } C_B = \mathbb{R}, \quad C_U = \{x \,|\, x \geq 0\}, \quad C_T(y) = \{\boldsymbol{x} \,|\, \|\boldsymbol{x}\|_2 \leq \mu\, |y|\},\ y \in \mathbb{R}$$

Die unabhängigen Hilfsparameter $r_B > 0, r_U > 0$ und $r_T > 0$ können mathematisch gesehen beliebig gewählt werden, beeinflussen jedoch maßgeblich die Konvergenzgeschwindigkeit der numerischen Lösungsverfahren [63].

2.4 Kontaktkinematik

Wie in Abschnitt 2.2 genannt, können Mehrkörpersysteme in massebehaftete Körper und masselose Verbindungselemente eingeteilt werden. Aufgabe der Kontaktkinematik ist es, die möglichen Kontaktpunkte und die zugehörigen Kraftrichtungen zweier in Kontakt stehender Körper zu ermitteln. In einem zweiten Schritt können dann mittels entsprechender Kraftgesetze (Abschnitt 2.3) die zugehörigen Kontaktreaktionen berechnet werden. Zur Beschreibung der Kinematik werden jedem Körper eine

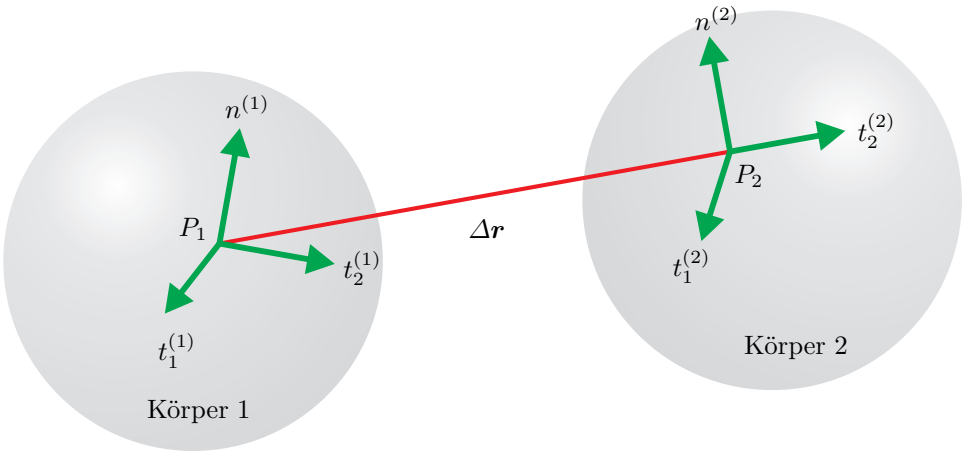


Abbildung 2.4: Kontaktkinematik zwischen zwei Kugelkonturen.

oder mehrere Konturen zugeordnet. Die Konturen werden durch den Ortsvektor

$$\boldsymbol{r} = \boldsymbol{r}(\boldsymbol{q}, \boldsymbol{s}) \tag{2.17}$$

definiert. Aus dem Ortsvektor ergeben sich die nach außen zeigende Normale

$$\boldsymbol{n} = \boldsymbol{n}(\boldsymbol{q},\boldsymbol{s}) \tag{2.18}$$

und die beiden Tangenten

$$\begin{aligned} \boldsymbol{t}_1 &= \boldsymbol{t}_1(\boldsymbol{q},\boldsymbol{s}) \\ \boldsymbol{t}_2 &= \boldsymbol{t}_2(\boldsymbol{q},\boldsymbol{s}) \\ \boldsymbol{\Gamma} &= \left(\boldsymbol{t}_1, \boldsymbol{t}_2\right) . \end{aligned} \tag{2.19}$$

Die notwendigen Bedingungen sollen am Beispiel eines räumlichen Punktkontakts zwischen zwei Kugelkonturen (tiefgestellter Index $_1$ und $_2$) gezeigt werden. Gesucht sind die beiden Punkte P_1 und P_2 auf den kugelförmigen Konturen der Körper, die einen minimalen Abstand $g = |\boldsymbol{r}_2 - \boldsymbol{r}_1|$ zueinander haben. Dies kann mathematisch nach Gleichung (2.20) formuliert werden.

$$\boldsymbol{\Gamma}_1^T(\boldsymbol{s}_1)\,(\boldsymbol{r}_2(\boldsymbol{s}_2) - \boldsymbol{r}_1(\boldsymbol{s}_1)) = \boldsymbol{0} \tag{2.20a}$$

$$\boldsymbol{\Gamma}_2^T(\boldsymbol{s}_2)\,(\boldsymbol{r}_2(\boldsymbol{s}_2) - \boldsymbol{r}_1(\boldsymbol{s}_1)) = \boldsymbol{0} \tag{2.20b}$$

Gleichung (2.20) stellt jedoch nur eine notwendige, jedoch keine hinreichende Bedingung für einen Kontaktpunkt dar. Gleichung (2.20) kann mehrere mögliche Lösungen liefern, aus denen diejenige ausgesucht werden muss, die einen minimalen Abstand $g = g_{\min}$ der Kontaktpartner liefert.

Nun können mit

$$g_n = \boldsymbol{n}_1^T(\boldsymbol{s}_1)\,(\boldsymbol{r}_2(\boldsymbol{s}_2) - \boldsymbol{r}_1(\boldsymbol{s}_1)) \tag{2.21}$$

der Normalabstand und mit

$$\begin{pmatrix} \dot{g}_n \\ \dot{\boldsymbol{g}}_t \end{pmatrix} = \begin{pmatrix} \boldsymbol{n}^T \\ \boldsymbol{\Gamma}^T \end{pmatrix} (\boldsymbol{v}_2(\boldsymbol{s}_2) - \boldsymbol{v}_1(\boldsymbol{s}_1)) \tag{2.22}$$

die zugehörigen Geschwindigkeiten in Normal- und Tangentialrichtung des entsprechenden Körpers berechnet werden.

Der numerische Aufwand zur Lösung von Gleichung (2.20) hängt wesentlich von den beiden beteiligten Konturen ab. Für einfache geometrische Primitive kann die Gleichung in den meisten Fällen analytisch gelöst werden. Viele Konturen in der Mehrkörperdynamik können jedoch nicht analytisch dargestellt werden. An dieser Stelle seien stellvertretend die Konturen der Nocken von Nockenwellen genannt, die in der Regel nur als Punktewolke gegeben sind. Diese Punktewolke wird meist mit mindestens zweimal stetig differenzierbaren Splines (z.B. kubischen Splines) interpoliert. Zur Lösung der Gleichung (2.20) werden in diesem Fall numerische Lösungsverfahren wie das Newton-Verfahren oder das Regula-Falsi-Verfahren [60] herangezogen.

Der unterschiedliche numerische Aufwand zur Lösung der Gleichung (2.20) resultiert

in einem entsprechenden zeitlichen Aufwand, der sich je nach Konturpaarung um mehrere Zehnerpotenzen unterscheiden kann.

2.5 Konzept der Implementierung

In den vorangegangen Abschnitten wurde ein Überblick über die theoretischen Grundlagen der nicht-glatten Mehrkörperdynamik gegeben. Zur Erklärung der in dieser Arbeit diskutierten Parallelisierungskonzepte (siehe Kapitel 4) ist es notwendig - zusätzlich zu den theoretischen Grundlagen - auch das Konzept der Implementierung in eine Mehrkörpersimulationsumgebung zu erläutern. Da alle diskutierten Methoden in der Mehrkörpersimulationsumgebung MBSim [13, 128] implementiert sind, wird das Implementierungskonzept von MBSim herangezogen.

Zur Erklärung wird ein einzelner Integrationsschritt eines halb-expliziten Time-Stepping Integrationsverfahrens [59] herangezogen, der in Algorithmus 2.1 zusammengefasst ist.

Neben der mathematischen Formulierung des Integrationsverfahrens sind zusätzlich die notwendigen Schritte innerhalb der Simulationsumgebung MBSim angegeben. Die Abkürzungen der einzelnen Berechnungen sind jeweils in Klammern angegeben, sodass im Folgenden beispielsweise unter **updateg** die Neuberechnung aller im Mehrkörpersystem vorhandenen Kontaktkinematiken zu verstehen ist. An dieser Stelle sei bereits darauf hingewiesen, dass die in Kapitel 4 vorgestellten Parallelisierungsmethoden genau an diesen Neuberechnungsprozessen ansetzen.

Im **1. Schritt** werden die neuen generalisierten Lagen $\boldsymbol{q}^{l+1}$ des Systems aus den Größen des letzten Integrationsschrittes berechnet.

Im **2. Schritt** werden alle im System vorhandenen Kontaktkinematiken $\boldsymbol{g}^{l+1}$ ausgewertet. Dazu gehört die Auswertung der Körperkinematiken (**updateSDV**), die Neuberechnung aller Kontaktabstände $\boldsymbol{g}^{l+1}$ (**updateg**) und die Berechnung aller Kontaktabstandsgeschwindigkeiten $\dot{\boldsymbol{g}}^{l+1}$ (**updategd**).

Im **3. Schritt** werden die neuen verallgemeinerten Geschwindigkeiten unter Berücksichtigung aktiver mengenwertiger Nebenbedingungen (Indexmenge $_a$) berechnet. Dazu werden die Jacobi-Matrizen $\boldsymbol{J}^{l+1}$ (**updateJacobians**)[1], der Vektor der einwertigen internen, externen und gyroskopischen Kräfte $\hat{\boldsymbol{h}}^{l+1}$ (**updateh**), die Massenmatrix $\boldsymbol{M}^{l+1}$ (**updateM**) und die Kraftrichtungsmatrix $\boldsymbol{W}^{l+1}$ (**updateW**) neu berechnet. Zusätzlich werden die Kraftgesetze der aktiven mengenwertigen Bindungen gelöst (**solveImpacts**). Die Reihenfolge der Auswertungen ergibt sich daraus, welche Abhängigkeiten zwischen den Größen bestehen[2].

1 Die Jacobi-Matrizen dienen der Projektion von $\boldsymbol{h}$, $\boldsymbol{M}$ und $\boldsymbol{W}$ in die Richtung der generalisierten Koordinaten (siehe Abschnitt 2.2).

2 Zum Beispiel müssen die Jacobi-Matrizen $\boldsymbol{J}^{l+1}$ bereits neu berechnet sein, bevor der Vektor der rechten Seite $\hat{\boldsymbol{h}}^{l+1}$ neu berechnet wird, da er nach Gleichung 2.4 die Jacobi-Matrizen zur Berechnung benötigt.

Algorithmus 2.1 Halb-explizites Time-Stepping Integrationsverfahren

1. Berechnung der neuen generalisierten Lagen:

$$\boldsymbol{q}^{l+1} = \boldsymbol{q}^{l} + \boldsymbol{T}^{l}\,\boldsymbol{u}^{l}\,\Delta t$$

2. Berechnung aller Kontaktkinematiken:

   Notwendige Neuberechnungen innerhalb der Simulationsumgebung:
   - Aktualisierung der Zeit: $t^{l+1} = t^{l} + \Delta t$
   - Neuberechnung der Körperkinematik (**updateSDV**)
   - Neuberechnung aller Kontaktkinematiken:
     - Kontaktabstände (**updateg**)
     - Kontaktgeschwindigkeiten (**updategd**)

$$\boldsymbol{g}^{l+1} = \boldsymbol{g}(\boldsymbol{q}^{l+1},t^{l+1})$$

3. Berechnung der neuen verallgemeinerten Geschwindigkeiten unter Berücksichtigung aktiver mengenwertiger Nebenbedingungen:

   Notwendige Neuberechnungen innerhalb der Simulationsumgebung:
   - Neuberechnung der JACOBI-Matrizen $\boldsymbol{J}$ (**updateJacobians**)
   - Neuberechnung von $\hat{\boldsymbol{h}}$ (**updateh**)
   - Neuberechnung von $\boldsymbol{M}$ (**updateM**)
   - Neuberechnung von $\boldsymbol{W}$ (**updateW**)
   - Lösen der aktiven mengenwertigen Bindungen (**solveImpacts**)

$$\boldsymbol{u}^{l+1} = \boldsymbol{u}^{l} + \left(\boldsymbol{M}^{l+1}\right)^{-1} \left(\hat{\boldsymbol{h}}^{\,l+1}\,\Delta t + \boldsymbol{W}_{a}^{l+1}\,\boldsymbol{\Lambda}_{a}^{l+1}\right), \tag{2.23a}$$

$$\dot{\boldsymbol{g}}_{a}^{l+1} = \dot{\boldsymbol{g}}_{a}(\boldsymbol{q}^{l+1},\boldsymbol{u}^{l+1},t^{l+1}), \tag{2.23b}$$

$$(\boldsymbol{\Lambda}_{a}^{l+1},\dot{\boldsymbol{g}}_{a}^{l+1}) \in \mathcal{N} \tag{2.23c}$$

$$\text{mit } \boldsymbol{M}^{l+1} = \boldsymbol{M}(\boldsymbol{q}^{l+1}) \text{ und } \hat{\boldsymbol{h}}^{\,l+1} = \boldsymbol{h}(\boldsymbol{u}^{l},\boldsymbol{q}^{l+1},t^{l+1}) \tag{2.23d}$$

Nun kann die Berechnung des nächsten Zeitschrittes wieder mit dem **1. Schritt** beginnen.

Analog zur Theorie der Mehrkörpersysteme, die alle Bestandteile eines Mehrkörpersystems in massebehaftete Körper und masselose Bindungen einteilt, wird auch softwareseitig zwischen OBJECTS (massebehaftete Körper) und LINKS (masselose Bindungen) unterschieden.

2.6 Erweiterung um weitere physikalische Bereiche

In den Abschnitten 2.2 bis 2.5 wurden die Herleitung und die softwareseitige Implementierung ein- und zweiseitig gebundener nicht-glatter Mehrkörpersysteme beschrieben. Die resultierende Form der Bewegungsgleichungen (2.6) gilt jedoch nicht nur für die Mehrkörperdynamik, sondern auch für weitere physikalische Bereiche. Pfeiffer et al. [116], Borchsenius [29] und Schneider und Krüger [131] zeigen die Anwendung der nicht-glatten Theorie auf die Simulation hydraulischer Systeme. Glocker [71] wendet die Theorie auf die Simulation elektrischer Systeme an.

3 Parallelisierung

3.1 Literaturüberblick

Die Thematik der Parallelisierung wird in allen Bereichen der Softwareentwicklung aufgegriffen. Auch im Bereich der Simulation von dynamischen Systemen wurde der Gedanke, die Simulationen auf parallelen Rechnern zu implementieren, sehr früh aufgegriffen. Bereits ab Mitte der achtziger Jahre[1] waren die ersten Werke zur parallelen Berechnung von dynamischen Systemen zu finden.

Zur Eingliederung vorhandener Arbeiten ist die Art der Simulationen sehr genau zu klassifizieren. In dieser Arbeit werden hauptsächlich Methoden zur parallelen Berechnung von starren und flexiblen Mehrkörpersystemen betrachtet. Parallelisierung kann auch in allen anderen Bereichen der numerischen Simulation angewandt werden. Stellvertretend seien hier die CFD-Simulation[2] [106] und die FE-Simulation[3] genannt.

Die dynamische Simulation von mechanischen Systemen kann grundlegend in die starre Mehrkörperdynamik, die flexible Mehrkörperdynamik und die Strukturdynamik eingeteilt werden. Die Kombination von starrer und flexibler Mehrkörperdynamik wird im Allgemeinen als hybride Mehrkörperdynamik bezeichnet.
Im Bereich der Strukturdynamik seien die FE-Methoden genannt, die sich im Gegensatz zur Mehrkörperdynamik sehr gut für parallele Implementierungen eignen. Nähere Informationen zur parallelen Berechnung von FE-Strukturen sind in den Werken von Gee et al. [67] und Nölting [111] zu finden. Speziell FE-Methoden eignen sich gut zur parallelen Berechnung auf GPUs[4]. Hierzu seien die Werke [45, 46] von Dick et al. genannt. GPUs werden im Gegensatz zu CPUs mit mehreren Rechenkernen für die sogenannte *massive* Parallelisierung genutzt. Die Gleichungsstruktur von FE-Methoden lässt sich sehr gut auf die Architektur von GPU's abbilden, die in der Regel über mehrere hundert Rechenkerne verfügen. Nutzt man GPUs zur Berechnung von Systemen, die nur wenige Rechenkerne benötigen und sequentielle Codeanteile beinhalten, so werden GPUs kaum ausgelastet. In Verbindung mit den sehr geringen Cache-Größen der GPU's führt dies dazu, dass CPUs für derartige Programme wesentlich besser geeignet sind.

Nahezu alle Algorithmen und Implementierungen zur parallelen Berechnung von Mehrkörpersystemen stammen aus dem Bereich der Roboterdynamik. Schon sehr

1 Im Jahr 1986 wurde der erste shared-memory Parallelrechner Sequent Balance 8000 mit acht CPUs vorgestellt.
2 CFD (computational fluid dynamics): Numerische Strömungsmechanik
3 FE: Finite Elemente
4 GPU (Graphics Processing Unit): Grafikprozessor

früh wurde das Potential erkannt, parallele Rechnerarchitekturen sowohl zur parallelen Berechnung der inversen Dynamik als auch zur parallelen Berechnung der Vorwärts-Dynamik von Robotersystemen heranzuziehen. Der Ursprung der parallelen Methoden in der Robotordynamik kann dadurch erklärt werden, dass vor allem bei der inversen Dynamik, der Bahnplanung und der Regelung von Robotern die Thematik der Echtzeitfähigkeit eine entscheidende Rolle spielt.

Im Folgenden werden die Arbeiten in drei wesentliche Gruppen eingeteilt. Arbeiten zur Berechnung der inversen Dynamik von Robotersystemen, Arbeiten zur Berechnung der Vorwärts-Dynamik von Robotern und Arbeiten zur allgemeinen Berechnung von Mehrkörpersystemen.

Lathrop [95] stellt in seiner Arbeit zwei Algorithmen zur parallelen Berechnung der inversen Dynamik von Robotern vor. Seine Algorithmen basieren auf einem rekursiven Newton-Euler Formalismus, die auf speziellen Hardware-Architekturen (systolische Pipeline Architekturen[5], VLSI) implementiert sind, sodass eine einfache Portierung auf beliebige Mehrkern-Systeme nicht möglich ist. Die Auswertereihenfolge des rekursiven Newton-Euler Formalismus wird durch die Datenabhängigkeiten im Simulationsmodell bestimmt. In seiner Arbeit werden keine Beispiele und keine erzielten Beschleunigungen angegeben.

Um die Einschränkung der Datenabhängigkeiten zu umgehen, arbeitet der Algorithmus von Binder und Herzog [28] mit geschätzten Bindungskräften. Diese Herangehensweise ermöglicht einerseits einen höheren Parallelitätsgrad, andererseits müssen Approximationsfehler in Kauf genommen werden. Der Algorithmus verwendet einen Rechenkern pro Gelenk und ist somit ebenfalls sehr hardwarenah implementiert. Auch in dieser Arbeit sind keine Anwendungsbeispiele angegeben.

In den folgenden Jahren wurden auch von Lee und Chang [96] und Fijany und Bejczy [54] Algorithmen zur parallelen Berechnung der inversen Dynamik von Robotern entwickelt. Ihre Algorithmen basieren ebenfalls auf speziellen Hardware-Architekturen (SIMD[6] Architekturen) und besitzen somit einen sehr problemspezifischen Charakter. In diesem Bezug sei auch die Arbeit von Gosselin [73] genannt, die ebenfalls die parallele Berechnung der inversen Dynamik von Manipulatoren basierend auf dem rekursiven Newton-Euler Formalismus präsentiert.

Die ersten Methoden zur parallelen Berechnung der Vorwärts-Dynamik von Robotersystemen sind in Amin-Javaheri und Orin [22], Lee und Chang [97] und von Fijany und Bejczy [55] zu finden. Alle drei Methoden basieren auf SIMD Vektorprozessoren. Der Algorithmus von Amin-Javaheri und Orin erzielt beispielsweise eine Beschleunigung von zwei unter der Nutzung von sieben Prozessoren. Eine Nutzung der Algorithmen auf herkömmlichen SISD[7] Desktop-Rechnern scheidet aufgrund der Konzeption zur Implementierung auf SIMD Prozessoren aus.

5 Unter systolischen Pipeline Architekturen werden spezielle SIMD Hardwarearchitekturen verstanden, die im Wesentlichen für Matrix/Vektor Operationen genutzt werden.
6 SIMD: single instruction, multiple data
7 SISD: single instruction, single data

Seit der Arbeit von Amin-Javaheri und Orin entstand eine Vielzahl sogenannter $O(log_2(n))$ Verfahren. Diese Verfahren basieren in der Regel auf rekursiven Newton-Euler Formalismen (Komplexität $O(n)$), deren Komplexität unter bestimmten Rahmenbedingungen mittels Parallelisierung auf $O(log_2(n))$ gesenkt werden kann. Somit skaliert bei diesen Methoden die Rechenzeit lediglich mit dem Logarithmus zur Basis zwei in der Anzahl der Körper. Zu den bekanntesten Werken zu dieser Thematik gehören die Werke von Featherstone [52, 53], Fijany und Featherstone [57], Fijany und Bejczy [56] und Fijany et al. [58].
In [52] stellt Featherstone die Grundzüge seines Algorithmus dar, der in dieser Form nur für kinematische Ketten gültig ist. Eine Erweiterung auf kinematische Schleifen und Bäume stellt er in [53] vor, wobei er zusätzlich auf die Genauigkeit seiner Verfahren eingeht.
Malczyk und Fraczek [102] und Malczyk et al. [103] stellen in ihren Arbeiten eine Erweiterung der $O(log_2(n))$ Verfahren auf Basis von Lagrange-Multiplikatoren vor. Neben diesen Arbeiten seien auch die Arbeiten von Critchley and Anderson [43], Anderson und Duan [23] und Mukherjee und Anderson [109] genannt. In der Arbeit von Mukherjee und Anderson werden auch flexible Körper betrachtet. Einen möglichen Vergleich der $O(log_2(n))$ Verfahren geben Yamane und Nakamura [150].

Alle bisher dargestellten Arbeiten sind dadurch charakterisiert, dass sie teils sehr komplexe Einzellösungen für bestimmte Problemstellungen bieten und dass sie auf rekursiven Newton-Euler Formalismen basieren. Erst die Arbeiten [53] von Featherstone und [57] von Fijany und Featherstone gehen in die Richtung allgemeiner Formulierungen für Mehrkörpersysteme. Rechenzeitvergleiche zwischen den parallelen $O(log_2(n))$ Verfahren und sequentiellen Implementierungen sind nicht zu finden. Die Verfahren eignen sich hervorragend für die Simulation von Baumstrukturen mit einer sehr hohen Anzahl an Körpern, wobei eine ausreichend große Anzahl an Rechenkernen zur Verfügung stehen muss.

Zu den ersten Arbeiten im Bereich der allgemeinen Mehrkörperdynamik gehören die Arbeiten von Hwang et al. [87], Tsai [144] und Eichberger [49]. Hwang et al. beschäftigen sich in ihrer Arbeit mit dem Potential der Parallelisierung von rekursiven $O(n)$ Verfahren und kommen zu dem Schluss, dass das Potential der Verfahren als eher gering einzuschätzen ist. Tsai ist einer der ersten Vertreter, deren Ziel es war eine hardwareunabhängige Simulationsumgebung zur Berechnung von Mehrkörpersystemen zu schaffen. In seiner Arbeit sind Angaben zu den erreichten Beschleunigungen an realen Anwendungsbeispielen zu finden. Eichberger geht auf die Parallelisierung des Mehrkörperalgorithmus an sich ein und stellt eine entsprechende Implementierung mit dynamischer Lastverteilung vor. In seiner Arbeit finden sich mehrere reale Anwendungsbeispiele mit genauen Analysen der möglichen Beschleunigungen.

Koziara stellt in seiner Arbeit [93] die Simulationsumgebung Solfec vor. Solfec ist eine auf verteiltem Speicher (engl. distributed memory) basierende Simulationsumgebung zur parallelen Berechnung von kontaktmechanischen Systemen. Die Simulationsumgebung basiert auf MPI [14] zur Parallelisierung und auf Zoltan [18]

zur Lastverteilung. SOLFEC ist für die Simulation von sehr großen Systemen konzipiert.

Neben der Entwicklung von neuen Methoden und Algorithmen auf der Ebene der Mehrkörperformulierung findet man auch Arbeiten, in denen die lineare Algebra innerhalb des Systems parallelisiert wird [72,79,110]. Wie in Abschnitt 3.5 gezeigt wird, zeigt diese Herangehensweise nur geringes Erfolgspotential. Parallelisierungsmethoden auf Ebene der linearen Algebra können ihr Potential nur in Simulationsbereichen zeigen, in denen die Anzahl der Freiheitsgrade im System sehr hoch ist (z.B. FE- und CFD-Simulationen).

In dieser Arbeit wird auf die Parallelisierung eines Newton-Euler Formalismus der Komplexität $O(n^3)$ eingegangen, der für allgemeine ein- und zweiseitig gebundene Mehrkörpersysteme mit mengenwertigen Nebenbedingungen geeignet ist. Dabei werden keine Approximationen wie in den $O(log_2(n))$ Algorithmen benötigt. Ähnliche Herangehensweisen sind auch in [72] und [126] zu finden. Beiden Arbeiten ist gemeinsam, dass die erzielten Beschleunigungen sehr niedrig ausfallen. Nähere Angaben zu den Ergebnissen dieser Arbeiten sind in Abschnitt 4.1 zu finden.

Methoden zur massiven Parallelisierung von Mehrkörpersystemen sind nur sehr spärlich zu finden, da Mehrkörpersimulationsmodelle in der Regel über eine geringe Anzahl an Körpern verfügen. Mehrkörpersysteme mit mehreren tausend Körpern stellen eher akademische Beispiele dar. Eine Übersicht über die massive Parallelisierung von Mehrkörpersystemen ist in MRÁZ UND VALÁŠEK [107] zu finden. Eine mögliche Formulierung der Systemgleichungen wird in ANDERSON UND OGHBAEI [24] präsentiert.

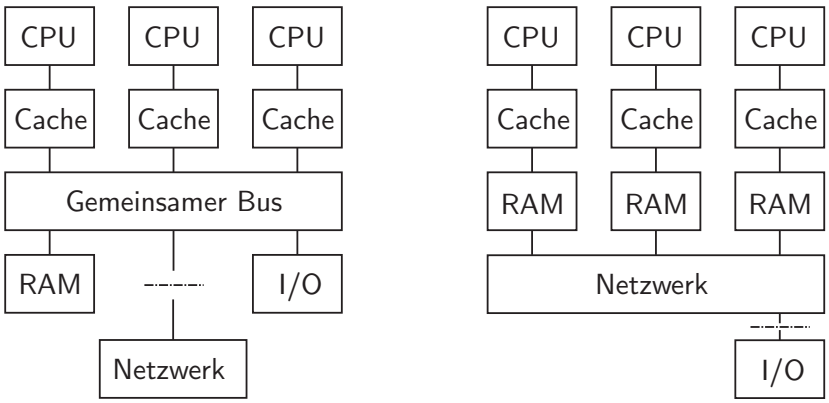
VALASEK UND MRAZ [147] trennen Mehrkörpersysteme in Gelenken auf und verbinden diese wieder mit einwertigen Kraftgesetzen. Dadurch bewirken sie eine Entkopplung der Bewegungsgleichungen, erzeugen jedoch numerisch steife Differentialgleichungen. Die entkoppelten Bewegungsgleichungen werden parallelisiert berechnet. Durch spezielle numerische Methoden versuchen sie die hohen Frequenzen wieder aus dem System zu entfernen.

Eine weitere Methode zur parallelen Berechnung von dynamischen Systemen stellen parallele Co-Simulationen dar. Die parallele Co-Simulation sieht vor, mehrere physikalische Systeme mit je einem eigenen Integrator parallel zu integrieren und diese über eine Co-Simulationsschnittstelle zu koppeln. Für nähere Informationen wird auf die Dissertation von FRIEDRICH [65] und auf die Veröffentlichung von PRESCOTT [119] verwiesen.

Eine Methode zur Parallelisierung von bestehendem Quellcode bieten in der Regel die verwendeten Compiler. Compiler können unabhängige Schleifen erkennen und entsprechend parallelisieren. Dies funktioniert jedoch nur für sehr einfache Softwarestrukturen und ist noch weit davon entfernt, dass unabhängige Schleifen in komplexen Simulationsumgebungen automatisch erkannt werden können. Zu den Compilern, die diese Funktionalität zur Verfügung stellen, gehören die GNU Compiler [4] und die Intel Compiler [8].

3.2 Parallele Plattformen

In diesem Abschnitt wird der Unterschied zwischen Shared-Memory Hardwarearchitekturen und Distributed-Memory Architekturen[8] erläutert. Das Kapitel soll keinen vollständigen Überblick über vorhandene Hardwarearchitekturen geben, sondern soll lediglich die Grundlagen vermitteln, die notwendig sind, um den Unterschied zwischen der Shared-Memory und der Distributed-Memory Parallelisierung zu erklären. Für weitere Informationen zur Thematik der Hardwarearchitekturen kann das Buch von Tanenbaum [140] herangezogen werden.



(a) Shared-Memory. (b) Distributed-Memory.

Abbildung 3.1: Shared-Memory und Distributed-Memory Architekturen.

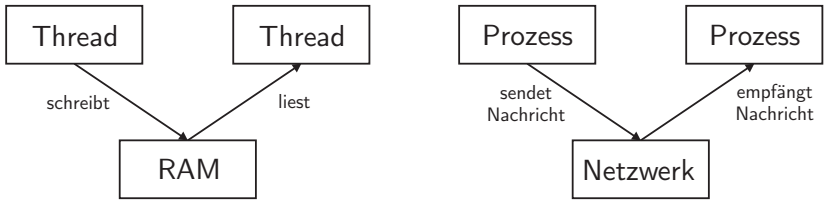
In Abbildung 3.1(a) ist die schematische Darstellung einer Shared-Memory Architektur dargestellt, in Abbildung 3.1(b) die einer Distributed-Memory Architektur. Der wesentliche Unterschied zwischen den Architekturen besteht darin, dass bei Shared-Memory Architekturen alle CPUs über einen gemeinsamen Bus auf den gleichen Arbeitsspeicher (RAM) zugreifen. Bei der Distributed-Memory Architektur besitzt jede CPU einen eigenen Arbeitsspeicher. Die Kommunikation zwischen den CPUs kann zum Beispiel über ein Netzwerk erfolgen. Beiden Architekturen ist gemeinsam, dass jede CPU in der Regel einen eigenen Cache hat.

Normale Arbeitsplatzrechner oder auch Workstations gehören zu der Gruppe der Shared-Memory Architekturen. Moderne Arbeitsplatzrechner besitzen in der Regel ein bis zwei CPUs, die je über mehr als einen Rechenkern verfügen (*Multicore-CPUs*). Diese CPUs verwenden dabei den gleichen Arbeitsspeicher sowie die gleiche Festplatte (I/O). Im Gegensatz dazu stehen Distributed-Memory Rechner, zu denen die meisten HPC[9]-Rechner gehören. Sie bestehen aus einer Vielzahl einzelner Rechner, die über Netzwerke verbunden sind.

8 In dieser Arbeit werden zur Bezeichnung der Hardwarearchitekturen die englischen Begriffe verwendet, da deutsche Übersetzungen äußerst selten verwendet werden.

9 HPC = High Performance Computing

Die beiden Architekturen verwenden entsprechend angepasste Methoden zur Kommunikation zwischen Threads/Prozessen. In Abbildung 3.2(a) ist die Kommunikation von Shared-Memory Architekturen schematisch dargestellt, in Abbildung 3.2(b) die von Distributed-Memory Architekturen.



(a) Shared-Memory. (b) Distributed-Memory.

Abbildung 3.2: Shared-Memory und Distributed-Memory Kommunikation.

Bei Shared-Memory Architekturen schreibt ein Thread[10] seine Daten in den Arbeitsspeicher, die anderen Threads können diese Daten danach im Arbeitsspeicher lesen. Im Gegensatz dazu steht bei Distributed-Memory Architekturen kein gemeinsamer Arbeitsspeicher zur Verfügung, sodass Prozesse[11] untereinander explizit Nachrichten über einen entsprechenden Kommunikationskanal (z.B. Netzwerke) schicken.

Für die Auswahl der Hardware-Architektur ist dabei zu beachten, dass die Kommunikationszeiten auf Shared-Memory Architekturen wesentlich niedriger sind als auf Distributed-Memory Architekturen. Daher sind Distributed-Memory Architekturen vornehmlich für Systeme geeignet, in denen die einzelnen parallelen Abschnitte sehr hohe Rechenzeiten haben (z.B. Co-Simulationen [65]).

Im Bereich des High-Performance Computing ist auch die Kombination beider Architekturen üblich. Abbildung 3.3 zeigt die Kombination schematisch. Dabei sind einzelne Shared-Memory Architekturen über ein passendes Medium (z.B. Netzwerke) untereinander verbunden. Dadurch kann die Parallelisierung auf zwei Ebenen erreicht werden. Einerseits zwischen den Rechnern und andererseits intern auf den einzelnen Rechnern.

3.3 Begriffe und Herausforderungen

Zum Verständnis der vorliegenden Arbeit sind grundlegende Begriffe und Definitionen aus dem Bereich der Parallelprogrammierung notwendig, die in Abschnitt 3.3.1 kurz dargestellt werden. Der zweite Abschnitt 3.3.2 beschäftigt sich mit Problemstellungen und Herausforderungen bei der Konzeptionierung von parallelem Programmcode.

10 Ein Thread bezeichnet in der Informatik einen Ausführungsstrang in der Abarbeitung eines Programms (Prozess). Ein Prozess kann dabei aus mehreren Threads bestehen.

11 Unter einem Prozess wird in der Informatik ein ablaufendes Computerprogramm verstanden.

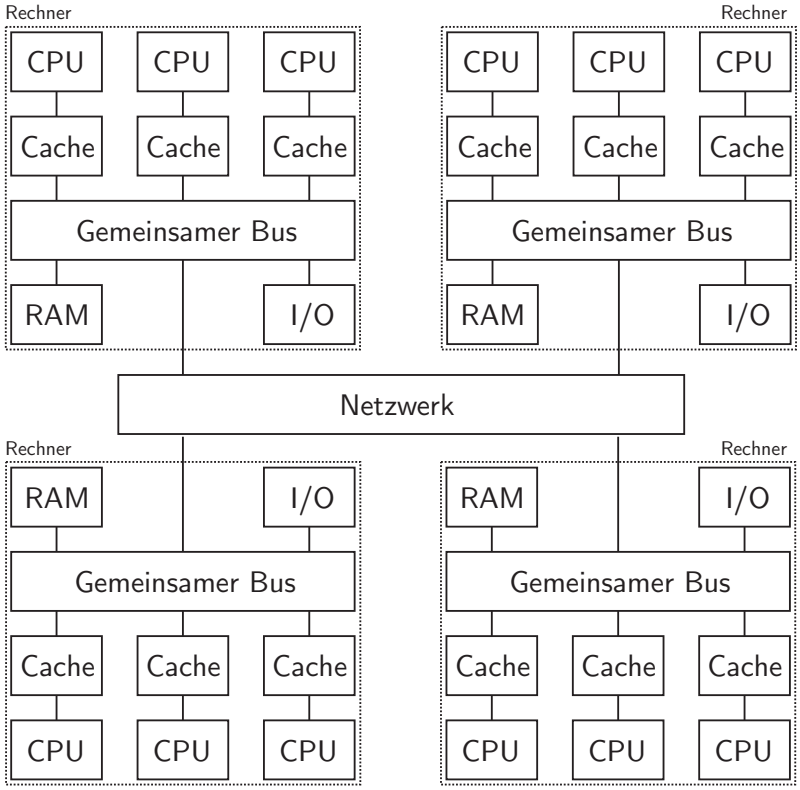


Abbildung 3.3: Kombination aus Shared- und Distributed-Memory Architekturen.

3.3.1 Begriffe und Definitionen

Die in diesem Abschnitt eingeführten Begriffe und Definitionen aus dem Bereich der Parallelprogrammierung gehen im Wesentlichen auf die Werke [49, 80, 143] zurück.

Zunächst müssen die Begriffe *Beschleunigung* (engl. *speedup*) und *Effizienz* (engl. *efficiency*) eingeführt werden.
Unter der Beschleunigung eines parallelen Programms versteht man den Quotienten

$$s_n := \frac{t_1}{t_n} \tag{3.1}$$

aus der Rechenzeit auf einem Kern t_1 und der Rechenzeit t_n auf n Kernen. Somit würde eine Beschleunigung von $s_n = 2$ anschaulich bedeuten, dass die parallelisierte Rechnung nur die Hälfte der Rechenzeit benötigt als die sequentielle Rechnung.

Unter der Effizienz eines parallelen Programms versteht man den Quotienten

$$e_n := \frac{s_n}{n} \tag{3.2}$$

aus der Beschleunigung s_n des Programms und der verwendeten Anzahl an Kernen n. Die Effizienz kann Werte zwischen $e_n = 0$ und $e_n = 1$ annehmen, wobei $e_n = 1$

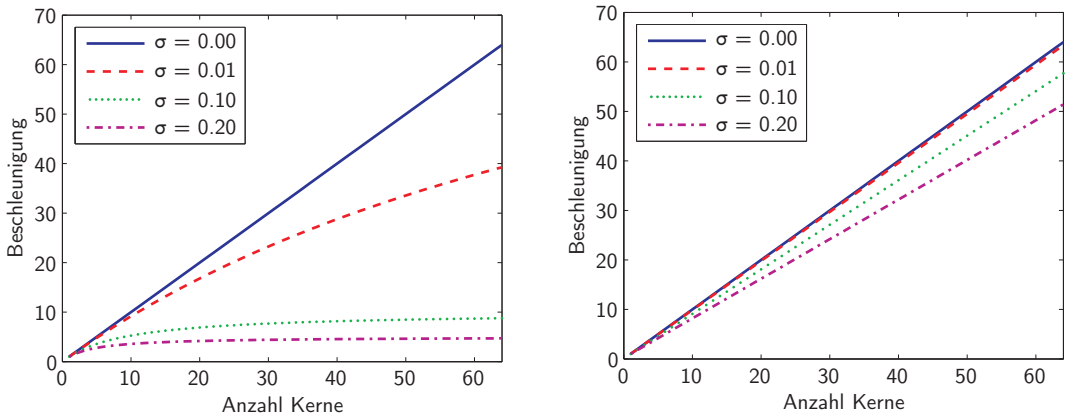
eine optimale Parallelisierung bedeuten würde, die aber in der Realität nicht erreicht werden kann.

In der Literatur sind zwei Gesetze zur Bestimmung der möglichen Beschleunigung eines parallelen Programms in Abhängigkeit der verwendeten Anzahl an Kernen n zu finden.
Das Amdahl'sche Gesetz kann als eine sehr pessimistische Abschätzung der möglichen Beschleunigung gesehen werden und das Gesetz von Gustafson als eine sehr optimistische.

Jedes parallele Programm besteht aus einem Anteil, der sequentiell berechnet werden muss und einem Anteil, der parallel berechnet werden kann. Normiert man die Gesamtrechenzeit des Programms auf eins, so kann die Rechenzeit auf einem Kern angegeben werden als

$$t_1 = 1 = \sigma + (1 - \sigma)\,,$$

mit σ dem sequentiell ausgeführten sequentiellen Codeanteil und $(1 - \sigma)$ dem sequentiell ausgeführten parallelen Codeanteil.



(a) Gesetz von Amdahl.

(b) Gesetz von Gustafson.

Abbildung 3.4: Beschleunigung in Abhängigkeit der Anzahl der Kerne.

Amdahl'sches Gesetz

Amdahl betrachtet für seine Überlegung ein Programm, das mit einer zunehmenden Anzahl an Rechenkernen abgearbeitet wird. Dabei geht er davon aus, dass der sequentielle Codeanteil σ auch bei zunehmender Anzahl an Rechenkernen konstant bleibt, was er durch den notwendigen Overhead für die Parallelisierung begründet. Im Gegensatz dazu sinkt in seinem Gesetz die Ausführungszeit für den parallelen Anteil des Programms mit zunehmender Anzahl an Rechenkernen. Die Rechenzeit

des Programms auf n Rechenkernen kann somit angegeben werden als

$$t_n = \sigma + \frac{1-\sigma}{n}\,. \tag{3.3}$$

Somit ergibt sich für seine Betrachtungen eine maximale Beschleunigung von

$$s_{\mathrm{max,n}} = \frac{1}{\sigma + \frac{1-\sigma}{n}}\,. \tag{3.4}$$

Gesetz von Gustafson

Gustafson legt seinem Gesetz eine andere Betrachtung zugrunde. Er geht nicht davon aus, dass das gleiche Programm mit einer höheren Anzahl an Rechenkernen abgearbeitet wird, sondern dass die Anzahl der Rechenkerne nur dann erhöht wird, wenn ein größeres Problem zu lösen ist.

Mit t_s der Rechenzeit des sequentiellen Codeanteils und t_p der Rechenzeit des parallelen Codeanteils ergibt sich für die sequentielle Ausführung eines parallelen Programms

$$t_1 = t_s + n \cdot t_p\,. \tag{3.5}$$

Nach seinen Überlegungen ergibt sich folglich für die Beschleunigung

$$s_{\mathrm{max,n}} = \frac{t_1}{t_n} = \frac{t_s + n \cdot t_p}{t_s + t_p} = 1 + (n-1)\frac{t_p}{t_s + t_p}\,. \tag{3.6}$$

Abbildung 3.4 zeigt für beide Gesetze die maximale Beschleunigung in Abhängigkeit der Anzahl an Rechenkernen n. Bereits bei einem sequentiellen Anteil von lediglich 10 % fällt die mögliche Beschleunigung laut dem Gesetz von Amdahl sehr gering aus.
Ein direkter Vergleich der beiden Gesetze ist in Abbildung 3.5 zu sehen, in der die mögliche Beschleunigung für den Fall von $n = 48$ Rechenkernen in Abhängigkeit des Anteils an sequentiellem Code σ angegeben ist.

Welches der beiden Gesetze in Realität zutrifft, ist aufgrund der Vielzahl an Einflussfaktoren nicht zu bestimmen. Je nach CPU, Arbeitsspeicher, Cache-Größen und vielen weiteren Faktoren wird sich die Beschleunigung in Realität zwischen den beiden Gesetzen bewegen.

Eine weitere Möglichkeit der Definition der Rechenzeit findet sich in [49]. Eichberger schlägt vor, die gesamte Simulationsdauer t_n auf n Rechenkernen in die reine Rechenzeit t_r und den Parallelisierungs-Overhead t_o einzuteilen.

$$t_n(n) = t_r(n) + t_o(n)$$

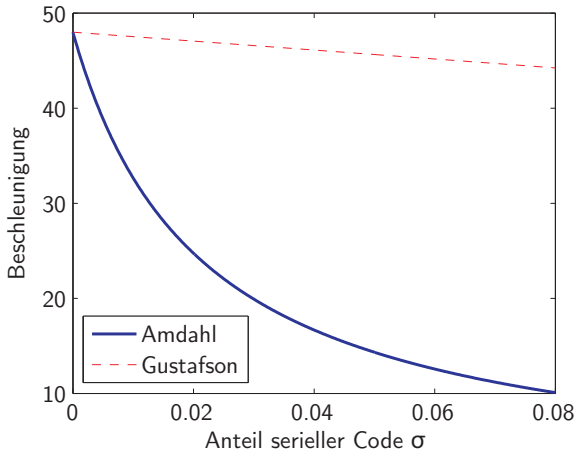


Abbildung 3.5: Vergleich der Gesetze von AMDAHL und GUSTAFSON - 48 Rechenkerne.

3.3.2 Herausforderungen

Die Programmierung von parallelen Quellcodes birgt neben den großen Potentialen auch einige Herausforderungen, auf die in diesem Abschnitt eingegangen wird.

Die erste Herausforderung gilt weniger den Hardware-Architekturen oder den Programmiersprachen, sondern vielmehr dem Programmierer selbst. Das Denken eines Menschen ist sequentieller Natur, sodass die Entwicklung parallel rechnender Software bereits bei der Konzipierung einen hohen Anspruch an die Denkweise des Menschen stellt. Oftmals gerät außer Acht, welche Daten zuerst vorliegen müssen bzw. welche Abhängigkeiten unter den Rechnungen bestehen.

Im Wesentlichen können drei weitere Herausforderungen bei der Entwicklung von parallelen Programmen genannt werden: *Data Races*, *Dead Locks* und *Overhead*.

Data Races / Race Conditions

Data Races oder auch *Race Conditions* treten auf, wenn das Ergebnis eines Programmabschnitts von der zufälligen Zugriffsreihenfolge der beteiligten Threads auf einen Speicherbereich abhängt. Ist der Schreib- und Lesevorgang nicht synchronisiert, sodass die Threads in definierter Reihenfolge einen Speicherbereich nutzen, so ist nicht sichergestellt, dass am Ende des parallelen Abschnitts auch wirklich das gewünschte Ergebnis in dem Speicherbereich steht. Problematisch an dieser Stelle ist auch, dass das Programm bei *Race Conditions* nicht abstürzen muss, sodass der Programmierer oftmals nicht merkt, dass das Ergebnis falsch ist.

In diesem Zusammenhang kann auch der Begriff der *Datenlokalität* eingeführt werden. Objektorientierte Programmiersprachen wie C++ bieten im Gegensatz zu Programmiersprachen wie Fortran den großen Vorteil, dass alle Objekte über ihren eigenen geschützten Speicherbereich verfügen (hohe *Datenlokalität*). Dies verhindert bereits einen Großteil der möglichen *Race Conditions*.

Dead Locks

Dead Locks treten auf, wenn mehrere Threads gegenseitig aufeinander warten. Durch Synchronisationspunkte, Barrieren oder ähnliche Konstrukte kann man Threads explizit in Wartestellung versetzen, bis ein gewünschtes Ereignis eintritt. Wenn jedoch alle Threads gegenseitig aufeinander warten, wird die Ausführung des Programms ausgesetzt.

Zum Debuggen[12] von sequentiellem Programmcode und zum Auffinden von *Memory Leaks*[13] stehen dem Programmierer viele verschiedene Softwarehilfsmittel zur Verfügung. Im Gegensatz dazu sind Hilfsmittel zur Detektion von *Race Conditions* und *Dead Locks* noch in der Entwicklung. Derzeit sind zwei Tools vertreten, *Helgrind* [7] und der *Intel Inspector* [9].

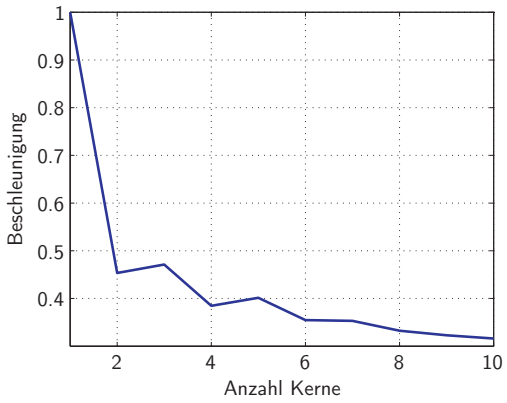
Administrativer Overhead

Die Beherrschung des *Overheads* ist die komplexeste Herausforderung, weshalb sich Kapitel 4 intensiv mit der Minimierung des Overheads der Parallelisierung beschäftigt.
Wie auch im Abschnitt 3.5 zu sehen ist, ist für jede parallele Ausführung eines Codeabschnitts ein nicht zu verachtender Anteil an administrativem Overhead für die Parallelisierung notwendig. Die folgenden Abläufe tragen zum Overhead bei:

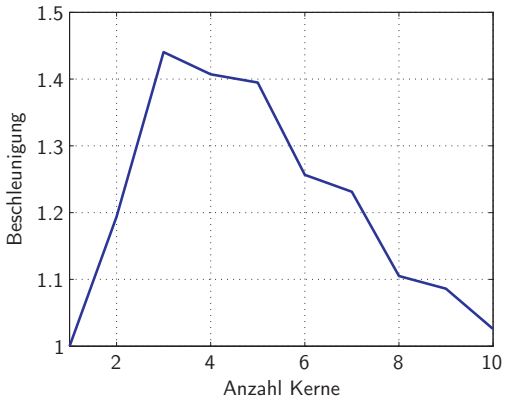
- Erstellung der Threads
- Lastverteilung
- Synchronisierungspunkte
- Kommunikation zwischen Threads
- Overhead durch Einbindung zusätzlicher Bibliotheken
- Beenden der Threads
- Zusammenführung der Daten

Der Overhead nimmt an Bedeutung zu, je kleiner die eigentliche Rechenzeit pro Thread ist. Rechnen die einzelnen Threads mehrere Sekunden an ihrem Rechenjob, so können die wenigen Mikrosekunden[14] für den Overhead vernachlässigt werden. Benötigen die Rechenoperationen jedoch selber nur wenige Mikrosekunden, so fällt der Overhead sehr stark ins Gewicht.
Dies kann dazu führen, dass ein parallel rechnendes Programm höhere Rechenzeiten benötigt, als das selbe Programm sequentiell auf einem Rechenkern. Dieser Fall

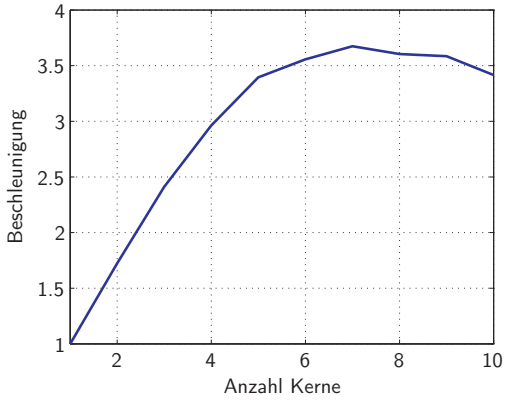
12 Ein Debugger (engl. bug = Programmfehler) ist ein Werkzeug zum Diagnostizieren und Auffinden von Fehlern in Computersystemen.
13 Unter *Memory Leak* versteht man den Fall, dass allokierter Speicher nach der Verwendung nicht mehr korrekt deallokiert wird.
14 Genaue Angaben über den Overhead lassen aufgrund der hohen Anzahl an Einflussfaktoren nicht angeben.



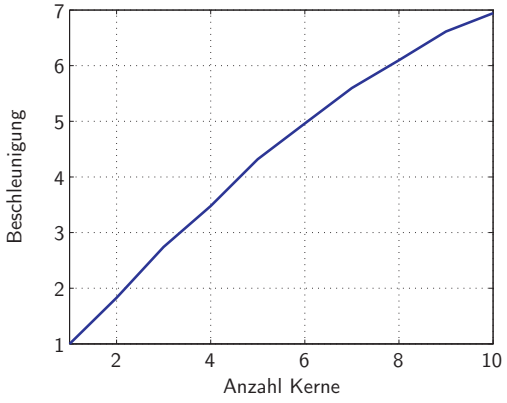
(a) Rechendauer $t = 0.8236 \cdot 10^{-5}$s.



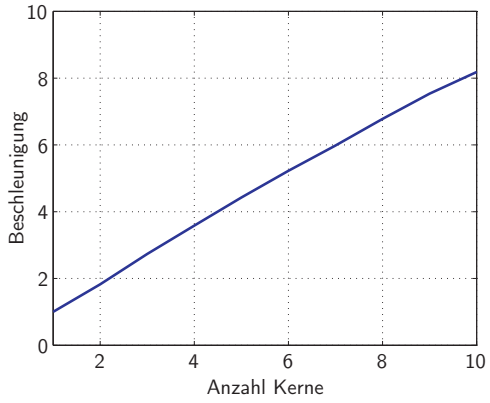
(b) Rechendauer $t = 2.6823 \cdot 10^{-5}$s.



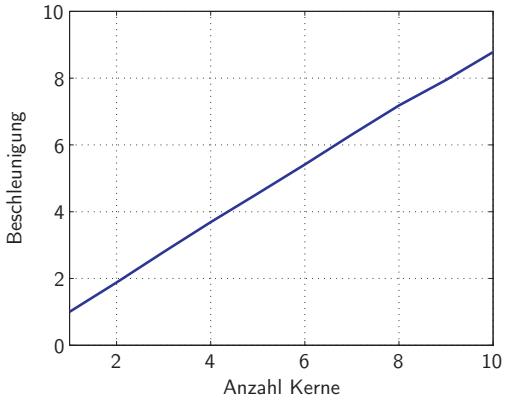
(c) Rechendauer $t = 7.8876 \cdot 10^{-5}$s.



(d) Rechendauer $t = 19.5711 \cdot 10^{-5}$s.



(e) Rechendauer $t = 38.0413 \cdot 10^{-5}$s.



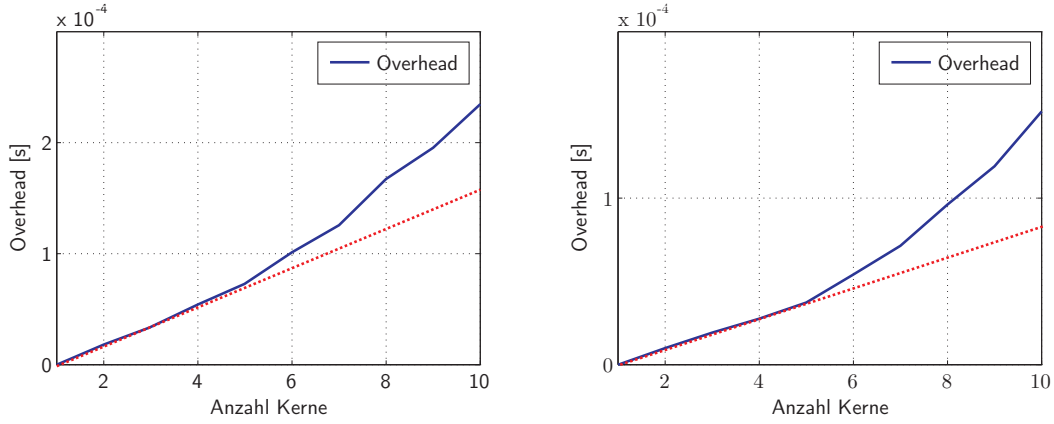
(f) Rechendauer $t = 71.3 \cdot 10^{-5}$s.

Abbildung 3.6: Beschleunigung in Abhängigkeit der Anzahl der Kerne für unterschiedliche Rechenzeiten t der einzelnen Rechenoperationen.

tritt in der Mehrkörperdynamik häufig ein. Im Gegensatz zu anderen physikalischen Domänen wie den FE-Methoden, dauern viele Rechenoperationen in der Mehrkörperdynamik (vgl. updateh, updateM, etc. in Abschnitt 2.5) nur sehr kurz, sodass der Overhead so gering wie möglich gehalten werden muss, damit die Parallelisierung

einen guten Speedup und vor allem auch eine hohe Effizienz erreichen kann.

Da der Overhead von sehr vielen Einflussfaktoren abhängt (CPU, Cache-Größen, Arbeitsspeicher, etc.) und somit eine theoretische Untersuchung den Umfang dieser Arbeit überschreiten würde, wird daher auf die Arbeiten von Graham [74], Gunther [75] und Artis [25] verwiesen. In der vorliegenden Arbeit wird der Overhead an einem einfachen Beispiel experimentell untersucht. Angenommen eine gleiche mathematische Rechnung soll n-mal berechnet werden, so kann man dies auf n Rechenkernen parallel ausführt werden, wobei für die Parallelisierung ein administrativer Overhead anfällt. Abbildung 3.6 zeigt für verschieden lange einzelne Rechenzeiten t der mathematischen Rechnung die erreichten Beschleunigungen in Abhängigkeit der Anzahl der verwendeten Rechenkerne.
Abbildung 3.6(a) zeigt einen Extremfall, der im Bereich der parallelen Mehrkörperdynamik häufig anzutreffen ist. Die mathematische Berechnung selbst benötigt eine sehr geringe Rechenzeit von $t = 0.8236 \cdot 10^{-5}$ s. Führt man die Rechnung zweimal verteilt auf zwei Rechenkernen aus, so fällt die Beschleunigung auf etwa 0.45 ab. Das bedeutet anschaulich gesprochen, dass die parallele Berechnung von zwei derartigen mathematischen Rechnungen etwa doppelt solange dauert als die sequentielle Abarbeitung auf einem Kern. Hieran ist wieder zu erkennen, dass der Overhead eine entscheidende Rolle bei der Parallelisierung spielt.
Abbildung 3.6(b) und 3.6(c) zeigen, dass es auch bei kurzen Rechenzeiten der individuellen Operationen möglich ist Beschleunigungen über $s = 1$ zu erzielen, wobei dazu die Anzahl der verwendeten Kerne beschränkt werden muss. Auch hier gilt, je höher die Anzahl der Kerne, umso höher auch der administrative Overhead.
Zuletzt zeigt Abbildung 3.6(f), dass bei ausreichend langen Rechenzeiten der einzelnen Threads sehr gute Beschleunigungen erreicht werden können. In diesem Beispiel kann mit $n = 10$ Rechenkernen eine Beschleunigung von $s_{10} = 8.8$ und eine Effizienz von $e_{10} = 0.88$ erreicht werden.



(a) Rechendauer $t = 2.6823 \cdot 10^{-5}$s.

(b) Rechendauer $t = 7.8876 \cdot 10^{-5}$s.

Abbildung 3.7: Administrativer Overhead in Abhängigkeit der Anzahl an Kernen für unterschiedliche Rechenzeiten t.

Die Abbildung 3.6(b) und 3.6(c) lassen den Rückschluss zu, dass der Overhead in diesem Beispiel offensichtlich mehr als nur linear mit der Anzahl der verwendeten Kerne ansteigt. Diese Vermutung kann mit Abbildung 3.7 bestätigt werden, in welcher der errechnete Overhead für die Rechenzeiten $t = 2.6823 \cdot 10^{-5}\,\mathrm{s}$ und $t = 7.8876 \cdot 10^{-5}\,\mathrm{s}$ dargestellt ist. Ein linearer Trend ist als rote Linie eingezeichnet. Dieser Trend bestätigt sich in der systeminternen Parallelisierung (4).
Analoge Zusammenhänge werden auch in den Arbeiten [25, 74, 75] gezeigt. Gunther [75] zeigt in seiner Veröffentlichung drei mögliche Modelle zur Beschreibung des Overheads von parallelen Programmen. Das Amdahl'sche Gesetz (siehe Abschnitt 3.3.1, das *geometrische* Gesetz und das *quadratische* Gesetz. Mit dem quadratischen Gesetz können die experimentellen Untersuchungen dieses Kapitels bestätigt werden.

3.4 Eingliederungsformen der Parallelisierung

In diesem Abschnitt werden zwei grundlegende Methoden zur Eingliederung von Parallelisierungsmethoden dargestellt. Die erste Eingliederung kann der Dissertation von Eichberger [49] entnommen werden, die zweite Eingliederung ist während dieser Arbeit entstanden.

Eichberger schlägt vor, Parallelisierungsalgorithmen je nach Detaillierungsgrad in die drei Gruppen

- *feinstrukturierte* Parallelisierung
- *mittelstrukturierte* Parallelisierung
- *grobstrukturierte* Parallelisierung

einzugliedern.

Feinstrukturierte Parallelisierung findet zum Beispiel auf Ebene der Matrix/Vektor Operationen statt. Wie auch in Abschnitt 3.5 gezeigt, können nahezu alle Matrix/-Vektor Operationen parallelisiert werden. Die parallel berechneten Operationen haben jedoch eine sehr kurze Rechenzeit, sodass der Overhead stark ins Gewicht fällt. Somit können die Beschleunigungen wie in Abschnitt 3.3.2 beschrieben äußerst gering oder sogar kleiner eins ausfallen. Allgemein kann festgestellt werden, dass diese Art der Parallelisierung für die Mehrkörperdynamik eine sehr untergeordnete Rolle spielt, da die Dimension der Matrix/Vektor Operationen zu gering ist.

Grobstrukturierte Parallelisierung findet beispielsweise auf Integratorebene statt (siehe auch Kapitel 5). Bei parallelen Extrapolationsverfahren wird das gleiche Mehrkörpersystem mit unterschiedlichen Parametern (z.B. unterschiedliche Integrationsschrittweite Δt) gerechnet. Die einzelnen parallelen Rechnungen, also die gesamte Berechnung eines einzelnen Zeitschrittes des Mehrkörpersystems, benötigt im Vergleich zu Matrix/Vektor Operationen signifikant mehr Zeit und eignet sich somit

hervorragend für die Parallelisierung. Parallelisierungsmethoden auf der *grobstrukturierten* Ebene lassen im Allgemeinen hohe Beschleunigungen (Gleichung (3.1)) und hohe Effizienz-Werte (Gleichung (3.2)) zu.

Methoden zwischen diesen Grenzgruppen werden in die *mittelstrukturierte* Parallelisierung eingeordnet. Das in Kapitel 4 vorgestellte Verfahren kann in diese Gruppe eingeordnet werden. Die Rechenzeiten der einzelnen parallelen Rechnungen sind in dieser Gruppe zwar bereits sehr niedrig, jedoch können mit spezialisierten Verfahren dennoch gute Beschleunigungen erreicht werden.

Eine Möglichkeit, um speziell Parallelisierungsmethoden für die Simulation dynamischer Systeme einzugliedern, ist in Abbildung 3.8 gezeigt. Die Unterscheidung findet auf Basis dessen statt, mit welchem Systemen die parallele Rechnung stattfindet.

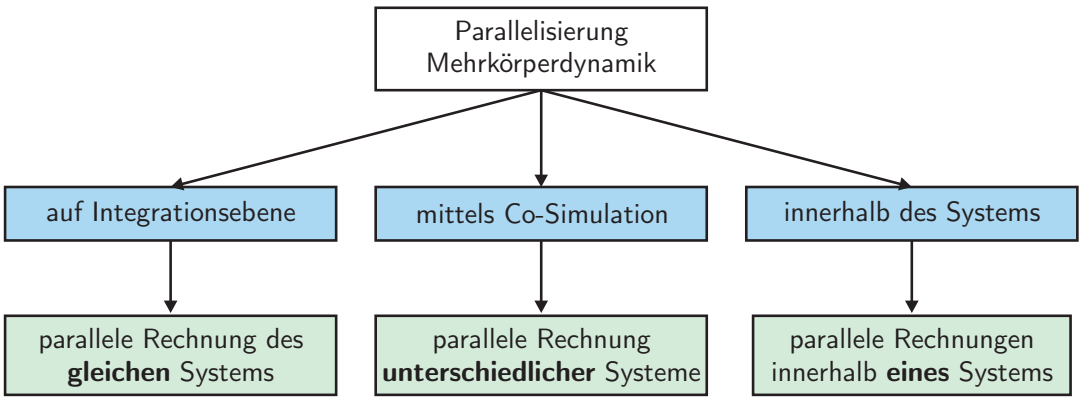


Abbildung 3.8: Eingliederung von Parallelisierungsmethoden.

Die Parallelisierung auf Integrationsebene, wie zum Beispiel parallele Extrapolationsverfahren, sind dadurch charakterisiert, dass sie das **gleiche** System mit unterschiedlichen Parametern berechnen. Zu dieser Gruppe gehören auch die in Abschnitt 5 gezeigten Methoden. Methoden dieser Gruppe können in die Gruppe der *grobstrukturierten* Parallelisierung eingeordnet werden.
Parallele Co-Simulationen erlauben es **verschiedene** Systeme, die nicht zwingend der gleichen physikalischen Domäne angehören müssen, parallel zu rechnen. Auch diese Methode gehört in die Gruppe der *grobstrukturierten* Parallelisierung. Weitere Informationen zu der parallelen Co-Simulation können in der Dissertation von Friedrich [65] gefunden werden.
Die dritte Gruppe beinhaltet parallele Berechnungen innerhalb **eines** Systems. Zu dieser Gruppe gehören die in Kapitel 4 beschriebenen Methoden. Diese Methoden gehören in die Gruppe der *mittelstrukturierten* Parallelisierung.

Die ersten beiden Methoden zeigen zwei wesentliche Vorteile gegenüber der dritten Gruppe. Es werden getrennte Systeme parallel gerechnet, sodass die Gefahr von *Race Conditions* oder *Dead Locks* äußerst gering ist. Zudem sind in der Regel die einzelnen Rechenzeiten der parallelen Abschnitte sehr hoch, sodass der Overhead der Parallelisierung als unwesentlich betrachtet werden kann. Die dritte Gruppe birgt

somit die größten Herausforderungen an den Umgang mit den drei in Abschnitt 3.3.2 genannten Herausforderungen.

3.5 Vergleich von Matrix/Vektor-Bibliotheken

Als Basis jeder numerischen Simulation können mathematische Methoden gesehen werden, so auch in der Mehrkörperdynamik. Geht man im Abstraktionsgrad genügend weit zurück, so kann jede Mehrkörpersimulation auf grundlegende Matrix/Vektor Operationen zurückgeführt werden.

Daher liegt es nahe die beiden wichtigsten Bibliotheken für lineare Algebra hinsichtlich ihrer Eignung für die Nutzung in der Mehrkörperdynamik zu untersuchen. Die zwei wesentlichen Bibliotheken sind LAPACK/BLAS[15] [2, 11] und ATLAS[16] [1].
Die Bibliotheken unterscheiden sich unwesentlich in den enthaltenen mathematischen Methoden, jedoch sehr darin, wie effizient die einzelnen Methoden implementiert sind. LAPACK/BLAS kann als eine grundlegende Referenzimplementierung gesehen werden, bei deren Entwicklung jedoch nicht im Vordergrund stand die maximale Performance zu erzielen. ATLAS wendet hochoptimierte Algorithmen an und zielt dabei insbesondere auf eine maximale Performance ab.
ATLAS kann in einer rein sequentiellen als auch in einer parallelen Version verwendet werden. Bei der Kompilierung von ATLAS wird automatisch bestimmt, ab welcher Systemgröße die parallele Version verwendet wird. Dies wird auch an den Beispielen in diesem Kapitel ersichtlich.

Wie bereits in Abschnitt 3.4 erläutert, können auch Matrix/Vektor Operationen in vielen Fällen parallelisiert werden, wobei diese Art der Parallelisierung zu der Gruppe der *feinstrukturierten* Parallelisierung gehört.
Die Parallelisierung von Matrix/Vektor Operationen kann an einem sehr einfachen Beispiel erläutert werden. Betrachtet man eine einfache Matrix/Vektor Multiplikation der Dimension n

$$\boldsymbol{y} = \boldsymbol{A} \cdot \boldsymbol{x}\,, \tag{3.7}$$

so kann jeder Eintrag von $\boldsymbol{y}$ unabhängig von den übrigen Einträgen berechnet werden. Stehen n Prozessoren zur Verfügung, so kann theoretisch jeder Eintrag von $\boldsymbol{y}$ parallel auf einem Prozessor berechnet werden. Angenommen die Dimension der Operation sei $n = 4$ und die Anzahl der Prozessoren sei ebenfalls vier, so kann die Rechenverteilung wie folgt aussehen:

$$\begin{aligned}
&\text{Prozessor 1:} && y_1 = A_{11}x_1 + A_{12}x_2 + A_{13}x_3 + A_{14}x_4 \\
&\text{Prozessor 2:} && y_2 = A_{21}x_1 + A_{22}x_2 + A_{23}x_3 + A_{24}x_4 \\
&\text{Prozessor 3:} && y_3 = A_{31}x_1 + A_{32}x_2 + A_{33}x_3 + A_{34}x_4 \\
&\text{Prozessor 4:} && y_4 = A_{41}x_1 + A_{42}x_2 + A_{43}x_3 + A_{44}x_4
\end{aligned}$$

15 LAPACK - Linear Algebra PACKage; BLAS - Basic Linear Algebra Subprograms
16 ATLAS - Automatically Tuned Linear Algebra Software

Somit wäre in diesem Beispiel eine theoretische Beschleunigung von vier möglich. Aufgrund des Overheads kann jedoch der Fall eintreten, dass die sequentielle Rechnung weniger Zeit in Anspruch nimmt als die parallele Rechnung.

An dieser Stelle sei explizit darauf hingewiesen, dass dieser Abschnitt keineswegs einen ganzheitlichen und objektiven Vergleich der genannten Matrix/Vektor Bibliotheken darstellen soll, sondern dazu dient, die Probleme und Eigenheiten der *feinstrukturierten* Parallelisierung aufzuzeigen.

Um die Matrix/Vektor Bibliotheken vergleichen zu können, werden zwei Testgleichungen herangezogen.
Die erste Testgleichung beinhaltet lediglich Matrix/Vektor Multiplikationen, Additionen und Zuweisungen. Sie ist an die bekannte Form der Bewegungsgleichungen (2.1) von dynamischen Systemen angelehnt:

$$\boldsymbol{k} = \boldsymbol{h} + \boldsymbol{W}_N\boldsymbol{\lambda}_N + \boldsymbol{W}_T\boldsymbol{\lambda}_T \tag{3.8}$$

Dabei haben die Vektoren $\boldsymbol{h}, \boldsymbol{\lambda}_N, \boldsymbol{\lambda}_T$ und die Matrizen $\boldsymbol{W}_N$,$\boldsymbol{W}_T$ die Dimension n bzw. $n \times n$ und werden mittels eines Zufallsgenerators erzeugt. Das Ergebnis der Gleichung wird dem Vektor $\boldsymbol{k}$ zugewiesen. Diese Gleichung steht stellvertretend für alle Matrix/Vektor Multiplikationen und Additionen, die während der Simulation auftreten.
Die zweite wichtige Klasse von mathematischen Methoden ist die Lösung von linearen Gleichungssystemen. Die Lösung von linearen Gleichungssystemen wird nicht nur zur expliziten Lösung von Gleichungen benötigt, sondern auch zur Invertierung von Matrizen. Dies wird im Wesentlichen für die Berechnung der Massenwirkungsmatrix

$$\boldsymbol{G} = \boldsymbol{W}^T\boldsymbol{M}^{-1}\boldsymbol{W} \tag{3.9}$$

benötigt. Darin kann der Anteil

$$\boldsymbol{X} = \boldsymbol{M}^{-1}\boldsymbol{W}$$

als Lösung des linearen Gleichungssystems

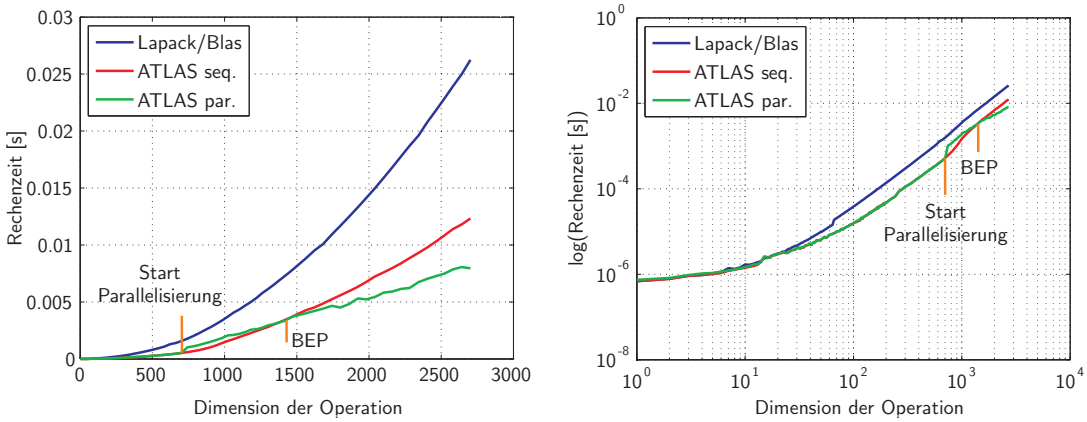
$$\boldsymbol{M}\boldsymbol{X} = \boldsymbol{W}$$

betrachtet werden. Die Lösung des linearen Gleichungssystems erfolgt über ein Cholesky-Verfahren [120].
Der wesentliche Unterschied bei den beiden Testgleichungen (3.8) und (3.9) liegt in der Größe der Dimension n, die in realen Simulationsumgebungen auftritt. Die in der Mehrkörpersimulation[17] auftretenden Dimensionen der Matrix/Vektor Multiplikationen und Additionen auf OBJECT bzw. LINK-Ebene liegen in der Regel bei $n = 3$ bis 6 bzw. auch geringfügig darüber. Die Dimension der Massenwirkungsmatrix (siehe Abschnitt 5.2.4) entspricht der Anzahl der mengenwertigen Kraftparameter $\boldsymbol{\lambda}$

17 An dieser Stelle wird stellvertretend die Implementierung in MBSim als Simulationsumgebung angenommen.

im Mehrkörpersystem, d.h. die Lösung von linearen Gleichungssystem kann auch Dimensionen von $n = 1000$ (zum Beispiel im CVT-Simulationsmodell - siehe Abschnitt 6.2.2) oder höher annehmen.

Alle Beispiele wurden jeweils auf der in Abschnitt 1.2 dargestellten Hardware-Architektur gerechnet. Die FMATVEC wird als Interface zu den Bibliotheken verwendet.



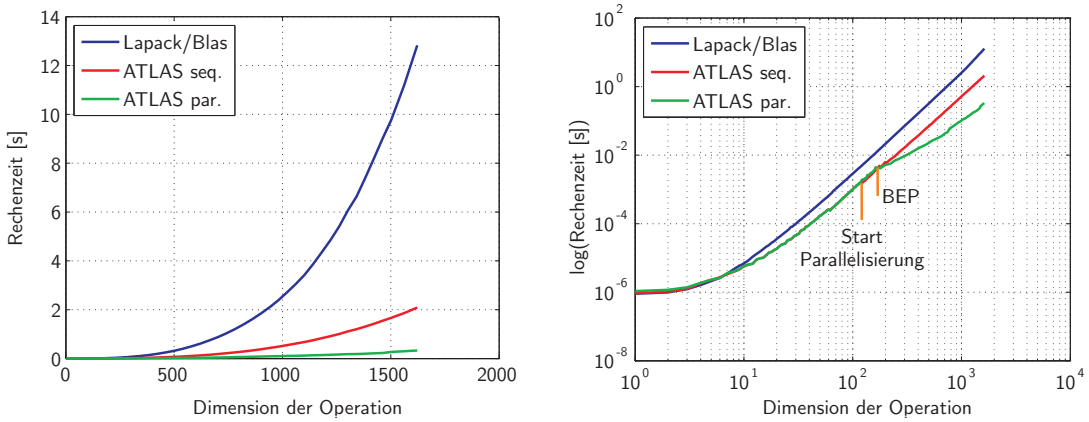
(a) Lineare Skalierung. (b) Logarithmische Skalierung.

Abbildung 3.9: Rechenzeit in Abhängigkeit der Dimension - Gleichung (3.8).

Abbildung 3.9 zeigt die Rechenzeit der Testgleichung (3.8) in Abhängigkeit der Dimension n der Matrizen und Vektoren. Auf der linken Seite (3.9(a)) in linearer Skalierung und auf der rechten Seite (3.9(b)) in doppelt logarithmischer Darstellung. Es ist zu erkennen, dass sich LAPACK/BLAS mit zunehmender Dimension der Operation schlechter verhält als die ATLAS Bibliothek. Vergleicht man die sequentielle und parallele Version von ATLAS, so sieht man, dass ATLAS bei einer Dimension von etwa $n = 700$ in den parallelen Modus schaltet. Vor dieser Schwelle liegen die Rechenzeiten der sequentiellen und parallelen Version nahezu übereinander. Wie bereits erwähnt, muss jedoch eine parallele Berechnung nicht zwingend auch eine Beschleunigung mit sich ziehen. ATLAS schaltet für die Testgleichung etwas verfrüht in den parallelen Modus um, wodurch die Rechenzeit bis zum Break-Even-Point (BEP) leicht höher und erst ab diesem Punkt niedriger ist als bei der sequentiellen Version. Vernachlässigt man Dimensionen[18] unter $n = 20$, so zeigt Abbildung 3.9(b), dass die Rechenzeit quadratisch in der Dimension der Rechnung ansteigt (ohne Parallelisierung).

Abbildung 3.10 zeigt die Rechenzeit der Testgleichung (3.9) ebenfalls in Abhängigkeit der Dimension n der Matrizen. Die möglichen Rückschlüsse sind analog zu Abbildung 3.9 zu sehen. Die Implementierung von LAPACK/BLAS ist mit zunehmender Größe am langsamsten, wobei die parallele Version von ATLAS bei etwa $n = 118$ aktiviert wird und bei etwa $n = 170$ ihren Break-Even-Point findet.

18 Unterhalb einer Dimension von etwa $n = 20$ ist die pure Rechenzeit der Gleichungen nicht mehr dominant, sodass keine eindeutige Skalierung abgelesen werden kann.



(a) Lineare Skalierung. (b) Logarithmische Skalierung.

Abbildung 3.10: Rechenzeit in Abhängigkeit der Dimension - Gleichung (3.9).

Bereits an dieser Stelle liegt der Rückschluss nahe, dass sich bei üblichen Dimension von Mehrkörpersimulationen parallele Matrix/Vektor Multiplikationen und Additionen (3.8) nicht lohnen. Bei der Lösung von linearen Gleichungssystemen bzw. bei der Invertierung von Matrizen liegt der Break-Even-Point in diesem Beispiel schon bei etwa $n = 200$. Systeme mit mehr als 200 mengenwertigen Kraftparametern sind durchaus verbreitet, sodass eine parallele Lösung der linearen Gleichungssysteme eine Zeitersparnis bedeuten kann.

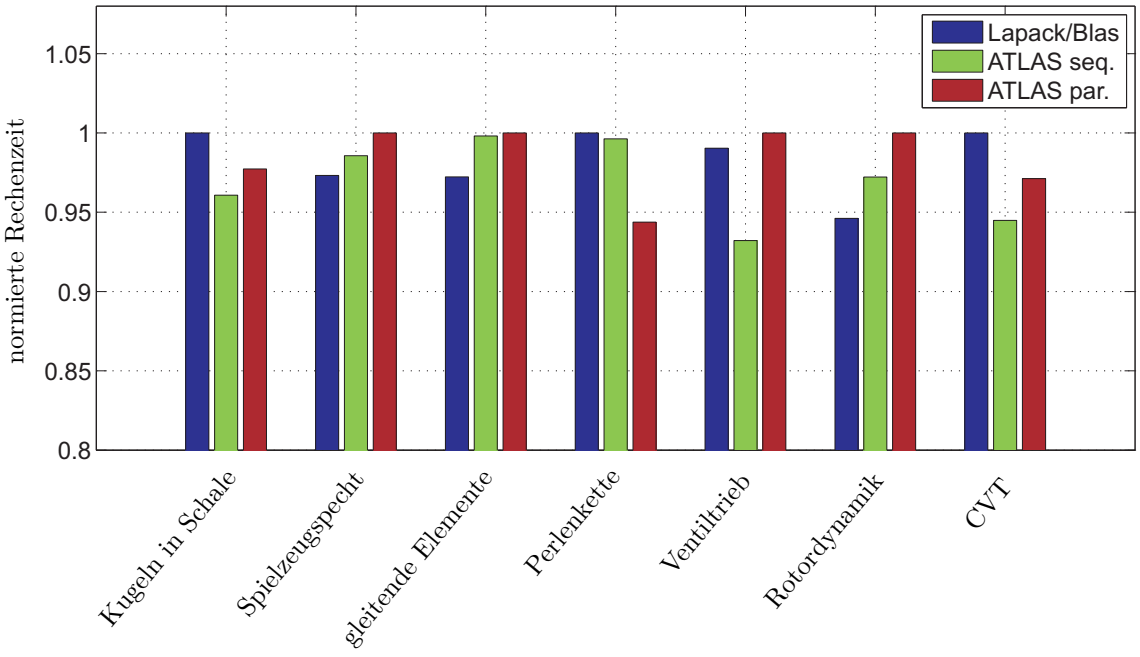
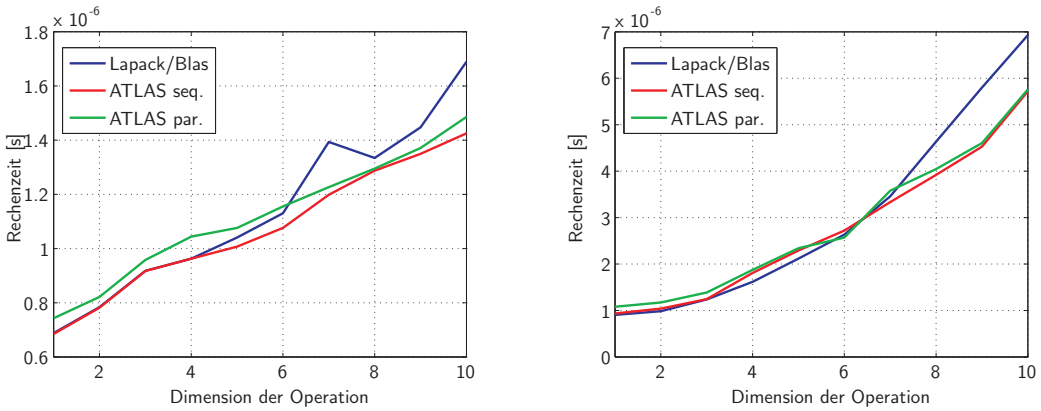


Abbildung 3.11: Vergleich von Matrix/Vektor Bibliotheken anhand von Beispielen.

Um diese eher theoretischen Aussagen auch anhand von realen Simulationen un-

tersuchen zu können, wurden die Beispiele aus Kapitel 6 mit allen drei Varianten gerechnet. Das Ergebnis der Rechnungen kann in Abbildung 3.11 gesehen werden, in der die normierten Rechenzeiten der Beispiele für die einzelnen Bibliotheken dargestellt sind. Auffällig ist, dass lediglich in einem einzigen Beispiel (*Perlenkette*) die parallele Version von ATLAS schneller ist als die sequentielle Version. In allen anderen Beispielen führt der Overhead durch die Parallelisierung bzw. durch die allgemein etwas niedrigere Performance der parallelen Version von ATLAS bei kleinen Dimensionen (siehe Abbildung 3.12) zu höheren Rechenzeiten. Eine der Ursachen liegt darin, dass in der parallelen Version von ATLAS vor der eigentlichen Berechnung der mathematischen Operation Abfragen stattfinden, ob sich die Parallelisierung lohnt und wenn ja, wie genau die Parallelisierung ablaufen muss.
Lapack/Blas verhält sich ähnlich wie die sequentielle Version von ATLAS. Teils ist Lapack/Blas schneller, teils die sequentielle Version von ATLAS. Dieser Effekt kann mit Abbildung 3.12 erklärt werden, die die gleichen Informationen enthält wie die Abbildungen 3.9 und 3.10, jedoch mit einem reduzierten Dimensionsbereich bis $n = 10$. Hier ist zu erkennen, dass die Performance von Lapack/Blas für kleine Dimensionen sehr hoch ist. Ein Großteil der Performance von ATLAS wird durch effiziente Speichernutzung erzielt, die jedoch bei geringen Dimensionen nicht zum Tragen kommt.



(a) Gleichung (3.8). (b) Gleichung (3.9).

Abbildung 3.12: Rechenzeit in Abhängigkeit der Dimension - verkleinerter Dimensionsbereich.

Ähnliche Vergleiche sind von Clauberg und Ulbrich [42] bezüglich der IntelMKL durchgeführt worden. Die Interpretation der Ergebnisse lässt die gleichen Rückschlüsse zu wie auch die Interpretation der Ergebnisse der ATLAS Bibliothek in diesem Kapitel. Die wesentliche Aussage dieses Kapitels ist, dass die *feinstrukturierte* Parallelisierung auf Ebene der Matrix/Vektor Operationen in MBSim keine nennenswerte Beschleunigungen erzielt. Auf die gleichen Aussagen bezüglich der *feinstrukturierten* Parallelisierung kommt auch Eichberger in seiner Dissertation [49].

3.6 OpenMP

OpenMP ist eine offene Programmierschnittstelle für die Parallelisierung von Software, die in C, C++ oder Fortran geschrieben ist. Für weiterführende Informationen sei auf die Bücher von Chapman et al. [36] und Hoffmann und Lienhart [80] verwiesen. OpenMP muss dabei von dem verwendeten Compiler unterstützt werden.

OpenMP ist im Gegensatz zu MPI[19], das für die Distributed-Memory Parallelisierung entwickelt wird, für die Shared-Memory Parallelisierung konzipiert. Daher findet die Parallelisierung auf Thread- bzw. Schleifenebene statt und nicht auf Basis von Prozessen, die Nachrichten über definierte Protokolle austauschen können, wie es zum Beispiel bei der MPI-Bibliothek vorgesehen ist.
Auch die Kombination von OpenMP und MPI ist möglich und wird im Bereich des HPC[20] angewandt. Dabei werden Prozesse parallel auf mehreren Rechnern gestartet und kommunizieren mittels MPI (Distributed-Memory Parallelisierung), die jeweils intern wieder parallel auf den einzelnen Rechnern mittels OpenMP arbeiten (Shared-Memory Parallelisierung).

Ein wesentlicher Vorteil von OpenMP ist darin zu sehen, dass sich bereits bestehender Quellcode mit den Direktiven, die OpenMP bereitstellt, parallelisieren lässt. Dieser Quellcode ist in der Regel auch weiterhin lauffähig, wenn der Compiler bzw. die Rechnerarchitektur keine Shared-Memory Parallelisierung unterstützt.

Dieses Kapitel soll keineswegs umfassende Erklärungen zu OpenMP bieten, sondern soll dem Leser einen einfachen Einblick in die Möglichkeiten der Shared-Memory Parallelprogrammierung mit OpenMP bieten.

Betrachtet wird eine einfache C++ Klasse `Addition` (siehe Quellcode 3.1). Der Klasse werden in ihrem Konstruktor (Zeile 4) zwei Zahlen a und b zugewiesen, die mit der Methode `addiere` (Zeile 5) addiert und in die Variable c gespeichert werden können.

```
1  class Addition {
2
3    public:
4      Addition(double a_, double b_) {a = a_; b = b_;}
5      void addiere() { c=a+b;}
6      void gebe_aus() {cout << "a + b = " << c << endl;}
7
8    private:
9      double a, b, c;
10 };
```

Quellcode 3.1: Einfache Additionsklasse in C++.

19 MPI: Message Passing Interface
20 HPC: High Performance Computing

Um die Möglichkeiten von OpenMP zur Parallelprogrammierung zu zeigen, sollen vier Additionen parallel ausgeführt werden.
In Quellcode 3.2 wird ein Array aus `Anzahl_Additionen = 4` Zeigern auf Additionen angelegt. Den Additionen werden in den Zeilen 5 bis 8 je zwei Zahlen zugewiesen. Aufgabe `*Arbeit[0]` soll die Zahlen `1` und `2`, `*Arbeit[1]` die Zahlen `6` und `7`, `*Arbeit[2]` die Zahlen `9` und `10` und `*Arbeit[3]` die Zahlen `12` und `13` addieren.

```
1   int Anzahl_Additionen = 4;
2
3   Addition *Arbeit[Anzahl_Additionen];
4
5   Arbeit[0] = new Addition(1,2);
6   Arbeit[1] = new Addition(6,7);
7   Arbeit[2] = new Addition(9,10);
8   Arbeit[3] = new Addition(12,13);
```

Quellcode 3.2: Array an Additionen anlegen.

Grundsätzlich bietet OpenMP drei Möglichkeiten parallele Programmabschnitte umzusetzen. Die Möglichkeit des `parallel for` Konstrukts ist in Quellcode 3.3, die des `parallel sections` Konstrukts in Quellcode 3.4 und die des `task` Konstrukts in Quellcode 3.5 gezeigt.

Das `parallel for` Konstrukt (siehe Quellcode 3.3) stellt die einfachste Möglichkeit dar, vorhandene Schleifen parallel rechnen zu lassen. Dabei ist darauf zu achten, dass die Schleifendurchläufe unabhängig voneinander sind. Einerseits ist dieses Konstrukt sehr einfach in der Anwendung, andererseits bietet es jedoch sehr wenig Freiräume, da der zu parallelisierende Abschnitt eine Schleife sein muss, deren Größe beim Start der Schleife feststeht.
Mit der Compiler Direktive `#pragma omp parallel for` (Zeile 1) wird dem Compiler mitgeteilt, dass die folgende Schleife parallel abgearbeitet werden soll. Durch die zusätzliche Angabe von `num_threads(4)` kann bei Bedarf die Anzahl der Threads festgelegt werden (hier z.B. 4).

```
1 #pragma omp parallel for num_threads(4)
2   for (int i=0; i<Anzahl_Additionen; i++) {
3     Arbeit[i]->addiere();
4   }
```

Quellcode 3.3: Parallelisierung mit `parallel for` Konstrukt.

Die zweite Möglichkeit zur Parallelprogrammierung stellt das `parallel sections` Konstrukt dar (siehe Quellcode 3.4). `Parallel sections` dient nicht zur Parallelisierung von Schleifen, sondern bietet die Möglichkeit mehrere Codeblöcke (`sections`) parallel rechnen zu lassen.
Dazu wird in Zeile 1 mit der Anweisung `#pragma omp parallel sections` ein Abschnitt zur Parallelrechnung eröffnet. Mit der Anweisung `#pragma omp section` in den Zeilen 3, 5, 7 und 9 können die parallel zu rechnenden Abschnitte definiert werden.

Diese Methode hat analog zu der Methode der Schleifenparallelisierung den gravierenden Nachteil, dass die Struktur der Parallelisierung äußerst statisch ist. Bereits bei der Erstellung des Quellcodes muss angeben werden, wie viele `sections` benötigt werden.

```
1  #pragma omp parallel sections
2    {
3  #pragma omp section
4      Arbeit[0]->addiere();
5  #pragma omp section
6      Arbeit[1]->addiere();
7  #pragma omp section
8      Arbeit[2]->addiere();
9  #pragma omp section
10     Arbeit[3]->addiere();
11   }
```

Quellcode 3.4: Parallelisierung mit `parallel sections` Konstrukt.

Ab dem OpenMP Standard 3.0 kann zusätzlich mit dem `task` Konstrukt parallelisiert werden. Diese Möglichkeit stellt eine der wichtigsten Neuerungen von OpenMP dar und macht OpenMP flexibel und dynamisch.
Mit der Compiler Direktive `#pragma omp task` kann an jeder Stelle des Quellcodes die Anweisung gegeben werden, dass an dieser Stelle eine neue Aufgabe (Task) erstellt werden soll, die einem beliebigen freien Thread zugewiesen wird. Somit ist die Beschränkung auf starre Schleifen oder auf `sections` Abschnitte mit fester Größe aufgehoben.
Ein Beispiel für das `task` Konstrukt ist im Quellcode 3.5 zu finden. Mit der Anweisung `#pragma omp parallel` wird ein neuer Abschnitt zur Parallelrechnung angelegt. Während des Programmablaufs werden an dieser Stelle n Threads (je nach Rechner und weiteren Anweisungen im Quellcode) erstellt. Die Schleife (Zeile 5) darf jedoch nur von einem einzelnen Thread durchlaufen werden, da die Schleife ansonsten n-mal abgearbeitet werden würde. Dies wird mit der Direktive `#pragma omp single` (Zeile 3) erreicht.
Bei jedem Schleifendurchgang `i` wird durch die Direktive `#pragma omp task` eine neue Aufgabe angelegt, die von einem beliebigen freien Thread bearbeitet werden kann.

```
1  #pragma omp parallel
2    {
3  #pragma omp single
4      {
5        for (int i=0; i<Anzahl_Additionen; i++) {
6  #pragma omp task
7          Arbeit[i]->addiere();
8        }
9      }
10   }
```

Quellcode 3.5: Parallelisierung mit `task` Konstrukt.

Auch wenn der `task` Konstrukt im Quellcode 3.5 auf die Parallelisierung einer Schleife angesetzt worden ist, bietet er die notwendige Freiheit zur Parallelisierung von komplexem Quellcode.

Neben den vorgestellten Direktiven zur Aufteilung von Arbeit, bietet OpenMP auch entsprechende Methoden zur Synchronisation und zur gemeinsamen Verwendung von Daten. Da die genaue Darstellung des OpenMP Standards den Umfang der Arbeit überschreiten würde, wird auf diesen verwiesen [16].

3.7 Lastverteilung

Einer der entscheidenden Verursacher des administrativen Overheads ist die Lastverteilung, die bei jeder Parallelisierung notwendig ist. In diesem Abschnitt soll der Begriff *Lastverteilung* erklärt und die wesentlichen Vertreter erläutert werden.

In Abbildung 3.13 ist der Ablauf eines parallelen Codeabschnitts schematisch dargestellt. Es wird eine Stelle des Quellcodes betrachtet, an der ein paralleler Codeschnitt beginnt. Die Abarbeitung des Quellcodes bis zu dieser Stelle erfolgt durch den sogenannten *Master*-Thread.
In einem ersten Schritt werden die parallel zu berechnenden Aufgaben durch den *Master* Prozess erstellt. Dieser Vorgang läuft ebenfalls rein sequentiell ab. Darauf

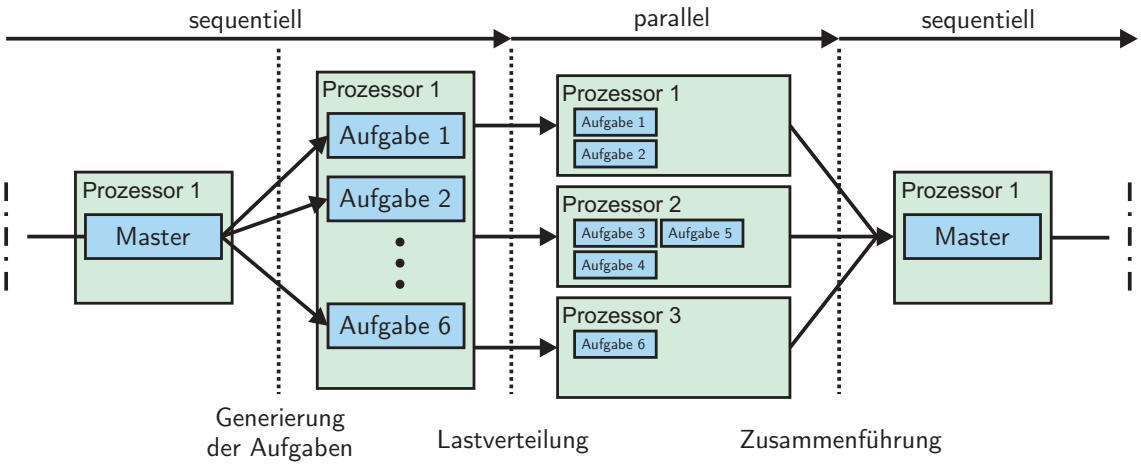


Abbildung 3.13: Schematische Darstellung der Lastverteilung.

folgend werden die Aufgaben so auf die verfügbaren Prozessoren verteilt, dass die Gesamtrechenzeit des parallelen Abschnitts möglichst gering ist. Dies ist gleichbedeutend damit, dass die Prozessoren möglichst gleichmäßig belastet werden sollen. Nach der parallelen Abarbeitung des parallelen Codeabschnitts werden die Ergebnisse gegebenenfalls zusammengeführt. Nach der Zusammenführung erfolgt die weitere Abarbeitung des Quellcodes durch den *Master* Prozess.

Die Herausforderung liegt nun darin, wie eine gegebene Anzahl an Aufgaben so verteilt werden kann, dass die Gesamtrechenzeit auf n Prozessoren möglichst gering ist.

Im Wesentlichen existieren hierzu zwei Klassen an Algorithmen. *Statische* Lastverteilungsalgorithmen und *dynamische* Lastverteilungsalgorithmen.
Statische Lastverteilungsalgorithmen sind dadurch gekennzeichnet, dass sie zu Beginn des parallelen Abschnitts die Aufgaben in zahlenmäßig gleich große Untermengen aufteilen und den Prozessoren somit gleichmäßig zuweisen. *Statische* Lastverteilungsalgorithmen zeichnen sich durch einen sehr geringen Overhead aus. *Dynamische* Lastverteilungsalgorithmen versuchen während der Laufzeit des parallelen Codeabschnitts eine möglichst gleichmäßige Auslastung der Prozessoren zu erwirken. Der bekannteste Algorithmus ist der sogenannte *Farming* Algorithmus. In der Literatur sind eine Vielzahl möglicher Herangehensweisen genannt, die sich alle in eine der beiden Gruppen einteilen lassen. Einige Beispiele sind in den Werken [21, 101, 149, 153] zu finden.

Im Folgenden werden die statische Lastverteilung und der *Farming* Algorithmus als Stellvertreter der dynamischen Lastverteilungsalgorithmen anhand eines Beispiels erklärt.
Angenommen es soll eine parallele Schleife ausgeführt werden, die aus insgesamt acht Elementen besteht (siehe Tabelle 3.1) und für deren parallele Berechnung drei Prozessoren zur Verfügung stehen.
Die acht Elemente werden durchnummeriert und als Aufgaben 1-8 bezeichnet. An dieser Stelle sei explizit darauf hingewiesen, dass in dem folgenden Beispiel zwar die Rechenzeit für jedes Element der Schleife angegeben wird, jedoch während der laufenden Simulation nicht zur Lastverteilung zur Verfügung steht.

Tabelle 3.1: Beispiel einer Schleife mit 8 Elementen für die Lastverteilung.

Aufgabe Nr.	1	2	3	4	5	6	7	8
Rechendauer [s]	1	2	9	7	8	3	13	16

Der in Abbildung 3.14 dargestellte statische Lastverteilungsalgorithmus teilt die acht Aufgaben in zahlenmäßig gleiche Mengen auf und verteilt sie auf die drei verfügbaren Prozessoren. Somit erhält der Prozessor 1 die Aufgaben 1-3, der Prozessor 2 die Aufgaben 4-6 und der Prozessor 3 die Aufgaben 7-8. Aus dieser Lastverteilung

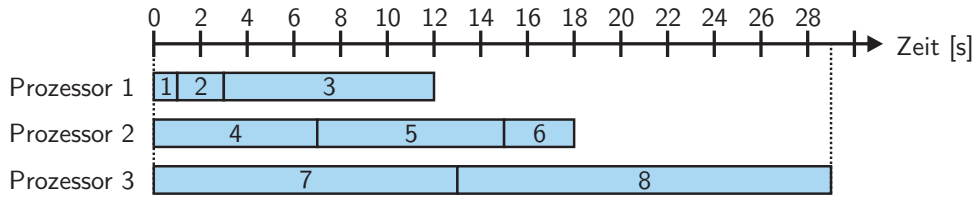


Abbildung 3.14: Statische Lastverteilung - Beispiel.

resultiert eine Gesamtrechenzeit ohne Overhead von 29 s, wobei der Overhead bei der statischen Lastverteilung als sehr gering eingestuft werden kann.

Der in Abbildung 3.15 dargestellte Lastverteilungsalgorithmus (*Farming* Algorithmus) teilt die Aufgaben dynamisch den drei Prozessoren zu. Die ersten drei Aufgaben

werden den Prozessoren sofort zugeteilt. Darauf folgend werden die verbleibenden Aufgaben der Reihe nach dem Prozessor zugeteilt, der seine vorherige Aufgabe als erster abgeschlossen hat. Aus dieser Lastverteilung resultiert eine Gesamtrechenzeit

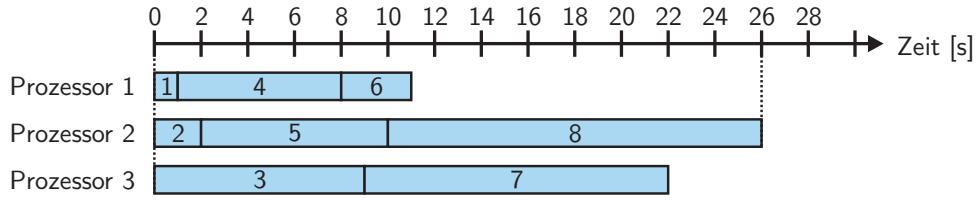


Abbildung 3.15: Dynamische Lastverteilung - Beispiel.

ohne Overhead von 26 s, wobei der Overhead bei der dynamischen Lastverteilung deutlich höher ist als bei der Statischen. In diesem Beispiel benötigt die Parallelisierung basierend auf dem statischen Algorithmus in der Regel weniger Zeit als die Parallelisierung basierend auf dem dynamischen Algorithmus, da die einzelnen Rechenzeiten sehr hoch sind. Würde es sich bei den Angaben nicht um Sekunden, sondern um Mikrosekunden handelt, so würde der Overhead deutlich stärker ins Gewicht fallen, woraus die invertierte Situation resultieren könnte.

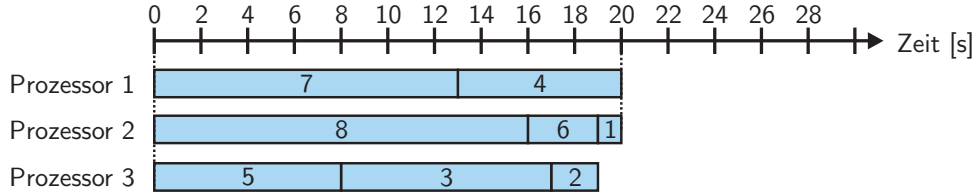


Abbildung 3.16: Optimale Lastverteilung - Beispiel.

Trotz allem stellen beide keine optimale Lastverteilung zur Verfügung. Eine mögliche optimale Lastverteilung ist in Abbildung 3.16 abgebildet. Die optimale Lastverteilung kommt auf eine Gesamtrechenzeit von nur 20 s (23 % weniger als *Farming* Algorithmus).

In Kapitel 4 wird ein speziell an die Anforderungen der Mehrkörpersimulation angepasster Lastverteilungsalgorithmus vorgestellt, der mit großer Sicherheit eine nahezu optimale Lastverteilung findet und zudem den Overhead möglichst gering hält.

4 Adaptive interne Parallelisierung

In diesem Kapitel werden zwei neue Methoden zur systeminternen Parallelisierung von dynamischen Systemen vorgestellt und analysiert.

Das Kapitel beginnt mit der Darstellung der Problemstellung der internen Parallelisierung von Mehrkörpersystemen und der grundlegenden Idee der neu entwickelten Parallelisierungsmethoden (Abschnitt 4.1). Für die parallele Mehrkörpersimulation können spezielle Randbedingungen gesetzt werden, auf denen die beiden Parallelisierungsmethoden basieren.
Die erste Parallelisierungsmethode (*Grenzzeit* Parallelisierungsmethode - Abschnitt 4.2) basiert auf dem dynamischen Lastverteilungsalgorithmus von OpenMP, der jedoch nur bedingt für die parallele Mehrkörpersimulation geeignet ist. Daher wird in Abschnitt 4.3 eine Parallelisierungsmethode vorgeschlagen (*semi-dynamische* Parallelisierungsmethode), die auf einem neu entwickelten statischen Lastverteilungsalgorithmus basiert.
Beide Methoden werden sowohl getrennt voneinander an Beispielen aus Kapitel 6 analysiert, als auch an einem industriellen Beispiel verglichen.

4.1 Systeminterne Parallelisierung

Grundlegende Idee

Wie bereits in Abschnitt 2.5 erläutert, müssen in jedem Zeitschritt der numerischen Integration von dynamischen Systemen alle notwendigen Größen einmal bzw. mehrmals neu berechnet werden. An dieser Stelle sei nochmals das Beispiel eines halbexpliziten Time-Stepping Integrationsverfahrens [59] wie auch in Abschnitt 2.5 (Methode 2.1) aufgegriffen.

Die jeweiligen Update-Prozesse sind aus der Sicht der softwareseitigen Umsetzung als Schleifen implementiert. Dies soll anhand der Neuberechnung der Massenmatrix $\boldsymbol{M}$, der Kraftrichtungsmatrix $\boldsymbol{W}$ und der Neuberechnung der rechten Seite $\boldsymbol{h}$ erklärt werden. Wie in Gleichung 4.1 dargestellt, setzen sich die globalen Systemmatrizen $\boldsymbol{M}$, $\boldsymbol{W}$ und $\boldsymbol{h}$ aus den jeweiligen Anteilen der Objekte (OBJECTS) und Verbindungen (LINKS) im System zusammen (Laufindex i). Während des Durchlaufs der Schleife über alle Elemente im System schreiben die einzelnen Elemente ihren jeweiligen Anteil in die globalen Systemmatrizen.

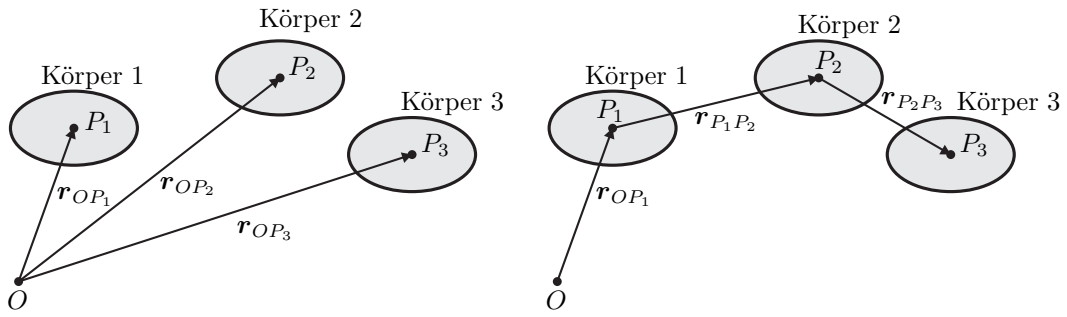
$$\boldsymbol{M} = \sum_{i=1}^{n_i} \boldsymbol{J}_i^T \boldsymbol{M}_i \boldsymbol{J}_i, \qquad \boldsymbol{h} = \sum_{i=1}^{n_i} \boldsymbol{J}_i^T \boldsymbol{h}_i, \qquad \boldsymbol{W} = \sum_{i=1}^{n_i} \boldsymbol{J}_i^T \boldsymbol{W}_i \tag{4.1}$$

An dieser Stelle wird ersichtlich, dass bei dem Schreiben der individuellen Anteile in den Speicher der globalen Matrizen sogenannte *Race Conditions* (siehe Abschnitt 3.3.2) auftreten können. Diese müssen explizit durch sogenannte *Barrieren* bzw. durch *Synchronisationspunkte* unterbunden werden. Je nach System kann dies zu Wartezeiten der einzelnen Threads und somit zu geringeren Beschleunigungen führen. In den Arbeiten [126] und [129] werden ähnliche Ansätze zur parallelen Berechnung von CVT-Getrieben verwendet. Für den gesamtem Update-Prozess werden je nach verwendeter Hardwarearchitektur Beschleunigungen von 0.83 bis 1.11 erreicht.

Analog zu der Berechnung des Vektors $\boldsymbol{h}$ und der Matrizen $\boldsymbol{M}$ und $\boldsymbol{W}$ erfolgt auch die Berechnung aller anderen globalen Systemvektoren und -matrizen.

Es ist ersichtlich, dass die jeweiligen Anteile der Objekte (OBJECTS) bzw. Verbindungen (LINKS) an den globalen Matrizen parallel berechnet werden können. Lediglich der Schreibvorgang der Ergebnisse in den Speicherbereich der globalen Matrizen muss sequentiell erfolgen. An dieser Stelle setzen die zwei systeminternen Parallelisierungsmethoden an.

Die Modellierung von Mehrkörpersystemen kann in die absolute und relative Modellierung eingeteilt werden. Der wesentliche Unterschied liegt darin, dass bei der absoluten Modellierung die Kinematik jedes einzelnen Körpers bezüglich eines inertialen Koordinatensystems beschrieben wird (siehe Abbildung 4.1(a)). Bei der relativen Beschreibung wird die Kinematik eines Körpers relativ zu einem anderen Körper beschrieben (siehe Abbildung 4.1(b)). Relative Beschreibungen schränken



(a) Absolute Parametrierung. (b) Relative Parametrierung (Baumstruktur).

Abbildung 4.1: Absolute und relative Parametrierung von Mehrkörpersystemen.

die Parallelisierungsmöglichkeiten auf Ebene des System-Updates ein, da Abhängigkeiten zwischen den Körper bestehen (Reihenfolge der Auswertung spielt eine Rolle). Für den Fall der absoluten Parametrierung können in dieser Arbeit folgende Update-Prozesse parallel berechnet werden:

- **updateg**
- **updategd**

- **updateh**
- **updateSDV**
- **updateJacobians**
- **updateM**

Für den Fall der relativen Parametrierung werden die folgenden - von der gewählten Parametrisierung unabhängigen - Update-Prozesse parallel berechnet:

- **updateg**
- **updategd**

Bei der relativen Parametrierung besteht zudem die Möglichkeit die Update-Prozesse **updateh**, **updateSDV**, **updateJacobians**, und **updateM** auf Ebene des gesamten Baums parallel zu berechnen. Dies ist nur dann lohnenswert, wenn das System aus mehr als einem Baum besteht.

Herausforderungen und Probleme

Eine mögliche Herangehensweise an die Problemstellung kann darin gesehen werden, alle Schleifen der Update-Prozesse auf allen zur Verfügung stehen Rechenkernen parallel auszuführen. Diese Herangehensweise wird auch in den Arbeiten [126, 129] verfolgt. Bereits aus diesen Arbeiten geht hervor, dass die Beschleunigungen entweder sehr gering ausfallen oder die Rechenzeiten länger werden. Die Ursache liegt analog zu Abschnitt 3.5 am Parallelisierungsoverhead.

Die Herausforderung bei der systeminternen Parallelisierung von Mehrkörpersimulationen liegt folglich darin, den Overhead möglichst gering zu halten. Diese Herausforderung stellt die Grundlage der zwei in diesem Kapitel vorgeschlagenen systeminternen Parallelisierungsmethoden dar.

Rahmenbedingungen

Im Gegensatz zu allgemeinen Parallelisierungsmethoden der Informatik, die mit jedem softwaretechnischen Problem funktionieren müssen, können in der Mehrkörperdynamik hochspezialisierte Methoden entwickelt werden, die einen wesentlich besseren Beschleunigungseffekt erzielen können als allgemeine Methoden.

Dazu können in der Mehrkörperdynamik drei wesentliche Annahmen getroffen werden:

1. Die Rechenzeiten der einzelnen Schleifenelemente sind annähernd konstant bei jedem Schleifendurchlauf.
2. Die Rechenzeiten der einzelnen Schleifendurchläufe sind messbar.
3. Die Anzahl der Schleifenelemente der Schleife ändert sich nicht oder nur selten.

Zwar können die Rechenzeiten der einzelnen Schleifenelemente als annähernd konstant angenommen werden, jedoch unterscheiden sie sich erheblich in ihrer absoluten Größe. So kann die Kontaktberechnung zwischen einer Freiformkontur und einem Punkt mehrere Zehnerpotenzen länger dauern als die Kontaktberechnung zwischen zwei Kugeln (siehe Abschnitt 2.4). Dieser Umstand muss in jeder Methode explizit berücksichtigt werden.

Die zwei Methoden in diesem Kapitel erzielen eine signifikante Reduktion des Overheads aus der

- Lastverteilung (siehe Abschnitt 3.7)
- und der Verwaltung der Threads.

4.2 Grenzzeit Parallelisierungsmethode

Die *Grenzzeit* Parallelisierungsmethode basiert auf folgenden Ideen zur Reduktion des Overheads:

1. Die Anzahl der Threads bzw. Rechenkerne wird auf eine sinnvolle Anzahl limitiert.
2. Der Overhead für die Lastverteilung wird dadurch verringert, dass alle Schleifenelemente, die eine kürzere Rechenzeit benötigen als eine gewisse *Grenzzeit*, aus der Lastverteilung herausgenommen werden. Alle Elemente, die eine Rechenzeit unter dieser *Grenzzeit* haben, werden einer Schleife hinzugefügt, die rein sequentiell abgearbeitet wird. Alle restlichen Elemente werden einer Schleife hinzugefügt, die parallel abgearbeitet wird.

4.2.1 Ablauf der Methodik

Die Methodik ist ausführlich in Ablaufplan 4.2 und in kurzer Form in Algorithmus 4.1 dargestellt.

Die Methodik benötigt zwei Vorgaben:

1. `LäufeZeitmessung`: Anzahl der Schleifendurchläufe für die Zeitmessung.
2. `Grenzzeit`: Zeitgrenze zwischen sequentieller und paralleler Schleife.

Zu Beginn der Simulation liegt eine beliebige Schleife der Update-Prozesse vor, die mit der *Grenzzeit* Parallelisierungsmethode effizient parallel berechnet werden soll.

Dazu wird im ersten Schleifendurchlauf ① der Zähler für die Durchläufe der Schleife (`Durchläufe`) auf Null gesetzt und die `sequentielle` und `parallele` Schleife geleert. Darauf folgend ② wird der Zähler für die Durchläufe (`Durchläufe`) inkrementiert und die Schleifengröße gespeichert.

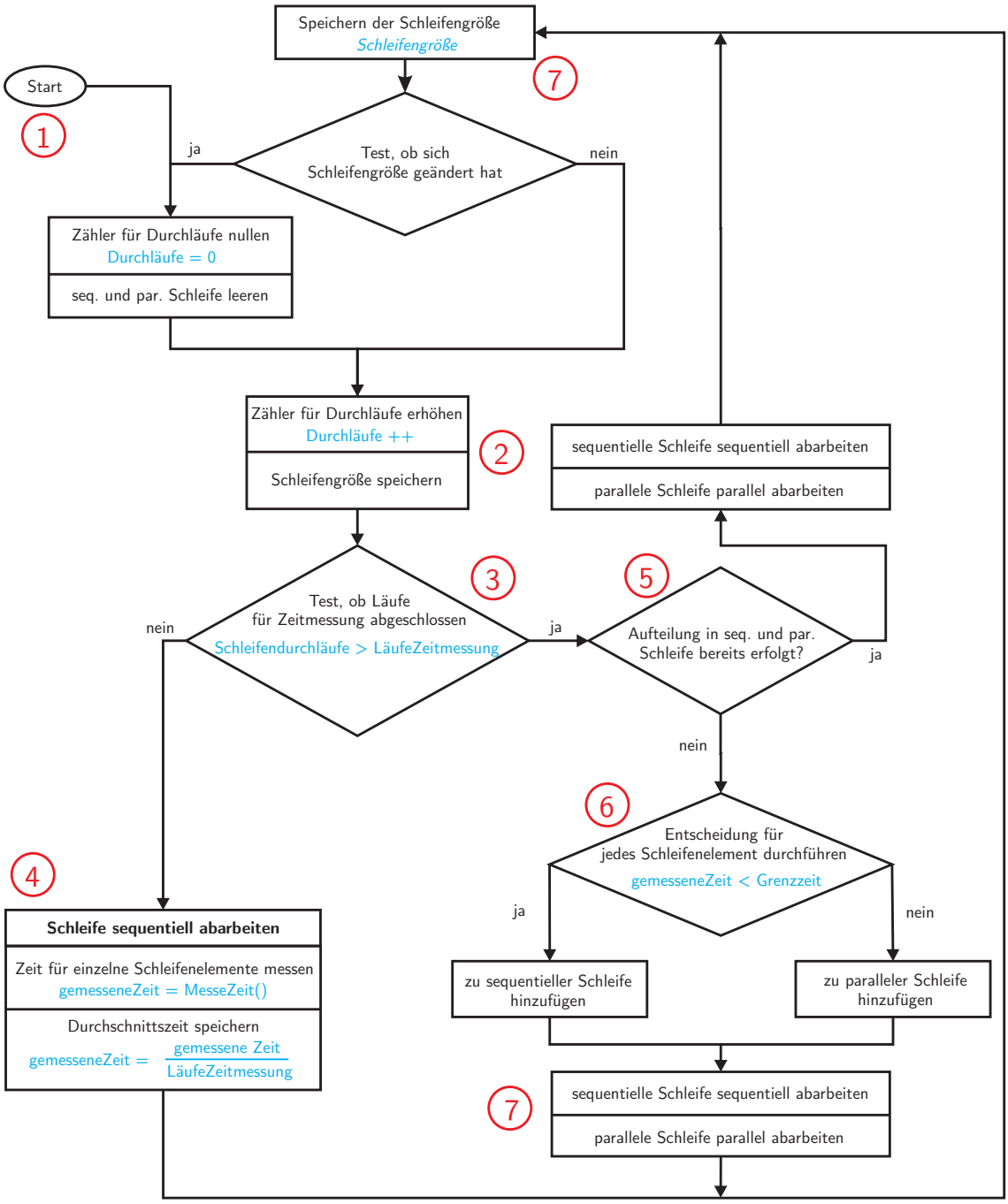


Abbildung 4.2: Ablaufplan der Grenzzeit Parallelisierungsmethode.

Die ersten `LäufeZeitmessung` Schleifendurchläufe dienen zur Messung der Rechenzeiten der einzelnen Schleifenelemente. Daher wird an Stelle ③ entschieden, ob die Zeitmessung bereits abgeschlossen ist. Falls die Zeitmessung noch nicht abgeschlossen ist, werden die Zeiten t_i für die einzelnen Schleifendurchläufe gemessen und gespeichert, dabei wird die Schleife sequentiell abgearbeitet ④. Falls die Zeitmessung abgeschlossen ist, jedoch noch keine Aufteilung in sequentielle und parallele Schleife erfolgt ist ⑤, wird an Stelle ⑥ für jedes Schleifenelement entschieden, ob es in die sequentielle oder parallele Schleife hinzugefügt wird. Die Abarbeitung der

parallelen Schleife erfolgt nun parallel und die der sequentiellen Schleife sequentiell ⑦.
Falls sowohl die Zeitmessung als auch die Aufteilung in sequentielle und parallele Schleife bereits vollzogen sind ⑤, werden die beiden Schleifen direkt abgearbeitet.
Zuletzt wird die aktuelle Schleifengröße (`Schleifengröße`) gespeichert, bevor der Schleifendurchlauf wieder von vorne beginnt.

Algorithmus 4.1 *Grenzzeit* Parallelisierungsmethode - updateg

1. Zeitmessung: Messe Rechenzeit t_i für $i = 1...n$ von $g_i = f(\boldsymbol{q},\boldsymbol{u},t)$

2. Entscheidung zwischen sequentieller und paralleler Schleife
   a) für t_i von $g_i \leq$ *Grenzzeit*: Füge g_i zu $\boldsymbol{g}_s$ (sequentielle Schleife) hinzu.
   b) für t_i von $g_i >$ *Grenzzeit*: Füge g_i zu $\boldsymbol{g}_p$ (parallele Schleife) hinzu.

3. Führe Berechnung durch
   a) Berechne $g_{i,s}$ für $i = 1...n_s$ sequentiell
   a) Berechne $g_{i,p}$ für $i = 1...n_p$ parallel

Die Methodik wird noch um zwei weitere Prozesse ergänzt, die jedoch aus Gründen der Übersichtlichkeit nicht in dem Ablaufplan 4.2 enthalten sind.

- Das parallele Abarbeiten der parallelen Schleife beginnt auf einem Rechenkern. Während der Simulation wird die Anzahl der Rechenkerne erhöht, bis keine wesentliche Steigerung der Beschleunigung mehr erzielt werden kann. Als Kriterium kann beispielsweise gesetzt werden, dass die Rechenzeit um mindestens weitere 5 % reduziert werden kann. Dadurch kann sichergestellt werden, dass kein unnötiger Overhead durch eine zu hohe Anzahl an Threads verursacht wird.
- Die Zeitmessung und die Aufteilung in sequentielle und parallele Schleife wird nach bestimmten Zeiten wiederholt.

Diese Methodik ermöglicht es, den Overhead gering zu halten, der durch die Anzahl der Threads und durch die Lastverteilung verursacht wird. Da die einzelnen Schleifenelemente sehr unterschiedliche Rechenzeiten aufweisen können, ist es notwendig eine dynamische Lastverteilung innerhalb der parallelen Schleife zu verwenden (siehe Abschnitt 3.7). Für die *Grenzzeit*-Parallelisierungsmethode wird daher der in Abschnitt 3.7 vorgestellte *Farming*-Algorithmus verwendet.

4.2.2 Anwendungsbeispiele und Diskussion

Zur Diskussion und Analyse der *Grenzzeit* Parallelisierungsmethode werden die Kontaktabstandsberechnungen (**updateg**) von drei akademischen Beispielen und einem

industriellen Beispiel herangezogen. Um die Eigenschaften der Methode anschaulich analysieren zu können, wird in den Abbildungen 4.3 bis 4.6 die auf die sequentielle Rechenzeit normierte Rechenzeit des Gesamtsystems in Abhängigkeit der verwendeten `Anzahl an Kernen` und der gesetzten `Grenzzeit` dargestellt. Im realen Simulationslauf werden die beiden Parameter automatisch bestimmt. Genauere Angaben zu den verwendeten Beispielen sind in Kapitel 6 zu finden. Alle Simulationen liefen auf der in Abschnitt 1.2 beschrieben Hardware und Software.

Spielzeugspecht an flexibler Stange

Das Beispiel *Spielzeugspecht an flexibler Stange* enthält acht Kontaktabstandsberechnungen (**updateg**).
Die Rechenzeiten der jeweiligen Kontaktpaarungen sind in Tabelle 4.1 gelistet. Die hohe Rechenzeiten der Kontaktpaarungen zwischen der Muffe und dem flexiblen Balken und des Schnabels und dem flexiblen Balken sind darin begründet, dass die Kontaktkinematik je mittels eines numerischen Suchverfahrens (Newton-Verfahren) ausgewertet werden muss, da keine analytische Lösung möglich ist.

Tabelle 4.1: Kontaktkinematiken des Beispiels *Spielzeugspecht an flexibler Stange*.

Anzahl	**Rechenzeit** $[10^{-6}\,\mathrm{s}]$	**Beschreibung**
4	~ 300	zwischen Muffe und flexiblem Balken
1	~ 400	zwischen Schnabel und flexiblem Balken
3	~ 1	Einspannung des Balkens

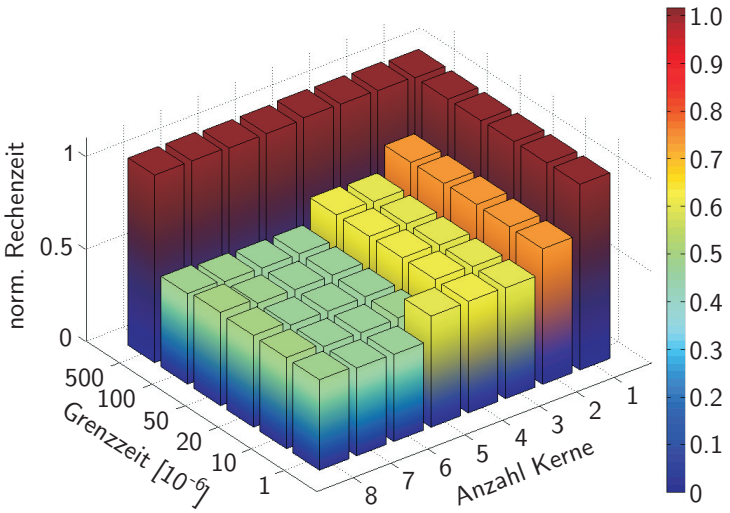


Abbildung 4.3: Analyse Grenzzeit-Parallelisierungsmethode - Spielzeugspecht.

Abbildung 4.3 zeigt die Gesamtrechenzeit (normiert auf sequentielle Rechenzeit) des Systems in Abhängigkeit der verwendeten `Anzahl an Kernen` und der gesetzten `Grenzzeit`. Die Struktur des Systems lässt erwarten, dass die fünf Abstandsberechnungen mit langer Rechenzeit sehr gut zu parallelisieren sind und die restlichen drei Abstandsberechnungen eine eher untergeordnete Rolle spielen.
Abbildung 4.3 zeigt, dass die Erhöhung der Rechenkerne bis zu keiner Anzahl von fünf die Rechenzeit maßgeblich verringert. Die weitere Erhöhung der Anzahl an Rechenkernen erzielt keine weitere Beschleunigung, da nur fünf Abstandsberechnungen mit ausreichend langer Rechenzeit im System vorliegen. Da die Rechenzeiten der Abstandsberechnungen sehr hoch sind, äußerst sich der zusätzliche Overhead durch eine Rechenkernanzahl größer fünf nur minimal in den Beschleunigungen. Die Änderung der `Grenzzeit` zwischen 10 und $100 \cdot 10^{-6}$ s führt zu keiner Änderung der Rechenzeiten, da die Rechenzeiten der Abstandsberechnungen mit hoher Rechendauer in diesem Intervall liegen. Die `Grenzzeit` höher als $500 \cdot 10^{-6}$ s zu wählen führt dazu, dass alle Abstandsberechnungen sequentiell erfolgen.

Somit kann in diesem Beispiel mit der Parameterkombination

$$\begin{aligned} \texttt{Grenzzeit} &= 10 \cdot 10^{-6}\,\mathrm{s} \\ \texttt{Anzahl Kerne} &= 5 \end{aligned}$$

eine Beschleunigung von 2.13 erzielt werden.

Dieses Beispiel ist ein Vertreter der Systeme, die sehr gutmütige Eigenschaften zur Parallelisierung aufweisen. Auch die Wahl einer sehr hohen Anzahl an Rechenkernen führt zu keiner wesentlich schlechteren Beschleunigung aufgrund des zusätzlichen Overheads.

Gleitende Elemente auf flexiblem Balken

Das Beispiel *Gleitende Elemente auf flexiblem Balken* enthält 321 Abstandsberechnungen, deren Rechenzeiten in Tabelle 4.2 aufgeführt sind.

Tabelle 4.2: Kontaktkinematiken des Beispiels *Gleitende Elemente auf flexiblem Balken*.

Anzahl	**Rechenzeit** $[10^{-6}\,\mathbf{s}]$	**Beschreibung**
160	$\sim$ 120	zwischen Elementen und flexiblem Balken
158	$\sim$ 3.5	zwischen Elementen untereinander
3	$\sim$ 5	inertiale Einspannung des Balkens

An diesem Beispiel wird ersichtlich, dass die korrekte Wahl der `Grenzzeit` einen signifikanten Einfluss auf die erzielten Beschleunigungen aufweist. Wird die `Grenzzeit` auf 10^{-6} s gesetzt, so werden sowohl die Kontaktberechnungen zwischen den Elementen untereinander als auch zwischen den Elementen und dem flexiblen Balken parallel berechnet. Abbildung 4.4 zeigt, dass die Wahl einer `Grenzzeit` höher oder

gleich $10 \cdot 10^{-6}$ s niedrigere Rechenzeiten zulässt. Diese Wahl führt dazu, dass die Berechnungen zwischen den Elementen untereinander sequentiell ablaufen und somit der Parallelisierungsoverhead für diese Berechnungen entfällt. Die Anzahl der Rechenkerne kann in diesem Beispiel auf acht erhöht werden, da eine genügend große Anzahl an Kontaktberechnungen mit hoher Rechenzeit vorliegt.

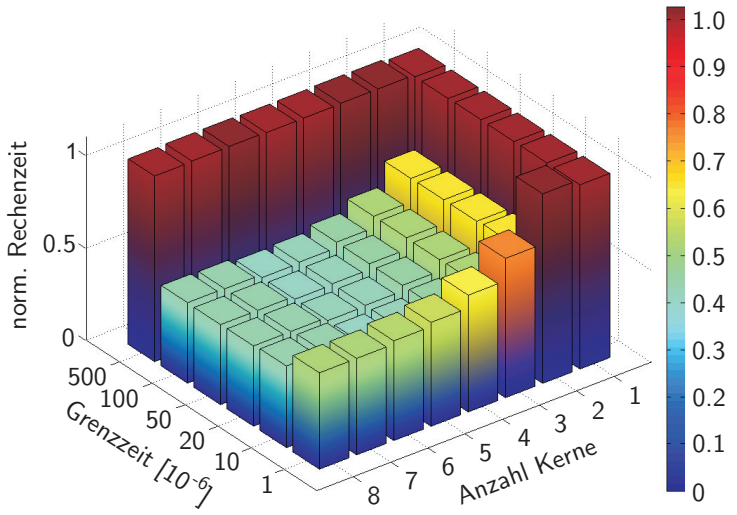


Abbildung 4.4: Analyse *Grenzzeit* Parallelisierungsmethode - Gleitende Elemente auf flexiblem Balken.

Dieses Beispiel ist ein Vertreter der Systeme, bei denen die `Anzahl Kerne` sehr hoch gewählt werden kann, jedoch die Wahl der `Grenzzeit` einen entscheidenden Einfluss auf die erzielten Rechenzeiten aufweist. Somit kann in diesem Beispiel mit der Parameterkombination

$$\texttt{Grenzzeit} = 20 \cdot 10^{-6}\,\text{s}$$
$$\texttt{Anzahl Kerne} = 8$$

eine Beschleunigung von 2.38 erzielt werden.

Kugeln in Schale

Das Beispiel *Kugeln in Schale* enthält 6776 Abstandsberechnungen, die jeweils analytisch ausgewertet werden und somit eine äußerst kurze Rechenzeit benötigen (etwa $5 \cdot 10^{-6}$ s).
Somit muss in diesem Beispiel der Parallelisierungsoverhead außerordentlich gering gehalten werden, damit eine effiziente Parallelisierung möglich ist. Abbildung 4.5 zeigt, dass mit der Parameterkombination

$$\texttt{Grenzzeit} = 7 \cdot 10^{-6}\,\text{s}$$

$$\texttt{Anzahl Kerne} = 3$$

eine Beschleunigung von 1.56 erzielt werden kann.

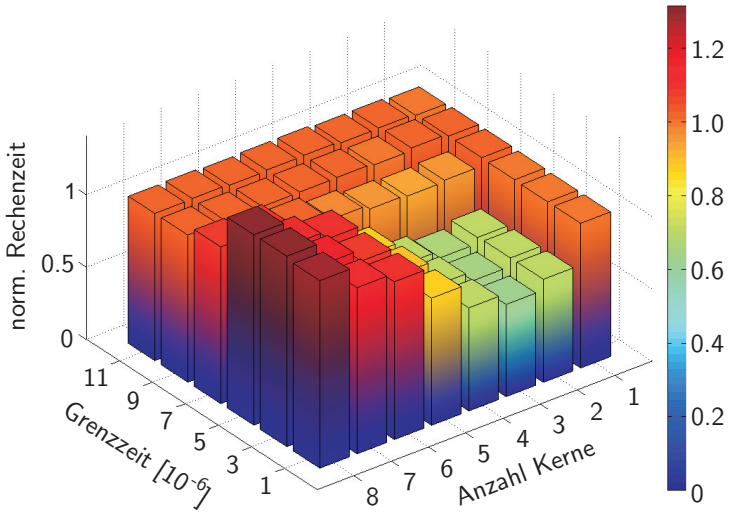


Abbildung 4.5: Analyse *Grenzzeit* Parallelisierungsmethode - Kugeln in Schale.

Eine Wahl der Anzahl an Rechenkernen höher als drei führt zu signifikant höheren Rechenzeiten, die ab sechs Rechenkernen die sequentielle Rechenzeit des Systems überschreiten. Oberhalb einer `Grenzzeit` von $7 \cdot 10^{-6}\,\mathrm{s}$ werden alle Abstandsberechnungen sequentiell ausgeführt.

Dieses Beispiel ist ein Vertreter der Systeme, bei denen sowohl die `Grenzzeit` als auch die `Anzahl Kerne` einen entscheidenden Einfluss auf parallele Rechenzeit des Systems haben. Insbesondere zeigt dieses Beispiel, dass die Herangehensweise alle Schleifenelemente parallel auf der maximalen Anzahl an Kernen rechnen zu lassen,

$$\begin{aligned} \texttt{Grenzzeit} &= 1 \cdot 10^{-6}\,\mathrm{s} \\ \texttt{Anzahl Kerne} &= 8 \end{aligned}$$

zu einer Beschleunigung deutlich unterhalb eins führt.

CVT (Continuously Variable Transmission)

Zuletzt wird ein großes industrielles Beispiel (CVT-Getriebe) betrachtet, das 645 Abstandsberechnungen beinhaltet. Die Analyse der Parallelisierungsmethode ist analog zu den vorherigen Beispielen in Abbildung 4.6 dargestellt.

Eine genaue Analyse des Diagramms wie in den vorherigen Beispielen ist aufgrund

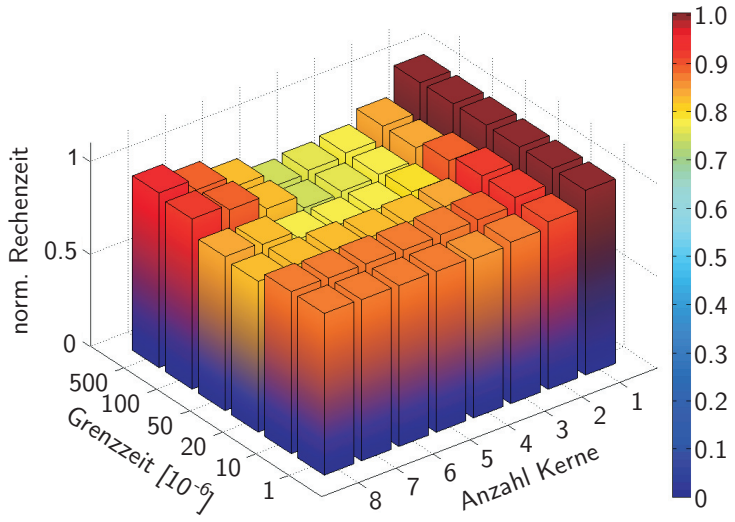


Abbildung 4.6: Analyse *Grenzzeit* Parallelisierungsmethode - CVT.

der Komplexität des Simulationsmodells nicht möglich. Mit der Parameterkombination

$$\texttt{Grenzzeit} = 100 \cdot 10^{-6}\,\mathrm{s}$$
$$\texttt{Anzahl Kerne} = 5$$

kann eine Beschleunigung von 1.72 erzielt werden kann.

Diskussion der Methode

Abschnitt 4.2.2 zeigt anhand von drei akademischen Beispielen und einem komplexen industriellen Anwendungsbeispiel, dass mit der *Grenzzeit* Parallelisierungsmethode signifikante Beschleunigungen erzielt werden können.

Weiteres Entwicklungspotential kann an zwei Stellen gesehen werden. Der Overhead durch die notwendige dynamische Lastverteilung innerhalb der parallelen Schleife kann durch den Einsatz spezieller Lastverteilungsalgorithmen reduziert werden. Die Suche der Parameter der *Grenzzeit* Parallelisierungsmethode (`Anzahl an Kernen` und `Grenzzeit`) erfolgt mittels einer globalen Suche innerhalb vorgegebener Grenzen. Eine Parallelisierungsmethode, die keine Bestimmung dieser Parameter benötigt, würde ebenfalls höhere Beschleunigungen zulassen.
An diesen Stellen setzt die zweite Parallelisierungsmethode in Abschnitt 4.3 an.

4.3 Semi-dynamische Parallelisierungsmethode

Die *semi-dynamische* Parallelisierungsmethode basiert auf folgenden Ideen zur Reduktion des Overheads:

1. Die Anzahl der Threads bzw. Rechenkerne wird auf eine sinnvolle Anzahl limitiert.
2. Der Overhead für die Lastverteilung wird dadurch verringert, dass ein eigener statischer Lastverteilungsalgorithmus verwendet wird, der bei bestimmten Ereignissen neu ausgewertet wird (semi-dynamisch).

4.3.1 Ablauf der Methodik

Die grundlegende Idee der *semi-dynamischen* Parallelisierungsmethode ist es, die Schleifenstruktur gänzlich aufzulösen.
Dazu werden in einem ersten Schritt die Rechenzeiten der einzelnen Schleifenelemente gemessen. Basierend auf diesen Zeiten werden durch einen neuen statischen Lastverteilungsalgorithmus sogenannte *Arbeitspakete* erstellt, die daraufhin parallel abgearbeitet werden können.

Ein *Arbeitspaket* besteht aus einer beliebigen Anzahl an Schleifenelementen. Die Anzahl der *Arbeitspakete* entspricht der Anzahl an verwendeten Rechenkernen, somit wird jedes *Arbeitspaket* durch einen Rechenkern ausgeführt. Die Verteilung der Schleifenelemente auf die *Arbeitspakete* muss derart erfolgen, dass die Rechenzeiten der *Arbeitspakete* möglichst gleich sind. Dies erfolgt durch den statischen Lastverteilungsalgorithmus, der in Abschnitt 4.3.2 beschrieben ist.
Unter der Annahme, dass sich die Rechenzeiten der einzelnen Schleifenelemente während der Schleifendurchläufe nur unwesentlich ändern, muss der statische Lastverteilungsalgorithmus nur einmalig ausgeführt werden. Dies ermöglicht den Overhead durch die Lastverteilung signifikant zu reduzieren. Ändert sich die Schleifengröße, muss der Lastverteilungsalgorithmus neu ausgeführt werden.

Analog zu der *Grenzzeit* Parallelisierungsmethode erfolgt auch bei der *semi-dynamischen* Parallelisierungsmethode die Abarbeitung der Arbeitspakete vorerst auf einem Rechenkern. Die Erhöhung der Anzahl an Rechenkernen erfolgt solange, bis sich die Beschleunigung durch einen weiteren Rechenkern um weniger als 5 % erhöhen lässt.

Der Ablauf der Methode ist analog zu der *Grenzzeit* Parallelisierungsmethode in Abbildung 4.7 dargestellt, wobei wiederum aus Gründen der Übersichtlichkeit die Variation der Anzahl der Kerne nicht dargestellt ist.

Zu Beginn ① wird der Zähler für die Anzahl der Durchläufe (`Durchläufe`) auf null gesetzt. Darauf folgend wird der Zähler inkrementiert und die Größe der Schleife (`Schleifengröße`) gespeichert ②.

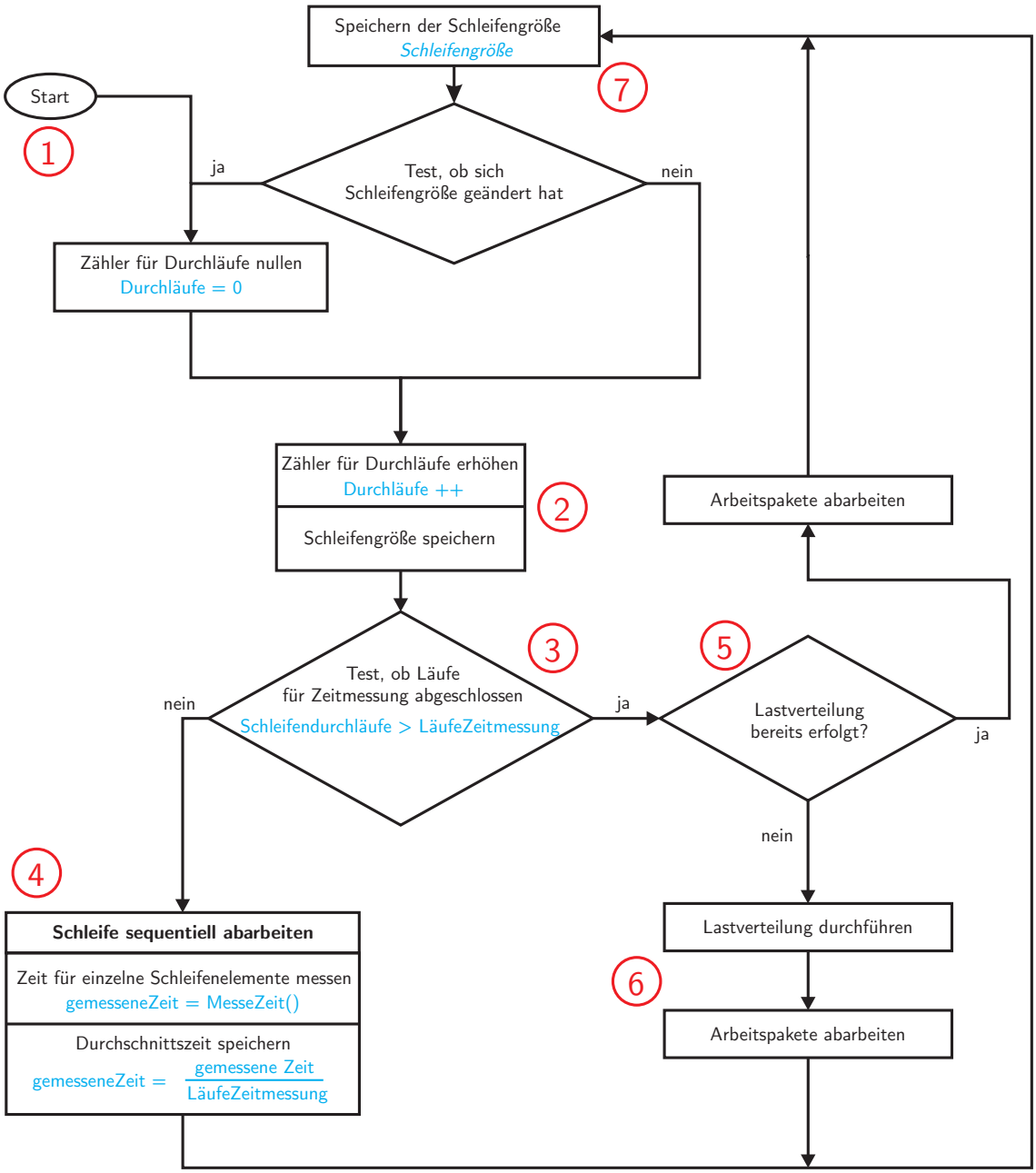


Abbildung 4.7: Ablaufplan der semi-dynamischen Parallelisierungsmethode.

Falls die Durchläufe für die Zeitmessung (`LäufeZeitmessung`) noch nicht abgeschlossen sind ③, werden die einzelnen Rechenzeiten t_i der Schleifenelemente gemessen, wobei die Schleife währenddessen sequentiell abgearbeitet wird. Falls die Durchläufe für die Zeitmessung abgeschlossen sind und die Lastverteilung noch nicht ausgeführt worden ist ⑤, wird die Lastverteilung ausgeführt, bevor die Arbeitspakete abgearbeitet werden ⑥. Falls sowohl die Durchläufe für die Zeitmessung als auch die Lastverteilung abgeschlossen sind ⑤, so werden die Arbeitspakete direkt abgearbeitet.

Die *semi-dynamische Parallelisierungsmethode* weist folgende drei Vorteile gegen-

über der *Grenzzeit* Parallelisierungsmethode auf:

1. Der Overhead durch die Lastverteilung kann nahezu auf ein absolutes Minimum reduziert werden.
2. Es muss im Gegensatz zu der *Grenzzeit* Parallelisierungsmethode keine `Grenzzeit` bestimmt werden.
3. Die Wahrscheinlichkeit einer optimalen Lastverteilung ist sehr hoch.

4.3.2 Lastverteilung

Für die *semi-dynamische* Parallelisierungsmethode wird ein statischer Lastverteilungsalgorithmus benötigt. Die Problemstellung bzw. die Aufgabe des Lastverteilungsalgorithmus kann wie folgt formuliert werden:

Gegeben sei eine Schleife mit n Schleifenelementen. Die Schleifenelemente haben die individuellen Rechenzeiten t_i. Verteile die n Schleifenelemente auf j Arbeitspakete, sodass die Rechenzeiten der Arbeitspakete möglichst gleich sind.

Die Problemstellung soll an einem sehr einfachen Beispiel gezeigt werden. An dieser Stelle sei explizit darauf hingewiesen, dass für die Anschaulichkeit ein sehr einfaches und kleines Beispiel herangezogen wird. In realen Simulationen können die Schleifengrößen in Bereichen von mehreren tausend Elementen liegen, wobei sich die Rechenzeiten der Schleifenelemente stark unterscheiden können.

Es soll eine Schleife mit $n = 6$ Schleifenelementen mit den individuellen Rechenzeiten $t_{1...6}$ auf $j = 3$ Arbeitspakete verteilt werden (siehe Tabelle 4.3).

Tabelle 4.3: Beispiel einer Schleife mit 6 Elementen für die Lastverteilung.

Schleifenelement Nr.	1	2	3	4	5	6
Rechendauer [s]	4	5	1	3	5	9

Abbildung 4.8 zeigt schematisch eine mögliche optimale Lastverteilung für dieses Beispiel. Das Arbeitspaket 1 besteht aus zwei Schleifenelementen, das Arbeitspaket 2 aus drei Schleifenelementen und das Arbeitspaket 3 aus nur einem Schleifenelement. Dadurch kann erreicht werden, dass alle Arbeitspakete je eine Rechenzeit von $t = 9\,\mathrm{s}$ Sekunden aufweisen.

Eine mathematische Formulierung der gegebenen Problemstellung lautet wie folgt:

Teile eine gegebene Anzahl ganzer Zahlen in eine Gruppe von Untermengen, sodass die jeweiligen Summen innerhalb der Untermengen möglichst ähnlich sind.

Dabei ist unter der *gegebenen Anzahl ganzer Zahlen* die Menge der individuellen Rechenzeiten t_i der Schleifenelemente gemeint und unter *der Gruppe von Untermengen* die Gruppe an Arbeitspaketen.

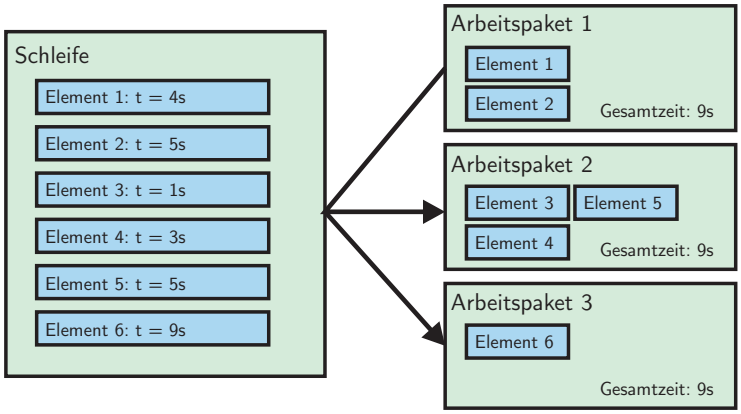


Abbildung 4.8: Schematische Darstellung der Lastverteilung.

Diese mathematische Problemstellung gehört in den Forschungsbereich des sogenannten *Multi-Way-Number-Partitioning*. Für weitere Informationen können die Arbeiten von Korf [89–92] als Standardwerke betrachtet werden.
Die Problemstellung gehört dabei in die Gruppe der *NP vollständigen* mathematischen Probleme. *NP vollständige* Probleme sind dadurch gekennzeichnet, dass in polynomieller Zeit keine optimale Lösung gefunden werden kann, jedoch in polynomieller Zeit getestet werden kann, ob eine gefundene Lösung optimal ist. Unter polynomieller Zeit wird ein polynomieller Zusammenhang zwischen der benötigten Rechenzeit und den Eingangsgrößen der Problemstellung verstanden.

Da keine Methodik zur Verfügung steht, mittels derer in berechenbarer Zeit eine optimale Lösung gefunden werden kann, werden Näherungsverfahren herangezogen. Die Literatur bietet dazu im Wesentlichen zwei Algorithmen:

- *Greedy* Algorithmen
- *Karmarkar-Karp* Algorithmen

Beide Verfahren finden nach einer berechenbaren Zeit eine Näherungslösung unbekannter Qualität. Von beiden Verfahren existieren zusätzlich sogenannte *complete* Varianten, die die optimale Lösung in nicht deterministischer Zeit finden. Neben diesen vier Methoden existieren noch weitere Abwandlungen und Kombinationen der Algorithmen. Dazu gehört auch der *RNP*-Algorithmus (Recursive Number Partitioning), der eine vielversprechende Lösung darstellt.

Einen kurzen Überblick über die Grundideen hinter den Methoden sollen die nächsten Abschnitte geben.

Greedy Algorithmus

Im ersten Schritt des *Greedy* Algorithmus werden zunächst alle Zahlen in absteigender Reihenfolge sortiert. Im nächsten Schritt werden alle Zahlen mit der Größten

beginnend nacheinander auf die Untermengen verteilt, dabei wird die nächste Zahl immer der Untermenge zugewiesen, die die kleinste Summe aufweist.

Complete Greedy Algorithmus

Der normale *Greedy* Algorithmus findet nach genau einem Durchlauf seine Lösung. Der *complete Greedy* Algorithmus findet dagegen die optimale Lösung, jedoch kann nicht berechnet werden, wie lange er dafür benötigt.
Er beginnt ebenfalls mit der Sortierung der Zahlen ihrer Größe nach. Im Gegensatz zum normalen *Greedy* Algorithmus wird die nächste Zahl jedoch nicht nur der Untermenge hinzugefügt, die die kleinste Summe hat, sondern auch der mit der 2. kleinsten, 3. kleinsten, etc. Summe. Dadurch wird eine Art Baum aufgespannt, den der Algorithmus absucht.
Durch spezielle Abbruchkriterien werden dabei Äste des Baums abgebrochen, die keine bessere Lösung, als die bereits Gefundene, mehr erzielen können.

Karmarkar-Karp Algorithmen

Bezeichne k die Anzahl an Untermengen und n die Anzahl der zu verteilenden Zahlen, dann besteht eine sogenannte Partition aus k Untermengen, die jeweils Zahlen enthalten. Eine Untermenge wird durch die Summe ihrer Elemente (Zahlen) repräsentiert. Eine Partition wird normiert, indem von den Summen aller Untermengen die Summe der kleinsten Untermenge abgezogen wird. Damit entspricht die Summe einer Untermenge zwar nicht mehr der tatsächlichen Summe der darin enthaltenen Zahlen, jedoch bleibt die Differenz zwischen der größten und der kleinsten Summe gleich.
Zu Beginn wird für jede Zahl eine Partition gebildet, wobei jeweils eine Untermenge der Partition die entsprechende Zahl enthält, während alle anderen Untermengen der Partitionen leer bleiben. Im nächsten Schritt werden die Partitionen in absteigender Reihenfolge der Summe ihrer Zahlen sortiert.
Nun werden die beiden größten Partitionen miteinander kombiniert. Dazu werden die größte Untermenge der einen Partition und die kleinste Untermenge der anderen Partition kombiniert, d.h. die neue Untermenge ergibt sich aus der Summe der beiden Untermengensummen. Der Reihenfolge nach wird die 2. größte und die 2. kleinste Untermenge der Partitionen, die 3. größte und die 3. kleinste Untermenge der Partitionen, usw. miteinander kombiniert. Sind alle Untermengen kombiniert, wird die neu entstandene Partition normiert und wieder in die Menge aller Partitionen einsortiert. Solange noch mehr als eine Partition vorhanden sind, werden weiterhin Partitionen nach dem dargestellten Verfahren kombiniert. Die verbleibende Partition stellt die gefundene Lösung des Zahlenaufteilunsgsproblems dar.

Auf die Darstellung des *complete Karmarkar-Karp* Algorithmus wird an dieser Stelle verzichtet, da er im Weiteren nicht verwendet wird.

RNP Algorithmus

Aufgrund der Komplexität des *RNP* Algorithmus ist es nicht möglich den Algorithmus innerhalb dieses Abschnitts zu erläutern. Daher wird auf die Veröffentlichung [92] von Korf verwiesen, die sich explizit mit diesem Algorithmus beschäftigt. Er basiert auf einem *complete 2-way Karmarkar-Karp* Algorithmus in Verbindung mit weiteren numerischen Methoden der Wahrscheinlichkeitsrechnung (Binärbäume).

Vergleich der Algorithmen

Als Entscheidungsgrundlage, welcher der Algorithmen für die *semi-dynamische* Parallelisierungsmethode herangezogen wird, dienen die Implementierungen des

- *Greedy* Algorithmus,
- *Karmarkar-Karp* Algorithmus,
- *complete Greedy* Algorithmus,
- und des *RNP* Algorithmus.

Die vier Algorithmen sind aus der Sicht der Softwareentwicklung optimiert worden. Der *Greedy* und *Karmarkar-Karp* Algorithmus liefern ihre Ergebnisse nach einem Durchlauf, der *complete Greedy* und *RNP* Algorithmus benötigen eine Laufzeitgrenze, nach welcher der Lauf spätestens abgebrochen wird.

Die Anwendung der vier Algorithmen als statische Lastverteilungsalgorithmen erfolgt auf alle in Abschnitt 4.3.3 diskutierten Update-Prozesse. Als Laufzeitgrenze für den *complete Greedy* und *RNP* Algorithmus wird 1 s gesetzt. Falls die Algorithmen vor dieser Laufzeitgrenze eine optimale Lösung finden, wird der Lauf abgebrochen. Die Ergebnisse der Vergleiche sind in den Tabellen 4.4 bis 4.8 angegeben. Dabei wird für jeden Update-Prozess und jeden betrachteten Algorithmus die Abweichung zwischen der Rechenzeit des Arbeitspakets mit der geringsten und mit der höchsten Rechenzeit in Promille angegeben. Die Anzahl der Arbeitspakete richtet sich nach dem betrachteten Update-Prozess und ist in Abschnitt 4.3.3 angegeben.

Eine optimale Lastverteilung ist mit möglichst ähnlichen Rechenzeiten der Arbeitspakete gleichzusetzen. Je nach System ist jedoch eine Gleichverteilung der Rechenzeiten nicht möglich. Dies wird beispielsweise an einem Update-Prozess deutlich, der aus zwei Elementen mit den Rechenzeiten $100 \cdot 10^{-6}$ s und $1 \cdot 10^{-6}$ s besteht. Die Rechenzeitdifferenz zwischen dem Arbeitspaket mit der höchsten Rechenzeit und mit der geringsten Rechenzeit ist mindestens $99 \cdot 10^{-6}$ s.
Somit müssen in den Tabellen die Promille-Angaben der Verfahren untereinander verglichen werden. Die absolute Größe der Angaben spielt eine untergeordnete Rolle.

Tabelle 4.4: Vergleich der Lastverteilungsalgorithmen am Beispiel *2D CVT*.

Update-Prozess	**Greedy** [‰]	**KK** [‰]	**complete Greedy** [‰]	**RNP** [‰]
updategd(SetValued)	0.11	0.11	0.11	0.11
updategd(SetValued)	0.11	0.11	0.11	0.11
updateg(SingleValued)	0.45	0.03	0.45	0.03
updateh(SingleValued)	0.38	0.11	0.38	0.08
updateSDV	4.35	3.90	4.35	3.90
updateJacobians(object)	1.02	0.38	0.94	0.38

Tabelle 4.5: Vergleich der Lastverteilungsalgorithmen am Beispiel *Spielzeugspecht*.

Update-Prozess	**Greedy** [‰]	**KK** [‰]	**complete Greedy** [‰]	**RNP** [‰]
updateg(SetValued)	110.76	110.76	110.76	110.76

Tabelle 4.6: Vergleich der Lastverteilungsalgorithmen am Beispiel *Ventiltriebsdynamik*.

Update-Prozess	**Greedy** [‰]	**KK** [‰]	**complete Greedy** [‰]	**RNP** [‰]
updateg(SetValued)	2.15	0.00	1.72	0.00
updategd(SetValued)	1.41	1.41	1.41	1.41
updateh(object)	2.44	1.71	1.95	0.49
updateSDV	4.30	2.66	3.27	0.20
updateJacobians(link)	0.73	0.73	0.73	0.73
updateJacobians(object)	1.16	0.87	0.87	0.29

Tabelle 4.7: Vergleich der Lastverteilungsalgorithmen am Beispiel *Perlenkette*.

Update-Prozess	**Greedy** [‰]	**KK** [‰]	**complete Greedy** [‰]	**RNP** [‰]
updateg(SetValued)	8.86	6.80	8.86	6.18
updateh(object)	747.09	747.09	747.09	747.09
updateJacobians(link)	27.02	23.44	25.39	14.97
updateJacobians(object)	37.25	35.01	35.01	30.97

Tabelle 4.8: Vergleich der Lastverteilungsalgorithmen am Beispiel *Kugeln in Schale*.

Update-Prozess	**Greedy** [‰]	**KK** [‰]	**complete Greedy** [‰]	**RNP** [‰]
updateg(SetValued)	0.10	0.08	0.10	0.06
updateJacobians(object)	6.71	5.68	6.58	0.13

Tabelle 4.9: Vergleich der Lastverteilungsalgorithmen am Beispiel *Rotordynamik*.

Update-Prozess	**Greedy** [‰]	**KK** [‰]	**complete Greedy** [‰]	**RNP** [‰]
updateg(SingleValued)	31.91	10.64	10.64	10.64
updateh(object)	34.09	34.09	0.00	0.00
updateSDV	219.22	219.22	219.22	219.22
updateJacobians(object)	243.87	243.87	219.35	219.35
updateM(object)	40.20	40.20	30.15	30.15

Tabelle 4.10: Vergleich der Lastverteilungsalgorithmen am Beispiel *Gleitende Elemente*.

Update-Prozess	**Greedy** [‰]	**KK** [‰]	**complete Greedy** [‰]	**RNP** [‰]
updateg(SetValued)	0.12	0.01	0.12	0.01
updateSDV	11.61	10.63	11.28	10.63
updateJacobians(object)	11.87	11.42	11.87	11.42

Um einen Vorschlag abzuleiten, welcher Algorithmus für die *semi-dynamische* Parallelisierungsmethode am besten geeignet ist, müssen zwei Kriterien betrachtet werden:

- Die Qualität der statischen Lastverteilung, also die Differenz zwischen dem Arbeitspaket mit der niedrigsten und mit der höchsten Rechenzeit.
- Die benötigte Rechenzeit der Algorithmen.

Abbildung 4.9 zeigt am Beispiel des Update-Prozesses **updateSDV** des Beispiels *Ventiltriebsdynamik* die erzielten verbleibenden Differenzen der Rechenzeiten zwischen dem Arbeitspaket mit der höchsten und mit der niedrigsten Rechenzeit in Abhängigkeit der Laufzeit der Algorithmen.

Aus den Ergebnissen können folgende Schlussfolgerungen gezogen werden:

- Der *Greedy* Algorithmus benötigt die geringste Rechenzeit, erzielt jedoch in der Regel die schlechtesten Ergebnisse.
- Der *Karmarkar-Karp* Algorithmus benötigt geringfügig mehr Rechenzeit als der *Greedy* Algorithmus, erzielt jedoch bessere Ergebnisse.
- Der *complete Greedy* Algorithmus liefert innerhalb der betrachteten Zeitspanne keine überzeugenden Ergebnisse.
- Der *RNP* Algorithmus findet sehr schnell eine sehr gute Lösung, jedoch ist nicht sichergestellt, wie viel Zeit der Algorithmus im Allgemeinen zur Ermittlung der optimalen Lösung benötigt.

Aufgrund der undefinierten Laufzeit des *complete Greedy* und *RNP* Algorithmus eignen sich diese Algorithmen nicht für die *semi-dynamische* Parallelisierungsmethode.

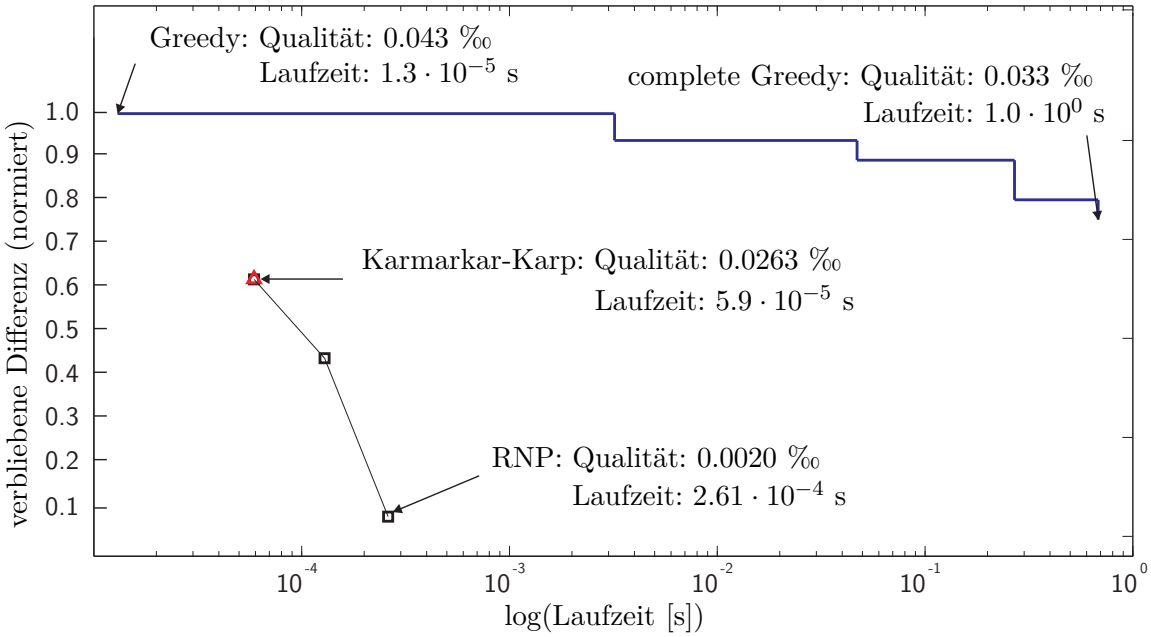


Abbildung 4.9: Rechenzeiten der Lastverteilungsalgorithmen.

Die wichtigste Erkenntnis liegt darin, dass die Algorithmen sich zwar relativ zueinander sehr wohl in der Qualität der Ergebnisse unterscheiden, jedoch absolut gesehen alle Algorithmen in allen Beispielen sehr gute Lösungen finden. Betrachtet man beispielsweise den **updateg(SetValued)** Update-Prozess der *Ventiltriebssimulation*, so findet der *RNP* Algorithmus die optimale Lösung (Rechenzeiten der Arbeitspakete sind identisch) und der normale *Greedy* Algorithmus eine Lösung, bei der sich die Rechenzeiten des Arbeitspakets mit der höchsten und mit der niedrigsten Rechenzeit um 2.15 ‰ unterscheiden. Relativ gesehen findet der *RNP* Algorithmus eine deutlich bessere Lösung als der normale *Greedy* Algorithmus. Absolut gesehen ist jedoch eine Rechenzeitdifferenz von 2.15 ‰ vernachlässigbar.

Daher wird auf den normalen *Greedy* und den *Karmarkar-Karp* Algorithmus für die statische Lastverteilung zurückgegriffen. Beide Algorithmen werden parallel ausgeführt und die bessere Lösung fließt in die Parallelisierungsmethode ein.

In Tabelle 4.11 sind die wichtigsten Vor- und Nachteile der Algorithmen zusammengefasst.

Tabelle 4.11: Vor- und Nachteile der statischen Lastverteilungsalgorithmen.

Algorithmus	**Vorteile**	**Nachteile**
Greedy	- sehr schnell	- Qualität der Ergebnisse gering
complete Greedy		- sehr langsam - Rechendauer nicht absehbar
Karmarkar-Karp	- schnell	- Qualität der Ergebnisse gering
RNP	- sehr gute Ergebnisse - schnell	- Rechendauer nicht absehbar

4.3.3 Anwendungsbeispiele und Diskussion

In diesem Kapitel wird die Leistungsfähigkeit und Effizienz der *semi-dynamischen* Parallelisierungsmethode gezeigt.

Dazu wird die Parallelisierungsmethode auf alle Beispiele, die in Kapitel 6 dargestellt sind, angewendet. Bei jedem Beispiel werden diejenigen Update-Prozesse betrachtet, die einen wesentlichen Anteil an der Gesamtrechenzeit aufweisen.
In den Tabellen werden jeweils der betrachtete **Update-Prozess**, die erreichte **Beschleunigung** bezogen auf den jeweiligen Update-Prozess, die verwendete **Anzahl Kerne**, die erreichte **Effizienz** und der **Anteil**, den der betrachtete Update-Prozess an der Gesamtrechenzeit hat, angegeben.

Folgende Update-Prozesse werden betrachtet:

- **updateg(SetValued)**
  Berechnung der Kontaktabstände aller mengenwertigen Kraftgesetze.
- **updategd(SetValued)**
  Berechnung der Kontaktabstandsgeschwindigkeiten aller mengenwertigen Kraftgesetze.
- **updateg(SingleValued)**
  Berechnung der Kontaktabstände aller einwertigen Kraftgesetze.
- **updategd(SingleValued)**
  Berechnung der Kontaktabstandsgeschwindigkeiten aller einwertigen Kraftgesetze.
- **updateh(SingleValued)**
  Berechnung der Anteile am Vektor der einwertigen externen, internen und gyroskopischen Kräfte $\boldsymbol{h}$ aller einwertigen Kraftgesetze (LINKS).
- **updateh(object)**
  Berechnung der Anteile am Vektor der einwertigen externen, internen und gyroskopischen Kräfte $\boldsymbol{h}$ aller Objekte (OBJECTS).
- **updateSDV**
  Berechnung der Körperkinematiken.
- **updateJacobians(Link)**
  Berechnung der JACOBI-Matrizen $\boldsymbol{J}$ aller LINKS.
- **updateJacobians(object)**
  Berechnung der JACOBI-Matrizen $\boldsymbol{J}$ aller OBJECTS.
- **updateM**
  Berechnung aller Anteile an der Massenmatrix $\boldsymbol{M}$.

Den Tabellen 4.12 bis 4.18 kann entnommen werden, dass mit der *semi-dynamischen* Parallelisierungsmethode sehr gute Beschleunigungen und sehr hohe Effizienzwerte erzielt werden können.

Tabelle 4.12: Ergebnisse des Beispiels *2D CVT*.

Update Prozess	Beschleunigung	Anzahl Kerne	Effizienz	Anteil
updateg(SetValued)	5.56	10	0.55	43.5 %
updategd(SetValued)	1.72	7	0.24	1.5 %
updateg(SingleValued)	3.96	9	0.44	7.7 %
updateh(SingleValued)	3.83	10	0.38	6.5 %
updateSDV	6.50	10	0.65	15.0 %
updateJacobians(object)	5.11	10	0.51	17.6 %

Tabelle 4.13: Ergebnisse des Beispiels *Spielzeugspecht an flexibler Stange*.

Update Prozess	Beschleunigung	Anzahl Kerne	Effizienz	Anteil
updateg(SetValued)	4.0	6	0.80	60.6 %

Tabelle 4.14: Ergebnisse des Beispiels *Ventiltriebsdynamik*.

Update Prozess	Beschleunigung	Anzahl Kerne	Effizienz	Anteil
updateg(SetValued)	3.25	4	0.81	10.1 %
updategd(SetValued)	1.36	4	0.34	1.4 %
updateh(object)	3.38	5	0.68	6.8 %
updateSDV	3.34	5	0.67	8.6 %
updateJacobians(Link)	1.32	4	0.33	5.2 %
updateJacobians(object)	3.52	5	0.70	5.8 %

Tabelle 4.15: Ergebnisse des Beispiels *Perlenkette*.

Update Prozess	Beschleunigung	Anzahl Kerne	Effizienz	Anteil
updateg(SetValued)	4.76	9	0.52	40.5 %
updateh(object)	1.14	2	0.57	7.3 %
updateJacobians(Link)	1.46	4	0.37	15.1 %
updateJacobians(object)	2.85	4	0.71	5.7 %

Tabelle 4.16: Ergebnisse des Beispiels *Rotordynamik*.

Update Prozess	Beschleunigung	Anzahl Kerne	Effizienz	Anteil
updateg(SingleValued)	1.36	3	0.45	2.8 %
updateh(object)	1.23	2	0.62	4.8 %
updateSDV	1.48	2	0.74	19.3 %
updateJacobians(object)	1.85	3	0.62	21.3 %
updateM	1.69	3	0.56	5.4 %

Tabelle 4.17: Ergebnisse des Beispiels *Gleitende Elemente auf flexiblem Balken.*

Update Prozess	Beschleunigung	Anzahl Kerne	Effizienz	Anteil
updateg(SetValued)	5.40	7	0.77	81.1 %
updateSDV	3.26	5	0.65	5.3 %
updateJacobians(object)	3.95	6	0.66	7.8 %

Tabelle 4.18: Ergebnisse des Beispiels *Kugeln in Schale.*

Update Prozess	Beschleunigung	Anzahl Kerne	Effizienz	Anteil
updateg(SetValued)	4.66	6	0.78	75.9 %
updateJacobians(object)	3.691	5	0.74	4.0 %

Abbildung 4.10 zeigt die Gesamtbeschleunigungen, die durch die Kombination der Parallelisierungen der einzelnen Update-Prozesse erreicht werden.

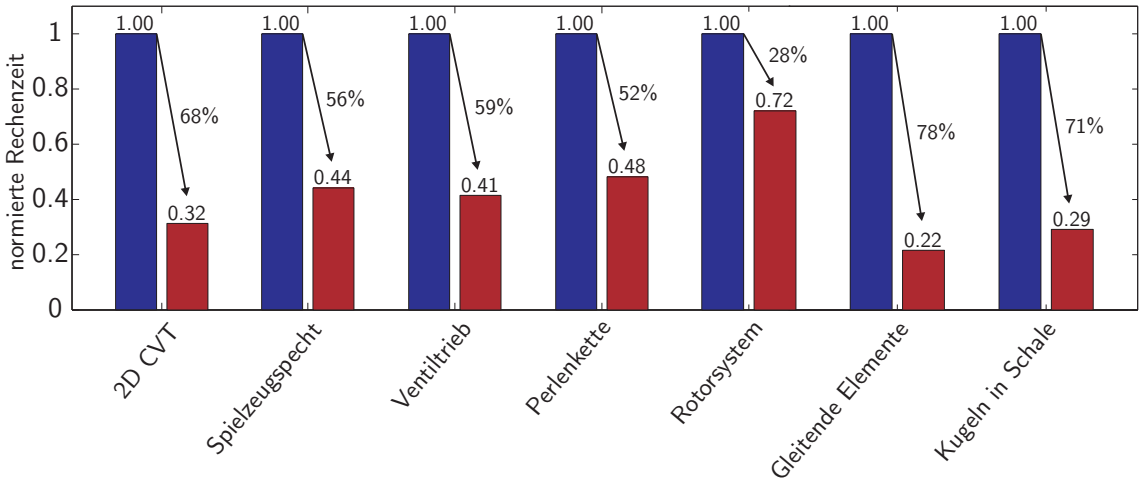


Abbildung 4.10: Semi-dynamische Parallelisierungsmethode - Beschleunigungen.

4.4 Vergleich der beiden Methodiken

Abschließend wird ein Vergleich der beiden Parallelisierungsmethoden gezeigt und diskutiert. Als Beispiel dient ein Rotorsystem, das aus vier kleinen Rotorsystemen (siehe Abschnitt 6.2.3) besteht.

Für den Vergleich werden folgende Update-Prozesse betrachtet:

- **updateh(object)**
- **updateSDV**
- **updateJacobians(object)**
- **updateM**

In Tabelle 4.19 sind die erreichten Beschleunigungen der beiden Parallelisierungsmethoden angegeben. Für die *Grenzzeit* Parallelisierungsmethode ist die Kombination aus `Grenzzeit` und `Anzahl an Kernen` in Klammern nach der Beschleunigung (Format: (`Anzahl Kerne`, `Grenzzeit`)) und für die *semi-dynamische* Parallelisierungsmethode die `Anzahl an Kernen` angegeben.

Tabelle 4.19: Vergleich der Beschleunigungen der zwei Parallelisierungsmethoden anhand einer Rotorsimulation.

Update Prozess	**Beschleunigung *Grenzzeit* Methode**	**Beschleunigung *semi-dyn.* Methode**
updateJacobians(object)	1.47 (4 Kerne, $6 \cdot 10^{-6}$)	2.97 (6 Kerne)
updateSDV	2.19 (4 Kerne, $1 \cdot 10^{-6}$)	2.21 (4 Kerne)
updateh(object)	1.30 (3 Kerne, $4 \cdot 10^{-6}$)	2.25 (3 Kerne)
updateM	2.08 (6 Kerne, $1 \cdot 10^{-6}$)	2.55 (6 Kerne)

Tabelle 4.19 zeigt, dass die *semi-dynamische* Parallelisierungsmethode in diesem Beispiels höhere Beschleunigungen erzielt als die *Grenzzeit* Parallelisierungsmethode. Diesem Effekt liegen zwei Faktoren zugrunde. Der Overhead der *semi-dynamischen* Parallelisierungsmethode fällt geringer aus als der Overhead der *Grenzzeit* Parallelisierungsmethode, da die statische Lastverteilung der *semi-dynamischen* Parallelisierungsmethode lediglich einmal zu Beginn der Simulation ausgewertet werden muss. Zusätzlich ist bei der *Grenzzeit* Parallelisierungsmethode nicht sichergestellt, dass durch die Wahl der `Grenzzeit` eine optimale Lastverteilung erzielt werden kann. Da für die *semi-dynamische* Parallelisierungsmethode keine `Grenzzeit` bestimmt werden muss, stellt sie in der Regel die bessere Wahl dar.

5 Numerische Integration

Dieses Kapitel behandelt die effiziente numerische Integration von nicht-glatten dynamischen Systemen. Dabei liegt das Hauptaugenmerk auf impliziten Verfahren mit exakter und inexakter Ableitung der rechten Seite $\boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t)$ und auf Parallelisierungsmethoden.

Das Kapitel beginnt mit einem Überblick über mögliche Integrationsverfahren für nicht-glatte dynamische Systeme und über entsprechende Literatur (Abschnitt 5.1). Die Integrationsverfahren können grundsätzlich in ereignisbasierte Verfahren und Time-Stepping Verfahren eingeteilt werden. Time-Stepping Verfahren können weiter in klassische und erweiterte Time-Stepping Verfahren unterteilt werden. In Abschnitt 5.2 werden vier gebräuchliche klassische Time-Stepping Verfahren mit Diskretisierung der Nebenbedingungen auf Geschwindigkeitsebene dargestellt.
Da in dieser Arbeit das Hauptaugenmerk auf impliziten Verfahren liegt, werden in Abschnitt 5.3 die Potentiale und Herausforderungen von impliziten Integrationsverfahren erläutert. Die wesentliche Herausforderung liegt in der Reduktion des hohen numerischen Aufwands zur Berechnung der Jacobi-Matrizen. Zwei mögliche Ansätze zur Lösung dieser Herausforderungen werden in den darauf folgenden Abschnitten vorgeschlagen. Einerseits kann mit den in Abschnitt 5.4 vorgestellten Time-Stepping Verfahren mit inexakten Jacobi-Matrizen die Auswertehäufigkeit der Jacobi-Matrizen reduziert werden und andererseits kann durch die Parallelisierungsmethoden in Abschnitt 5.6.2 die Rechenzeit für die Berechnung der Jacobi-Matrizen reduziert werden.
Neben der parallelen Berechnung der Jacobi-Matrizen werden in Abschnitt 5.6 weitere Möglichkeiten zur Parallelisierung auf der Ebene der numerischen Integration gezeigt. In Abschnitt 5.7 werden die vorgestellten Verfahren anhand von Beispielen diskutiert. Dabei wird auf die Diskretisierung der Verfahren (explizit/implizit), auf die Behandlung der Jacobi-Matrizen (exakt/inexakt), auf die Parallelisierung und auf die Ordnungserhöhung durch Extrapolation eingegangen. Das Kapitel schließt mit der Kombination der Parallelisierungsmethoden auf systeminterner Ebene (siehe Kapitel 4) und auf Integratorebene.

5.1 Überblick über Integrationsverfahren und Literatur

Numerische Integrationsverfahren für Mehrkörpersysteme mit Diskontinuitäten, die aufgrund von mengenwertigen Nebenbedingungen auftreten, können in ereignisbasierte Verfahren und Time-Stepping Methoden eingeteilt werden. Abbildung 5.1

zeigt eine Übersicht möglicher Lösungsverfahren für nicht-glatte dynamische Systeme.

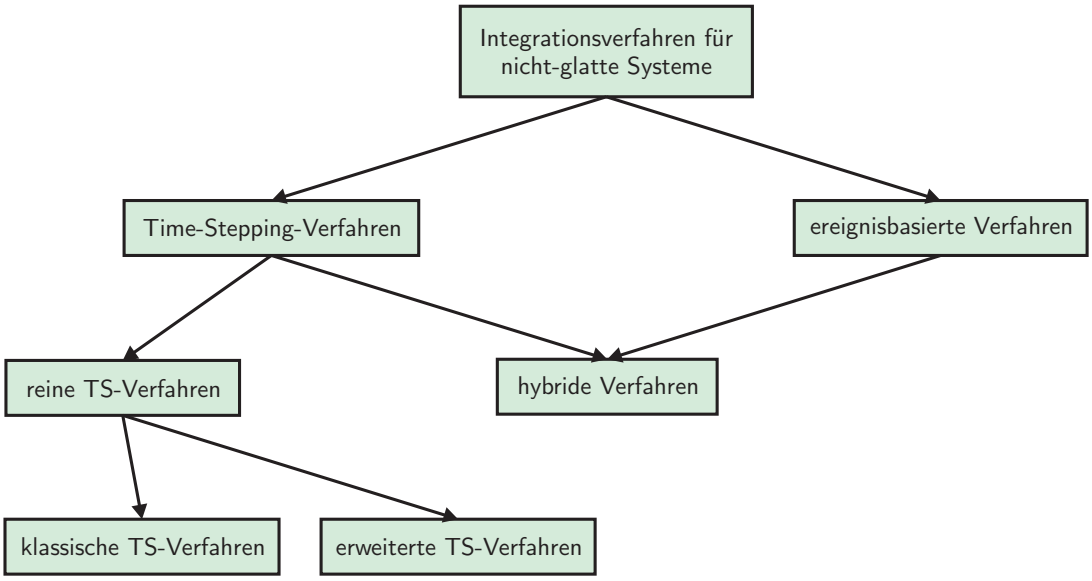


Abbildung 5.1: Übersicht: Integrationsverfahren für nicht-glatte dynamische Systeme.

Ereignisbasierte Verfahren basieren auf der Idee, Diskontinuitäten zeitlich aufzulösen und gesondert neben der normalen Zeitintegration zu behandeln. Betrachtet man das einfache Beispiel eines Balls, der auf eine Tischplatte fällt, so wird der Zeitpunkt, zu dem der Ball den Tisch berührt, aufgelöst und die Stoßgleichung gelöst. Der Integrator wird unter Berücksichtigung der Lösung der Stoßgleichung reinitialisiert. Die Integration der glatten Bereiche zwischen den Diskontinuitäten kann mit bewährten DAE[1]- oder ODE[2]-Integratoren durchgeführt werden.
Der entscheidende Vorteil von ereignisbasierten Methoden liegt in ihrer Genauigkeit, da die Stoßzeitpunkte zeitlich sehr genau aufgelöst werden und die jeweiligen Stoßgesetze zu diesen Zeitpunkten ausgewertet werden können. Dagegen stehen die Nachteile, dass das Auflösen der Stoßzeitpunkte Rechenzeit in Anspruch nimmt und sogenannte *Zeno*-Phänomene auftreten können. Darunter versteht man in der Mechanik eine unendliche Anzahl an Stößen in einem endlichen Zeitintervall. Dieses Phänomen muss explizit abgefangen und behandelt werden.

Time-Stepping Methoden basieren auf der gleichzeitigen Diskretisierung der Bewegungsgleichungen und der Nebenbedingungen. Vorteil dieser Verfahren ist, dass die Stoßzeitpunkte nicht explizit aufgelöst werden müssen. Zudem weisen Time-Stepping Methoden eine hohe numerische Robustheit und Effizienz auf. Ein Vergleich der beiden Verfahren kann in [61] nachgelesen werden, worin beide Methoden auf ein nicht-glattes industrielles Problem angewendet werden.

1 Unter DAEs (differential algebraic equations) versteht man Differentialgleichungen mit algebraischen Nebenbedingungen.
2 Unter ODEs (ordinary differential equations) versteht man gewöhnliche Differentialgleichungen.

Eine weitere Möglichkeit basiert auf der Kombination von Time-Stepping Integratoren und normalen DAE- oder ODE-Integratoren [51,85]. Diese Herangehensweise wird auch als hybrides Integrationsverfahren bezeichnet.
Dabei wird ein herkömmlicher DAE- bzw. ODE-Integrator [76,77] für die Integration der glatten Bewegungsphasen genutzt und der Time-Stepping Integrator zur Lösung der Stoßzeitpunkte. Diese Verfahren versuchen die Effizienz der hochentwickelten DAE- und ODE-Integratoren mit der numerischen Robustheit der Time-Stepping Methoden zu kombinieren. Durch die Verwendung von DAE- und ODE-Integratoren werden automatisch eine Schrittweitensteuerung und eine Ordnungserhöhung in den glatten Bewegungsphasen zur Verfügung gestellt.

Time-Stepping Verfahren können in *klassische* Time-Stepping Verfahren und *erweiterte* Time-Stepping Verfahren eingeteilt werden.
Unter den *klassischen* Verfahren versteht man das Moreau-Jean Time-Stepping [114]. Die klassischen Verfahren weisen keine Schrittweitenadaption und keine Ordnungserhöhung in den glatten Phasen auf (ausgenommen Ordnungserhöhungen durch die Verwendung von Trapezregeln).
Förg [59] behandelt in seiner Dissertation ein halb-explizites Time-Stepping Verfahren und ein linear-implizites Time-Stepping Verfahren. Die Diskretisierung der Nebenbedingungen erfolgt jeweils auf Geschwindigkeitsebene.
Zander [152] behandelt in seiner Dissertation ein linear-implizites Theta-Time-Stepping Verfahren. Darunter versteht man die Kombination aus einem expliziten und einem impliziten Verfahren mit der Gewichtung Θ und $(1-\Theta)$. Der Vorteil dieses Verfahrens liegt darin, dass in Abwesenheit von mengenwertigen Nebenbedingungen bei $\Theta = 0.5$ eine Ordnung zwei erreicht werden kann und dass das Verfahren allgemein durch den impliziten Anteil eine höhere Stabilität als ein explizites Verfahren aufweist. In diesem Zusammenhang sei auch die Arbeit von Stiegelmeyr [138] erwähnt, in der die Eigenschaften der Theta-Verfahren ausführlich diskutiert werden.
Unter den *erweiterten* Time-Stepping Verfahren werden Verfahren verstanden, bei denen die Integrationsordnung in glatten Phasen erhöht und die Zeitschrittweite adaptiert wird. Methoden dieser Art werden von Huber [82], Studer [139] und Schindler und Acary [127] beschrieben.
Huber erzielt eine Ordnungserhöhung in glatten Integrationsphasen basierend auf Extrapolationsverfahren. Als Basisverfahren nutzt er ein halb-explizites Time-Stepping Verfahren auf Geschwindigkeitsebene. Zur Schrittweitenadaption nutzt Huber in glatten Bewegungsphasen Methoden aus der ODE-Theorie und in nicht-glatten Phasen Methoden aus der DAE-Theorie.
Studer setzt in seiner Arbeit ebenfalls auf Extrapolationsmethoden zur Ordnungserhöhung.
Schindler und Acary stellen Time-Stepping Methoden auf Basis von diskontinuierlichen Galerkin-Methoden vor. Die Ordnungserhöhung wird bei diesen Verfahren durch die Anwendung von Trapezregeln erzielt. Methoden zur Schrittweitenadaption werden in dieser Arbeit [127] nicht angewendet.

In Zusammenhang mit der Integration von nicht-glatten dynamischen Systemen mit mengenwertigen Nebenbedingungen seien auch die Werke [70,104] für weiterführende

Informationen genannt. Ein umfassender Überblick ist in dem Buch von Acary und Brogliato [19] zu finden.

5.2 Klassische Time-Stepping Integrationsschemata

In diesem Abschnitt soll ein Überblick über die gebräuchlichsten klassischen Time-Stepping Verfahren gegeben werden. Ziel ist es nicht, eine ausführliche mathematische Herleitung der Verfahren zu geben, sondern eine Übersicht in der gleichen mathematischen Nomenklatur zu geben.

In Abschnitt 5.2.1 wird ein halb-explizites Time-Stepping Verfahren dargestellt, das aufgrund seiner geringen Komplexität und seines geringen numerischen Aufwands sehr häufig angewendet wird. Die halb-explizite Diskretisierung bedingt jedoch einen vergleichsweise geringen Stabilitätsbereich, der sich unter anderem in der Notwendigkeit niedrigerer Zeitschrittweiten äußert. Einen erweiterten Stabilitätsbereich bietet beispielsweise ein linear-implizites Verfahren, das in Abschnitt 5.2.1 dargestellt wird. Die Kombination der Eigenschaften einer expliziten und einer impliziten Diskretisierung kann durch Theta-Verfahren erreicht werden, deren Diskretisierung sowohl einen expliziten als auch einen impliziten Anteil umfasst (siehe Abschnitt 5.2.3). Linear-implizite Verfahren stellen eine Vereinfachung von voll-impliziten Verfahren da. Abschnitt 5.2.4 stellt ein mögliches semi-voll-implizites Verfahren dar, das aus den Ideen der vorherigen Verfahren hervorgeht.

Alle Verfahren in diesem Abschnitt basieren auf einer Diskretisierung der Nebenbedingungen auf Geschwindigkeitsebene. Der wesentliche Vorteil der Diskretisierung der mengenwertigen Nebenbedingungen auf Geschwindigkeitsebene liegt darin, dass die kontinuierliche und die diskontinuierliche Dynamik gleich behandelt werden können.
Die Darstellung der Lösung der mengenwertigen Nebenbedingungen auf Geschwindigkeitsebene würde den Rahmen dieser Arbeit überschreiten, weshalb diesbezüglich auf die Arbeiten [59] und [139] verwiesen wird. Basis aller Time-Stepping Verfahren ist die Maßdifferentialgleichung, die in Kapitel 2 eingeführt worden ist.

Aufgabe eines jeden Integrationsverfahrens ist es, von einem gegebenen Zustand $\boldsymbol{z}^l$ zu dem nächsten Zustand $\boldsymbol{z}^{l+1}$ zu gelangen. Dazu müssen die Differenzen der generalisierten Lagen

$$\int_{t^l}^{t^{l+1}} d\boldsymbol{q} = \int_{t^l}^{t^{l+1}} \dot{\boldsymbol{q}} dt = \Delta \boldsymbol{q}^l \tag{5.1}$$

und der verallgemeinerten Geschwindigkeiten

$$\int_{t^l}^{t^{l+1}} d\boldsymbol{u} = \int_{t^l}^{t^{l+1}} \dot{\boldsymbol{u}}\, dt + \int_{t^l}^{t^{l+1}} (\boldsymbol{u}^+ - \boldsymbol{u}^-)\, d\eta = \Delta \boldsymbol{u}^l \tag{5.2}$$

bekannt sein.

Unter Berücksichtigung der Maßdifferentialgleichung kann der Anteil (5.1) nach Gleichung

$$\Delta \boldsymbol{q}^l = \int_{t^l}^{t^{l+1}} \boldsymbol{T}\boldsymbol{u}\, dt$$

und der Anteil (5.2) nach Gleichung

$$\Delta \boldsymbol{u}^l = \int_{t^l}^{t^{l+1}} \boldsymbol{M}^{-1} \left(\boldsymbol{h} dt + \boldsymbol{W} d\boldsymbol{\Lambda}\right) \tag{5.3}$$

berechnet werden.

5.2.1 Halb-explizites Verfahren

Weiterführende Informationen über das halb-explizite Time-Stepping Integrationsverfahren sind in [59] zu finden.

Die Diskretisierung der Differenzen (5.1) und (5.2) erfolgt bei der halb-expliziten Diskretisierung sequentiell:

$$\int_{t^l}^{t^{l+1}} \boldsymbol{T}\,\boldsymbol{u}\, dt \approx \boldsymbol{T}^l\, \boldsymbol{u}^l\, \Delta t^l\,, \tag{5.4a}$$

$$\int_{t^l}^{t^{l+1}} \boldsymbol{M}^{-1}\, \left(\boldsymbol{h}\, dt + \boldsymbol{W} d\boldsymbol{\Lambda}\right) \approx \left(\boldsymbol{M}^{l+1}\right)^{-1} \left(\hat{\boldsymbol{h}}^{\,l+1} \Delta t + \boldsymbol{W}^{l+1}\, \boldsymbol{\Lambda}^{l+1}\right) \tag{5.4b}$$

mit $\boldsymbol{M} = \boldsymbol{M}(\boldsymbol{q}^{l+1})$, $\boldsymbol{W} = \boldsymbol{W}(\boldsymbol{q}^{l+1})$ und $\hat{\boldsymbol{h}}^{\,l+1} = \boldsymbol{h}(\boldsymbol{u}^l, \boldsymbol{q}^{l+1}, t^{l+1})$.

Daraus folgt unter Berücksichtigung der aktiven mengenwertigen Bindungen (Indexmenge $_a$) für die Berechnung der neuen generalisierten Zustände $\boldsymbol{q}^{l+1}$ und $\boldsymbol{u}^{l+1}$

$$\begin{aligned}
&\boldsymbol{q}^{l+1} = \boldsymbol{q}^l + \boldsymbol{T}^l\, \boldsymbol{u}^l\, \Delta t\,,\\
&\boldsymbol{u}^{l+1} = \boldsymbol{u}^l + \left(\boldsymbol{M}^{l+1}\right)^{-1}\, (\hat{\boldsymbol{h}}^{\,l+1}\, \Delta t + \boldsymbol{W}_a^{l+1}\, \boldsymbol{\Lambda}_a^{l+1})\,,\\
&\dot{\boldsymbol{g}}_a^{l+1} = \dot{\boldsymbol{g}}_a(\boldsymbol{q}^{l+1}, \boldsymbol{u}^{l+1}, t^{l+1})\,,\\
&\quad (\boldsymbol{\Lambda}_a^{l+1}, \dot{\boldsymbol{g}}_a^{l+1}) \in \mathcal{N}\,.
\end{aligned}$$

Die Bezeichnung *halb-explizit* lässt sich dadurch erklären, dass in die Berechnung der neuen verallgemeinerten Geschwindigkeiten $\boldsymbol{u}^{l+1}$ bereits die neuen generalisierten Positionen $\boldsymbol{q}^{l+1}$ und die neue Zeit t^{l+1} eingehen.

5.2.2 Linear-implizites Verfahren

Eine ausführliche Herleitung des Verfahrens kann in den Arbeiten [59] und [138] nachgelesen werden.

Die Diskretisierung der Differenzen (5.1) und (5.2) findet bei impliziten Verfahren im Gegensatz zu halb-expliziten Verfahren parallel statt:

$$\int_{t^l}^{t^{l+1}} \boldsymbol{T}\,\boldsymbol{u}\,dt \approx \boldsymbol{T}^{l+1}\,\boldsymbol{u}^{l+1}\,\Delta t\,, \tag{5.5a}$$

$$\int_{t^l}^{t^{l+1}} \boldsymbol{M}^{-1}(\boldsymbol{h}\,dt + \boldsymbol{W}\,d\boldsymbol{\Lambda}) \approx \left(\boldsymbol{M}^{l+1}\right)^{-1}\left(\boldsymbol{h}^{l+1}\,\Delta t + \boldsymbol{W}^{l+1}\,\boldsymbol{\Lambda}^{l+1}\right)\,. \tag{5.5b}$$

Um den Berechnungsaufwand zu reduzieren, werden folgende Annahmen für die Matrizen $\boldsymbol{M}^{l+1}$, $\boldsymbol{T}^{l+1}$ und $\boldsymbol{W}^{l+1}$ getroffen (siehe [59]):

$$\boldsymbol{M}^{l+1} = \boldsymbol{M}^{l}\,, \qquad \boldsymbol{T}^{l+1} = \boldsymbol{T}^{l}\,, \qquad \boldsymbol{W}^{l+1} = \boldsymbol{W}^{l}\,. \tag{5.6}$$

Aufgrund der Annahme (5.6) kann für $\Delta\boldsymbol{q}^l$ der explizite Zusammenhang

$$\Delta\boldsymbol{q}^l = \boldsymbol{T}^l\,\left(\boldsymbol{u}^l + \Delta\boldsymbol{u}^l\right)\,\Delta t$$

geschrieben werden.

Zusätzlich wird der numerische Aufwand dadurch reduziert, dass die voll-implizite Diskretisierung (5.5) auf eine linear-implizite Diskretisierung reduziert wird.

Für die linear-implizite Darstellung wird der Vektor $\boldsymbol{h}^{l+1}$ als Taylorreihe dargestellt, die nach dem ersten Glied abgebrochen wird.

$$\boldsymbol{h}^{l+1} \approx \boldsymbol{h}^l + \left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\right|_l \Delta\boldsymbol{q}^l + \left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{u}}\right|_l \Delta\boldsymbol{u}^l + \left.\frac{\partial\,\boldsymbol{h}}{\partial\,t}\right|_l \Delta t$$

Nach dem Einsetzen der Entwicklung für den Vektor $\boldsymbol{h}^{l+1}$ ergibt sich für das linear implizite Verfahren die Gleichung (5.7).

$$\boldsymbol{u}^{l+1} = \boldsymbol{u}^l + \left(\tilde{\boldsymbol{M}}^l\right)^{-1}\left(\tilde{\boldsymbol{h}}^{\,l} + \boldsymbol{W}_a^l\,\boldsymbol{\Lambda}_a^{l+1}\right)\,, \tag{5.7a}$$

$$\dot{\boldsymbol{g}}_a^{l+1} = \dot{\boldsymbol{g}}_a(\boldsymbol{q}^{l+1}, \boldsymbol{u}^l, t^{l+1})\,, \tag{5.7b}$$

$$(\boldsymbol{\Lambda}_a^{l+1}, \dot{\boldsymbol{g}}_a^{l+1}) \in \mathcal{N}\,, \tag{5.7c}$$

$$\boldsymbol{q}^{l+1} = \boldsymbol{q}^l + \boldsymbol{T}^l\boldsymbol{u}^{l+1}\Delta t \tag{5.7d}$$

mit

$$\tilde{\boldsymbol{M}}^l = \boldsymbol{M}^l - \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}\right|_l \Delta t - \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}}\right|_l \boldsymbol{T}^l \, \Delta t^2 \,,$$
$$\tilde{\boldsymbol{h}}^l = \boldsymbol{h}^l \Delta t + \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}}\right|_l \boldsymbol{T}^l \, \boldsymbol{u}^l \, \Delta t^2 + \left.\frac{\partial \boldsymbol{h}}{\partial t}\right|_l \Delta t^2 \,.$$

Das linear-implizite Verfahren kann gegenüber dem halb-explizitem Verfahren einen höheren Stabilitätsbereich aufweisen, fordert jedoch einen deutlich höheren numerischen Aufwand, der sich hauptsächlich in der Berechnung der Ableitungen der rechten Seite (Jacobi-Matrizen)

$$\left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}}\right|_l, \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}\right|_l \text{ und } \left.\frac{\partial \boldsymbol{h}}{\partial t}\right|_l$$

begründen lässt.

5.2.3 Linear-implizites Theta-Verfahren

Zander [152] verwendet in seiner Dissertation ein linear-implizites Theta-Time-Stepping Verfahren.

Die grundlegende Idee der Theta-Diskretisierungen liegt in der Kombination eines expliziten und eines impliziten Integrationsschritts [76]. Dabei wird der explizite Anteil mit Θ und der implizite Anteil mit $(1-\Theta)$ gewichtet.
Je nach Wahl des Parameters Θ kann das numerische Verhalten des Verfahren gezielt geändert werden. Eine ausführliche Diskussion ist in [138] zu finden. Einen besonderen Stellenwert nimmt die Wahl von $\Theta = 0.5$ ein. In Abwesenheit mengenwertiger Nebenbedingungen beträgt die Ordnung des Integrationsverfahrens zwei.

Die Diskretisierung erfolgt wiederum parallel:

$$\begin{aligned}\int_{t^l}^{t^{l+1}} \boldsymbol{T}\,\boldsymbol{u}\,dt &\approx \left((1-\Theta)\,\boldsymbol{T}^l\,\boldsymbol{u}^l + \Theta\,\boldsymbol{T}^{l+1}\,\boldsymbol{u}^{l+1}\right)\,\Delta t \\ &= \boldsymbol{T}\left(\boldsymbol{u}^l + \Theta\,\Delta\boldsymbol{u}^l\right)\,\Delta t = \Delta\,\boldsymbol{q}^l\,, \end{aligned} \tag{5.8}$$

$$\int_{t^l}^{t^{l+1}} \boldsymbol{M}^{-1}(\boldsymbol{h}\,dt + \boldsymbol{W}\,d\boldsymbol{\Lambda}) \approx \left(\boldsymbol{M}^l\right)^{-1}\left((1-\Theta)\boldsymbol{h}^l + \Theta\,\boldsymbol{h}^{l+1}\right)\Delta t + \boldsymbol{W}^l\,\boldsymbol{\Lambda}^{l+1}\Big)\,, \tag{5.9}$$

dabei werden die Annahmen

$$\boldsymbol{M}^{l+1} = \boldsymbol{M}^l\,, \qquad \boldsymbol{T}^{l+1} = \boldsymbol{T}^l\,, \qquad \boldsymbol{W}^{l+1} = \boldsymbol{W}^l \tag{5.10}$$

getroffen.

Mit der Taylorentwicklung

$$((1-\Theta)\boldsymbol{h}^l+\Theta\,\boldsymbol{h}^{l+1})\,\Delta t \approx \left(\boldsymbol{h}^l+\Theta\,\left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\right|_l\Delta\boldsymbol{q}^l+\Theta\,\left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{u}}\right|_l\Delta\boldsymbol{u}^l+\Theta\,\left.\frac{\partial\,\boldsymbol{h}}{\partial\,t}\right|_l\Delta t\right)\,\Delta t$$

ergibt sich die Vorschrift (5.11) für das linear implizite Theta-Time-Stepping Verfahren

$$\boldsymbol{u}^{l+1} = \boldsymbol{u}^l + \left(\tilde{\boldsymbol{M}}^l\right)^{-1}\,\left(\tilde{\boldsymbol{h}}^l\,\Delta t + \boldsymbol{W}_a^l\,\boldsymbol{\Lambda}_a^{l+1}\right)\,, \tag{5.11a}$$

$$\dot{\boldsymbol{g}}_a^{l+1} = \dot{\boldsymbol{g}}_a(\boldsymbol{q}^{l+1},\boldsymbol{u}^l,t^{l+1})\,, \tag{5.11b}$$

$$(\boldsymbol{\Lambda}_a^{l+1},\dot{\boldsymbol{g}}_a^{l+1}) \in \mathcal{N}\,, \tag{5.11c}$$

$$\boldsymbol{q}^{l+1} = \boldsymbol{q}^l + \boldsymbol{T}^l(\boldsymbol{u}^l+\Theta\,\Delta\boldsymbol{u}^l)\Delta t \tag{5.11d}$$

mit

$$\tilde{\boldsymbol{M}}^l = \boldsymbol{M}^l - \Theta\,\left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{u}}\right|_l\Delta t - \Theta\,\left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\right|_l\boldsymbol{T}^l\,\Delta t^2\,,$$

$$\tilde{\boldsymbol{h}}^l = \boldsymbol{h}^l\Delta t + \Theta\,\left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\right|_l\boldsymbol{T}^l\,\boldsymbol{u}^l\,\Delta t^2 + \Theta\,\left.\frac{\partial\,\boldsymbol{h}}{\partial\,t}\right|_l\Delta t\,.$$

Analog zu dem linear-impliziten Verfahren kann auch das linear-implizite Theta-Verfahren einen erweiterten Stabilitätsbereich aufgrund des impliziten Anteils (es sei denn $\Theta = 0$ wird gewählt) aufweisen (siehe [152]). Erkauft wird auch bei diesem Verfahren die höhere Stabilität durch eine höhere Rechenzeit, denn auch bei diesem Verfahren müssen die Ableitungen der rechten Seite

$$\left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\right|_l,\ \left.\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{u}}\right|_l\ \text{und}\ \left.\frac{\partial\,\boldsymbol{h}}{\partial\,t}\right|_l$$

berechnet werden.

5.2.4 Semi-voll-implizites Theta-Verfahren

In diesem Abschnitt soll die Herleitung eines semi-voll-impliziten Theta-Time-Stepping Verfahrens gezeigt werden.

Der Unterschied zwischen linear-impliziten und voll-impliziten Theta-Time-Stepping Verfahren liegt darin, dass der Vektor $\boldsymbol{h}^{l+1}$ bei voll-impliziten Verfahren nicht als Taylorreihe dargestellt wird. Anstelle dessen wird zur Lösung der impliziten Gleichung für $\boldsymbol{u}^{l+1}$ ein NEWTON-Verfahren verwendet.

Neben dem erhöhten numerischen Rechenaufwand aufgrund der Ableitungen der rechten Seite

$$\left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}}\right|_l, \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}}\right|_l \text{ und } \left.\frac{\partial \boldsymbol{h}}{\partial t}\right|_l,$$

kann bei voll-impliziten Verfahren keine analytisch reduzierte Massenwirkungsmatrix

$$\boldsymbol{G} = \boldsymbol{W}^T \boldsymbol{M}^{-1} \boldsymbol{W}$$

angegeben werden. Dies wirkt sich nachteilig aus, da die sogenannten r-Faktor-Strategien zur Lösung der mengenwertigen Nebenbedingungen nur herangezogen werden können, falls die Massenwirkungsmatrix $\boldsymbol{G}$ vor der Berechnung der Kraftreaktionen der mengenwertigen Nebenbedingungen angegeben werden kann. Für ein voll-implizites Verfahren müssten die Matrizen $\boldsymbol{M}^{l+1}$ und $\boldsymbol{W}^{l+1}$ also vorliegen, bevor die Kraftreaktion der mengenwertigen Nebenbedingungen berechnet werden. Die r-Faktor-Strategien ermöglichen eine möglichst schnelle Konvergenz der Lösung der Nebenbedingungen. Für weitere Informationen wird auf die Dissertation von Förg [59] und von Studer [139] verwiesen.

Die in den letzten Abschnitten dargestellten Verfahren ermöglichen es jedoch, in jedem Integrationsschritt die Massenwirkungsmatrix anzugeben, da entsprechende Annahmen für die Matrizen $\boldsymbol{M}^{l+1}$ und $\boldsymbol{W}^{l+1}$ getroffen werden (siehe beispielsweise (5.10)). Basierend auf dem Gedanken, dass ein linear-implizites Verfahren den ersten Newton-Iterationsschritt eines voll-impliziten Verfahrens darstellt, kann sich folglich auch unter den selben Annahmen ein voll-implizites Verfahren angeben lassen, das die Angabe einer analytisch reduzierten Massenwirkungsmatrix $\boldsymbol{G}$ in jedem Zeitschritt zulässt. Ähnliche Gedanken werden auch in dem Werk [19] verfolgt. Aufgrund der Annahmen wird die Bezeichnung *semi-voll-implizit* verwendet, da es sich streng genommen um kein voll-implizites Verfahren handelt.

Die Diskretisierung erfolgt wieder parallel:

$$\int_{t^l}^{t^{l+1}} \boldsymbol{T}\,\boldsymbol{u}\,dt \approx \left((1-\Theta)\,\boldsymbol{T}^l\,\boldsymbol{u}^l + \Theta\,\boldsymbol{T}^{l+1}\,\boldsymbol{u}^{l+1}\right)\,\Delta t\,, \tag{5.12}$$

$$\int_{t^l}^{t^{l+1}} \boldsymbol{M}^{-1}(\boldsymbol{h}\,dt + \boldsymbol{W}\,d\boldsymbol{\Lambda}) \approx \left(\boldsymbol{M}^l\right)^{-1}\left((1-\Theta)\boldsymbol{h}^l + \Theta\,\boldsymbol{h}^{l+1})\,\Delta t + \boldsymbol{W}^l\,\boldsymbol{\Lambda}^{l+1}\right)\,. \tag{5.13}$$

Mit den Annahmen für $\boldsymbol{M}^{l+1}$, $\boldsymbol{T}^{l+1}$ und $\boldsymbol{W}^{l+1}$ analog zu denen des linear-impliziten Theta-Time-Stepping Verfahrens

$$\boldsymbol{M}^{l+1} = \boldsymbol{M}^l\,, \qquad \boldsymbol{T}^{l+1} = \boldsymbol{T}^l\,, \qquad \boldsymbol{W}^{l+1} = \boldsymbol{W}^l$$

kann die Theta-Integrationsdiskretisierung (Iterationsvariable für die Integration l)

$$\boldsymbol{u}^{l+1} = \boldsymbol{u}^{l} + \left(\boldsymbol{M}^{l}\right)^{-1} \left[(1-\Theta)\,\boldsymbol{h}^{l}\Delta t + \Theta\,\boldsymbol{h}^{l+1}\,\Delta t + \boldsymbol{W}_{a}^{l}\,\boldsymbol{\Lambda}_{a}^{l+1}\right], \tag{5.15a}$$

$$\dot{\boldsymbol{g}}_{a}^{l+1} = \dot{\boldsymbol{g}}_{a}(\boldsymbol{q}^{l+1}, \boldsymbol{u}^{l}, t^{l+1}), \tag{5.15b}$$

$$(\boldsymbol{\Lambda}_{a}^{l+1}, \dot{\boldsymbol{g}}_{a}^{l+1}) \in \mathcal{N}, \tag{5.15c}$$

$$\boldsymbol{q}^{l+1} = \boldsymbol{q}^{l} + \boldsymbol{T}^{l}\left[(1-\Theta)\,\boldsymbol{u}^{l} + \Theta\,\boldsymbol{u}^{l+1}\right]\Delta t \tag{5.15d}$$

angegeben werden.

Da bei voll-impliziten Verfahren die neue Lösung $\boldsymbol{u}^{l+1}$ nicht explizit berechnet werden kann, muss die Lösung der Gleichung (5.15a) mit einem iterativen Lösungsverfahren bestimmt werden. Dazu wird ein NEWTON-Verfahren herangezogen [120]. Die Iterationsdiskretisierung (Iterationsvariable für das NEWTON-Verfahren n) des NEWTON-Verfahrens lautet

$$\begin{aligned}
\boldsymbol{q}^{n+1} &= \boldsymbol{q}^{l} + (1-\Theta)\boldsymbol{T}^{l}\boldsymbol{u}^{l}\Delta t + \Theta\boldsymbol{T}^{l}\boldsymbol{u}^{n+1}\Delta t \\
\boldsymbol{u}^{n+1} &= \boldsymbol{u}^{n} + (\tilde{\boldsymbol{M}}^{l})^{-1}(\tilde{\boldsymbol{h}}^{n}\Delta t + \boldsymbol{W}^{l}\boldsymbol{\Lambda}^{l+1}) \\
\text{mit } \tilde{\boldsymbol{M}}^{l} &= \boldsymbol{M}^{l} - \Theta^{2}\,\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\bigg|_{n}\Delta\boldsymbol{T}^{l}\Delta t^{2} - \Theta\,\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{u}}\bigg|_{n}\Delta t \\
\text{und } \tilde{\boldsymbol{h}}^{n} &= -\frac{\boldsymbol{M}^{l}}{\Delta t}\boldsymbol{u}^{n} + \frac{\boldsymbol{M}^{l}}{\Delta t}\boldsymbol{u}^{l} + \Theta^{2}\,\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\bigg|_{n}\Delta t\boldsymbol{T}^{l}\boldsymbol{u}^{n} - \Theta^{2}\,\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\bigg|_{n}\Delta t\boldsymbol{T}^{l}\boldsymbol{u}^{l} \\
&\quad - \Theta\,\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\bigg|_{n}\boldsymbol{q}^{n} + \Theta\,\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\bigg|_{n}\boldsymbol{q}^{l} - \Theta\boldsymbol{h}^{l} + \Theta\boldsymbol{h}^{n} + \boldsymbol{h}^{l} + \Theta\,\frac{\partial\,\boldsymbol{h}}{\partial\,\boldsymbol{q}}\bigg|_{n}\boldsymbol{T}^{l}\Delta t\boldsymbol{u}^{l}\,.
\end{aligned}$$

5.3 Potentiale und Herausforderungen impliziter Integrationsverfahren

Implizite Integrationsverfahren sind weit verbreitet für die Integration von numerisch steifen Systemen. Die Literatur bietet keine eindeutige Definition des Begriffs *numerisch steif*. Eine mögliche Definition lautet:

Die Integrationsschrittweite von numerisch steifen Systemen wird nicht durch die Toleranzgrenzen des Integrationsverfahrens festgelegt, sondern durch seine Stabilitätseigenschaften.

Diese Definition lässt sich an den Beispielen aus Kapitel 6 nachvollziehen. Sowohl das akademische Beispiel *Perlenkette* als auch das industrielle Beispiel *Rotordynamik* benötigen bei der Verwendung eines halb-expliziten Time-Stepping Verfahrens (siehe Abschnitt 5.2.1) eine Integrationsschrittweite von $\Delta t = 1 \cdot 10^{-6}\,\mathrm{s}$ oder geringer um stabil integriert werden zu können. Die Genauigkeitsanforderungen können dabei problemlos eingehalten werden.

Eine Integrationsschrittweite von $\Delta t = 1 \cdot 10^{-6}\,\mathrm{s}$ zieht sehr lange Rechenzeiten mit sich, die oftmals nicht toleriert werden können. Einen Ausweg bieten implizite Integrationsverfahren, die einen deutlich größeren Stabilitätsbereich aufweisen. Durch den erweiterten Stabilitätsbereich können größere Integrationsschrittweiten genutzt werden, die wiederum geringere Rechenzeit bedeuten. Für nähere Information zu der Stabilität von numerischen Integrationsverfahren wird auf die Bücher [76] und [77] verwiesen.

Implizite Verfahren bieten einerseits das Potential höhere Integrationsschrittweiten zu nutzen und dadurch Rechenzeit einzusparen, andererseits benötigen sie aber wie bereits in den Abschnitten 5.2.2, 5.2.3 und 5.2.4 erwähnt die partiellen Ableitungen der rechten Seite (Jacobi-Matrizen). In der Mechanik sind dies die partiellen Ableitungen des Vektors $\boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t)$ nach den generalisierten Lagen $\boldsymbol{q}$, den verallgemeinerten Geschwindigkeiten $\boldsymbol{u}$ und der Zeit t.

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial h_1}{\partial q_1}, & \frac{\partial h_1}{\partial q_2}, & \cdots, & \frac{\partial h_1}{\partial q_n} \\ \frac{\partial h_2}{\partial q_1}, & \frac{\partial h_2}{\partial q_2}, & \cdots, & \frac{\partial h_2}{\partial q_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial h_n}{\partial q_1}, & \frac{\partial h_n}{\partial q_2}, & \cdots, & \frac{\partial h_n}{\partial q_n} \end{bmatrix} \tag{5.17a}$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{u}} = \begin{bmatrix} \frac{\partial h_1}{\partial u_1}, & \frac{\partial h_1}{\partial u_2}, & \cdots, & \frac{\partial h_1}{\partial u_n} \\ \frac{\partial h_2}{\partial u_1}, & \frac{\partial h_2}{\partial u_2}, & \cdots, & \frac{\partial h_2}{\partial u_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial h_n}{\partial u_1}, & \frac{\partial h_n}{\partial u_2}, & \cdots, & \frac{\partial h_n}{\partial u_n} \end{bmatrix} \tag{5.17b}$$

$$\frac{\partial \boldsymbol{h}}{\partial t} = \begin{bmatrix} \frac{\partial h_1}{\partial t} \\ \frac{\partial h_2}{\partial t} \\ \vdots \\ \frac{\partial h_n}{\partial t} \end{bmatrix} \tag{5.17c}$$

(5.17d)

Somit steht dem Potential der *höheren Integrationsschrittweite* die Herausforderung der *Berechnung der* Jacobi-*Matrizen* gegenüber.

Numerischer Aufwand der Jacobi-Matrizen

Es existieren unterschiedliche Ansätze zur Berechnung der Jacobi-Matrizen. Grundlegend können diese Ansätze in analytische Methoden und in numerische Methoden unterteilt werden. Analytische Methoden versuchen die Jacobi-Matrizen für alle Elemente im System analytisch zu berechnen. Dies ist oftmals nur mit unangebracht hohem Aufwand oder gar nicht möglich. Daher werden zur Berechnung der Jacobi-Matrizen in der Regel numerische Verfahren herangezogen. Die partiellen Ableitungen lassen sich sehr einfach mit dem Differenzenquotienten berechnen. Angewandt

auf die Ableitung des Vektors $\boldsymbol{h}$ nach der ersten generalisierten Position q_1 ergibt sich

$$\left.\frac{\partial \boldsymbol{h}}{\partial q_1}\right|_{(\boldsymbol{q},\boldsymbol{u},t)} \approx \frac{\boldsymbol{h}(\boldsymbol{q}+\boldsymbol{e}_1\sqrt{\epsilon},\boldsymbol{u},t)-\boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t)}{\sqrt{\epsilon}}$$

für den Differenzenquotienten. Mit ϵ der Maschinengenauigkeit und $\boldsymbol{e}_1$ dem Einheitsvektor in Richtung q_1.

An dieser Darstellung lässt sich erkennen, dass die rechte Seite $\boldsymbol{h}$ einmal pro Spalte der JACOBI-Matrix neu ausgewertet werden muss. Dies bedeutet einen erheblichen numerischen Aufwand. Für Systeme mit einer geringen Anzahl an Freiheitsgraden ist die Bedeutung der zusätzlichen Rechenzeit zu vernachlässigen, für größere Systeme nimmt sie maßgeblich zu. Zur Verdeutlichung wird in Abbildung 5.2 der Anteil der Berechnung der JACOBI-Matrizen an der Gesamtsimulationsdauer für einen Drei-Massen-Schwinger und für einen Ventiltrieb dargestellt.

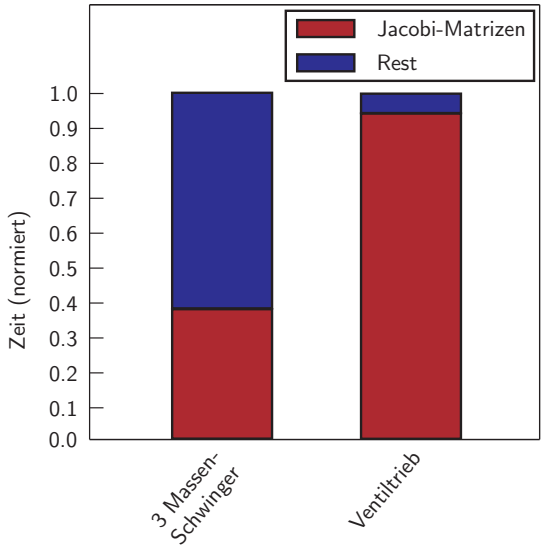


Abbildung 5.2: Anteil der Berechnung der JACOBI-Matrizen an der Gesamtsimulationsdauer.

Die Abbildung zeigt, dass bereits bei einem Drei-Massenschwinger der Anteil an der Gesamtsimulationsdauer für die Berechnung der JACOBI-Matrizen erheblich ist, jedoch durch eine höhere Integrationsschrittweite ausgeglichen werden kann. Bei dem Beispiel eines Ventiltriebs nimmt die Berechnung der JACOBI-Matrizen über 90 % der Gesamtrechendauer ein. Um diesen Aufwand auszugleichen müssen beachtlich höhere Integrationsschrittweiten genutzt werden.

Folglich müssen Methoden entwickelt werden, um den Anteil der Berechnung der JACOBI-Matrizen an der Gesamtrechendauer zu reduzieren. In dieser Arbeit werden zur Erreichung dieses Ziels zwei Ansätze verfolgt. In Abschnitt 5.6.2 wird auf die parallelisierte Berechnung der JACOBI-Matrizen eingegangen und in Abschnitt

5.4 werden zwei Time-Stepping Verfahren vorgestellt, die keine Neuberechnung der JACOBI-Matrizen in jedem Zeitschritt erfordern.

5.4 Implizite Integrationsschemata mit inexakten Jacobi-Matrizen

In diesem Abschnitt werden zwei Time-Stepping Verfahren vorgestellt, die es erlauben die JACOBI-Matrizen nicht in jedem Zeitschritt neu auszuwerten. In Abschnitt 5.4.1 wird ein linear-implizites Verfahren vorgestellt, dass die Auswertung der JACOBI-Matrizen an die Schrittweitensteuerung des Integrators koppelt und in Abschnitt 5.4.2 wird ein semi-voll-implizites Verfahren vorgestellt, dass die JACOBI-Matrizen nur nach Bedarf auswertet.

5.4.1 Makro/Mikro Time-Stepping Integrationsschema

Das Makro/Mikro-Time-Stepping Verfahren basiert auf dem in Abschnitt 5.2.3 dargestellten linear-impliziten Theta-Time-Stepping Verfahren. Die grundlegende Idee ist, die JACOBI-Matrizen nicht in jedem Zeitschritt auszuwerten, sondern in möglichst vielen Zeitschritten zu approximieren.

Aus der gewöhnlichen ODE-Theorie und insbesondere aus den linear-impliziten Runge-Kutta-Verfahren, Rosenbrock-Verfahren und W-Verfahren lässt sich folgern, dass es *unter Umständen* erlaubt ist, Näherungen für die JACOBI-Matrizen zu nutzen, sodass trotzdem die Integrationsordnung des Verfahrens erhalten bleibt. Die W-Verfahren gehen dabei sogar so weit, dass nahezu beliebige JACOBI-Matrizen genutzt werden können. Für weitere Informationen wird hierzu auf die Bücher von HAIRER [76,77] verwiesen. Zusätzlich können die Arbeiten [34,112,136,148] herangezogen werden. In diesen Arbeiten wird unter anderem vorgeschlagen, die Auswertung der JACOBI-Matrizen an die Schrittweitensteuerung des Integrators zu koppeln.

Dieser Vorschlag kann auch bei dem betrachteten linear-impliziten Time-Stepping Verfahren angewendet werden. Zusätzlich wird in den nächsten Abschnitten eine Herangehensweise vorgeschlagen, um den Fehler, der durch die inexakten JACOBI-Matrizen entsteht, möglichst gering zu halten. Abweichend von den zitierten Werken, werden in dem Makro/Mikro-Time-Stepping Verfahren die JACOBI-Matrizen nicht für eine bestimmte Zeit konstant gehalten, sondern innerhalb eines Makro-Schrittes linear interpoliert.

Das Schema des Verfahrens ist in Abbildung 5.3 gezeigt. An einem gegebenen Zustand z_n wird zunächst ein größerer Makro-Schritt mit der Schrittweite H zum neuen Zustand $\tilde{z}_{n+1}$ ausgeführt. Darauf folgend werden an dem neuen Zustand die JACOBI-Matrizen ausgewertet. Die JACOBI-Matrizen an den Stellen z_n und $\tilde{z}_{n+1}$ werden für die Berechnung der k Mikro-Schritte innerhalb des Makro-Schrittes mit der Schrittweite $h = \frac{H}{k}$ linear interpoliert.

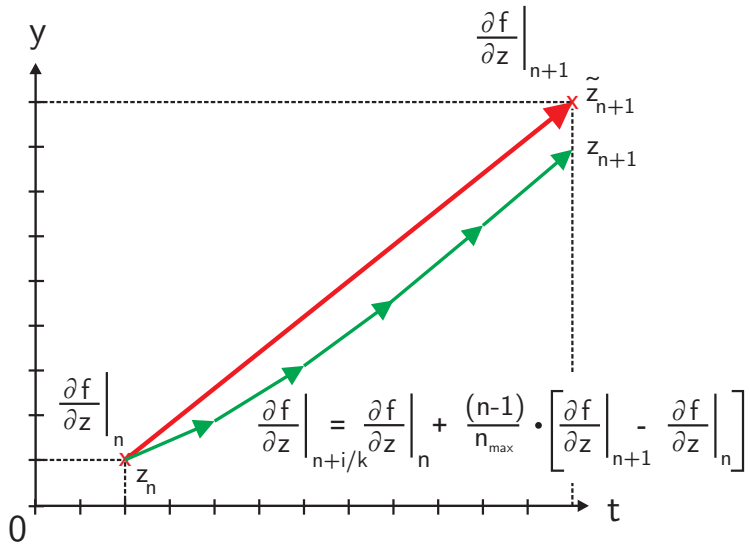


Abbildung 5.3: Makro/Mikro-Time-Stepping Verfahren: Schematische Darstellung.

Diese Herangehensweise wird nur für kontinuierliche Integrationsphasen des System verwendet. Treten Stöße auf, so wird der Schritt verworfen und zur Auflösung des Stoßes das herkömmliche Theta-Time-Stepping Verfahren genutzt. Die Anzahl der Mikro-Schritte pro Makro-Schritt wird dabei heuristisch gewählt.

Ordnungs- und Fehleranalyse

Zur Ordnungs- und Fehlerabschätzung des Verfahrens wird ein Vergleich der Taylorentwicklung des Verfahrens mit der Taylorentwicklung der exakten Lösung (Index $_a$) herangezogen [76, 77]. Die ausführliche Berechnung würde den Rahmen dieses Abschnitts überschreiten, daher wird nur das Ergebnis interpretiert und validiert. Der Vergleich der beiden Entwicklungen gibt Aufschluss darüber, ob die Ordnung des Verfahrens trotz der Approximation der JACOBI-Matrizen erhalten bleibt und mit welchen Koeffizienten der Integrationsfehler ansteigt.

Um das Potential der Interpolation der JACOBI-Matrizen zu zeigen, wird sowohl die Fehleranalyse des Verfahrens mit konstanter JACOBI-Matrix als auch mit linear interpolierter JACOBI-Matrix gezeigt. Es wird die Annahme getroffen, dass keine mengenwertigen Nebenbedingungen aktiv sind, sodass das Basisverfahren mit exakten JACOBI-Matrizen und $\Theta = 0.5$ die Integrationsordnung zwei besitzt.

Für die Fehleranalyse sei

$$y' = f(y) \tag{5.18}$$

eine gewöhnliche Differentialgleichung in y. Die Anzahl Mikro-Schritte pro Makro-Schritt wird mit k bezeichnet.

Gleichung (5.19) gibt den Fehler des Verfahrens mit konstant gehaltenen JACOBI-Matrizen an und Gleichung (5.20) den Fehler des Verfahrens mit linear interpolierten

JACOBI-Matrizen.

$$y_{n+1} - y_{n+1,a} = -h^3 \left(\frac{k}{6} \frac{3k-1}{2} f'' f^2 + \frac{k}{6} f f''^2 \right) + O(h^4) \tag{5.19}$$

$$y_{n+1} - y_{n+1,a} = -h^3 \left(\frac{k}{6} f'' f^2 - \frac{k}{6} f f''^2 \right) + O(h^4) \tag{5.20}$$

Wie aus der Theorie der W-Verfahren [77] zu erwarten war, bleibt die Ordnung der Verfahren trotz Approximation der JACOBI-Matrizen erhalten. Jedoch wächst der Fehler 3. Ordnung bei konstant gehaltener Approximation der JACOBI-Matrizen quadratisch in der Anzahl der Mikro-Schritte k an, wobei der Fehler 3. Ordnung bei linear interpolierten JACOBI-Matrizen nur linear in der Anzahl der Mikro-Schritte k anwächst.
Die Fehlerkoeffizienten 3. Ordnung sind in Abbildung 5.4 für beide Verfahren gezeigt.

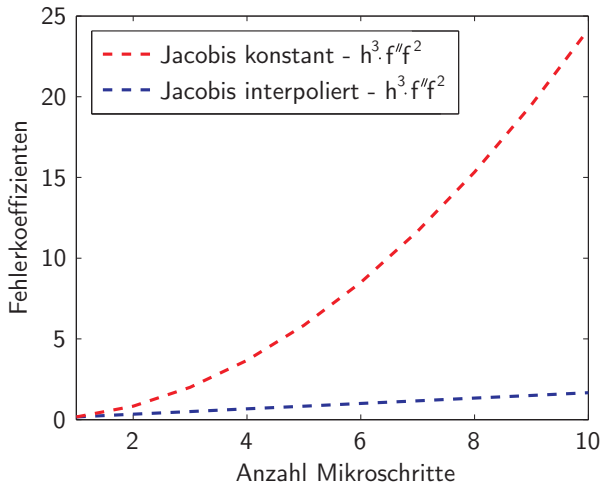


Abbildung 5.4: Fehlerkoeffizienten 3. Ordnung.

Zusätzlich zu der theoretischen Bestimmung der Ordnung und des Fehlerwachstums werden im Folgenden beide Charakteristiken numerisch bestätigt. Dazu wird die Simulation eines Einfachpendels herangezogen, dessen Lösung analytisch hergeleitet werden kann. Auf die Herleitung und Darstellung der analytischen Lösung wird an dieser Stelle verzichtet, da sie in allen gängigen Mechanik-Werken nachgeschlagen werden kann. Durch den Vergleich der Verfahren mit inexakten JACOBI-Matrizen mit der analytischen Lösung unter Variation der Anzahl der Mikro-Schritte n und der Integrationsschrittweite können Aussagen über die Ordnung des Verfahrens und den Trend der Fehlerentwicklung getroffen werden. Abbildung 5.5 zeigt die numerisch errechnete Integrationsordnung des Verfahrens mit linear interpolierten JACOBI-Matrizen in Abhängigkeit der Anzahl der Mikro-Schritte und von Theta.

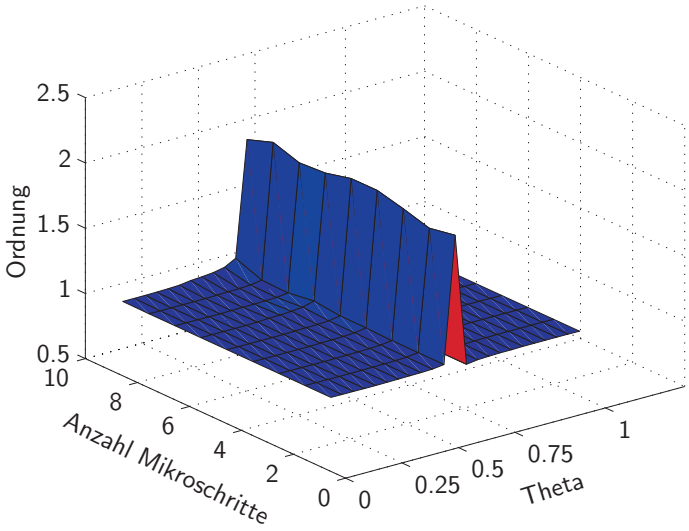


Abbildung 5.5: Integrationsordnung in Abhängigkeit der Anzahl an Mikro-Schritten und von Θ.

Die Abbildung zeigt, dass die Ordnung des Ausgangsverfahrens (Ordnung 2 für $\Theta = 0.5$, sonst Ordnung 1) trotz der Approximation der JACOBI-Matrizen erhalten bleibt. Analog dazu hat die Anzahl der Mikroschritte ebenfalls keinen Einfluss auf die Ordnung des Verfahrens. Abbildung 5.6 zeigt den Fehler der beiden Approximationsverfahren in Abhängigkeit der Anzahl an Mikroschritten. Es lässt sich

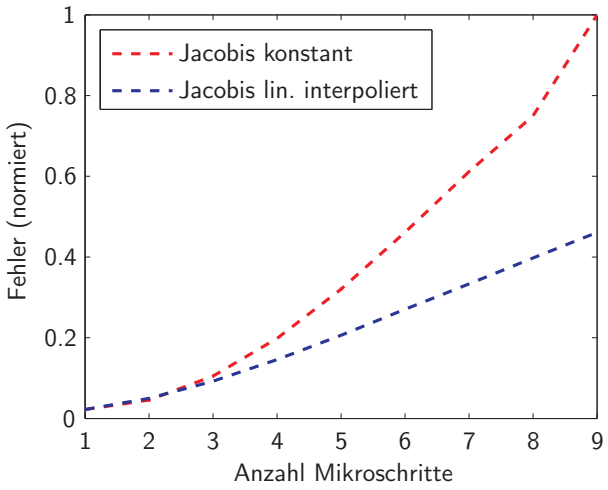


Abbildung 5.6: Fehlertrend der Integrationsverfahren mit approximierten JACOBI-Matrizen ($\Theta = 0.5$).

der prognostizierte Verlauf erkennen. Der Fehler mit linear interpolierten JACOBI-Matrizen steigt linear an und der Fehler mit konstant gehaltenen JACOBI-Matrizen ähnelt einem quadratischen Anstieg. Somit ist das Potential sowohl theoretisch (Taylorentwicklung) als auch numerisch nachgewiesen.

Diskussion

Durch die lineare Interpolation der JACOBI-Matrizen wird erreicht, dass der Fehler lediglich linear in der Anzahl der Mikro-Schritte ansteigt. Im Gegensatz hierzu würde der Fehler mit konstant gehaltenen JACOBI-Matrizen quadratisch in der Anzahl der Mikro-Schritte ansteigen. Somit ist es möglich die Häufigkeit der Neuauswertung der JACOBI-Matrizen im Vergleich zu Verfahren mit konstant gehaltenen JACOBI-Matrizen zu reduzieren und somit die Rechenzeit des Verfahrens zu senken.

5.4.2 Semi-voll-implizites Time-Stepping Integrationsschema

Das in diesem Abschnitt vorgestellte Time-Stepping Verfahren basiert auf dem in Abschnitt 5.2.4 vorgestellten Verfahren. Analog zu dem Makro/Mikro-Verfahren ist auch bei diesem Verfahren das Ziel, die Auswertehäufigkeit der JACOBI-Matrizen zu reduzieren. Im Vergleich zum Makro/Mikro-Verfahren wird hier jedoch ein gänzlich verschiedener Ansatz verfolgt. Die Auswertung der JACOBI-Matrizen erfolgt nicht in vorgegebenen Abständen, sondern orientiert sich daran, ob das Verfahren auch mit inexakten JACOBI-Matrizen konvergiert.
Im ersten Abschnitt werden die grundlegende Idee des Verfahrens und die verwendeten Heuristiken vorgestellt. Die Effizienz des Verfahrens wird an einem akademischen und einem industriellen Beispiel gezeigt.

Darstellung des Verfahrens

In jedem Integrationsschritt muss ein iteratives NEWTON-Verfahren zur Lösung der impliziten Gleichung für $\boldsymbol{u}_{l+1}$ gelöst werden (5.15a). Analog zu den Gedanken, die auch im DASSL-Integrator [112] verfolgt werden, werden die JACOBI-Matrizen solange konstant gehalten, bis die NEWTON-Iteration zur Lösung der impliziten Gleichung für $\boldsymbol{u}_{l+1}$ nicht mehr konvergiert.

Da die JACOBI-Matrizen lediglich für die Lösung des NEWTON-Verfahrens genutzt werden, gehen sie nur indirekt in die neue Lösung $\boldsymbol{u}_{l+1}$ ein. Je weiter die verwendeten JACOBI-Matrizen von den exakten JACOBI-Matrizen abweichen, umso mehr NEWTON-Iterationen werden benötigt, um die implizite Gleichung für $\boldsymbol{u}_{l+1}$ zu lösen. Erst wenn sich keine Konvergenz mehr einstellt, werden die JACOBI-Matrizen neu berechnet. Die Heuristik für die Neuberechnung der JACOBI-Matrizen ist in Abbildung 5.7 dargestellt.

Zu Beginn ① der NEWTON-Iteration werden der Durchlaufszähler auf 1 und der Iterationszähler auf 0 gesetzt. Nachdem die NEWTON-Iteration ausgeführt worden ist ③, wird getestet, ob die gesetzte Toleranz für das NEWTON-Verfahren eingehalten wird ④

$$\texttt{Abweichung} < \texttt{Toleranz}\,.$$

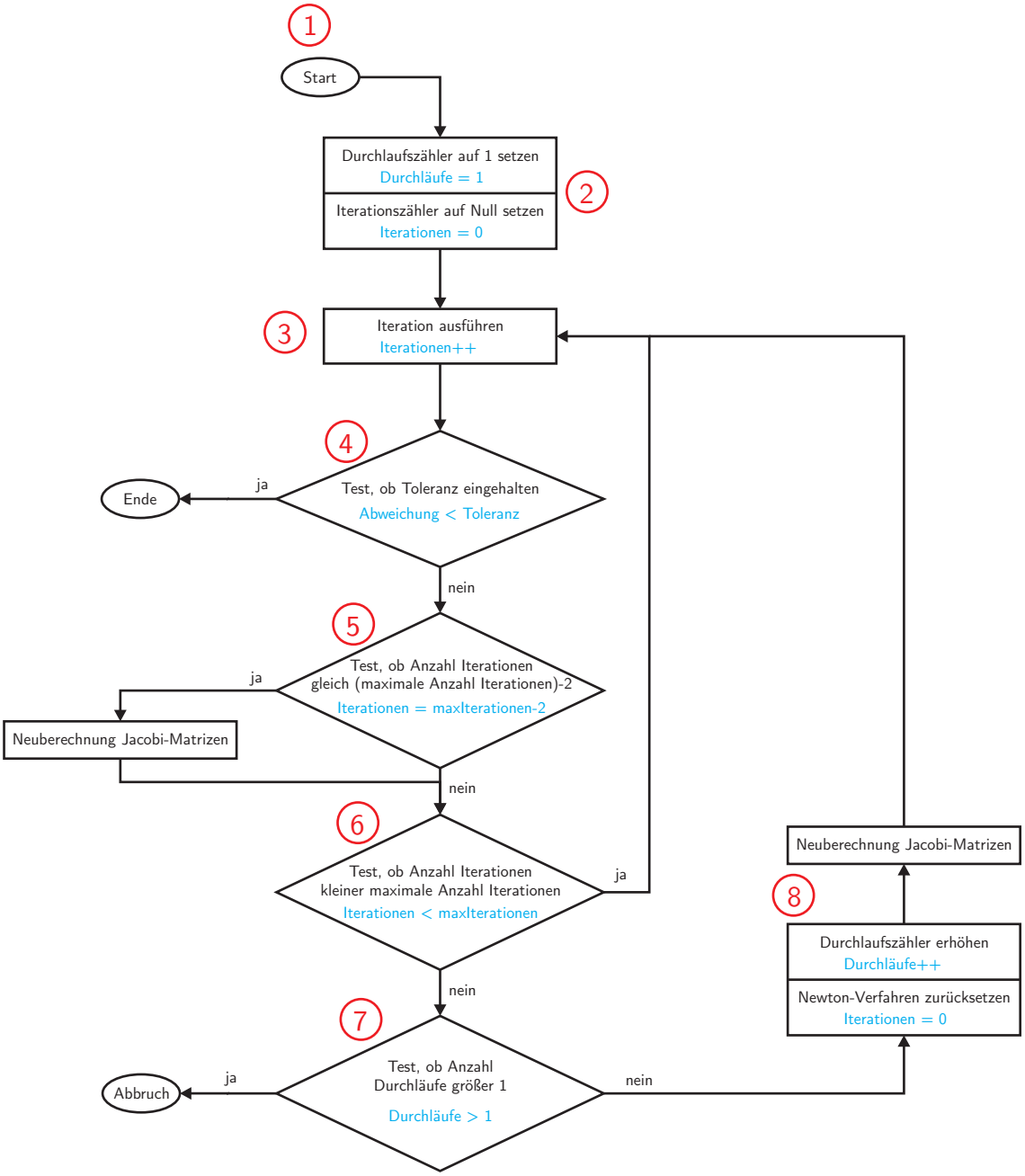


Abbildung 5.7: Heuristik für die Neuberechnung der JACOBI-Matrizen.

Falls die Toleranz eingehalten werden konnte, wird das NEWTON-Verfahren beendet. Ansonsten wird im nächsten Schritt ⑤ geprüft, ob die Anzahl der Iterationen bereits der maximalen Anzahl an Iterationen minus zwei

```
Iterationen = maxIterationen-2
```

entspricht. Falls dies zutrifft, werden die JACOBI-Matrizen neu berechnet bevor zu Schritt ⑥ übergegangen wird. Dadurch soll erreicht werden, dass der aktuelle Durchlauf des NEWTON-Verfahrens noch konvergiert. In Schritt ⑥ wird geprüft, ob bereits

die maximale Anzahl an NEWTON-Iterationen durchgeführt worden ist

$$\texttt{Iterationen} < \texttt{maxIterationen}\,.$$

Falls die maximale Anzahl noch nicht erreicht ist, wird wieder Schritt ③ ausgeführt. Andernfalls wird geprüft, ob das NEWTON-Verfahren bereits zum zweiten Mal fehlgeschlagen ist

$$\texttt{Durchläufe} > 1\,.$$

Falls dies der Fall ist, wird das NEWTON-Verfahren mit einem Fehler abgebrochen. Falls das NEWTON-Verfahren erst einmal durchgelaufen ist, wird der Durchlaufszähler erhöht, das gesamte Verfahren zurückgesetzt und die JACOBI-Matrizen neu berechnet ⑧. Dadurch wird versucht, dass das Verfahren konvergiert, indem direkt zu Beginn mit exakten JACOBI-Matrizen gerechnet wird.

Falls auch der zweite Durchlauf des NEWTON-Verfahrens nicht konvergiert, so wird ein dritter Durchlauf gestartet, in dem in jeder einzelnen NEWTON-Iteration eine Neuberechnung der JACOBI-Matrizen durchgeführt wird. Dieser Vorgang ist der Übersichtlichkeit halber nicht in Abbildung 5.7 dargestellt. Sobald ein Stoß im System auftritt, wird die Anzahl der Iterationen auf eins reduziert.

Anwendung und Analyse

In diesem Abschnitt wird das Verfahren auf das akademische Beispiel *Perlenkette* (siehe Abschnitt 6.1.3) und auf das industrielle Beispiel *Rotordynamik* (siehe Abschnitt 6.2.3) angewendet.

Eine Analyse der Simulation des Rotorsystems ist in Abbildung 5.8 gegeben. In diesem Beispiel wird der Rotor während der ersten Sekunde auf eine Geschwindigkeit von $2067\,{}^{\mathrm{deg}}/_{\mathrm{s}}$ hochgefahren. Ab diesem Zeitpunkt wird die Winkelgeschwindigkeit konstant gehalten.
Die erste Abbildung zeigt die Winkelposition des Rotors und die zweite die Winkelgeschwindigkeit. In der dritten Abbildung wird die Anzahl der NEWTON-Iterationen für die Lösung der impliziten Gleichung für $\boldsymbol{u}_{l+1}$ gezeigt. Die vierte Abbildung gibt die Zeitpunkte an, zu denen die JACOBI-Matrizen neu berechnet werden müssen. Die letzte Abbildung zeigt den Winkel, der zwischen den Neuberechnungen der JACOBI-Matrizen verstreicht.
An diesem stoßfreien Beispiel ist zu erkennen, dass die Anzahl der Iterationen für das NEWTON-Verfahren solange ansteigt, bis die JACOBI-Matrizen neu berechnet werden. Die Anzahl an Iterationen steigt dabei von etwa zwei bis auf etwa 13 an. Sobald die Winkelgeschwindigkeit konstant gehalten wird, müssen die JACOBI-Matrizen etwa alle $80\,^{\circ}$ neu berechnet werden.

Ein gänzlich anderes Verhalten zeigt das Beispiel *Perlenkette*, das als Voruntersuchung zur Simulation von CVT-Getrieben gesehen werden kann (siehe Abbildung 5.9).

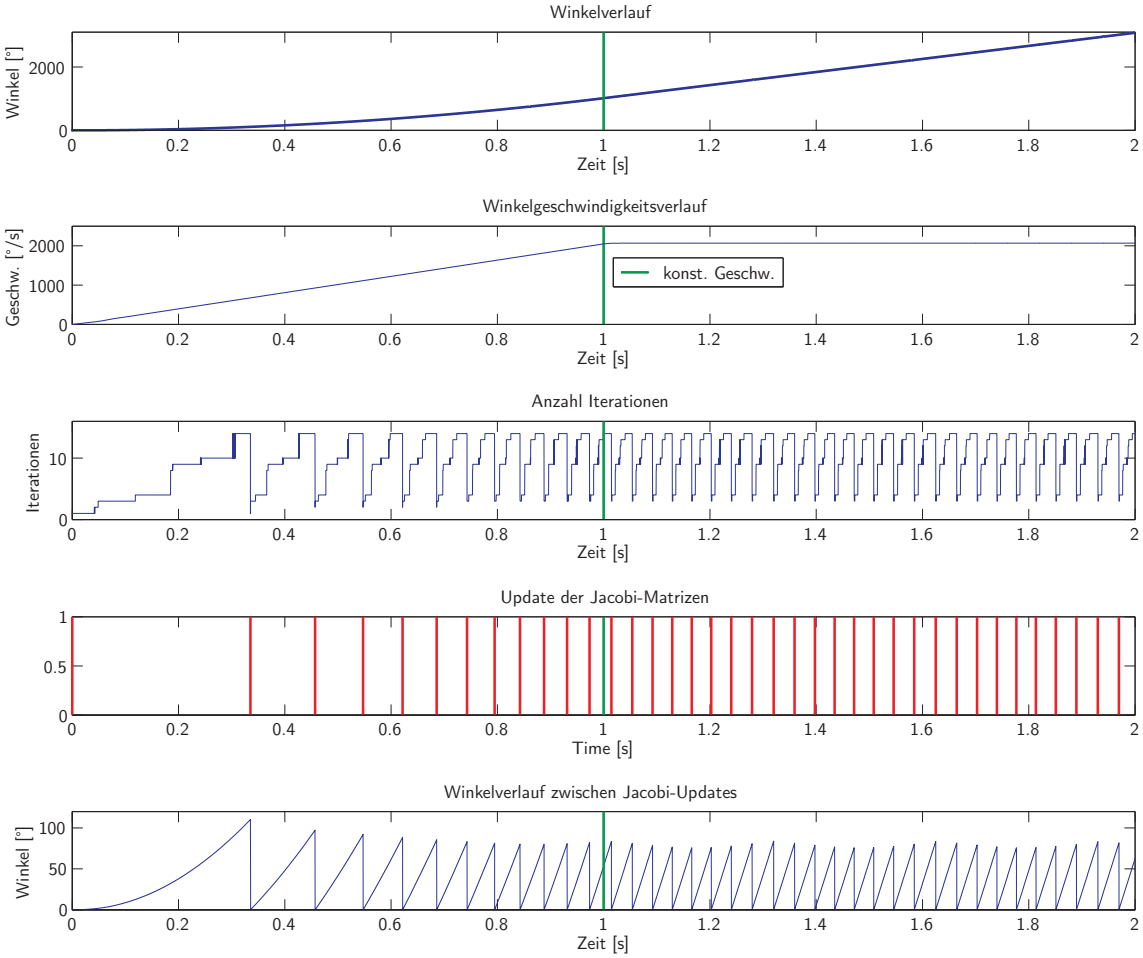


Abbildung 5.8: Analyse des Beispiels *Rotordynamik*.

Die erste Abbildung zeigt die Anzahl der NEWTON-Iterationen für die Lösung der impliziten Gleichung für $\boldsymbol{u}_{l+1}$, die zweite Abbildung gibt die Stoßzeitpunkte im System an. Die dritte Abbildung zeigt die Zeitpunkte, an denen die JACOBI-Matrizen neu berechnet werden. Die letzte Abbildung gibt die Anzahl an Iterationen an, die notwendig sind, um die mengenwertigen Nebenbedingungen zu lösen. Markant an diesem Beispiel ist, dass die JACOBI-Matrizen nur einmalig zu Beginn der Simulation neu berechnet werden müssen. Die Anzahl der Iterationen des NEWTON-Verfahrens bleibt relativ konstant bei drei und wird nur auf eins abgesenkt, falls ein Stoß im System auftritt.

Diskussion

Die Rechenzeitreduktion durch das semi-voll-implizite Time-Stepping Verfahren ist in Abbildung 5.10 gezeigt. Dabei wird das semi-voll-implizite Verfahren einem herkömmlichen halb-expliziten Verfahren gegenübergestellt. Die Zeitschrittweiten werden jeweils knapp unter der Stabilitätsgrenze gewählt, um eine Vergleichbarkeit zu

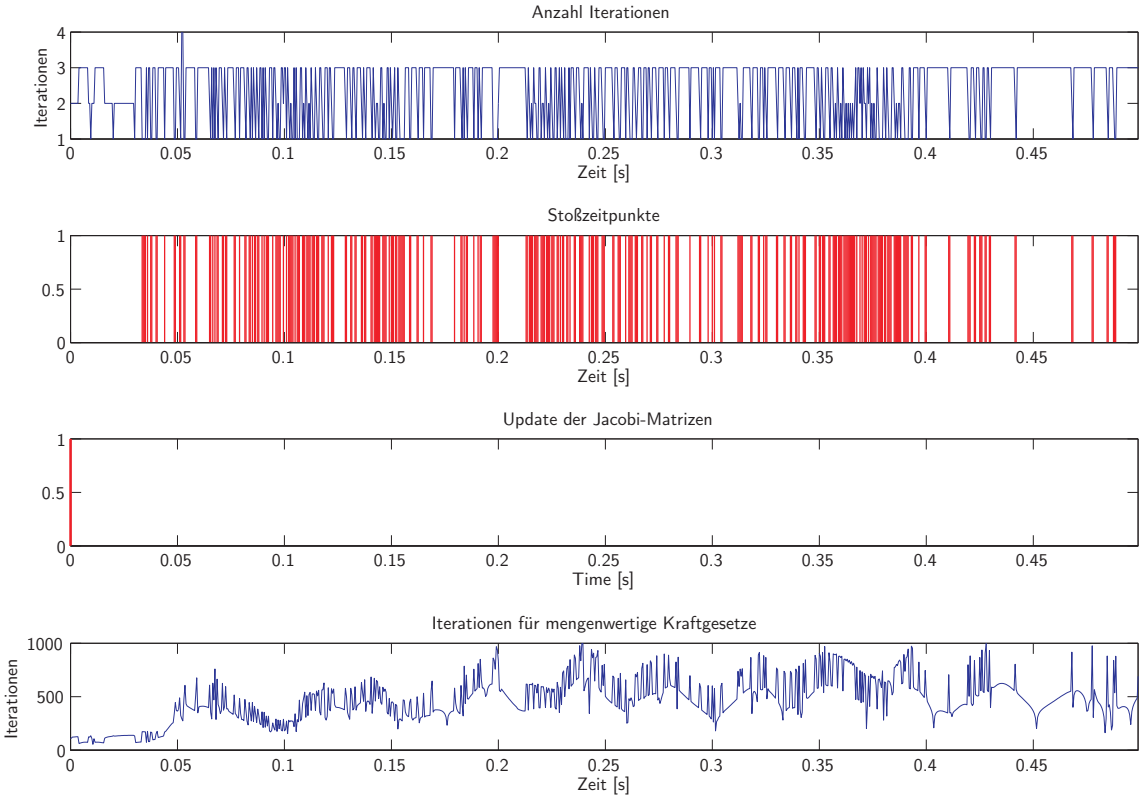


Abbildung 5.9: Analyse des Beispiels *Perlenkette*.

erzielen. Das halb-explizite Verfahren benötigt in beiden Beispielen eine Zeitschrittweite von $\Delta t = 1 \cdot 10^{-6}\,\mathrm{s}$, wobei das semi-voll-implizite Verfahren lediglich eine Zeitschrittweite von $\Delta t = 1 \cdot 10^{-3}\,\mathrm{s}$ benötigt.

Zu sehen ist, dass die Rechenzeiten um 96 % bzw. 97 % reduziert werden können. Dies ermöglicht es vor allem die Rotorsimulation für numerische Optimierungen und für Monitoring-Zwecke einzusetzen (siehe Abschnitt 6.2.3). Ein weiterer positiver Aspekt des Verfahrens ist, dass die Genauigkeit des Verfahrens nicht unter der Approximation der JACOBI-Matrizen leidet. Die JACOBI-Matrizen werden lediglich als Gradienten für das NEWTON-Verfahren benötigt. Aus diesem Grund haben die auftretenden Fehler lediglich einen Einfluss auf die Konvergenzgeschwindigkeit, nicht aber auf die Lösung. Aus beiden Beispielen geht jedoch hervor, dass das Verfahren auf die *Gutmütigkeit* des Systems angewiesen ist. Je nach dem, wie stark sich die JACOBI-Matrizen während der Integration ändern, wird für das NEWTON-Verfahren bei machen Systemen nur sehr selten eine Neuberechnung der JACOBI-Matrizen benötigt, bei manchen Systemen häufiger.

5.5 Schrittweitenadaption und Ordnungserhöhung

Klassische Time-Stepping Verfahren weisen den Nachteil auf, dass sie über keine Schrittweitenadaption und keine Ordnungserhöhung verfügen. In diesem Abschnitt

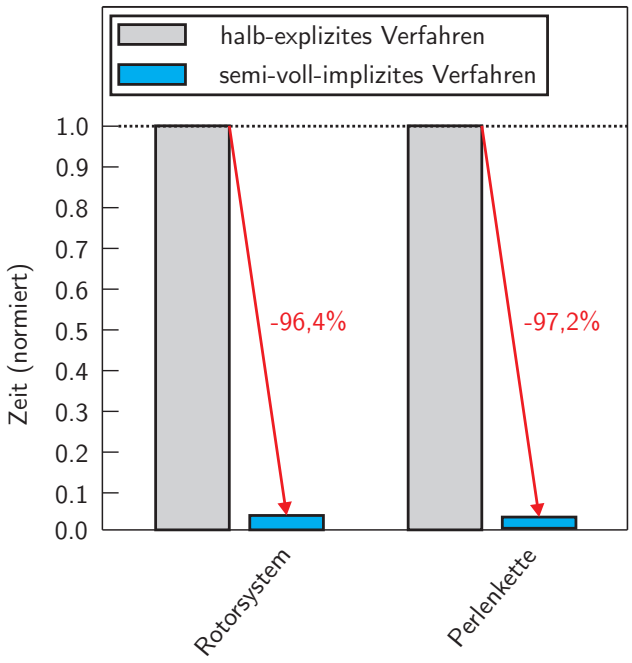


Abbildung 5.10: Normierte Rechenzeiten der Beispiele.

werden Möglichkeiten gezeigt, wie die klassischen Time-Stepping Verfahren um die genannten Aspekte erweitert werden können. Abschnitt 5.5.1 stellt eine Möglichkeit dar, wie die Integrationsschrittweite in glatten und nicht-glatten Bereichen adaptiert werden kann. Abschnitt 5.5.2 geht auf Extrapolationsverfahren zur Ordnungserhöhung ein.

Sowohl die Heuristiken für die Schrittweitenadaption als auch die Anwendung von Extrapolationsverfahren zur Ordnungserhöhung von Time-Stepping Integrationsverfahren gehen auf Huber und Ulbrich [85,86] zurück. Deshalb wird nur ein Überblick über die Verfahren gegeben, für weitere Informationen wird auf die genannten Arbeiten verwiesen. Extrapolationsverfahren zur Ordnungserhöhung werden auch von Studer [139] verwendet.

5.5.1 Schrittweitenadaption

Betrachtet wird die skalare Differentialgleichung

$$\dot{z} = f(z,t) \tag{5.21}$$

mit den Anfangswerten

$$z(t_0) = z_0 \,.$$

Die Verfahren zur Schrittweitensteuerung von ODE-Integratoren können in zwei Gruppen eingeteilt werden. In Richardson-Verfahren und eingebettete Verfahren.

RICHARDSON-Verfahren führen den gleichen Integrationsschritt mit unterschiedlichen Zeitschrittweiten Δt_i aus und errechnen durch einen Vergleich der Lösungen in Abhängigkeit gegebener Toleranzen die optimale Schrittweite.
Sogenannte eingebettete Verfahren führen den gleichen Integrationsschritt mit Verfahren unterschiedlicher Ordnung aus, woraus in Abhängigkeit gegebener Toleranzen die optimale Schrittweite berechnet werden kann. Dabei wird versucht die Anzahl der Auswertungen der Differentialgleichung möglichst gering zu halten, indem in die Berechnung der Lösung der zweiten Ordnung möglichst viele Informationen aus der Berechnung der Lösung der ersten Ordnung eingehen.

Die impliziten Time-Stepping Verfahren, die in dieser Arbeit vorgeschlagen werden, verwenden den Ansatz nach RICHARDSON.

In glatten Bereichen

In glatten Bereichen wird das RICHARDSON-Verfahren [121] zur Bestimmung einer optimalen Schrittweite herangezogen. Dazu wird der gleiche Integrationsschritt mit den Schrittweiten Δt und $\Delta t/2$ durchgeführt. Die zugehörigen Ergebnisse werden mit $z^{\Delta t}(t_0 + \Delta t)$ und $z^{\Delta t/2}(t_0 + \Delta t)$ bezeichnet. Die Ordnung des Integrationsverfahrens wird mit p bezeichnet.

Mit der Definition der Integrationstoleranz

$$\mathrm{Tol} = \mathrm{aTol} + |z| \cdot \mathrm{rTol}$$

kann die optimale Schrittweite nach Gleichung (5.22)

$$\Delta t_{\mathrm{opt}} = \Delta t \cdot \sqrt[p+1]{\frac{\mathrm{Tol}}{||z^{\Delta t/2}(t_0 + \Delta t) - z^{\Delta t}(t_0 + \Delta t)||}\,(2^p - 1)} \tag{5.22}$$

berechnet werden. aTol bezeichnet die absolute Toleranz und rTol die relative Toleranz. Es hat sich als sinnvoll erwiesen die Schrittweite während der Integration nicht zu schnell zu verändern und zusätzlich die berechnete optimale Schrittweite Δt_{opt} mit einem Sicherheitsfaktor kleiner eins zu multiplizieren.

In nicht-glatten Bereichen

Die Methoden zur Schrittweitensteuerung von ODE-Integratoren sind während Stößen aufgrund der Diskontinuitäten in den Geschwindigkeiten nicht anwendbar. Daher schlagen HUBER UND ULBRICH [85, 86] drei Methoden vor, um die Schrittweite in nicht-glatten Bereichen anzupassen.
Die erste Methode (*Gap Control*) zielt darauf ab, die Diskontinuitäten mittels Extrapolation vorauszusagen und darauf basierend die Schrittweite mit verschiedenen Heuristiken zu adaptieren. Die zweite und dritte Methode verwenden den RICHARDSON-Fehlerschätzer aus der ODE-Theorie, fügen jedoch zwei Änderungen hinzu. Die

zweite Methode sieht vor, die Geschwindigkeiten aus der Fehlerschätzung herauszunehmen, sodass der RICHARDSON-Fehlerschätzer wieder seine Gültigkeit erlangt. Die dritte Methode bedient sich Methoden aus der DAE-Theorie [77], die vorsieht, dass die Geschwindigkeiten mit der Zeitschrittweite Δt multipliziert werden, bevor sie in die Fehlerschätzung eingehen.

5.5.2 Ordnungserhöhung durch Extrapolation

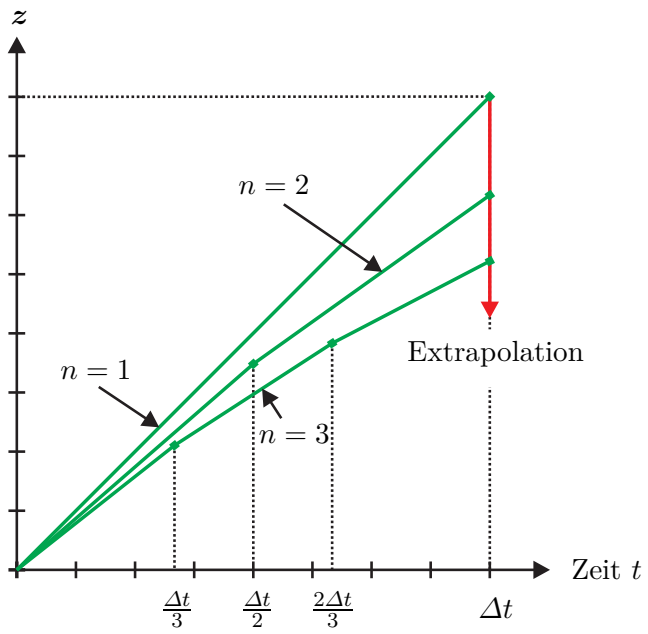


Abbildung 5.11: Extrapolationsverfahren: Schematische Darstellung [85].

Die in dieser Arbeit vorgeschlagenen impliziten Time-Stepping Verfahren besitzen eine Integrationsordnung von eins. Zur Ordnungserhöhung werden analog zu dem Vorgehen in [82] Extrapolationsmethoden eingesetzt, jedoch wird im Gegensatz zu den Verfahren in [82] als Basisverfahren kein halb-explizites Time-Stepping Verfahren verwendet, sondern Verfahren mit impliziten Anteilen.

Die grundlegende Idee von Extrapolationsverfahren [44, 76] besteht darin, den gleichen Zeitschritt mit unterschiedlichen Zeitschrittweiten Δt zu berechnen. Dazu werden k Approximationen

$$z^{\Delta t_i}(t_0 + \Delta t)$$

mit den Zeitschrittweiten

$$\Delta t_i = \frac{\Delta t}{n_i} \quad \text{mit} \quad n_1 = 1 < n_2 < \ldots < n_k (n_i \in \mathcal{Z})$$

berechnet.

Durch den Grenzübergang $\Delta t \to 0$ wird versucht aus den k Stützstellen eine Lösung $z(t_0 + \Delta t)$ höherer Ordnung zu berechnen. Diese Vorgehensweise ermöglicht es, mit einem Basisverfahren der Ordnung p und k Stützstellen eine Integrationsordnung von $p + k - 1$ zu erzielen. Die Idee von Extrapolationsverfahren ist in Abbildung 5.11 schematisch dargestellt.
Wird beispielsweise ein Basisintegrationsverfahren der Ordnung $p = 1$ verwendet und der Integrationsschritt mit $k = 2$ Approximationsstellen (Δt und $\frac{\Delta t}{2}$) unterteilt, so kann auf eine Integrationsordnung von $p + k - 1 = 2$ extrapoliert werden.

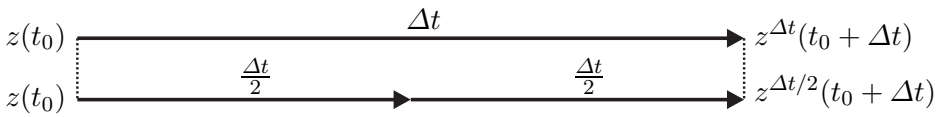


Abbildung 5.12: Extrapolationsverfahren: k=2.

Unter Verwendung der ersten Approximation $z^{\Delta t}(t_0 + \Delta t)$, die durch einen Integrationsschritt mit der Zeitschrittweite Δt berechnet wurde und der zweiten Approximation $z^{\Delta t/2}(t_0 + \Delta t)$, die durch zwei sequentielle Schritte mit jeweils der Zeitschrittweite $\frac{\Delta t}{2}$ berechnet wurde, kann die Lösung zweiter Ordnung $z(t_0 + \Delta t)$ nach Gleichung (5.23)

$$z(t_0 + \Delta t) = 2 \cdot z^{\Delta t/2}(t_0 + \Delta t) - z^{\Delta t}(t_0 + \Delta t) \tag{5.23}$$

berechnet werden. Die Extrapolationsvorschriften für allgemeine Verfahren der Ordnung p, die mit k Stützstellen approximiert werden, sind in dem Buch [77] von Hairer zu finden.

Huber [82] extrapoliert in seiner Arbeit bis zu einer Ordnung von vier. Für die Schrittweitensteuerung nutzt er sowohl das Richardson-Verfahren als auch eingebettete Verfahren. In dieser Arbeit wird die Extrapolation bis auf eine Ordnung von drei eingesetzt. Die Schrittweitensteuerung basiert auf dem Richardson-Fehlerschätzer, wobei in nicht-glatten Bereichen die Geschwindigkeiten mit der Zeitschrittweite Δt multipliziert werden. Während Stößen wird die Ordnung auf eins reduziert, da die Theorie der Extrapolation keine Diskontinuitäten in den Zuständen erlaubt.

5.6 Parallelisierung

In diesem Abschnitt werden drei Methoden gezeigt, wie die Parallelisierung auf der Integratorebene effizient eingesetzt werden kann. Da es sich bei der Integratorebene laut der Einteilung von Eichberger [49] um die grobstrukturierte Ebene handelt, lassen sich hohe Beschleunigungen und gute Effizienzwerte erzielen.
Im ersten Abschnitt wird auf die Parallelisierung der Zwischenschritte der Extrapolationsmethoden und der Speicherung der Simulationsergebnisse eingegangen. Im zweiten Abschnitt wird die Parallelisierung der Berechnung der Jacobi-Matrizen gezeigt.

5.6.1 Extrapolation und Datenspeicherung

Extrapolationsverfahren eignen sich hervorragend zur parallelen Berechnung. Diese Eigenschaft wurde bereits in den Arbeiten [85,86] umgesetzt. Analoge Überlegungen gehen auch in die Parallelisierungsmethoden ein, die in dieser Arbeit vorgestellt werden.

Neben der Parallelisierung der Zwischenschritte der Extrapolation lässt sich auch die Speicherung der Simulationsergebnisse parallelisieren. Der grundlegende Gedanke ist dabei, dass der vorherige Zeitschritt gespeichert wird, während der nächste Zeitschritt bereits neu berechnet wird. Die Beschleunigung durch diese Herangehensweise hängt davon ab, wie oft Daten gespeichert werden. Unter dem Begriff *Datenspeicherung* werden in diesem Bezug sowohl die notwendigen Postprocessing-Schritte als auch das Speichern der Daten auf einem Speichermedium verstanden.
Für Systeme mit hoher Dynamik ist es nicht unüblich die Zeitschrittweite gleich der Plotschrittweite[3] zu wählen, wodurch hohe Beschleunigungen ermöglicht werden. Für Verfahren mit variabler Zeitschrittweite, aber konstanter Plotschrittweite, müssen die Ergebnisse interpoliert werden. Auch dies erhöht den Aufwand der Datenspeicherung und bietet dadurch die Möglichkeit hoher Beschleunigungen.

Sowohl bei der Parallelisierung der Zwischenschritte der Extrapolation als auch bei der Parallelisierung der Datenspeicherung ist darauf zu achten, dass sowohl *Race Conditions* als auch *Deadlocks* verhindert werden (siehe Abschnitt 3.3.1).
Die Implementierungen in dieser Arbeit lösen die Problematik dadurch, dass sowohl für die parallel laufenden Zwischenschritte der Extrapolation als auch für die parallel laufende Datenspeicherung eine eigenständige Instanz des Mehrkörpersystems verwendet wird. Anschaulich gesprochen werden zu Beginn mehrere gleiche Instanzen des Mehrkörpersystems (aus der Sicht der Software) angelegt, mit denen unabhängig voneinander parallel gerechnet werden kann. Dies hat den entscheidenden Vorteil, dass *Race Conditions* oder *Deadlocks* nur bedingt auftreten können. Andererseits wird mehr Speicher auf den Simulationsrechnern benötigt. In heutigen Desktoprechnern ist in der Regel bereits so viel Speicher verbaut, dass dieses Vorgehen auch bei Simulationen von Systemen mit einer hohen Anzahl an Freiheitsgraden (z.B. CVT) kein Problem darstellt.

In dieser Arbeit wird die Extrapolation bis zu einer Ordnung von $p = 3$ genutzt, wobei die Verfahren sowohl mit als auch ohne Schrittweitensteuerung genutzt werden. Für das Konzept der Extrapolation wird auf die Arbeit [82] verwiesen. Die Ergänzung der Ideen aus dieser Arbeit um die Methode der parallelen Datenspeicherung wird an den nächsten drei Beispielen gezeigt.

Das Schema 5.13 zeigt die Strategie für ein Verfahren ohne Extrapolation und konstanter Zeitschrittweite. Während der nächste Zeitschritt $\boldsymbol{z}_l \rightarrow \boldsymbol{z}_{l+1}^{\Delta t}$ auf einem Rechenkern (`Thread 1`) berechnet wird, werden die Simulationsergebnisse des vorherigen Zeitschritts parallel gespeichert. Die Rechenleistung stellt dabei ein weiterer

3 Unter Plotschrittweite versteht man analog zu der Integrationsschrittweite die Schrittweite, mit der Daten gespeichert werden.

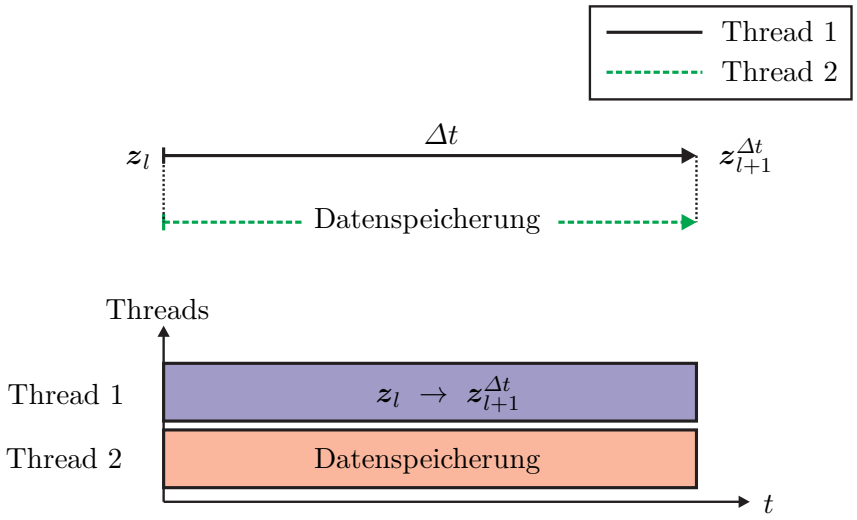


Abbildung 5.13: Keine Schrittweitensteuerung, Ordnung 1: Parallelisierungsstrategie.

Rechenkern (`Thread 2`) zur Verfügung.

Abbildung 5.14 zeigt die Strategie für ein Verfahren der Ordnung $p = 2$ ohne Schrittweitensteuerung. Wieder wird die Rechenlast auf zwei Rechenkerne (`Thread 1` und `Thread 2`) verteilt. Das analoge Schema kann auch für ein Verfahren der Ordnung $p = 1$ mit Schrittweitensteuerung verwendet werden.

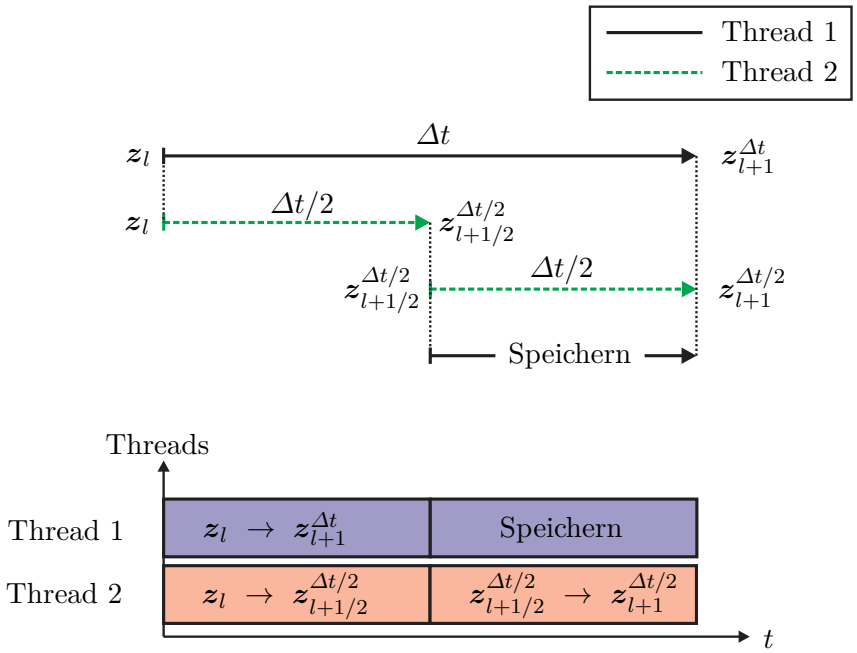


Abbildung 5.14: Keine Schrittweitensteuerung, Ordnung 2: Parallelisierungsstrategie.

Der Vollständigkeit halber ist in Abbildung 5.15 die Strategie für ein Verfahren der Ordnung $p = 3$ ohne Schrittweitensteuerung angegeben.

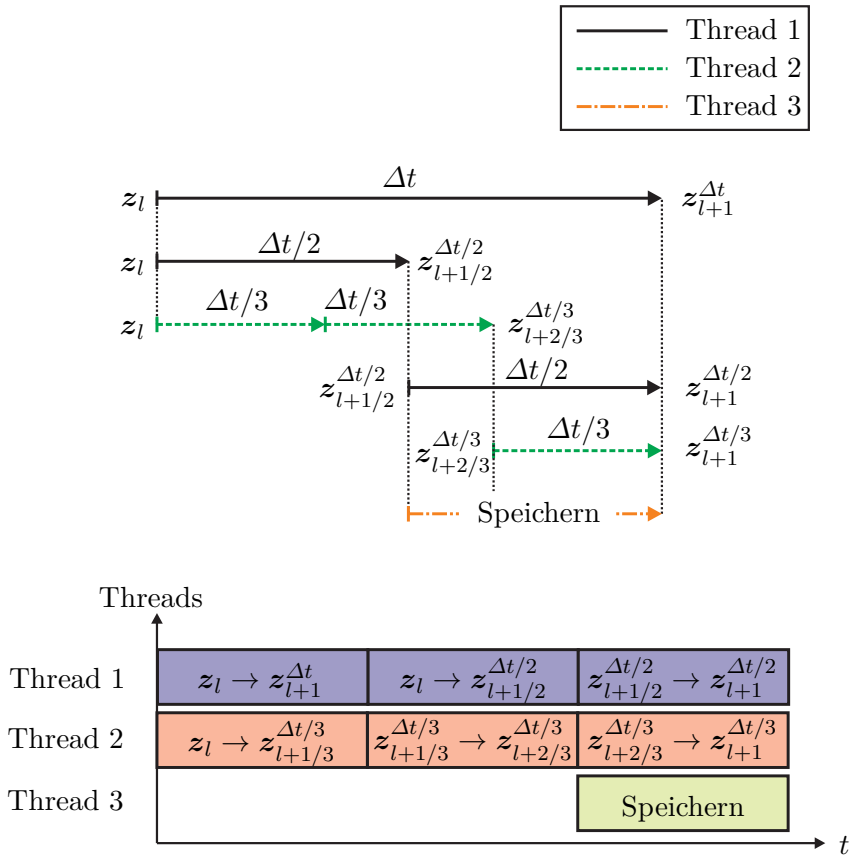


Abbildung 5.15: Keine Schrittweitensteuerung, Ordnung 3: Parallelisierungsstrategie.

Die Effizienz wird im Folgenden an einem halb-expliziten Verfahren erster Ordnung (siehe Abschnitt 5.2.1) und konstanter Zeitschrittweite (Abbildung 5.13) gezeigt. Als Beispiel dient der Ventiltrieb aus Abschnitt 6.2.1. Die Plotschrittweite wird in diesem Beispiel gleich der doppelten Integrationsschrittweite gesetzt

$$\Delta t_{\text{plot}} = 2\Delta t\,.$$

Das Ergebnis ist in Abbildung 5.16 dargestellt. In diesem Beispiel teilt sich die Rechenzeit pro Integrationsschritt in den Anteil für die Integration (etwa 68 %) und in den Anteil für die Datenspeicherung (etwa 32 %) auf. Die dargestellte Rechenzeit ist normiert auf die sequentielle Rechenzeit des Systems (linke Seite). Nutzt man das vorgeschlagene Konzept für die parallele Datenspeicherung, so ergibt sich die Verteilung auf der rechten Seite. Die absoluten Zeiten für die Integration und für die Datenspeicherung bleiben konstant, jedoch muss ein Overhead addiert werden, der die Beschleunigung reduziert.
Der Overhead setzt sich aus zwei Anteilen zusammen. Zum einen aus dem bekannten Overhead für die Parallelisierung und zum anderen aus dem Overhead für die Übertragung der simulierten Daten aus der ersten Instanz des Systems in eine weitere Instanz, mit derer die Daten parallel zu der ersten Instanz gespeichert werden können. Es ergibt sich in diesem Beispiel eine Reduzierung der Rechenzeit von etwa 21 %.

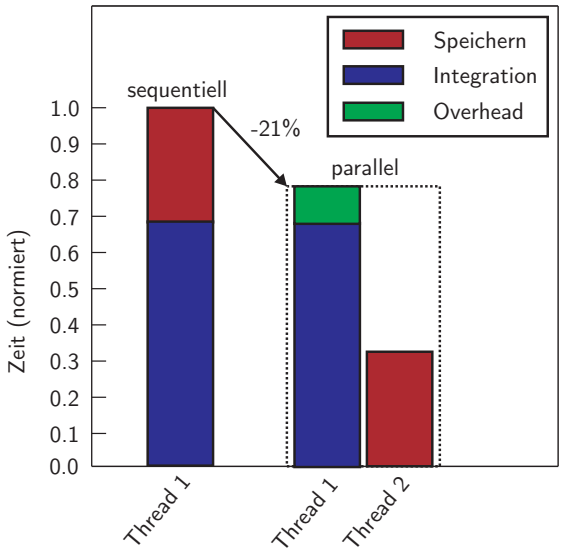


Abbildung 5.16: Parallele Datenspeicherung am Beispiel des Ventiltriebs.

5.6.2 Parallele Berechnung der Jacobi-Matrizen

In diesem Abschnitt wird die Parallelisierung der Berechnung der JACOBI-Matrizen dargestellt. Wie bereits in Abschnitt 5.3 gezeigt, lässt sich die numerische Berechnung der JACOBI-Matrizen sehr gut parallelisieren. Analog zu dem Vorgehen zur parallelen Berechnung der Zwischenschritte der Extrapolation und zur parallelen Datenspeicherung, werden auch zur parallelen Berechnung der JACOBI-Matrizen aus der Sicht der Software mehrere Instanzen des Systems genutzt.

Betrachtet man die partielle Ableitung der rechten Seite $\boldsymbol{h}$ nach den generalisierten Koordinaten $\boldsymbol{q}$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{q}} = \begin{bmatrix} \frac{\partial h_1}{\partial q_1}, & \frac{\partial h_1}{\partial q_2}, & \cdots, & \frac{\partial h_1}{\partial q_n} \\ \frac{\partial h_2}{\partial q_1}, & \frac{\partial h_2}{\partial q_2}, & \cdots, & \frac{\partial h_2}{\partial q_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial h_n}{\partial q_1}, & \frac{\partial h_n}{\partial q_2}, & \cdots, & \frac{\partial h_n}{\partial q_n} \end{bmatrix},$$

so ist ersichtlich, dass jede Spalte der JACOBI-Matrizen parallel berechnet werden kann, da die Spalten untereinander keine Abhängigkeiten aufweisen. Dies ist auch das grundlegende Konzept zur Parallelisierung der numerischen Berechnung der JACOBI-Matrizen.

Je nach Anzahl der Systemzustände $\boldsymbol{z} = \left(\boldsymbol{q},\ \boldsymbol{u},\ t\right)^T$ und der Anzahl an verfügbaren Prozessoren wird eine entsprechende Lastverteilung durchgeführt. Für die Lastverteilung ist zu berücksichtigen, dass die Ableitungen nach den generalisierten Lagen $\boldsymbol{q}$ eine höhere Rechenzeit in Anspruch nehmen als die Ableitungen nach den verallgemeinerten Geschwindigkeiten $\boldsymbol{u}$. Dies ist darin begründet, dass für die

Ableitungen unterschiedliche Neuberechnungen innerhalb des Systems notwendig sind (siehe Tabelle 5.1).

Tabelle 5.1: Notwendige Update-Prozesse für die Berechnung der JACOBI-Matrizen.

Ableitungen nach q: $\left(\frac{\partial h}{\partial q}\right)$	**Ableitungen nach u:** $\left(\frac{\partial h}{\partial u}\right)$
- updateSDV	- updateSDV
- updateg	- updategd
- updategd	- updateh
- updateT	
- updateJacobians	
- updateh	

Die Systemmatrizen $\boldsymbol{T}$ (updateT) und $\boldsymbol{J}$ (updateJac), sowieso die Kontaktabstandsberechnungen (updateg) sind unabhängig von den generalisierten Positionen $\boldsymbol{q}$ und müssen somit bei Variation der verallgemeinerten Geschwindigkeiten $\boldsymbol{u}$ nicht neu berechnet werden. Die Lastverteilung erfolgt wie auch in der systeminternen *semi-dynamischen* Parallelisierungsmethode auf Basis des *Greedy* Algorithmus (siehe Abschnitt 4.3.2), wobei die Rechenzeiten für die Ableitungen nach $\boldsymbol{q}$ bzw. $\boldsymbol{u}$ gemessen werden.

Race Conditions oder *Deadlocks* sind bei dieser Parallelisierungsmethode nur bedingt möglich, da die einzelnen Spalten der JACOBI-Matrizen unabhängig voneinander berechnet werden können. Analog dazu können die parallel berechneten Spalten unabhängig voneinander in die globale JACOBI-Matrix geschrieben werden.

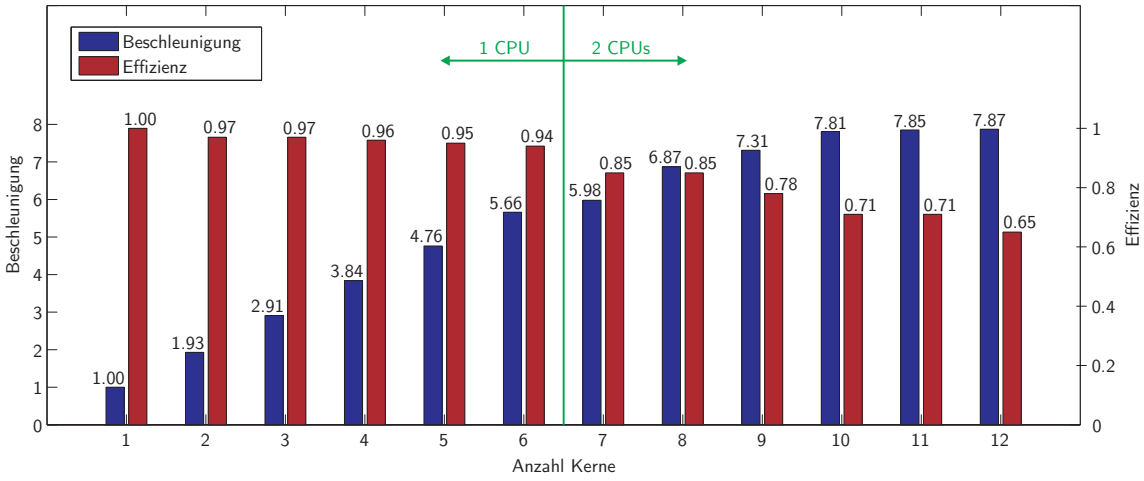


Abbildung 5.17: Parallele Berechnung der JACOBI-Matrizen: Beschleunigung und Effizienz in Abhängigkeit der Anzahl der Kerne.

Abbildung 5.17 zeigt die erzielbaren Beschleunigungen und die zugehörigen Effizienzwerte der Parallelisierung in Abhängigkeit der Anzahl an verwendeten Kernen. Als Beispiel dient der Ventiltrieb aus Abschnitt 6.2.1. Da sich die Parallelisierungsmethode in die *grobstrukturierte* Parallelisierung eingruppieren lässt, zeigen sich erwartungsgemäß sehr hohe Beschleunigungen. Gut zu erkennen ist, dass sich bei dem

Übergang von der Berechnung auf einer CPU (mit sechs Kernen) auf zwei CPUs die Effizienz schlagartig verschlechtert. Dies ist darin begründet, dass die Kommunikation auf einer CPU erheblich schneller ist wie zwischen zwei CPUs, die über das Mainboard des Rechners verbunden sind. Bis zu einer Anzahl von sechs Kernen sind bemerkenswerte Effizienzen von knapp unter 100 % erreichbar. Ab etwa acht Kernen ist die weitere Erhöhung der Rechenkerne in diesem Beispiel nicht mehr lohnenswert. Der minimalen Steigerung der Beschleunigung steht eine stark fallende Effizienz gegenüber.

5.7 Vergleich von Time-Stepping Verfahren

In diesem Abschnitt werden die vorgestellten Time-Stepping Verfahren mit Ordnungserhöhung und Schrittweitenadaption und die Parallelisierungsmethoden auf Integratorebene diskutiert.
Der erste Abschnitt 5.7.1 behandelt einen möglichen Vergleich zwischen expliziten und impliziten Time-Stepping Verfahren. Der zweite Abschnitt 5.7.2 analysiert die Ordnungserhöhung von impliziten Time-Stepping Verfahren durch Extrapolation anhand des Beispiels eines Ventiltriebs.

5.7.1 Vergleich von expliziten und impliziten Verfahren

Für den Vergleich zwischen expliziten und impliziten Time-Stepping Verfahren werden in diesem Abschnitt ein

1. halb-explizites Time-Stepping Verfahren mit Schrittweitensteuerung und Ordnung $p = 1$,
2. linear-implizites Time-Stepping Verfahren mit Schrittweitensteuerung, Ordnung $p = 1$ und exakten Jacobi-Matrizen,
3. linear-implizites Time-Stepping Verfahren mit Schrittweitensteuerung, Ordnung $p = 1$ und inexakten Jacobi-Matrizen,
4. und ein semi-voll-implizites Time-Stepping Verfahren mit Schrittweitensteuerung, Ordnung $p = 1$ und inexakten Jacobi-Matrizen

verglichen, wobei die vorgeschlagenen Parallelisierungsmethoden aus Abschnitt 5.6 auf Integratorebene zum Einsatz kommen.

Ein Vergleich zwischen expliziten und impliziten Verfahren ist im Allgemeinen sehr schwierig, da die Verfahren je ihre eigenen Einsatzzwecke haben. Um die Verfahren besser vergleichen zu können, werden für alle vier Verfahren die gleichen Toleranzen für die Schrittweitensteuerung gesetzt. Dadurch ist dennoch nicht sichergestellt, dass die Verfahren die gleiche Frequenzauflösung besitzen.
Als System kommt das Beispiel *Gleitende Elemente auf flexiblem Balken* (siehe Abschnitt 6.1.4) zum Einsatz. Die Tabellen zeigen je die vier Integrationsverfahren,

wobei in Tabelle 5.2 nur die Berechnung der JACOBI-Matrizen parallelisiert ist, in Tabelle 5.3 ist zusätzlich die Berechnung der Schrittweitensteuerung parallelisiert und in Tabelle 5.4 ist zusätzlich die Datenspeicherung parallelisiert.

Tabelle 5.2: Parallele Berechnung der JACOBI-Matrizen.

Verfahren	JACOBI **Updates**	**durchschnittl. Schrittweite** [s]	**Rechendauer** [s]
halb. exp.	0	$3.35 \cdot 10^{-6}$	728
lin.-imp. (exakt. Jac.)	2241	$2.73 \cdot 10^{-4}$	153
lin.-imp. (inexakt. Jac.)	1454	$2.79 \cdot 10^{-4}$	111
semi-voll-imp. (inexakt. Jac.)	18	$2.79 \cdot 10^{-4}$	27

Tabelle 5.3: Parallele Berechnung der JACOBI-Matrizen und Parallelisierung der Schrittweitensteuerung.

Verfahren	JACOBI **Updates**	**durchschnittl. Schrittweite** [s]	**Rechendauer** [s]
halb. exp.	0	$3.35 \cdot 10^{-6}$	521
lin.-imp. (exakt. Jac.)	2241	$2.73 \cdot 10^{-4}$	100
lin.-imp. (inexakt. Jac.)	1454	$2.79 \cdot 10^{-4}$	69
semi-voll-imp. (inexakt. Jac.)	18	$2.79 \cdot 10^{-4}$	20

Tabelle 5.4: Parallele Berechnung der JACOBI-Matrizen, Parallelisierung der Schrittweitensteuerung und parallele Datenspeicherung.

Verfahren	JACOBI **Updates**	**durchschnittl. Schrittweite** [s]	**Rechendauer** [s]
halb. exp.	0	$3.35 \cdot 10^{-6}$	506
lin.-imp. (exakt. Jac.)	2241	$2.73 \cdot 10^{-4}$	93
lin.-imp. (inexakt. Jac.)	1454	$2.79 \cdot 10^{-4}$	62
semi-voll-imp. (inexakt. Jac.)	18	$2.79 \cdot 10^{-4}$	17

Da es sich bei dem System *Gleitende Elemente auf flexiblem Balken* um ein numerisch steifes System handelt, sind implizite Verfahren im Vorteil. Abbildung 5.18 zeigt die erzielbaren Beschleunigungen, die je nach verwendetem Verfahren und verwendeten Parallelisierungsmethoden erreicht werden können.

Der Übergang von einem halb-expliziten Verfahren auf ein linear-implizites Verfahren mit paralleler Berechnung der JACOBI-Matrizen erzielt einen Rechenzeitvorteil von 79 %.
Weitere Rechenzeitreduktionen können durch die Anwendung der in dieser Arbeit vorgeschlagenen impliziten Time-Stepping Integrationsverfahren erzielt werden. Die Verfahren nutzen jeweils die parallele Berechnung der JACOBI-Matrizen. Die Anwendung des linear-impliziten Verfahrens mit inexakten JACOBI-Matrizen reduziert

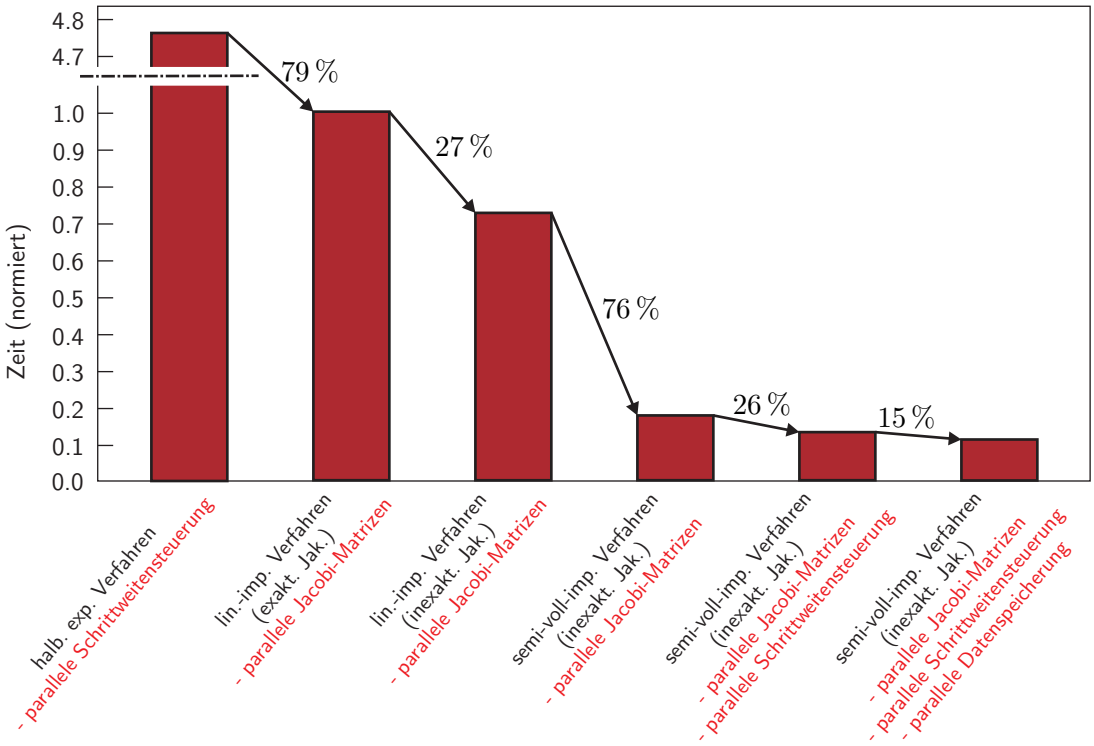


Abbildung 5.18: Erzielbare Beschleunigungen (E-Modul und Dichte von Stahl).

die Rechenzeit um 27 % gegenüber dem gleichen Verfahren mit exakten JACOBI-Matrizen, da die Auswertehäufigkeit der JACOBI-Matrizen um 35 % verringert werden kann. Die Nutzung des semi-voll-impliziten Verfahrens mit inexakten JACOBI-Matrizen erzielt gegenüber dem linear-impliziten Verfahren mit inexakten JACOBI-Matrizen eine Rechenzeitreduktion von 76 %. Diese signifikante Reduktion der Rechenzeit kann bei diesem System dadurch erreicht werden, dass die Anzahl der Auswertungen der JACOBI-Matrizen gegenüber dem linear-impliziten Verfahren mit inexakten JACOBI-Matrizen um 98.8 % reduziert werden kann.
Die Rechenzeit des semi-voll-impliziten Verfahrens mit inexakten JACOBI-Matrizen kann durch den Einsatz weiterer Parallelisierungsmethoden auf Integratorebene weiter gesenkt werden. Die Anwendung der Parallelisierung der Schrittweitensteuerung erzielt eine Rechenzeitreduktion von 26 %, die Anwendung der Parallelisierung der Datenspeicherung von 15 %.

An dieser Stelle sei explizit darauf hingewiesen, dass diese enorme Beschleunigung des impliziten Verfahrens gegenüber des halb-expliziten Verfahrens nur bei numerisch steifen Systemen zu erzielen ist. Die Beschleunigungen, die durch Parallelisierungsmethoden erzielt werden, sind im Gegensatz dazu vom System weitgehend unabhängig (siehe Tabelle 5.5).

Verwendet man in diesem Beispiel zur Modellierung des Balkens statt des E-Moduls und der Dichte von Stahl, den E-Modul und die Dichte von Gummi, so ändern sich die Ergebnisse signifikant (siehe Tabelle 5.5). Zwar können durch die Anwendung von

Tabelle 5.5: Ergebnisse bei Verwendung des E-Moduls und der Dichte von Gummi.

Verfahren	JACOBI **Updates**	**durchschnittl. Schrittweite**	**Rechendauer** [s]
halb. exp.	0	$1.50 \cdot 10^{-5}$ s	196
lin.-imp. (exakt. Jac.)	7302	$8.70 \cdot 10^{-5}$ s	492
lin.-imp. (inexakt. Jac.)	4678	$8.75 \cdot 10^{-5}$ s	372
semi-voll-imp. (inexakt. Jac.)	11	$8.83 \cdot 10^{-5}$ s	81
semi-voll-imp. (inexakt. Jac.) *- par. Schrittweitensteuerung*	11	$8.83 \cdot 10^{-5}$ s	58
semi-voll-imp. (inexakt. Jac.) *- par. Schrittweitensteuerung* *- par. Datenspeicherung*	11	$8.83 \cdot 10^{-5}$ s	49

impliziten Verfahren die Schrittweiten auch bei dem E-Modul und der Dichte von Gummi erhöht werden, jedoch nicht um den gleichen Faktor wie bei der numerisch steifen Konfiguration. Mit dem semi-voll-impliziten Verfahren können aber dennoch Beschleunigungen von etwa vier erreicht werden. Das linear-implizite Verfahren mit exakten JACOBI-Matrizen benötigt mehr als die doppelte Rechenzeit als das halbexplizite Verfahren, da es sich bei dem Beispiel nun um kein numerisch steifes System handelt.

5.7.2 Ordnungserhöhung impliziter Time-Stepping Verfahren

In diesem Abschnitt wird die Ordnungserhöhung durch Extrapolation von impliziten Time-Stepping Verfahren an dem industriellen Beispiel *Ventiltriebsdynamik* gezeigt. Dazu wird der Ventiltrieb von 100 auf 700 $^1/_{\text{min}}$ beschleunigt. Als Integrationsverfahren kommt beispielsweise ein linear-implizites Verfahren mit exakten JACOBI-Matrizen und Parallelisierung zum Einsatz.

Tabelle 5.6: Ordnungserhöhung durch Extrapolation am Beispiel *Ventiltriebsdynamik*.

Ordnung	**1**	**2**	**3**
Integrationsschritte	4777	1206	948
Auswertungen MKS	14949	9728	17497
abgelehnte Schritte	72	15	86
durchschnittl. Schrittweite [s]	$2.09 \cdot 10^{-5}$	$8.29 \cdot 10^{-5}$	$1.05 \cdot 10^{-4}$
Rechendauer [s]	721	446	668

Tabelle 5.6 beinhaltet die Ergebnisse des Vergleichs. Für die Integrationsordnungen eins, zwei und drei sind jeweils die Anzahl benötigter Integrationsschritte, die Anzahl an Auswertungen des Mehrkörpersystems, die Anzahl abgelehnter Schritte durch die Schrittweitensteuerung, die durchschnittliche Integrationsschrittweite und die Rechendauer angegeben.

Den Ergebnissen kann entnommen werden, dass die Ordnungserhöhung von Ordnung eins auf Ordnung zwei einen Rechenzeitvorteil von etwa 38 % erzielt. Eine weitere Erhöhung der Ordnung von zwei auf drei führt zwar zu einer nochmals höheren Integrationsschrittweite, jedoch verlängert sich die Rechenzeit trotzdem. Dies liegt daran, dass für das Verfahren 3. Ordnung das Mehrkörpersystem insgesamt 17497-mal ausgewertet werden muss, wohingegen das Verfahren 2. Ordnung lediglich 9728 Auswertungen benötigt. Die Erhöhung der Integrationsordnung ist wesentlich von der Kontinuität der rechten Seite der Differentialgleichung abhängig. Approximationen wie die Interpolation der Nockenkonturen mit kubischen Splines beschränken die Kontinuität der rechten Seite.

5.8 Kombination der Parallelisierungsmethoden

In Kapitel 4 werden zwei neue Parallelisierungsmethoden für die systeminterne Parallelisierung gezeigt und in Abschnitt 5.6 werden parallele Methoden auf Integratorebene dargestellt. In diesem Abschnitt sollen diese Parallelisierungsmethoden kombiniert und an einem Beispiel angewendet werden.

Die besondere Herausforderung bei der Kombination der Parallelisierungsmethoden liegt in den Ebenen der Parallelisierung. Die parallele Schrittweitensteuerung sieht beispielsweise vor, dass mehrere Instanzen des Systems mit unterschiedlichen Zeitschrittweiten Δt_i parallel berechnet werden. Parallelisiert man zudem die Berechnung der JACOBI-Matrizen, so wird innerhalb eines parallelen Abschnitts (Schrittweitensteuerung) ein zweites Mal parallelisiert. Gleiches gilt für die systeminterne Parallelisierung. Auch hier werden innerhalb eines parallelen Abschnitts weitere parallele Abschnitte verwendet. Dieses Vorgehen wird als *verschachtelte* Parallelisierung (engl. *nested* parallelisation) bezeichnet. Bei der verschachtelten Parallelisierung muss im besonderen Maße auf *Race Conditions* und *Dead Locks* geachtet werden.

In diesem Abschnitt wird die *semi-dynamische* Parallelisierungsmethode aus Abschnitt 4.3 herangezogen. Auf Integratorebene werden die Parallelisierung der JACOBI-Matrizen, die Parallelisierung der Schrittweitensteuerung und die Parallelisierung der Speicherung der Simulationsergebnisse betrachtet. Als Beispiel kommt das System *Gleitende Elemente auf flexiblem Balken* zum Einsatz (siehe Abschnitt 5.7), das mit dem semi-voll-impliziten Time-Stepping Verfahren mit inexakten JACOBI-Matrizen aus Abschnitt 5.4.2 integriert wird.

Die Ergebnisse sind in Abbildung 5.19 dargestellt. Die Parallelisierung der Berechnung der JACOBI-Matrizen erzielt in diesem Beispiel lediglich eine Beschleunigung von 5 %, da die JACOBI-Matrizen bei dem semi-voll-impliziten Verfahren mit inexakten JACOBI-Matrizen lediglich 11 mal ausgewertet werden müssen. Die Parallelisierung der Schrittweitensteuerung ergibt eine Rechenzeitreduktion von 28.4 %. Der theoretisch maximal zu erreichende Wert (siehe Abbildung 5.14) liegt bei 33.3 %. Die Parallelisierung der Datenspeicherung erzielt in diesem Beispiel eine Beschleunigung von zusätzlichen 13.2 %, die jedoch abhängig von der Plotschrittweite ist.

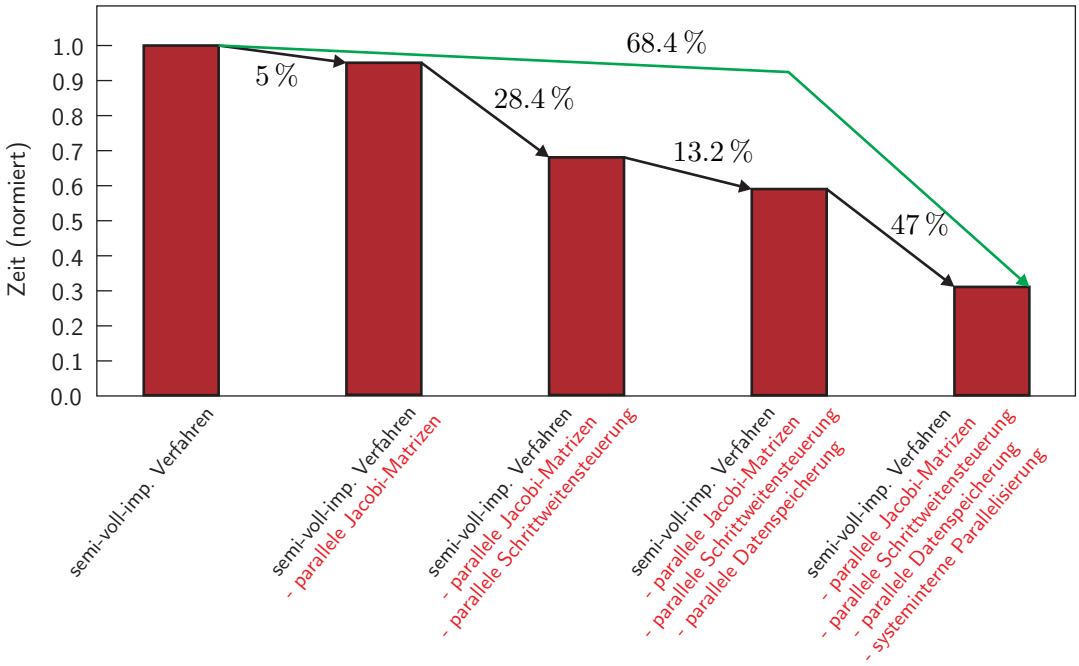


Abbildung 5.19: Kombination der Parallelisierung auf System- und Integratorebene.

Zuletzt können zusätzlich 47 % der Rechenzeit durch die systeminterne Parallelisierung eingespart werden. Insgesamt kann die Rechenzeit dieses Beispiels rein durch Parallelisierung um 68.4 % reduziert werden.

6 Anwendungsbeispiele

Die Effizienz aller in dieser Arbeit entwickelten Methoden wird sowohl an akademischen als auch an industriellen Beispielen gezeigt. Die verwendeten Beispiele werden in diesem Kapitel näher dargestellt, dabei wird sowohl auf den Modellaufbau als auch auf die Modellierung und mögliche Validierungen eingegangen, um einen tieferen Einblick in die Systeme zu gewähren.

In Abschnitt 6.1 werden vier typische akademische Beispiele gezeigt. Für die Simulation von Systemen mit vielen Körpern die untereinander in Kontakt treten können, wird das Beispiel *Kugeln in Schale* (Abschnitt 6.1.1) herangezogen, in dem eine große Anzahl gleicher Kugeln in eine Schale fällt. Das System *Spielzeugspecht an flexibler Stange* (Abschnitt 6.1.2) wird als akademisches Beispiel mit flexiblen und starren Körpern sowie mit Reibung, Spiel und Stößen verwendet. Die Systeme *Perlenkette* (Abschnitt 6.1.3) und *Gleitende Elemente auf flexiblem Balken* (Abschnitt 6.1.4) können als akademische Vorstudien zur Simulation von CVT-Getrieben (Abschnitt 6.2.2) gesehen werden.

In Abschnitt 6.2 werden drei industrielle Anwendungsbeispiele gezeigt. Als Vertreter für Simulationsmodelle in der Entwicklung von Verbrennungsmotoren wird in Abschnitt 6.2.1 ein Ventiltrieb gezeigt. Die Ventilfedern des Ventiltriebs werden mit einem kontinuierlichen Ventilfedermodell modelliert, dessen Herleitung und Validierung im Anhang A.1 zusammengefasst wird. Aus dem Bereich der Antriebstechnik wird in Abschnitt 6.2.2 das Modell eines CVT-Getriebes erläutert. Der letzte Abschnitt 6.2.3 beschäftigt sich mit einem flexiblen Rotorsystem, das zum modellbasierten Monitoring von Rotorsystemen eingesetzt wird.

Die Systeme werden in diesem Kapitel anhand weniger charakteristischen Kenngrößen klassifiziert:

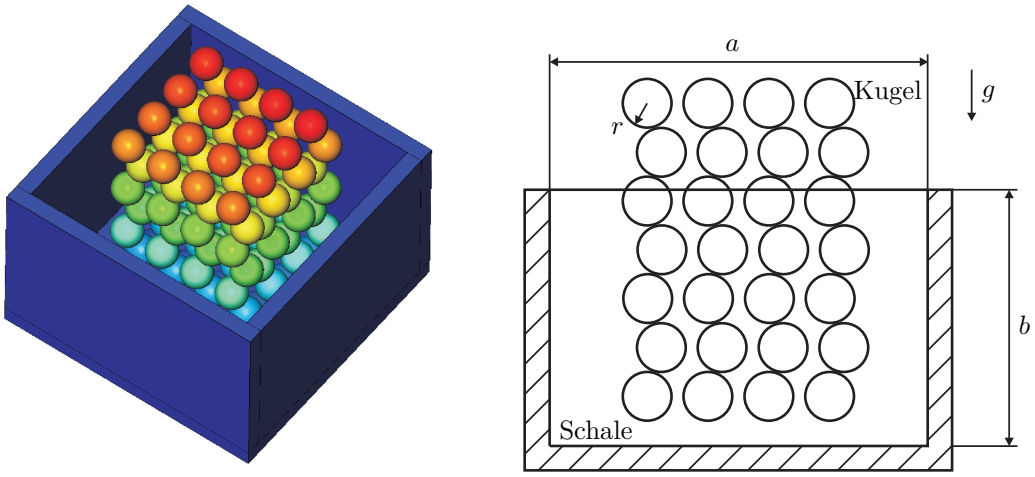
- **objectSize**
  Anzahl der Körper (OBJECTS) in der Simulation
- **linkSize**
  Anzahl der Bindungen (LINKS) in der Simulation
- **gSize**
  Anzahl der Abstandsberechnungen $\boldsymbol{g}$
- **$\boldsymbol{\lambda}$Size**
  Anzahl der mengenwertigen Kraftparameter $\boldsymbol{\lambda}$
- **hSize**
  Größe des Vektors der einwertigen externen, internen und gyroskopischen Kräfte $\boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t)$

Die verwendete Parametrisierung der generalisierten Positionen $\boldsymbol{q}$ und verallgemeinerten Geschwindigkeiten $\boldsymbol{u}$ führt dazu, dass die Anzahl der generalisierten Positionen, die Anzahl der verallgemeinerten Geschwindigkeiten und die Größe des Vektors $\boldsymbol{h}(\boldsymbol{q},\boldsymbol{u},t)$ identisch sind. Daher wird in den Beispielen lediglich die Größe **hSize** angegeben. Analog dazu entspricht die Anzahl der Abstandsgeschwindigkeitsberechnungen $\dot{\boldsymbol{g}}$ der Anzahl der mengenwertigen Kraftparameter $\boldsymbol{\lambda}$, weshalb nur der Vektor $\boldsymbol{\lambda}$ als charakteristische Kenngröße herangezogen wird.

6.1 Akademische Beispiele

6.1.1 Kugeln in Schale

Eine große Anzahl gleicher Körper interagieren zu lassen, ist ein bekanntes akademisches Beispiel, um die Simulation von Systemen mit sehr vielen Freiheitsgraden zu untersuchen. Ähnliche Simulationen findet man unter anderem in der Arbeit von Zander [151], der eine große Anzahl an Würfeln in eine flexible Schale fallen lässt. Förg et al. [64] untersuchen die Lösung von mengenwertigen Bindungen anhand der Simulation von granularen Medien in einer Sanduhr. Huber und Ulbrich [86] testen ihren Time-Stepping Integrator mit Schrittweitenadaption und Ordnungserhöhung anhand einer großen Anzahl an Kugeln, die gegen eine elastisch gelagerte Wand geschossen werden. Das System besteht aus einer Schale, die aus



(a) OpenMBV Visualisierung. (b) Geometriedaten *Kugeln in Schale*.

Abbildung 6.1: Visualisierung und Geometriedaten des Beispiels *Kugeln in Schale*.

fünf Begrenzungsflächen besteht. Die Begrenzungsflächen sind dabei nicht als eigenständige Körper modelliert, sondern sind als Flächen-Konturen Teil der Umgebung. Die n Kugeln sind versetzt zueinander angeordnet und berühren sich in ihrer Ausgangslage nicht. Durch den Einfluss der Gravitation $\boldsymbol{g}$ fallen sie in die modellierte Schale. Dabei können sowohl die Kugeln untereinander, als auch die Kugeln und

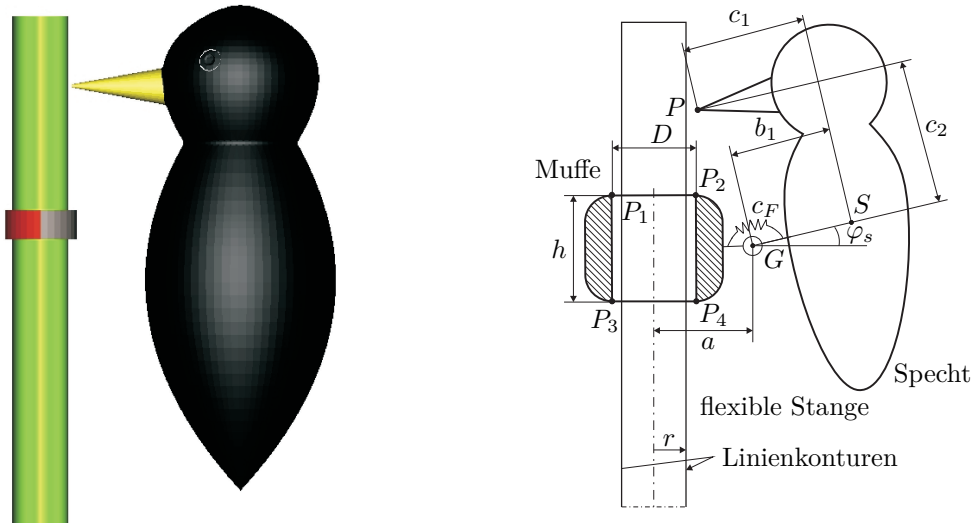
die Begrenzungsflächen in Kontakt treten. Dies führt zu einer sehr hohen Anzahl an Kontaktpaarungen, die zu jedem Zeitschritt ausgewertet werden müssen. Die Kontakte zwischen den Kugeln sind dabei als NEWTON-Stöße mit dem Stoßkoeffizienten ε_{KK} modelliert, die Kontakte zwischen den Kugeln und den Begrenzungsflächen sind ebenfalls als NEWTON-Stöße mit dem Stoßkoeffizienten ε_{KS} modelliert.
Die Geometrie- und Simulationsdaten des Beispiels sind in Tabelle A.4 zu finden, die charakteristischen Kenngrößen in Tabelle 6.1.

Tabelle 6.1: Charakteristische Kenngrößen des Beispiels *Kugeln in Schale*.

Kenngröße	
objectSize	112
linkSize	6776
gSize	6776
λSize	6776
hSize	336

6.1.2 Spielzeugspecht an flexibler Stange

Der *Spielzeugspecht an flexibler Stange* ist ein beliebtes Beispiel für die nicht-glatte hybride Mehrkörpersimulation. Unter anderem ist er auch in den Arbeiten von ZANDER [151], LEINE UND GLOCKER [98] und PFEIFFER [113] zu finden. Der Specht



(a) OpenMBV Visualisierung. (b) Geometriedaten Spielzeugspecht.

Abbildung 6.2: Visualisierung und Geometriedaten des Beispiels *Spielzeugspecht*.

besitzt als akademisches Beispiel nahezu alle Merkmale einer modernen Mehrkörpermodellierung.

Die Stange, an dem der Specht nach unten rutscht, ist als ebener flexibler Balken modelliert. Nähere Informationen zu dem verwendeten Balkenmodell können in der Dissertation von Zander [151] nachgelesen werden. Zur Modellierung werden $n_{\mathrm{FE}} = 5$ Finite-Elemente verwendet. Der Balken ist über eine mengenwertig formulierte zweiseitige Bindung mit der inertialen Umgebung verbunden, die die translatorische ebene Bewegung und die verbleibende rotatorische Bewegung am unteren Ende der Stange verhindert. Die Balkenkontur ist auf beiden Längsseiten des Balkens durch zwei zweimal stetig differenzierbare Linienkonturen modelliert.
Die Muffe kann sich frei in der Ebene translatorisch und rotatorisch bewegen. Der Kontakt zwischen der Muffe und dem flexiblen Balken wird durch die vier Punktkonturen P_1 bis P_4 modelliert, die in Kontakt zu den entsprechenden Linienkonturen auf der flexiblen Stange treten können. Die Kontakte sind als Newton-Stöße mit dem Stoßkoeffizienten ε modelliert, in denen die Reibung als Coulomb-Reibung (Reibkoeffizient μ) modelliert wird.
Der Specht ist über ein Drehgelenk mit der Muffe verbunden. Zwischen Specht und Muffe wirkt eine Drehfeder. An der Spitze des Schnabels ist eine Punktkontur angelegt, die mit der entsprechenden Linienkontur der flexiblen Stange interagieren kann.
Alle Geometrie- und Simulationsparameter sind in Tabelle A.5 gelistet. Der Specht bezieht seine Energie aus der Gravitation, wobei die reibungsbehafteten Kontakte der Muffe als eine Art Schalter dienen, um die Drehfeder zwischen Muffe und Specht zu spannen. Die dadurch entstehende Schwingung des Spechts führt zu einem Öffnen und Schließen der Kontakte innerhalb der Muffe. Bei geöffneten Kontakten bzw. rutschenden geschlossenen Kontakten gleitet der Specht nach unten. Nach einer gewissen Falldistanz gehen die Kontakte wieder in selbsthemmende Haftreibung über, wodurch der Zyklus wieder von Neuem beginnt.

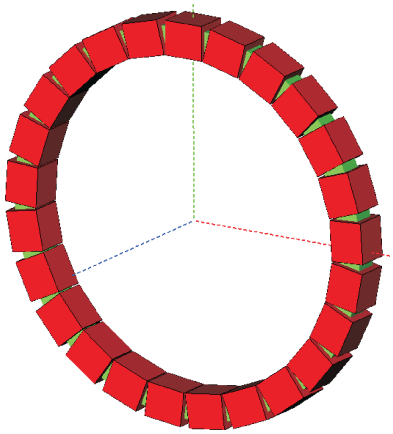
Die charakteristischen Kenngrößen sind in Tabelle 6.2 zu finden.

Tabelle 6.2: Charakteristische Kenngrößen des Beispiels *Spielzeugspecht*.

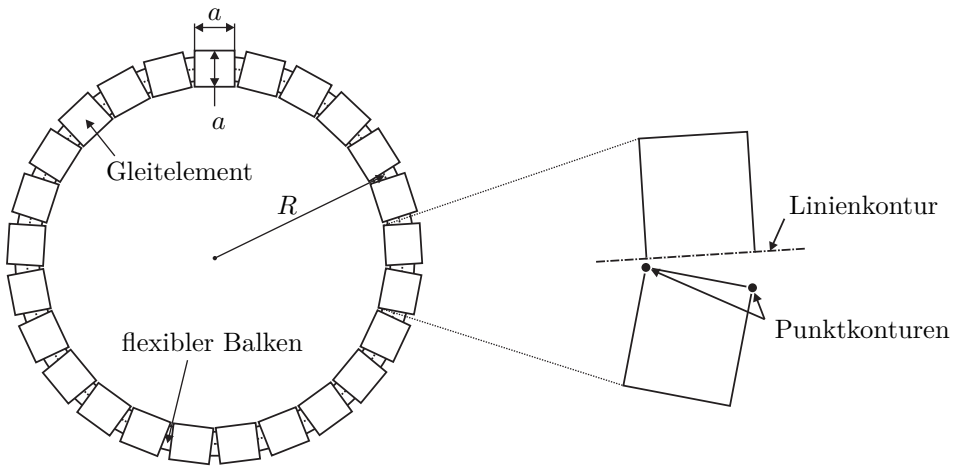
Kenngröße	
objectSize	3
linkSize	7
gSize	8
λSize	18
hSize	32

6.1.3 Perlenkette

Das Beispiel der *Perlenkette* kann als Vorstudie zur Simulation von Umschlingungsgetrieben [35, 68, 126] (Abschnitt 6.2.2) gesehen werden. Es ist in ähnlichen Formen auch in den Arbeiten von Zander [151], Schindler [126] und Friedrich [65] zu finden.



(a) OpenMBV Visualisierung.



(b) Geometriedaten *Perlenkette*.

Abbildung 6.3: Visualisierung und Geometriedaten des Beispiels *Perlenkette*.

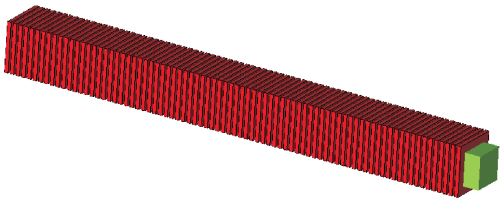
Basis des Modells ist ein ebener flexibler Balken, dessen undeformierte Lage einen Kreis mit Radius R darstellt. Die Modellierung des ebenen flexiblen Balkens ist in der Dissertation von Zander [151] nachzulesen. Der Balken besitzt den E-Modul E, die Länge $L = 2\pi R$, die Querschnittfläche A, die Dichte ρ und wird mit $n_{\mathrm{FE}} = 20$ Elementen modelliert. Der Balken erlaubt große Deformationen.
Auf dem Balken gleiten $n = 25$ Elemente, von denen das oberste Element inertial befestigt ist. Die Fesselung der gleitenden Elemente auf dem flexiblen Balken ist als mengenwertige zweiseitige Bindung realisiert. Zudem können die Elemente untereinander in Kontakt treten. Dazu besitzt jedes Element auf einer Seite eine Linienkontur und auf der anderen Seite jeweils an den äußeren Ecken zwei Punktkonturen (siehe Abbildung 6.3). Die Kontakte sind als Newton-Stöße mit dem Stoßkoeffizienten ε modelliert. Das System unterliegt der Erdbeschleunigung g, sodass die gleitenden Elemente (ausgenommen dem obersten gefesselten Element) nach unten rutschen und dort zum Liegen kommen.

Die Geometrie- und Simulationsdaten des Beispiels sind in Tabelle A.6 zu finden, die charakteristischen Kenngrößen in Tabelle 6.3.

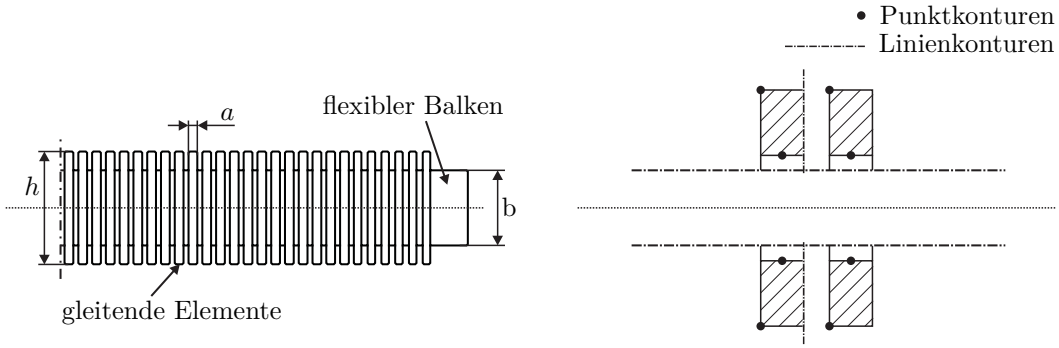
Tabelle 6.3: Charakteristische Kenngrößen des Beispiels *Perlenkette*.

Kenngröße	
objectSize	26
linkSize	75
gSize	77
λSize	77
hSize	175

6.1.4 Gleitende Elemente auf flexiblem Balken



(a) OpenMBV Visualisierung.



(b) Geometriedaten *Gleitende Elemente auf flexiblem Balken*.

Abbildung 6.4: Visualisierung und Geometriedaten des Beispiels *Gleitende Elemente auf flexiblem Balken*.

Das Beispiel der *gleitenden Elemente auf einem flexiblem Balken* kann wie auch das Beispiel *Perlenkette* (siehe 6.1.3) als Vorstudie zur Simulation von Umschlingungsgetrieben [35, 68, 126] (Abschnitt 6.2.2) gesehen werden.

Basis des Modells ist ein flexibler ebener Balken, dessen Modellierung in der Dissertation von ZANDER [151] nachgelesen werden kann. Der Balken besitzt den E-Modul E, die Länge L, die Querschnittfläche A, die Dichte ρ und wird mit $n_{\mathrm{FE}} = 4$ Finiten Elementen modelliert. Der Balken erlaubt große ebene Deformationen.
Der Balken ist auf der linken Seite mit einer zweiseitigen mengenwertigen Lagerung gegenüber der Umgebung gefesselt.

Auf dem Balken gleiten $n = 80$ Elemente, die jedoch - im Gegensatz zum Beispiel *Perlenkette* - nicht über eine zweiseitige Bindung mit dem Balken verbunden sind, sondern Spiel besitzen und sich somit gegenüber dem Balken senkrecht zur Balkenlängsachse bewegen können. Die Bindung ist durch jeweils zwei Punktkontakte gegenüber dem flexiblen Balken realisiert, die als NEWTON-Stöße mit dem Stoßkoeffizienten $\varepsilon_{\mathrm{EB}}$ modelliert sind. Zudem können die gleitenden Elemente untereinander in Kontakt treten. Die Kontakte zwischen den gleitenden Elementen sind ebenfalls als NEWTON-Stöße mit dem Stoßkoeffizienten $\varepsilon_{\mathrm{EE}}$ modelliert.

Tabelle 6.4: Charakteristische Kenngrößen des Beispiels *Gleitende Elemente auf flexiblem Balken.*

Kenngröße	
objectSize	81
linkSize	318
gSize	321
λSize	321
hSize	263

Dieses System eignet sich im Gegensatz zum System *Perlenkette* auch zum Testen von Integrationsmethoden für steife Systeme mit häufigen Stößen. Das System unterliegt der Erdbeschleunigung g.
Die Geometrie- und Simulationsdaten des Beispiels sind in Tabelle A.7 zu finden, die charakteristischen Kenngrößen in Tabelle 6.4.

6.2 Industrielle Anwendungsbeispiele

6.2.1 Ventiltriebsdynamik

Die Ventiltriebsdynamik stellt einen wesentlichen Anwendungszweck der Mehrkörpersimulation dar. Moderne Ventiltriebsimulationen benötigen die Kombination von flexibler (z.B. Ventilfedern) und starrer Mehrkörperdynamik, Hydraulik (z.B. HVAs[1]) und Regelungstechnik (z.B. Regelung der Nockenwellenverstellung). Deshalb liegt

1 HVA - Hydraulisches Ventilspielausgleichselement. Durch das HVAs wird das Ventilspiel ausgeglichen.

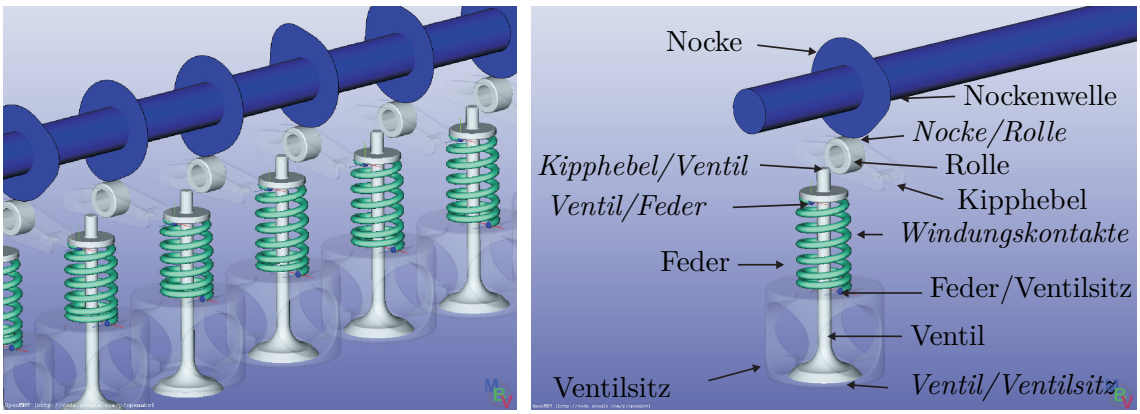


Abbildung 6.5: Visualisierung und Geometriedaten des Beispiels *Ventiltriebsdynamik*.

es nahe, auch ein Modell von einem typischen Ventiltrieb als industrielles Anwendungsbeispiel heranzuziehen. Weitere Informationen zur Simulation von Ventiltrieben können in den Werken [50, 62, 84, 94, 131] gefunden werden.

Der in diesem Kapitel präsentierte Ventiltrieb ist in Abbildung 6.5 abgebildet, die Geometriedaten sind in Tabelle A.8 angegeben. Er besteht aus 12 Ventilmechanismen, die jeweils aus den auf der rechten Seite der Abbildung 6.5 gezeigten Elementen bestehen. In der Abbildung sind Körper in normaler Schrift bezeichnet und Kontaktpaarungen in kursiver Schrift.
Die Nockenwelle kann wahlweise als elastischer oder starrer Körper modelliert werden. In dieser Arbeit wird die flexible Variante verwendet, insofern nicht explizit angemerkt ist, dass die starre Version verwendet wird. Bei einer elastischen Formulierung wird sie als elastischer Balken mit dem E-Modul E, dem G-Modul (Schubmodul) G, der Dichte ρ, der Querschnittsfläche A, der Länge L und dem Radius r modelliert. Die Diskretisierung erfolgt mit $n_{\mathrm{FE}} = 5$ Finiten-Elementen. Als Kontaktpartner zu den Ventilmechanismen trägt die Nockenwellen die Nockenkonturen.
Jeder Ventilmechanismus besteht aus den gleichen Elementen. Auf der Nocke rollt die Rolle ab, die mit dem Kipphebel verbunden ist. Der Kipphebel ist auf einer Seite drehbar gegenüber der Umgebung gelagert und überträgt die Bewegung der Nocke auf das Ventil. Die Ventilfeder presst das Ventil gegen den Kipphebel, wodurch ein sicheres Schließen und Öffnen des Ventils sichergestellt werden soll. Die Ventilfeder stützt sich auf der anderen Seite gegen den Ventilsitz ab. Ventil und Ventilsitz können ebenfalls in Kontakt treten.
Die Ventilfedern sind mit dem in Abschnitt A.1 vorgestellten massebehafteten Ventilfedermodell modelliert, deren Windungen in Kontakt treten können.
Alle in der Abbildung 6.5 kursiv benannten Kontaktpaarungen sind in Tabelle 6.5 mit den zugehörigen Kontaktgeometrien aufgeführt. Tabelle 6.6 beinhaltet die charakteristischen Kenngrößen für das Ventiltriebsmodell, je für die Modellierung mit einer flexiblen Nockenwelle und für die Modellierung mit einer starren Nockenwelle.

Körper 1/Körper 2	Kontaktgeometrie
Nocke/Rolle	Spline/Kreis
Kipphebel/Ventil	Linie/Punkt
Feder/Ventil	Punkt/Ebene
Feder/Ventilsitz	Punkt/Ebene
Ventil/Ventilsitz	Punkt/Linie

Tabelle 6.5: Kontaktpaarungen im Ventilmechanismus.

Tabelle 6.6: Charakteristische Kenngrößen des Beispiels *Ventiltriebsdynamik*.

Kenngröße	flex. Nockenwelle	starre Nockenwelle
objectSize	61	61
linkSize	132	132
gSize	156	156
λSize	156	156
hSize	307	278

6.2.2 CVT (Continuously Variable Transmission)

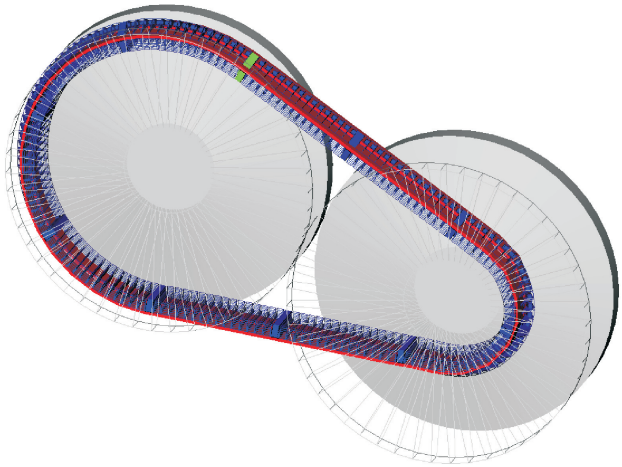


Abbildung 6.6: OpenMBV Visualisierung des Beispiels *CVT*.

Das Simulationsmodell eines CVTs[2] besitzt eine sehr hohe Anzahl an Freiheitsgraden und an uni- und bilateralen Bindungen.
Das Modell geht auf die Arbeiten von Geier [68] und Schindler [126] zurück, in deren Arbeiten auch nähere Informationen zu der Modellierung gefunden werden können. Geier stellt in seiner Dissertation ein ebenes Modell (2D) des Getriebes dar, auf dessen Grundlage Schindler in seiner Dissertation die Modellierung auf

2 Continuously Variable Transmission - Getriebe mit variabler Übersetzung. Die variable Übersetzung wird bei diesem Typ von Schubglieder-CVTs durch das axiale Verstellen der Antriebs- und Abtriebsscheibe erreicht.

eine räumliche (3D) Dynamik erweitert.

Tabelle 6.7: Charakteristische Kenngrößen des Beispiels *2D CVT*.

Kenngröße	
linkSize	1760
gSize	326
λSize	646
hSize	657

Tabelle 6.8: Charakteristische Kenngrößen des Beispiels *3D CVT*.

Kenngröße	
linkSize	3040
gSize	645
λSize	1285
hSize	1211

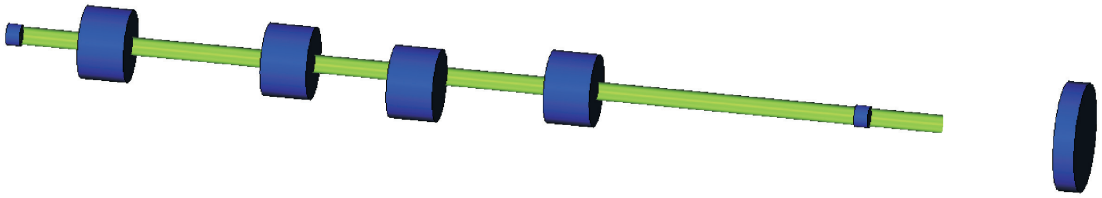
Zu den wesentlichen Merkmalen des Modells gehört, dass die Dynamik des Schubgliederbandes durch ein räumliches Balkenmodell abgebildet wird, das große Deformationen erlaubt. Das 3D-System verfügt über insgesamt 1211 Freiheitsgrade und über 3000 Kontaktberechnungen. Diese enorme Größe stellt hohe Anforderungen an die Simulationsumgebung. Die charakteristischen Größen sind für das 2D-Modell in Tabelle 6.7 und für das 3D-Modell in Tabelle 6.8 angegeben.

6.2.3 Rotordynamik

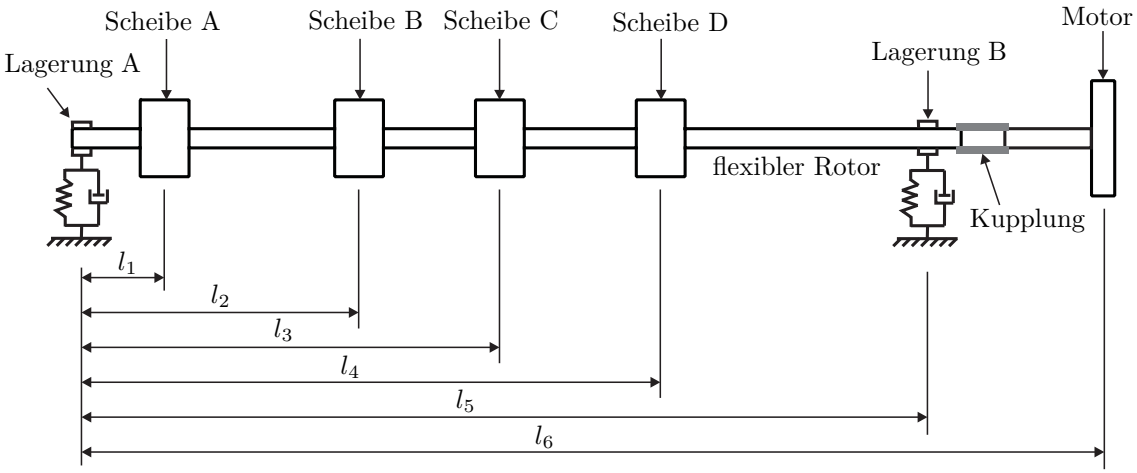
Das in Abbildung 6.7 gezeigte Rotorsystem dient dem modellbasierten Monitoring von Rotorsystemen. Heckmann et al. [78] detektieren basierend auf diesem Simulationsmodell plötzlich auftretende Unwuchten im System. In ähnlicher Form nutzte Ginzinger [69] das System für die Auslegung und Regelung von aktiven Fanglagern an Rotorsystemen.

Tabelle 6.9: Charakteristische Kenngrößen des Beispiels *Rotordynamik*.

Kenngröße	
objectSize	8
linkSize	9
gSize	15
λSize	15
hSize	60



(a) OpenMBV Visualisierung.



(b) Geometriedaten Rotorsystem.

Abbildung 6.7: Visualisierung und Geometriedaten des Beispiels *Rotordynamik*.

Das System besteht aus einem flexiblen Rotor mit dem E-Modul E, dem G-Modul (Schubmodul) G, der Dichte ρ, der Querschnittsfläche A, der Länge L und dem Radius r. Die Modellierung erfolgt mittels $n_{\mathrm{FE}} = 5$ Finiten-Elementen. Der Rotor ist an den Stellen *Lagerung A* und *Lagerung B* mit der Umgebung über elastische Lager gelagert. Der Antrieb des Rotors erfolgt über den *Motor*, der über eine *Kupplung* an den Rotor angebunden ist. Die Drehzahl des Motors wird über einen PID-Regler [100] eingeregelt. Auf dem Rotor sind die Scheiben *A,B,C* und *D* mittels einer mengenwertigen zweiseitigen Bindung befestigt. Die Unwuchten der Scheiben können beliebig variiert werden, um plötzlich auftretende Unwuchten modellieren zu können. Die Geometrie- und Simulationsdaten des Beispiels sind in Tabelle A.9 zu finden, die charakteristischen Kenngrößen in Tabelle 6.9. Auf das System wirkt die Erdbeschleunigung g.

7 Zusammenfassung

Simulationsmethoden stellen einen integralen Bestandteil des heutigen Produktentwicklungsprozesses dar. Durch einen stetig steigenden Anteil an virtuellen Methoden am gesamten Produktentwicklungsprozess wird versucht auf den immer höher werdenden Kostendruck zu reagieren.
Simulationsmethoden erlauben es zeit- und kostenaufwendige reale Hardware Versuche durch schnelle und kostengünstige virtuelle Simulationen zu ersetzen. Neben der Kosten- und Zeiteinsparung ermöglichen Simulationsmethoden in der Regel auch mehr Flexibilität. Beispielsweise können virtuelle Sensoren an nahezu beliebigen Stellen des virtuellen Systems appliziert werden, wohingegen die Applikation von Sensoren in realen Systemen teils sehr komplex bzw. nicht möglich ist.

Mit dem zunehmenden Einsatz von Simulationsmethoden steigt jedoch auch der Anspruch an die Modelle, vor allem wenn Simulationen in späteren Stadien des Produktentwicklungsprozesses eingesetzt werden sollen. Eine höhere Komplexität und eine zunehmende Größe der Simulationsmodelle bedeutet jedoch auch eine steigende Rechenzeit. Bisher konnte ein Teil des höheren numerischen Aufwands dadurch ausgeglichen werden, dass die Taktfrequenzen der CPUs üblicher Simulationsrechner stetig gestiegen sind. Derzeit geht der Trend jedoch in die Richtung, dass einerseits die Taktfrequenz der Prozessoren stagniert, jedoch andererseits immer mehr Rechenkerne pro CPU zur Verfügung stehen.
Folglich müssen Methoden entwickelt werden, die das enorme Potential von heutigen Mehrkern-Arbeitsplatzrechnern oder Workstations ausnutzen können.

Eine zentrale Rolle unter den Simulationsmethoden nimmt die Mehrkörpersimulation ein, die als Basis vieler Simulationsanwendungen gesehen werden kann. Die nichtglatte Mehrkörperdynamik bietet Methoden um auch komplexe Mehrkörpersysteme mit einer hohen Anzahl an ein- und zweiseitigen Bindungen effizient simulieren zu können. Zusätzlich können Stöße und Reibung physikalisch korrekt modelliert werden, indem die zugehörigen physikalischen Gesetze mengenwertig und nicht - wie in der glatten Mehrkörpertheorie - regularisiert formuliert werden.

Anders als in der Mehrkörpersimulation haben parallele Methoden in anderen Bereichen der Simulationsmethoden schon vor Jahren Einzug gehalten, wie beispielsweise in den FE[1]-Methoden oder auch den CFD[2]-Methoden.

In dieser Arbeit werden effiziente parallele Methoden zur Berechnung von nichtglatten dynamischen Systemen, insbesondere von Mehrkörpersystemen, vorgestellt.

1 FE = Finite Elemente
2 CFD = Computational Fluid Dynamics

Ein wesentliches Ziel ist dabei, dass alle entwickelten Methoden für bereits bestehende Simulationsmodelle und auch für neue Simulationsmodelle angewendet werden können.

Unter anderem aus diesem Grund werden alle Methoden und Verfahren an sinnvollen Beispielen angewendet und diskutiert. In Kapitel 6 werden sowohl typische akademische als auch industrielle Anwendungsbeispiele vorgestellt, die sich als eine Art roter Faden durch alle Verfahren und Methoden der Arbeit ziehen. Die Anwendungsbeispiele reichen von einfachen akademischen Beispielen, wie einem Spielzeugspecht an einer flexiblen Stange, bis hin zu hochkomplexen Simulationsmodellen von Ventiltrieben und CVT-Getrieben.

Um das Potential der entwickelten Methoden zeigen zu können, wurden alle Methoden in die Mehrkörpersimulationsumgebung MBSim [13] implementiert. Dadurch kann einerseits auf eine hohe Anzahl vorhandener Simulationsmodelle zurückgegriffen werden und andererseits stehen die Methoden direkt für die Anwendung zur Verfügung. Die Simulationsumgebung und die verwendete Hardwarearchitektur werden in Abschnitt 1.2 dargestellt.

Die in dieser Arbeit entwickelten Verfahren werden an Beispielen der nicht-glatten Mehrkörperdynamik demonstriert, können jedoch analog für weitere physikalische Bereiche verwendet werden. Die Dynamik von nicht-glatten Mehrkörpersystemen, die auch die Dynamik von glatten Mehrkörpersystemen beinhaltet, wird in Kapitel 2 dargestellt. Begonnen bei den Bewegungsgleichungen für glatte Mehrkörpersysteme, wird die Erweiterung auf die Bewegungsgleichungen für nicht-glatte ein- und zweiseitig gebundene Mehrkörpersysteme gezeigt. Neben den Bewegungsgleichungen wird auf die wichtigsten mechanischen mengenwertigen Kraftgesetze und mögliche Lösungsverfahren eingegangen. Da die Arbeit neben der Theorie der Methoden auch auf die Implementierung und die Anwendung der Methoden eingeht, wird am Ende des Kapitels eine mögliche Umsetzung der Struktur der nicht-glatten Mehrkörperdynamik in eine Simulationsumgebung dargestellt, die sich an der Simulationsumgebung MBSim orientiert.

Kapitel 3 beschäftigt sich allgemein mit der Thematik der Parallelisierung. Nachdem ein Überblick über mögliche Hardwarearchitekturen gegeben worden ist, auf denen sich parallele Methoden zur Berechnung dynamischer Systeme umsetzen lassen, wird auf wichtige Begriffe und Definitionen eingegangen. Es wird dargestellt, dass sich Parallelisierungsmethoden in drei Ebenen eingliedern lassen. In die *feinstrukturierte*, *mittelstrukturierte* und *grobstrukturierte* Ebene. Alle drei Ebenen werden in dieser Arbeit bezüglich der Anwendung zur parallelen Berechnung nichtglatter mechanischer Systeme untersucht. In Abschnitt 3.5 wird gezeigt, dass die *feinstrukturierte* Ebene, zu der unter anderem die Parallelisierung der linearen Algebra gehört, wenig Potential bietet. Die *mittelstrukturierte* Ebene wird in Kapitel 4 und die *grobstrukturierte* Ebene in Kapitel 5 behandelt.
Abschließend führt das Kapitel 3 in die Parallelprogrammierung mit OpenMP [16] ein und erklärt die Herausforderung der Lastverteilung innerhalb paralleler Software.

Zu den Parallelisierungsmethoden der *mittelstrukturierten* Ebene gehören unter anderem die Methoden zur systeminternen Parallelisierung, die in Kapitel 4 vorgeschlagen werden. Zu Beginn des Kapitels wird auf die Idee und die Herausforderungen der systeminternen Parallelisierung eingegangen.
Die Bewegungsgleichungen nicht-glatter mechanischer Systeme lassen sich mit dem projektiven Newton-Euler Formalismus implementieren, der es erlaubt, die Anteile der einzelnen Körper an den gesamten Bewegungsgleichungen parallel zu berechnen.
Jede Form der Parallelisierung benötigt einen administrativen Overhead für die Verwaltung der Parallelisierung. Je länger die einzelnen parallelen Abschnitte dauern, umso weniger relevant ist der Overhead. Die Berechnung der Anteile der einzelnen Körper an den gesamten Bewegungsgleichungen benötigt jedoch vergleichsweise geringe Ausführungszeiten, sodass der Overhead einen maßgeblichen Einfluss auf die Effizienz der Parallelisierung hat. In Kapitel 4 werden zwei neu entwickelte Methoden vorgestellt, mit denen sich der Overhead bei der parallelen Berechnung dynamischer Systeme signifikant reduzieren lässt. Die *Grenzzeit* Parallelisierungsmethode und die *semi-dynamische* Parallelisierungsmethode.
Die *Grenzzeit* Parallelisierungsmethode setzt dabei auf herkömmliche Lastverteilungsalgorithmen auf, die beispielsweise von OpenMP bereitgestellt werden. Für die *semi-dynamische* Parallelisierungsmethode wird ein eigenständiger statischer Lastverteilungsalgorithmus präsentiert, der auf der Theorie des Multi-Way Number Partitioning basiert, einem Teilgebiet der Stochastik [90].
Beide Methoden werden an den Beispielen aus Kapitel 6 angewendet und diskutiert. Sowohl mit der *Grenzzeit* Parallelisierungsmethode als auch mit der *semi-dynamischen* Parallelisierungsmethode können signifikante Beschleunigungen bei gleichzeitig sehr guten Effizienzwerten erreicht werden. Am Ende des Kapitels wird ein Vergleich der beiden Methoden gegeben.

Neben den Parallelisierungsmethoden auf der *mittelstrukturierten* Ebene, sind für die Mehrkörperdynamik vor allem Methoden der *grobstrukturierten* Ebene von hoher Bedeutung, zu denen auch die Methoden auf Integratorebene in Kapitel 5 gehören. Kapitel 5 beinhaltet zwei Aspekte der effizienten Integration von nicht-glatten dynamischen Systemen. Einerseits werden zwei implizite Time-Stepping Verfahren mit inexakten Jacobi-Matrizen vorgestellt und andererseits wird auf die Parallelisierung auf Integratorebene eingegangen.
Das Kapitel beginnt mit einem Überblick über die gebräuchlichsten Time-Stepping Verfahren und ihre Vor- und Nachteile, bevor die Potentiale und Herausforderungen von impliziten Verfahren dargestellt werden. Implizite Verfahren bieten das Potential die Integrationsschrittweite bei numerisch steifen Systemen höher zu wählen als explizite Verfahren. Jedoch werden für implizite Verfahren die Ableitungen der rechten Seite nach den Systemzuständen (Jacobi-Matrizen) benötigt. Die Berechnung der Jacobi-Matrizen kann je nach System einen maßgeblichen Anteil an der Gesamtsimulationsdauer einnehmen, sodass Methoden gefunden werden müssen, um diesen Anteil zu reduzieren.
Daher werden zwei implizite Time-Stepping Verfahren vorgestellt, die mit inexakten Jacobi-Matrizen arbeiten. Ein linear-implizites Verfahren und ein semi-voll-

implizites Verfahren. Das linear-implizite Verfahren koppelt die Auswertung der JACOBI-Matrizen an die Schrittweitensteuerung des Verfahrens und das semi-voll-implizite Verfahren erlaubt es die JACOBI-Matrizen nur dann auszuwerten, wenn dies notwendig ist. Um die Notwendigkeit zu erkennen, wird ein entsprechendes Verfahren präsentiert. Das enorme Potential des semi-voll-impliziten Verfahrens wird an zwei Beispielen gezeigt und diskutiert. Es können unter den richtigen Rahmenbedingungen Beschleunigungen von etwa 30 erreicht werden.
Um die Integrationsordnung der vorgestellten Verfahren zu erhöhen, werden Extrapolationsverfahren herangezogen. Zur Schrittweitenadaption wird ein modifizierter RICHARDSON-Fehlerschätzer eingesetzt.
Der zweite Teil des Kapitels 5 stellt Methoden zur Parallelisierung auf der Integratorebene dar. Extrapolationsverfahren eignen sich hervorragend zur parallelen Implementierung, da sich die einzelnen Zwischenschritte der Extrapolation (unterschiedliche Zeitschrittweiten Δt) unabhängig voneinander parallel berechnen lassen. Neben den Zwischenschritten der Extrapolationsverfahren wird gezeigt, dass auch das Speichern der Ergebnisse parallelisiert werden kann. Dazu werden die Ergebnisse des vorherigen Integrationsschritts gespeichert, während auf einem anderen Rechenkern bereits der nächste Schritt berechnet wird. Neben der Reduktion der Rechenzeit für die JACOBI-Matrizen mittels der vorgeschlagenen Time-Stepping Verfahren mit inexakten JACOBI-Matrizen, kann die Rechenzeit auch durch Parallelisierung der Berechnung der JACOBI-Matrizen reduziert werden. Diese Anwendung der Parallelisierung zeigt sehr hohe Beschleunigungen bei außerordentlich guten Effizienzen.
Das Potential aller Methoden und Verfahren aus Kapitel 5 wird an akademischen und industriellen Beispielen aus Kapitel 6 gezeigt.

Die im Rahmen dieser Arbeit entwickelten Parallelisierungs- und Integrationsmethoden sind in der Mehrkörpersimulationsumgebung MBSIM implementiert. Sie ermöglichen es, die Rechenzeiten von vorhandenen und neuen Simulationsmodellen durch effiziente Integrationsverfahren und verschiedene Parallelisierungsmethoden signifikant zu senken. Dabei benötigen die Methoden keine Parametrisierung durch den Anwender.

Weitere Forschungsaktivitäten können in der Kombination der Shared-Memory und Distributed-Memory Parallelisierung (siehe Abschnitt 3.2) gesehen werden. Beispielsweise könnten durch diese Vorgehensweise die Rechenzeiten der JACOBI-Matrizen weiter gesenkt werden.

A Anhang

A.1 Dynamik von Ventilfedern

Der Ventiltrieb von Verbrennungskraftmaschinen steuert das Öffnen und Schließen der Ventile, die den Gas-Austausch steuern und somit maßgeblich die Leistung und Effizienz des gesamten Motors beeinflussen. Die Dynamik der Ventiltriebe wird unter anderem durch die Dynamik der Ventilfedern beeinflusst, die das Bauteil mit der niedrigsten Eigenfrequenz im Ventiltrieb darstellen. Zur Auslegung konventioneller Ventiltriebe ist es somit unerlässlich, dass effiziente und genaue Modelle von Ventilfedern zur Verfügung stehen.

Die wesentliche Aufgabe der Ventilfedern ist es, die Ventile verlässlich zu schließen und geschlossen zu halten. Dabei entsteht der Zielkonflikt, dass die Federkraft einerseits hoch genug sein muss, dass die Kontakte zwischen den Nocken und Rollen geschlossen bleiben und die Ventile sicher geschlossen bleiben, andererseits sollte die Federkraft möglichst gering gehalten werden, damit die Reibung möglichst gering und somit der Wirkungsgrad möglichst hoch ist.

Fast alle derzeit in Serienproduktion verbauten Ventilfedern besitzen eine progressive Kennlinie. Diese Kennlinie wird durch eine nicht-konstante Steigung zwischen den einzelnen Federwindungen erreicht. Windungen mit geringerem Abstand kommen früher in Kontakt und führen somit zu einer Erhöhung der Steifigkeit der Ventilfeder und somit zu höheren Resonanzfrequenzen. Während des Betriebs wird hauptsächlich die erste Eigenfrequenz der Ventilfedern angeregt, was zu starken Schwingungen innerhalb der Feder oder auch zu Stößen zwischen den Federwindungen führen kann. Beide Effekte bedeuten hohe Kraftamplituden, die in der Auslegung der Ventiltriebe berücksichtigt werden müssen.

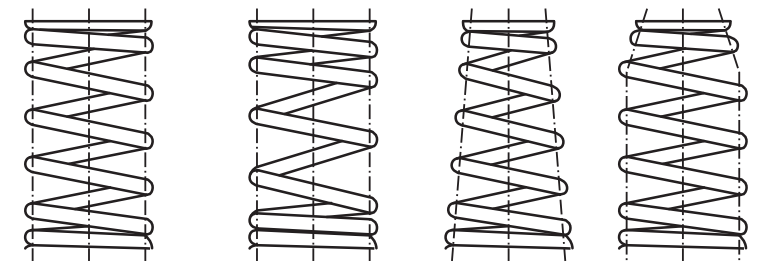


Abbildung A.1: Gebräuchliche Ventilfederformen.

Es haben sich hauptsächlich vier Federformen (siehe Abbildung A.1) in der Motorenentwicklung etabliert [26]. Zylindrische symmetrische Ventilfedern zeichnen sich durch einen konstanten Drahtdurchmesser und eine symmetrische Bauform aus. Eng verwandt mit den zylindrischen symmetrischen Ventilfedern sind die zylindrischen asymmetrischen Ventilfedern, die eine engere Wicklung auf der zum Motorblock gewandten Seite zeigen. Konische Ventilfedern bieten den Vorteil einer kürzeren Blocklänge[1] und geringeren bewegten Massen, verfügen in der Regel aber über ein weniger progressives Verhalten. Zuletzt seien bienenkorbförmige Ventilfedern genannt, die die Vorteile aus zylindrischen und konischen Bauformen vereinen. Geringe bewegte Massen werden durch den oberen Teil der Federn erreicht, der konisch geformt ist, wobei das progressive Verhalten durch den unteren Teil der Federn erreicht wird, der an die Bauform der zylindrischen Federn angelehnt ist.

Für die Simulation von Ventilfedern ist es nicht ausreichend, nur das statische Verhalten zu berücksichtigen (z.B. in Form von nicht-linearen Kennfeldern). Diese Methodik würde die Dynamik der Federn an sich vernachlässigen und somit könnten Phänomene, wie Resonanzfrequenzen und Windungsschlagen, nicht modelliert werden.

Aus diesem Grund wurde im Rahmen mehrerer Arbeiten am Lehrstuhl für Angewandte Mechanik [37, 38, 40, 41, 81–83, 135] ein effizientes Federmodell hergeleitet und experimentell abgeglichen, das zum Ziel hat, eine möglichst geringe Rechenzeit zu benötigen, aber trotzdem alle relevanten dynamischen Effekte abbilden zu können.

Die Modellierung basiert auf einem kontinuierlichen, gekrümmten Balkenmodell. Eine nicht-glatte Kontaktmodellierung zwischen den Federwindungen erlaubt Effekte, wie ein progressives Verhalten und Windungsschlagen, abzubilden. Geringe Rechenzeiten werden durch eine niedrige Anzahl an Freiheitsgraden und die Vermeidung steifer Differentialgleichungen erreicht, die eine Beschränkung der möglichen Integrationsschrittweite bedeuten würden.

Im ersten Abschnitt wird ein Überblick an vorhandenen Simulationsmodellen und Arbeiten zum Thema Federmodellierung und -simulation gegeben. Die darauf folgenden Abschnitte erläutern die wichtigsten Aspekte des entwickelten Federmodells, begonnen bei der Herleitung der Bewegungsgleichungen bis hin zur Erweiterung der Kontaktmodellierung für Federn mit nicht-konstantem Radius. Der letzte Abschnitt zeigt experimentelle statische und dynamische Validierungen des Federmodells anhand einer konischen und einer bienenkorbförmigen Ventilfeder.

A.1.1 Literatur- und Modellüberblick

In der Ventiltriebsimulation sind drei Modellierungsvarianten gebräuchlich. Modale Federmodelle, Mehr-Massen-Federmodelle und Federmodelle, die auf geraden Balkenelementen basieren.

1 Als Blocklänge wird diejenige Länge bezeichnet, die die Feder im komplett gestauchten Zustand einnimmt.

Das modale Federmodell basiert auf der gewichteten Überlagerung der ersten Eigenfrequenzen. Durch diese Herangehensweise kann die Anzahl der Freiheitsgrade gering gehalten werden, jedoch können keine Windungskontakte modelliert werden. Der Ansatz eines reduzierten modalen Modells wird von Philips et al. verfolgt [118], eine Erweiterung des Modells auf nicht-lineare Ventilfedern ist in [124] zu finden.
Mehr-Massen-Federmodelle basieren auf dem Gedanken die Federwindungen mit einer hohen Anzahl an Massen zu approximieren, die über Feder-Dämpfer-Elemente gekoppelt sind. Engelhardt [50] zeigt in seiner Dissertation die Anwendung und Validierung dieser Modellierungsvariante. Nachteilig ist bei dieser Modellierungsvariante die hohe Anzahl an Freiheitsgraden, wobei jedoch eine Modellierung der Kontakte zwischen den Windungen problemlos möglich ist.
Federmodelle, die auf geraden Balkenelementen basieren, versuchen die gekrümmte Windungslinie einer Ventilfeder mit einer hohen Anzahl an geraden Elementen abzubilden. Tichanek et al. [142] verwenden beispielsweise etwa 24 gerade Balkenelemente pro Windung. Die Vor- und Nachteile sind mit denen der Mehr-Massen-Modelle nahezu identisch, jedoch können durch Verwendung geeigneter Ansatzfunktionen für die geraden Balkenelemente steife Differentialgleichungen vermieden werden.
Eine kurze Zusammenfassung der Vor- und Nachteile sind in Tabelle A.1 zu finden.

Tabelle A.1: Vor- und Nachteile gebräuchlicher Ventilfedermodelle.

Typ	Vorteile	Nachteile
modale Modelle	- wenig Freiheitsgrade	- keine Kontaktmodellierung möglich
Mehr-Massen Modelle	- Kontaktmodellierung möglich	- hohe Anzahl an Freiheitsgraden - steife Differentialgleichungen
Modelle basierend auf geraden Balkenelementen	- Kontaktmodellierung möglich	- hohe Anzahl an Freiheitsgraden

Für weitere Informationen seien auch die Werke von Dreyer [47], Dreyer und Fritzen [48], Muhr [108], Schamel et al. [124], Speckens [134] und Tichanek et al. [142] genannt.

A.1.2 Federmodell

Ziel dieses Abschnittes ist es, die wesentlichen Ideen der Herleitung des Federmodells zu nennen. Für eine ausführlichere Herleitung der Bewegungsgleichungen wird auf die Arbeit von Huber und Ulbrich [83] verwiesen.

Ausgangspunkt für die Herleitung der Bewegungsgleichungen der Feder ist die kinematische Beschreibung der Federgeometrie.

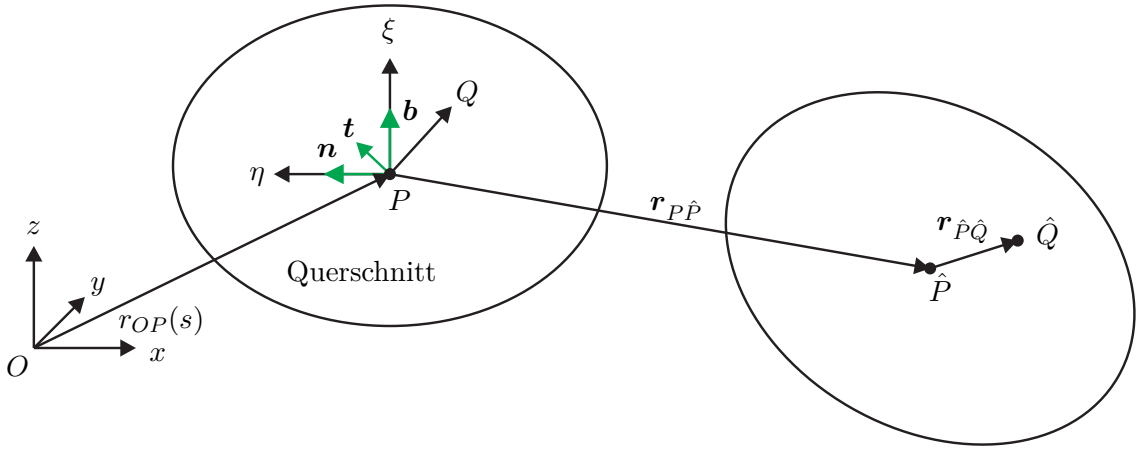


Abbildung A.2: Kinematische Beschreibung der Feder.

Die Wicklungslinie wird mittels des Laufparameters s (Drahtlänge) beschrieben. Die Darstellung eines beliebigen Punktes Q auf dem Querschnitt erfolgt im mitlaufenden *Frenet-Seret*-Koordinatensystem [135], das aus einer normalen $\boldsymbol{n}(s)$, tangentialen $\boldsymbol{t}(s)$ und binormalen $\boldsymbol{b}(s)$ Richtung besteht (siehe Abbildung A.2). Für die Herleitung der Bewegungsgleichungen wird angenommen, dass sich jeder Verformungszustand und somit jeder Verzerrungszustand aus

- der Translation des Mittelpunktes P des Querschnitts,
- der Rotation des Querschnitts
- und der Wölbung des Querschnitts

zusammensetzt.

Um aus einem gegebenen Verformungs- bzw. Verzerrungszustand (Verzerrung ϵ) die zugehörigen Kräfte und Momente berechnen zu können, muss ein entsprechendes Materialgesetz ausgewählt werden. Es wird ein reduziertes isotropes HOOKEsches Materialgesetz mit dem E-Modul E und der Querkontraktionszahl ν verwendet.

$$\sigma_{tt} = E\epsilon_{tt}\,, \qquad \tau_{tn} = \frac{E}{1+\nu}\epsilon_{tn}\,, \qquad \tau_{tb} = \frac{E}{1+\nu}\epsilon_{tb}\,. \tag{A.1}$$

Die resultierenden Schnittkräfte und -momente können durch Integration der Spannungen σ und τ über den Querschnitt berechnet werden. Dabei wird von einem elliptischen Querschnitt mit den Halbachsen a und b und einer Querschnittsfläche A ausgegangen.

Durch Anwendung von Impuls- und Drallsatz auf ein infinitesimales Stück des Querschnitts ds können die Bewegungsgleichungen der Feder hergeleitet werden, wobei folgende Annahmen getroffen werden [135]:

1. Der Einfluss der Krümmung auf Kräfte und Momente ist vernachlässigbar.
2. Äußere Lasten sind konstant im betrachteten infinitesimalen Abschnitt ds.

3. Die Bewegung aufgrund der Wölbung des Querschnitts ist vernachlässigbar.
4. Der Radius des Querschnitts ist klein gegenüber dem Radius der Wicklungslinie.
5. Die Steigung der Wicklungslinie ist gering.

Für die räumliche Bewegung der Feder ergeben sich somit sechs gekoppelte partielle Differentialgleichungen [83].
Da ein wesentliches Ziel des Federmodells eine geringe Rechenzeit darstellt und für die Mehrkörpersimulation von Ventiltrieben hauptsächlich der Freiheitsgrad entlang der Federachse von Interesse ist, wird das Federmodell unter der Annahme, dass der Hauptteil der elastischen Energie in die Torsion des Querschnitts entfällt, auf einen Freiheitsgrad reduziert. Dabei ergibt sich die Gleichung (A.2), die auch unter dem Begriff *1-dimensionale Wellengleichung* bekannt ist.

$$\rho A \frac{\mathrm{d}^2 u_z}{\mathrm{d}\, t^2} - \frac{GJ}{R^2} \frac{\mathrm{d}^2 u_z}{\mathrm{d}\, s^2} = f_z \tag{A.2}$$

Darin beschreibt u_z die Stauchung der Ventilfeder entlang ihrer Wicklungsachse, f_z von außen wirkende Kräfte, ρ die Dichte, J die Torsionskonstante, R den Windungsradius und G den Schubmodul (siehe [135]).

A.1.3 Diskretisierung der partiellen Differentialgleichung

Um das Ventilfedermodell in einer Mehrkörpersimulationsumgebung wie MBSim implementieren zu können, muss die partielle Differentialgleichung (A.2) in eine Form analog zu

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{D}\dot{\boldsymbol{q}} + \boldsymbol{K}\boldsymbol{q} = \boldsymbol{h} \tag{A.3a}$$

überführt werden. Dies geschieht durch Anwendung der Methode der Finiten-Elemente [30]. Für die Implementierung werden Lagrange Polynome 1. Ordnung, Lagrange Polynome 2. Ordnung, Lagrange Polynome 5. Ordnung und Hermitesche Polynome 3. Ordnung verwendet (siehe Tabelle A.2). Detaillierte Herleitungen und Validierungen der einzelnen Modellierungsstufen sind den Arbeiten von Staahl [135], Huber et al. [83] und Clauberg [38] zu finden.

Tabelle A.2: FE-Ansatzfunktionen.

Elementtyp	Knoten pro Element	Freiheitsgrade pro Knoten
Lagrange 1. Ordnung	2	1
Lagrange 2. Ordnung	3	1
Lagrange 5. Ordnung	6	1
Hermitesch 3. Ordnung	2	2

Die sich nach der Diskretisierung ergebende Form der Bewegungsgleichung

$$\underbrace{\int_k^l \rho A \boldsymbol{N}\boldsymbol{N}^T \, ds}_{\boldsymbol{M}} \cdot \frac{\mathrm{d}^2 \boldsymbol{q}}{\mathrm{d}\, t^2} + \underbrace{\int_k^l \frac{GJ}{R^2} \frac{\mathrm{d}\, \boldsymbol{N}}{\mathrm{d}\, s} \frac{\mathrm{d}\, \boldsymbol{N}^T}{\mathrm{d}\, s} \, ds}_{\boldsymbol{K}} \cdot \boldsymbol{q} = \underbrace{[\boldsymbol{N} \cdot T_z]_k^l + \int_k^l \boldsymbol{N} \cdot f_z \, ds}_{\boldsymbol{F}} \tag{A.4}$$

wurde in die Mehrkörpersimulationsumgebung MBSim implementiert. In Gleichung (A.4) bezeichnet $\boldsymbol{N}$ die FE-Ansatzfunktionen, T_z gegebenenfalls auftretende Volumenkräfte und $\boldsymbol{q}$ die FE-Freiheitsgrade.

A.1.4 Dämpfungsmodellierung

Da Gleichung (A.4) noch keine Dämpfungsanteile beinhaltet, wird von Huber et al. [83] ein geschwindigkeitsproportionaler Dämpfungsterm nach Gleichung (A.5) vorgeschlagen.

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{D}\dot{\boldsymbol{q}} + \boldsymbol{K}\boldsymbol{q} = \boldsymbol{F} \qquad \text{mit } \boldsymbol{D} = \boldsymbol{D}_1 + \boldsymbol{D}_2 + \boldsymbol{D}_3 \tag{A.5}$$

Die Dämpfungsmatrix $\boldsymbol{D}$ wird in drei Anteile aufgeteilt. Der Anteil $\boldsymbol{D}_1$ beschreibt die Materialdämpfung, die als Rayleigh-Dämpfung[2] (A.6) [145] modelliert ist. Der Anteil $\boldsymbol{D}_2$ beschreibt einen Dämpfungsanteil, der diskret auf die einzelnen FE-Freiheitsgrade des Modells wirkt (A.7) und der Anteil $\boldsymbol{D}_3$ modelliert Dämpfung, die aufgrund der Reibung der Windungen untereinander auftritt (A.8). Da sich die in Kontakt befindlichen Windungen in jedem Zeitschritt ändern, muss der Dämpfungsanteil $\boldsymbol{D}_3$ in jedem Zeitschritt neu berechnet werden.

$$\boldsymbol{D}_1 = d_M \cdot \boldsymbol{M} + d_K \cdot \boldsymbol{K} \tag{A.6}$$

$$\boldsymbol{D}_2 = \mathrm{diag}(d_{\mathrm{dis}}) \tag{A.7}$$

$$\boldsymbol{D}_3 = \begin{pmatrix} d_{\mathrm{cont}} & 0 & \dots & \dots & \dots & 0 \\ 0 & \ddots & & & & \vdots \\ \vdots & & d_{\mathrm{cont}} & & & \vdots \\ \vdots & & & 0 & & \vdots \\ \vdots & & & & \ddots & \vdots \\ 0 & \dots & & & \dots & 0 \end{pmatrix} \tag{A.8}$$

2 Unter Rayleigh-Dämpfung versteht man die Zusammensetzung der geschwindigkeitsproportionalen Dämpfung $\boldsymbol{D}$ aus einem Anteil, der proportional zur Massenmatrix $\boldsymbol{M}$ ist und einem Anteil, der proportional zur Steifigkeitsmatrix $\boldsymbol{K}$ ist.

Zusätzlich wird Energie in den mengenwertig modellierten Windungskontakten[3] dissipiert.

A.1.5 Kontaktmodellierung

Das Schema der Kontaktmodellierung ist in Abbildung A.3 gezeigt.

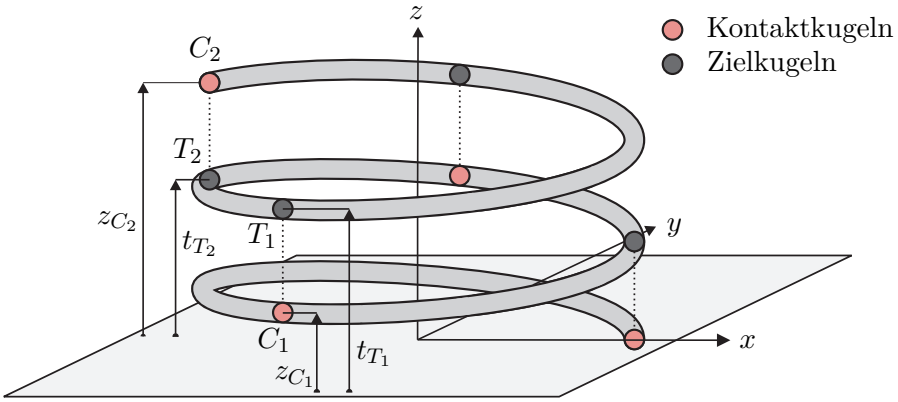


Abbildung A.3: Kontaktmodellierung.

Zur Modellierung der Kontakte kommen Kugel/Kugel-Kontakte zur Anwendung (siehe auch Abschnitt 2.4). Es kann eine beliebige Anzahl an Kugel-Paaren auf der Windungslinie verteilt werden. Jedes Kugel-Paar besteht dabei aus einer sogenannten *Kontaktkugel* und einer sogenannten *Zielkugel*. Die erste *Kontaktkugel* wird am unteren Ende der Feder positioniert, die letzte Kontaktkugel wird am oberen Ende der Feder positioniert. Die verbleibenden Kugel-Paare werden gleichmäßig auf der Windung verteilt.

Um die Effizienz des Federmodells beizubehalten, jedoch trotzdem auch konische und bienenkorbförmige Ventilfedern modellieren und simulieren zu können, wurden von Clauberg et al. [39] entsprechende Modellerweiterungen vorgeschlagen.
Diese Erweiterungen sehen vor, dass weiterhin Kugelgeometrien für die Kontakte verwendet werden, jedoch der Radius k_K der beteiligten Kugelgeometrien in Abhängigkeit des Ventilfederradius und des elliptischen Querschnitts berechnet wird.

Auf der linken Seite von Abbildung A.4 sind beispielsweise die Geometriedaten eines Kugel/Kugel Kontakts einer Ventilfeder mit nicht-konstantem Radius und elliptischen Querschnitt gezeigt. Der notwendige Radius k_r zur Modellierung dieser Konfiguration als Kugel/Kugel Kontaktgeometrien mit dem vorhandenen Federmodell kann nach Gleichung (A.9) bis (A.11) berechnet werden.

$$x = \frac{1}{2}(r_u - r_o) \tag{A.9}$$

3 Die Windungskontakte werden als vollplastische Stöße mit dem Stoßkoeffizienten $\varepsilon = 0$ modelliert.

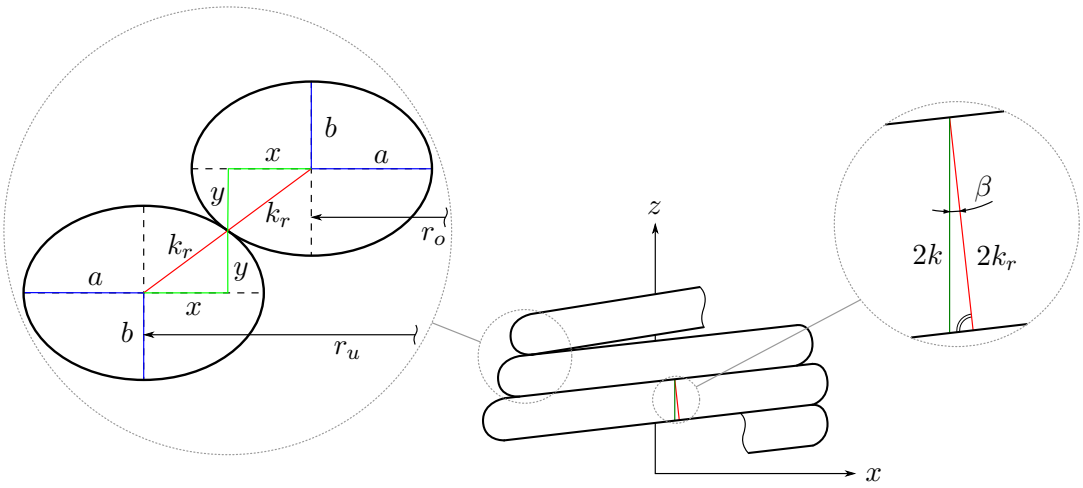


Abbildung A.4: Kontaktmodellierung.

$$y = b\sqrt{1 - \left(\frac{x}{a}\right)^2} \tag{A.10}$$

$$k_r = \sqrt{x^2 + y^2} \tag{A.11}$$

Der Einfluss der Steigung auf die Kontaktmodellierung ist auf der rechten Seite von Abbildung A.4 gezeigt. Unter der Annahme, dass die Steigungswinkel β in allen geschlossenen Kontakten gleich sind, kann der endgültige Radius k_K der Kontaktkugeln nach Gleichung (A.12) und (A.13) angenähert werden.

$$\beta \approx \arctan\left(\frac{b}{\pi R_0}\right) \tag{A.12}$$

$$k_K \approx k_r \frac{1}{\cos\beta} \tag{A.13}$$

R_0 bezeichnet den Federradius am unteren Ende.

A.1.6 Statische und dynamische Validierung

Zuletzt soll in diesem Abschnitt die Genauigkeit des Federmodells gezeigt werden. Huber et al. [83] haben bereits ausführliche Validierungen des Federmodells anhand einer zylindrisch asymmetrischen Feder gezeigt. Im Folgenden werden die Ergebnisse für die Modellierung und Simulation konischer und bienenkorbförmiger Ventilfedern gezeigt.
Die Validierung gliedert sich in die statische Validierung und die dynamischen Validierung im Frequenz und Zeitbereich.

Die Materialparameter (E-Modul E, Querkontraktionszahl ν, Dichte ρ, G-Modul G) wurden soweit vorhanden vom Hersteller übernommen und ansonsten im für

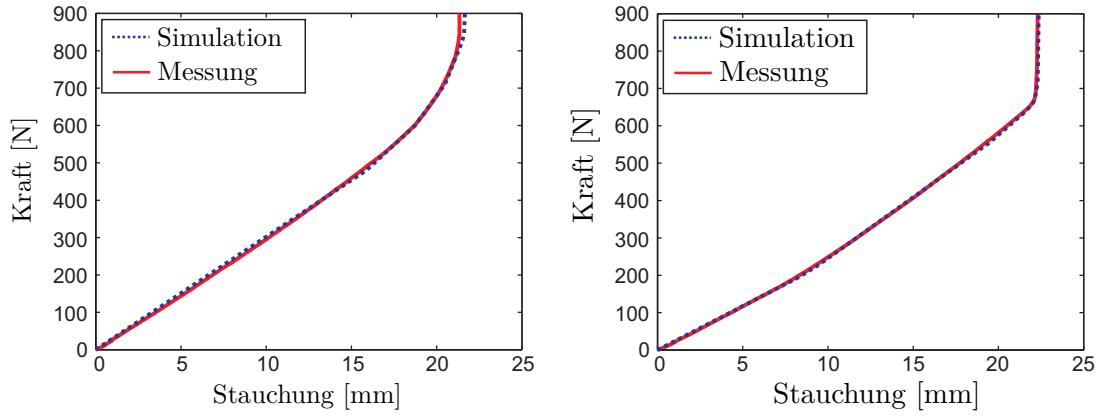
Ventilfedern üblichen Bereich gewählt. Die geometrischen Daten der Federn wurden mittels spezieller Messsysteme vermessen.

Für die Simulation der konischen Federn wurden 7 Elemente (HERMITEsche Polynome 3. Grades) und 7 Kugel/Kugel-Kontaktpaarungen verwendet, für die Simulation der bienenkorbförmigen Feder wurden ebenfalls 7 Elemente (HERMITEsche Polynome 3. Grades) und 8 Kugel/Kugel-Kontaktpaarungen verwendet (siehe Tabelle A.3).

Tabelle A.3: Typ der Elemente, Anzahl der Elemente und Anzahl der Kugel/Kugel-Kontaktpaarungen.

Federtyp	Elementtyp	Anzahl Elemente	Anzahl Kontaktpaarungen
konisch	HERMITE 3. Ord.	7	8
bienenkorb	HERMITE 3. Ord.	7	8

Für die statische Validierung[4] wird die Feder experimentell bis zur Blocklänge gestaucht und die Kraft in Abhängigkeit der Stauchung aufgezeichnet.
Die statische Validierung ist in Abbildung A.5 gezeigt. Das Simulationsmodell kann die progressive Kennlinie der realen Feder sehr gut nachbilden. Die Knicke in den simulierten Kurven sind diejenigen Zeitpunkte, zu denen Windungskontakte schließen. Schließt eine Kontaktpaarung, so wird die effektive Länge der Feder verkürzt, was zu einer höheren Steifigkeit der Feder führt. In Abbildung A.5(a) ist zu erkennen, dass das Simulationsmodell etwas später auf Blocklänge geht als die reale Feder. Dies hat in der Anwendung des Simulationsmodells jedoch keine Auswirkungen, da die Feder im realen Betrieb niemals auf Blocklänge gestaucht wird.



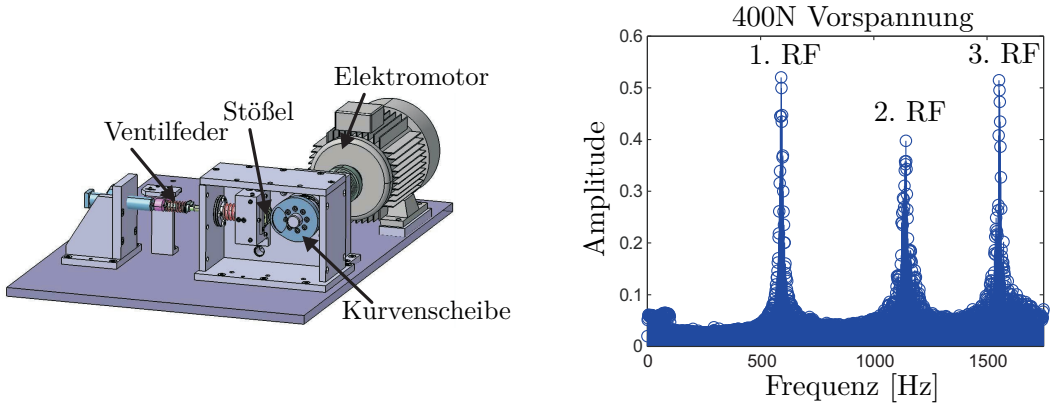
(a) Konische Feder. (b) Bienenkorbfeder.

Abbildung A.5: Statische Validierung einer konischen und bienenkorbförmigen Feder.

Die dynamische Validierung unterteilt sich in die Validierung im Frequenzbereich und die Validierung im Zeitbereich.

4 Unter statischer Validierung wird an dieser Stelle das Weg-Kraft Kennfeld der Feder verstanden.

Für die Validierung im Frequenzbereich werden die Resonanzstellen der Feder in Messung und Simulation bestimmt und verglichen. Zur dynamischen Vermessung der Federn wurde am Lehrstuhl für Angewandte Mechanik ein Prüfstand konzipiert und aufgebaut (Abbildung A.6(a)), mittels dem man nahezu beliebige Anregebewegungen auf die Federn aufprägen kann [83]. Der Prüfstand basiert dazu auf dem Prinzip eines Kurvengetriebes. Für die Bestimmung der Resonanzfrequenzen wird eine harmonische Anregebewegung mit 0.005 mm Hub auf die Feder aufgebracht und die Frequenz in Form eines quasi-statischen Hochlaufs von 0 bis 2000 Hz variiert. Aus den gemessenen Zeitverläufen der Kraft am unteren eingespannten Federende können mittels einer FFT-Analyse die ersten Resonanzfrequenzen bestimmt werden. Der Hochlauf wird zur Bestimmung der gesamten Charakteristik der Feder unter verschiedenen Vorspannungen der Feder durchgeführt. Abbildung A.6(b) zeigt exemplarisch die FFT-Analyse eines Hochlaufs.



(a) Ventilfederprüfstand.

(b) FFT-Analyse.

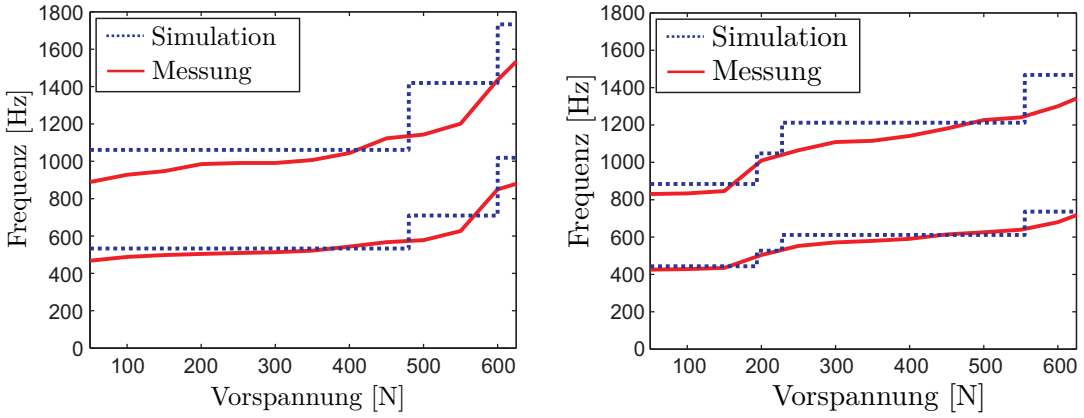
Abbildung A.6: Ventilfederprüfstand und beispielhafte FFT-Analyse.

Abbildung A.7 zeigt die Validierung der konischen und der bienenkorbförmigen Feder im Frequenzbereich.

Auffällig sind dabei die treppenförmigen Verläufe der simulierten Resonanzstellen. Die Ursache der treppenförmigen Verläufe liegt darin, dass sich die Steifigkeit und somit die Resonanzfrequenzen des Federmodells nur dann ändern können, wenn sich Kontakte schließen oder öffnen, da es sich bei dem Balkenmodell um ein lineares Modell handelt. Dieser Vorgang geschieht nur an diskreten Stellen, die sich als Treppen in den Verläufen widerspiegeln. Trotzdem folgen die simulierten Verläufe sehr schön den gemessenen Verläufen.

Die höchste Aussagekraft haben dynamische Validierungen im Zeitbereich. Dazu wird auf die Feder eine Anregebewegung ähnlich der in einem realen Verbrennungsmotor aufgeprägt. Die gemessene Kraft am unteren Federende[5] dient daraufhin als Validierungsgröße. Dazu wird die gemessene Anregebewegung als kinematische Randbedingung auf das Federmodell gegeben. Um einen konsistenten Verlauf von

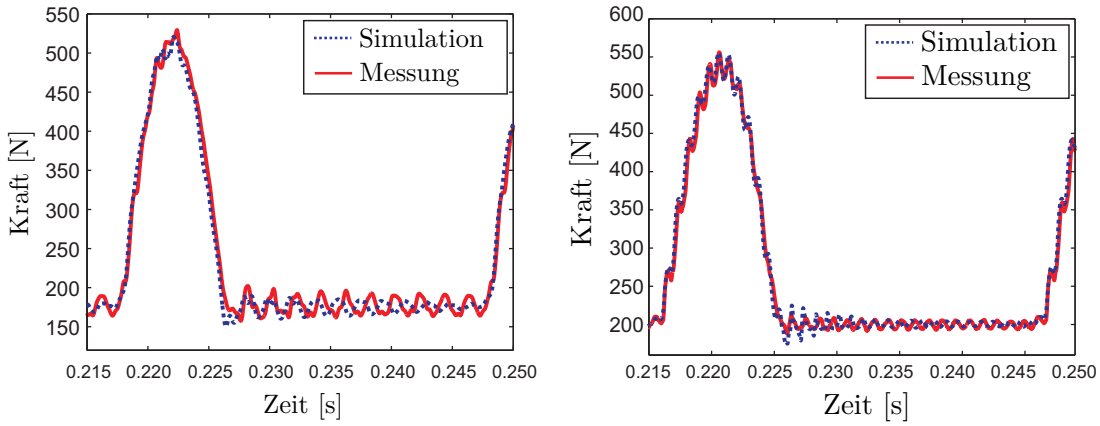
5 Das untere Federende bezeichnet das dem Motorblock zugewandte Ende.



(a) Konische Feder. (b) Bienenkorbfeder.

Abbildung A.7: Dynamische Validierung einer konischen und bienenkorbförmigen Feder im Frequenzbereich.

Anregeweg, -geschwindigkeit und -beschleunigung zu erhalten, kommt ein KALMAN-Filter [82] zum Einsatz.
Die Ergebnisse der dynamischen Validierung der konischen und bienenkorbförmigen Feder im Zeitbereich sind in Abbildung A.8 zu sehen.



(a) Konische Feder. (b) Bienenkorbfeder.

Abbildung A.8: Dynamische Validierung einer konischen und bienenkorbförmigen Feder im Zeitbereich.

Zu erkennen sind die sehr gut übereinstimmenden Kraftniveaus von Simulation und Messung. Sowohl in der Ruhelage ($\sim$ 150-200 N) als auch bei höchster Stauchung ($\sim$ 500-550 N) stimmen die Kräfte sehr gut überein. Ebenso zeigen die Frequenzen bei hohen Vorspannungen eine sehr gute Übereinstimmung. In Abbildung A.8(a) ist zu erkennen, dass die Frequenz zwischen Messung und Simulation auf niedrigem Kraftniveau geringe Unterschiede aufweist. Dies lässt sich anhand von Abbildung A.7(a) erklären, an der sich ablesen lässt, dass die simulierte Resonanzfrequenz bei

etwa 100-150 N Vorspannung etwas oberhalb der Gemessenen liegt. Dies spiegelt sich auch in Abbildung A.8(a) wieder, in der die simulierte Kraft in der Ruhephase geringfügig schneller schwingt als die Gemessene. Insgesamt können diese Ergebnisse jedoch problemlos für die Simulation von Gesamtventiltrieben herangezogen werden.

A.2 Modellierungsgrößen der Anwendungsbeispiele

A.2.1 Kugeln in Schale

Tabelle A.4: Geometrie- und Simulationsdaten des Beispiels *Kugeln in Schale*.

Variable	Wert	Beschreibung
g	$9.81\,\mathrm{m/s^2}$	Erdbeschleunigung
r	$0.1\,\mathrm{m}$	Radius der Kugeln
a	$2.3\,\mathrm{m}$	Abstand der seitlichen Begrenzungsflächen (links/rechts; vorne/hinten)
b	$1.5\,\mathrm{m}$	Tiefe der Schale
m	$24.5\,\mathrm{kg}$	Masse einer Kugel
n	112	Anzahl an Kugeln
ε_{KS}	0.5	Stoßkoeffizient zwischen Schale und Kugeln
ε_{KK}	0.5	Stoßkoeffizient zwischen Kugeln

A.2.2 Spielzeugspecht an flexibler Stange

Tabelle A.5: Geometrie- und Simulationsdaten des Beispiels *Spielzeugspecht*.

Variable	Wert	Beschreibung
g	$9.81\,\mathrm{m/s^2}$	Erdbeschleunigung
r	$5\,\mathrm{mm}$	Radius der flexiblen Stange
c_F	$0.5\,\mathrm{Nm/rad}$	Steifigkeit der Drehfeder zwischen Specht und Muffe
μ	0.2	Reibkoeffizient zwischen Muffe/Schnabel und Stange
ε	0.0	Stoßkoeffizient zwischen Muffe/Schnabel und Stange
m_M	$10\,\mathrm{g}$	Masse der Muffe
J_M	$5\cdot10^{-6}\,\mathrm{kgm^2}$	Drehträgheit der Muffe
D	$11.4\,\mathrm{mm}$	Innendurchmesser der Muffe
h	$2.85\,\mathrm{mm}$	Höhe der Muffe
a	$27\,\mathrm{mm}$	Abstand des Gelenkpunktes zum Muffen-Bezugspunkt
m_S	$100\,\mathrm{g}$	Masse des Spechts
J_S	$4\cdot10^{-4}\,\mathrm{kgm^2}$	Drehträgheit des Spechts
b_1	$54\,\mathrm{mm}$	horizontaler Abstand des Schwerpunktes zum Gelenkpunkt
c_1	$70\,\mathrm{mm}$	horizontaler Abstand des Schnabels zum Schwerpunkt
c_2	$50\,\mathrm{mm}$	vertikaler Abstand des Schnabels zum Schwerpunkt
n_{FE}	5	Anzahl FE-Elemente für den flexiblen Balken

A.2.3 Perlenkette

Tabelle A.6: Geometrie- und Simulationsdaten des Beispiels *Perlenkette*.

Variable	Wert	Beschreibung
g	$9.81\,\mathrm{m/s^2}$	Erdbeschleunigung
a	$0.03\,\mathrm{m}$	Seitenlänge der gleitenden Elemente (alle 3 Dimensionen)
R	$0.399\,\mathrm{m}$	Radius des flexiblen Balkens in Ausgangsform
E	$2.5 \cdot 10^9\,\mathrm{N/m^2}$	E-Modul des flexiblen Balkens
ρ	$2.5 \cdot 10^3\,\mathrm{kg/m^3}$	Dichte der Körper
n	25	Anzahl gleitender Elemente
A	$400\,\mathrm{mm}^2$	Querschnittfläche des flexiblen Balkens
L	$1.0\,\mathrm{m}$	Länge des flexiblen Balkens
m	$0.0675\,\mathrm{kg}$	Masse eines gleitenden Elements
ε	1.0	Stoßkoeffizient zwischen gleitenden Elementen
n_{FE}	20	Anzahl Finite-Elemente für flexiblen Balken

A.2.4 Gleitende Elemente auf flexiblem Balken

Tabelle A.7: Geometrie- und Simulationsdaten des Beispiels *Gleitende Elemente auf flexiblem Balken*.

Variable	Wert	Beschreibung
g	$9.81\,\mathrm{m/s^2}$	Erdbeschleunigung
a	$0.0211\,\mathrm{m}$	Breite der gleitenden Elemente
h	$0.15\,\mathrm{m}$	Höhe der gleitenden Elemente (quadratisch)
L	$1.5\,\mathrm{m}$	Länge des Balkens
E	$2.1 \cdot 10^{11}\,\mathrm{N/m^2}$	E-Modul des flexiblen Balkens
ρ	$7.85 \cdot 10^2\,\mathrm{kg/m^3}$	Dichte der Körper
n	80	Anzahl gleitender Elemente
b	$0.1\,\mathrm{m}$	Seitenlängen der Querschnittfläche des flexiblen Balkens
A	$0.01\,\mathrm{m}^2$	Querschnittfläche des flexiblen Balkens
m	$2.0\,\mathrm{kg}$	Masse eines gleitenden Elements
$\varepsilon_{\mathrm{EB}}$	0.7	Stoßkoeffizient zwischen gleitenden Elementen und flexiblem Balken
$\varepsilon_{\mathrm{EE}}$	0.0	Stoßkoeffizient zwischen gleitenden Elementen
n_{FE}	4	Anzahl Finite-Elemente für flexiblen Balken

A.2.5 Ventiltriebsdynamik

Tabelle A.8: Geometrie- und Simulationsdaten des Beispiels *Ventiltriebsdynamik*.

Variable	Wert	Beschreibung
E	$2.1 \cdot 10^{11}\,\mathrm{N/m^2}$	E-Modul der flexiblen Nockenwelle
G	$0.81 \cdot 10^{11}\,\mathrm{N/m^2}$	G-Modul der flexiblen Nockenwelle
ρ	$7.85 \cdot 10^{3}\,\mathrm{kg/m^3}$	Dichte der Körper
A	$3.14 \cdot 10^{-4}\,\mathrm{m^2}$	Querschnittfläche der Nockenwelle
L	$0.42\,\mathrm{m}$	Länge der Nockenwelle
n_{FE}	5	Anzahl FE-Elemente für die flexible Nockenwelle
r	$10.0\,\mathrm{mm}$	Radius der Nockenwelle

A.2.6 Rotordynamik

Tabelle A.9: Geometrie- und Simulationsdaten des Beispiels *Rotordynamik*.

Variable	Wert	Beschreibung
g	$9.81\,\mathrm{m/s^2}$	Erdbeschleunigung
l_1	$76\,\mathrm{mm}$	Abstand zur Scheibe A
l_2	$203\,\mathrm{mm}$	Abstand zur Scheibe B
l_3	$295\,\mathrm{mm}$	Abstand zur Scheibe C
l_4	$400\,\mathrm{mm}$	Abstand zur Scheibe D
l_5	$590\,\mathrm{mm}$	Abstand zur Lagerung B
l_6	$740\,\mathrm{mm}$	Abstand zum Motor
m_A	$2.98\,\mathrm{kg}$	Masse Scheibe A
m_B	$2.98\,\mathrm{kg}$	Masse Scheibe B
m_C	$0.43\,\mathrm{kg}$	Masse Scheibe C
m_D	$3.16\,\mathrm{kg}$	Masse Scheibe D
E	$2.1 \cdot 10^{11}\,\mathrm{N/m^2}$	E-Modul des Rotors
G	$0.81 \cdot 10^{11}\,\mathrm{N/m^2}$	G-Modul des Rotors
ρ	$7.85 \cdot 10^{3}\,\mathrm{kg/m^3}$	Dichte der Körper
A	$4.91 \cdot 10^{-4}\,\mathrm{m^2}$	Querschnittfläche des Rotors
L	$0.64\,\mathrm{m}$	Länge des Rotors
n_{FE}	5	Anzahl FE-Elemente für den flexiblen Rotor
r	$12.5\,\mathrm{mm}$	Radius des Rotors
c_L	$6 \cdot 10^{6}\,\mathrm{N/m}$	Steifigkeit in den Lagerungen
d_L	$10\,\mathrm{Ns/m}$	Dämpfung in den Lagerungen
c_K	$75 \cdot 10^{3}\,\mathrm{N/m}$	Steifigkeit der Kupplung

Literaturverzeichnis

[1] *ATLAS - Automatically Tuned Linear Algebra Software.* `http://math-atlas.sourceforge.net/`. Version: Juli 2012

[2] *BLAS - Basic Linear Algebra Subprograms.* `http://www.netlib.org/blas/`. Version: Juli 2012

[3] *fmatvec- C++-library for high performance matrix-vector computations based on Blas/Lapack and Atlas.* `http://code.google.com/p/fmatvec/`. Version: Juli 2012

[4] *GCC - the GNU Compiler Collection.* `http://gcc.gnu.org/`. Version: Juli 2012

[5] *HDF5 is a data model, library, and file format for storing and managing data.* `http://www.hdfgroup.org/HDF5/`. Version: Juli 2012

[6] *HDF5Serie - Template based high level C++ interface/wrapper to HDF5 (http://www.hdfgroup.org) especially/just for reading and writing series of data.* `http://code.google.com/p/hdf5serie/`. Version: Juli 2012

[7] *Helgrind - a thread error detector.* `http://valgrind.org/docs/manual/hg-manual.html`. Version: Juli 2012

[8] *Intel Compilers.* `http://software.intel.com/en-us/articles/intel-compilers/`. Version: August 2012

[9] *Intel Inspector - Detecting memory and threading defects.* `http://software.intel.com/en-us/articles/intel-inspector-xe/`. Version: Juli 2012

[10] *IntelMKL - Intel Math Kernel Library.* `http://software.intel.com/en-us/articles/intel-mkl/`. Version: Juli 2012

[11] *LAPACK - Linear Algebra Package.* `http://www.netlib.org/lapack/`. Version: Juli 2012

[12] *Mathworks.* `http://www.mathworks.de/`. Version: Juli 2012

[13] *MBSim - Multibody Simulation Environment MBSim.* `http://code.google.com/p/mbsim-env/`. Version: Juli 2012

[14] *MPI - The Message Passing Interface standard.* `http://www.mcs.anl.gov/research/projects/mpi/`. Version: Juli 2012

[15] *OpenMBV - A visualisation tool, specially for three dimensional multi body simulations.* `http://code.google.com/p/openmbv/`. Version: Juli 2012

[16] *OpenMP - The OpenMP API specification for parallel programming.* `http://openmp.org/wp/`. Version: Juli 2012

[17] *OpenSuse Linux.* `http://de.opensuse.org/`. Version: Juli 2012

[18] *Zoltan - Parallel Partitioning, Load Balancing and Data-Management Services.* `http://www.cs.sandia.gov/Zoltan/`. Version: Juli 2012

[19] Acary, V. ; Brogliato, B.: *Applications on Mechanics and Electronics.* Bd. 35: *Numerical Methods for Nonsmooth Dynamical Systems.* Berlin : Springer, 2008. – ISBN 978–3–540–75391–9

[20] Acary, V. ; Pérignon, F. u. a.: An introduction to Siconos. (2007)

[21] Ahn, Y. ; Kim, W.J. ; Chung, K.S. ; Kim, S.H. ; Kim, H.S. ; Han, T.H.: A novel load balancing method for multi-core with non-uniform memory architecture. In: *International SoC Design Conference (ISOCC)* IEEE, 2010, S. 412–415

[22] Amin-Javaheri, M. ; Orin, D.: A systolic architecture for computation of the manipulator inertia matrix. In: *Proceedings: IEEE International Conference on Robotics and Automation.* Bd. 4 IEEE, 1987, S. 647–653

[23] Anderson, K.S. ; Duan, S.: Highly parallelizable low-order dynamics simulation algorithm for multi-rigid-body systems. In: *Journal of Guidance, Control, and Dynamics* 23 (2000), Nr. 2, S. 355–364

[24] Anderson, K.S. ; Oghbaei, M.: A State–Time Formulation for Dynamic Systems Simulation Using Massively Parallel Computing Resources. In: *Nonlinear Dynamics* 39 (2005), Nr. 3, S. 305–318

[25] Artis, P.H.: Quantifying MultiProcessor Overheads. In: *Proceedings of CMG '91*, 1991, S. 363–365

[26] Basshuyssen, R. ; Schäfer, F.: *Handbuch Verbrennungsmotoren.* 5. Aufl. Wiesbaden : GWV Fachverlag GmbH, 2010. – ISBN 978–3–8348–0699–4

[27] Berard, S. ; Trinkle, J. ; Nguyen, B. ; Roghani, B. ; Fink, J. ; Kumar, V.: daVinci Code: A multi-model simulation and analysis tool for multi-body systems. In: *IEEE International Conference on Robotics and Automation.* IEEE, 2007, S. 2588–2593

[28] Binder, E.E. ; Herzog, J.H.: Distributed computer architecture and fast parallel algorithms in real-time robot control. In: *IEEE Transactions on Systems, Man and Cybernetics* 16 (1986), Nr. 4, S. 543–549

[29] Borchsenius, F.: *Simulation ölhydraulischer Systeme.* Munich : VDI Verlag, 2002. – 152 S. – ISBN 978–3–18–500508–4

[30] Braess, D.: *Finite Elemente.* Springer, 2003. – ISBN 9783540001225

[31] Bremer, H. ; Pfeiffer, F.: *Elastische Mehrkörpersysteme.* Stuttgart : Teubner, 1992. – ISBN 3–519–02374–1

[32] Brogliato, B.: *Nonsmooth impact mechanics: models, dynamics, and control.* London : Springer London, 1996. – ISBN 3–540–76079–2

[33] Brogliato, B. ; Ten Dam, A.A. ; Paoli, L. ; Genot, F. ; Abadie, M.: Numerical simulation of finite dimensional multibody nonsmooth mechanical systems. In: *Applied Mechanics Reviews* 55 (2002), S. 107–150. – ISSN 0003–6900

[34] Butcher, J.C.: On The Implementation of Implicit Runge-Kutta Methods. In: *Bit Numerical Mathematics* 16 (1976), Nr. 3, S. 237–240. http://dx.doi.org/10.1007/BF01932265. – DOI 10.1007/BF01932265

[35] Cebulla, T. ; Schindler, T. ; Ulbrich, H. ; Velde, A. van d. ; H., Pijpers: Räumliche Dynamik und Simulation von Schubglieder-CVTs. In: *VDI-Fachkonferenz - Umschlingungsgetriebe 2010*. Sindelfingen, Germany, December 2010

[36] Chapman, B. ; Jost, G. ; Van Der Pas, R.: *Using OpenMP: portable shared memory parallel programming*. Bd. 10. The MIT Press, 2007

[37] Clauberg, J.: *Auslegung und Konstruktion eines Ventilfederprüfstandes*. München : Semesterarbeit, AMM TUM, 2008

[38] Clauberg, J.: *Implementierung und Abgleich eines effizienten kontinuierlichen Federmodells mit nichtglatter Kontaktmodellierung für die Ventiltriebsimulation*. München, TU München, Diplomarbeit, 2009

[39] Clauberg, J. ; Huber, B. ; Ulbrich, H.: Simulation and Validation of Valve Springs in Valve Train Simulations. In: *Proceedings of the Eighth International Conference on Engineering Computational Technology*. Dubrovnik, Croatia, September 2012

[40] Clauberg, J. ; Huber, R. ; Ulbrich, H.: Modellierung und Abgleich eines effizienten Federmodells für die Mehrkörpersimulation auf Basis eines gekrümmten Balkens. In: *VDI: Schwingungsanalyse und Identifikation*. Germany, Leonberg : VDI, 2010

[41] Clauberg, J. ; Schneider, M. ; Ulbrich, H.: Simulation of a Non-Smooth Continuous System. In: *Vibration Problems ICOVP 2011*. Berlin : Springer, 2011 (Proceedings in Physics 139). – ISBN 978–94–007–2068–8

[42] Clauberg, J. ; Ulbrich, H.: An Adaptive Internal Parallelization Method for Multibody Simulations. In: *Proceedings of the 12th Pan-American Congress of Applied Mechanics - PACAM XII*. Port of Spain, Trinidad and Tobago, January 2012 (Proceedings of PACAM XII)

[43] Critchley, J.H. ; Anderson, K.S.: A parallel logarithmic order algorithm for general multibody system dynamics. In: *Multibody System Dynamics* 12 (2004), Nr. 1, S. 75–93

[44] Deuflhard, F. P. und B. P. und Bornemann: *Numerische Mathematik 2: Gewöhnliche Differentialgleichungen*. 3. Aufl. de Gruyter, 2008. – 530 S. – ISBN 3110203561

[45] Dick, C. ; Georgii, J. ; Westermann, R.: A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. In: *Simulation Modelling Practice and Theory* 19 (2011), S. 801–816

[46] Dick, C. ; Georgii, J. ; Westermann, R.: A Hexahedral Multigrid Approach for Simulating Cuts in Deformable Objects. In: *IEEE Transactions on Visualization and Computer Graphics*, 2012

[47] Dreyer, M.R.: *Untersuchung zur nichtlinearen Mechanik von Schraubendruckfedern*. Der Andere Verlag, Osnabrück, 2004. – ISBN 3–89959–161–5

[48] Dreyer, M.R. ; Fritzen, C.P.: Numerical and Experimental Investigations of the Dynamics of Valve Trains in Combustion Engines. In: *Proceedings ISMA23: Noise and Vibration Engineering* Bd. 1. Leuven, Belgium, Sept. 1998

[49] Eichberger, A.: *Simulation von Mehrkörpersystemen auf parallelen Rechnerarchitekturen*. Düsseldorf, Diss., 1993

[50] Engelhardt, T.: *Dynamik von Steuer- und Ventiltrieben*. VDI Verlag, 2007

[51] Esefeld, B. ; Ulbrich, H.: A Hybrid Integration Scheme for Nonsmooth Mechanical Systems. In: *Multibody Dynamics 2011, Eccomas Thematic Conference*. Brussels, July 2011

[52] Featherstone, R.: A Divide-and-Conquer Articulated-Body Algorithm for Parallel O(log(n)) Calculation of Rigid-Body Dynamics. Part 1: Basic Algorithm. In: *The International Journal of Robotics Research* 18 (1999), S. 867–875. http://dx.doi.org/10.1177/02783649922066619. – DOI 10.1177/02783649922066619

[53] Featherstone, R.: A Divide-and-Conquer Articulated-Body Algorithm for Parallel O(log(n)) Calculation of Rigid-Body Dynamics. Part 2: Trees, Loops and Accuracy. In: *The International Journal of Robotics Research* 18 (1999), S. 876–892. http://dx.doi.org/10.1177/02783649922066628. – DOI 10.1177/02783649922066628

[54] Fijany, A. ; Bejczy, A.: Parallel Algorithms and Architectures for Computation of Manipulator Inverse Dynamics. In: *Fourth International Conference on Advanced Robotics.*

[55] Fijany, A. ; Bejczy, A.: Parallel algorithms and architecture for computation of manipulator forward dynamics. In: *Proceedings of the IEEE International Conference on Robotics and Automation.* IEEE, 1991, S. 1156–1162

[56] Fijany, A. ; Bejczy, A.K.: A class of parallel algorithms for computation of the manipulator inertia matrix. In: *IEEE Transactions on Robotics and Automation.* 5 (1989), Nr. 5, S. 600–615

[57] Fijany, A. ; Featherstone, R.: A new factorization of the mass matrix for optimal serial and parallel calculation of multibody dynamics. In: *Multibody System Dynamics* (2012), S. 1–19

[58] Fijany, A. ; Sharf, I. ; D'Eleuterio, G.M.T.: Parallel O (log n) algorithms for computation of manipulator forward dynamics. In: *IEEE Transactions on Robotics and Automation.* 11 (1995), Nr. 3, S. 389–400

[59] Förg, M.: *Mehrkörpersysteme mit mengenwertigen Kraftgesetzen – Theorie und Numerik*. VDI Verlag, 2008

[60] Förg, M.: *Vorlesungsskript Mehrkörpersimulation.* Version vom 12.05.2009. München : AM TU Mïchen, 2009

[61] Förg, M. ; Engelhardt, T. ; Ulbrich, H.: Analysis of Different Time-Integration Methods Applied to a Non-Smooth Industrial Problem. In: *Proceedings of ENOC Fifth EUROMECH Nonlinear Dynamics Conference*, 2005

[62] Förg, M. ; Engelhardt, T. ; Ulbrich, H.: Contacts within Valve Train Simulations: a Comparison of Models. In: *JSME Technical Journal* 1 (2006), Nr. 1

[63] Förg, M. ; Geier, T. ; Neumann, L. ; Ulbrich, H.: R-Factor Strategies for the Augemented Lagrangian Approach in Multi-Body Contact Mechanics. In: *Proceedings of III European Conference on Computational Mechanics.* Lisbon, Portugal, 5-8 June 2006

[64] Förg, M. ; Pfeiffer, F. ; Ulbrich, H.: Simulation of Granular Media by Rigid Body Dynamics. In: *Proceedings of ECCOMAS Thematic Conferences on advances in computational Multibody Dynamics.* Lisbon, Portugal, 2003

[65] Friedrich, M.: *Parallel Co-Simulation for Mechatronic Systems.* München : Verlag Dr. Hut, 2011

[66] Funk, K.: *Simulation eindimensionaler Kontinua mit Unstetigkeiten.* VDI-Verlag, 2004

[67] Gee, M. ; Ramm, E. ; Wall, W.A.: Parallel multilevel solution of nonlinear shell structures. In: *Computer methods in applied mechanics and engineering* 194 (2005), Nr. 21, S. 2513–2533

[68] Geier, T.: *Dynamics of Push Belt CVTs.* Munich : VDI-Verlag, 2007

[69] Ginzinger, L. B.: *Control of a Rubbing Rotor using an Active Auxiliary Bearing.* Munich : Verlag Dr. Hut, 2009

[70] Glocker, C.: *Set-Valued Force Laws: Dynamics of non-smooth systems.* Bd. 1. Berlin : Springer Verlag, 2001. – ISBN 3–540–41436–3

[71] Glocker, C.: Models of non-smooth switches in electrical systems. In: *International journal of circuit theory and applications* 33 (2005), Nr. 3, S. 205–234

[72] González, F. ; Luaces, A. ; Lugrís, U. ; González, M.: Non-intrusive parallelization of multibody system dynamic simulations. In: *Computational Mechanics* 44 (2009), Nr. 4, S. 493–504

[73] Gosselin, C.M.: Parallel computational algorithms for the kinematics and dynamicsof parallel manipulators. In: *IEEE International Conference on Robotics and Automation* IEEE, 1993, S. 883–888

[74] Graham, R.L.: Bounds on multiprocessing timing anomalies. In: *SIAM Journal on Applied Mathematics* 17 (1969), Nr. 2, S. 416–429

[75] Gunther, N.J.: Understanding the MP effect: Multiprocessing in pictures. In: *CMG Conference* COMPSCER MEASUREMENT GROUP INC, 1996, S. 957–968

[76] Hairer, E. ; Nørsett, S.P. ; Wanner, G.: *Solving ordinary differential equations: Nonstiff problems*. Bd. 1. 2. Aufl. Springer Verlag, 1993. – ISBN 3–540–56670–8

[77] Hairer, E. ; Nørsett, S.P. ; Wanner, G.: *Solving ordinary differential equations: Stiff and Differential-Algebraic Problems*. Bd. 2. Springer Verlag, 2002. – ISBN 3–540–60452–9

[78] Heckmann, B. ; Ginzinger, L. ; Thuemmel, T. ; Ulbrich, H.: Modellbasiertes Monitoring für Rotorsysteme. In: *Kurzfassungen SIRM 2011*. Darmstadt, Deutschland, Februar 2011

[79] Hidalgo, A.F. ; Jalon, J.G. de ; Tapia, S.: High Performance Algorithms and Implementations Using Sparse and Parallelization Techniques on MBS. In: *Multibody Dynamics 2011, ECCOMAS Thematic Conference*. Brussel, July 2011

[80] Hoffmann, S. ; Lienhart, R.: *OpenMP*. Berlin : Springer, 2008. – ISBN 978–3–540–73122–1

[81] Huber, B.: *Untersuchung des dynamischen Verhaltens unterschiedlicher Arten von Ventilfedern*. München, TU München, Diplomarbeit, 2009

[82] Huber, R.: *Dynamics of Variable Valve Trains and Extrapolation Methods for Time-Stepping Schemes*. Munich : VDI Verlag, 2012

[83] Huber, R. ; Clauberg, J. ; Ulbrich, H.: An Efficient Spring Model Based on a Curved Beam with Non-Smooth Contact Mechanics for Valve Train Simulations. In: *SAE* 3(1) (2010), S. 28–34

[84] Huber, R. ; Ulbrich, H.: Simulation of a Valve Train Using Non-Smooth Mechanics. In: *SAE International Journal of Engines* 1(1) (2008), S. 208–217

[85] Huber, R. ; Ulbrich, H.: Integration of Non-Smooth Systems using Time-Stepping based Extrapolation Methods and DAE Solver Combined with Time-Stepping. In: *2nd South-East European Conference on Computational Mechanics*. Rhodos, Greece, 2009

[86] Huber, R ; Ulbrich, H.: Higher Order Integration of Non-Smooth Dynamical Systems Using Parallel Computed Extrapolation Methods Based on Time-Stepping Schemes. In: *The 1st Joint International Conference on Multibody System Dynamics*. Lappeenranta, Finland, May 25-27 2010

[87] Hwang, R.S. ; Bae, D.S. ; Kuhl, J.G. ; Haug, E.J.: Parallel processing for real-time dynamic system simulation. In: *Journal of Mechanical Design* 112 (1990), S. 520

[88] Kell, T. ; Fritz, P. ; Pfeiffer, F.: Vibrations in roller chain drives. In: *5th International Congress on Sound and Vibration*. Adelaide, December 1997

[89] Korf, R.E.: A complete anytime algorithm for number partitioning. In: *Artificial Intelligence* 106 (1998), Nr. 2, S. 181–203

[90] Korf, R.E.: Multi-way number partitioning. In: *International Joint Conference on Artificial Intelligence*, 2009, S. 538–543

[91] Korf, R.E.: Objective functions for multi-way number partitioning. In: *Third Annual Symposium on Combinatorial Search*, 2010

[92] Korf, R.E.: A Hybrid Recursive Multi-Way Number Partitioning Algorithm. In: *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011

[93] Koziara, T. ; Bicanic, N.: SOLFEC: a distributed memory parallel implementation of Contact Dynamics. Grenoble, 2011

[94] Krüger, K.: *Einfluss hydraulischer Nockenwellenversteller auf die Dynamik von Ventil- und Steuertrieben.* Munich : Verlag Dr. Hut, 2008

[95] Lathrop, R.H.: Parallelism in manipulator dynamics. In: *The International journal of robotics research* 4 (1985), Nr. 2, S. 80–102

[96] Lee, C.S. ; Chang, P.R.: Efficient parallel algorithm for robot inverse dynamics computation. In: *IEEE Transactions on Systems, Man and Cybernetics.* 16 (1986), Nr. 4, S. 532–542

[97] Lee, C.S.G. ; Chang, P.R.: Efficient parallel algorithms for robot forward dynamics computation. In: *IEEE Transactions on Systems, Man and Cybernetics.* 18 (1988), Nr. 2, S. 238–251

[98] Leine, R.I. ; Glocker, C.: A set-valued force law for spatial Coulomb-Contensou friction. In: *European Journal of Mechanics-A/Solids* 22 (2003), Nr. 2, S. 193–216

[99] Leine, R.I. ; Wouw, N. van d.: *Lecture Notes in Applied and Computational Mechanics.* Bd. 36: *Stability and convergence of mechanical systems with unilateral constraints.* Berlin : Springer, 2008. – ISBN 978–3–540–76974–3

[100] Lunze, J.: *Regelungstechnik 1.* 5. Berlin Heidelberg : Springer, 2005. – 661 S. – ISBN 3540283269

[101] Maeng, H.S. ; Lee, H.S. ; Han, T.D. ; Yang, S.B. ; Kim, S.D.: Dynamic load balancing of iterative data parallel problems on a workstation cluster. In: *High Performance Computing on the Information Superhighway - HPC Asia'97* IEEE, 1997, S. 563–567

[102] Malczyk, P. ; Fraczek, J.: Parallel Efficiency of Lagrange Multipliers Based Divide and Conquer Algorithm for Dynamics of Multibody Systems. In: *Proceedings of the ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.* Washington, DC, USA, 2011

[103] Malczyk, P. ; Fraczek, J. ; Tomulik, P.: A Parallel Algorithm for Constrained Multibody System Dynamics Based on Augemented Lagrangian Method with Projections. In: *Multibody Dynamics, ECCOMAS Thematic Conference.* Brussels, Belgium, 2011

[104] Möller, M.H.: *Consistent Integrators for Non-Smooth Dynamical Systems*, ETH Zurich, Diss., 2011

[105] Moreau, J.J.: Unilateral contact and dry friction in finite freedom dynamics. In: *In Nonsmooth Mechanics and Applications, CISM Courses and Lectures* (1988), Nr. 302, S. 1–82. ISBN 3–211–82066–3

[106] Moureau, V. ; Domingo, P. ; Vervisch, L.: Design of a massively parallel CFD code for complex geometries. In: *Comptes Rendus Mécanique* 339 (2011), Nr. 2, S. 141–148

[107] Mráz, L. ; Valášek, M.: Solution of three key problems for massive parallelization of multibody dynamics. In: *Multibody System Dynamics* (2012), S. 1–19

[108] Muhr, T.H.: *Zur Konstruktion von Ventilfedern in hochbeanspruchten Verbrennungsmotoren*. Rhein.-Westf. Techn. Hochschule, 1992

[109] Mukherjee, R.M. ; Anderson, K.S.: A logarithmic complexity divide-and-conquer algorithm for multi-flexible articulated body dynamics. In: *Journal of computational and nonlinear dynamics* 2 (2007), S. 10

[110] Negrut, D.: Linear algebra considerations for the multi-threaded simulation of mechanical systems. In: *Multibody System Dynamics* 10 (2003), Nr. 1, S. 61–80

[111] Nölting, S.: *Parallelisierung eines Komplexen Finite-Elemente-Programmsystems*. Holzgartenstr. 16, 70174 Stuttgart, Universität Stuttgart, Diss., 2000. `http://elib.uni-stuttgart.de/opus/volltexte/2001/737`

[112] Petzold, L.R.: Description of DASSL: a differential/algebraic system solver / Sandia National Labs., Livermore, CA (USA). 1982. – Forschungsbericht

[113] Pfeiffer, F.: *Einführung in die Dynamik*. 2. Auflage. Stuttgart : Teubner, 1992. – ISBN 3–519–12367–3

[114] Pfeiffer, F.: *Lecture Notes in Applied and Computational Mechanics*. Bd. 40: *Mechanical System Dynamics*. 1. Berlin : Springer, 2005. – ISBN 978–3–540–79435–6

[115] Pfeiffer, F. ; Borchsenius, F.: New hydraulic system modelling. In: *Journal of Vibration and Control* 10 (2004), S. 1493–1515

[116] Pfeiffer, F. ; Borchsenius, F. ; Lebrecht, W. ; Ulbrich, H.: Kurze Simulationszeiten in der Hydraulik - Theorie und Praxis. In: *Proceedings of 4th Int. Fluid Power Conference*. Dresden, Germany, 2004, S. 185–196

[117] Pfeiffer, F. ; Glocker, C.: *Multibody dynamics with unilateral contacts*. 1. New York : John Wiley & Sons Inc., 1996 (Wiley Series in Nonlinear Science). – ISBN 0–471–15565–9

[118] Philips, P. ; Schamel, A. ; Meyer, J.: An Efficient Model for Valve Train and Spring Dynamics. In: *SAE Technical Paper 890619* (1989)

[119] Prescott, W.C.: Parallel Processing of Multibody Systems for Real Time Analysis. In: *Multibody Dynamics 2011, ECCOMAS Thematic Conference*. Brussels, July 2011

[120] Quarteroni, A. ; Sacco, R. ; Saleri, F.: *Numerische Mathematik 1*. Berlin : Springer, 2001. – ISBN 3–540–67878–6

[121] Richardson, L.F. ; Gaunt, J.A.: The deferred approach to the limit. Part I. Single lattice. Part II. Interpenetrating lattices. In: *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 226 (1927), S. 299–361

[122] Rockafellar, R.T.: *Convex analysis*. Bd. 28. Princeton University Press, 1970

[123] Rockafellar, R.T.: Augmented Lagrangians and applications of the proximal point algorithm in convex programming. In: *Mathematics of operations research* (1976), S. 97–116

[124] Schamel, A. ; Hammacher, J. ; Utsch, D.: Modeling and Measurement Techniques for Valve Spring Dynamics in High Revving Internal Combustion Engines. In: *Design of Racing and High Performance Engines* (1993), S. 83–99

[125] Schiehlen, W.: Multibody system dynamics: Roots and perspectives. In: *Multibody system dynamics* 1 (1997), Nr. 2, S. 149–188. – ISSN 1384–5640

[126] Schindler, T.: *Spatial Dynamics of Pushbelt CVTs*. Düsseldorf : VDI Verlag, 2010

[127] Schindler, T. ; Acary, V.: Timestepping schemes for nonsmooth dynamics based on discontinuous Galerkin methods: definition and outlook. In: *INRIA Research Report*, 2011

[128] Schindler, T. ; Foerg, M. ; Friedrich, M. ; Schneider, M. ; Esefeld, B. ; Huber, R. ; Zander, R. ; Ulbrich, H.: Analysing Dynamical Phenomenons: Introduction to MBSim. In: *Proceedings of 1st Joint International Conference on Multibody System Dynamics*. Lappeenranta, Finland, 2010

[129] Schindler, T. ; Friedrich, M. ; Ulbrich, H.: Computing Time Reduction Possibilities in Multibody Dynamics. In: *Proceedings of the Eccomas Multibody Dynamics Conference*. Warsaw, Poland, June 2009

[130] Schindler, T. ; Nguyen, B. ; Trinkle, J.: Understanding the difference between prox and complementarity formulations for simulation of systems with contact. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011* IEEE, 2011, S. 1433–1438

[131] Schneider, M. ; Krüger, H. K. amd U. K. amd Ulbrich: Simulation of Hydraulic Systems with Set-Valued Force Laws. In: *The 1st Joint International Conference on Multibody System Dynamics*. Lappeenranta, Finland, May 25-27 2010

[132] Schneider, M. ; Krüger, K. ; Ulbrich, H.: Experiments and Simulation of Hydraulic Cam Phasing Systems. In: *SAE World Congress 2008 - Variable Valve Optimization*. Detroit, 2008

[133] Shabana, A.A. ; Bauchau, O.A. ; Hulbert, G.M.: Integration of large deformation finite element and multibody system algorithms. In: *Journal of Computational and Nonlinear Dynamics* 2 (2007), S. 351–359. – ISSN 1555–1415

[134] Speckens, F.W.: *Optimierungsstrategien für die Auslegung von Ventilfedern Tassenstösselventiltrieben*. Aachen, Technische Hochschule Aachen, Diss., 1994

[135] Staahl, C.: *An Efficient Spring Model for Valve Train Simulations.* München, TU München, Diplomarbeit, 2008

[136] Steihaug, T. ; Wolfbrandt, A.: An Attempt to Avoid Exact Jacobian and Nonlinear Equations in the Numerical Solution of Stiff Differential Equations. In: *Mathematics of Computation* 33 (1979), Nr. 146, S. 521–534

[137] Stewart, D.E.: Rigid-body dynamics with friction and impact. In: *SIAM review* (2000), Nr. 42, S. 3–39. – ISSN 1095–7200

[138] Stiegelmeyr, A.: *Zur numerischen Berechnung strukturvarianter Mehrkörpersysteme.* München : VDI Verlag, 2001

[139] Studer, C.: *Lecture Notes in Applied and Computational Mechanics.* Bd. 47: *Numerics of Unilateral Contacts and Friction: Modeling and Numerical Time Integration in Non-smooth Dynamics.* Berlin : Springer Verlag, 2009. – ISBN 3–642–01099–7

[140] Tanenbaum, A.S.: *Computerarchitektur. Strukturen - Konzepte - Grundlagen.* 5. Pearson Studium;, 2005. – ISBN 3827371511

[141] Thomsen, P.G. ; True, H.: *Non-smooth Problems in Vehicle Systems Dynamics.* Berlin : Springer, 2010 (Proceedings of the Euromech Colloquium). – ISBN 978–3–642–01355–3

[142] Tichanek, R. ; Fremut, D. ; Cihak, R.: *The Over-Head Cam (OHC) Valve Train Computer Model*

[143] Trottenberg, U. ; Solchenbach, K.: Parallele Algorithmen und ihre Abbildung auf parallele Rechnerarchitekturen. In: *Informationstechnik it* 30 (1988), Nr. 2, S. 71–82

[144] Tsai, F.F.: *Automated methods for high speed simulation of multibody dynamic systems.* 1989

[145] Ulbrich, H.: *Maschinendynamik.* Stuttgart : Teubner Verlag, 1996. – ISBN 3–519–03233–3

[146] Ulbrich, H.: Some Selected Research Activities in Mechatronic Applications - Case Studies. In: *Proceedings of 2nd International Conference on Engineering Mechanics, Structures, Engineering Geology.* Rhodes Island, Greece, July 22-24 2009

[147] Valasek, M. ; Mraz, L.: Parallelization of Multibody System Dynamics by Heterogeneous Multiscale Method. In: *Multibody Dynamics 2011, ECCOMAS Thematic Conference.* Brussels, July 2011

[148] Varah, J.M.: On the efficient implementation of implicit Runge-Kutta methods. In: *Mathematics of Computation* 33 (1979), Nr. 146, S. 557–561

[149] Wang, J. ; Chen, J. ; Wang, Y. ; Zheng, D.: Intelligent load balancing strategies for complex distributed simulation applications. In: *International Conference on Computational Intelligence and Security - CIS'09.* Bd. 2 IEEE, 2009, S. 182–186

[150] Yamane, K. ; Nakamura, Y.: Comparative Study on Serial and Parallel Forward Dynamics Algorithms for Kinematic Chains. In: *The International Journal of Robotic Research* 28 (2009), S. 622. http://dx.doi.org/10.1177/0278364909102350. – DOI 10.1177/0278364909102350

[151] Zander, R.: *Flexible Multi-Body Systems with Set-Valued Force Laws*. VDI Verlag, 2008

[152] Zander, R. ; Schindler, T. ; Friedrich, M. ; Huber, R. ; Förg, M. ; Ulbrich, H.: Non-smooth dynamics in academia and industry: recent work at TU München. In: *Acta Mechanica* 195 (2008), January, Nr. 195, S. 167–183. http://dx.doi.org/10.1007/s00707-007-0570-5. – DOI 10.1007/s00707–007–0570–5

[153] Zomaya, A.Y. ; Teh, Y.H.: Observations on using genetic algorithms for dynamic load-balancing. In: *IEEE Transactions on Parallel and Distributed Systems* 12 (2001), Nr. 9, S. 899–911