Auswirkungen der Peripherieanbindung auf das Realzeitverhalten PC-basierter Multiprozessorsysteme

Jürgen Stohr

Dissertation

Lehrstuhl für Realzeit-Computersysteme

Auswirkungen der Peripherieanbindung auf das Realzeitverhalten PC-basierter Multiprozessorsysteme

Jürgen Stohr

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. U. Schlichtmann
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr.-Ing. G. Färber
	2. Univ.-Prof. Dr.-Ing. K. Diepold

Die Dissertation wurde am 04.10.2005 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 29.03.2006 angenommen.

Danksagung

Diese Dissertation entstand als Ergebnis meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Realzeit-Computersysteme der Technischen Universität München. Teile der Arbeit wurden von der *Deutschen Forschungsgemeinschaft* (DFG) unter dem Förderkennzeichen Fa 109/15-1 („Real-Time with Commercial Off-the-Shelf Multiprocessor Systems“) gefördert.

Mein besonderer Dank geht an Professor Färber, der diese Arbeit ermöglicht hat.

Weiterhin danke ich allen Mitarbeitern des Lehrstuhls für die gute Zusammenarbeit und das kollegiale Arbeitsklima. Besonderer Dank gilt meinem Kollegen Herrn Alexander von Bülow, der zusammen mit mir das Thema der PC-basierten Multiprozessorsysteme bearbeitete.

München, im September 2005

Für meine Frau Monika.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACPI	Advanced Configuration and Power Interface
API	Application Programming Interface
APM	Advanced Power Management
AGP	Accelerated Graphics Port
AGTL	Assisted Gunning Transceiver Logic
APIC	Advanced Programmable Interrupt Controller
BIOS	Basic Input/Output System
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
DDR	Double Data Rate
FSB	Frontside-Bus
GART	Graphics Address Relocation Table
GPOS	General Purpose Operating System
GPU	General Purpose Unit
GTL	Gunning Transceiver Logic
GUI	Graphical User Interface
HT	HyperTransport
HPET	High Precision Event Timer
ICC	Interrupt Controller Communication
ICH	I/O Controller Hub
IPI	Inter-Processor Interrupt
IO	Input Output
ISA	Industrial Standard Architecture
ISR	Interrupt Service Routine
MCH	Memory Controller Hub
NMI	Non-Maskable Interrupt
NUMA	Non-Uniform Memory Access
PC	Personal Computer
PCI	Peripheral Component Interconnect
PIC	Programmable Interrupt Controller
PIO	Programmed Input/Output
PLL	Phase-Locked Loop
PMC	Performance Monitoring Counter

RTOS	Real-Time Operating System
RTU	Real-Time Unit
SCI	System Control Interrupt
SIMD	Single Instruction, Multiple Data
SMI	System Management Interrupt
SMM	System Management Mode
SMP	Symmetric Multi-Processing
SMT	Simultaneous Multithreading
TSC	Time Stamp Counter
WCAT	Worst-Case Access Time
WCET	Worst-Case Execution Time

Kurzfassung

Falls Software ausschließlich auf Code und Daten zugreift, die bereits im Cache eines Prozessors verfügbar sind, ist die Ausführungszeit dieser Software schnell und nahezu konstant. Erfolgen jedoch Zugriffe auf außerhalb der Prozessoren liegende Ressourcen, kommt es zu einer erheblichen Verlängerung der Laufzeit. Sind bei Zugriffen auf eine Ressource parallele Datenströme vorhanden, die von anderen Prozessoren und Peripheriegeräten initiiert werden, verlängern sich die Zugriffszeiten und damit die Ausführungszeiten der Software nochmals. In dieser Arbeit wird die Auswirkung paralleler Datenübertragungen bei Zugriffen auf den Hauptspeicher und auf Peripheriegeräte für SMP- und für NUMA-Systeme untersucht. Dabei wird eine Methodik vorgestellt, mit der mittels Messung die relevanten Parameter zur Bestimmung maximaler Zugriffszeiten auf gemeinsam genutzte Ressourcen ermittelt werden können. Diese Untersuchung bildet die Grundlage für die Bestimmung maximaler Ausführungszeiten von Realzeitsoftware. Zusätzlich wird erläutert, wie gegenseitige Beeinflussungen beim Ressourcenzugriff reduziert werden können. Dabei wird der Parallelbetrieb eines Standard- und eines Realzeitbetriebssystems ermöglicht. Die Untersuchungen werden mit PC-Multiprozessorsystemen ausgeführt.

1 Einleitung

1.1 Motivation

PC-Standardkomponenten erfahren eine stetige Steigerung ihrer Leistungsfähigkeit. Eine Abwärtskompatibilität zu bestehender Software ist dabei im Allgemeinen gewährleistet. So ist es beispielsweise auf modernen CPUs wie Pentium 4 oder Athlon64 immer noch möglich, Software aus dem Geburtsjahr des PCs, dem Jahre 1981, auszuführen. Dabei ist die Ausführungszeit dieser Software auf einer modernen CPU um ein Vielfaches geringer. Die Möglichkeit, durch den Einsatz neuerer Komponenten bestehende Systeme aufzurüsten, sowie eine hohe Leistungsfähigkeit, verbunden mit einem geringen Preis, machen PC-Systeme für den Einsatz in Realzeitsystemen attraktiv. Heutzutage wird unter einem PC im Allgemeinen ein Rechner verstanden, der eine CPU beinhaltet, die zur *Intel Architecture IA-32* kompatibel ist. Dies sind alle Prozessoren, die nach der Einführung des 80386 im Jahre 1986 vorgestellt wurden. Dabei blieb der Befehlssatz dieser Prozessoren weitgehend unverändert.

Problem bei der Verwendung von PC-Komponenten in einem Realzeitsystem ist, dass diese Komponenten nicht für den Einsatz in einer Realzeitumgebung entwickelt werden. Ziel beim Design von Standardkomponenten ist es, eine hohe *durchschnittliche* Rechenleistung zu erlangen. Sporadisch auftretende maximale Laufzeiten sind deshalb möglich. Gerade diese maximalen Laufzeiten von Software, die *Worst-Case Execution Times* (WCET), sind jedoch für Realzeitsysteme von besonderem Interesse.

In dieser Arbeit werden Multiprozessorsysteme auf PC-Basis hinsichtlich ihrer Realzeitfähigkeit untersucht. Dabei soll eine parallele Ausführung von Standard- und Realzeittasks ermöglicht werden. Die Realzeittasks werden unter der Kontrolle eines Realzeitbetriebssystems ausgeführt. Sonstige Applikationen werden von einem Standardbetriebssystem bedient. Dies bietet die Möglichkeit, neben der Ausführung der Realzeitapplikationen den Komfort eines Standardbetriebssystems und alle verfügbaren Applikationen dieses Betriebssystems nutzen zu können. Insbesondere kann das Standardbetriebssystem eine grafische Benutzeroberfläche bereitstellen, über die das Realzeitsystem konfiguriert werden kann. Damit ist es beispielsweise auch möglich, Prozesszustände zu visualisieren.

Weiterhin können Peripheriegeräte über das Standardbetriebssystem angesprochen werden. Da für Realzeitbetriebssysteme im Allgemeinen nur eine geringe Anzahl an Gerätetreibern zur Verfügung steht, kann so — zumindest unter weichen Realzeitbedingungen — auch von den Realzeittasks auf alle vorhandenen Peripheriegeräte zugegriffen werden. So könnte beispielsweise ein Ethernetadapter vom Standardbetriebssystem bedient werden, über den bestimmte Statusinformationen des Realzeitsystems per E-Mail verschickt werden.

Voraussetzung für ein solches Szenario ist, dass die Ausführung der Realzeittasks nicht durch Aktivitäten des Standardbetriebssystems und dessen Applikationen beeinflusst wird. Um die Leistungsfähigkeit eines PC-basierten Multiprozessorsystems ausschöpfen zu können, sollen Realzeittasks so schnell wie möglich ausgeführt werden. Beim gleichzeitigen Zugriff von Standard- und Realzeitapplikationen auf gemeinsam genutzte Betriebsmittel kann es jedoch zu Beeinflussungen kommen, die teilweise erheblich sind. Ziel ist es, diese Beeinflussungen so weit als möglich zu vermeiden. Da dies nicht immer möglich ist, werden Methoden bereitgestellt, die bei der Ermittlung maximaler Lauf- und Zugriffszeiten verwendet werden können.

1.2 Ziele dieser Arbeit

Ohne geeignete Maßnahmen kann Realzeitsoftware, die auf einem PC-basierten Multiprozessorsystem ausgeführt wird, starken Laufzeitschwankungen unterworfen sein. Diese Laufzeitschwankungen sind dabei auf die Aktivität parallel arbeitender Prozessoren und Peripheriegeräte zurückzuführen.

Ziel dieser Arbeit ist es, alle Beeinflussungen zu ermitteln, die sich auf die Laufzeit von Realzeitsoftware auswirken und deren Ursache auf die Hardware zurückzuführen ist, die die Prozessoren umgibt. Im Allgemeinen sind dies die Beeinflussungen, die im Chipsatz und auf den Peripheriebussen entstehen. Es soll untersucht werden, unter welchen Umständen diese Beeinflussungen auftreten und bei welcher Größenordnung deren Einfluss auf Realzeitsoftware liegt. Dabei soll eine obere Schranke für die maximale Ausführungszeit der Software angegeben werden können.

Weiterhin sollen Methoden bereitgestellt werden, die gegenseitige Beeinflussungen beim Zugriff einer Realzeittask auf gemeinsam verwendete Ressourcen des Multiprozessorsystems so weit als möglich vermeiden bzw. minimieren. Dies kann durch geeignete Soft- und Hardwarekonfigurationen geschehen. Allerdings darf zur Ermittlung und Vermeidung maximaler Zugriffszeiten nur eine bestehende PC-Hardware verwendet werden. Spezialhardware darf nicht eingesetzt werden. Hardwaremodifikationen dürfen nicht vorgenommen werden.

Zusätzlich soll darauf eingegangen werden, wie die gegenseitigen Beeinflussungen in einem Realzeitsystem gehandhabt werden müssen. Ziel ist es, die Leistungsfähigkeit eines PC-basierten Multiprozessorsystems für harte Realzeitaufgaben nutzbar zu machen und den parallelen Betrieb von Standard- und Realzeitapplikationen auf einem Multiprozessorsystem zu ermöglichen.

1.3 Gliederung

Die vorliegende Arbeit ist wie folgt gegliedert: In Kapitel 2 werden verwandte Arbeiten zur Abschätzung maximaler Ausführungszeiten in Computersystemen vorgestellt. Hier erfolgt eine

Abgrenzung der vorliegenden Arbeit zu bereits durchgeführten Untersuchungen. Im darauffolgenden Kapitel 3 werden Grundlagen erläutert, die sich mit PCs und Realzeitsystemen im Allgemeinen beschäftigen und dem besseren Verständnis dieser Arbeit dienen. Anschließend wird in Kapitel 4 die Größenordnung der Laufzeitschwankungen in Multiprozessorsystemen auf PC-Basis untersucht. In diesem Kapitel wird beschrieben, mit welchen gegenseitigen Beeinflussungen in SMP- und NUMA-Multiprozessorsystemen zu rechnen ist und wie maximale Zugriffszeiten auf gemeinsam genutzte Ressourcen bestimmt werden können. In Kapitel 5 wird eine Softwarearchitektur vorgestellt, mit der die gegenseitigen Beeinflussungen in einem Multiprozessorsystem gehandhabt werden können. Insbesondere wird hier erläutert, wie Realzeitsoftware schnell und ohne Beeinflussung durch das Standardbetriebssystem ausgeführt werden kann. In Kapitel 6 sind Anwendungsbeispiele aufgeführt, die mögliche Einsätze eines Multiprozessorsystems in einer Realzeitumgebung demonstrieren. Die Zusammenfassung dieser Arbeit erfolgt in Kapitel 7.

2 Stand der Technik

Die meisten Untersuchungen, die auf dem Gebiet der WCET-Bestimmung bei der Verwendung von Standardkomponenten durchgeführt wurden, beschäftigen sich mit Latenzzeiten, die durch die Prozessoren selbst verursacht werden. Insbesondere werden hier die Einflüsse der Caches, der Pipelines und der Sprungvorhersage eingehend untersucht. Die Einflüsse der Komponenten außerhalb eines Prozessors auf die Laufzeit von Realzeitsoftware fanden bis jetzt nur wenig Beachtung. Die Untersuchungen selbst können dabei mithilfe eines Modells oder mittels Messung auf der Zielplattform durchgeführt werden. Teilweise kommt auch eine Kombination aus beiden Varianten zum Einsatz.

In Abschnitt 2.1 werden Arbeiten vorgestellt, die sich u. a. mit der Verwendung von Standardkomponenten in Realzeitsystemen beschäftigen. Anschließend werden in Abschnitt 2.2 Methoden zur WCET-Bestimmung vorgestellt. In Abschnitt 2.3 werden Architekturmodelle und Erweiterungen für Realzeitbetriebssysteme beschrieben, die die Realzeiteigenschaften eingesetzter Standardkomponenten verbessern. In Abschnitt 2.4 folgt eine Zusammenfassung.

2.1 Untersuchung von Standardkomponenten

In den folgenden Unterabschnitten ist Literatur aufgeführt, die sich mit dem Einsatz von Standardkomponenten in Realzeitsystemen beschäftigt bzw. Thematiken aufgreift, die für die in dieser Arbeit durchgeführte Untersuchung von Standardkomponenten relevant sind. Insbesondere ist hier Literatur aufgeführt, die den Einsatz eines PCs oder PowerPCs in Realzeitsystemen untersucht. Hierzu werden zunächst Arbeiten vorgestellt, die den Einsatz von Standardkomponenten analysieren und das System aus Hard- und Software als Gesamtheit betrachten. Die darauffolgenden Unterabschnitte befassen sich mit Untersuchungen, bei denen einzelne Komponenten oder Komponentenverbunde analysiert werden.

2.1.1 Allgemeine Untersuchungen

Burmberger gibt in seiner Dissertation [11] einen Überblick über den Einsatz von PC-Komponenten in Realzeitsystemen. In dieser Arbeit werden die Einsatzgebiete der PCs in zeitkritischen technischen Prozessen untersucht. Anschließend werden verfügbare Realzeitbetriebssysteme miteinander verglichen. Danach werden die verschiedenen Eigenschaften der Hardware, wie beispielsweise Bruttoübertragungsraten vorhandener Bussysteme, zueinander in Relation gesetzt. Ziel dieser Arbeit ist es, die verschiedenen PC-basierten Lösungsmöglichkeiten aufzuzei-

gen, in wenige Klassen von Systemarchitekturen zu gruppieren und hinsichtlich der Einhaltung von Realzeitkriterien zu analysieren.

Hopfner u. a. untersuchen in [26] die prinzipielle Eignung einer Dual-SMP-Architektur für den Realzeitbetrieb. Hierbei wird auf einem Prozessor das Standardbetriebssystem Linux ausgeführt. Auf dem anderen Prozessor werden Realzeittasks bearbeitet. Die Realzeittasks werden dabei wahlweise unter der Kontrolle eines Realzeitbetriebssystems, oder ohne Betriebssystem in Form von Interrupthandlern ausgeführt. Bei den Untersuchungen zeigt sich, dass sich die beiden Prozessoren bei der Ausführung von Software gegenseitig beeinflussen. Diese Beeinflussungen sind teilweise erheblich. Die Ergebnisse dieser Arbeit zeigen auf, dass bei einem entkoppelten Betrieb zweier CPUs auf einem SMP-System mit hohen Latenzzeiten gerechnet werden muss. Aus diesem Grund werden in der vorliegenden Arbeit parallel arbeitende Prozessoren in die Untersuchen mit einbezogen und gegenseitige Abhängigkeiten beim Ressourcenzugriff berücksichtigt.

2.1.2 Einflüsse innerhalb der Prozessoren

Die Einflüsse innerhalb einer CPU werden von vielen Forschungsgruppen untersucht. Im Folgenden ist jeweils ein Vertreter für die Untersuchung der einzelnen Beschleunigungsmechanismen innerhalb der Prozessoren aufgeführt. Diese Literaturliste ließe sich beliebig erweitern. Da der Schwerpunkt der vorliegenden Arbeit auf der Untersuchung der Beeinflussungen auf Zugriffe auf *außerhalb* der Prozessoren angesiedelter Hardware liegt, werden die folgenden Veröffentlichungen nur zur Vollständigkeit aufgeführt.

Die Sprungvorhersage wird beispielsweise von Bate und Reutemann für moderne Mikroprozessoren in [6] betrachtet. In dieser Veröffentlichung wird anhand der Semantik des Quelltextes analysiert, ob ein Sprung einfach, oder schwer vorherzusagen ist. In Abhängigkeit der somit durchgeführten Sprungklassifizierung kann mit Hilfe einer statischen Analyse eine obere Grenze für die Anzahl der falsch vorhergesagten Sprünge angegeben werden. Damit kann die Komplexität des durchzuführenden Realzeitnachweises verringert werden. Die Autoren kommen zu dem Ergebnis, dass dynamische Sprungvorhersagen in Realzeitnachweisen prinzipiell berücksichtigt werden können. Allerdings existieren auch Varianten, die für die Verwendung in Realzeitsystemen ungeeignet sind. Im Gegensatz hierzu stellen Bodin und Puaut in [7] fest, dass das Ergebnis dynamischer Sprungvorhersagen moderner Prozessoren im Allgemein nicht vorhersagbar ist. Statt dessen wird empfohlen, ausschließlich statische Sprungvorhersagen zu verwenden.

Mit Hilfe eines Prozessormodells und statischer Analyse versuchen Heckmann, Wilhelm u. a. in [22], die maximale Ausführungszeit von Realzeitsoftware vorherzusagen. Gegenstand der Untersuchungen ist eine Motorola ColdFire 5307 CPU und ein Motorola PowerPC 775. Untersucht wird der Einfluss der auf den Prozessoren vorhandenen Caches und Pipelines. Die Autoren stellen fest, dass die Wechselbeziehungen der einzelnen Komponenten (Caches, Pipelines, Sprungvorhersage) es verbieten, diese mit Analysewerkzeugen separat zu untersuchen. Die verwendeten Strategien bei der Cacheverdrängung und der dynamischen Sprungvorhersage

verhindern eine exakte Modellierung. Die Autoren kommen zu der Schlussfolgerung, dass moderne Prozessoren nur auf durchschnittliche Rechenleistung optimiert sind. Die Komponenten einer CPU, die zur Erhöhung dieser durchschnittlichen Rechenleistung beitragen, erschweren die Untersuchung der Rechenleistung im Worst-Case. Die separate Analyse liefert nicht ausreichend genaue Ergebnisse.

Die Verwendung der auf einem Prozessor vorhandenen Caches für den Realzeitbetrieb wird beispielsweise von Arnaud und Puaut in [4] untersucht. Diese Veröffentlichung beschäftigt sich mit der Thematik, Teile des Codes im Instruction-Cache eines Prozessors zu fixieren. Dabei werden dynamische und statische Szenarien betrachtet. Die Vorteile dieser Fixierung sind, dass die Rechenleistung im Worst-Case, sowie die durchschnittliche Reaktionszeit einer Realzeittask erhöht werden kann. Aufgrund der vorgenommenen Fixierung treten keine Konflikte innerhalb einer Task und zwischen den Tasks auf. Zusätzlich wird der Realzeitnachweis vereinfacht, da bei einem Taskwechsel die Einflüsse des Caches konstant sind. Allerdings kann dieses Verfahren nur dann sinnvoll eingesetzt werden, falls das Verhältnis zwischen der Größe der zu fixierenden Tasks und der Cachegröße nicht zu groß ist.

2.1.3 Einflüsse außerhalb der Prozessoren

Huang, Liu und Hull untersuchen in [27] den Einfluss eines DMA-Controllers auf Realzeitsoftware, der im Cycle-Stealing Mode betrieben wird. Dabei werden ausschließlich einzelne DMA-Requests betrachtet. Der Einfluss von Burst-Transfers wird nicht untersucht. Grundlage der Untersuchungen ist ein Computersystem, das auf einem Motorola MC68030 Mikroprozessor basiert. Die I/O-Last wird von VMEbus-Geräten erzeugt. In dieser Veröffentlichung wird zunächst davon ausgegangen, dass die Ausführungszeit für jeden Maschinenbefehl konstant ist. Dabei wird ein Prozessor betrachtet, der keinen Cache und keine Pipeline besitzt. Anschließend werden die entwickelten Methoden erweitert, um einen ggf. vorhandenen Instruction-Cache zu berücksichtigen. Die angewandten Methoden werden auf einem MC68030-Simulator verifiziert. Da die Untersuchungen davon ausgehen, dass den einzelnen Busteilnehmern Prioritäten vergeben werden können, sind die Ergebnisse dieser Publikation nicht ohne Modifikation auf PC-Systeme übertragbar. Zusätzlich ist der Cycle-Stealing Betrieb für PC-Systeme nicht relevant.

Moll und Shand betrachten in [35] die Anbindung eines rekonfigurierbaren PCI-Geräts in einem Rechensystem. Nach der Erläuterung der prinzipiellen Funktionsweise des PCI-Busses werden in dieser Veröffentlichung Messungen bezüglich der maximalen Übertragungsrate des Bussystems durchgeführt. Die Messungen werden auf verschiedenen Rechensystemen (Intel/Alpha) vorgenommen und miteinander verglichen. Die Autoren beobachten dabei einen erstaunlichen Unterschied in der Performanz des PCI-Busses bei nahezu identischen Systemen. Ihrer Ansicht nach ist es unmöglich, den Datendurchsatz eines Geräts, das an einem speziellen Rechner betrieben wird, nur anhand der technischen Dokumentation zu bestimmen. Zusätzlich stellen die Autoren fest, dass viele PCI-Geräte Fehler beinhalten und teilweise mit verminderter Leistungsfähigkeit betrieben werden. Die Ergebnisse dieser Veröffentlichung bekräftigen die Methodik,

Zugriffszeiten mittels Messung zu bestimmen. Leider werden in [35] keine Aussagen bzgl. maximaler Zugriffszeiten getroffen.

Schönberg untersucht in [43] den Einfluss von PCI-Geräten auf die Laufzeit von Realzeitsoftware. Es wird ein *Slowdown Factor* definiert, der die Laufzeitverlängerung von Realzeitsoftware bei parallelen Zugriffen von Peripheriegeräten auf den Hauptspeicher beschreibt. Dieser Faktor wird mittels Messung ermittelt. Der Autor geht bei der Ermittlung maximaler Ausführungszeiten von einer bekannten Zusammensetzung der (Assembler-)Instruktionen aus, die Lese- oder Schreibzugriffe, bzw. keine Zugriffe auf den Hauptspeicher erzeugen. Dabei wird eine konstante Ausführungszeit der einzelnen Instruktionen vorausgesetzt. In [42] wird zusätzlich analysiert, wie ein PCI-Arbiter modifiziert werden könnte, um die Realzeitfähigkeit des PCI-Busses zu verbessern. Da hierfür die verwendete Hardware (bzw. die Firmware des Chipsatzes) modifiziert werden muss, sind diese Überlegungen von geringer Relevanz. Der Autor kommt zu dem Ergebnis, dass der Slowdown Faktor, der von PCI-Geräten verursacht wird, gering ist. Grund hierfür ist, dass der PCI-Bus mit 33 MHz arbeitet und die Speicheranbindung bei modernen CPUs im Vergleich zu der der Peripheriegeräte schnell geworden ist. Zusätzlich stellt der Autor fest, dass bei der Verwendung von Multiprozessorsystemen der Einfluss bei Zugriffen auf den Hauptspeicher sehr viel höher sein dürfte. Diese Arbeit ist eine der wenigen Veröffentlichungen, die sich mit dem PCI-Bus in PC-Systemen beschäftigen. Allerdings wird hier nur der Einfluss paralleler PCI-Last auf die Ausführungszeit einzelner Algorithmen untersucht. Vorhandene Caches und Prefetch-Algorithmen werden nicht in die Betrachtungen einbezogen. Die Latenzzeiten beim Zugriff der CPU auf Peripheriegeräte werden nicht untersucht. Eine genaue Analyse der Einwirkung paralleler Datentransfers auf einzelne oder mehrere Ressourcenzugriffe erfolgt nicht. Zusätzlich lassen sich die vorgestellten Methoden nicht ohne Modifikation auf Multiprozessorsysteme übertragen.

Hahn u. a. analysieren in [21] die maximale Antwortzeit einer DMA-Anforderung. Dabei wird ein Rechensystem aus einer CPU und mehreren DMA-Controllern betrachtet. In Abhängigkeit von der Anzahl der Zugriffe der CPU und der DMA-Controller auf ein gemeinsam verwendetes Bussystem wird die maximale Ausführungszeit einer DMA-Datenübertragung bestimmt. Diese Ausführungszeit wird insbesondere für den Realzeitnachweis benötigt. Experimentelle Ergebnisse zeigen, dass die Überabschätzung der Antwortzeit einer DMA-Anforderung maximal 20% beträgt. Bei den Untersuchungen wird vorausgesetzt, dass der CPU und den DMA-Controllern eine feste Priorität zugeordnet werden kann. Dies ist im Allgemeinen beim Einsatz von Standardkomponenten nicht möglich, da hier die verwendeten Arbitrierungsverfahren nicht bekannt sind.

2.2 Abschätzung maximaler Ausführungszeiten

Laufzeiten können mithilfe analytischer Methoden oder durch Messungen auf der Zielplattform ermittelt werden. Im ersteren Fall wird ein Modell der Hardware angefertigt. Mit diesem wird anschließend das zu untersuchende Szenario simuliert. Während dieser Simulation wird die WCET der Realzeitsoftware bestimmt. Bei der Untersuchung anhand von Messungen wird die

Ausführungszeit von Software oder einzelner Instruktionen auf der Zielplattform gemessen. Diese Methode ist insbesondere dann geeignet, falls nicht alle benötigten Parameter für die Erstellung eines Modells bekannt sind bzw. das Modell zu komplex werden würde.

Wegener und Müller vergleichen in [56] beide Methoden. Dabei werden die Ergebnisse der statischen Analyse mithilfe eines Modells mit den Resultaten des *Evolutionary Testing* verglichen. Als Zielplattform wird dabei SPARC verwendet. Die Autoren kommen zu dem Ergebnis, dass sich die Modellierung und die Bestimmung anhand von Messungen gegenseitig ergänzen. Beide Ansätze liefern in etwa vergleichbare Resultate. Bei der statischen Methode müssen die Eigenschaften der simulierten Hardware sehr genau bekannt sein. Im Gegensatz hierzu muss bei der Ermittlung der Ausführungszeiten anhand von Messungen eine große Anzahl unterschiedlicher Messungen durchgeführt werden, wobei eine sichere Abschrankung maximaler Ausführungszeiten nicht garantiert werden kann. Allerdings bietet die WCET-Bestimmung anhand von Messungen den Vorteil, dass Fehler im Zeitverhalten des Realzeitsystems frühzeitig erkannt werden.

2.2.1 Abschätzung durch Modellierung

Stappert und Altenbernd verwenden in [47] das Modell eines PowerPC-Prozessors. Hierbei werden sowohl die Caches, als auch die Pipelines modelliert. Dabei wird ausschließlich linearer Code betrachtet, insbesondere Schleifendurchläufe werden nicht berücksichtigt. In dem vorgestellten Modell wird die sequentielle und parallele Ausführung der Assembleranweisungen in einer CPU untersucht. Zusätzlich können Aussagen darüber getroffen werden, ob zur Laufzeit Speicherzugriffe einen Cache-Hit oder einen Cache-Miss erzeugen. Das eingesetzte Analyseprogramm generiert in den vorgestellten Beispielen eine Überabschätzung von bis zu 14% bei der Anzahl der Instruction-Cache Misses und eine Überabschätzung von bis zu 13% bei der Bestimmung der WCET.

Müller untersucht in [36] die maximale Ausführungszeit von Software in Abhängigkeit einer beliebigen Assoziativität des Instruction-Caches. Mit Hilfe eines Modells wird untersucht, ob Zugriffe aus dem Cache bedient werden können, oder ob diese Zugriffe auf den Hauptspeicher erfolgen müssen. Eine Pipeline wird in die Untersuchungen mit einbezogen. Der Autor kommt zu dem Ergebnis, dass die Ausführungszeiten bei der Verwendung set-assoziativer Caches im Vergleich zu direkt abgebildeten Caches mit einer ähnlichen Genauigkeit vorherbestimmt werden können. In den Messergebnissen liefert diese Methode eine Überabschätzung der WCET von bis zu 14%.

Schneider und Ferdinand untersuchen in [41] die Pipeline eines superskalaren Prozessors. Hierbei wird das Verhalten der Pipeline eines SuperSPARC I Prozessors anhand eines Modells vorherbestimmt. Eine Out-Of-Order Execution wird nicht berücksichtigt. Die Autoren vertreten die Ansicht, dass eine ausschließliche Bestimmung der WCET anhand von Messungen aufgrund der Vielzahl möglicher Eingangsparameter nicht möglich ist. Mit *Abstract Interpretation* wird versucht, die WCET statisch zu bestimmen. Hierbei wird die Semantik des zu untersuchenden Codeabschnitts mit in die Untersuchungen einbezogen. Im Gegensatz zu der später erschienenen Veröffentlichung [22] der gleichen Forschungsgruppe rund um Prof. Wilhelm kommen

die Autoren zu dem Ergebnis, dass die Untersuchung des Verhaltens der Pipelines und des Instruction-Caches voneinander unabhängig durchgeführt werden können.

2.2.2 Abschätzung durch Messungen

Petters beschreibt in [39] ein Methode, mit der mittels Messung die maximale Laufzeit von Realzeitsoftware abgeschätzt werden kann. Die Untersuchungen werden dabei auf PC-basierten Einprozessormaschinen durchgeführt. In dieser Arbeit werden zunächst die Beschleunigungsmechanismen moderner Prozessoren beschrieben. Anschließend wird erläutert, wie die längsten Pfade in Realzeitsoftware ermittelt werden können. Danach wird versucht, die maximale Ausführungszeit dieser Pfade durch Messung zu bestimmen. Dabei wird vor jeder Messung der WCET eines Pfades der Prozessor in den schlimmsten einstellbaren Zustand gebracht. Beispielsweise werden die Caches mit modifizierten Daten gefüllt, die vor dem Laden neuer Cachelines in den Hauptspeicher zurückgeschrieben werden müssen. Bei den Messungen der Ausführungszeiten der einzelnen Pfade stellt Petters bei der Wiederholung der Messungen eine große Schwankung der Messergebnisse fest. Um eine obere Schranke angeben zu können, wird mittels *Extremwertstatistik* versucht, diese zu bestimmen. Die Untersuchungen von Petters sind für die vorliegende Arbeit von großer Relevanz. Insbesondere die verwendete Messmethodik wird erweitert und auf die Untersuchung von Multiprozessorsystemen angepasst. Die Untersuchungen von Petters zeigen auch, dass die WCET nicht ohne Weiteres mittels Messung bestimmt werden kann. Dies führt zu der in der vorliegenden Arbeit vorgenommenen getrennten Betrachtung von externen und internen Zugriffen einer CPU während der Ausführung von Software.

Burns und Edgar stellen in [12] eine Methode vor, mit der mittels Extremwertstatistik maximale Ausführungszeiten von Realzeitsoftware ermittelt werden können. Falls Ausführungszeiten von Codeabschnitten mit Messungen bestimmt werden, ist nicht gewährleistet, dass die gemessenen maximalen Ausführungszeiten dem jeweils tatsächlichen Worst-Case entspricht. In dieser Veröffentlichung wird deshalb vorgeschlagen, die gemessenen Werte statisch mit Hilfe statistischer Methoden zu analysieren. Die Autoren kommen zu dem Ergebnis, dass die Verwendung der Extremwertstatistik brauchbare Ergebnisse für die Modellierung der Rechenzeit von Software liefert. Dennoch wird in der vorliegenden Arbeit auf Statistik verzichtet, da harte Realzeitanforderungen mit statistischen Methoden nicht garantiert werden können.

Mehnert, Hohmuth und Härtig untersuchen in [33] die Kosten für separate Adressräume in Realzeitapplikationen. In dieser Arbeit werden Interruptlatenzzeiten in Abhängigkeit der Privilegierungsstufe bestimmt, in der die Realzeitsoftware ausgeführt wird. Als Plattform werden PC-basierte Einprozessormaschinen verwendet. Als Betriebssystem kommt Linux bzw. eine RTLinux-Variante zum Einsatz. Grund hierfür ist die Verfügbarkeit des Quelltextes. Die Interruptlatenzzeiten werden dabei ausschließlich mit Messungen ermittelt; die hierfür verwendeten Messmethoden sind jedoch nicht erläutert. Auf einem Pentium 4, der mit 1,6GHz getaktet ist, wird eine Latenzzeit bis zum Start des Interrupthandlers von $14\mu s$ gemessen. Die Größenordnung dieses Ergebnisses deckt sich mit den in der vorliegenden Arbeit ermittelten Werten (vgl. S. 91).

Atanassov, Kirner und Puschner bestimmen in [5] mittels systematischer Messungen die Ausführungszeit einzelner Instruktionen, sowie Folgen von Instruktionen auf einem Infineon C167 Prozessor. Die Messergebnisse werden anschließend in einem Modell verwendet. Motivation für dieses Vorgehen ist, dass die vom Hersteller zur Verfügung gestellten Handbücher nicht exakt sind, einige interne Eigenschaften des Prozessors geheim gehalten werden und dass es aufgrund von Abweichungen der Implementierung von Spezifikation und Design der Prozessoren, sowie aufgrund von Fehlern in der Hardware zu verlängerten Ausführungszeiten kommen kann. Als Ergebnis dieser Arbeit lässt sich festhalten, dass eine Bestimmung der WCET unter ausschließlicher Verwendung der Spezifikationen zu einer Unterabschätzung der WCET führen kann.

Heursch u. a. untersuchen in [24] und [25] die (weichen) Realzeiteigenschaften des Linux-Kernels. Dabei werden spezielle Patches verwendet, die die Reaktionsfähigkeit des Kernels erhöhen sollen. Bei diesen Untersuchungen werden die jeweiligen Ausführungszeiten über ein Messverfahren bestimmt. Die durchgeführten Untersuchungen beschäftigen sich ausschließlich mit den Realzeiteigenschaften des Kernels selbst. Beispielsweise werden die Stellen im Quelltext untersucht, an denen keine Unterbrechung auftreten darf. Die Einflüsse der zugrundeliegenden Hardware auf Realzeitsoftware werden dagegen nicht berücksichtigt.

2.3 Architekturmodelle und Erweiterungen für Realzeitbetriebssysteme

Liedtke, Härtig und Hohmuth beschreiben in [29] ein Verfahren, bei dem die Zuordnung physikalischer Adressen des Hauptspeichers auf bestimmte Sets im Cache einer CPU dafür verwendet werden kann, Teile einer Applikation (Code und Daten) im Cache einer CPU zu fixieren. Dies kann beispielsweise über die Speicherverwaltung des (Realzeit-)Betriebssystems realisiert werden. Grundlage der Untersuchungen sind PC-basierte Einprozessorsysteme. Problem bei dem vorgestellten Verfahren ist, dass in Cache und Hauptspeicher jeweils der gleiche Anteil für eine bestimmte Applikation reserviert werden muss. Dies führt zu einem hohen Speicherverbrauch bei der Verwendung von Einprozessormaschinen. Diese Veröffentlichung bildet u. a. die Grundlage für die, in dieser Arbeit angeschnittene und in [14] durchgeführte Methode zur Anordnung von Code und Daten in den Caches der CPUs eines Multiprozessorsystems.

Bucar erläutert in [9] eine Methode, mit der Teile eines Interrupt-Handlers im L1-Cache eines PowerPC-Prozessors fixiert werden können. Ziel dieser Maßnahme ist es, die Interruptlatenzzeit eines Realzeitsystems zu verringern. Im Gegensatz zu den CPUs der IA-32-Familie wird bei dieser PowerPC-CPU die Fixierung von Code und Daten im Cache von der Hardware unterstützt. Der Autor kommt zu dem Ergebnis, dass durch die Reservierung einzelner Cachelines für den Interrupthandler dessen Ausführungszeit verringert wird. Allerdings stehen die reservierten Cachelines nicht mehr für die restlichen Applikationen zur Verfügung, so dass es hier zu einer Verlängerung der WCET kommen kann.

Srinivasan u. a. beschreiben in [46], welche Erweiterungen für das Standardbetriebssystem Linux durchgeführt werden müssen, um die Realzeitfähigkeit zu steigern. Die vorgestellten Methoden führen zu der Realzeiterweiterung KURT (Kansas University Real-Time). Da die Realzeiteigenschaften dieses Systems besser als die des Betriebssystems Linux sind, aber schlechter als die Realzeiteigenschaften reiner Echtzeitsysteme, werden die Realzeiteigenschaften von KURT als *firm* bezeichnet. Die wesentlichen Erweiterungen von KURT sind die Verbesserung der Zeitauflösung, sowie ein spezieller Scheduler. Die Eigenschaften der eingesetzten Hardwareplattform werden von KURT nicht weitergehend berücksichtigt.

Yodaiken stellt in [57] eine Methode vor, in der er beschreibt, wie auf einem Standard-PC harte Realzeittasks parallel zu einem Standardbetriebssystem ausgeführt werden können. Dabei wird ein kleiner Realzeitkernel zwischen die Hardware und dem Standardbetriebssystem geschoben. Die Realzeittasks werden von diesem Realzeitkernel ausgeführt; das Standardbetriebssystem ist die niederpriorste Task des Realzeitbetriebssystems. Diese Untersuchungen bilden die Grundlage für RTLinux und RTAI. In der vorliegenden Arbeit wird die Architektur von RTLinux/RTAI erweitert, um die Untersuchung der Realzeiteigenschaften von PC-basierten Multiprozessorsystemen durchführen zu können. Damit wird auch eine Separation von Standard- und Realzeitapplikationen auf den einzelnen CPUs ermöglicht.

2.4 Zusammenfassung

Viele Forschungsgruppen beschäftigen sich mit Detailproblemen, die oft für die Abschätzung maximaler Lauf- und Zugriffszeiten von Realzeitsoftware von geringer Relevanz sind. Insbesondere der in modernen CPUs vorhandenen Sprungvorhersage, sowie den Pipelines wird eine zu große Bedeutung beigemessen. Da die Zeitstrafen für einen falsch vorhergesagten Sprung bzw. für einen Pipeline-Flush bei modernen Prozessoren im Nanosekundenbereich liegen [14], müssen diese Einflüsse nicht weiter berücksichtigt werden.

Die auf den Prozessoren vorhandenen Caches sind dagegen für die Abschätzung der WCET von hoher Relevanz. In Abhängigkeit davon, ob ein Datum im Cache liegt oder nicht, kann es zu extremen Unterschieden in der Ausführungszeit kommen. Durch parallel stattfindende Datentransfers anderer Prozessoren und Peripheriegeräte wird dieser Laufzeitunterschied zusätzlich verstärkt. Mit dieser Thematik befassen sich die Untersuchungen in Kapitel 4.

Die Einflüsse von außerhalb eines Prozessors liegender Hardware auf die Ausführungszeit von Realzeitsoftware wurden bis jetzt nur von wenigen Forschungsgruppen untersucht. Eine systematische Untersuchung möglicher Einflussfaktoren auf die Laufzeit erfolgte hierbei nicht. Im Allgemeinen wurde die Ausführungszeit einzelner Codeabschnitte gemessen, ohne vorhergehender Überprüfung, ob und wie viele Hauptspeicher- und Peripheriezugriffe in dem betrachteten Codeabschnitt tatsächlich ausgeführt werden. Gerade diese Information ist jedoch für die Abschätzung maximaler Ausführungszeiten von wesentlichem Interesse.

Modellierung wird meist für einfache Systeme durchgeführt. Für komplexere Computersysteme können Ausführungszeiten im Allgemeinen nur durch Messung abgeschätzt werden. Grund

hierfür ist, dass u. a. die betrachteten Architekturen mittels Modellierung nicht mehr gehandhabt werden können und die benötigten Spezifikationen teilweise nicht zugänglich oder fehlerhaft sind.

Die in dieser Arbeit durchgeführte Untersuchung von Multiprozessorsystemen hat auf den ersten Blick den Nachteil, dass im Vergleich zu Einprozessormaschinen auf einem Multiprozessorsystem prinzipiell mit größeren gegenseitigen Beeinflussungen gerechnet werden muss. Allerdings bietet diese Architektur auch den Vorteil, dass bei geeigneter Konfiguration das Messergebnis weniger durch die Messsoftware beeinflusst wird. Somit wird ermöglicht, gezielt einzelne Beeinflussungen zu bestimmen, was auf einem Einprozessorsystem nur bedingt durchführbar ist.

3 Grundlagen

Das folgende Kapitel beschreibt einige Grundlagen, die für das Verständnis der vorliegenden Arbeit benötigt werden. Diese werden dabei nur grob umrissen. Für weiterführende Informationen muss die jeweils aufgeführte Literatur herangezogen werden.

3.1 Realzeitsysteme

An Realzeitsysteme wird die Anforderung gestellt, innerhalb einer bestimmten *Zeitspanne* zu reagieren. Dies kann beispielsweise die Reaktion auf ein Ereignis sein, oder eine Berechnung, die innerhalb einer gewissen Zeit abgeschlossen werden muss. Geschwindigkeit spielt bei Realzeitsystemen primär keine Rolle. Einzig die Garantie, dass eine Reaktion spätestens zu einem bestimmten Zeitpunkt erfolgt, ist bei Realzeitsystemen von Interesse. Realzeitsysteme sind beispielsweise in der Signalverarbeitung, in digitalen Reglern und in Telekommunikationseinrichtungen vorhanden. Im Folgenden sind die wichtigsten Einsatzgebiete von Realzeitsystemen aufgeführt, vergleiche hierzu auch [32]:

Regelungen: Viele Realzeitsysteme arbeiten als digitale Steuergeräte und sind hierfür mit Sensoren und Aktoren verbunden. Dieser Typ von Realzeitsystemen ist beispielsweise in Motorsteuerungen, in der Automatisierungstechnik und in Haushaltsgeräten zu finden. Im Allgemeinen werden hier analoge Eingangssignale in digitale gewandelt und von einem Regelalgorithmus verarbeitet. Dieser Algorithmus ist dabei meist in Software programmiert und wird von einem Prozessor unter der Kontrolle eines Realzeitbetriebssystems ausgeführt. Wurden die jeweiligen Stellgrößen berechnet, werden diese digitalen Werte in analoge gewandelt und an die Aktoren des zu regelnden Systems geleitet. In Abbildung 3.1 ist dieses Szenario verdeutlicht: In diesem Beispiel wird ein PID-Regler betrachtet. Mit Hilfe der analogen Führungsgröße $r(t)$ und der mit einem analogen Sensor gemessenen Regelgröße $y(t)$ wird nach einer A/D-Wandlung die Differenz vom Soll- zum

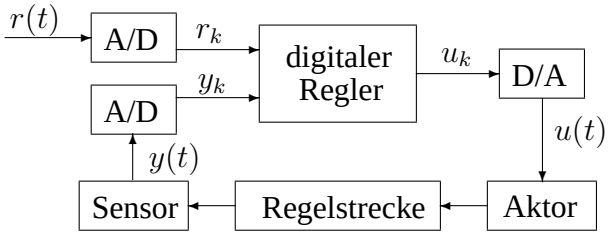


Abbildung 3.1: digitaler Regler

Ist-Zustand berechnet. In Abhängigkeit von dieser Differenz und der Vorgeschichte des Signalverlaufs wird die Stellgröße u_k ermittelt. Diese wird anschließend zur Ansteuerung des Aktors in die analoge Größe $u(t)$ gewandelt.

Hierarchische Realzeitsysteme: Viele Realzeitsysteme bestehen aus komplexen Hierarchien. Diese Systeme beinhalten mehrere Ebenen, wobei die jeweils übergeordnete Ebene zeitliche Anforderungen an die darunterliegende hat. Dabei agiert nur die unterste Ebene mit dem physikalischen System; die darüberliegenden Ebenen verwenden virtuelle Regel- und Stellgrößen. Beispiele hierfür lassen sich in der Automatisierungstechnik finden, wo die unterste Ebene des Realzeitsystems direkt mit den jeweiligen Sensoren und Aktoren kommuniziert. In einer darüberliegenden Ebene wird dem Operator eine Benutzerschnittstelle zur Verfügung gestellt, mit der beispielsweise die jeweiligen Führungsgrößen konfiguriert werden können.

Signalverarbeitung: Viele Applikationen der Signalverarbeitung haben Realzeitanforderungen, wobei hier die Reaktionszeit im Allgemeinen im Millisekunden- bis Sekundenbereich liegt. Beispiele sind die digitale Filterung, die Kompression, Dekompression und Auswertung von Audio- und Videodaten, sowie die Signalverarbeitung in der Radartechnik.

Um eine schritthaltende Verarbeitung im Realzeitbetrieb zu ermöglichen, werden hierfür einige wichtige Parameter definiert, die das Realzeitsystem beschreiben (vgl. [17]):

- Die Prozesszeit t_P ist der zeitliche Abstand zwischen zwei gleichen Ereignissen eines technischen Prozesses. Diese Zeit ist bei periodischen Ereignissen konstant.

- Die Wartezeit t_W hängt davon ab, wie lange eine Task wegen anderer Prozesse warten muss. Wartezeiten treten auf, falls eine Task zwar lauffähig ist, aber höherpriore Prozesse die Ausführung verzögern. Insbesondere kann während der Laufzeit die Task durch Interrupts oder durch höherpriore Tasks, die lauffähig werden, unterbrochen werden. Durch Kontextwechsel entstehen zusätzliche Latenzzeiten. Falls mehrere Tasks auf ein gemeinsam genutztes Betriebsmittel zugreifen, treten ggf. weitere Wartezeiten auf, da die betrachtete Task warten muss, bis das jeweilige Betriebsmittel freigegeben wird. Im Wesentlichen wird die Wartezeit durch die im Realzeitsystem verwendeten Scheduling- und Ressourcenzuteilungsprotokolle bestimmt.

- Die Verarbeitungszeit t_V wird zur Durchführung der Realzeittask benötigt. Diese Zeit ist die reine Ausführungszeit der Task auf dem Prozessor. Die Verarbeitungszeit hängt zum einen von der Semantik des Codes ab, zum anderen muss hier auch die Ausführungsgeschwindigkeit der eingesetzten Hardware berücksichtigt werden. Durch ggf. vorhandene Beschleunigungsmechanismen, wie beispielsweise Caches, treten große Differenzen in der Laufzeit auf. Weiterhin können sich parallel arbeitende Prozessoren und Peripheriegeräte auf die Ausführungszeit der Realzeittask auswirken. Im Wesentlichen beschäftigt sich diese Arbeit mit der Bestimmung der maximalen Verarbeitungszeit auf Multiprozessorsystemen.

- Die Reaktionszeit t_R setzt sich aus der Verarbeitungszeit t_V und der Wartezeit t_W zusammen. Deshalb gibt die Reaktionszeit an, wie lange das Realzeitsystem benötigt, um auf ein Ereignis zu reagieren. Daher gilt:

$$t_R = t_V + t_W$$

- Die maximal zulässige Reaktionszeit t_{Rmax} wird auch als Deadline bezeichnet, wobei im Allgemeinen $t_{Rmax} < t_P$ gilt. Die Deadline einer Task ist der Zeitpunkt, an dem die Ausführung der Task spätestens beendet sein muss.

Des Weiteren werden Realzeitbedingungen in *weiche* und *harte* Realzeitbedingungen unterteilt. Die Unterscheidung erfolgt dabei über die *Kosten*, die entstehen, falls eine Deadline nicht eingehalten werden kann. Falls bei Nichteinhaltung einer Deadline keine nennenswerten Kosten entstehen, spricht man von einer weichen Realzeitbedingung. Hier ist beispielsweise die Benutzerschnittstelle (GUI) eines Realzeitsystems zu nennen, bei der ein kurzer "Hänger" keine nennenswerten Auswirkungen hat. Im Gegensatz hierzu existieren harte Realzeitbedingungen, bei denen ein Überschreiten der Deadline zu hohen Kosten führt. Beispiel hierfür sind zeitkritische Regelungsaufgaben, wie sie in der Automatisierungstechnik oder in Flugzeugen zu finden sind.

3.2 Realzeitbetriebssysteme

Realzeitbetriebssysteme haben die Aufgabe, die Realzeittasks unter Realzeitbedingungen auszuführen. Bei geeigneter Parametrisierung gewährleistet ein Realzeitbetriebssystem, dass jede Task ihre Deadline einhalten kann. Im Allgemeinen werden von einem Realzeitbetriebssystem mehrere Dienste bereitgestellt, die im Folgenden kurz beschrieben werden sollen:

- Wichtigste Komponente in einem Realzeitbetriebssystem ist ein Realzeit-Scheduler, der ein preemptives Multitasking ermöglicht. In Abhängigkeit ihrer Priorität bzw. ihrer zeitlichen Anforderungen wird eine Task auf einer CPU ausgeführt. Andere Tasks, die ebenfalls ausgeführt werden sollen, die also lauffähig sind, müssen warten, bis ihnen der Scheduler den Zugriff auf einen Prozessor gewährt. Beispiele für Schedulingverfahren sind prioritätsgesteuertes Scheduling — hier wird jeder Task eine festgelegte Priorität zugeordnet — und Deadline-Scheduling. Im letzteren Fall wird die Priorität in Abhängigkeit von der jeweiligen Deadline dynamisch verändert.
- Im verwendeten Scheduler sind meist zusätzliche Protokolle enthalten, die der Synchronisation der Ressourcenzugriffe der Realzeittasks dienen. Ziel hierbei ist es, eine Prioritätsinversion zu vermeiden. Beispiel hierfür ist das Priority Ceiling Protocol (PCP) für prioritätsgesteuertes Scheduling.
- Ein wichtiges Element eines Realzeitbetriebssystems ist die Zeitverwaltung. Im Allgemeinen wird von einem Zeitgeber-Baustein periodisch ein Interrupt erzeugt, der die Granularität des Realzeitsystems vorgibt. Über diesen Zeitgeber wird der Multitaskingbetrieb ermöglicht; ein Taskwechsel ist hier u. a. bei jedem Interrupt möglich.

- Zusätzlich stellt ein Realzeitbetriebssystem Mechanismen bereit, um auf Interrupts reagieren zu können. So ist es im Allgemeinen möglich, über das Realzeitbetriebssystem einen Interrupthandler zu registrieren. Über diesen Handler kann anschließend auf externe Ereignisse reagiert werden und ggf. eine Task des Realzeitsystems angestoßen werden.

- Weiterhin werden von einem Realzeitbetriebssystem Methoden zur Synchronisation und zur Kommunikation zwischen den jeweiligen Tasks bereitgestellt. Dies kann beispielsweise über Semaphore und Mailboxes geschehen.

- Im Allgemeinen sind in Realzeitbetriebssystemen Treiber vorhanden, um Hardware anzubinden, auf die von einer Realzeittask zugegriffen werden soll. Beispiel hierfür sind Treiber für AD/DA-Wandlerkarten.

Ob es sich bei einem Realzeitbetriebssystem um ein weiches oder um ein hartes Realzeitbetriebssystem handelt, hängt davon ab, ob das Betriebssystem für eine gegebene Kombination aus Soft- und Hardware garantieren kann, dass die maximalen Reaktionszeiten *immer* eingehalten, oder *meistens* eingehalten werden können. Wesentliches Kriterium hierfür ist die Bestimmung der maximalen Verarbeitungszeit t_V auf einer gegebenen Hardware. Diese Zeit wird auch als die maximale Ausführungszeit von Software bzw. als *Worst-Case Execution Time (WCET)* bezeichnet.

In dieser Arbeit werden die Einflüsse der Hardware auf die maximale Ausführungszeit von Software untersucht. Grundlage der Untersuchungen sind Multiprozessorsysteme auf PC-Basis. Untersuchungen bzgl. der Semantik von Realzeitapplikationen, sowie Latenzzeiten, die durch die verwendeten Scheduling-Mechanismen des Realzeitsystems entstehen, sind nicht Gegenstand dieser Arbeit.

Im Folgenden sollen einige wichtige Eigenschaften der PC-Architektur vorgestellt werden.

3.3 PC-Architektur

PC ist die Abkürzung für *Personal Computer* und besagt, dass jeder Benutzer mit seinem *persönlichen* Computer arbeiten kann. Ursprung des PCs ist das Jahr 1981. Hier erschien der Original-PC von IBM, der einen 16-Bit Mikroprozessor, den 8088, beinhaltete.

Die PC- oder x86-Architektur[1)] ist die im Büro- und Heimbereich am häufigsten anzutreffende Architektur. Gründe hierfür sind das gute Preis-Leistungs-Verhältnis, der hohe Bekanntheitsgrad, der insbesondere auch über Schulen gefördert wird, sowie die Abwärtskompatibilität. Ein umfassender Überblick über die PC-Architektur ist beispielsweise in [34] zu finden.

[1)] Der Ausdruck *x86* rührt von der ursprünglichen Benennung der Prozessoren der Firma Intel her. Diese endete jeweils auf 86. Diese Nomenklatur änderte sich erst bei Einführung des Pentium. Da neben der IA-32 Architektur auch die Architekturen AMD64 bzw. EM64T existieren, wird der Begriff x86 als Synonym für diese PC-Architekturen verwendet.

3.3.1 Überblick über die Prozessoren der x86-Familie

Urvater der PC-Architektur ist der 8086-Prozessor der Firma Intel, der 1978 als Nachfolger des 8080 vorgestellt wurde. Dieser Prozessor implementiert eine 16-Bit Architektur; der Befehlssatz umfasst 123 Befehle. Diese CPU beinhaltet einen gemultiplexten Daten- und Adressbus, arbeitet ausschließlich im Real-Mode und kann 1 MByte Hauptspeicher adressieren.

Der 8088-Prozessor wurde 1981 als abgespeckte Version des 8086 herausgebracht. Dieser Prozessor beinhaltet den gleichen Befehlsvorrat und verwendet die gleichen Adressierungen wie sein großer Bruder, besitzt aus Kostengründen allerdings nur einen 8 Bit breiten Datenbus. Somit ließen sich mit dem 8088 insbesondere preisgünstigere Platinen herstellen.

Relevanter Nachfolger des 8086 ist der 80286, der den *Protected Virtual Addressing Mode* einführte. Dieser Prozessor ist ebenfalls nur eine 16-Bit CPU, besitzt einen 24-Bit Adressbus und einen 16-Bit Datenbus und kann maximal 24 MByte adressieren.

Der erste 32-Bit Prozessor für PCs ist der 80386, wobei die grundlegende Architektur der Prozessoren bis zum Pentium III unverändert bleibt. Dieser Prozessor kann einen Adressraum von maximal 4 GByte adressieren und besitzt eine *Memory Management Unit (MMU)*, mit der der Speicher in Segmente und Seiten eingeteilt werden kann. Zusätzlich beinhaltet der 80386 einen Cache. Der 80386 ist die erste CPU der IA-32 Prozessorfamilie. Diese hat bis heute große Bedeutung.

Nachfolgende Prozessoren binden einen mathematischen Coprozessor an die CPU an (i486) und implementieren eine bzw. mehrere Pipelines (Pentium). Anschließend kommen spezielle Erweiterungen wie die *Multi-Media Extension (MMX*, Pentium) und *Streaming SIMD Extension (SSE*, Pentium III) hinzu. Der Intel Pentium war dabei die erste CPU der x86-Familie, die in Zweiprozessormaschinen eingesetzt werden konnte. Ab dem Pentium beinhalten die Prozessoren der x86-Familie die *Performance Monitoring Counter (PMC)*, verfügen über eine Sprungvorhersage, haben einen getrennten L1-Cache für Code und Daten und unterstützen das Power Management.

Mit dem Pentium IV wurden die x86-Prozessoren des Herstellers Intel neu gestaltet. Ziel hierbei war die Unterstützung hoher Taktfrequenzen. Diese CPUs verwenden eine 20-stufige Pipeline und eine modifizierte Architektur der Caches. Zusätzlich unterstützen diese Prozessoren Hyper-Threading. Bei Hyper-Threading stellt ein Prozessor zwei virtuelle CPUs bereit. Damit lässt sich die Auslastung der einzelnen Ressourcen eines Prozessors steigern.

Seit der Einführung der Opteron- bzw. Athlon64-CPUs des Herstellers AMD sind 64-Bit Prozessoren verfügbar, die zur x86-Architektur kompatibel sind. Die 64-Bit Erweiterung wurde von Intel lizenziert und wird hier unter dem Namen *Extended Memory 64-Bit Technology (EM64T)* vermarktet. Weiterhin werden mit den AMD Opteron CPUs NUMA-Multiprozessorsysteme für die x86-Architektur angeboten.

3.3.2 Architekturen für Multiprozessorsysteme

Multiprozessorsysteme, die auf der x86-Architektur basieren, sind als *Symmetric Multi-Processing (SMP)* Systeme und als *Non-Uniform Memory Access (NUMA)* Systeme erhältlich. Der prinzipielle Unterschied beider Varianten ist, dass bei SMP alle Prozessoren auf einen gemeinsam genutzten Hauptspeicher zugreifen. NUMA-Systeme sind dagegen in Knoten (Nodes) eingeteilt. Jeder Knoten enthält im Allgemeinen einen oder mehrere Prozessoren und einen Teil des Hauptspeichers. Zugriffe auf den eigenen Knoten werden normalerweise schneller durchgeführt als Hauptspeicherzugriffe auf einen anderen Knoten. Diese Zugriffe sind deshalb *non-uniform*. Peripheriegeräte werden bei beiden Varianten über den Chipsatz angebunden; bei NUMA-Systemen können diese einem Knoten zugeordnet werden. Aus der Sicht von Software ist die NUMA-Architektur transparent und unterscheidet sich nicht von einem SMP-System.

Den Prozessoren beider Architekturen ist gemeinsam, dass diese gleichberechtigt sind. Insbesondere darf jede x86-CPU auf jede Adresse des Hauptspeichers und auf jedes Peripheriegerät zugreifen. Interrupts können von jeder möglichen Interruptquelle an die Prozessoren geliefert werden. Weiterhin darf jede CPU mit jedem anderen Prozessor kommunizieren, wobei zum Nachrichtenaustausch *Inter-Processor Interrupts (IPIs)* verwendet werden. Die eigentlichen Daten werden hierbei über gemeinsame Speicherbereiche (Shared-Memory) übertragen.

Die Realzeiteigenschaften von SMP- und NUMA-Systemen werden ausführlich in den Abschnitten 4.3.1 und 4.3.2 untersucht. Für weiterführende Informationen sei deshalb auf diese Abschnitte verwiesen.

3.3.3 Chipsatz und Peripheriegeräte

Der *Chipsatz* eines Rechensystems hat die Aufgabe, die Prozessoren mit dem Hauptspeicher und mit den Peripheriegeräten zu verbinden. Im Allgemeinen lässt sich ein Chipsatz in eine *North*- und in eine *South-Bridge* einteilen. Diese Namensgebung rührt von der Lage dieser Bausteine auf dem Schaltplan einer Mutterplatine her: Die North-Bridge ist normalerweise oben zu finden, also in Analogie zu Landkarten “im Norden”. Die South-Bridge ist unten angesiedelt.

North- und South-Bridge können jeweils aus einem oder aus mehreren Bausteinen bestehen. Die North-Bridge bindet dabei die Prozessoren an den Hauptspeicher an und stellt eine oder mehrere Bridges zu Peripheriebussen zur Verfügung. Deshalb wird eine North-Bridge auch als *Host-Bridge* bezeichnet. Weitere Bezeichnung für eine North-Bridge ist auch *Memory Controller Hub (MCH)*. In der South-Bridge sind u. a. einige Peripheriegeräte und Gerätecontroller integriert, die bereits auf dem Mainboard vorhanden sind und deshalb nicht in Form von Einsteckkarten an das Rechensystem angebunden werden müssen. Die South-Bridge wird deshalb auch als *I/O Controller Hub (ICH)* bezeichnet. Meist werden über die South-Bridge ein oder mehrere zusätzliche Peripheriebusse bereitgestellt, die langsamer als die von der North-Bridge zur Verfügung gestellten arbeiten und der Anbindung weiterer Peripheriegeräte dienen.

Der am häufigsten anzutreffende Peripheriebus ist der *Peripheral Component Interconnect Bus (PCI-Bus)*. Dieser Bus bindet die einzelnen Peripheriegeräte an das Rechensystem an. Bei eini-

gen Chipsätzen wird der PCI-Bus auch für den Datenaustausch zwischen North- und South-Bridge verwendet. Bei anderen Chipsätzen wird dagegen für diese Kommunikation *Hyper-Transport* oder eine proprietäre Punkt-zu-Punkt Verbindung eingesetzt.

Über Gerätecontroller, die beispielsweise an einem PCI-Bus angeschlossen sind, können weitere Peripheriebusse angebunden werden. So wird beispielsweise über den Controller einer FireWire-Karte ein FireWire-Bus an das Rechensystem angeschlossen. Über diesen Bus können weitere Peripheriegeräte betrieben werden, wie beispielsweise Kameras oder Massenspeicher. Weitere Bussysteme sind SCSI (Small Computer System Interface), IDE (Integrated Drive Electronics), USB (Universal Serial Bus) und ISA (Industrial Standard Architecture). Letzteres Bussystem hat allerdings heutzutage keine Bedeutung mehr.

Eng verbunden mit dem Chipsatz ist dessen Firmware, die auch BIOS (Basic Input/Output System) genannt wird. Das BIOS hat die Aufgabe, die Hardware des Rechensystems bei einem Neustart zu initialisieren. Zusätzlich kann mit den Funktionen des BIOS auf Massenspeicher zugegriffen werden, wodurch das initiale Laden eines Betriebssystems ermöglicht wird.

Peripheriegeräte dienen dazu, dass das Rechensystem mit seiner Umwelt kommunizieren kann. Insbesondere dienen Peripheriegeräte der Ein- und Ausgabe digitaler Informationen. Beispiele für Peripheriegeräte sind Massenspeicher, Grafikkarten, Eingabegeräte wie Tastatur und Maus, Netzwerkkarten und Prozesssignaladapter. Im Folgenden werden als Peripheriegeräte alle Geräte und Gerätecontroller bezeichnet, die *direkt* mit den Bussystemen des Chipsatzes verbunden sind. Dies sind alle Geräte, auf deren Register und Speicherbereiche der *direkte* Zugriff eines Prozessors möglich ist. So wird beispielsweise der Controller eines Massenspeichers als Peripheriegerät bezeichnet.

4 Hardwarebedingte Schwankungen der Ausführungszeiten

In modernen PC-Systemen werden Schwankungen in der Laufzeit von Realzeitsoftware von den verwendeten Techniken bei der Befehlsbearbeitung, von der Anbindung der Prozessoren an Speicher und Peripherie und von den eingesetzten Beschleunigungsmechanismen verursacht. Weiteren Einfluss haben parallel arbeitende Prozessoren und DMA-Controller, sowie aktivierte Mechanismen des *Power Managements*. Schwankungen in der Ausführungszeit sind dabei meist auf eine Kombination mehrerer Einflüsse zurückzuführen, die sich jeweils unterschiedlich stark auf die Laufzeit von Realzeitsoftware auswirken.

Die Ursachen unterschiedlicher Ausführungszeiten von Software lassen sich in die folgenden drei Gruppen einteilen:

- *Einflüsse der Prozessorarchitektur:* Hierunter fallen alle Einflüsse, die durch die verwendeten Mechanismen zur Erhöhung der Rechenleistung *innerhalb* einer CPU hervorgerufen werden. Insbesondere sind dies die Einflüsse der *Sprungvorhersage (Branch Prediction)*, der *Pipelines* und der auf den Prozessoren vorhandenen *Caches.*
- *Einflüsse durch die Anbindung von Speicher und Peripherie:* Falls die Prozessoren und die DMA-Controller der Peripheriegeräte auf den gemeinsamen Hauptspeicher zugreifen, können aufgrund paralleler Anforderungen Zugriffe einer CPU verzögert ausgeführt werden. Falls ein Prozessor auf den Adressraum eines Peripheriegeräts zugreifen muss, können diese Zugriffe ebenfalls durch DMA-Transfers anderer Prozessoren und Peripheriegeräte verzögert werden.
- *Einflüsse des Power Managements:* Diese Beeinflussungen entstehen aufgrund aktivierter *PC Health* und *Power Management* Funktionalität. Beispielsweise verlängern sich die Laufzeiten von Realzeitsoftware, falls ein Prozessor aufgrund überhöhter Temperatur seine Taktfrequenz verringert, oder die Hardware einen *System Management Interrupt (SMI)* auslöst.

In diesem Kapitel werden die Auswirkungen der verschiedenen Einflüsse auf das Laufzeitverhalten von Realzeitsoftware untersucht. Ziel ist es, die Größenordnung der verschiedenen Beeinflussungen anzugeben, um hieraus Vorschläge zur Konstruktion eines Realzeitbetriebssystems in Kapitel 5 ableiten zu können. Hierzu wird in Abschnitt 4.1 das verwendete Messverfahren vorgestellt. In Anschnitt 4.2 werden anschließend die zu erwartenden Laufzeitschwankungen beschrieben, die durch die verwendeten Mechanismen innerhalb der Prozessoren hervorgerufen werden. Da solche Untersuchungen nicht Kern dieser Arbeit sind, wird hierauf nur kurz eingegangen. Abschnitt 4.3 befasst sich mit den Einflüssen auf Laufzeiten von Software,

die durch die Anbindung von Speicher und Peripherie an die CPUs hervorgerufen werden. Hier wird beschrieben, wie maximale Zugriffszeiten auf den Hauptspeicher und auf Peripheriegeräte für SMP- und für NUMA-Systeme bestimmt werden können. Alle wichtigen Peripheriebusse werden betrachtet und deren Einfluss auf Ausführungszeiten von Realzeitsoftware diskutiert. Die Bestimmung maximaler Interruptlaufzeiten runden diesen Abschnitt ab. In Abschnitt 4.4 wird der Einfluss des Power Managements untersucht. Die Zusammenfassung der in diesem Kapitel erlangten Erkenntnisse erfolgt abschließend in Abschnitt 4.5.

4.1 Messung von Ausführungszeiten

Um die Einflüsse der verschiedenen Komponenten der PC-Architektur auf das Laufzeitverhalten von Realzeitsoftware bestimmen und gemessene Ausführungszeiten miteinander vergleichen zu können, benötigt man eine Messmethodik, die das System vor jeder Messung in einen definierten Zustand versetzt. Somit können Laufzeitschwankungen aufgrund unterschiedlicher Ausgangszustände zu Messbeginn vermieden werden, womit eine Klassifizierung der einzelnen Einflussfaktoren auf die Ausführungszeiten von Software ermöglicht wird.

Da für Realzeitsysteme meist die WCET von Interesse ist, müssen alle Parameter zu Messbeginn so eingestellt werden, dass sie die Ausführungszeit jeweils um ihr Maximum verlängern. Insbesondere müssen vor der jeweiligen Messung die Caches der Prozessoren in einen definierten Zustand versetzt werden. Während der Messung dürfen nicht betrachtete Einflüsse parallel arbeitender Prozessoren und Peripheriegeräte das Messergebnis nicht verfälschen.

4.1.1 Der Time Stamp Counter

Zur Messung von Laufzeiten eignet sich der auf allen modernen Prozessoren der x86-Architektur vorhandene *Time Stamp Counter (TSC)*. Dieses 64-Bit Register wird bei jedem Prozessortakt um eins erhöht. Die damit zu erreichende Zeitauflösung der Messdatenerfassung t_A ist der Kehrwert des Prozessortaktes f_P:

$$t_A = \frac{1}{f_P} \tag{4.1}$$

Die Zeit $t_{\ddot{U}}$ bis zu einem Überlauf des TSC ist gegeben durch:

$$t_{\ddot{U}} = \frac{2^{64}}{f_P} \tag{4.2}$$

Tabelle 4.1 zeigt die Auflösung und die Zeit bis zum Überlauf des TSC in Abhängigkeit von der Frequenz. Da ein Überlauf des TSC nicht auftreten kann und eine Zeitauflösung auf Basis des Prozessortaktes ermöglicht wird, eignet sich dieses Register in besonderem Maße zur Bestimmung von Laufzeiten in PC-Systemen. Voraussetzung ist, dass während der Messung die Taktfrequenz des Prozessors nicht verändert wird.

f_P in MHz	t_A in ns	$t_{\ddot{U}}$ in Jahren
233	4,29	2510
1533	0,65	382
3000	0,33	195

Tabelle 4.1: Auflösung und Überlauf des TSC

Der Prozessortakt wird von einem Quarzoszillator festgelegt, der mit Hilfe einer PLL-Schaltung den jeweiligen Takt liefert. Der Temperaturkoeffizient der Resonanzfrequenz bei Quarzoszillatoren ist sehr klein. Die erreichbare Frequenzstabilität $\frac{\triangle f}{f}$ liegt nach [53] in der Größenordnung von 10^{-6} bis 10^{-10}.

Da die zu messenden Laufzeiten kurz sind, müssen eventuelle Abweichungen zwischen dem nominellen und dem tatsächlichen Prozessortakt nicht weiter berücksichtigt werden. Auf die mit dem TSC zu erreichende Messgenauigkeit wird in Abschnitt 4.1.5 nochmals eingegangen.

4.1.2 Ablauf einer Messung

Abbildung 4.1 illustriert den prinzipiellen Ablauf einer Messung: Zunächst muss vor der jeweiligen Messung das System in einen definierten Zustand versetzt werden ①. Um vergleichbare Messwerte zu erhalten, dürfen nicht betrachtete Komponenten des PCs das Messergebnis nicht beeinflussen. Falls dies nicht erreicht werden kann, müssen die jeweiligen Komponenten so konfiguriert werden, dass sie sich auf die Messung immer in der gleichen Art und Weise auswirken. Mögliche Konfigurationsmaßnahmen, die in Abhängigkeit der zu untersuchenden Einflüsse vor jeder Messung ausgeführt werden können, sind:

- Die Caches der jeweiligen CPU können mit modifizierten Daten gefüllt werden, die vor der Ausführung des zu vermessenden Codeabschnitts im Hauptspeicher gesichert werden müssen. Jede Cacheline enthält somit ungültige Daten; das *Invalid*-Bit ist gesetzt. Damit werden für jeden Zugriff der CPU auf den Hauptspeicher ein Schreib- und ein Lesezugriff durchgeführt. Die Caches befinden sich folglich in dem Zustand, der die Ausführungszeit um das Maximum verlängert.

- Die CPU, die das Standardbetriebssystem ausführt, kann in einen Zustand versetzt werden, in dem sie eine definierte Funktionalität durchführt. Beispielsweise kann sie während der Messung in einer Schleife gehalten werden, die sich vollständig im Cache befindet, oder mit Hilfe von verschachtelten NMI-Handlern mit gesperrten Interrupts in den Zustand “Prozessorhalt” gebracht werden. Alternativ sind auch definierte Speicher- oder I/O-Zugriffe möglich.

- Der Prozessor, der das Standardbetriebssystem ausführt, kann daran gehindert werden, auf definierte I/O-Bereiche zuzugreifen (Portzugriffe und Memory-Mapped-I/O). Eine Task des Standardbetriebssystems wird bis zum Ende der aktuellen Messung blockiert, falls diese einen I/O-Zugriff durchführt.

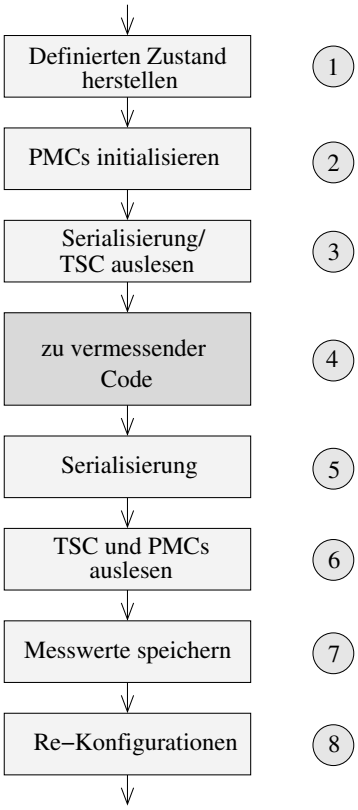


Abbildung 4.1: Programmablaufplan einer Messung

- Einzelne Peripheriegeräte können so programmiert werden, dass sie während einer Messung keine DMA-Burst-Transfers anstoßen. Somit haben diese Transfers keinen Einfluss auf I/O- und Speicherzugriffe der CPUs. Durch externe Maßnahmen können Peripheriegeräte dazu gebracht werden, relativ viele DMA-Burst-Transfers durchzuführen. Allerdings kann hier keine definierte Buslast eingestellt werden.

Nachdem das System in einen definierten Zustand versetzt wurde, werden in ② die *Performance Monitoring Counter (PMC)* initialisiert. Die PMCs sind spezielle Register der Prozessoren, die bestimmte Ereignisse über einen definierten Zeitraum protokollieren können. Beispielsweise können mit diesen Registern die Anzahl der I/O-Wartezyklen und die Anzahl der Zugriffe auf den Hauptspeicher ermittelt werden.

Da aufgrund der in den Prozessoren vorhandenen Pipelines die Befehle nicht zwingend in der programmierten Reihenfolge ausgeführt werden, wird in ③ eine *Serialisierung* erzwungen. Alle ausstehenden Befehle werden vor dieser Anweisung beendet. Anschließend wird der TSC ausgelesen.

In ④ wird nun die zu vermessende Software ausgeführt. Da hier nur Laufzeiten und diverse Ereignisse mit Hilfe des TSC und der PMCs gemessen werden können, darf dieser Code nicht zu komplex sein. Da ausschließlich Einflüsse der Hardware auf das Laufzeitverhalten von Realzeitsoftware bestimmt werden sollen, darf die Ausführungszeit insbesondere nicht von variablen

Eingangsparametern abhängen, die beispielsweise zu alternativen Pfaden in dem Codeabschnitt führen oder eine variable Anzahl an Schleifendurchläufen erzeugen.

Nachdem der zu vermessende Codeabschnitt durchlaufen wurde, wird in ⑤ wieder eine Serialisierung durchgeführt. Anschließend werden in ⑥ die PMCs und der TSC ausgelesen. Die benötigte Ausführungszeit c des untersuchten Codeabschnitts lässt sich direkt aus der Differenz der Werte $z_{TSC,Beginn}$ und $z_{TSC,Ende}$ des TSC zu Beginn und am Ende der Messung angeben:

$$c = (z_{TSC,Ende} - z_{TSC,Beginn}) \cdot t_A \tag{4.3}$$

Die ermittelten Messwerte werden anschließend in ⑦ im Hauptspeicher abgespeichert. Falls hier der Speicherbereich voll sein sollte, wird die Aufnahme von Messwerten unterbrochen — die Realzeitsoftware läuft dabei weiter — und die Messwerte werden auf einem Massenspeicher abgelegt. In ⑧ werden die unter ① durchgeführten Konfigurationsmaßnahmen wieder rückgängig gemacht.

4.1.3 Durchführung einer Messung

Die in Kapitel 5 vorgestellte Architektur sieht vor, dass auf einer CPU des Multiprozessorsystems ein Standardbetriebssystem ausgeführt wird und alle weiteren Prozessoren für den Realzeitbetrieb zur Verfügung stehen. Die verwendete Architektur ermöglicht es, die Messsoftware während des Realzeitbetriebs zu konfigurieren. Aufgenommene Messwerte, die über einen Shared-Memory dem Standardbetriebssystem zugänglich gemacht werden, können von diesem verarbeitet werden.

Abbildung 4.2 veranschaulicht die verwendete Messumgebung. Um Messungen durchführen zu können, müssen zunächst Messpunkte in den Quelltext der Realzeitapplikationen eingefügt werden. Dies kann entweder manuell, oder mit Hilfe einer Meta-Modellierungssprache, wie beispielsweise MetaC, realisiert werden. Die für die Messung benötigte Bibliothek der Messumgebung wird beim Binden der Realzeitapplikation hinzugebunden.

Wenn die Realzeitsoftware gestartet wird, werden zunächst keine Messwerte aufgenommen. Hier entscheidet ein Flag, ob tatsächlich Messungen durchgeführt werden sollen. Mit einer GUI kann nun zunächst die Messsoftware konfiguriert werden. Insbesondere können hier die durchzuführenden Konfigurationsmaßnahmen zu Beginn der Messungen festgelegt werden.

Nachdem die Konfiguration der Messsoftware abgeschlossen ist, wird die Aufnahme von Messwerten aktiviert, indem das entsprechende Flag gesetzt wird. Die Messwerte werden von der Messsoftware zunächst im Hauptspeicher zwischengespeichert. Wenn dieser Speicherbereich voll ist, wird die Aufnahme von Messwerten gestoppt. Die Messsoftware signalisiert anschließend der Steuersoftware, dass Messwerte auf den Massenspeicher übertragen werden können. Nachdem diese Übertragung abgeschlossen wurde, werden von der Messsoftware erneut Messungen durchgeführt.

Nachdem die gewünschte Anzahl an Messungen erreicht wurde, können die gespeicherten Messwerte auf dem Standardbetriebssystem visualisiert und ausgewertet werden.

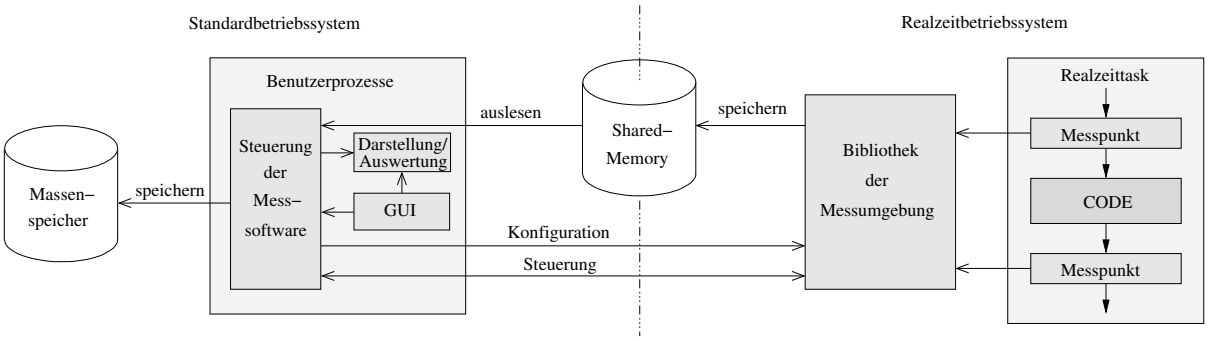


Abbildung 4.2: Messumgebung

4.1.4 Laufzeitparameter der Messsoftware

Die durchgeführten Methoden zur Ermittlung von Messwerten verlängern die Ausführungszeit der Software. Den größten Einfluss hat hierbei das Füllen der Caches mit nutzlosen Daten (siehe Seite 22); die Größenordnung liegt hierbei im Millisekundenbereich.

Falls ein Prozessor — dies ist normalerweise die CPU, die das Standardbetriebssystem ausführt — definierten Code während einer Messung bearbeiten soll, muss hierzu die Ausführung unterbrochen und die entsprechende Routine aufgerufen werden. Die hierfür benötigte Zeit hängt davon ab, wie schnell die CPU auf einen *Inter-Processor Interrupt (IPI)* reagieren kann. Um möglichst kurze Reaktionszeiten zu erreichen, wird von der Messsoftware ein nicht maskierbarer Interrupt (NMI) ausgelöst. Der entsprechende Prozessor reagiert innerhalb weniger Mikrosekunden.

Während der Aufnahme von Messwerten kann die CPU, auf der das Standardbetriebssystem ausgeführt wird, daran gehindert werden, auf definierte I/O-Bereiche zuzugreifen. Da hier nur eine globale Konfigurationsvariable gesetzt werden muss, beschränkt sich die benötigte Ausführungszeit auf wenige Maschinenbefehle.

Falls während einer Messung Peripheriegeräte keine DMA-Burst-Transfers durchführen sollen, beträgt die benötigte Zeit zur Konfiguration der Peripheriegeräte nur wenige Mikrosekunden. Diese Zeit hängt dabei von der Anzahl der zu konfigurierenden Peripheriegeräte ab. Die benötigten Zeiten zum Ausführen der Serialisierungsanweisungen, zum Auslesen des TSC und der PMCs und zum Abspeichern der Messwerte betragen ebenfalls nur wenige Mikrosekunden.

In Tabelle 4.2 sind die gemessenen *maximalen* Laufzeitparameter der Messsoftware aufgeführt. Die Werte wurden auf einem Dual-Athlon MP System mit einem CPU-Takt von 1533 MHz und auf einem Quad-Opteron System mit einem CPU-Takt von 1,8 GHz ermittelt.

4.1.5 Messgenauigkeit

Obwohl die Messungen mit Hilfe des TSC in Abhängigkeit vom Prozessortakt eine Zeitauflösung von t_A ermöglichen, hängt die tatsächlich zu verwendende Messgenauigkeit von den

durchgeführte Funktionalität	ist optional	benötigte Laufzeit Athlon/Opteron
Invalidierung Prozessorcaches	ja	$0{,}9ms$ / $3{,}2ms$
Ausführung definierten Codes auf CPUs	ja	$6{,}8\mu s$ / $4{,}6\mu s$
Konfiguration Peripheriegeräte	ja	$5{,}8\mu s$ / $6{,}8\mu s$
Konfiguration definierter I/O-Zugriffe	ja	$0{,}7\mu s$ / $0{,}7\mu s$
Serialisierungsanweisungen, Auslesen/Abspeichern TSC und PMCs	nein	$1{,}0\mu s$ / $1{,}3\mu s$

Tabelle 4.2: Laufzeitparameter der Messroutine

zu untersuchenden Parametern ab. Da die Einflussfaktoren, die sich nur unterschwellig auf das Messergebnis auswirken, vernachlässigt werden sollen, wird der Faktor c_A definiert, mit dessen Hilfe die zu verwendende Messgenauigkeit berechnet werden kann. Beispielsweise sollen die Auswirkungen des mit deutlich niedrigerem Takt arbeitenden Host-Busses oder der Konfigurationszustand einiger interner Parameter des Prozessors, wie beispielsweise die der Sprungvorhersage, bei der Messauswertung unberücksichtigt bleiben. c_A ist dabei die um eins inkrementierte (dimensionslose) Differenz zwischen der gemessenen maximalen und minimalen Anzahl c_{max} und c_{min} von Prozessortakten der Routine, deren Laufzeit bestimmt werden soll, bei definierter Ausgangssituation ohne der Einwirkung parallel arbeitender Prozessoren oder Peripheriegeräte.

$$c_A = c_{max} - c_{min} + 1 \tag{4.4}$$

Damit lässt sich die zu verwendende Genauigkeit t'_A der Messwerte mit

$$t'_A = c_A \cdot t_A \tag{4.5}$$

angeben. Um die bei den jeweiligen Messungen verwendete Auflösung t^*_A zu erhalten, wird der ermittelte Wert von t'_A auf die am nächsten liegende Zehnerpotenz gerundet:

$$t^*_A = p(t'_A), \text{ mit } p(x) = 10^{\lfloor lg(x) \rfloor}, \tag{4.6}$$

wobei $\lfloor x \rfloor$ die größte ganze Zahl darstellt, die kleiner oder gleich x ist. Damit werden auch geringe Einflüsse gemessen. Aufgrund dieser Maßnahme dürfen jedoch Abweichungen in der letzten Nachkommastelle beim Vergleich zweier Messwerte nicht allzu hoch bewertet werden.

Zur Illustration sind in Tabelle 4.3 die verwendeten Werte für die Messgenauigkeit t^*_A für das untersuchte Dual-Athlon System und für das Quad-Opteron System angegeben. Beim Athlon System erfolgt der Zugriff auf die Host-Bridge, beim Opteron System wird ein PCI-Peripheriegerät angesprochen.

f_P in MHz	t_A in ns	c_A [in Takten]	t^*_A in μs
1533	0,65	23	0,01
1800	0,56	82	0,01

Tabelle 4.3: verwendete Messgenauigkeit

4.2 Einflüsse der Prozessorarchitektur

In modernen Prozessoren sind mehrere Beschleunigungsmechanismen vorhanden, die insbesondere die *durchschnittliche* Ausführungszeit von Software verringern: *Caches* können Teile des Hauptspeichers zwischenspeichern und somit Zugriffszeiten auf Code und Daten verringern. Mit Hilfe von *Pipelines* werden Befehle innerhalb des Steuerwerks parallel verarbeitet. Die *Sprungvorhersage* (engl. Branch Prediction) versucht, das Ergebnis von bedingten Sprüngen vorherzusagen, damit der Prozessor die Pipeline mit den wahrscheinlich auszuführenden Befehlen füllen kann. Um die Auslastung der Funktionseinheiten einer CPU weiter zu steigern, kann sich mit Hilfe von *Simultaneous Multithreading (SMT)* bzw. Hyper-Threading ein Prozessor als mehrere logische CPUs ausgeben.

Die einzelnen Mechanismen sollen im Folgenden auf ihre Auswirkung auf die Laufzeit von Realzeitsoftware untersucht werden. Hierzu wird zunächst in Abschnitt 4.2.1 der Einfluss der auf den Prozessoren vorhandenen Caches auf Software evaluiert. Da die weiteren auf den Prozessoren vorhandenen Beschleunigungsmechanismen nur unwesentlich zu auftretenden Laufzeitschwankungen beitragen, wird auf diese nur kurz in Abschnitt 4.2.2 eingegangen. Eine Zusammenfassung der Einflüsse der Prozessorarchitektur erfolgt in Abschnitt 4.2.3.

4.2.1 Caches

In modernen Computersystemen werden Caches benötigt, um die unterschiedliche Leistung von Prozessor und Hauptspeicher auszugleichen: Da die Leistungssteigerung der Prozessoren sehr viel schneller voranschreitet als die erlangten Fortschritte zur Verringerung der Zugriffszeiten auf den Hauptspeicher, wird mit Hilfe von schnellen Zwischenspeichern versucht, die Auswirkungen dieser Leistungsdifferenz zu verringern.

Die in der PC-Architektur verwendeten Caches sind hierarchisch angeordnet, wobei bis zu drei Cache-Level verwendet werden.[1)] Auf den schnellen und kleinen 1st-Level Cache kann die CPU innerhalb eines Taktes zugreifen. 2nd- und 3rd-Level Cache sind etwas größer und meist langsamer im Vergleich zum 1st-Level Cache. Letztere bieten dabei die Möglichkeit, größere Teile des Hauptspeichers zwischenzuspeichern und Zugriffe auf die zwischengespeicherten Daten in akzeptabler Zeit zu ermöglichen. Für die verwendeten Athlon und Opteron CPUs, sowie für eine Intel Xeon MP CPU (Gallatin, 2,8 GHz) sind in Tabelle 4.4 die jeweiligen Größen der Caches angegeben.

4.2.1.1 Auswirkung auf den Realzeitbetrieb

Da Caches nur einen kleinen Ausschnitt des Hauptspeichers zwischenspeichern, ist nicht immer gewährleistet, dass sich die von der Realzeitapplikation angeforderten Daten im Cache befinden. Falls ein *Cache-Miss* auftritt, müssen die Daten aus dem wesentlich langsameren Haupt-

[1)] Grund für die hierarchische Anordnung der Caches ist insbesondere der dadurch resultierende Kostenvorteil.

	1st-Level (Code/Daten)	2nd-Level	3rd-Level
Athlon MP	64 KByte / 64 KByte	256 KByte	—
Opteron	64 KByte / 64 KByte	1024 KByte	—
Xeon MP	12K μOPs[2)]/ 8 KByte	512 KByte	4096 KByte

Tabelle 4.4: Cachegrößen moderner Prozessoren

speicher geladen werden. Wird durch den Zugriff eine Cacheline verdrängt, die modifizierte Daten enthält, dann müssen diese zusätzlich in den Hauptspeicher zurückgeschrieben werden.

Zu welchen Zeitpunkten es zu einem Cache-Miss in einer Realzeitapplikation kommt, hängt von der Struktur des Codes und von der Semantik der Anwendung ab. Hier bestimmt insbesondere die relative Lage der einzelnen Programmsegmente im Hauptspeicher, wann Verdrängungen auftreten. Durch geeignete Anordnung der Realzeitsoftware im Hauptspeicher kann die Anzahl der auftretenden Verdrängungen minimiert und der Zeitpunkt des Auftretens der Cacheverdrängungen vorherbestimmt werden. Für weiterführende Informationen zu dieser Thematik vergleiche auch Abschnitt 5.2.6.

Durch parallel arbeitende Prozessoren und DMA-Controller können die in einem Cache zwischengespeicherten Daten ihre Gültigkeit verlieren. Das MESI bzw. das MOESI Protokoll stellt dabei sicher, dass sich die Caches und der Hauptspeicher immer in einem konsistenten Zustand befinden. Falls ein Prozessor oder ein Peripheriegerät Daten modifiziert, die in einem Cache zwischengespeichert sind, wird die entsprechende Cacheline als ungültig (invalid) markiert. Bei der Konstruktion von Realzeitsystemen muss dieser Umstand bei der Festlegung gemeinsam genutzter Speicherbereiche berücksichtigt werden.

4.2.1.2 Größenordnung der Beeinflussungen

Um die Größenordnung der Kosten von Cache-Misses zu bestimmen, werden in diesem Abschnitt die folgenden Messungen durchgeführt:

1. Die Verlängerung der Ausführungszeit, die sich aufgrund von Cache-Misses ergibt, wird bestimmt. Hierzu werden Daten modifiziert, wobei sich diese Daten zum einen bereits im Cache befinden, zum anderen vor dem jeweiligen Zugriff aus dem Hauptspeicher geladen werden müssen.

2. Die benötigte Ausführungszeit zur Sortierung von 16k Worten mit Hilfe des Quicksort Algorithmus wird bestimmt. Dabei werden die beiden Szenarien — “Code und Daten vor Sortierbeginn im Cache” und “Caches enthalten modifizierte Einträge vor Sortierbeginn” zueinander in Relation gesetzt. Mit dieser Messung soll die Auswirkung der Caches auf Software demonstriert werden.

Bei letzterer Messung befindet sich ein Datum “im Cache”, falls es im L1- oder L2-Cache zwischengespeichert ist (ein L3-Cache ist bei den betrachteten Prozessorarchitekturen nicht

[2)] Der x86-Code wird vor der Bearbeitung durch den Prozessor in RISC-artige Grundbefehle, den μOPs, übersetzt. Bei dieser CPU wird der einmal übersetzte Code im *Trace Cache* zwischengespeichert.

vorhanden). Der Einfluss der verschieden Cache-Hierarchien soll hier nicht weiter untersucht werden, da die auftretenden Laufzeitschwankungen im Vergleich zu Zugriffen auf den Hauptspeicher vernachlässigbar sind (vgl. [14]).

Die Messungen werden auf einem Dual-Athlon System und auf einem Opteron System durchgeführt. Für andere Prozessorarchitekturen sind ähnliche Ergebnisse zu erwarten. Während der Messungen werden alle Aktivitäten parallel arbeitender Prozessoren und DMA-Controller deaktiviert. Somit kann ein Zugriff eines Prozessors auf den Hauptspeicher sofort erfolgen und wird nicht durch parallel ablaufende Speichertransfers behindert.

Hauptspeicherzugriffe versus Cachezugriffe

Um die Kosten für Cacheverdrängungen abschätzen zu können, wird in der folgenden Messung auf ein Datenfeld zugegriffen. Dabei wird bei jedem Zugriff das entsprechende Datum geändert. Der Code befindet sich während der Messungen im Cache und wird nicht verdrängt. Um Hauptspeicher- und Cachezugriffe miteinander vergleichen zu können, müssen in der ersten Messreihe die Daten aus dem Hauptspeicher geladen werden. In der zweiten Messreihe sind die Daten bereits im Cache vorhanden:

Hauptspeicherzugriffe: Die Caches werden vor den Zugriffen so konfiguriert, dass das jeweilige Set mit veränderten und ungesicherten Daten gefüllt ist. Der Lese- und Schreibzugriff auf das entsprechende Datum löst deshalb zwei Hauptspeicherzugriffe aus: Mit einem Schreibzugriff werden die ungesicherten Daten in den Hauptspeicher gerettet, im folgenden Lesezugriff wird das angeforderte Datum in den Prozessor geladen.

Cachezugriffe: Die Daten sind vor den Zugriffen bereits im Cache vorhanden. Bei diesen Messungen wird der Prozessor ausschließlich aus den Daten des Caches bedient — es erfolgen keine Zugriffe auf den Hauptspeicher.

Das Ergebnis dieser Messreihe ist für das Athlon System in Abbildung 4.3 dargestellt. Auf der Abszisse ist die Anzahl der durchgeführten Zugriffe auf das Datenfeld angetragen. Falls die Daten nicht im Cache sind, folgen hier aus einem Zugriff zwei Zugriffe auf den Hauptspeicher. Auf der Ordinate ist die für die Zugriffe benötigte Zeit in Mikrosekunden angetragen. Da die Messungen mehrmals[3)] wiederholt werden, ist bei den Hauptspeicherzugriffen die minimal und die maximal gemessene Ausführungszeit für eine bestimmte Anzahl an Zugriffen angetragen.

Falls eine CPU auf den Hauptspeicher zugreifen muss, lässt sich für eine große Anzahl durchgeführter Zugriffe die dafür maximal benötigte Ausführungszeit t_{HS} direkt aus der Anzahl z der durchgeführten Zugriffe und der maximal benötigten Zugriffszeit $t_{RAM,WCET}$ ermitteln:

$$t_{HS} = z \cdot t_{RAM,WCET} \tag{4.7}$$

[3)] Eine Messung wird 10.000 mal wiederholt.

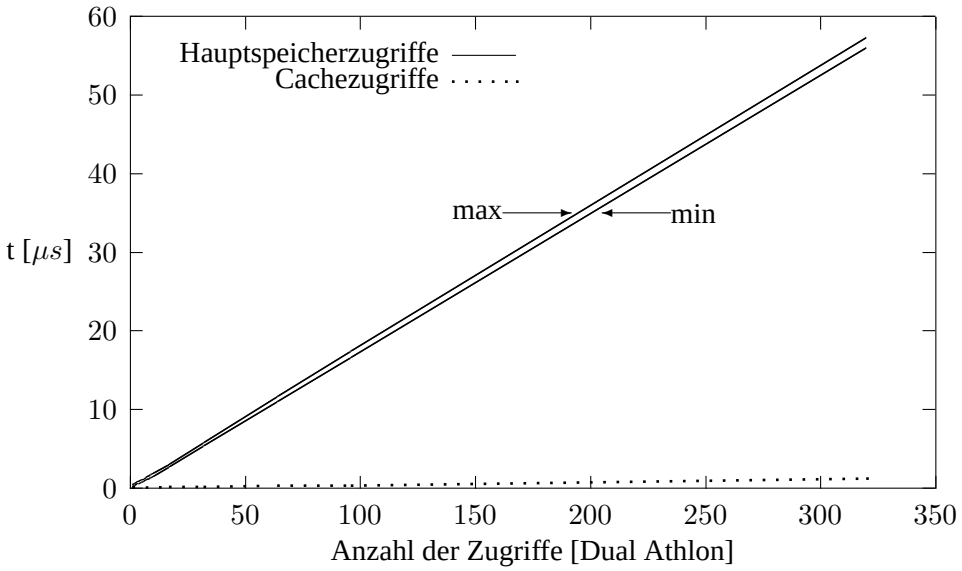


Abbildung 4.3: Hauptspeicherzugriffe vs. Cachzugriffe

Die Verlängerung der Ausführungszeit bei Hauptspeicherzugriffen gegenüber Cachezugriffen beträgt sowohl für das Athlon System als auch für das Opteron System im schlimmsten Fall ca. 5000%. Die einzelnen Messdaten sind in Tabelle 4.5 aufgeführt.

	$t_{RAM,BCET}$	$t_{RAM,WCET}$	t_{Cache}	Verhältnis
Athlon	$0{,}16\mu s$	$0{,}18\mu s$	$0{,}003\mu s$	5160%
Opteron	$0{,}12\mu s$	$0{,}13\mu s$	$0{,}003\mu s$	4900%

Tabelle 4.5: Zugriffszeiten auf Hauptspeicher und Cache

Quick-Sort Sortieralgorithmus

Um die entstehende Laufzeitverlängerung beurteilen zu können, die sich aufgrund von Cache-Misses bei “realer” Software ergibt, werden in dieser Messung 16k Worte sortiert. Ein Wort besteht dabei aus vier Byte. Im einen Fall werden vor der Messung die Prozessorcaches mit modifizierten Daten gefüllt, um das worst-case Szenario herzustellen; im anderen Fall befinden sich der Code und die zu sortierenden Worte komplett im Cache der CPU. Während des Sortierens wird das Programm nicht unterbrochen und es erfolgt keine weitere Modifikation der Caches.

Die beschriebene Messung wird auf dem Dual-Athlon System durchgeführt. Das Messergebnis ist in Abbildung 4.4 dargestellt. Auf der Abszisse ist die benötigte Zeit zum Sortieren der 16k Worte angetragen, die Ordinate spiegelt in einem logarithmischen Maßstab die relative Häufigkeit des Auftretens einzelner Messwerte wider. Insgesamt werden pro Messreihe 5000 Messungen durchgeführt.

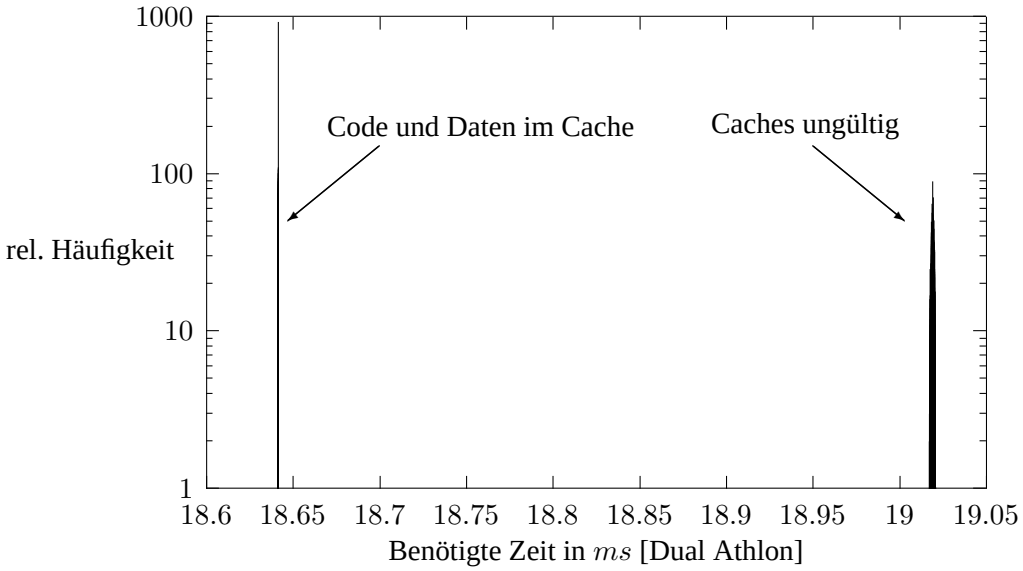


Abbildung 4.4: Sortieren von 16k Worten

Als Ergebnis lässt sich festhalten, dass sich die Ausführungszeit in diesem Fall nur gering erhöht, obwohl 64 KByte aus dem Hauptspeicher während der Ausführung des Sortieralgorithmus geladen werden müssen. Grund hierfür ist, dass die Daten zwar initial aus dem Hauptspeicher geladen werden, das eigentliche Sortieren aber anschließend vollständig im Cache der jeweiligen CPU abläuft.

Für Realzeitsysteme bedeutet dies, dass der Zustand der Caches bei der Ermittlung maximaler Ausführungszeiten eine wesentliche Rolle spielt. Insbesondere die Art und Ursache auftretender Cacheverdrängungen im Code der Realzeitsoftware müssen bei der Abschätzung maximaler Laufzeiten berücksichtigt werden.

4.2.2 Weitere Beschleunigungsmechanismen

Pipelines und Sprungvorhersage

Mit Hilfe von Pipelines ist der Prozessor in der Lage, mehrere Befehle parallel zu verarbeiten. Da die Bearbeitung eines Maschinenbefehls aus mehreren Phasen besteht, wie beispielsweise der Hol-, Dekodier- und Ausführungsphase, kann eine CPU mehrere Befehle in verschiedenen Verarbeitungsstufen parallel ausführen.

Da ein Befehl einige Stufen der Pipeline durchlaufen muss bis sein Ergebnis feststeht, wird die Pipeline auf Verdacht mit neuen Befehlen gefüllt. Die *Sprungvorhersage* versucht hier, das Ergebnis von bedingten Sprüngen zu bestimmen, damit die Pipeline mit den Instruktionen des vorhergesagten Programmzweigs gefüllt werden kann.

Wird ein Sprung falsch vorausgesagt, dann müssen alle Befehle des fälschlich angenommen Programmzweigs aus der Pipeline entfernt werden. Anschließend wird die Codierung der Befehle im korrekten Zweig des Programms fortgesetzt. Je nach Länge der Pipeline — der Opteron verwendet eine 12-stufige Pipeline, der Pentium 4 eine 20-stufige — wird die Ausführung der Software entsprechend verzögert. Die Zeitstrafen für einen falsch vorhergesagten Sprung bewegen sich in der Größenordnung von einigen Nanosekunden.

Simultaneous Multithreading

Bei Simultaneous Multithreading (SMT) gibt sich ein Prozessor als mehrere logische CPUs aus. Jede logische CPU erhält dabei einen eigenen Registersatz und einen eigenen APIC. Die Größe der Caches, sowie die Anzahl der Rechenwerke bleibt im Gegensatz zu den Multi-Core CPUs unverändert — die logischen Prozessoren verwenden diese gemeinsam. Mit SMT lässt sich eine bessere Auslastung der Rechenwerke einer CPU erzielen. Den dadurch resultierenden Performancegewinn gibt die Firma Intel für ihre SMT-Prozessoren[4)] mit 35% an.

Für Realzeitsysteme ist SMT nicht geeignet, da hier die Ausführungszeit der Maschinenbefehle von der Aktivität der anderen logischen Prozessoren abhängt. Deshalb treten aufgrund von SMT Laufzeitschwankungen bei der Ausführung von Software auf. Da die logischen Prozessoren zusätzlich den Cache gemeinsam verwenden, werden diese Laufzeitschwankungen verstärkt. Insbesondere können andere logische Prozessoren Cachelines verdrängen, so dass zur Abschätzung von worst-case Laufzeiten grundsätzlich von modifizierten Cachelines ausgegangen werden muss.

4.2.3 Zusammenfassung

Die Untersuchungen zeigen, dass sich die Pipelines und die Sprungvorhersage nur minimal auf Laufzeitschwankungen von Realzeitsoftware auswirken. Die resultierende Verzögerung aufgrund eines falsch vorhergesagten Sprungs und dem damit verbundenen Verwerfen einzelner Operationsergebnisse in der Pipeline bewegt sich im Nanosekundenbereich. Für die Abschätzung der Realzeiteigenschaften der PC-Architektur — insbesondere für die Untersuchung der Auswirkungen der Peripherieanbindung — müssen diese Einflüsse nicht weiter betrachtet werden. Gleiches gilt für SMT; diese Technik ist für harte Realzeitsysteme ungeeignet und wird daher im Folgenden nicht weiter untersucht.

Der Zustand der Caches hat dagegen einen großen Einfluss auf die Laufzeit von Realzeitsoftware. Greift ein Prozessor auf den Hauptspeicher zu, kommt es zu großen Schwankungen in den Ausführungszeiten. Die Laufzeitverlängerung beträgt in Abhängigkeit der Struktur des Codes bis zu 5000%. Da Peripheriegeräte im Zuge von DMA-Transfers auf den Hauptspeicher zugreifen, kann dies zum einen zu verlängerten Zugriffszeiten führen, zum anderen ist eine Invalidierung einzelner Cachelines möglich.

[4)] Intel vermarktet unter dem Schlagwort Hyper-Threading die SMT-Technik. Bei Hyper-Threading gibt sich eine physikalische CPU als zwei logische CPUs aus.

Im folgenden Abschnitt erfolgt die Untersuchung der Anbindung von Speicher und Peripherie an Multiprozessorsysteme. Insbesondere werden hier die Einflüsse untersucht, die aufgrund paralleler Zugriffe von Prozessoren und Peripheriegeräten auf gemeinsam genutzte Ressourcen entstehen.

4.3 Einflüsse durch Anbindung von Speicher und Peripherie

Im vorhergehenden Abschnitt wurden die Einflüsse der verschiedenen Beschleunigungsmechanismen diskutiert, die von den Prozessoren verwendet werden. Dabei zeigte sich, dass insbesondere die Caches einen großen Einfluss auf die Laufzeit von Realzeitsoftware haben. Da eine CPU aber auch mit ihrer Umwelt kommunizieren muss, werden die hieraus resultierenden Aspekte für Realzeitsysteme in diesem Abschnitt diskutiert. Insbesondere wird die Anbindung des Hauptspeichers an die Prozessoren bzw. an den Chipsatz untersucht, um die Auswirkung der Peripherieanbindung für Realzeitsysteme zu erörtern. Die gewonnenen Erkenntnisse bilden die Grundlage für die in Kapitel 5 vorgeschlagene Softwarearchitektur.

Die Prozessoren in einem Multiprozessorsystem werden über den *Chipsatz* an Hauptspeicher und Peripherie angebunden. Über den Chipsatz werden die Prozessoren und der Hauptspeicher mit den PCI-Bussen und dem — nicht immer vorhandenen — *Accelerated Graphics Port (AGP)* verbunden. Über die PCI-Busse erfolgt die Anbindung weiterer Peripheriegeräte.

Multiprozessorsysteme auf PC-Basis können als SMP- oder als NUMA-System aufgebaut sein. Aus Sicht der Software ist die Architektur des Multiprozessorsystems transparent. Wird diese Architektur jedoch bei der Konstruktion eines Realzeitsystems nicht berücksichtigt, sind große Schwankungen in der Laufzeit der Realzeitsoftware die Folge. Wegen der relativ großen Unterschiede beider Architekturen werden diese separat diskutiert: In Abschnitt 4.3.1 wird die SMP-Architektur untersucht. Insbesondere wird hier auf die zentrale Stellung der *Host-Bridge* in einem SMP-System eingegangen. In Abschnitt 4.3.2 werden die Realzeiteigenschaften der NUMA-Architektur erörtert und der *HyperTransport* Verbindungsstandard diskutiert.

Der PCI-Bus ist der zentrale Peripheriebus in einem Multiprozessorsystem. Der PCI-Bus ist das Bindeglied zwischen der Host-Bridge und den Peripheriegeräten und wird ausführlich in Abschnitt 4.3.3 behandelt.

Über den PCI-Bus werden im Allgemeinen weitere Peripheriebusse und Peripheriegeräte angebunden. Mit dieser Thematik befasst sich Kapitel 4.3.4. Die Interruptlaufzeiten in einem Multiprozessorsystem werden in Abschnitt 4.3.5 untersucht.

4.3.1 SMP-Architektur

Kennzeichen der SMP-Architektur ist, dass alle Prozessoren gleichberechtigt auf *einen gemeinsam verwendeten* Speicher zugreifen. Dabei ist der Aufbau der SMP-Architektur vollständig

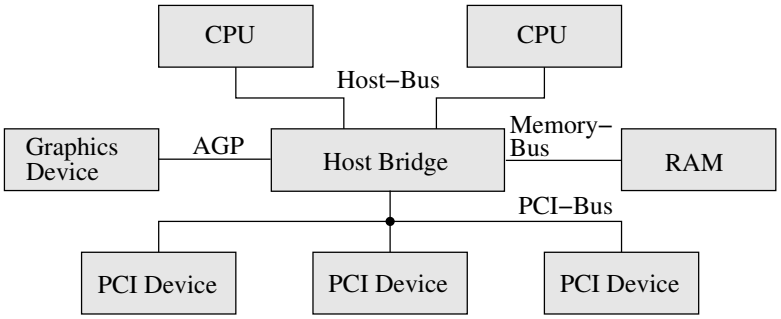


Abbildung 4.5: SMP-Architektur

symmetrisch: In einem SMP-System sind nur Prozessoren gleichen Typs vorhanden, die alle die gleichen Rechte besitzen. Jeder Prozessor kann mit jedem anderen kommunizieren und Interrupts aus jeder möglichen Quelle erhalten.

Der schematische Aufbau eines SMP-Systems ist in Abbildung 4.5 für ein Dual-CPU Multiprozessorsystem dargestellt: Die Prozessoren sind über den Host-Bus mit der Host-Bridge verbunden. Oft wird der Host-Bus auch als *Frontside-Bus (FSB)* oder als *System-Bus* bezeichnet. Diese Verbindung kann, wie in Abbildung 4.5 dargestellt, eine Punkt-zu-Punkt Verbindung oder ein Bus sein.[5)]

Die Host-Bridge ist das zentrale Verbindungselement in einem SMP-basierten Multiprozessorsystem und Bestandteil des Chipsatzes. Sie verbindet die Prozessoren mit dem Speicher (RAM) und dem Grafikadapter. Zusätzlich bietet die Host-Bridge eine Anbindung für einen oder mehrere schnelle PCI-Busse. Hierzu sind wesentliche Funktionalitäten für PCI, wie beispielsweise PCI-Arbiter, in der Host-Bridge integriert (siehe hierzu auch Abschnitt 4.3.3). Um einen höheren Durchsatz zu erreichen, sind in der Host-Bridge auch mehrere Pufferspeicher enthalten. Insbesondere sind hier der CPU-to-DRAM, der CPU-to-PCI und der PCI-to-DRAM Pufferspeicher zu nennen. Diese Pufferspeicher können die Daten einiger (weniger) Übertragungszyklen zwischenspeichern.

Meist wird die beschriebene Funktionalität der Host-Bridge von nur einem Hardwarebaustein zur Verfügung gestellt. Es existieren allerdings auch Chipsätze, bei denen die Anbindung mehrerer PCI-Busse an die Host-Bridge über separate Bausteine realisiert ist, die untereinander über proprietäre Punkt-zu-Punkt Verbindungen kommunizieren. Zur Vollständigkeit sei an dieser Stelle noch die *South-Bridge* erwähnt. Die South-Bridge wird in SMP-Systemen über den PCI-Bus bzw. ebenfalls über proprietäre Punkt-zu-Punkt Verbindungen an die Host-Bridge angeschlossen.

In den folgenden Abschnitten sollen die Einflüsse der SMP-Architektur auf Realzeitsoftware untersucht werden. Hierzu werden zunächst die Ursachen möglicher Latenzzeiten in einem SMP-System in Abschnitt 4.3.1.1 diskutiert. In den darauffolgenden Abschnitten wird anhand von Messungen die Größenordnung dieser Beeinflussungen bestimmt.

[5)] Multiprozessorsysteme mit Prozessoren der Firma Intel verwenden hier den AGTL+ Bus. Prozessoren der Firma AMD sind mit EV6 Punkt-zu-Punkt Verbindungen an die Host-Bridge angeschlossen.

4.3.1.1 Latenzzeiten in SMP-Systemen

Bei SMP-basierten Multiprozessorsystemen beeinflussen sich die einzelnen Komponenten in vielerlei Hinsicht. Soll eine SMP-Architektur für Realzeitaufgaben eingesetzt werden, so können prinzipiell die folgenden Architekturmerkmale Ursache für das Auftreten von Latenzzeiten sein:

- Beim Zugriff auf den Host-Bus muss der zugreifende Prozessor warten, bis er den Bus zugeteilt bekommt. Hier hängt es vom verwendeten Arbitrierungsverfahren ab, in welcher Größenordnung sich die entstehenden Latenzzeiten bewegen. Falls die Host-Bridge zu jeder CPU eine Punkt-zu-Punkt Verbindung besitzt, können hier keine Latenzzeiten auftreten.

- Die Host-Bridge ist das zentrale Verbindungselement in einem SMP-basierten Multiprozessorsystem. Sämtliche Kommunikation der Prozessoren mit ihrer Umwelt durchläuft die Host-Bridge. Insbesondere erfolgt hier die Anbindung des Hauptspeichers, des Grafikadapters und die Anbindung der PCI-Busse. Auftretende Latenzzeiten werden dabei von der Architektur der Host-Bridge bestimmt. Wesentliche Parameter sind die Größe vorhandener Pufferspeicher, die zeitliche Auswirkung parallel ablaufender Datentransfers und die von der Host-Bridge verwendeten Arbitrierungsverfahren. Bei letzteren ist die Reihenfolge der einzelnen Komponenten beim Zugriff auf eine bestimmte Ressource und die maximale Anzahl erlaubter Datentransfers von Interesse.

- Zusätzlich werden die Ausführungszeiten der Prozessoren von der Reaktionszeit der angesprochenen Komponenten beeinflusst. Zu nennen sind hier insbesondere die auftretenden Latenzzeiten beim Zugriff auf den Hauptspeicher und die Latenzzeit, bis der PCI-Bus den durchzuführenden Befehl verarbeiten kann.

Problem bei der Bestimmung der aufgeführten Latenzzeiten ist, dass die Spezifikationen des Host-Busses und der Host-Bridge nicht vollständig zugänglich sind. Deshalb ist es nicht möglich, anhand dieser Spezifikationen die maximale Ausführungszeit für bestimmte Zugriffsvarianten und die zeitlichen Auswirkungen parallel stattfindender Datenübertragungen zu bestimmen. Auch über die Größe vorhandener Pufferspeicher werden in der Literatur meist keine genauen Angaben gemacht. Die von einer Host-Bridge verwendeten Protokolle und Arbitrierungsverfahren sind ebenfalls nicht dokumentiert.

Ein weiteres Problem bei der allgemeinen Bestimmung von Latenzzeiten in SMP-basierten Multiprozessorsystemen ist, dass hier eine Vielzahl unterschiedlicher Chipsätze vorhanden ist, die alle ein unterschiedliches zeitliches Verhalten aufweisen. Weiterhin können sich von einer Revision zur nächsten Revision eines Chipsatzes einige interne Parameter ändern, so dass sich prinzipiell gleiche Chipsätze unterschiedlich verhalten können. Zusätzlich bieten viele Chipsätze die Möglichkeit, innerhalb gewisser Schranken die Werte einiger zentraler Konfigurationsparameter zu verändern — wie beispielsweise den Takt des Host-Busses oder die Timing-Parameter des Hauptspeichers. Auch dies führt zu unterschiedlichem zeitlichen Verhalten, so dass bei identischen Chipsätzen unterschiedliche Ausführungszeiten auftreten können.

Weiterhin sind in PC-Systemen viele Bausteine miteinander kombinierbar. Insbesondere kann eine North-Bridge mit verschiedenen South-Bridges verbunden werden; an einen Chipsatz können oft unterschiedliche Speichertypen angebunden werden. Beispielsweise unterscheiden sich die Speicherbausteine in ihren Timing-Parametern, in der Anzahl der Paritätsbits, in der maximal möglichen Datenrate, etc. Manchmal hängt das zeitliche Verhalten des Hauptspeichers auch davon ab, wie die einzelnen Speicherslots bestückt sind: Bietet der Chipsatz zwei separate Speicherkanäle, so müssen hier die entsprechenden Slots mit identischem Speicher bestückt werden. Dadurch kann der Speicher im *Interleaved-Modus* betrieben werden, wodurch sich die Datenübertragungsrate zwischen Host-Bridge und Speicher verdoppeln lässt.

In SMP-Systemen wird das zeitliche Verhalten beim Zugriff auf eine bestimmte Systemressource auch davon beeinflusst, ob und mit welchen Geräten die einzelnen Slots des AGP und der PCI-Busse bestückt sind. Hier hängt es von den einzelnen Anforderungen der jeweiligen Geräte ab, ob und in welcher Weise sich diese auf mögliche Latenzzeiten von Realzeitsoftware auswirken. Beispielsweise sind in einem Computersystem durch den AGP keine Latenzzeiten zu erwarten, falls in dem entsprechenden System die Grafikkarte nicht an AGP, sondern an PCI angeschlossen wird. Weiterhin ist auch damit zu rechnen, dass sich ein an das System angebundener SCSI-Controller aufgrund des höheren Datenaufkommens in einem größeren Ausmaß auf mögliche Latenzzeiten auswirken wird als beispielsweise der Controller einer Soundkarte.

Die einzelnen Probleme, die bei der Bestimmung der Latenzzeiten in SMP-basierten Multiprozessorsystemen auftreten, lassen sich somit wie folgt zusammenfassen:

- Die Spezifikationen der einzelnen Komponenten des Chipsatzes sind nicht zugänglich.
- Der Entwicklungszyklus bei Prozessoren, Speicher und Chipsätzen ist sehr schnell, so dass es nicht möglich ist, hier jedes Modell separat und schritthaltend zu untersuchen.
- Die zeitlichen Parameter eines Chipsatzes können sich von Revision zu Revision ändern, insbesondere auch durch Einspielen einer neuen Firmware.
- Einige Parameter, die Laufzeiten von Software beeinflussen, können im BIOS des Rechensystems verändert werden.
- Falls die Hardwarebestückung des Multiprozessorsystems geändert wird, können sich die maximalen Laufzeiten von Realzeitsoftware verändern.

Selbst wenn die Spezifikationen der einzelnen Komponenten des Multiprozessorsystems verfügbar wären, wäre die Bestimmung maximaler Laufzeiten von Software mittels Modellierung mit unverhältnismäßig großem Aufwand verbunden. Hier müssten die Laufzeiten für jeden Chipsatz mit jeder möglichen Hardware- und Firmwarekombination (Prozessoren, Speicher, Peripheriegeräte) ermittelt werden können — ein Aufwand, der in der Praxis nicht realisierbar ist. Eine Möglichkeit, dieses Problem zu lösen, wäre die Untersuchung eines *bestimmten* Multiprozessorsystems, bei dem alle Parameter bekannt sind. Dann müsste ausschließlich dieses System für Realzeitaufgaben eingesetzt werden; eine nachträgliche Modifikation zentraler Komponenten dürfte nicht vorgenommen werden. Zusätzlich könnte aufgrund des aufwändigen Modellierungsprozesses nur Hardware in dem Computersystem verbaut werden, die bereits län-

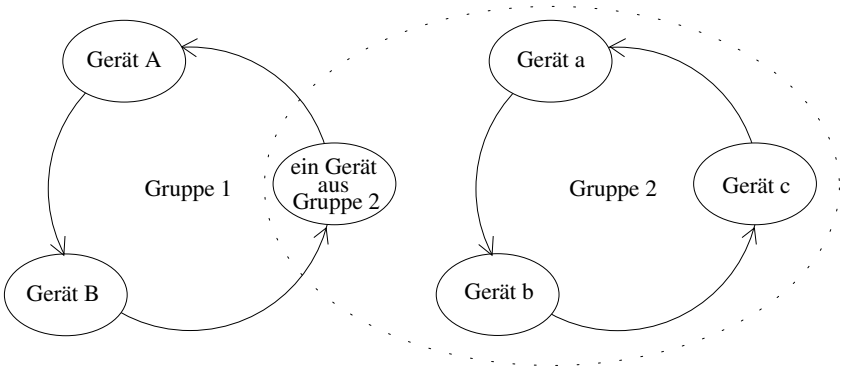


Abbildung 4.6: Kaskadierte Round-Robin-Arbitrierungsverfahren

gere Zeit auf dem PC-Markt verfügbar ist. Zusammengefasst würde diese Methode zu großen Einschränkungen verbunden mit einem hohen Realisierungsaufwand führen.

Deshalb wird ein anderer Ansatz gewählt, um die Realzeiteigenschaften eines SMP-basierten Multiprozessorsystems bestimmen zu können: Um die Einflüsse der verschiedenen Komponenten auf die Laufzeit von Realzeitsoftware ermitteln zu können, werden die auftretenden Latenzzeiten anhand von *Messungen* bestimmt. Dabei werden insbesondere die Einflüsse des Host-Busses und der Host-Bridge untersucht, sowie die Einflüsse, die durch die Anbindung des Hauptspeichers und der Peripheriegeräte hervorgerufen werden. In den folgenden Abschnitten wird beschrieben, *welche* Messungen durchgeführt werden müssen und *wie* diese durchzuführen sind. Um die Größenordnung der Beeinflussungen innerhalb einer SMP-Architektur abschätzen zu können, sind die Messergebnisse aufgeführt, die man bei Verwendung eines Dual-Athlon Multiprozessorsystems erhält.

4.3.1.2 Bestimmung der Latenzzeiten anhand von Messungen

Um die auftretenden Latenzzeiten bei einem SMP-basierten Multiprozessorsystem anhand von Messungen bestimmen zu können, müssen die Auswirkungen möglicher Ursachen für das Auftreten maximaler Laufzeiten untersucht werden. Da mit Ausnahme einiger weniger Ereignisse der PMCs nur die Laufzeiten von Software gemessen werden können, müssen sich bei jeder Messung die zu untersuchenden Komponenten möglichst deterministisch verhalten. Die ermittelten Messwerte müssen anschließend auf ihre Plausibilität überprüft werden und können anschließend beispielsweise für die Durchführung eines Realzeitnachweises herangezogen werden.

Prinzipiell kann davon ausgegangen werden, dass sämtliche in PC-Systemen anzutreffende Arbitrierungsverfahren *fair* sind. Dabei wird angenommen, dass kein Gerät innerhalb des Rechensystems ein anderes dazu zwingen kann, seine Daten *nie* zu übertragen. Einzig die Dauer, bis eine bestimmte Anzahl Daten übertragen wird, hängt von der Aktivität der anderen Teilnehmer ab. Hier kommen *Round-Robin-Verfahren* zum Einsatz, bei den die verschieden Teilnehmergruppen — wie beispielsweise CPUs, PCI-Geräte, AGP — unterschiedlich gewichtet sind. Oft werden auch Verfahren verwendet, bei denen mehrere Round-Robin-Arbitrierungsverfahren

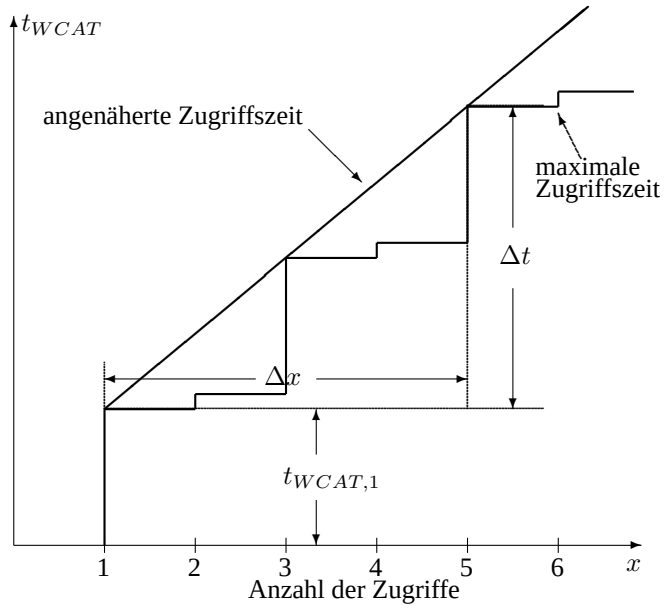


Abbildung 4.7: Linearität maximaler Zugriffszeiten

kaskadiert sind, wie in Abbildung 4.6 gezeigt: In diesem Beispiel wird der Zugriff auf eine gemeinsam genutzte Ressource reihum zwischen den Geräten aus Gruppe 1 und jeweils einem Gerät aus Gruppe 2 vergeben.

Prinzipiell kann zur Abschätzung der maximalen Dauer aufeinanderfolgender Zugriffe ein *lineares* Laufzeitverhalten approximiert werden; wobei die Linearität in Abhängigkeit von der maximal möglichen Anzahl aufeinanderfolgender Datentransfers des zu untersuchenden Gerätes mehrere, verschieden breite Stufen aufweist. Dies folgt unmittelbar aus den in PC-Systemen verwendeten Round-Robin-Verfahren: Falls alle Geräte und Prozessoren in dem Computersystem mit ihrer maximal möglichen Rate Daten senden und empfangen, ergibt sich ein sich wiederholendes Arbitrierungsmuster.

In Abbildung 4.7 ist die approximierte maximale Zugriffszeit t_{WCAT} in Abhängigkeit der Anzahl x der auf eine Ressource sequentiell und kontinuierlich durchgeführten Zugriffe eines Gerätes aufgetragen. Diese maximale Zugriffszeit wird im Folgenden als *Worst-Case Access Time (WCAT)* bezeichnet. In dem dargestellten Fall beträgt die maximal benötigte Zeit bis zur Vollendung des *ersten* Zugriffs $t_{WCAT,1}$. Der folgende Zugriff kann direkt im Anschluss daran ausgeführt werden, da in dem illustrierten Fall das Gerät maximal zwei Zugriffe aufeinanderfolgend durchführen darf. Nach dem zweiten Zugriff kommt das nächste Gerät an die Reihe und das ursprüngliche muss warten, bis es den nächsten Transfer initiieren darf.

Um die maximale Zugriffszeit für eine bestimmte Anzahl an Zugriffen einer CPU auf eine Ressource bestimmen zu können, wird diese Zeit mit Hilfe einer *Geradengleichung* angenähert, die zu einer geringen Überabschätzung der maximalen Ausführungszeiten führt. Durch Messung müssen hier der *Offset* und die *Steigung* dieser Geraden gemessen werden. Hierzu muss zunächst die Zeit $t_{WCAT,1}$ ermittelt werden, die maximal bis zur Vollendung *eines* Zugriffs auf eine Ressource benötigt wird. Mit einer weiteren Messung wird anschließend die

Steigung der Geraden bestimmt. Dabei muss für eine hinreichend große Anzahl x an Ressourcenzugriffen ebenfalls die maximale Zugriffszeit gemessenen werden. Um zu vermeiden, dass man aufgrund möglicher Burst-Transfers die Zugriffszeit zu optimistisch einschätzt, kann die Anzahl x der durchgeführten Messungen geringfügig modifiziert werden. Für genügend große x sind die Auswirkungen möglicher Burst-Transfers vernachlässigbar.

Somit lässt sich die benötigte maximale Zugriffszeit t_{WCAT} in Abhängigkeit der Anzahl x der kontinuierlich durchgeführten Zugriffe, der maximal benötigten Zeit $t_{WCAT,1}$ für den ersten Zugriff und der Steigung $m_{WCAT} = \Delta t/\Delta x$ wie folgt angeben:

$$t_{WCAT}(x) = t_{WCAT,1} + m_{WCAT}(x-1); \qquad x \in \mathbb{N} \tag{4.8}$$

In den folgenden Abschnitten wird diese Gleichung verwendet, um die Auswirkungen des Host-Busses und der Host-Bridge auf das Laufzeitverhalten von Realzeitsoftware zu beschreiben. Hierzu werden für verschiedene Belastungsszenarien die jeweiligen Parameter der Gleichung (4.8) bestimmt und zueinander in Relation gesetzt. Insbesondere wird in den folgenden Abschnitten darauf eingegangen, welche gegenseitigen Beeinflussungen in einem SMP-System prinzipiell auftreten können, welche davon für Realzeitaufgaben relevant sind und wie die entsprechenden Parameter für die Gleichung (4.8) ermittelt werden können.

4.3.1.3 Auswirkungen von Host-Bus und Host-Bridge

Um die Realzeiteigenschaften einer SMP-Architektur anhand von Messungen bestimmen zu können, müssen zunächst alle Beeinflussungsmöglichkeiten, die bei der Verwendung einer solchen Architektur auftreten können, erörtert werden. Aus der Menge aller möglichen Beeinflussungen müssen anschließend jene untersucht werden, die prinzipiell im Realzeitbetrieb auftreten können. Ziel ist es, für alle möglichen Belastungsszenarien die maximale Zugriffszeit für eine bestimmte Anzahl aufeinanderfolgend durchzuführender Ressourcenzugriffe eines Prozessors angeben zu können.

Falls eine SMP-Architektur für Realzeitaufgaben eingesetzt wird, können sich aufgrund parallel ablaufender und konkurrierender Ressourcenanforderungen Beeinflussungen einer CPU beim Ressourcenzugriff ergeben. Prinzipiell sind überall dort Beeinflussungen zu erwarten, wo *überlappende* Datenpfade vorhanden sind. Durchlaufen diese Datenpfade einen Baustein, ist die Frage zu klären, ob hier diese Datenpfade ggf. *gleichzeitig* — also parallel — ausgeführt werden können, oder ob aufgrund eines internen Arbitrierungsverfahrens immer nur ein Teilnehmer auf eine Ressource zugreifen darf. Im letzteren Fall führt dies zur Entstehung von Latenzzeiten.

Sollen die Realzeiteigenschaften einer SMP-Architektur untersucht werden, muss davon ausgegangen werden, dass *alle* über die Host-Bridge angeschlossenen Funktionseinheiten eine Datenübertragung durchführen möchten. Die zu betrachtenden Funktionseinheiten sind dabei alle Baugruppen, die direkt an den Host-Bus und an die Host-Bridge angeschlossen sind. Dies sind alle über den Host-Bus angeschlossenen CPUs, AGP (falls vorhanden) und alle PCI-Busse, die von der Host-Bridge bereitgestellt werden. Falls die South-Bridge eine separate Verbindung zur

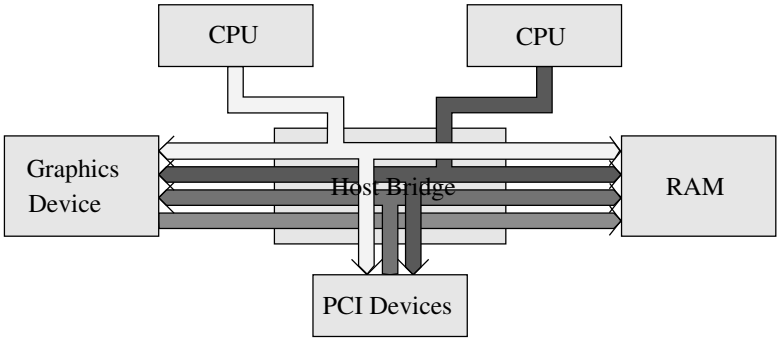


Abbildung 4.8: Gegenseitige Beeinflussungsmöglichkeiten bei einer SMP Architektur

Host-Bridge besitzt — falls diese also nicht über einen PCI-Bus angeschlossen ist — kann auch die South-Bridge Ursache für das Auftreten von Latenzzeiten sein.

In Abbildung 4.8 sind für eine SMP-Architektur alle Datenpfade eingetragen, die Ursache für das Auftreten von Latenzzeiten sein können. Aus Sicht der Realzeitsoftware können Latenzzeiten dabei prinzipiell bei Zugriffen eines Prozessors auf den Hauptspeicher, auf den Grafikadapter, oder auf ein Peripheriegerät entstehen. Dies wird durch den hell gezeichneten Pfeil in Abbildung 4.8 verdeutlicht. Die Hauptspeicherzugriffe resultieren dabei im Allgemeinen indirekt aus der Code- und Datenstruktur der Realzeitsoftware. Zugriffe auf den Speicher des Grafikadapters oder auf den Speicher eines Peripheriegerätes werden direkt von einer Softwareroutine angestoßen. Diese Zugriffe können dabei als Portzugriffe oder über Memory-Mapped-I/O durchgeführt werden.

Die Zugriffe sind Laufzeitschwankungen unterworfen, falls parallel hierzu andere Prozessoren oder Peripheriegeräte ebenfalls Zugriffe auf andere Peripheriegeräte, auf den Grafikadapter, oder auf den Hauptspeicher durchführen. Dabei spielt es keine Rolle, ob diese Zugriffe physikalisch *parallel* durchgeführt werden können, oder ob die Zugriffe mit Hilfe eines Schedulingverfahrens *sequentiell* ausgeführt werden. Bei der sequentiellen Ressourcenzuteilung muss jedoch aufgrund der gemeinsam genutzten Ressourcen mit höheren Latenzzeiten gerechnet werden. Prinzipiell sind die folgend aufgeführten Zugriffe möglich, wobei sich diese aus allen möglichen Datenpfaden, die über die Host-Bridge verlaufen, zusammensetzen. Diese Zugriffe sind in Abbildung 4.8 durch dunkle Pfeile dargestellt:

- Andere Prozessoren können — wie auch der Prozessor, auf dem die Realzeitapplikation ausgeführt wird — auf den Hauptspeicher, auf den Grafikadapter und auf Peripheriegeräte zugreifen. Dabei wird durch die Architektur des jeweils verwendeten Chipsatzes bestimmt, ob diese Transfers ggf. parallel ausgeführt werden können. Beispielsweise könnte im dargestellten Fall eine CPU auf den Speicher des Grafikadapters zugreifen und eine andere CPU auf den Hauptspeicher, ohne dass sich überlappende Datenpfade ergeben. Falls im Gegensatz hierzu der Chipsatz nur einen Host-Bus verwendet, resultieren hieraus überlappende Datenpfade, so dass prinzipiell mit höheren Zugriffszeiten gerechnet werden muss.

- Parallel zu den Zugriffen der Prozessoren können alle an der Host-Bridge angeschlossenen PCI-Busse — im dargestellten Fall ist nur ein PCI-Bus an die Host-Bridge angebunden — auf den Speicher des Grafikadapters oder auf den Hauptspeicher zugreifen. Zugriffe auf den Speicher der Grafikadapters werden beispielsweise von TV-Karten durchgeführt, die ihr Bild direkt im Grafikspeicher ablegen.

- Der Grafikadapter kann im Zuge von 3D-Anwendungen auf den Hauptspeicher zugreifen, falls er beispielsweise eine Textur laden muss. Diese Zugriffe können indirekt von Software angestoßen werden. Falls jedoch der Speicher auf dem Grafikadapter knapp wird, kann der Grafikadapter auch selbstständig während einer 3D-Anwendung Texturen nachladen.

- Bei manchen Chipsätzen ist die South-Bridge über eine separate Verbindung an die Host-Bridge angebunden (nicht dargestellt in Abbildung 4.8). Da die South-Bridge eine Reihe von Peripheriegeräten bzw. Gerätecontrollern integriert, die teilweise eine hohe I/O-Last erzeugen, kann auch ein Gerät der South-Bridge im Zuge eines DMA-Burst-Transfers lesend oder schreibend auf den Hauptspeicher zugreifen.

4.3.1.4 Bestimmung der relevanten Messungen

Im vorhergehenden Abschnitt wurde beschrieben, mit welchen gegenseitigen Beeinflussungen in einem SMP-basierten Multiprozessorsystem prinzipiell gerechnet werden muss. Im Gegensatz dazu soll in diesem Abschnitt diskutiert werden, welche gegenseitigen Beeinflussungen tatsächlich im Realzeitbetrieb auftreten können. Anschließend werden die Auswirkungen dieser Beeinflussungen auf Realzeitsoftware anhand von Messungen untersucht. In dem nächsten Abschnitt wird darauf eingegangen, auf welche Art und Weise die entsprechenden Messungen durchzuführen sind.

Bei der Bestimmung der relevanten Messungen, die zur Abschätzung der Realzeiteigenschaften des Multiprozessorsystems durchzuführen sind, muss davon ausgegangen werden, dass *alle* an die Host-Bridge und an den Host-Bus angeschlossenen Komponenten mit ihrer jeweils maximal möglichen Rate Daten senden und empfangen. Dieses Szenario entspricht damit dem anzunehmenden Worst-Case. Jedoch gibt es einige Konstellationen, die *keinen* Einfluss auf die Zugriffszeiten von Software haben und damit nicht untersucht werden müssen:

- PCI-Busse, die von der Host-Bridge zur Verfügung gestellt werden, haben nur dann einen Einfluss auf die Zugriffszeit von Realzeitsoftware, wenn an diesen Bussen tatsächlich Geräte angeschlossen sind. Somit kann beispielsweise das Umstecken eines PCI-Gerätes in den Slot eines anderen PCI-Busses zu verbesserten Latenzzeiten führen.

- Peripheriegeräte, und damit an die Host-Bridge angeschlossene PCI-Busse, wirken sich nur dann auf die Zugriffszeit von Realzeitsoftware aus, falls diese vom Betriebssystem verwendet werden. Beispielsweise liefert ein SCSI-Controller, an dem keine SCSI-Geräte angeschlossen sind und der auch nicht von Treibern des Betriebssystems angesprochen wird, keinen Beitrag zu maximalen Zugriffszeiten von Realzeitsoftware.

- Peripheriegeräte greifen in Realzeitsystemen im Allgemeinen nicht auf den Speicher des Grafikadapters zu. Diese Funktionalität wird nur von Video- und TV-Karten verwendet.

Die durchzuführenden Messungen lassen sich weiter einschränken, falls aufgrund der symmetrischen Eigenschaften des Multiprozessorsystems davon ausgegangen wird, dass gleiche Komponenten von den in dem jeweiligen Chipsatz vorhandenen Arbitrierungsverfahren gleich gewichtet werden. Somit kann erwartet werden, dass gleiche Komponenten bei gleicher parallel stattfindender Last den gleichen Latenzzeiten unterworfen sind. Insbesondere kann angenommen werden, dass alle Prozessoren und alle von der Host-Bridge zur Verfügung gestellten PCI-Busse gleichen Typs von den verwendeten Arbitrierungsverfahren gleich gewichtet werden.

Die Anzahl der durchzuführenden Messungen kann zusätzlich verringert werden, wenn davon ausgegangen wird, dass bestimmte Belastungsszenarien im Realzeitbetrieb immer oder nie auftreten. Beispielsweise kann davon ausgegangen werden, dass andere Prozessoren im Zuge der Ausführung ihrer Programme immer auf den Hauptspeicher zugreifen können. Somit müssen die Messungen ohne parallele Hauptspeicherzugriffe anderer Prozessoren nicht durchgeführt werden. Im Gegensatz hierzu könnten auch andere Prozessoren zu bestimmten Zeiten daran gehindert werden, auf den Hauptspeicher zuzugreifen — in diesem Fall sind die entsprechenden Messungen durchzuführen. Auf diese Problematik wird nochmals in Kapitel 5 eingegangen.

Die verbleibenden Messungen müssen anschließend in allen Variationen durchgeführt werden. Insbesondere sind dies alle Messungen, bei denen überlappende Datenpfade vorhanden sind. Da alle Datenpfade in der Host-Bridge zusammenlaufen, muss geprüft werden, ob hier bei Datenpfaden, die keine gemeinsam verwendeten Schnittstellen oder Busse besitzen, Beeinflussungen auftreten.

4.3.1.5 Durchführung der relevanten Messungen

Im folgenden Abschnitt wird beschrieben, *wie* die entsprechenden Messungen durchzuführen sind. Ziel hierbei ist es, dem gewünschten worst-case Szenario so nahe wie möglich zu kommen.

Grundsätzlich muss bei der Durchführung der Messungen zur Abschätzung der Realzeiteigenschaften eines SMP-basierten Multiprozessorsystems darauf geachtet werden, dass nur die Belastungsszenarien auftreten, deren Einfluss auf Realzeitsoftware auch tatsächlich bestimmt werden soll. Insbesondere bedeutet dies:

- Komponenten, deren Einfluss auf maximale Zugriffszeiten bestimmt werden soll, übertragen mit ihrer maximal möglichen Rate auf den zu untersuchenden Datenpfaden. Falls beispielsweise der gegenseitige Einfluss zweier Prozessoren bei Zugriffen auf den Hauptspeicher untersucht werden soll, müssen während der Messung beide Prozessoren mit maximaler Geschwindigkeit auf den Hauptspeicher zugreifen. Zugriffe auf den Speicher von PCI-Geräten dürfen dann während der Messung nicht erfolgen.

- Alle verbleibenden Komponenten, deren Einfluss *nicht* in der jeweiligen Messung untersucht werden soll, dürfen während der Messung *keine* Datenübertragungen anstoßen.

Um dieses Szenario zu erreichen, müssen alle Komponenten zu Beginn jeder Messung so konfiguriert werden, dass sie das gewünschte Verhalten aufweisen (vergleiche hierzu auch Abschnitt 4.1.2). Die Messsoftware kann die Peripheriegeräte für die Dauer einer Messung daran hindern, Datentransfers durchzuführen. Gleichzeitig können andere CPUs ebenfalls eine definierte Funktionalität ausführen, wie beispielsweise der permanente Zugriff auf bestimmte Speicher- oder I/O-Bereiche. Zusätzlich können andere CPUs in den Zustand “Prozessorhalt” versetzt werden.

Sollen während einer Messung einige Komponenten mit ihrer maximal möglichen Rate Daten übertragen, so müssen diese Übertragungen vor bzw. während der jeweiligen Messung angestoßen werden. Im Falle der CPUs kann durch die Ausführung einer entsprechenden Softwareroutine die zu erzeugende Last relativ exakt konfiguriert werden. Da allerdings der Grafikadapter und die Peripheriegeräte selbstständig ihre Datentransfers durchführen, können diese Geräte nur indirekt zur Durchführung einer Datenübertragung angeregt werden:

- Bei der Untersuchung der Einflüsse durch AGP kann der Grafikadapter durch das Senden geeigneter Befehle dazu gezwungen werden, Texturen vom Hauptspeicher in seinen Speicher hineinzuladen. Durch permanentes Wiederholen der entsprechenden Befehle entsteht so eine definierte Last.
- Bei der Untersuchung der Einflüsse durch Datenübertragungen der an der Host-Bridge angeschlossenen PCI-Busse müssen die angebundenen Geräte zur Übertragung ihrer Daten angeregt werden. Beispielsweise kann ein IDE-Controller zur Übertragung von Daten angeregt werden, indem vor einer Messung ein DMA-Datentransfer zu einer Festplatte angestoßen wird. Alternativ bieten sich auch Netzwerkkarten zur Lasterzeugung an: Beim Erhalt geeigneter Pakete führen diese autonom Transfers zum Hauptspeicher durch.

Da nicht garantiert werden kann, dass bei jeder Messung der Worst-Case erreicht wird, sind die Messungen so lange zu wiederholen, bis sich die gemessene maximale Zugriffszeit nicht mehr ändert, falls die Anzahl der durchgeführten Messungen weiter erhöht wird.

4.3.1.6 Einschränkungen des verwendeten Verfahrens

Problem bei dem verwendeten Verfahren ist, dass während einer Messung die Last der zu untersuchenden Komponenten sehr genau konfiguriert werden muss. Greift beispielsweise während einer Messung eine CPU nicht mit ihrer maximal möglichen Rate auf den Hauptspeicher oder auf den Speicher eines Peripheriegerätes zu, werden die entsprechenden Zugriffszeiten zu optimistisch eingeschätzt. Dasselbe Problem tritt bei der Lasterzeugung durch PCI-Busse und durch AGP auf. Auch hier ist nicht immer gewährleistet, dass grundsätzlich mit der maximal möglichen Rate Daten gesendet bzw. empfangen werden.

Diese Problematiken können abgeschwächt werden, falls zum einen die Anzahl der durchgeführten Messungen erhöht wird — damit erhöht sich die Wahrscheinlichkeit, den Worst-Case zu erreichen. Zum anderen müssen die ermittelten Werte auf Plausibilität und Korrektheit überprüft werden. Weiterhin muss die Anzahl der untersuchten aufeinanderfolgenden Zugriffe so gering wie möglich gehalten werden, da so die Wahrscheinlichkeit steigt, bei jedem Zugriff

worst-case Bedingungen bei einer bestimmten Messung zu erreichen: Falls beispielsweise die maximale Zugriffszeit für einen Zugriff auf eine bestimmte Ressource ermittelt werden soll, dann wird mit hoher Wahrscheinlichkeit dieser eine Zugriff um das maximale Maß verzögert. Soll dagegen die maximale Ausführungszeit für sehr viele aufeinanderfolgende Zugriffe bestimmt werden, so ist die Wahrscheinlichkeit äußerst gering, dass *alle* diese Zugriffe jeweils um das maximal mögliche Maß verzögert ausgeführt werden.

Letztendlich kann davon ausgegangen werden, dass bestimmte worst-case Belastungsszenarien, die sich *nicht* künstlich während einer Messung erzeugen lassen, im Produktivbetrieb auch nicht auftreten werden. Insbesondere trifft dies für die Lasten zu, die durch PCI-Geräte verursacht werden. Hier wäre es sinnvoll, in den Gerätetreibern der einzelnen Peripheriegeräte einen Betriebsmodus vorzusehen, der das Peripheriegerät in einen Zustand versetzt, in dem es maximale Datenlast erzeugt.

4.3.1.7 Bestimmung der Realzeiteigenschaften eines Dual-Athlon

In dem folgenden Abschnitt sollen nun die Realzeiteigenschaften eines Dual-Athlon SMP-Systems untersucht werden. Als Chipsatz kommt hierbei der AMD-760MPX zum Einsatz — dieser Chipsatz verwendet als Host-Bridge einen AMD-762. Als South-Bridge findet der AMD-768 Verwendung. Als Prozessoren werden zwei AMD Athlon MP 1800+ eingesetzt. Der Takt der Prozessoren beträgt hierbei 1533MHz. Der Takt des Host-Busses beträgt 133MHz. Der Hauptspeicher ist mit 512 MByte DDR SDRAM bestückt; der Takt der Speichermodule entspricht dabei dem Takt des Host-Busses. An AGP ist ein GeForce2 Grafikadapter angeschlossen.

Der Beschreibung des Chipsatzes lässt sich entnehmen, dass die Host-Bridge einen PCI-66 PCI-Bus bereitstellt. Die South-Bridge ist über diesen PCI-Bus an die Host-Bridge angebunden. Weiterhin wird der Speicher nicht im Interleaved-Modus betrieben, wodurch bei modifizierter Speicherbestückung keine veränderten Zugriffszeiten zu erwarten sind. Weiterhin ist der Beschreibung des Chipsatzes zu entnehmen, dass die Prozessoren über separate Punkt-zu-Punkt Verbindungen an die Host-Bridge angebunden sind. Damit entspricht das vorliegende Szenario dem in Abbildung 4.5 dargestellten Fall.

Da die verwendete Architektur für Realzeitaufgaben — und nicht als Multimedia-PC — eingesetzt werden soll, treten im Realzeitbetrieb keine Zugriffe eines PCI-Gerätes auf den Speicher des Grafikadapters auf. Alle sonstigen Beeinflussungen sind dagegen möglich. Somit müssen zur Untersuchung der Realzeiteigenschaften dieses SMP-Systems die beiden folgenden Zugriffsvarianten des Prozessors, auf dem die zu untersuchende Realzeittask ausgeführt wird, analysiert werden:

1. Die CPU greift auf den Hauptspeicher zu.

2. Die CPU greift auf den Speicher eines Peripheriegerätes zu.

Parallel hierzu können vom anderen Prozessor, vom Grafikadapter und von Peripheriegeräten die folgenden Zugriffe auf die Hardware durchgeführt werden, wobei diese Zugriffe teilweise auch gleichzeitig ausgeführt werden können:

1. Die andere CPU greift auf den Hauptspeicher zu.
2. Die andere CPU greift auf den Speicher des Grafikadapters zu.
3. Die andere CPU greift auf den Speicher eines Peripheriegerätes zu.
4. PCI-Peripheriegeräte greifen auf den Hauptspeicher zu.
5. Der Grafikadapter greift auf den Hauptspeicher zu.

Die Auswirkungen dieser parallelen Lasten sollen nun für das Multiprozessorsystem bestimmt werden. Hierzu werden für die einzelnen Lastszenarien die jeweiligen Parameter der Gleichung 4.8 bestimmt.

Zugriffe auf den Hauptspeicher

In diesen Messreihen werden die Latenzzeiten beim Zugriff auf den Hauptspeicher bestimmt. Hierzu werden zunächst die Beeinflussungen untersucht, bei denen überlappende Datenpfade vorhanden sind: In der ersten Messreihe werden die Einwirkungen gemessen, die durch parallele Hauptspeicherzugriffe des anderen Prozessors entstehen. Anschließend werden in einer weiteren Messreihe die Auswirkungen parallel stattfindender Hauptspeicherzugriffe des Grafikadapters und der PCI-Peripheriegeräte analysiert.

Ob Datenpfade tatsächlich parallel bearbeitet werden können, wird in den folgenden Messungen untersucht. Hierbei greift der andere Prozessor während der Hauptspeicherzugriffe auf den Speicher des Grafikadapters oder auf den Speicher von Peripheriegeräten zu.

1. Gleichzeitiger Zugriff beider Prozessoren

Zunächst werden die Beeinflussungen bestimmt, die durch parallele Speicherzugriffe der anderen CPU entstehen. Durch AGP und durch den PCI-Bus werden während der Messungen keine Datentransfers durchgeführt.

Die Ergebnisse dieser Messungen sind in Abbildung 4.9 dargestellt. Dabei sind die benötigten Zeiten für eine bestimmte Anzahl aufeinanderfolgender Hauptspeicherzugriffe aufgetragen. Aufgrund von Verdrängungen im Cache besteht ein Zugriff — wie im Worst-Case möglich — aus einem Lese- und aus einem Schreibzugriff auf den Hauptspeicher. In der ersten Messreihe werden die Zugriffszeiten auf den Hauptspeicher ohne dem Einfluss paralleler Hauptspeicherzugriffe bestimmt (vergleiche hierzu auch Abbildung 4.3). In der zweiten Messreihe erfolgen parallele Hauptspeicherzugriffe der anderen CPU. In der dritten Messreihe erfolgen Speicherzugriffe der anderen CPU, wobei hier beide Prozessoren auf den gleichen Speicherbereich zugreifen. In Abbildung 4.9 sind die jeweils ermittelten Messwerte eingetragen.

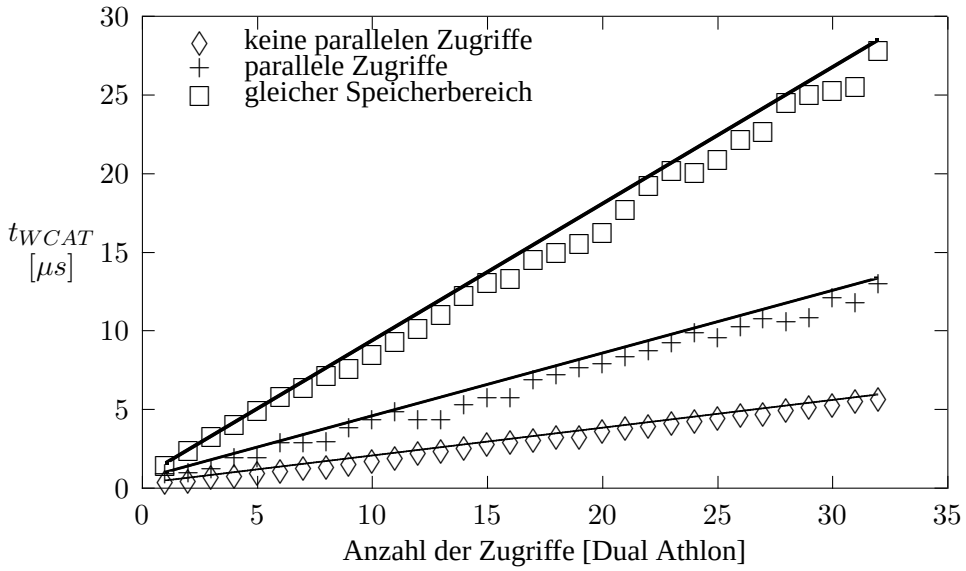


Abbildung 4.9: parallele Hauptspeicherzugriffe (CPU-CPU)

Als Ergebnis lässt sich festhalten, dass bei parallelem Zugriff der anderen CPU auf den Hauptspeicher prinzipiell mit ca. doppelt so langen Ausführungszeiten gerechnet werden muss. Falls jedoch eine Speicherzelle modifiziert wird, die die andere CPU in ihrem Cache hat, kommt es aufgrund der Synchronisation der Caches (Cache-Snooping) zu erheblich verlängerten Zugriffszeiten. Hier vervierfacht sich in etwa die Zugriffszeit auf den Hauptspeicher. Für weiterführende Informationen zum Thema Cache-Snooping sei auf [14] verwiesen.

2. Gleichzeitiger Zugriff von Peripheriegeräten

In dieser Messreihe soll untersucht werden, in welcher Art und Weise sich parallel stattfindende Hauptspeicherzugriffe des Grafikadapters und des von der Host-Bridge bereitgestellten PCI-Busses auf die Hauptspeicherzugriffe eines Prozessors auswirken. Der andere Prozessor führt während dieser Messungen keine Operationen aus; dieser wird in den Zustand “Prozessorhalt” versetzt.

Die Messergebnisse sind in Abbildung 4.10 eingetragen. Hier sind die benötigten Ausführungszeiten ohne parallele Zugriffe anderer Komponenten aufgetragen. In den weiteren Messungen werden die jeweils benötigten Ausführungszeiten bestimmt, falls der Grafikadapter oder ein PCI-Peripheriegerät ebenfalls auf den Hauptspeicher zugreifen möchte. In der letzten Messung werden die resultierenden Zugriffszeiten gemessen, falls gleichzeitig von AGP und von PCI Speichertransfers generiert werden.

Als Ergebnis lässt sich festhalten, dass der verwendete Chipsatz die PCI- und die AGP-Schnittstelle gleich gewichtet. Hierbei resultieren gleiche Latenzzeiten beim Zugriff auf den Hauptspeicher, falls der Grafikadapter *oder* ein PCI-Gerät parallel auf den Hauptspeicher zugreift.

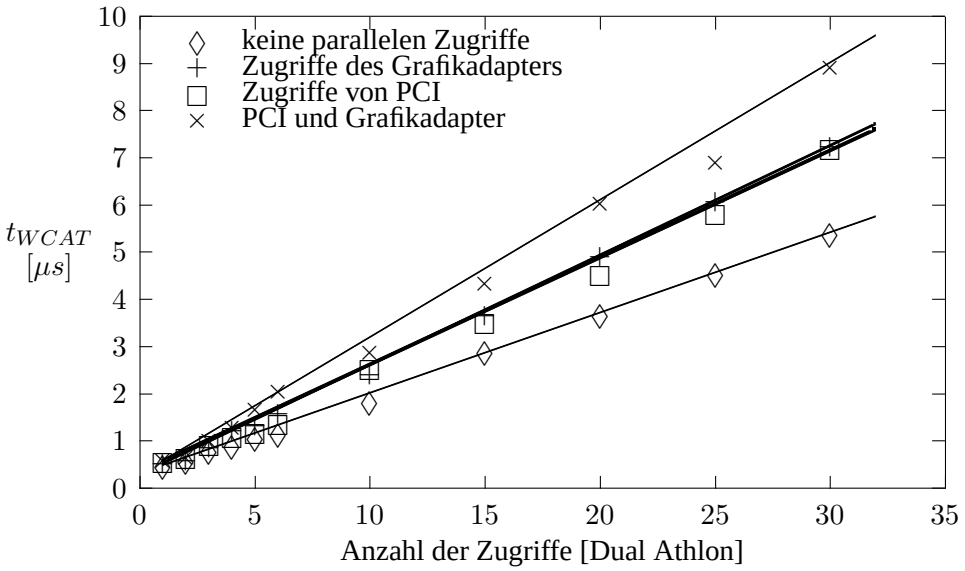


Abbildung 4.10: parallele Hauptspeicherzugriffe (CPU-Grafik, CPU-PCI)

Greifen PCI und AGP gemeinsam auf den Hauptspeicher zu, verlängert sich nochmals die maximal mögliche Zugriffszeit.

3. Auswirkungen sonstiger Datenübertragungen

In den folgenden Messreihen wird untersucht, welche Beeinflussungen entstehen, falls der andere Prozessor auf den Speicher des Grafikadapters oder auf den Speicher eines Peripheriegerätes zugreift. Damit wird überprüft, ob die Host-Bridge die jeweiligen Zugriffe parallel verarbeiten kann.

Als Ergebnis dieser Messungen lässt sich festhalten, dass aufgrund dieser parallelen Zugriffe keine nennenswerten Beeinflussungen auftreten. Dies ist insbesondere auch auf die vorhandenen Punkt-zu-Punkt Verbindungen zu beiden Prozessoren zurückzuführen. Die Ergebnisse dieser Messreihe sind in Tabelle 4.6 mit aufgeführt.

Zusammenfassung der Messergebnisse

Die Ergebnisse der Beeinflussungen beim Zugriff auf den Hauptspeicher sind in Tabelle 4.6 zusammengefasst. Hierbei sind die Einflüsse der jeweils parallel zu den Hauptspeicherzugriffen durchgeführten Transfers aufgeführt: Von der lasterzeugenden CPU werden Transfers auf den Hauptspeicher (HS), auf den Grafikadapter (GRA) und auf den Speicher eines PCI-Peripheriegeräts (PCI) durchgeführt. Parallel hierzu finden Transfers des Grafikadapters und von PCI-Geräten statt. Um die Messwerte vergleichbar zu machen, sind diese aufgerundet.

Zugriffe auf den Hauptspeicher werden insbesondere dann beeinflusst, falls parallel hierzu der andere Prozessor oder Peripheriegeräte ebenfalls auf den Hauptspeicher zugreifen. Bei allen

Lastsituation	$t_{WCAT,1}[\mu s]$	$m_{WCAT}[\mu s]$
keine parallele Last	0,50	0,20
Prozessorzugriffe (GRA)	0,80	0,20
Prozessorzugriffe (PCI)	0,60	0,20
Prozessorzugriffe (HS)	1,00	0,40
Prozessorzugriffe (HS, gleicher Bereich)	1,60	0,90
PCI-Transfers	0,60	0,25
Zugriffe des Grafikadapters	0,60	0,25
PCI und Grafikadapter	0,60	0,30
PCI, Grafikadapter und CPU parallel	1,00	0,60

Tabelle 4.6: Latenzzeiten beim Zugriff auf den Hauptspeicher

anderen Zugriffsszenarien ergibt sich nur eine minimale Veränderung der Zugriffszeit, die sich in dem veränderten Offset $t_{WCAT,1}$ bemerkbar macht.

Zugriffe auf Peripheriegeräte

In diesem Abschnitt sollen die Latenzzeiten des AMD-760MPX Chipsatzes bei Zugriffen auf ein PCI-Peripheriegerät untersucht werden. An dieser Stelle werden ausschließlich die Latenzzeiten betrachtet, die durch den Chipsatz verursacht werden — der PCI-Bus als zentraler Peripheriebus wird gesondert in Abschnitt 4.3.3 behandelt. Hierzu werden zunächst die Beeinflussungen gemessen, die entstehen, falls beide Prozessoren gleichzeitig auf den Speicher eines Peripheriegerätes zugreifen möchten. Diese Speicherzugriffe werden hierbei als Portzugriffe durchgeführt. Als Portadresse wird dabei immer die gleiche Adresse verwendet. In einer weiteren Messreihe werden anschließend die Einflüsse sonstiger, parallel ablaufender Datentransfers untersucht.

Bei den genannten Messreihen soll analysiert werden, ob und mit welchen *prinzipiellen* Beeinflussungen beim Zugriff auf Peripheriegeräte gerechnet werden muss. Insbesondere dürfen die hier erhaltenen Messwerte nicht als absolute Größen betrachtet werden, da sich in Abhängigkeit von der jeweiligen Adresse — und damit ggf. in Abhängigkeit des adressierten Peripheriegerätes — unterschiedliche Zugriffszeiten ergeben. In diesen Messungen greift der ausführende Prozessor auf die Portadressen eines IDE-Controllers zu. Der lasterzeugende Prozessor verwendet die Adressen einer Soundkarte.

1. Gleichzeitiger Zugriff beider Prozessoren

In dieser Messreihe soll untersucht werden, welche Auswirkungen auf Zugriffszeiten entstehen, falls beide Prozessoren auf Peripheriegeräte gleichzeitig zugreifen möchten. Da die Host-Bridge den PCI-Bus jeweils nur einer CPU zuteilen kann, ist mit Beeinflussungen zu rechnen.

Das Ergebnis dieser Messung ist in Abbildung 4.11 grafisch dargestellt. Hierbei ist die Anzahl x der sequentiell durchgeführten Zugriffe in Abhängigkeit von der dafür benötigten Zeit

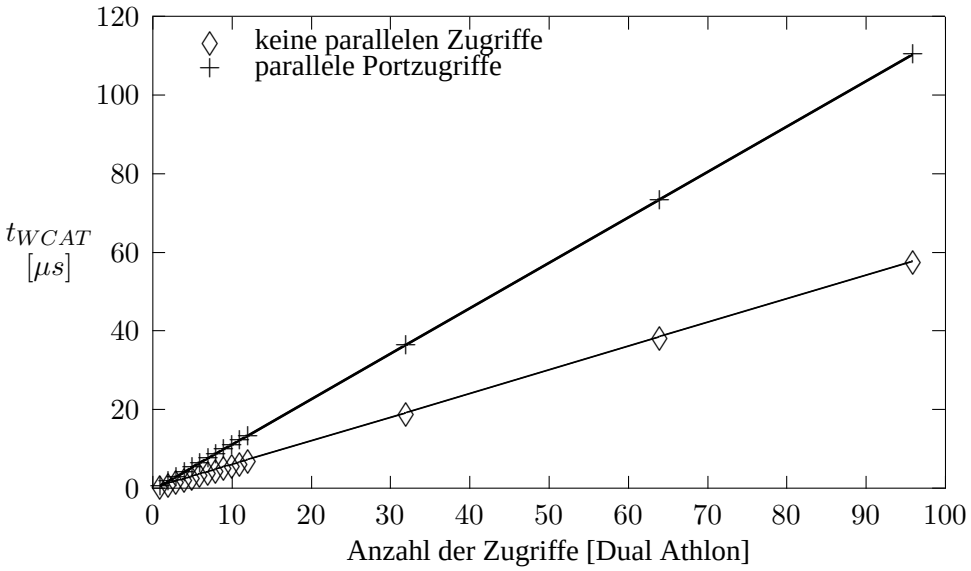


Abbildung 4.11: parallele Portzugriffe (CPU-CPU)

aufgetragen. Eine Messreihe wird ohne paralleler Last, die andere Messreihe mit parallelen Portzugriffen des anderen Prozessors durchgeführt.

Wie aus Abbildung 4.11 zu entnehmen ist, muss bei dieser Architektur bei parallel stattfindenden Portzugriffen mit doppelt so langen Ausführungszeiten für eine bestimmte Anzahl an Zugriffen gerechnet werden. Da in dem SMP-System die Prozessoren gleichberechtigt sind, entspricht dieses Ergebnis den Erwartungen.

2. Auswirkungen sonstiger Datenübertragungen

In einer weiteren Messreihe werden die Beeinflussungen sonstiger, parallel stattfindender Datenübertragungen untersucht. Insbesondere greift hier der lasterzeugende Prozessor während der Portzugriffe auf den Hauptspeicher und auf den Speicher des Grafikadapters zu. Zusätzlich wird untersucht, welche Beeinflussungen entstehen, falls der Grafikadapter während der Portzugriffe Datentransfers mit dem Hauptspeicher durchführt.

Bei diesen Untersuchungen kann festgestellt werden, dass hier nicht mit nennenswerten Beeinflussungen gerechnet werden muss. Die Ergebnisse dieser Messungen sind in Tabelle 4.7 mit aufgeführt.

Zusammenfassung der Messergebnisse

Die gemessenen Zugriffszeiten auf Peripheriegeräte, mit denen prinzipiell in dem untersuchten SMP-System gerechnet werden muss, sind in Tabelle 4.7 zusammengefasst. Der lasterzeugende Prozessor führt hierbei Zugriffe auf Portadressen, auf den Hauptspeicher (HS) und auf den Speicher des Grafikadapters (GRA) durch. Zusätzlich wird der Einfluss parallel stattfindender Zugriffe des Grafikadapters auf den Hauptspeicher untersucht.

Es sei nochmals daran erinnert, dass die gemessenen Ausführungszeiten nur in Relation gesetzt werden dürfen. Beim Zugriff auf andere Adressbereiche muss mit veränderten Zugriffszeiten gerechnet werden. Für eine bessere Übersichtlichkeit sind die Werte aufgerundet. Man erkennt, dass insbesondere beim parallelen Zugriff der Prozessoren auf Peripheriegeräte gegenseitige Beeinflussungen auftreten.

Lastsituation	$t_{WCAT,1}[\mu s]$	$m_{WCAT}[\mu s]$
keine parallele Last	0,70	0,60
parallele Portzugriffe	1,40	1,20
Prozessorzugriffe (HS)	0,70	0,60
Prozessorzugriffe (GRA)	0,75	0,60
Zugriffe des Grafikadapters	0,70	0,60

Tabelle 4.7: Latenzzeiten beim Zugriff auf Peripheriegeräte

4.3.1.8 Zusammenfassung

In diesem Abschnitt wird untersucht, mit welchen Latenzzeiten bei einem SMP-basierten Multiprozessorsystem gerechnet werden muss. Problem bei der Verwendung einer solchen Architektur ist, dass die Spezifikationen der vorhandenen Komponenten größtenteils nicht zugänglich sind. Ein weiteres Problem bei der Verwendung von PC-Komponenten ist die hohe Typenvielfalt, sowie die große Anzahl an Variations- und Kombinationsmöglichkeiten.

Aus diesen Gründen wird ein Verfahren beschrieben, mit dem die Eigenschaften des Chipsatzes und insbesondere die Eigenschaften der Host-Bridge, auf Eignung für den Realzeitbetrieb anhand von Messungen untersucht werden können. Dabei wird darauf eingegangen, welche Messungen durchgeführt werden müssen und auf welche Art und Weise diese durchzuführen sind. Mit den erhaltenen Resultaten der Messungen können die Einflüsse der SMP-Architektur auf Hauptspeicher- und I/O-Zugriffe unter definierten Lastsituationen vorherbestimmt werden. Die Größenordnung der resultierenden Beeinflussungen wird für den AMD-760MPX Chipsatz bestimmt.

4.3.2 NUMA-Architektur

Nachdem im letzten Abschnitt SMP-Systeme beschrieben wurden, soll nun auf NUMA-Multiprozessorsysteme auf PC-Basis eingegangen werden. NUMA ist die Abkürzung für *Non-Uniform Memory Access*. Bei NUMA-Systemen werden mehrere Knoten (Nodes) über ein Bussystem miteinander verbunden.

Der prinzipielle Aufbau einer NUMA-Architektur ist in Abbildung 4.12 dargestellt. Das Multiprozessorsystem besteht aus n Knoten. Jeder Knoten beinhaltet mindestens eine CPU. An einem Knoten *können* Speicher (RAM) und/oder Peripheriekomponenten angeschlossen sein.

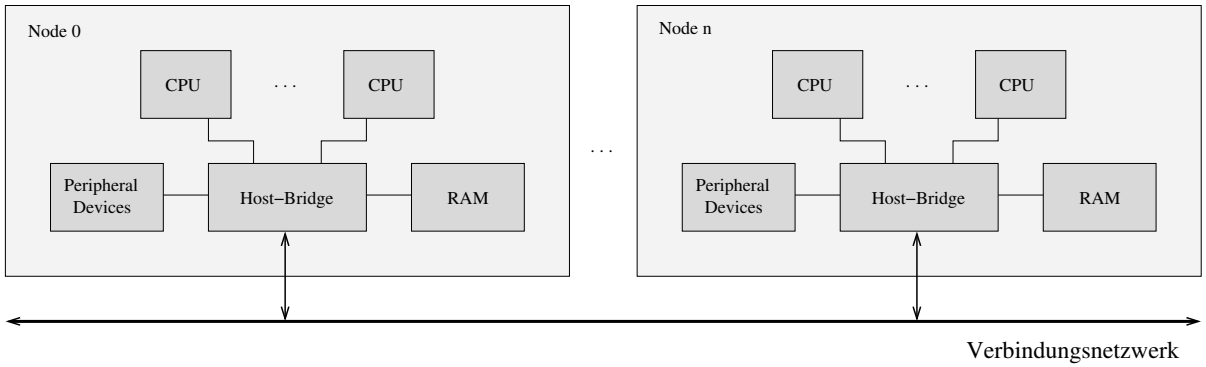


Abbildung 4.12: NUMA-Architektur

Die Kommunikation zwischen den einzelnen Knoten erfolgt über ein Verbindungsnetzwerk. Hier findet beispielsweise HyperTransport (siehe Abschnitt 4.3.2.4) Verwendung. Dabei wird ein Knoten über eine oder mehrere Schnittstellen an das Verbindungsnetzwerk angeschlossen.

Es existieren zwei unterschiedliche Varianten der NUMA-Architektur:

- *cc-NUMA:* Beim Cache-Coherent NUMA wird die Cache-Kohärenz über das gesamte Multiprozessorsystem gewährleistet.
- ncc-NUMA: Beim Non-Cache-Coherent NUMA wird die Cache-Kohärenz nur innerhalb eines Knotens gewährleistet.

Da ncc-NUMA für PC-basierte Multiprozessorsysteme keine Bedeutung hat, wird im Folgenden der Ausdruck NUMA als Synonym für cc-NUMA verwendet.

Aus Sicht der Software verhält sich eine NUMA-Architektur wie ein SMP-System. Alle Prozessoren sind gleichberechtigt. Jeder Prozessor kann — wie auch bei der SMP-Architektur — auf jeden Bereich des Hauptspeichers, und damit auf den Speicher jedes Knotens zugreifen. Jede CPU kann Interrupts aus allen möglichen Interruptquellen erhalten und jedes Peripheriegerät und jede Komponente des Multiprozessorsystems adressieren. Im Gegensatz zu SMP-Systemen variieren jedoch die benötigten Zugriffszeiten in Abhängigkeit davon, ob die Zugriffe lokal ausgeführt werden können oder nicht. Diese Zugriffe sind deshalb *non-uniform*.

Im Vergleich zu SMP-Systemen bieten NUMA-Systeme den Vorteil, dass bei Zugriffen auf den lokalen Hauptspeicherbereich und auf lokale Peripheriekomponenten im Allgemeinen mit weniger Beeinflussungen durch andere Prozessoren zu rechnen ist. Damit erhöht sich bei NUMA-Systemen die *durchschnittliche* Rechenleistung im Vergleich zu SMP-Systemen. Da es aber grundsätzlich allen Prozessoren und Peripheriegeräten erlaubt ist, auf sämtliche Systemressourcen zuzugreifen, resultiert aus der Verwendung einer NUMA-Architektur zunächst kein Vorteil für worst-case Zugriffszeiten. Da insbesondere auf Code und Daten des Betriebssystems von allen Prozessoren aus zugegriffen wird, dieser Code bzw. die jeweiligen Daten allerdings oft nicht auf dem lokalen Knoten abgelegt sind, resultieren hieraus Zugriffe einer CPU auf andere Knoten des NUMA-Systems. Zusätzlich ist es möglich, dass im Zuge eines Betriebssystemaufrufs auf die Peripheriegeräte eines anderen Knotens zugegriffen wird.

In den folgenden Abschnitten soll erläutert werden, wie auftretende Latenzzeiten beim Zugriff auf Speicher und Peripheriegeräte bestimmt werden können. Ziel ist es, maximale Zugriffszeiten für eine bestimmte Anzahl aufeinanderfolgender Zugriffe unter bestimmten Lastszenarien anzugeben. Hierzu wird in Abschnitt 4.3.2.1 erklärt, welche gegenseitigen Beeinflussungen bei einem NUMA-System auftreten und wie diese Beeinflussungen anhand von Messungen bestimmt werden können. Da in den meisten Multiprozessorsystemen auf PC-Basis HyperTransport zur Verbindung der einzelnen Komponenten des Chipsatzes verwendet wird, wird in Abschnitt 4.3.2.4 auf dieses Verbindungsprotokoll eingegangen. Anschließend werden beispielhaft in Abschnitt 4.3.2.5 die maximalen Zugriffszeiten bei der Verwendung eines Quad-Opteron Multiprozessorsystems bestimmt. In Abschnitt 4.3.2.6 folgt eine Zusammenfassung.

4.3.2.1 Latenzzeiten in NUMA-Systemen

Bei der Untersuchung der auftretenden Latenzzeiten innerhalb eines NUMA-Multiprozessorsystems sollen die bereits bekannten Erkenntnisse aus der Untersuchung von SMP-Architekturen herangezogen werden (Abschnitt 4.3.1). Somit wird im Folgenden insbesondere auf die Gemeinsamkeiten und Unterschiede zu SMP-Architekturen eingegangen.

Soll ein NUMA-Multiprozessorsystem für harte Realzeitaufgaben Verwendung finden, so ist man bei der Bestimmung maximaler Zugriffszeiten auf bestimmte Systemressourcen zunächst mit den gleichen Problemen wie bei SMP-Systemen konfrontiert. Insbesondere ist der Entwicklungszyklus bei den Komponenten eines NUMA-Systems ebenfalls sehr schnell, so dass auch hier eine schritthaltende Ermittlung der Latenzzeiten anhand von Spezifikationen — sofern diese zugänglich sind — nur mit großem Aufwand realisiert werden kann. Weiterhin können NUMA-Systeme aus den verschiedensten Hardwarekomponenten zusammengestellt werden. Diese Komponenten können sich hinsichtlich ihrer Programmierung (Firmware) und Konfiguration (BIOS) unterscheiden.

Aus den selben Gründen wie bei SMP-Systemen wird deshalb auch für NUMA-Systeme vorgeschlagen, die maximalen Zugriffszeiten für Zugriffe auf bestimmte Komponenten des NUMA-Multiprozessorsystems anhand von Messungen zu bestimmen. Dabei kann auch bei NUMA-Systemen davon ausgegangen werden, dass die maximalen Zugriffszeiten auf bestimmte Systemressourcen unter gleichen Lastsituationen ein *lineares* Verhalten aufweisen. Insbesondere wird angenommen, dass die in Abschnitt 4.3.1.2 eingeführte Gleichung 4.8 grundsätzlich auch für NUMA-Systeme gültig ist.

Welche Messungen durchgeführt werden müssen und wie diese durchzuführen sind, beschreiben die folgenden Abschnitte. Hierzu wird in Abschnitt 4.3.2.2 auf die durchzuführenden Untersuchungen bei Zugriffen *innerhalb* eines Knotens eingegangen. In Abschnitt 4.3.2.3 wird erklärt, was bei der Untersuchung der Zugriffe auf entfernte Knoten zu beachten ist.

4.3.2.2 Zugriffe innerhalb eines Knotens

Erfolgen die Zugriffe einer CPU ausschließlich auf Ressourcen innerhalb eines Knotens, so sind die auftretenden Latenzzeiten zu den in Abschnitt 4.3.1.1 beschriebenen äquivalent. Hierbei kann jeder Knoten — sofern er mit RAM bestückt ist — als ein eigenes SMP-System betrachtet werden. Parallel stattfindende Zugriffe anderer Prozessoren auf diesen Knoten können wie die Zugriffe der Peripheriegeräte in SMP-Systemen betrachtet werden. Somit treten beim Zugriff auf innerhalb eines Knotens befindliche Komponenten die folgenden Latenzzeiten auf:

- Die auf dem Knoten vorhandenen CPUs können sich beim Zugriff auf die Host-Bridge beeinflussen, falls hier ein Host-Bus arbitriert werden muss.
- Die Architektur der auf dem Knoten befindlichen Host-Bridge bestimmt, mit welchen Latenzzeiten zu rechnen ist, falls mehrere Datentransfers diesen Baustein durchlaufen. Wie bei SMP-Systemen erfolgt die Anbindung der CPUs an Speicher und Peripherie über die Host-Bridge.
- Zusätzlich hängt die Latenzzeit von der Reaktionszeit der angesprochenen Speicher- bzw. Peripheriekomponenten ab. Insbesondere müssen auch hier die Auswirkungen parallel stattfindender Zugriffe anderer Prozessoren und Peripheriegeräte berücksichtigt werden.

Sollen die maximalen Zugriffszeiten auf Komponenten innerhalb eines Knotens anhand von Messungen ermittelt werden, so müssen zunächst alle möglichen konkurrierenden Ressourcenanforderungen bestimmt werden: Prinzipiell können sich alle Zugriffe gegenseitig beeinflussen, bei denen überlappende Datenpfade vorhanden sind. Treffen sich diese Datenpfade in einem Baustein, muss überprüft werden, ob hier aufgrund interner Arbitrierungsverfahren gegenseitige Beeinflussungen auftreten. Während der Messungen müssen sich alle betrachteten Komponenten so verhalten, dass sie die Zugriffe um das maximal mögliche Maß verlängern.

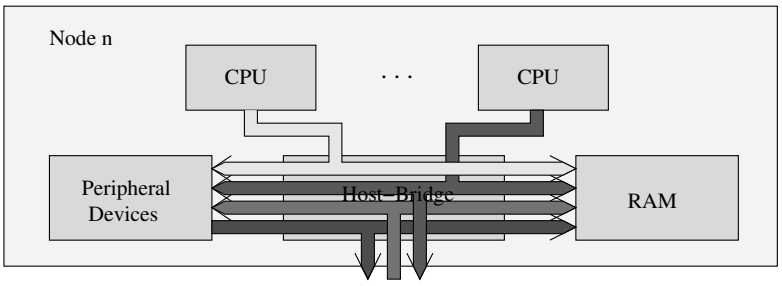


Abbildung 4.13: Parallele Datentransfers bei Zugriffen innerhalb eines Knotens

Abbildung 4.13 illustriert alle möglichen Datenpfade, die bei Zugriffen innerhalb eines Knotens auftreten können: Grundsätzlich kann ein Prozessor auf den lokalen Hauptspeicher (RAM) oder auf lokale Peripheriegeräte zugreifen. Diese Zugriffe sind mit einem hellen Pfeil dargestellt. Die Dauer dieser Zugriffe kann durch die im Folgenden aufgeführten Datentransfers beeinflusst werden, wobei diese im schlimmsten Fall gleichzeitig auftreten können. Diese Datenpfade sind in Abbildung 4.13 mit dunklen Pfeilen dargestellt:

- Lokale Prozessoren können auf den lokalen Hauptspeicher, auf lokale Peripheriegeräte oder auf andere Knoten zugreifen.
- Lokale Peripheriegeräte können auf den lokalen Hauptspeicher, auf andere lokale Peripheriegeräte oder auf andere Knoten zugreifen.
- Entfernte Prozessoren können auf den lokalen Hauptspeicher oder auf lokale Peripheriegeräte zugreifen.
- Entfernte Peripheriegeräte können auf den lokalen Hauptspeicher oder auf lokale Peripheriegeräte zugreifen.

Für die Untersuchung der Realzeiteigenschaften eines NUMA-Systems brauchen die Fälle, bei denen ein Peripheriegerät auf den Speicher eines anderen Peripheriegeräts zugreift, nicht untersucht werden. Diese Beeinflussungen treten im Allgemeinen nicht auf (vgl. auch S. 42).

In den Abbildungen 4.12 und 4.13 ist die Anbindung des Knotens eines NUMA-Systems an das Verbindungsnetzwerk mit nur einer Verbindung dargestellt. Die Anzahl der tatsächlich vorhandenen Anschlüsse (Links) in das Verbindungsnetzwerk wird durch die Architektur des Chipsatzes bestimmt. Sollen die Einflüsse entfernter Knoten und Peripheriegeräte bei Zugriffen auf lokale Ressourcen untersucht werden, so muss hier die Einwirkung jedes Anschlusses gesondert berücksichtigt werden. Hier muss ggf. auch der Fall untersucht werden, dass der betrachtete Knoten nur als Bindeglied zwischen den Datenübertragungen zweier benachbarter Knoten dient.

Prinzipiell kann in einem NUMA-System davon ausgegangen werden, dass sich bei Zugriffen innerhalb eines Knotens alle in dem System vorhandenen Knoten gleich verhalten. Insbesondere wird davon ausgegangen, dass identische Knoten — also Knoten, die mit identischen Hardwarekomponenten bestückt sind und auf die gleiche Art und Weise in das Verbindungsnetzwerk eingebunden sind — die gleichen Realzeiteigenschaften aufweisen. Da sich die Knoten jedoch beispielsweise in der Peripherieanbindung unterscheiden, müssen hier diejenigen Knoten untersucht werden, die alle möglichen Kombinationen der Datenpfade beinhalten:

Sei $\mathbb{D} = \{d_1,...,d_n\}$ die Menge aller n Datenpfade, die prinzipiell auf den m Knoten des NUMA-Systems auftreten können und sei $\mathbb{K} = \{0,..,m-1\}$ die Menge aller Knoten. Damit lässt sich für jeden Knoten i die Menge $\mathbb{P}_i \subset \mathbb{D}$ aller Datenpfade angeben, die in einem Knoten auftreten. Es müssen nun die Realzeiteigenschaften aller Knoten j untersucht werden, für die gilt:

$$\mathbb{P}_j \not\subset \mathbb{P}_i; \qquad i \neq j \tag{4.9}$$

mit $\{i,j\} \subset \mathbb{K}$.

Wurde die Menge aller zu untersuchenden Knoten bestimmt, so müssen für jeden Knoten die Auswirkungen paralleler Ressourcenzugriffe anhand von Messungen untersucht werden. Diese Auswirkungen müssen hierbei in allen relevanten Kombinationen analysiert werden. Insbesondere müssen diejenigen parallelen Beeinflussungen in Kombination untersucht werden, bei denen auch beim alleinigen Auftreten Latenzzeiten messbar sind: Falls sich beispielsweise ein paralleler Datenpfad d_0 nicht auf maximale Zugriffszeiten auswirkt und selbiges für den Daten-

pfad d_1 gilt, dann muss das gemeinsame Auftreten der Belastungen d_0 und d_1 nicht analysiert werden.

Weiterhin sollten die entsprechenden Beeinflussungen im späteren Realzeitbetrieb auch tatsächlich auftreten können. Die Parameter für Zugriffe innerhalb der restlichen Knoten sind dabei in den gemessenen Zugriffszeiten enthalten und lassen sich direkt übernehmen. Voraussetzung hierfür ist das angenommene ähnliche Verhalten der einzelnen Knoten des NUMA-Systems.

4.3.2.3 Zugriffe auf entfernte Knoten

Erfolgen die Zugriffe auf Ressourcen eines anderen Knotens, so werden diese Zugriffe sowohl durch parallele Datenübertragungen anderer Prozessoren und Peripheriegeräte im Verbindungsnetzwerk, als auch durch parallel auftretende Übertragungen im lokalen und im entfernten Knoten beeinflusst. Im Folgenden wird der Knoten, der einen Datentransfer initiiert, auch als *Quellknoten* bezeichnet. Der entfernte Knoten, mit dem jeweils kommuniziert wird, wird auch *Zielknoten* genannt.

Mittels Messung können die Einflüsse des Verbindungsnetzwerks, sowie die Einflüsse im lokalen und im entfernten Knoten, nicht separat analysiert werden. Grund hierfür ist, dass immer nur die *gesamte* Zugriffszeit beim Zugriff auf einen entfernten Knoten gemessen werden kann. Deshalb werden in den folgenden drei Unterabschnitten zunächst mögliche Einflüsse im Quell- und im Zielknoten und mögliche Einflüsse im Verbindungsnetzwerk diskutiert. Insbesondere wird hier erklärt, welche Beeinflussungen prinzipiell auftreten können und was bei der Ermittlung maximaler Zugriffszeiten anhand von Messungen beachtet werden muss. Basierend auf diesen Vorüberlegungen wird im darauffolgenden Unterabschnitt auf Seite 59 das tatsächliche Vorgehen bei der Bestimmung der relevanten Zugriffszeiten anhand von Messungen erläutert.

Einflüsse im Quellknoten

Prinzipiell treten beim Zugriff auf einen entfernten Knoten die gleichen Beeinflussungen auf, die auch schon im vorhergehenden Abschnitt bei der Diskussion der Zugriffe innerhalb eines Knotens behandelt wurden. Insbesondere kann der Zugriff einer CPU auf einen entfernten Knoten durch vorhandene Arbitrierungsverfahren des Host-Busses und der Host-Bridge verzögert ausgeführt werden.

In Abbildung 4.14 sind alle Datenpfade aufgetragen, die zur Bestimmung der maximalen Zugriffszeit eines Prozessors auf einen entfernten Knoten mit Hilfe von Messungen von Bedeutung sind. Es treten überall dort Beeinflussungen auf, wo sich Datenpfade mehrerer Quellen überlagern. Der Prozessor, der einen Datentransfer mit einem entfernten Knoten durchführen möchte, muss auf den Host-Bus und auf die Host-Bridge zugreifen. Anschließend erfolgt der Zugriff auf das Verbindungsnetzwerk. Dieser Datenpfad ist in Abbildung 4.16 mit einem hellen Pfeil dargestellt. Beeinflusst werden diese Datenübertragungen von Zugriffen auf die Ressourcen dieses Knotens (Speicher oder Peripheriegeräte) und von allen Zugriffen lokaler Prozessoren oder Pe-

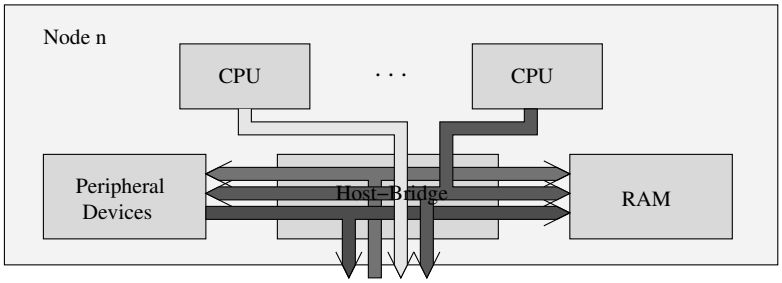


Abbildung 4.14: Parallele Datentransfers im Quellknoten

ripheriegeräte auf das Verbindungsnetzwerk. Diese Zugriffe sind in Abbildung 4.14 mit dunklen Pfeilen dargestellt.

Einflüsse des Verbindungsnetzwerks

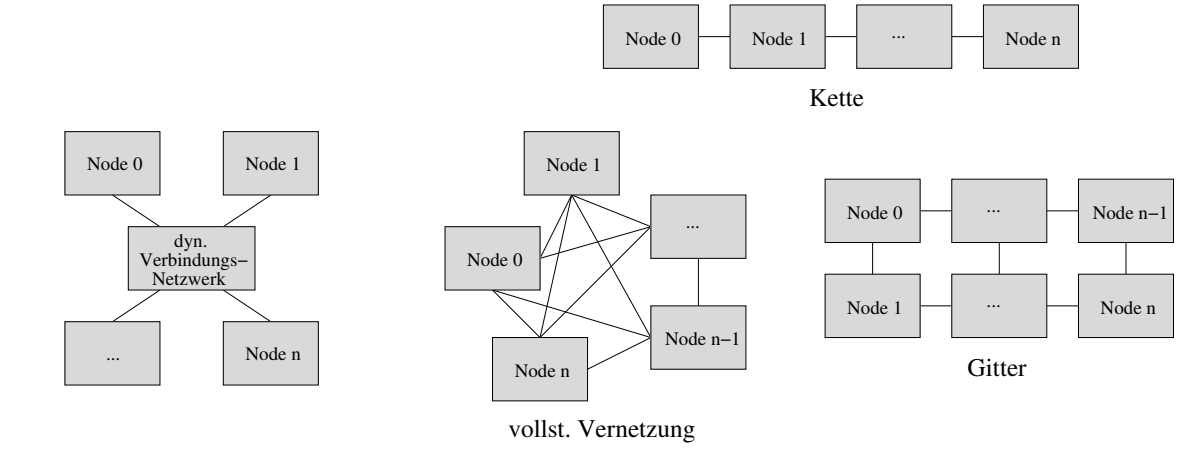
Die Architektur des Verbindungsnetzwerks bestimmt, ob und in welcher Art und Weise Zugriffe auf entfernte Knoten beeinflusst werden. Nach [55] lassen sich Verbindungsnetzwerke in *dynamische* und *statische* Verbindungsnetzwerke einteilen. Dynamische Verbindungsnetzwerke enthalten konfigurierbare Schaltelemente, die beliebige Ein- und Ausgänge miteinander verbinden können. Durch die Architektur des Verbindungsnetzwerks wird bestimmt, welche Ein- und Ausgänge *gleichzeitig* miteinander verbunden sein können. Durch Arbitrierungsverfahren wird festgelegt, in welcher Reihenfolge das Verbindungsnetzwerk geschaltet wird. Das einfachste dynamische Verbindungsnetzwerk ist der Bus; vergleiche hierzu auch Abbildung 4.15a.

Bei statischen Verbindungsnetzwerken ist die Kommunikationsstruktur fest vorgegeben. Diese besteht aus einzelnen Punkt-zu-Punkt Verbindungen zwischen den jeweiligen Knoten. Zwischen den Knoten werden Nachrichten bzw. Daten mit Hilfe von Paketen übertragen. Durch Routing-Tabellen und der Adressinformation in den Paketen wird festgelegt, wie diese zum Zielknoten gelangen.

Statische Verbindungsnetzwerke sind momentan die einzigen, die in NUMA-Systemen auf PC-Basis eingesetzt werden. Beispiele hierfür sind NUMA-Q (Sequent/IBM), Enterprise X-Architecture Technology (IBM) und HyperTransport. Da letzteres in Zukunft immer mehr an Bedeutung gewinnen wird — IBM ist beispielsweise dem HyperTransport Konsortium beigetreten — wird HyperTransport ausführlich in Abschnitt 4.3.2.4 behandelt.

Die Topologie eines statischen Verbindungsnetzwerks wird durch den verwendeten Chipsatz bestimmt. Gängige Implementierungen sind Ketten- (NUMA-Q) und Gitterstrukturen (HyperTransport). IBM verwendet für seine Enterprise X-Architecture die vollständige Vernetzung. Vergleiche hierzu auch Abbildung 4.15b.

In Abhängigkeit von der Architektur des verwendeten Verbindungsnetzwerks müssen die im Folgenden aufgeführten Aspekte berücksichtigt werden:



a) dynamisches Verbindungsnetzwerk b) statische Verbindungsnetzwerke

Abbildung 4.15: Verbindungsnetzwerke in NUMA-Systemen

- Wird ein dynamisches Verbindungsnetzwerk zur Verbindung der einzelnen Knoten verwendet, so werden auftretende Latenzzeiten durch die eingesetzten Arbitrierungsverfahren beeinflusst. Zusätzlich muss bei der Verwendung eines solchen Netzwerkes überprüft werden, ob mehrere Pfade parallel durch das Netzwerk geschaltet werden können, oder ob das Netzwerk bei bestimmten Kombinationen aus Ein- und Ausgängen blockiert.

- Bei statischen Verbindungsnetzwerken besitzt jeder Knoten eine oder mehrere Punkt-zu-Punkt Verbindungen in das Verbindungsnetzwerk. Zur Datenübertragung wird mindestens eine Punkt-zu-Punkt Verbindung bis zum Zielknoten benötigt, wobei die Daten mit Hilfe von Paketen übertragen werden. Latenzzeiten werden durch alle anderen parallel stattfindenden Datenübertragungen verursacht, die mindestens eine der benötigten Punkt-zu-Punkt Verbindungen gleichzeitig verwenden. Insbesondere werden auftretende Latenzzeiten von der maximalen Größe der Pakete, von den maximal möglichen Datenraten, von der Größe vorhandener Pufferspeicher und von den verwendeten Arbitrierungsverfahren beeinflusst. Soll hier der Einfluss des Verbindungsnetzwerks anhand von Messungen untersucht werden, so müssen alle Knoten, die mindestens auf eine der benötigten Verbindungen parallel zugreifen können, Pakete mit der jeweils maximal möglichen Rate über diese Verbindungen schicken. Um ausschließlich die Einflüsse des Verbindungsnetzwerks zu bestimmen, sollten die Zugriffe auf jeweils unterschiedliche Knoten erfolgen.

Zusammenfassend lässt sich festhalten, dass überall dort eine gegenseitige Beeinflussung der Datenübertragung auftritt, wo der Zugriff auf gemeinsam genutzte Komponenten des Verbindungsnetzwerks erfolgt. Kann für bestimmte Konstellationen nicht ausgeschlossen werden, dass die Datenübertragungen parallel ausgeführt werden können, so muss zur Bestimmung der Anzahl der durchzuführenden Untersuchungen von einer gegenseitigen Beeinflussung ausgegangen werden.

Einflüsse im Zielknoten

Im Zielknoten selbst können Datenübertragungen durch alle Transfers beeinflusst werden, die ebenfalls auf Ressourcen dieses Knotens zugreifen bzw. diese für die Datenübertragung benötigen. Durch die Architektur der Host-Bridge wird bestimmt, welche Latenzzeiten auftreten können. Aus Sicht der CPUs des Zielknotens ist das Verhalten der Host-Bridge äquivalent zu SMP-Systemen, unter der Annahme, dass der zugreifende Prozessor wie ein Peripheriegerät betrachtet wird. Zusätzlich hängt die Latenzzeit von der Reaktionszeit des Speichers bzw. von der Reaktionszeit der adressierten Peripheriekomponenten ab, wobei auch hier mögliche parallel ablaufende Datentransfers berücksichtigt werden müssen.

Weiterhin kann davon ausgegangen werden, dass im Zielknoten Zugriffe von entfernten Prozessoren und entfernten Peripheriegeräten äquivalent behandelt werden, da die Host-Bridge nur anhand der Adressinformation den jeweiligen Zugriff einem Knoten, nicht aber einer CPU oder einem Peripheriegerät dieses Knotens zuordnen kann.

Da die auftretenden Latenzzeiten im entfernten Knoten anhand von Messungen bestimmt werden sollen, müssen zunächst alle Datenpfade untersucht werden, die einen Einfluss auf die Entstehung von Latenzzeiten haben können. Auch bei der Untersuchung der Einflüsse im Zielknoten anhand von Messungen müssen sich alle beteiligten Komponenten so verhalten, dass sie die gemessenen Zugriffszeiten jeweils um das maximal mögliche Maß verlängern.

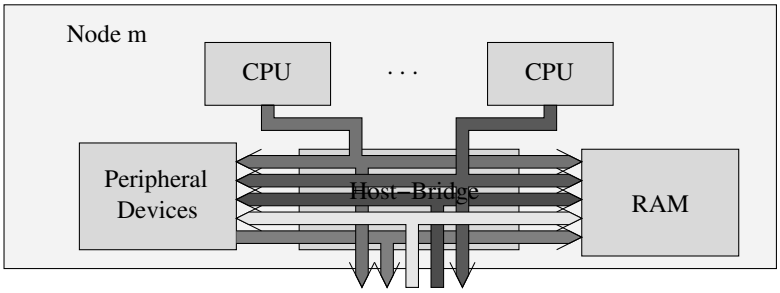


Abbildung 4.16: Parallele Datentransfers im Zielknoten

In Abbildung 4.16 sind alle Datenpfade eingezeichnet, die Zugriffe auf einen entfernten Knoten eines NUMA-Systems beeinflussen können: Die zu untersuchenden Zugriffe können dabei auf den Speicher des anderen Knotens oder auf dessen Peripheriegeräte erfolgen. Diese Zugriffe sind in Abbildung 4.16 mit einem hellen Pfeil dargestellt. Parallel hierzu können andere Prozessoren und Peripheriegeräte — jeweils lokale und entfernte — ebenfalls auf den Speicher bzw. auf Peripheriegeräte dieses Knotens zugreifen. Diese Zugriffe sind in Abbildung 4.16 mit dunklen Pfeilen dargestellt und äquivalent zu den Beeinflussungen beim lokalen Zugriff. Zusätzlich müssen — wie auch schon bei der Untersuchung der Zugriffe innerhalb eines Knotens — die Anzahl der vorhandenen Verbindungen in das Verbindungsnetzwerk berücksichtigt werden. Hier können parallel zu den untersuchten Zugriffen Transfers über bzw. auf den jeweiligen Knoten erfolgen.

Bestimmung der Latenzzeiten mittels Messung

Bei der Bestimmung der maximalen Zugriffszeit auf eine Ressource eines entfernten Knotens müssen alle Einflüsse paralleler Datentransfers berücksichtigt werden. Dazu gehören die Einflüsse des Knotens, der den jeweiligen Transfer initiiert, die Einflüsse des Verbindungsnetzwerks und die Einflüsse des entfernten Knotens. Diese Einflüsse wurden in den vorhergehenden Abschnitten diskutiert.

Problematisch bei der Bestimmung der maximalen Zugriffszeit auf einen entfernten Knoten ist, dass die Einflüsse der einzelnen Teile des Datenpfads — Zugriff auf den lokalen Knoten, Zugriff auf des Verbindungsnetzwerk, Zugriff auf den entfernten Knoten — nicht separat gemessen werden können. Somit können die Einflüsse dieser Komponenten nur indirekt aus dem Vergleich mehrerer Messwerte ermittelt werden. Beispielsweise lässt sich der Einfluss des Verbindungsnetzwerks bestimmen, falls in der ersten Messung ein Prozessor einen Zugriff auf einen entfernten Knoten ausführt und parallel hierzu alle anderen Prozessoren und Peripheriegeräte keine Datentransfers durchführen. In einer weiteren Messung wird anschließend eine Datenübertragung parallel zu den gemessen Zugriffen durchgeführt. Diese Datenübertragung darf dabei nicht von Knoten ausgeführt werden, die bereits im gemessenen Datentransfer involviert sind, ausgenommen des Falls, dass diese Knoten als Bindeglied zwischen einzelnen Punkt-zu-Punkt Verbindungen des Verbindungsnetzwerks dienen. Aus dem Vergleich der maximalen Zugriffszeiten beider Messungen lässt sich anschließend der Einfluss des Verbindungsnetzwerks ermitteln.

Ziel bei der Bestimmung der maximalen Latenzzeiten bei einem Zugriff auf einen entfernten Knoten ist es, die Latenzzeiten

- für alle möglichen Kombinationen aus Quell- und Zielknoten und
- unter allen möglichen Beeinflussungen paralleler Datentransfers

angeben zu können. Prinzipiell kann jede Kombination aus Quell- und Zielknoten in Verbindung mit allen möglichen Varianten paralleler Datenübertragungen einzeln gemessen werden; ein solches Vorgehen ist jedoch sehr aufwändig und mit zunehmender Knoten- und Prozessorenanzahl praktisch nicht mehr durchführbar. Deshalb soll im Folgenden untersucht werden, welche Messungen in einem NUMA-System tatsächlich durchgeführt werden müssen.

Für Quell- und Zielknoten können die gleichen Annahmen getroffen werden, die bereits bei der Untersuchung der Einflüsse bei Zugriffen innerhalb eines Knotens gemacht wurden (siehe Seite 54). Insbesondere kann in einem NUMA-Multiprozessorsystem auf PC-Basis davon ausgegangen werden, dass auf den Knoten die gleichen Latenzzeiten auftreten, unter der Voraussetzung, dass die Hardwarebestückung der Knoten identisch ist. Unterscheidet sich die Hardwarebestückung, so muss der Einfluss jeder Variante einzeln überprüft werden — alternativ kann auch nur die Variante überprüft werden, die zu den größtmöglichen Latenzzeiten führt. Falls beispielsweise die Knoten mit unterschiedlichen Peripheriegeräten bestückt sind ist es ausreichend, die Latenzzeiten des Knotens zu bestimmen, dessen Peripheriegeräte die größte Datenlast erzeugen. Ist eine Hardwarebestückung auf einem Knoten nicht vorhanden — dies könnte beispielsweise eine Peripherieanbindung oder ein Prozessor sein — so muss derjenige Kno-

ten untersucht werden, der die entsprechende Hardwarekomponente beinhaltet. Die maximale Latenzzeit für den anderen Knoten entspricht dann dem Fall, dass diese Hardwarekomponente keine Datenlast erzeugt.

Weiterhin kann für das Verbindungsnetzwerk angenommen werden, dass identische Latenzzeiten auftreten, falls die Zugriffe auf einen entfernten Knoten über gleich lange Verbindungspfade des Netzes und bei gleicher Last parallel stattfindender Datentransfers erfolgen. Voraussetzung hierbei ist, dass die betrachteten Verbindungspfade hinsichtlich ihrer jeweils maximal möglichen Übertragungsrate und hinsichtlich ihrer Topologie äquivalent sind.

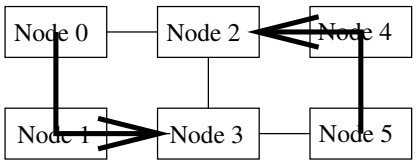


Abbildung 4.17: Äquivalente Zugriffe im Verbindungsnetzwerk

Abbildung 4.17 verdeutlicht das angenommene Szenario. Dargestellt ist ein NUMA-Multiprozessorsystem mit sechs Knoten. Wird die maximale Ausführungszeit eines Transfers von Knoten 0 über Knoten 1 auf Knoten 3 bestimmt, so ist für einen Zugriff von Knoten 5 auf Knoten 2 über Knoten 4 mit den gleichen maximal möglichen Latenzzeiten zu rechnen. Voraussetzung ist, dass hier jeweils die gleiche parallele Last anderer Prozessoren und Peripheriegeräte vorhanden ist. Unterscheidet sich diese maximal mögliche Last, so kann nur der Übertragungsweg mit den größeren Beeinflussungen untersucht werden — für den anderen Datenpfad des Netzwerks ergibt sich dann eine (geringe) Überabschätzung der maximalen Zugriffszeiten.

Separation der Einflüsse

Bei der Bestimmung der maximalen Zugriffszeiten in SMP-Systemen wurde festgestellt, dass sich unter gleicher Belastung ein lineares Verhalten der jeweiligen Zugriffszeit ergibt (vergleiche hierzu Seite 37). Dabei wurde für jede mögliche Kombination aus parallelen Einflüssen der Faktor m_{WCAT} und der Offset $t_{WCAT,1}$ der Gleichung 4.8 mittels Messung bestimmt.

Da bei NUMA-Multiprozessorsystemen ebenfalls von einem fairen Verfahren bei der Ressourcenzuteilung ausgegangen werden kann, soll zur Bestimmung maximaler Zugriffszeiten Gleichung 4.8 erweitert werden. Hierbei werden die Einflüsse in Quell- und Zielknoten und die Einflüsse im Verbindungsnetzwerk aufgrund paralleler Datentransfers zunächst voneinander getrennt betrachtet. Beispielsweise wirkt sich ein Zugriff einer CPU auf ihren lokalen Speicher nicht auf die Einflüsse im Verbindungsnetzwerk aus, falls hier eine andere CPU dieses Knotens auf einen entfernten Knoten zugreift. Unter worst-case Bedingungen verlängert sich beim Zugriff über das Verbindungsnetzwerk die Kette kaskadierter Round-Robin-Arbitrierungen. Gleichung 4.8 wird deshalb wie folgt erweitert:

$$\begin{aligned} t_{WCAT}(x) &= t_{WCAT,1} + m_{WCAT}(x-1) \\ &= t_U\tau_l\tau_n\tau_r k_{lnr} + (m_U\mu_l\mu_n\mu_r l_{lnr})(x-1); \qquad x \in \mathbb{N} \end{aligned} \tag{4.10}$$

mit

$$t_{WCAT,1} = t_U \tau_l \tau_n \tau_r k_{lnr} \quad (4.11)$$

$$m_{WCAT} = m_U \mu_l \mu_n \mu_r l_{lnr} \quad (4.12)$$

t_U und m_U bestimmen hierbei die maximalen Zugriffszeiten von einem Knoten A auf einen Knoten B im ungestörten Fall *ohne* dem Einfluss paralleler Datenübertragungen, wobei $A,B \in \mathbb{K}$ und $A \neq B$. $\mathbb{K}$ ist hierbei die Menge aller Knoten (vergleiche hierzu auch Seite 54).

Die Parameter k_{lnr} und l_{lnr} beschreiben die Korrelation der Datenpfade. Für eine einfache Abschätzung können $k_{lnr} = 1$ und $l_{lnr} = 1$ gesetzt werden. Trotzdem sollte vor dem Realzeiteinsatz eines NUMA-Multiprozessorsystems die Wechselbeziehung der Einflüsse zwischen Quell- und Zielknoten und dem Verbindungsnetzwerk überprüft werden.

Die Parameter τ_l, τ_n und τ_r definieren den Einfluss paralleler Datenübertragungen auf den ersten Zugriff von einer Serie aufeinanderfolgender Ressourcenzugriffe auf einen entfernten Knoten. Diese Parameter sind abhängig von den parallel ausgeführten Transfers, so dass jeder Kombination aus parallel stattfindenden Übertragungen im allgemeinen Fall ein anderes Wertetripel zugewiesen werden muss. τ_l beschreibt dabei den Einfluss paralleler Datenübertragungen in dem Knoten, der die Zugriffe ausführt (Knoten A). Der Einfluss paralleler Datenübertragungen im Verbindungsnetzwerk wird durch den Parameter τ_n bestimmt und die Einflüsse paralleler Datenübertragungen auf den ersten Zugriff im Zielknoten (Knoten B) wird durch den Parameter τ_r beschrieben. Für den Wertebereich von τ_l, τ_n und τ_r gilt:

$$\tau_l, \tau_n, \tau_r \geq 1 \quad (4.13)$$

Die Parameter μ_l, μ_n und μ_r definieren den Einfluss paralleler Datenübertragungen auf alle folgenden Zugriffe auf den entfernten Knoten. μ_l definiert hierbei die Einflüsse paralleler Datentransfers im lokalen Knoten (Knoten A), μ_n definiert die Einflüsse im Verbindungsnetzwerk und μ_r beschreibt die Einflüsse paralleler Übertragungen im Zielknoten (Knoten B). Für den Wertebereich von μ_l, μ_n und μ_r gilt:

$$\mu_l, \mu_n, \mu_r \geq 1 \quad (4.14)$$

Auch diese Parameter sind abhängig von den jeweils parallel ausgeführten Datenübertragungen, so dass sich bei geänderter Lastsituation unterschiedliche Wertetripel einstellen.

Bestimmung der Parameter

Bevor die einzelnen Parameter der Gleichung 4.10 bestimmt werden können, muss zunächst untersucht werden, welche Belastungen in Quell- und Zielknoten und im Verbindungsnetzwerk auftreten können. Für diese parallel stattfindenden Einflüsse sind anschließend die entsprechenden Parameter der Gleichung 4.10 durch Messungen zu ermitteln.

Bei der Bestimmung der Menge $\mathbb{D}$ der Datenpfade in den Quell- bzw. Zielknoten sind alle möglichen Datenübertragungen zu berücksichtigen, wobei die Anbindung des jeweiligen Knotens an das Verbindungsnetzwerk nicht außer Acht gelassen werden darf.

Weiterhin müssen nur diejenigen Knoten untersucht werden, bei denen die entsprechenden Beeinflussungen auch tatsächlich auftreten — ist in einem Knoten nur eine Untermenge der Beeinflussungen eines anderen Knotens vorhanden, so braucht dieser Knoten nicht extra untersucht werden. Dessen Parameter lassen sich aufgrund der Ähnlichkeit der Knoten des NUMA-Systems aus den Parametern des untersuchten Knotens ableiten. Formalisiert bedeutet dies, dass alle Knoten zu untersuchen sind, auf die Gleichung 4.9 zutrifft. Die so ermittelten Knoten müssen dann sowohl als Quell-, als auch als Zielknoten dienen.

Ebenso wie in den Knoten müssen im Verbindungsnetzwerk alle Datenpfade untersucht werden, bei denen sich überlappende Datenpfade mit anderen Knoten einstellen. Das im Folgenden vorgestellte Verfahren kann nur für statische Verbindungsnetzwerke verwendet werden; bei dynamischen Verbindungsnetzwerken müssen alle Kombinationen aus Ein- und Ausgängen getestet werden, falls die Architektur des jeweiligen Netzwerks nicht bekannt ist. Ansonsten lassen sich auch hier ggf. vorhandene Symmetrien ausnutzen.

Sei $\mathbb{V} = \{v_0, v_1, ...\}$ die Menge aller Pfade des statischen Verbindungsnetzwerks. Damit lässt sich für jeden Transfer von einem Quellknoten q mit einem Zielknoten z die Menge $\mathbb{T}_{qz} \subset \mathbb{V}$ der benötigten Pfade des Verbindungsnetzwerks angeben, wobei $q, z \in \mathbb{K}$ und $q \neq z$ gilt.

Einen Einfluss auf die Datenübertragung von einem Knoten q_1 mit einem Zielknoten z_1 haben alle Transfers von beliebigen Knoten q_2 und z_2, die sich mit dem betrachteten Datentransfer überlagern:

$$\mathbb{T}_{q_1 z_1} \cap \mathbb{T}_{q_2 z_2} \neq \emptyset \tag{4.15}$$

mit $\{q_1,z_1,q_2,z_2\} \subset \mathbb{K}$. Hierbei gilt, dass Quell- und Zielknoten jeweils verschieden sein müssen, also $\{q_1,z_1\} \cap \{q_2,z_2\} = \emptyset$, da ausschließlich die Einflüsse des Verbindungsnetzwerks bestimmt werden sollen. Die Beeinflussungen durch identische Quell- oder Zielknoten durch das Verbindungsnetzwerk werden den Beeinflussungen *im* lokalen bzw. entfernten Knoten zugeordnet. Grund hierfür ist, dass ausschließlich Zugriffszeiten gemessen werden können und nicht unterschieden werden kann, ob diese Beeinflussungen im Verbindungsnetzwerk oder auf dem entsprechenden Knoten auftreten.

Wurden allen Transfers jene Transfers zugeordnet, die die jeweilige Datenübertragung beeinflussen könnten, so müssen äquivalente Zugriffe auf das Verbindungsnetzwerk aussortiert werden, die sich aus der Symmetrie des Netzwerks ergeben. Unter der Voraussetzung, dass alle Pfade v_i des Verbindungsnetzwerks die gleichen physikalischen Eigenschaften aufweisen, kann folgendes Verfahren angewandt werden: Die Mengen $\mathbb{T}_{qz}$ werden hinsichtlich der Anzahl ihrer Elemente sortiert. Anschließend wird der Vertreter jeder Sortierung ausgewählt, der den größtmöglichen parallelen Beeinflussungen ausgesetzt ist. Falls hier mehrere $\mathbb{T}_{qz}$ äquivalenten Belastungen unterworfen sind, ist die Auswahl beliebig.

Wurden alle möglichen Beeinflussungen durch parallel stattfindende Datenübertragungen in den Knoten und im Verbindungsnetzwerk bestimmt, so müssen die Auswirkungen dieser parallelen Datenübertragungen auf die maximale Zugriffszeit von Realzeitsoftware mittels Messung ermittelt werden. Dabei ist zu beachten, dass die relevanten Einflüsse — also jene Einflüsse, die sich auch tatsächlich auf Laufzeiten auswirken — auch in Kombination untersucht werden müssen.

Bei der Ermittlung der benötigten Parameter sind zunächst die Zugriffszeiten zu bestimmen, die ein Knoten A auf einen *benachbarten* Knoten B benötigt, falls *keine* parallelen Transfers anderer Prozessoren oder Peripheriegeräte auf die benötigten Ressourcen zugreifen. Aus diesen Messwerten lassen sich die Parameter t_U und m_U aus Gleichung 4.10 ermitteln.

Anschließend werden die Einflüsse im Quell- und im Zielknoten und im Verbindungsnetzwerk — jeweils getrennt — untersucht. Dabei wird ausschließlich der Einfluss der zu untersuchenden parallelen Datenübertragung generiert, alle anderen Prozessoren und Peripheriegeräte stoßen nach wie vor keine zusätzlichen Datenübertragungen an, die sich mit dem untersuchten Zugriff überlagern würden. Aus den ermittelten Messwerten lassen sich die Parameter t_m (Offset) und m_m (Steigung) der resultierenden Geraden ermitteln. Im Falle der Untersuchung der Einflüsse im lokalen Knoten folgt daraus

$$\tau_l = t_m/t_U \tag{4.16}$$

$$\mu_l = m_m/m_U \tag{4.17}$$

für die jeweilige Belastung. Bei der Ermittlung der Parameter τ_n, μ_n und τ_r, μ_r sind beide Gleichungen analog zu verwenden.

Wurden diese Parameter ermittelt, müssen anschließend die Datentransfers, die sich auf die Zugriffe einer CPU auswirken, in Kombination untersucht werden. Hierzu werden die Auswirkungen der jeweiligen Belastungsszenarien vermessen und die Parameter k_{lnr} und l_{lnr} bestimmt.

4.3.2.4 HyperTransport

In Multiprozessorsystemen auf PC-Basis wird im zunehmenden Maße HyperTransport (HT) verwendet. Insbesondere sind dies alle Multiprozessorsysteme, die mit AMD Opterons bestückt sind. HT ermöglicht eine schnelle Verbindung zwischen den einzelnen Chips eines Rechensystems. Insbesondere können über HT die CPUs mit den Peripheriegeräten, sowie die CPUs bzw. die Knoten untereinander verbunden werden. Bei HyperTransport gibt es keine Steckverbindungen — es ist ausschließlich für Onboard-Verbindungen geeignet.

Der schematische Aufbau einer HT-Architektur ist in Abbildung 4.18 für ein Quad-Multiprozessorsystem dargestellt. Jeder Knoten (Node 0 – Node 3) enthält eine CPU und einen Teil des Hauptspeichers. Die einzelnen Knoten sind untereinander über kohärente HT-Links verbunden. Das gewählte Verbindungsnetzwerk entspricht hierbei der Gitterstruktur.

Zusätzlich können die Knoten Peripheriegeräte beinhalten; in Abbildung 4.18 enthalten Node 0 und Node 1 Peripheriegeräte. Die Anbindung der Peripheriegeräte bzw. von Peripheriebussen erfolgt ebenfalls über HT. Hierzu existieren *Tunnel*- und *Cave*-Devices:

- Ein HT-Tunnel ist ein Gerät, das zwei HT-Links besitzt und das keine HyperTransport-Bridge ist. Somit können über einen Tunnel Peripheriegeräte an HT angebunden werden. Gleichzeitig können über den zweiten Link weitere HT-Komponenten angeschlossen werden. In Abbildung 4.18 sind zwei PCI-X- und ein AGP-Tunnel dargestellt.

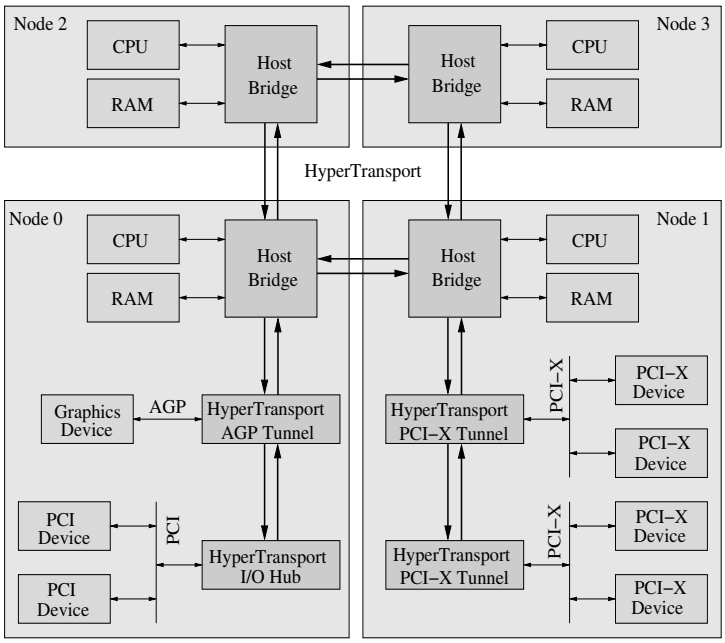


Abbildung 4.18: HyperTransport-Architektur

- Eine HT-Cave ist ein Gerät, das nur einen HT-Link besitzt. Somit beendet eine Cave eine Kette aus HT-Geräten. In Abbildung 4.18 ist der HT-I/O-Hub eine Cave.

Alle im Chipsatz integrierten Peripheriegeräte können als HT-Devices an eine HT-Kette angebunden werden. Alle sonstigen Peripheriegeräte — und teilweise auch jene, die in einer entsprechenden Bridge integriert sind — werden über einen Bus an das Rechensystem angeschlossen, der auch Steckverbindungen unterstützt. Hierzu existieren Bridges, die das jeweilige Busprotokoll an HT anbinden können. Insbesondere sind hier HT-to-AGP- und HT-to-PCI-X-Bridges zu nennen. Eine HT-to-PCI-Bridge ist meist im HT-I/O-Hub integriert. Da PCI-Express in letzter Zeit an zunehmender Bedeutung gewinnt, sind auch für dieses Protokoll entsprechende Bridges verfügbar.

Die HT-Geräte werden sequentiell miteinander verbunden. Jede Kette bildet hierbei einen Bus; pro Kette können bis zu 32 Geräte angeschlossen werden. Mehrere HT-Busse werden über Bridges miteinander verbunden. Die Anbindung von HT an die Prozessoren erfolgt über die Host-Bridge des jeweiligen Knotens.

Für Realzeitsysteme relevante Eigenschaften von HyperTransport

Die einzelnen Geräte werden bei HT über Punkt-zu-Punkt-Verbindungen miteinander verbunden, wobei für jede Übertragungsrichtung ein eigener Simplexkanal vorhanden ist. Die Kommunikation erfolgt durch den Austausch von Paketen. Hierfür sind in jedem HT-Gerät entsprechende Pufferspeicher vorhanden.

Für die Kommunikation wird pro Simplexkanal der CAD-Bus (Command/Address/Data), ein oder mehrere CLK-Signale (Clock) und das CTL-Signal (Control) benötigt. Weitere Signale, insbesondere Arbitrierungssignale, finden keine Verwendung:

- Der CAD-Bus besitzt 2, 4, 8, 16 oder 32 Signalleitungen, über die die Kommunikation zwischen den einzelnen HT-Geräten erfolgt. Die Anzahl der verwendeten Signalleitungen wird vom Hersteller des Chipsatzes festgelegt, wobei in Abhängigkeit des erwarteten Datenaufkommens die Anzahl der Signalleitungen pro Simplexkanal unterschiedlich sein darf.

- Mittels CLK wird der Takt des HT-Busses definiert. Möglich sind Taktraten von 200MHz bis 1600MHz.[6)]

- CTL legt fest, ob es sich um ein Daten- oder um ein Kontroll-Paket handelt.

HT kennt zwei Arten von Paketen: Daten- und Kontroll-Pakete. Daten-Pakete haben eine Größe zwischen 4 und 64 Byte und enthalten reine Nutzdaten *ohne* zusätzliche Header- bzw. Steuerinformationen. Kontroll-Pakete haben eine Größe von 4, 8 oder 12 Byte in Abhängigkeit von der verwendeten Adressbreite (keine Adresse, 40 Bit- oder 64 Bit-Adresse). Kontroll-Pakete sind unterteilt in Informations-, Anforderungs- und Antwort-Pakete:

- Informations-Pakete *(Information Packets)* haben eine Größe von 4 Byte und werden nur zwischen zwei benachbarten HT-Geräten verschickt. Insbesondere werden diese Pakete für die Flusskontrolle benötigt.

- Anforderungs-Pakete *(Request Packets)* haben in Abhängigkeit von der Adressbreite eine Größe von 4, 8 oder 12 Byte. Diese Pakete können eine Schreib- oder Leseanforderung, oder eine Nachricht (Broadcast, Flush, Fence) beinhalten. Falls Daten übertragen werden sollen — beispielsweise im Zuge eines Schreibzugriffs — folgt diesem Paket *unmittelbar* ein Daten-Paket.

- Antwort-Pakete *(Response Packets)* haben eine Größe von 4 Byte und werden einem Anforderungs-Paket zugeordnet. Falls Daten übertragen werden müssen — beispielsweise als Antwort auf eine Leseanforderung — folgt diesem Paket *unmittelbar* ein Daten-Paket.

In den Routing Tabellen kohärenter HT-Links — also der Links, die das Verbindungsnetzwerk zwischen den einzelnen Knoten aufspannen — wird zwischen Anforderungs-, Antwort- und *Broadcast*-Paketen unterschieden. Letztere sind vom Paket-Typ ein Anforderungs-Paket. Broadcast-Pakete haben jedoch die Eigenschaft, dass sie an alle Busteilnehmer, also an alle Knoten des Verbindungsnetzwerks, versandt werden.

Aufgrund des verwendeten Paketmechanismus muss — im Gegensatz zu PCI — der HT-Bus nicht auf die angeforderten Daten warten; satt dessen können andere Pakete transportiert werden. Dadurch erhöht sich der Durchsatz von HT im Vergleich zu PCI, maximale Zugriffszeiten

[6)] HyperTransport Link Specification 2.00a Draft3; ab 1400MHz jedoch elektrisch noch nicht vollständig spezifiziert.

werden jedoch nicht verringert. Aufgrund der Aufteilung der Transaktionen in Anforderung und Antwort ist im Gegenteil eher mit höheren Latenzzeiten im Worst-Case zu rechnen.

Um Latenzzeiten zu verringern bietet HT die Möglichkeit, Kontroll- und Daten-Pakete ineinander zu verschachteln (Interleaving). Hier kann an jeder 4-Byte Grenze eines Daten-Pakets ein Kontroll-Paket in den Datenstrom eingespeist werden — die Pakete werden hierbei mit dem Signal CTL unterschieden. Kontroll-Pakete können nicht unterbrochen werden, eine mehrfache Unterbrechung eines Daten-Pakets ist jedoch möglich. Weiterhin dürfen nur solche Kontroll-Pakete in den Datenstrom eingefügt werden, denen kein Daten-Paket folgen muss.

Damit HT in Hinblick auf die *Reihenfolge* der ausgeführten Transaktionen zu PCI kompatibel bleibt, müssen zwei Geräte, die sich an der gleichen HyperTransport-Kette befinden, *ausschließlich* über eine HyperTransport-Bridge miteinander kommunizieren. Soll beispielsweise bei der in Abbildung 4.18 in Node 0 dargestellten HyperTransport-Kette ein Daten-Paket vom I/O-Hub an den AGP-Tunnel — und damit an den Grafikadapter — übergeben werden, wird das Paket zunächst vom I/O-Hub an den AGP-Tunnel übertragen. Dieser reicht das Paket weiter an die HyperTransport-Bridge. In der Bridge wird das Paket umgelenkt und die HyperTransport-Kette hinuntergeschickt. Da das folgende Gerät der AGP-Tunnel ist, hat das Paket sein Ziel erreicht. Mit der HT-Revision 1.1 DirectPaket Spezifikation kann auch die direkte Kommunikation zwischen zwei HT-Geräten realisiert werden.

Weiterhin bieten HT die Möglichkeit der isochronen Datenübertragung. Isochrone Datenübertragungen werden vor allen anderen Transfers behandelt. Voraussetzung ist, dass die beteiligten HT-Geräte diese Art der Kommunikation unterstützen. Die isochrone Datenübertragung ist ausgenommen vom Prinzip der Fairness auf dem HT-Bus — theoretisch ist es möglich, dass hier nicht-isochrone Datenübertragungen nie zum Zuge kommen.

Bei den nicht-isochronen Datenübertragungen findet eine *faire Busbelegung* Verwendung. Falls das Bussystem ausgelastet ist, darf ein HT-Tunnel Pakete nur mit maximal der Rate in den Datenstrom einschleusen, mit der das aktivste Gerät unterhalb des Tunnels Pakete sendet. Diese Rate wird dynamisch bestimmt. Falls die Geräte unterhalb des Tunnels ihr Sendeverhalten ändern, wird die maximal erlaubte Einschleus-Rate neu bestimmt.

Folgerungen für die Bestimmung maximaler Zugriffszeiten

Obwohl die Spezifikation von HT offen gelegt ist, können maximale Zugriffszeiten über HT nicht direkt angegeben werden. Grund hierfür ist insbesondere, dass HT versucht, die Auslastung der einzelnen Links zu maximieren. Worst-case Zugriffszeiten werden dadurch jedoch nicht verringert. Insbesondere existieren die folgenden Aspekte, die die Bestimmung maximaler Zugriffszeiten erschweren:

- Das Arbitrierungsverfahren ist fair; die maximal erlaubte Rate, mit der Pakete in einen Datenstrom eingespeist werden dürfen, hängt von der Aktivität der restlichen am HT-Bus vorhandenen Geräte ab.

- Obwohl die Paketgröße eines Daten-Pakets maximal 64 Byte beträgt, hängt die maximale Übertragungsdauer von der Anzahl der Unterbrechungen durch Kontroll-Pakete ab. Diese wiederum wird von der Aktivität vorhandener Prozessoren und Peripheriegeräte bestimmt.
- Aufgrund des verwendeten Anforderungs-/Antwort-Verfahrens können sich im Vergleich zu PCI maximale Zugriffszeiten (geringfügig) verlängern. Grund hierfür ist, dass das Antwort-Paket aufgrund parallel stattfindender Transfers verzögert wird. Die Dauer dieser Verzögerung hängt vom Datenaufkommen anderer HT-Geräte ab.
- Die Größe vorhandener Pufferspeicher, die Breite der vorhandenen HT-Links und die verwendeten Taktraten hängen vom Hersteller des Chipsatzes bzw. vom Hersteller des Mainboards ab.
- Maximale Zugriffszeiten hängen von der Reaktionszeit der beteiligten Komponenten ab.

Da die Zugriffszeiten bei HT von teilweise unbekannten Parametern abhängen, ist es auch hier sinnvoll, die maximalen Zugriffszeiten anhand von Messungen zu bestimmen. Aufgrund der HT-Architektur ist dabei folgendes zu beachten:

- Die Richtung des durchgeführten Datentransfers (lesend/schreibend) muss berücksichtigt werden. Insbesondere hängt hiervon ab, ob Daten-Pakete miteinander konkurrieren und ob Kontroll- und Daten-Pakete verschachtelt werden.
- Der Einfluss parallel angestoßener Datentransfers muss das gewünschte worst-case Szenario herbeiführen. Jedes am Bus befindliche HT-Gerät muss hier Daten mit der jeweils maximal möglichen Rate senden und empfangen. Auch hier kann die Übertragungsrichtung einen Einfluss auf maximale Zugriffszeiten haben.
- Es muss überprüft werden, ob isochrone Datenübertragungen möglich sind. Bei der Bestimmung maximaler Zugriffszeiten muss der Einfluss dieser Datenübertragungen berücksichtigt werden.
- Es müssen nur diejenigen HT-Geräte berücksichtigt werden, die in der gleichen Bus- bzw. Baumstruktur angebunden sind.

Aufgrund der fairen Protokolle kann auch bei der Verwendung von HyperTransport von einem linearen Verhalten der Zugriffszeiten ausgegangen werden. Insbesondere kann auch hier Gleichung 4.10 verwendet werden. Voraussetzung dabei ist, dass keine isochronen Datenübertragungen durchgeführt werden.

4.3.2.5 Bestimmung der Realzeiteigenschaften eines Quad-Opterons

Im Folgenden sollen die Realzeiteigenschaften eines Quad-Opteron Multiprozessorsystems untersucht werden. Jeder Knoten ist mit einem Prozessor und mit Speicher bestückt. An zwei Knoten sind Peripheriegeräte angeschlossen. Die Knoten selbst sind in einer Gitterstruktur angeordnet. Als Verbindungsprotokoll zwischen den Knoten und zur Anbindung der Peripheriegeräte wird HyperTransport verwendet. AGP ist auf dem untersuchten System nicht verfügbar; der

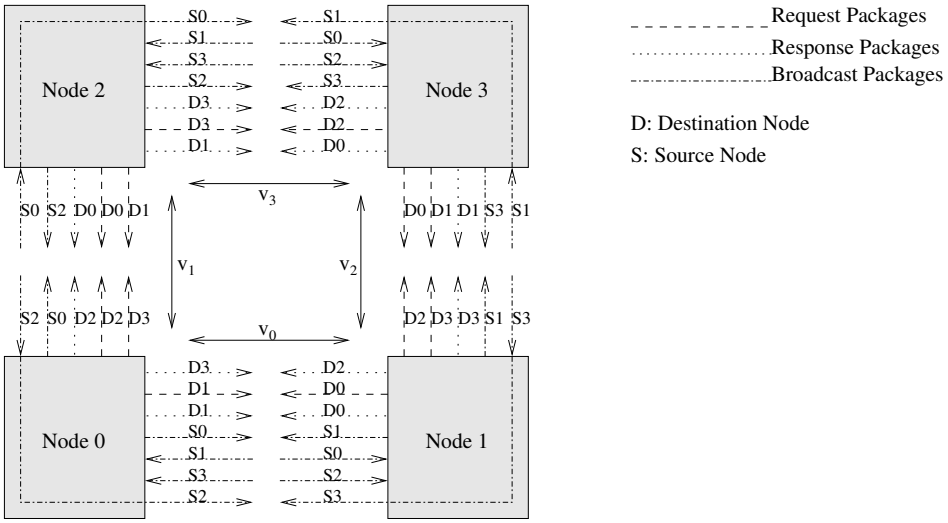


Abbildung 4.19: Routen des Verbindungsnetzwerks

Grafikadapter ist statt dessen über PCI an das System angebunden. Das Multiprozessorsystem entspricht somit — bis auf AGP — dem in Abbildung 4.18 dargestellten Szenario.

Isochrone Datentransfers werden von dem eingesetzten HyperTransport-Bus nicht verwendet. Dies bedeutet, dass die verwendeten Arbitrierungsverfahren fair sind und die Gleichungen 4.8 und 4.10 benützt werden können.

Jeder Knoten besitzt zwei Links in das Verbindungsnetzwerk. Wie die Pakete von einem Knoten zu einem anderen Knoten gelangen, wird durch Routing-Tabellen festgelegt. Für das betrachtete System ergibt sich das folgende Szenario: Pakete zum benachbarten Knoten werden direkt über den entsprechenden Link gesendet. Pakete, die zu dem jeweils nicht direkt erreichbaren Knoten gesendet werden sollen, werden über beide Links geschickt. Hierbei werden Anforderungs-Pakete über den einen und Antwort-Pakete über den anderen Link gesendet. Vereinfacht bedeutet dies, dass Schreibzugriffe über den einen Link gesendet werden. Die Antwort auf eine Leseanforderung wird über den anderen Link übertragen.

Das verwendete Routing-Verfahren ist in Abbildung 4.19 illustriert. Hierbei ist für jeden Knoten angezeichnet, welche Pakete auf welchem Pfad des Verbindungsnetzwerks versandt werden. Welcher Pfad verwendet wird, entscheidet bei Anforderungs- und Antwortpaketen der Zielknoten (D0-D3) des Pakets. Bei Broadcast-Paketen wird diese Entscheidung anhand des Quellknotens (S0-S3) des jeweiligen Pakets getroffen.

Alle im System vorhandenen Datenpfade, die auf den Knoten auftreten können, sind in Tabelle 4.8 aufgelistet. Die Menge aller Knoten ist hierbei $\mathbb{K} = \{0,1,2,3\}$. Wie aus Tabelle 4.8 ersichtlich ist, können die Untersuchungen der im System auftretenden Latenzzeiten prinzipiell auf Knoten 0 oder Knoten 1 beschränkt werden, da hier jeweils alle möglichen Datenpfade auftreten können. Da jedoch die an Knoten 1 angeschlossenen Peripheriegeräte (SCSI, Ethernet, FireWire) das höhere Datenaufkommen erzeugen, wird zur Untersuchung der gegenseitigen Beeinflussungen möglicher Datentransfers Knoten 1 gewählt. Dieser Knoten wird hierbei zur

Pfad	Beschreibung	Knoten
d_0	CPU greift auf lokalen Speicher zu	0,1,2,3
d_1	Peripheriegeräte greifen auf lokalen Speicher zu	0,1
d_2	HT-Link 1 greift auf lokalen Speicher zu	0,1,2,3
d_3	HT-Link 2 greift auf lokalen Speicher zu	0,1,2,3
d_4	HT-Link 1 greift auf lokale Peripheriegeräte zu	0,1
d_5	HT-Link 2 greift auf lokale Peripheriegeräte zu	0,1
d_6	CPU greift auf HT-Link 1 zu	0,1,2,3
d_7	CPU greift auf HT-Link 2 zu	0,1,2,3
d_8	CPU greift auf lokale Peripheriegeräte zu	0,1
d_9	Peripheriegeräte greifen auf HT-Link 1 zu	0,1
d_{10}	Peripheriegeräte greifen auf HT-Link 2 zu	0,1

Tabelle 4.8: Mögliche Datenpfade auf den Knoten des NUMA-Systems

Untersuchung der Zugriffe innerhalb eines Knotens, als auch für die Bestimmung der Zugriffszeiten auf entfernte Knoten verwendet. Hierbei wird Knoten 1 sowohl als Quell-, als auch als Zielknoten betrachtet.

Das Verbindungsnetzwerk beinhaltet die Pfade $\mathbb{V} = \{v_0,v_1,v_2,v_3\}$. Damit lassen sich für jeden Zugriff von einem Knoten auf einen anderen Knoten die benötigten Datenpfade im Verbindungsnetzwerk angeben. Obwohl die Datenpfade ggf. davon abhängen, ob das jeweilige Paket eine Anforderung oder eine Antwort ist, wird zwischen diesen beiden Transaktionen nicht unterschieden — statt dessen werden alle verwendeten Verbindungspfade berücksichtigt. Die Richtung der Datentransfers (lesend/schreibend) wird bei der Ermittlung der jeweiligen Parameter zur Abschätzung von worst-case Zugriffszeiten mit berücksichtigt. Die verwendeten Verbindungspfade des Netzwerks sind für jede mögliche Datenübertragung in Tabelle 4.9 aufgeführt, die anhand der Routing-Informationen aus Abbildung 4.19 ermittelt wurden.

Transfer	verwendete Pfade	Transfer	verwendete Pfade
$\mathbb{T}_{01}$	v_0	$\mathbb{T}_{20}$	v_1
$\mathbb{T}_{02}$	v_1	$\mathbb{T}_{21}$	v_0,v_1,v_2,v_3
$\mathbb{T}_{03}$	v_0,v_1,v_2,v_3	$\mathbb{T}_{23}$	v_3
$\mathbb{T}_{10}$	v_0	$\mathbb{T}_{30}$	v_0,v_1,v_2,v_3
$\mathbb{T}_{12}$	v_0,v_1,v_2,v_3	$\mathbb{T}_{31}$	v_2
$\mathbb{T}_{13}$	v_2	$\mathbb{T}_{32}$	v_3

Tabelle 4.9: Mögliche Datenpfade im Verbindungsnetzwerk

Aus Gleichung 4.15 folgt, dass sich die Datenpfade von $\mathbb{T}_{03}$ oder $\mathbb{T}_{30}$ mit $\mathbb{T}_{12}$ und $\mathbb{T}_{21}$ überlagern. Gleiche Aussage gilt für $\mathbb{T}_{12}$ oder $\mathbb{T}_{21}$ mit $\mathbb{T}_{30}$ und $\mathbb{T}_{03}$. Da diese Beeinflussungen alle als äquivalent betrachtet werden können, ist es ausreichend, wenn nur die Beeinflussungen von Transfer $\mathbb{T}_{12}$ mit parallelen Datentransfers von Knoten 0 mit Knoten 3 analysiert werden.

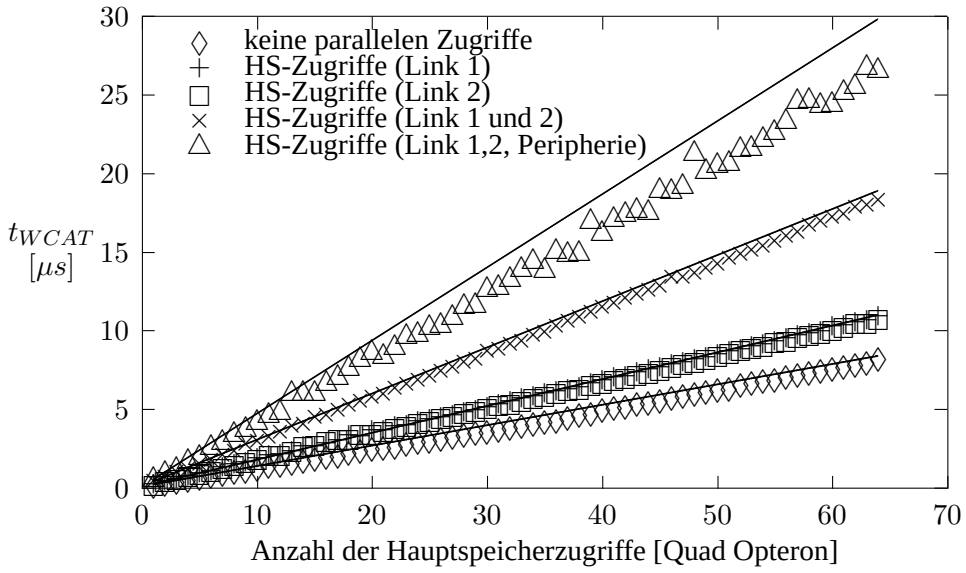


Abbildung 4.20: Zugriffe innerhalb eines Knotens (Hauptspeicherzugriffe)

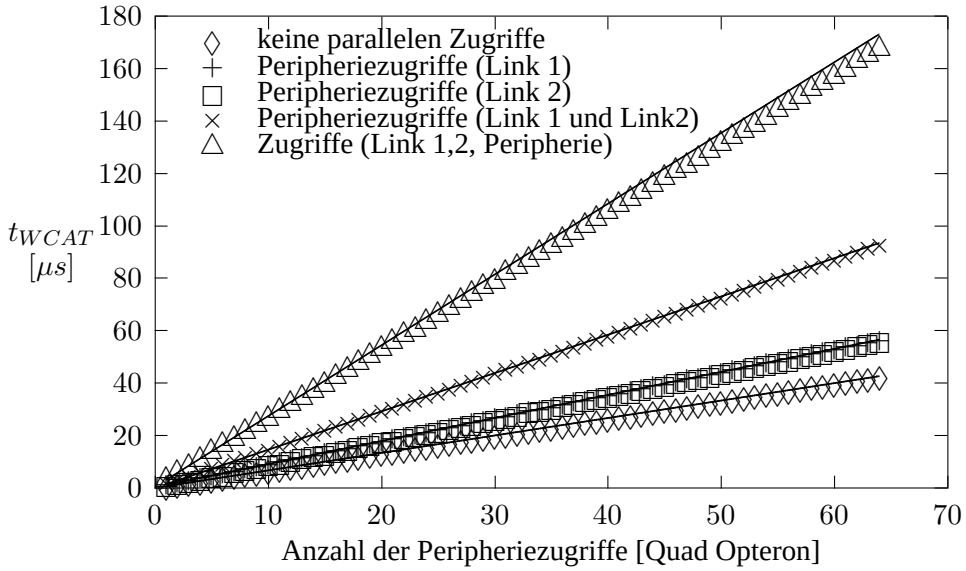


Abbildung 4.21: Zugriffe innerhalb eines Knotens (Peripheriezugriffe)

Bestimmung der Latenzzeit beim Zugriff auf den lokalen Knoten

In den folgenden Untersuchungen werden die Latenzzeiten analysiert, die bei Zugriffen innerhalb des lokalen Knotens auftreten können. Hierbei werden Zugriffe einer CPU auf ihren lokalen Hauptspeicher und auf die lokalen Peripheriegeräte durchgeführt. Um vergleichbare Messwerte zu erhalten, wird immer auf die gleiche Portadresse zugegriffen.

Während der Zugriffe kann die CPU durch andere Transfers beeinflusst werden. Diese Transfers können dabei von lokalen Peripheriegeräten, oder von entfernten Prozessoren und Peripheriege-

Zugriff	parallele Zugriffe	$t_{WCAT,1}[\mu s]$	$m_{WCAT}[\mu s]$
Hauptspeicherzugriffe (d_0)	—	0,24	0,13
	d_1	0,28	0,14
	d_2	0,29	0,19
	d_3	0,29	0,19
	d_2, d_3	0,45	0,29
	d_1, d_2, d_3	0,55	0,47
	d_4	0,24	0,13
	d_5	0,24	0,13
	d_9	0,24	0,13
	d_{10}	0,24	0,13
Peripheriezugriffe (d_8)	—	0,74	0,67
	d_1	1,65 (0,74)	0,90 (0,68)
	d_2	0,74	0,67
	d_3	0,74	0,67
	d_4	1,20 (0,74)	0,88 (0,67)
	d_5	1,20 (0,74)	0,88 (0,67)
	d_4, d_5	1,46 (0,74)	1,46 (0,67)
	d_1, d_4, d_5	3,12 (0,74)	2,70 (0,69)
	d_9	1,57 (0,74)	0,90 (0,67)
	d_{10}	1,59 (0,74)	0,90 (0,67)

Tabelle 4.10: Zugriffe innerhalb eines Knotens

räten initiiert werden. Die gemessenen Latenzzeiten sind in Tabelle 4.10 zusammengefasst. Bei der Untersuchung der maximalen Zugriffszeit auf ein Peripheriegerät werden Beeinflussungen sowohl durch parallele Zugriffe auf den *gleichen* PCI-Bus, als auch durch parallele Zugriffe auf einen anderen PCI-Bus erzeugt. Im letzteren Fall sind die jeweiligen Messwerte in Tabelle 4.10 in Klammern angegeben. Zusätzlich sind in Abbildung 4.20 und in Abbildung 4.21 die Messwerte sowie die entsprechenden Geraden relevanter Einflüsse für Hauptspeicher- bzw. Peripheriezugriffe graphisch dargestellt.

Bei den Zugriffen innerhalb eines Knotens lässt sich festhalten, dass Speicherzugriffe der CPU auf ihren lokalen Hauptspeicher durch parallele Transfers, die ebenfalls auf diesen Speicher zugreifen, beeinflusst werden. Das Ausmaß der Beeinflussung hängt dabei von der Anzahl parallel stattfindender Transfers ab. Im schlimmsten Fall greifen lokale Peripheriegeräte und andere Knoten ebenfalls auf diesen Speicher zu. Sonstige Datentransfers — insbesondere der Zugriff anderer Knoten auf lokale Peripheriegeräte, sowie der Zugriff lokaler Peripheriegeräte auf das Verbindungsnetzwerk — haben dagegen keinen nennenswerten Einfluss auf Zugriffe auf den Hauptspeicherbereich des Knotens.

Bei Zugriffen auf Peripheriegeräte ergibt sich ein ähnliches Bild. Diese Zugriffe werden nur durch Zugriffe anderer Knoten auf diese Peripheriegeräte, sowie durch die Aktivität der Peripheriegeräte selbst beeinflusst. Dabei ist zu beachten, dass nur Beeinflussungen beim Zugriff auf den gleichen PCI-Bus zu beobachten sind; bei Zugriffen auf unterschiedliche PCI-

Busse sind dagegen keine Beeinflussungen messbar. Dies bedeutet, dass die Einflüsse der HT-Verbindungen zwischen den einzelnen Peripheriegeräten bzw. -bussen vernachlässigbar sind. Grund hierfür ist, dass HT eine hohe Datenübertragungsrate ermöglicht, die von den angeschlossenen Peripheriegeräten bei weitem nicht ausgeschöpft wird.[7)] Statt dessen sind die Einflüsse des Arbitrierungsverfahrens des PCI-Busses messbar; hierauf wird gesondert in Abschnitt 4.3.3 eingegangen.

Bestimmung der Latenzzeit beim Zugriff auf einen entfernten Knoten

Zugriffe auf einen entfernten Knoten können auf dessen Hauptspeicher oder auf dessen Peripheriegeräte erfolgen. Unter anderem hängt die Dauer des jeweiligen Zugriffs von der Entfernung des Knotens vom zugreifenden Knoten und von den Beeinflussungen parallel stattfindender Datentransfers ab.

Bei der Bestimmung der auftretenden Latenzzeiten beim Zugriff auf einen entfernten Knoten werden zunächst die jeweiligen Zugriffszeiten beim Zugriff auf einen benachbarten Knoten *ohne* paralleler Einflüsse anderer Transfers bestimmt. Die resultierenden Parameter sind in Tabelle 4.11 aufgeführt.

Zugriff	$t_U[\mu s]$	$m_U[\mu s]$
Hauptspeicherzugriffe	$0{,}33$[8)]	$0{,}16$
Peripheriezugriffe	$0{,}79$	$0{,}71$

Tabelle 4.11: Zugriffe auf einen entfernten Knoten — keine Beeinflussungen

Einflüsse im Quellknoten

Im lokalen Knoten können sich alle Datentransfers beim Zugriff auf einen anderen Knoten auswirken, die mit dem Hauptspeicher und mit den Peripheriegeräten des lokalen Knotens stattfinden. Insbesondere die Auswirkungen parallel ausgeführter Datenübertragungen des Knotens, die über das Verbindungsnetzwerk verlaufen, müssen hier untersucht werden. Die Zugriffe der CPU erfolgen während der Untersuchungen über Link 1 auf den benachbarten Knoten. Die durchgeführten Hauptspeicherzugriffe verändern hierbei Daten, so dass pro Zugriff ein Lese- und ein Schreibzugriff erfolgt. Die Ergebnisse der Messungen sind in Tabelle 4.12 aufgeführt.

Als Ergebnis lässt sich bei dieser Untersuchung festhalten, dass sich nur dann eine geringe Beeinflussung der Datenübertragung nachweisen lässt, falls parallele Datenübertragungen über den selben Link in das Verbindungsnetzwerk erfolgen. Dass die Einflüsse des Verbindungsnetzwerks auf maximale Zugriffszeiten gering sind, zeigt auch die im Folgenden durchgeführte Untersuchung des Verbindungsnetzwerks.

[7)] HT ermöglicht pro Link eine maximale Datenübertragungsrate von 3.2GByte/s, wohingegen die angeschlossen Peripheriegeräte nur eine maximale Datenrate von ca. 200MByte/s erzeugen können.

[8)] Dieser Wert wurde interpoliert; der tatsächlich gemessene Wert beträgt nur $0{,}27\mu s$. Grund hierfür ist, dass bei der verwendeten Architektur zwei Speicherkanäle vorhanden sind. Dies führt zu einem nicht-proportionalen Anstieg der gemessen Ausführungszeit bei der Messung von zwei Speicherzugriffen.

paralleler Zugriff	τ_l	μ_l
—	1,00	1,00
d_1	1,00	1,00
d_2	1,03	1,06
d_3	1,00	1,00
d_4	1,03	1,06
d_5	1,00	1,00
d_9	1,03	1,00
d_{10}	1,00	1,00

Tabelle 4.12: Zugriffe auf einen entfernten Knoten — Einflüsse im Quellknoten

Einflüsse im Verbindungsnetzwerk

Bei den Beeinflussungen durch das Verbindungsnetzwerk muss überprüft werden, auf welche Weise sich die *Entfernung* des Zielknotens vom zugreifenden Knoten auf die Zugriffszeit auswirkt. Zusätzlich müssen mögliche, parallel stattfindende Datentransfers im Netzwerk untersucht werden. Die resultierenden Beeinflussungen, die sich aufgrund der Länge des Pfades im Verbindungsnetzwerk ergeben, sowie der Einfluss möglicher parallel stattfindender Datentransfers, sind in Tabelle 4.13 aufgeführt.

Zugriff	τ_n	μ_n
Zugriff auf benachbarten Knoten	1,00	1,00
Zugriff auf nicht-benachbarten Knoten (ohne parallele Transfers)	1,06	1,25
Zugriff auf nicht-benachbarten Knoten (mit parallelen Transfers)	1,09	1,38

Tabelle 4.13: Zugriffe auf einen entfernten Knoten — Einflüsse des Verbindungsnetzwerks

Wie auch schon bei der Untersuchung der Beeinflussungen im lokalen Knoten zeigt sich, dass die Einflüsse des Verbindungsnetzwerks bei dieser Architektur gering sind. Maximale Zugriffszeiten werden hier insbesondere durch die jeweilige Entfernung des Zielknotens im Netzwerk bestimmt. Parallele Datenübertragungen — die beteiligen Prozessoren führen während der Messungen jeweils Hauptspeicherzugriffe (lesend/schreibend) durch — wirken sich kaum auf die maximale Zugriffszeit aus.

Einflüsse im Zielknoten

Im Zielknoten ist der Zugriff einigen Beeinflussungen unterworfen. Bei den Untersuchungen erfolgt der Zugriff auf den Zielknoten jeweils über Link 1. Bei den parallelen Zugriffen d_2 und d_4 wird von einem anderen Knoten ebenfalls über Link 1 auf den Zielknoten zugegriffen. Die resultierenden Parameter sind — für Hauptspeicherzugriffe und für Peripheriezugriffe getrennt — in Tabelle 4.14 dargestellt. Um vergleichbare Messwerte zu erhalten, erfolgen die Peripheriezugriffe auf die gleiche Portadresse. Parallel hierzu erfolgen Zugriffe auf den gleichen PCI-Bus, oder auf Geräte (bzw. von Geräten) die nicht an dem gleichen PCI-Bus angeschlossen sind. Für den letzteren Fall sind die resultierenden Messwerte in Klammern angegeben. Aus Gründen der

	Hauptspeicherzugriffe		Peripheriezugriffe	
paralleler Zugriff	τ_r	μ_r	τ_r	μ_r
—	1,00	1,00	1,00	1,00
d_0	1,09	1,25	1,01	1,01
d_1	1,06	1,13	1,92 (1,00)	1,28 (1,01)
d_2	1,03	1,38	1,00	1,00
d_3	1,06	1,25	1,00	1,01
d_0, d_3	1,52	1,50	1,00	1,01
d_0, d_1, d_3	1,79	1,75	2,09 (1,00)	1,29 (1,01)
d_4	1,00	1,00	1,30 (1,00)	1,28 (1,01)
d_5	1,00	1,00	1,05 (1,00)	1,28 (1,01)
d_6	1,03	1,00	1,01	1,01
d_7	1,03	1,00	1,01	1,01
d_8	1,03	1,00	1,65 (1,00)	1,20 (1,01)
d_5, d_8	1,00	1,00	1,84 (1,00)	1,93 (1,01)
d_1, d_5, d_8	1,09	1,13	3,84 (1,00)	3,34 (1,03)
d_9	1,00	1,00	1,92 (1,06)	1,28 (1,01)
d_{10}	1,00	1,00	1,91 (1,00)	1,29 (1,01)

Tabelle 4.14: Zugriffe auf einen entfernten Knoten — Einflüsse im Zielknoten

Übersichtlichkeit sind nicht sämtliche Kombinationen aufgeführt — statt dessen sind nur jene Szenarien aufgelistet, die für die in Kapitel 5 diskutierte Softwarearchitektur relevant sind.

Bei den Beeinflussungen im Zielknoten zeigt sich, dass Hauptspeicherzugriffe durch parallel stattfindende Zugriffe auf den Hauptspeicherbereich des Zielknotens beeinflusst werden. Zugriffe auf Peripheriegeräte werden nur dann beeinflusst, falls hier Datentransfers mit dem gleichen PCI-Bus stattfinden. Weiterhin erhöht sich mit zunehmender Anzahl parallel stattfindender Datentransfers die jeweilige Zugriffszeit.

Korrelation der einzelnen Teilbeeinflussungen

Im Folgenden soll untersucht werden, wie sich die Wechselbeziehungen zwischen den Beeinflussungen in Quell- und Zielknoten, sowie im Verbindungsnetzwerk verhalten. Es wird deshalb auf Knoten 2 des NUMA-Systems eine Realzeittask ausgeführt, die auf den Speicher von Knoten 1 zugreift. Parallel hierzu greifen die lokalen Peripheriegeräte von Knoten 1 ebenfalls auf diesen Speicher zu. Während der Zugriffe greift Knoten 3 auf den Speicher von Knoten 0 und Knoten 0 auf den Speicher von Knoten 3 zu.

Mit den ermittelten Werten aus den Tabellen 4.11, 4.12, 4.13 und 4.14 folgt in diesem Fall $t_U = 0{,}33\mu s$, $m_U = 0{,}16\mu s$, $\tau_l = 1{,}00$, $\mu_l = 1{,}00$, $\tau_n = 1{,}09$, $\mu_n = 1{,}38$, $\tau_r = 1{,}06$ und $\mu_r = 1{,}13$. Mit der Annahme $k_{lnr} = 1$ und $l_{lnr} = 1$ liefern die Gleichungen 4.11 und 4.12 die Abschätzung: $t_{WCAT,1} = 0{,}38\mu s$ und $m_{WCAT} = 0{,}25\mu s$. Werden die Parameter dieser Zugriffe gemessen, ergibt sich $t_{WCAT,1} = 0{,}36\mu s$ und $m_{WCAT} = 0{,}24\mu s$. Falls eine Überabschätzung nicht gewünscht ist, kann folglich $k_{lnr} = 0{,}95$ und $l_{lnr} = 0{,}96$ gesetzt werden.

Zusammenfassung der Messergebnisse

Bei der untersuchten NUMA-Architektur werden Latenzzeiten durch parallele Zugriffe auf gleiche Hauptspeicherbereiche und auf gleiche PCI-Busse verursacht. Dabei spielt es keine Rolle, ob die entsprechenden Zugriffe lokal durchgeführt werden, oder ob auf einen entfernten Knoten zugegriffen wird. Die Einflüsse des Verbindungsnetzwerks sind dabei bei dieser Architektur gering.

Hauptspeicherzugriffe auf den Speicherbereich eines Knotens werden durch alle Transfers beeinflusst, die ebenfalls auf diesen Speicherbereich zugreifen. Dabei bestimmt die Anzahl der parallel durchgeführten Datenübertragungen die jeweils maximal mögliche Zugriffszeit.

Zugriffe auf ein Peripheriegerät werden durch alle Datenübertragungen beeinflusst, die auf den gleichen PCI-Bus zugreifen, an den das untersuchte Peripheriegerät angeschlossen ist. Dabei verlängert sich die maximal mögliche Zugriffszeit mit der Anzahl paralleler Zugriffe.

4.3.2.6 Zusammenfassung

In diesem Abschnitt wird gezeigt, welche Beeinflussungen bei der Verwendung eines NUMA-Systems auftreten können. Wie auch schon bei der Untersuchung der SMP-Architektur werden auch bei NUMA-Systemen die relevanten Parameter zur Bestimmung maximaler Zugriffszeiten anhand von Messungen bestimmt.

Mit zunehmender Anzahl vorhandener Knoten des NUMA-Systems steigt auch die Anzahl der Datentransfers, die sich gegenseitig beeinflussen könnten. Da hier nicht alle Kombinationen untersucht werden können, wird beschrieben, wie die Anzahl der durchzuführenden Messungen verringert werden kann. Hierbei wird angenommen, dass sich in einem NUMA-System die einzelnen Komponenten äquivalent verhalten.

Die Untersuchungen werden beispielhaft für ein Quad-Opteron Multiprozessorsystem durchgeführt. Dabei zeigt sich, dass sich insbesondere Zugriffe auf gleiche Hauptspeicherbereiche und auf gleiche PCI-Busse beeinflussen. Die Einflüsse des Verbindungsnetzwerks sind dabei eher gering.

4.3.3 Der PCI-Bus als zentraler Peripheriebus

Der PCI-Bus ist in allen modernen Computersystemen anzutreffen und wird für die Anbindung von Peripheriegeräten an das Multiprozessorsystem benötigt. Bei manchen Chipsätzen verbindet ein PCI-Bus auch die einzelnen Bausteine des Chipsatzes. Insbesondere werden hier die North- und die South-Bridge über PCI miteinander verbunden.

Die eigentliche Bezeichnung des PCI-Busses ist PCI Local Bus, wobei PCI die Abkürzung für *Peripheral Component Interconnect* ist. Der PCI-Bus wird als lokaler Bus bezeichnet, weil er meist direkt über die Host-Bridge an den Host-Bus und somit an die Prozessoren angebunden

ist. Der PCI-Bus wurde 1992 entwickelt, um mit dem wachsenden Bandbreitenbedarf der Peripheriegeräte Schritt halten zu können. Er wechselte den bis dahin eingesetzten ISA-Bus ab. Mittlerweile ist der PCI-Bus der in PC-Systemen am häufigsten anzutreffende Peripheriebus.

Der PCI-Bus existiert in den Varianten PCI, PCI-66 und PCI-X. Diese PCI-Busse sind jeweils abwärtskompatibel, so dass beispielsweise ein PCI-Gerät, das an einem PCI-66-Bus funktioniert, auch an einem PCI-X-Bus betrieben werden kann. Zusätzlich existiert seit 2002 PCI-Express; dieses Bussystem ist jedoch physikalisch nicht kompatibel zu den drei Erstgenannten.

In den folgenden Abschnitten werden die Realzeiteigenschaften des PCI-Busses untersucht. Hierzu werden in Abschnitt 4.3.3.1 wichtige Eigenschaften des PCI-Busses erläutert. Anschließend werden in Abschnitt 4.3.3.2 die für Realzeitapplikationen relevanten Aspekte des PCI-Busses analysiert. In Abschnitt 4.3.3.3 werden maximale Zugriffszeiten auf Peripheriegeräte erörtert und quantitativ anhand von Messungen bestimmt. In Abschnitt 4.3.3.4 werden Neuerungen modernerer PCI-Bussysteme in Hinblick auf ihre Auswirkung auf Realzeitsysteme untersucht.

4.3.3.1 Eigenschaften des PCI-Busses

Im Folgenden sollen die wichtigsten Eigenschaften des PCI-Busses kurz erklärt werden. Insbesondere wird hier auf die für Realzeitsysteme relevanten Aspekte eingegangen. Zur Verdeutlichung ist in Abbildung 4.22 ein System mit zwei PCI-Bussen dargestellt, die über eine PCI-to-PCI Bridge miteinander verbunden sind.

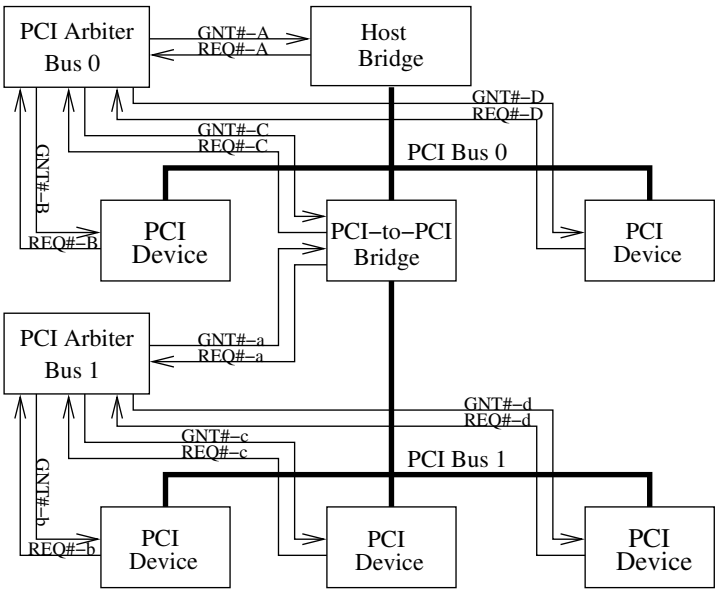


Abbildung 4.22: PCI-Busse

1. Funktionsweise

Der PCI-Bus ist ein Multi-Master Bus. Jedes Gerät kann Bus-Master (*Initiator*) werden und selbstständig einen Datentransfer zu einem anderen Gerät, dem *Target*, durchführen. Dies geschieht autonom, ohne die Unterstützung einer CPU. Damit es zu keinen Kollisionen kommt, muss ein Bus-Master vor jedem Zugriff den PCI-Bus erfolgreich arbitrieren. Anschließend darf er einen oder mehrere Zugriffe auf den Bus durchführen. Im letzteren Fall spricht man von einem *Burst-Transfer*. Lange Bursts minimieren den Adressierungs-Overhead und führen zu einem höheren Datendurchsatz.

Alle Zugriffe auf den PCI-Bus werden vom PCI-Bus-*Arbiter* koordiniert; die hierfür benötigte Logik ist in der Regel im Chipsatz integriert. Jedes Gerät besitzt zum Arbiter eine eigene Anforderungsleitung (*REQ#*) und eine eigene Zuteilungsleitung (*GNT#*). Der Arbiter entscheidet, welches Gerät als nächstes Zugriff auf den Bus bekommt und aktiviert dessen Zuteilungsleitung. Dies geschieht parallel zu dem jeweils vorhergehend ausgeführten Transfer — die Art dieser Arbitrierung wird deshalb auch als *Hidden Arbitration* bezeichnet. Der nächste Bus-Master darf erst dann aktiv werden, wenn der vorhergehende den Bus freigegeben hat.

Welche Strategie und welche Priorität der Arbiter bei der Busvergabe anwendet, ist vom Design des Chipsatzes abhängig. Die Spezifikation von PCI schreibt hier lediglich vor, dass das verwendete Arbitrierungsverfahren *fair* sein muss. Es darf hier also nie ein PCI-Gerät übervorteilt werden, so dass ein anderes seine Daten zu keinem Zeitpunkt übertragen kann.

Um den Durchsatzanforderungen gerecht zu werden, besitzt jedes Gerät, das mehr als zwei Datenphasen pro Übertragung ausführen kann, einen konfigurierbaren *Latency Timer*, der festlegt, wieviele Datenphasen ein PCI-Gerät *mindestens* am Stück durchführen darf. Ein Bus-Master, dessen Zuteilungsleitung deaktiviert wurde und der noch nicht die durch den Latency Timer spezifizierte Anzahl an Datenphasen durchgeführt hat, darf bis zu deren Erreichen weitere Datenphasen durchführen. Falls das Register des Latency Timers mit Null belegt wird, muss der Master nach Vollendung der ersten Datenphase den Bus freigeben.

Aus Sicht des PCI-Busses ist die Bridge, über die der PCI-Bus an das Computersystem angebunden ist, auch ein PCI-Gerät. Deshalb muss auch diese vor einem Zugriff den PCI-Bus erfolgreich arbitrieren, um Bus-Master zu werden. Diese Bridge arbeitet auch als Target für die an den PCI-Bus angeschlossenen Peripheriegeräte und erlaubt diesen den Zugriff auf den Hauptspeicher und auf andere Peripheriebusse.

2. PCI-Zugriffe

Der PCI-Bus arbeitet mit kombinierten Adress- und Datenleitungen und zusätzlichen Kommando- bzw. Byteauswahlleitungen. Ein PCI-Zugriff besteht aus einer bzw. zwei Adressphasen und einer oder mehreren Datenphasen. In der Adressphase legt der Master die gewünschte Zieladresse und die beabsichtigte Zugriffsart auf den Bus. Alle darauffolgenden Phasen sind Datenphasen; dabei zählt nach jeder Datenphase das Target die Adresse selbstständig hoch.

Die wichtigsten Zugriffsarten sollen im Folgenden kurz vorgestellt werden:

Speicherzugriffe: Ein Gerät greift auf den, auf einem anderen Gerät enthaltenen Speicher oder auf den Hauptspeicher zu. Beispielsweise greift ein TV-Adapter auf den Speicher der Grafikkarte zu.

I/O-Zugriffe: Hiermit wird auf einzelne Kommando- oder Statusregister der Geräte zugegriffen. Diese Zugriffe werden von der Bridge erzeugt, die den jeweiligen PCI-Bus an das Rechensystem anbindet. Im Allgemeinen handelt es sich hierbei um die Host-Bridge. I/O-Zugriffe können nicht gepuffert werden.

Konfigurationszugriffe: Die von einem PCI-Gerät belegten Ressourcen, wie Speicherbereiche, I/O-Bereiche und Interruptvektoren, werden in den Konfigurationsregistern der jeweiligen PCI-Geräte festgelegt.

Speicherzugriffe einer CPU auf den Adressraum eines PCI-Busses und die entsprechenden I/O-Zugriffe werden von der Host-Bridge automatisch in die entsprechenden PCI-Zugriffe umgesetzt.

Ein Zugriff auf den PCI-Bus beginnt mit der Aktivierung des Signals *FRAME#* (Cycle Frame). Dieses Signal bleibt bis zum Erreichen der letzten Datenphase gesetzt. Zur Datenflusssteuerung dienen die Signale *IRDY#* (Initiator Ready) und *TRDY#* (Target Ready). *IRDY#* zeigt an, dass der Master gültige Daten über den PCI-Bus schickt bzw. bereit ist, Daten zu empfangen. *TRDY#* ist das entsprechende Äquivalent für das Target. Falls eines der Signale *IRDY#* oder *TRDY#* inaktiv ist, werden Wartephasen eingelegt.

Eine laufende Übertragung kann nur durch die am Transfer beteiligten Geräte beendet werden. Insbesondere ist eine Beendigung nur durch den Master oder durch das Target möglich. Eine Ausnahme bildet der Neustart des Systems, der durch *RESET#* signalisiert wird.

Die Gründe für die Beendigung einer Übertragung durch den Master sind wie folgt:

- Es wurden alle Daten übertragen.
- Der Master hat die Bus-Zuteilung durch den Arbiter (Deaktivierung von *GNT#*) verloren und die durch den *Latency Timer* festgelegte Zeitspanne der minimalen Busbelegung überschritten.
- Auf eine Adressphase hat kein Target geantwortet.

Das Target beendet eine Übertragung durch Aktivierung des *STOP#*-Signals. Gründe für die Beendigung einer Übertragung sind:

- Das Target ist zu langsam, um weitere Daten zu empfangen.
- Das Target kann seine Adresse nicht selbstständig hochzählen.
- Das Target unterstützt den gewünschten Zugriffstyp nicht.
- Ein Speicherbereich ist durch einen anderen Master gesperrt.

Mittels der Leitungen *TRDY#* und *DEVSEL#* teilt das Target dem Master den Grund für das Übertragungsende mit. Unterschieden werden *Disconnect A/B/C* und *Retry*. Ein Retry zwingt

den Master, den Transfer erneut zu versuchen. Genaueres kann hierzu in [45] nachgelesen werden.

3. PCI-to-PCI Bridges

Eine PCI-to-PCI Bridge bietet die Möglichkeit, zwei PCI-Busse miteinander zu verbinden. Die Aufgabe der Bridge besteht darin, den Datenverkehr zwischen diesen beiden Bussen korrekt zu übertragen. Hierzu überwacht die Bridge beide PCI-Busse und überträgt in Abhängigkeit des angesprochenen Targets ggf. die entsprechende Transaktion auf den anderen Bus. Falls die Bridge feststellt, dass ein Zugriff auf ein Gerät *hinter* der Bridge erfolgen soll, dann fungiert die Bridge als Target auf dem Bus, an dem auch der Initiator angeschlossen ist. Auf dem anderen PCI-Bus arbeitet die Bridge als Initiator. Sowohl für den Initiator, als auch für das Target ist das Vorhandensein der Bridge transparent.

4. Master Enable Bit

Jedes PCI-Gerät, insbesondere auch eine PCI-Bridge, beinhaltet im PCI Command Register (Offset 0x04) das so genannte *Master Enable Bit*. Ist dieses Bit gesetzt, dann darf das jeweilige Gerät Bus-Master werden und somit Datentransfers durchführen. Ist es gelöscht, dann kann dieses Gerät zwar als Target für Datentransfers dienen; selbst darf es jedoch keine Datenübertragung anstoßen.

4.3.3.2 Realzeitaspekte

Die Auswirkung paralleler DMA-Transfers bei Zugriffen auf den Hauptspeicher wurde bereits in den vorhergehenden Abschnitten für SMP- und für NUMA-Systeme diskutiert. Dabei zeigte sich, dass parallele Speichertransfers der Peripheriegeräte einen nicht zu vernachlässigenden Einfluss auf die Zugriffszeit von Realzeitsoftware haben.

Hier soll nun untersucht werden, welche Latenzzeiten entstehen, falls ein PCI-Gerät auf den Bus zugreifen möchte. Da insbesondere die Bridge, über die der PCI-Bus an das Rechensystem angebunden ist — im Allgemeinen ist dies die Host-Bridge — auch ein PCI-Gerät ist, lassen sich aus diesen Überlegungen die maximalen Zugriffszeiten einer CPU auf ein Gerät des jeweiligen PCI-Busses ableiten.

Die Latenzzeit bis zur Beendigung des ersten Datentransfers beim Zugriff eines Bus-Masters auf den PCI-Bus besteht dabei aus den folgenden Teil-Latenzzeiten, vgl. hierzu auch [45]:

Arbitration Latency: Dies ist die Zeitspanne, die zwischen dem Anlegen des Signals *REQ#* und dem Erhalt des Signals *GNT#* durch den Arbiter verstreicht. Diese Zeitspanne hängt von dem verwendeten Arbitrierungsverfahren und der Aktivität anderer PCI-Geräte ab.

Bus Acquisition Latency: Dies ist die Zeitspanne zwischen dem Erhalt des Signals *GNT#* durch den Arbiter und dem Zeitpunkt, an dem der Initiator tatsächlich den PCI-Bus belegt. Diese Zeitspanne hängt davon ab, wie lange der aktive Transfer noch andauert. Dies

wiederum hängt vom Wert des Latency Timers und von der maximal zugestandenen Zeit zur Beendigung der ersten Datenphase ab. Diese Zeitspanne beträgt 16 Takte.

Target Latency: Dies ist die Zeitspanne, die das Target benötigt, bis es auf die Anforderung reagiert. Diese Zeitspanne ist auf 16 Takte begrenzt.

Während eines Burst-Transfers darf der Initiator für maximal acht PCI-Takte *IRDY#* während einer Datenphase löschen. Selbiges gilt auch für das Target: Falls es nicht innerhalb von acht Takten das nächste Datum senden bzw. verarbeiten kann, muss es ein *Disconnect* an den Initiator senden.

Soll ein PCI-Bus in einem Realzeitsystem eingesetzt werden, dann müssen die im Folgenden aufgeführten Eigenschaften dieser Busarchitektur berücksichtigt werden:

- Die PCI-Spezifikation schreibt lediglich vor, dass das verwendete Arbitrierungsverfahren fair sein muss. Das tatsächlich verwendete Verfahren hängt deshalb vom Hersteller des eingesetzten Chipsatzes ab — die *Arbitration Latency* kann nicht allgemein angegeben werden.
- Die *Target Latency* hängt von der Architektur des Targets ab.
- Falls ein Bus-Master den Zugriff auf den Bus erlangt hat, kann die tatsächlich benötigte Zeit bis zur Vollendung eines Zugriffs ebenfalls nicht angegeben werden. Die maximale Zeit, bis ein PCI-Gerät auf eine Anforderung reagieren muss, beträgt zwar 16 Takte — falls das Gerät jedoch nicht bereit ist, sendet es ein *Retry* an den Initiator. Dieser versucht dann zu einem späteren Zeitpunkt erneut, den Transfer durchzuführen.
- Während einer Datenphase darf sowohl der Initiator, als auch das Target bis zu acht Wartezyklen einfügen. Dieser Umstand muss bei worst-case Betrachtungen berücksichtigt werden.
- Mit Hilfe des *Latency Timers* lässt sich bei vielen Bus-Mastern die maximale Länge eines Burst-Transfers individuell einstellen. Die *Bus Acquisition Latency* ist unmittelbar von diesem Wert abhängig.
- Dem Target ist es erlaubt, einen Burst-Transfer abzubrechen, falls es die Daten nicht schritthaltend bereitstellen bzw. verarbeiten kann. Dann wird der Transfer zu einem späteren Zeitpunkt fortgesetzt. Wann dies der Fall sein wird, hängt wiederum vom Arbitrierungsverfahren und der Aktivität der anderen Busteilnehmer ab.
- Durch Löschen des *Master Enable Bits* kann ein PCI-Gerät zur Laufzeit umkonfiguriert werden, so dass es keine weiteren Datentransfers anstößt. Wird dieses Bit wieder gesetzt, darf dieses Gerät wieder Bus-Master werden und Transfers durchführen. Wird das *Master Enable Bit* einer Bridge gelöscht, kann kein PCI-Gerät hinter dieser Bridge Datenübertragungen initiieren.

Das Laufzeitverhalten eines PCI-Busses hängt maßgeblich von den Eigenschaften der angeschlossenen Geräte ab. Insbesondere wird die *Arbitration Latency* durch Anzahl und Aktivität der am PCI-Bus angeschlossenen Bus-Master bestimmt. Die Zeit, bis ein Transfer mit einem

Target abgeschlossen werden kann, hängt vom Hardwaredesign des Targets ab und ist hier individuell verschieden. Da es insbesondere dem Target erlaubt ist, Transfers abzubrechen, können keine allgemeingültigen Zugriffszeiten angegeben werden.

Deshalb werden auch beim Zugriff auf PCI-Geräte die jeweils benötigten Zugriffszeiten anhand von Messungen bestimmt. Wie diese Messungen durchgeführt werden müssen und was hierbei beachtet werden muss, wird im folgenden Abschnitt beschrieben.

4.3.3.3 Bestimmung der Zugriffszeiten auf PCI-Geräte

Bei der Bestimmung der Zugriffszeiten auf PCI-Geräte wird prinzipiell von den selben Annahmen ausgegangen, die auch schon zur Bestimmung der Zugriffszeiten in SMP- und NUMA-Systemen gemacht wurden (vergleiche hierzu auch Seite 39 und Seite 60). Insbesondere wird davon ausgegangen, dass hier maximale Zugriffszeiten in Abhängigkeit von der Anzahl der durchgeführten Zugriffe ebenfalls ein *lineares* Verhalten aufweisen. Zur Beschreibung der maximalen Zugriffszeit auf ein bestimmtes Peripheriegerät wird deshalb ebenfalls Gleichung 4.8 verwendet:

- Die Summe aus Arbitration Latency, Bus Acquisition Latency und Target Latency wird über die Messung der maximalen Zeitspanne bis zur Vollendung des ersten Zugriffs auf das jeweilige Peripheriegerät bestimmt.
- Die Geschwindigkeit, mit der das Peripheriegerät die jeweiligen Daten liefern bzw. verarbeiten kann, wird über die maximal benötigte Zeitspanne zur Verarbeitung mehrerer Transfers ermittelt. Hierbei macht es aus Sicht von Realzeitsoftware keinen Unterschied, ob der Bus während der Übertragung einem neuen Bus-Master zugeteilt wird, oder ob das Target aufgrund des Überlaufs interner Pufferspeicher keine weiteren Daten verarbeiten kann. Um zu vermeiden, dass man hier die maximal benötigte Zugriffszeit zur Durchführung mehrerer Transfers zu positiv abschätzt, muss die Anzahl der jeweils durchgeführten Zugriffe variiert werden.

Da die Beeinflussungen beim Zugriff auf einen PCI-Bus nur von der Last der am Bus angeschlossenen Geräte abhängen, lassen sich für jedes Lastszenario die Parameter τ_{PCI} und μ_{PCI} definieren, die die Verlängerung der maximalen Zugriffszeiten beschreiben. τ_{PCI} reflektiert dabei die Verlängerung des ersten Zugriffs, falls eine bestimmte Last auf dem PCI-Bus vorhanden ist, und μ_{PCI} beschreibt das Verhältnis des Anstiegs der resultierenden Geraden. Somit lassen sich die Gleichungen 4.8 bzw. 4.10 um die entsprechenden Faktoren erweitern:

$$t_{WCAT}(x) = t_{WCAT,1} + m_{WCAT}(x-1); \qquad x \in \mathbb{N}$$

mit

$$t_{WCAT,1} = t_{CS}\tau_{PCI}k_{CS,PCI} \tag{4.18}$$

$$m_{WCAT} = m_{CS}\mu_{PCI}l_{CS,PCI} \tag{4.19}$$

Hierbei beschreiben t_{CS} und m_{CS} die Beeinflussungen durch den Chipsatz, die in den Abschnitten 4.3.1 und 4.3.2 behandelt wurden. Die Parameter $k_{CS,PCI}$ und $l_{CS,PCI}$ beschreiben

die Korrelation zwischen dem Chipsatz und dem PCI-Bus. τ_{PCI} und μ_{PCI} lassen sich aus dem Vergleich der Messung ohne paralleler PCI-Aktivität ($t_{WCAT,1,ohne}$, $m_{WCAT,ohne}$) und der Messung mit der zu untersuchenden Belastung ($t_{WCAT,1,mit}$, $m_{WCAT,mit}$) auf dem PCI-Bus ermitteln. Dabei darf sich nur die Last auf dem PCI-Bus ändern, sonstige Beeinflussungen dürfen nicht vorhanden sein. Die zu erzeugende PCI-Last darf sich auch deshalb nicht mit sonstigen Datenpfaden überlagern, die sich zusätzlich auf das Messergebnis auswirken könnten.

$$\tau_{PCI} = t_{WCAT,1,mit}/t_{WCAT,1,ohne} \quad (4.20)$$

$$\mu_{PCI} = m_{WCAT,mit}/m_{WCAT,ohne} \quad (4.21)$$

Sollen die maximalen Zugriffszeiten auf einen PCI-Bus anhand von Messungen untersucht werden, dann müssen sich alle angeschlossenen Geräte so verhalten, dass sie die Zugriffszeiten jeweils um das maximal mögliche Maß verlängern. Hierbei muss sichergestellt werden, dass sich das jeweilige Gerät so verhält, wie es sich im schlimmsten Fall im Realzeitbetrieb auch verhalten kann.

Beim Zugriff auf die verschiedenen Geräte ist grundsätzlich mit unterschiedlichen Zugriffszeiten zu rechnen: Neben den Beeinflussungen durch andere PCI-Geräte hängen diese Zeiten davon ab, wie schnell das jeweilige Gerät die Datentransfers verarbeiten kann. Prinzipiell muss bei der Untersuchung der Zugriffszeiten auf den PCI-Bus von folgenden Annahmen ausgegangen werden:

- Es können nur die Zugriffszeiten von einer CPU auf ein PCI-Gerät gemessen werden. Diese Zugriffe erfolgen hierbei über die Bridge, die den jeweiligen PCI-Bus an das Rechensystem anbindet. Die Beeinflussungen, die durch den Chipsatz hervorgerufen werden, wurden bereits in den Abschnitten 4.3.1 und 4.3.2 diskutiert.
- Die maximale Zeit, bis eine Bridge Zugriff auf den hinter ihr liegenden PCI-Bus bekommt, ist unabhängig von dem adressierten Gerät. Allerdings kann dieses Gerät durch Senden eines oder mehrerer *Retrys* die gemessene maximale Zugriffszeit verlängern.
- Liegt ein Peripheriegerät hinter weiteren PCI-to-PCI Bridges, dann vervielfachen sich die Latenzzeiten, da hier nun mehrere PCI-Busse arbitriert werden müssen.
- Neben den durch den PCI-Bus verursachten Latenzzeiten hängen die gemessenen Zugriffszeiten davon ab, wie schnell das angesprochene Peripheriegerät die jeweiligen Daten bzw. Befehle verarbeiten kann. Insbesondere muss davon ausgegangen werden, dass Zugriffe auf unterschiedliche Adressen eines Peripheriegeräts unterschiedlich lange dauern. Beim Zugriff auf gleiche Adressen des Peripheriegeräts hängt die maximale Zugriffszeit von der Architektur des Peripheriegeräts und von der jeweils mit dem Zugriff ausgelösten Aktion ab. Diese Thematik wird nochmals in Abschnitt 4.3.4 aufgegriffen.

Um die durch den PCI-Bus verursachten Latenzzeiten bestimmen zu können, müssen Zugriffe auf eines der am PCI-Bus angeschlossenen Peripheriegeräte durchgeführt werden. Um vergleichbare Messergebnisse zu erhalten, muss hier jeweils die *gleiche* Aktion des Peripheriegeräts mit dem Zugriff ausgelöst werden. Gleichzeitig sollte die Latenzzeit, die mit dem jeweiligen

Zugriff vom Peripheriegerät selbst verursacht wird, konstant sein. Um ausschließlich die Einwirkungen des PCI-Busses zu messen, dürfen keine Zugriffe von anderen Prozessoren oder von nicht betrachteten Peripheriegeräten erfolgen.

Einfluss paralleler PCI-Transfers auf Zugriffe auf ein PCI-Gerät

In diesem Beispiel wird ein Szenario betrachtet, bei dem an einem PCI-Bus drei PCI-Geräte angeschlossen sind: Eine Netzwerkkarte, ein SCSI-Controller, sowie ein IDE-Controller. Auf den IDE-Controller wird ausschließlich von einer Realzeittask zugegriffen. Parallel zu den Zugriffen können sowohl die Netzwerkkarte, als auch der SCSI-Controller, Datentransfers durchführen. Die Realzeittask führt ausschließlich Festplatten-Schreibzugriffe im PIO-Modus durch. Im Folgenden sollen die maximalen Zugriffszeiten auf die Register *DATA* und *CMD* untersucht werden. *DATA* wird für die Übertragung der Daten auf die Festplatte benötigt; *CMD* definiert den Befehl für die Festplatte. Als Festplattenkommando wird hierbei der Befehl *No-Operation (NOP)* gewählt — für andere Befehle ergeben sich ggf. unterschiedliche Zugriffszeiten.

Beim Zugriff auf *CMD* wird immer nur ein Zugriff vom verwendeten Gerätetreiber durchgeführt. Deshalb wird zur Untersuchung der maximalen Zugriffszeit nur die maximale Zeit zur Durchführung eines Zugriffs betrachtet. Auf *DATA* werden mehrere aufeinanderfolgende Zugriffe vom Prozessor ausgeführt. Um hier konstante Latenzzeiten der Festplatte selbst zu erhalten, wird jeweils der gleiche Block auf einer äußeren Spur der Platte beschrieben.

Die Messergebnisse für die maximale Zugriffszeit beim Zugriff auf die Register *DATA* und *CMD* sind in Tabelle 4.15 aufgeführt.

	DATA		*CMD*		
paralleler Zugriff	$t_{WCAT,1}[\mu s]$	$m_{WCAT}[\mu s]$	$t_{WCAT,1}[\mu s]$	τ_{PCI}	μ_{PCI}
keine parallelen Zugriffe	0,70	0,52	0,73	1,00	1,00
Netzwerk	2,65	1,06	2,69	3,79	2,04
SCSI	2,91	2,30	2,87	4,51	4,42
SCSI, Netzwerk	7,97	3,22	8,20	11,38	6,44

Tabelle 4.15: Zugriffe auf einen PCI-IDE-Controller

In Abhängigkeit von der Aktivität des SCSI-Controllers und der Netzwerkkarte kommt es zu erheblichen Verlängerungen der jeweils gemessenen maximalen Zugriffszeit. Aufgrund des höheren Datendurchsatzes und der damit verbundenen höheren Anforderung an den PCI-Bus, wirkt sich der Einfluss von SCSI im Vergleich zum Einfluss des Netzwerks stärker aus.

Einfluss der Latency Timer

Es soll nun untersucht werden, in welchem Ausmaß sich eine Modifikation der Werte der Latency Timer der einzelnen Bus-Master auf maximale Zugriffszeiten auf ein PCI-Gerät auswirkt. Dabei wird von einer Realzeittask auf einen IDE-Controller zugegriffen. Parallel hierzu werden Datentransfers von weiteren PCI-Geräten initiiert. Hierbei wird von einem weiteren IDE-Controller, der an dem gleichen PCI-Bus angeschlossen ist, Last erzeugt. Zusätzlich werden

Konfiguration der Latency Timer	τ_{PCI}	μ_{PCI}
Standardeinstellung	1,76	1,62
alle Null (keine Bursts)	1,78	1,62
nur Bursts für Host-Bridge	1,72	1,59

Tabelle 4.16: Einfluss der Latency Timer auf Zugriffszeiten

von einem dritten IDE-Controller und einer Netzwerkkarte, die über eine PCI-to-PCI Bridge angebunden ist, DMA-Transfers initiiert.

Beim Zugriff auf den IDE-Controller ohne parallele Beeinflussungen wird $t_{WCAT,1,ohne} = 0{,}46\mu s$ und $m_{WCAT,ohne} = 0{,}39\mu s$ bestimmt. Mit paralleler Last und konfigurierten Standardwerten der Latency Timer resultiert $t_{WCAT,1,mit} = 0{,}81\mu s$ und $m_{WCAT,mit} = 0{,}63\mu s$. In der folgenden Messung werden alle Latency Timer der PCI-Geräte auf Null gesetzt. Damit wird die Durchführung von Burst-Transfers verhindert. Hier resultiert als Ergebnis $t_{WCAT,1,mit} = 0{,}82\mu s$ und $m_{WCAT,mit} = 0{,}63\mu s$. In der nächsten Messung wird der Latency Timer der Host-Bridge auf den maximalen Wert gesetzt; alle anderen Latency Timer sind mit Null belegt. In diesem Fall resultiert $t_{WCAT,1,mit} = 0{,}79\mu s$ und $m_{WCAT,mit} = 0{,}62\mu s$.

Die Ergebnisse dieser Messreihe sind in Tabelle 4.16 aufgeführt. Dabei zeigt sich, dass die Latency Timer zwar einen Einfluss auf maximale Zugriffszeiten haben. Im Worst-Case ergeben sich allerdings nur geringe Unterschiede in der Zugriffszeit. Grund hierfür ist, dass durch Modifikation der Latency Timer die Bus Acquisition Latency zwar verringert wird, die Arbitration Latency allerdings nach wie vor vom verwendeten Arbitrierungsverfahren abhängt. Für den Realzeiteinsatz wird empfohlen, den Latency Timer der Host-Bridge zu erhöhen und gleichzeitig die Latency Timer der restlichen Peripheriegeräte zu verringern. Hieraus resultiert eine geringe Reduzierung der WCAT.

Korrelation zwischen Chipsatz und PCI-Bus

Im folgenden Beispiel wird untersucht, wie sich Zugriffe auf einen PCI-Bus bei paralleler Last durch PCI-Geräte und einer anderen CPU verhalten. Die Messungen werden dabei auf einem Dual-Athlon Multiprozessorsystem durchgeführt. Dabei wird auf die Register eines IDE-Controllers zugegriffen; parallel hierzu wird Last von einem weiteren IDE-Controller und einer Netzwerkkarte erzeugt, die über eine PCI-to-PCI Bridge angeschlossen sind. Die Einflüsse des Chipsatzes sind dabei wie folgt:

Lastsituation	$t_{CS}\ [\mu s]$	$m_{CS}\ [\mu s]$
keine parallele Last	0,69	0,63
parallele CPU-Zugriffe	1,39	1,19

Tabelle 4.17: Einflüsse des Chipsatzes

Zusätzlich wird $\tau_{PCI} = 2{,}02$ und $\mu_{PCI} = 1{,}52$ durch Messung ermittelt. Mit den Gleichungen 4.18 und 4.19 und $k_{CS,PCI} = 1$ und $l_{CS,PCI} = 1$ folgt hieraus zunächst $t_{WCAT,1} =$

$2{,}81\mu s$ und $m_{WCAT} = 1{,}81\mu s$. Werden dagegen diese Zugriffszeiten gemessen, ergibt sich $t_{WCAT,1} = 2{,}97\mu s$ und $m_{WCAT} = 2{,}31\mu s$. Um die WCAT bei parallelen Zugriffen der anderen CPU bei gleichzeitiger Last von PCI-Geräten korrekt abzuschätzen, muss folglich für dieses System $k_{CS,PCI} = 1{,}05$ und $l_{CS,PCI} = 1{,}28$ gesetzt werden.

4.3.3.4 Weiterentwicklungen: PCI-X und PCI Express

Im Folgenden soll kurz auf modernere Verbindungsprotokolle eingegangen werden, die zumindest namentlich mit PCI verwandt sind: PCI-X und PCI-Express.

1. PCI-X

PCI-X ist eine Weiterentwicklung des PCI-Busses, die durch geeignete Maßnahmen versucht, die maximal möglichen, sowie die effektiven Übertragungsraten des PCI-Busses zu steigern. PCI-X ist hierbei rückwärtskompatibel zu PCI. Insbesondere können zu PCI-X kompatible Geräte sowohl mit dem PCI, als auch mit dem PCI-X-Protokoll umgehen. Wird eine Steckkarte in einen PCI-X-Steckplatz gesetzt, die nur das PCI-Protokoll beherrscht, dann arbeitet der gesamte Bus im PCI-Modus.

Im Gegensatz zu PCI, das nur eine maximale Taktfrequenz von 66 MHz erlaubt, ist PCI-X bis 133 MHz spezifiziert. Die Breite des Busses beträgt 64 Bit. PCI existiert als 32 Bit und als 64 Bit Version. Das PCI-X-Protokoll versucht durch geeignete Maßnahmen, die *durchschnittliche* Busauslastung zu erhöhen. Diese Maßnahmen sind in [44] beschrieben:

- Während einer Datenübertragung ist es sowohl dem Master, als auch dem Target nicht erlaubt, Wartezyklen einzufügen. Einzige Ausnahme ist hier der erste Datenzyklus.
- In PCI-X werden die Daten in Form von Blöcken mit einer Größe von 128 Byte (oder weniger) übertragen. Nur an den Grenzen eines Blocks darf der Master oder das Target die Übertragung abbrechen. Voraussetzung hierfür ist, dass das Target Burst-Transfers unterstützt.
- Ein Master darf nur dann den PCI-X-Bus arbitrieren, wenn er auch sofort mit der Datenübertragung nach Erhalt des Busses beginnen kann.
- Im Gegensatz zu PCI wird bei PCI-X dem Target zu Beginn einer Datenübertragung mitgeteilt, wieviele Daten übertragen werden sollen.
- Mit Hilfe von *Split Transactions* wird der Datendurchsatz auf dem PCI-X-Bus weiter erhöht. Falls hier ein Target nicht innerhalb von 16 Takten die Daten übertragen kann, wird bei PCI ein Retry an den Initiator geschickt. Dieser versucht dann zu einem späteren Zeitpunkt erneut, den Transfer durchzuführen. Im Gegensatz hierzu arbitriert bei PCI-X das ursprüngliche Target (der *Completer*) den Bus, sobald es die Daten liefern bzw. verarbeiten kann und adressiert den ursprünglichen Initiator (den *Requester*). Anschließend werden die jeweiligen Daten übertragen.

Zusätzlich bietet PCI-X noch einige weitere Verbesserungen, auf die nicht weiter eingegangen werden soll. Für weitere Informationen sei auf [44] verwiesen.

Für die Bestimmung maximaler Zugriffszeiten folgt bei der Verwendung von PCI-X keine Änderung der Vorgehensweise im Vergleich zu PCI. Prinzipiell ist hier zwar mit niedrigeren maximalen Zugriffszeiten zu rechnen; trotzdem müssen aufgrund des gemeinsam verwendeten Busses die Auswirkungen aller Busteilnehmer untersucht werden.

2. PCI-Express

Die Architektur von PCI-Express ist ähnlich zu dem in Abschnitt 4.3.2.4 vorgestellten HyperTransport. PCI-Express existiert seit 2002 und soll nach dem Willen der Firma Intel sowohl für Chip-zu-Chip Verbindungen, als auch für Board-zu-Board Verbindungen verwendet werden. Im Gegensatz zu HyperTransport ist PCI-Express nicht für die Verbindungen mehrerer CPUs geeignet — eine NUMA-Architektur lässt sich folglich mit PCI-Express nicht realisieren.

PCI-Express verwendet für die Datenübertragung serielle Punkt-zu-Punkt Verbindungen, wobei hier — wie auch bei HyperTransport — für jede Übertragungsrichtung ein eigener Simplexkanal vorhanden ist. Die Breite jedes Kanals besteht hierbei aus bis zu 32 Signalleitungen. Die Daten selbst werden bei PCI-Express mit Hilfe von Paketen übertragen. PCI-Express unterstützt neben “normalen” Datenübertragungen ebenfalls isochronen Datenverkehr.

Abbildung 4.23 gibt einen Überblick über die Architektur von PCI-Express: Der *Root Complex* entspricht der Host-Bridge. Hier werden die Prozessoren, der Hauptspeicher und die Peripherie angebunden. AGP wird nicht mehr verwendet; statt dessen wird die Grafikkarte über einen PCI-Express Link mit hoher Übertragungsrate und der damit verbundenen hohen Anzahl an Signalleitungen, meist x16 PCI-Express, angebunden. Die restlichen Peripheriegeräte werden über einzelne Punkt-zu-Punkt Verbindungen über einen oder mehrere PCI-Express *Switches* angeschlossen.

Falls bei PCI-Express mehrere Geräte auf eine gemeinsam verwendete Ressource zugreifen möchten, werden Arbitrierungsverfahren eingesetzt. Hierbei handelt es sich um gewichtete Round-Robin-Arbitrierungsverfahren. Zusätzlich ist es möglich, für isochronen Datenverkehr Prioritäten anzugeben.

Bei der Bestimmung der Realzeiteigenschaften eines PCI-Express Netzwerks anhand von Messungen müssen ebenfalls alle möglichen Datenpfade dieses Netzwerks untersucht werden. Prinzipiell kann auch bei PCI-Express davon ausgegangen werden, dass sich auf einen Zugriff alle parallel stattfindenden Datentransfers auswirken, die Teile des für den Zugriff benötigten Datenpfads benötigen. Insbesondere muss bei PCI-Express der Einfluss der vorhandenen Switches berücksichtigt werden. Bei der Untersuchung der maximalen Zugriffszeiten ist dabei folgendes zu beachten:

- Jedes Peripheriegerät, das auf einen gemeinsam verwendeten Datenpfad des PCI-Express Netzwerks zugreift, muss Daten mit der jeweils maximal möglichen Rate senden und empfangen.

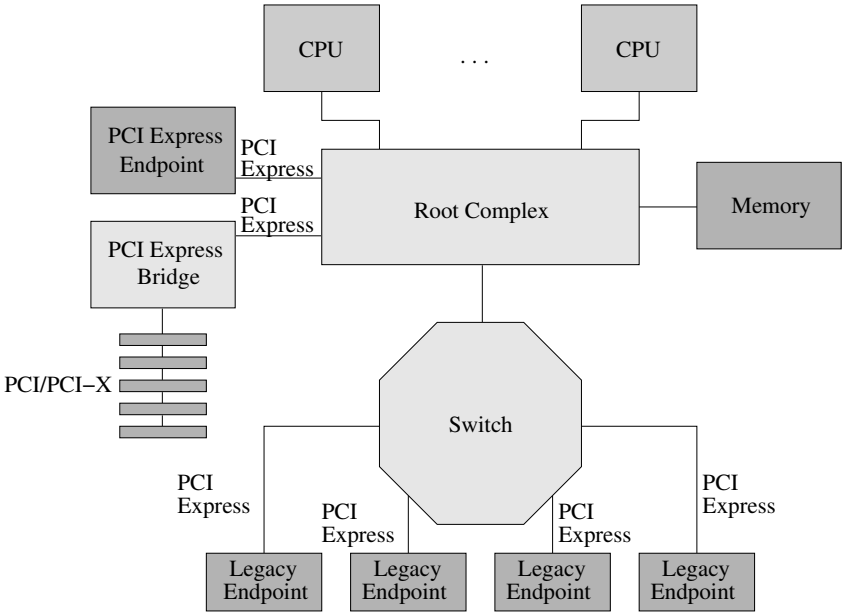


Abbildung 4.23: PCI-Express

- Falls einige Peripheriegeräte während des Realzeitbetriebs isochrone Datentransfers ausführen, muss diese parallele Last ebenfalls während der Bestimmung der Zugriffszeiten vorhanden sein.

Aufgrund der bei PCI-Express verwendeten Round-Robin-Arbitrierungsverfahren, ist ebenfalls mit einem linearen Verhalten der jeweiligen Zugriffszeit in Abhängigkeit von der Anzahl der Zugriffe zu rechnen. Folglich können auch bei PCI-Express die Parameter $t_{WCAT,1}$ und m_{WCAT} für bestimmte Zugriffe unter definierter paralleler Last angegeben werden.

4.3.4 Weitere Peripheriebusse und Peripheriegeräte

In den vorhergehenden Abschnitten wurde der Einfluss des Chipsatzes und des PCI-Busses auf die Ausführungszeit von Realzeitsoftware erörtert. Soll von einer Realzeittask auf ein Peripheriegerät zugegriffen werden, hängt die Zugriffszeit davon ab, ob das jeweilige Peripheriegerät direkt an den Chipsatz — beispielsweise über HyperTransport — angebunden ist, oder ob dieses Peripheriegerät über einen PCI-Bus an das Rechensystem angeschlossen ist. Zusätzlich existiert noch die Möglichkeit, dass das Peripheriegerät über weitere Bussysteme, wie beispielsweise IDE/ATA, SCSI, USB oder FireWire angeschlossen ist.

Generell kann davon ausgegangen werden, dass, je näher ein Peripheriegerät der Host-Bridge ist, desto weniger wird ein Zugriff auf dieses Gerät durch parallel stattfindende Datentransfers beeinflusst. Wird ein Peripheriegerät über weitere Bussysteme angeschlossen, ist mit zunehmender Anzahl der am jeweiligen Peripheriebus angebundenen Geräte mit höheren Latenzzeiten zu rechnen.

Aus diesen Gründen ist es sinnvoll, die für Realzeitapplikationen benötigten Peripheriegeräte möglichst nah an der Host-Bridge anzuschließen. Falls weitere Peripheriebusse verwendet werden müssen, dann sollten an dem jeweiligen Bus möglichst wenig Peripheriegeräte zusätzlich angeschlossen sein.

Weiterhin wird der Zugriff auf ein Peripheriegerät von der Zeit beeinflusst, die das jeweilige Peripheriegerät benötigt, um auf Datentransfers zu reagieren. Wird mit einem Zugriff auf ein Peripheriegerät eine Aktion auf diesem Gerät ausgelöst, so hängt hier die Ausführungszeit davon ab, wie lange das Peripheriegerät zum Ausführen dieser Aktion benötigt.

Deshalb kann für den Zugriff auf ein Peripheriegerät nicht pauschal gesagt werden, wie lange dieser Zugriff dauern wird. In den vorhergehenden Abschnitten wurde immer der gleiche Zugriff auf ein Peripheriegerät durchgeführt, um *vergleichbare* Messergebnisse zu erhalten, damit beispielsweise die Einwirkungen des PCI-Busses bestimmt werden können. Zur Bestimmung der maximalen Zugriffszeiten einer Realzeittask müssen deshalb zusätzlich die vom Peripheriegerät selbst verursachten Latenzzeiten berücksichtigt werden. Hierzu müssen alle innerhalb einer Realzeittask durchgeführten Zugriffe auf ein Peripheriegerät analysiert und die jeweils benötigten Zugriffszeiten bestimmt werden. Diese Zeiten lassen sich dabei durch Messung oder anhand der Spezifikation des Peripheriegeräts ermitteln.

Insbesondere muss bei den Untersuchungen festgelegt werden, ob hier von einer Realzeittask grundsätzlich immer nur *ein* Zugriff auf das Peripheriegerät erfolgt, oder ob ggf. *mehrere* direkt aufeinanderfolgende Zugriffe durchgeführt werden können. Falls immer nur ein Zugriff durchgeführt wird, dann muss jeweils nur die maximale Ausführungszeit für einen Zugriff bestimmt werden. Können mehrere Zugriffe erfolgen, dann muss zusätzlich die Steigung der resultierenden Geraden für maximale Zugriffszeiten angepasst werden.

Prinzipielles Vorgehen bei der Beschreibung eines Peripheriegeräts ist, dass zunächst die jeweils *minimale* Zugriffszeit auf das Peripheriegerät bestimmt wird, ohne Beeinflussungen im Chipsatz und auf den Peripheriebussen. Anschließend können relativ zu diesem Wert die entsprechenden Verlängerungen der Zugriffszeit angegeben werden. Im allgemeinen Fall ist es ausreichend, hier den Verzögerungsfaktor P für die entsprechenden Zugriffe zu ermitteln, der die Verlängerung der maximalen Zugriffszeit beschreibt. In Abhängigkeit von der maximalen Zugriffszeit $t_{WCAT,P,min}$ für den Zugriff, der am schnellsten vom Peripheriegerät bearbeitet wird, kann die WCAT mit

$$t_{WCAT} = t_{WCAT,P,min} \cdot P \tag{4.22}$$

bestimmt werden.

4.3.5 Interruptlaufzeit

In den vorigen Abschnitten wurde beschrieben, mit welchen Latenzzeiten beim Zugriff auf ein Peripheriegerät oder auf den Hauptspeicher gerechnet werden muss. In diesem Abschnitt soll nun erörtert werden, wie die Laufzeit eines Interrupts bestimmt werden kann.

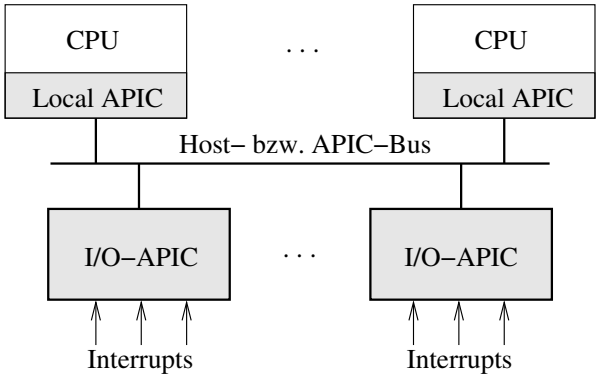


Abbildung 4.24: Zustellung der Interrupts in Multiprozessorsystemen

In Multiprozessorsystemen auf PC-Basis werden APICs für die Zustellung der Interrupts an die einzelnen Prozessoren eingesetzt. APIC ist dabei die Abkürzung für *Advanced Programmable Interrupt Controller*. Jeder Prozessor besitzt dabei einen eigenen APIC, den so genannten *Local APIC*. Zusätzlich ist in einem Multiprozessorsystem noch mindestens ein *I/O-APIC* vorhanden, der die Interrupts von den Peripheriegeräten empfängt und an die Prozessoren weiterleitet.

Die prinzipielle Architektur eines APIC-Verbindungsnetzwerks ist in Abbildung 4.24 dargestellt: Von den Peripheriegeräten wird eine Unterbrechungsanforderung an einen I/O-APIC gemeldet. Dieser entscheidet anhand seiner Routing-Tabellen, an welche CPUs und mit welchem Interruptvektor der Interrupt gemeldet werden soll. Anschließend wird der Interrupt über ein Bussystem an die lokalen APICs der entsprechenden CPUs geschickt.

Welches Bussystem hier zum Einsatz kommt, hängt von der Implementierung des Chipsatzes ab. Prinzipiell existieren die beiden folgenden Möglichkeiten:

- Die Interrupts werden über ein separates Bussystem an die Prozessoren gemeldet. Dieses Verfahren wird vor allem in älteren Multiprozessorsystemen (bis ca. 2002) eingesetzt. Beispiel für einen separaten APIC-Bus ist der *ICC-Bus*, der *Interrupt Controller Communication Bus*, der bei Multiprozessorsystemen mit Intel-CPUs eingesetzt wird.

- In moderneren Multiprozessorsystemen werden Interrupts über den Host-Bus an die Prozessoren gemeldet. Vorteil hiervon ist, dass weniger Signalleitungen auf dem Mainboard benötigt werden. Aus Sicht von Realzeitsystemen resultiert daraus eine höhere Latenzzeit aufgrund parallel stattfindender Datenübertragungen der CPUs. Insbesondere wird auch bei HyperTransport ein Interrupt mit einem Anforderungs-Paket geschickt (siehe Seite 65), dem somit *keine* gesonderte Priorität im Vergleich zu anderen Datentransfers zugeordnet ist.

Sollen Interruptlaufzeiten auf einem Multiprozessorsystem anhand vom Messungen bestimmt werden, dann ergibt sich zunächst das Problem, dass Interrupts — mit Ausnahme der IPIs — nicht von einer CPU aus getriggert werden können. Im Allgemeinen werden Interrupts von Peripheriegeräten ausgelöst, die mit dem jeweiligen Interrupt ein bestimmtes Ereignis signalisieren.

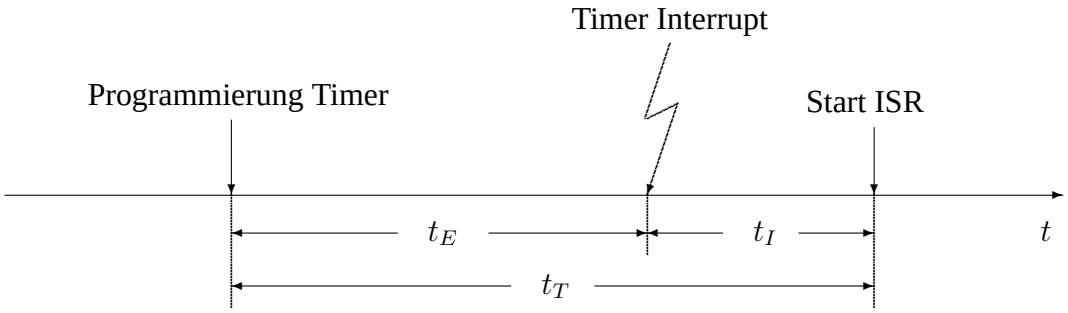


Abbildung 4.25: Bestimmung von Interruptlaufzeiten — One-Shot Timer

Da auf einem Computersystem nur relative Zeiten gemessen werden können — eine Synchronisation mehrerer Uhren ist mit unverhältnismäßig hohem Aufwand verbunden — werden im Folgenden die Interruptlaufzeiten mit den auf dem Multiprozessorsystem vorhandenen *Zeitgeber-Bausteinen (Timer)* bestimmt. Diese können normalerweise im *One-Shot* oder im periodischen Modus betrieben werden. Zusätzlich bieten diese Zeitgeber den Vorteil, dass Interruptlaufzeiten ohne Zusatzhardware bestimmt werden können. Damit ein Zeitgeber für die Untersuchung der Interruptlaufzeiten eingesetzt werden kann, muss er die folgenden Eigenschaften besitzen:

- Um eine hinreichend genaue Zeitauflösung zu gewährleisten, sollte der Zeitgeber mit einer Taktfrequenz im Megaherzbereich arbeiten.
- Der Zeitgeber sollte einen Interrupt auslösen können. Dieser Interrupt muss über einen I/O-APIC an die Prozessoren gemeldet werden.
- Falls der Zeitgeber nur im periodischen Modus programmiert werden kann, muss der Zählerstand des Zeitgebers von Software ausgelesen werden können.

Abbildung 4.25 beschreibt das Szenario, falls die Interruptlaufzeit mit einem im *One-Shot* Modus betriebenen Zeitgeber bestimmt werden soll. Für die Bestimmung der Laufzeit eines Interrupts t_I, wird die *erwartete* Ankunftszeit eines Interrupts t_E von der Zeit t_T, an der dieser *tatsächlich* eintrifft, subtrahiert. t_E entspricht dabei der relativen Zeit bezogen auf den Zeitpunkt der Programmierung des Zeitgebers, an der dieser einen Interrupt auslösen wird. t_T kann mittels des, auf dem Prozessor vorhandenen TSC bestimmt werden und entspricht der Zeitspanne, die zwischen dem Ende der Programmierung des Zeitgebers und dem Aufruf der entsprechenden ISR *(Interrupt Service Routine*) vergeht.

$$t_I = t_T - t_E \tag{4.23}$$

Falls ein Zeitgeber nur den *periodischen* Modus beherrscht, kann — falls die Möglichkeit der Auslesung des Zählerstandes besteht — auch ein solcher Zeitgeber für die Bestimmung der Laufzeit eines Interrupts verwendet werden. Hierbei illustriert Abbildung 4.26 das angenommene Szenario:

Ein periodischer Interrupt wird zu bestimmten Zählerständen des Zeitgebers ausgelöst. Diese Zeiten bzw. Zählerstände, an denen der Zeitgeber einen Interrupt auslöst, sind der jeweiligen

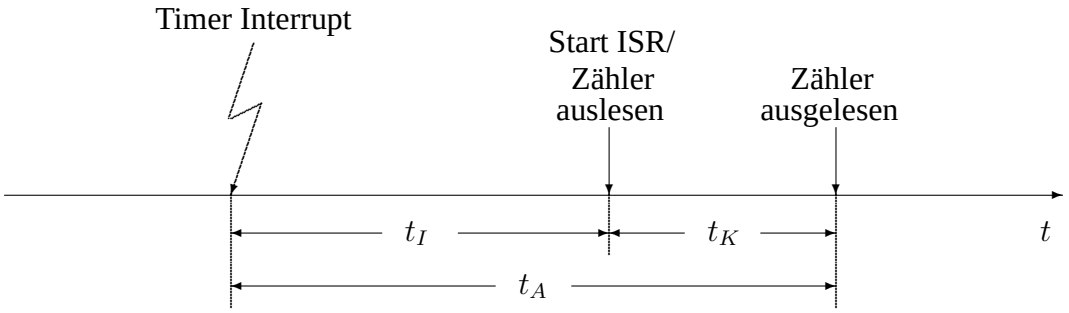


Abbildung 4.26: Bestimmung von Interruptlaufzeiten — Periodic Timer

Messsoftware bekannt. Löst der Zeitgeber einen Interrupt aus, wird der Zählerstand des Zeitgebers ausgelesen. Von diesem Wert wird der Zählerstand subtrahiert, an dem der Zeitgeber den Interrupt ausgelöst hat. Somit erhält man die Zeit t_A, die zwischen dem Auslösen des Interrupts und dem Auslesen des Zählerstandes verstrichen ist. Das Auslesen des Zählerstandes benötigt allerdings auch eine gewisse Zeit, da der Zeitgeber-Baustein als Peripheriegerät angebunden ist und hier ggf. mehrere Peripheriebusse arbitriert werden müssen. Um die Interruptlaufzeit t_I korrekt angeben zu können, wird deshalb der Wert t_A noch um die Zeit t_K korrigiert, die für das Auslesen des Zeitgebers benötigt wird. t_K lässt sich hierbei mit dem auf der CPU vorhandenen TSC bestimmen.

$$t_I = t_A - t_K \tag{4.24}$$

Für die Untersuchungen sind insbesondere der Zeitgeber des ACPI, der auch oft als *PM-Timer (Power Management Timer)* bezeichnet wird, und die HPET (*High Precision Event Timers*) geeignet. Der PM-Timer wird mit einem Takt von 3,58 MHz betrieben und kann periodisch zu definierten Zählerständen einen Interrupt auslösen. Die HPET laufen mit einer Taktfrequenz von mindestens 10 MHz und können im One-Shot, sowie teilweise im periodischen Modus betrieben werden. Die Zählerstände können sowohl bei den HPET, als auch beim PM-Timer ausgelesen werden. Beide Zeitgeber sind normalerweise in der South-Bridge integriert. Interrupts werden jeweils über einen I/O-APIC den Prozessoren gemeldet.

Zur Illustration werden im Folgenden die Interruptlaufzeiten auf einem Athlon und auf einem Opteron Multiprozessorsystem bestimmt. Auf dem Athlon System werden die Interruptlaufzeiten mit Hilfe des PM-Timers ermittelt; auf dem Opteron-System finden die HPET Verwendung. Bei beiden Messungen werden die Zeitgeber in einem periodischen Modus betrieben. Auf den Systemen wird parallel zu den Messungen Last von Peripheriegeräten bzw. von anderen Prozessoren erzeugt. Die Ergebnisse der Untersuchungen sind in Tabelle 4.18 aufgeführt.

Architektur	$t_{min}[\mu s]$	$t_{max}[\mu s]$
Athlon	4,75	15,08
Opteron	2,16	5,66

Tabelle 4.18: Interruptlaufzeiten

4.4 Einflüsse des Power Managements

Alle moderneren Computersysteme auf PC-Basis bieten Stromsparmechanismen an. Die hierzu benötigte Funktionalität wird über das *Advanced Power Management* (APM) bzw. über das modernere *Advanced Configuration and Power Interface* (ACPI) kontrolliert. Da beide Mechanismen einen nicht unerheblichen Einfluss auf Realzeitsoftware haben, soll im folgenden Abschnitt das APM und das ACPI diskutiert werden.

4.4.1 APM

APM bietet insbesondere die Möglichkeit, Prozessoren und Peripheriegeräte, wie beispielsweise Festplatten und Monitor, in einen Schlafzustand zu versetzen oder auch ganz abzuschalten. Dabei werden beim APM bestimmte Hardwareereignisse über einen *System Management Interrupt* (SMI) dem BIOS mitgeteilt. Die Ereignisse selbst können dabei vom Hersteller des Mainboards definiert werden.

Mit Hilfe des SMI kann das BIOS *unabhängig* vom verwendeten Betriebssystem auf ein bestimmtes Ereignis reagieren. Ein mögliches Ereignis wäre beispielsweise eine Temperaturüberschreitung. Ein SMI kann vom Betriebssystem nicht unterdrückt werden. Der SMI wird über das SMI#-Pin an eine CPU geliefert, die anschließend in den *System Management Mode* (SMM) wechselt und dort — in einem eigenen Kontext — den Interrupt bearbeitet. Die zugehörigen Softwareroutinen befinden sich im Speicherbereich des BIOS.

Ein SMI hat Priorität über alle anderen Interrupts. Zusätzlich ergibt sich für Realzeitsysteme die Problematik, dass nicht festgelegt werden kann,

- wann ein SMI auftritt, und
- wie lange eine Unterbrechung dauern wird.

Abhilfe hierzu bietet der Nachfolger des APM, das ACPI, das 1996 spezifiziert wurde und auf allen modernen Multiprozessorsystemen verfügbar ist.

4.4.2 ACPI

Der zentrale Unterschied im Vergleich zum APM ist, dass beim ACPI sämtliche Funktionalitäten des BIOS an das Betriebssystem übertragen werden. Mit Hilfe des ACPIs können beispielsweise Interruptvektoren konfiguriert werden. Weiterhin wird das Betriebssystem in die Lage versetzt, auf bestimmte Ereignisse zu reagieren, die vom Mainboard des PCs gemeldet werden.

Hierzu wird der SMI durch den *System Control Interrupt* (SCI) abgelöst. Der Chipsatz wird dabei so konfiguriert, dass ein Ereignis nicht mehr über das SMI#-Pin an eine CPU gemeldet wird, sondern statt dessen ein SCI ausgelöst wird.

Für Realzeitsysteme bietet dies den Vorteil, dass bei der Verwendung des ACPIs kein SMI das Realzeitverhalten des Gesamtsystems beeinträchtigen kann. Allerdings müssen bei aktivem ACPI die folgenden Aspekte berücksichtigt werden:

- Manche Prozessoren können mit verminderter Taktfrequenz betrieben werden. Gleiches gilt für den Throttling-Modus; hierbei wird für bestimmte Zeitspannen der Prozessortakt ausgesetzt. Durch die verminderte Prozessorleistung erhöht sich jeweils die Ausführungszeit der Software. Bei der Bestimmung von maximalen Laufzeiten muss deshalb davon ausgegangen werden, dass die CPUs in der niedrigsten Leistungsstufe arbeiten.
- Prozessoren und Peripheriegeräte lassen sich in verschiedene Schlafzustände versetzen. Das Aufwachen aus diesen Zuständen ist mit bestimmten Latenzzeiten verbunden, die wiederum vom eingestellten Schlafzustand und von der jeweils verwendeten Hardware abhängen.
- Falls das Betriebssystem nicht auf einen SCI reagiert, kann die Hardware Schaden nehmen. Beispielsweise sollte, falls eine CPU zu heiß wird, die Lüfterdrehzahl erhöht oder die Taktfrequenz verringert werden.

Bei der Verwendung eines Multiprozessorsystems auf PC-Basis für Realzeitaufgaben ist deshalb darauf zu achten, dass CPUs und Chipsatz immer ausreichend gekühlt sind. Somit können die Probleme vermieden werden, die aus einem reduzierten Prozessortakt resultieren. Zusätzlich sollten, um eine minimale Reaktionszeit zu gewährleisten, keine Komponenten, die von einer Realzeittask verwendet werden, in einen Schlafzustand versetzt werden.

4.5 Zusammenfassung

In diesem Kapitel wird beschrieben, welche Einwirkungen auf die Zugriffszeit von Realzeitsoftware bei der Verwendung eines PC-basierten Multiprozessorsystems auftreten können. Dabei zeigen die Untersuchungen, dass Code, der sich bereits im Cache einer CPU befindet, nur minimalen Laufzeitschwankungen unterworfen ist. Daraus folgt, dass sich insbesondere die Pipeline und die Sprungvorhersage kaum auf die Ausführungszeit von Software auswirken.

Wird jedoch der Kontext einer CPU verlassen, kommt es zum einen zu einer erheblichen Verlängerung des Ausführungszeit, zum anderen ist diese Ausführungszeit auch meist großen Laufzeitschwankungen unterworfen. Eine signifikante Verlängerung der Ausführungszeit tritt insbesondere dann auf, falls sich eine CPU nicht mehr aus Daten aus ihrem Cache bedienen kann, sondern auf den relativ langsamen Hauptspeicher zugreifen muss. Neben den Hauptspeicherzugriffen sind auch Zugriffe auf Peripheriegeräte erheblichen Laufzeitschwankungen unterworfen.

In diesem Kapitel wird zunächst untersucht, mit welchen prinzipiellen Beeinflussungen durch den Chipsatz zu rechnen ist. Dabei werden die Untersuchungen sowohl für SMP-, als auch für NUMA-Multiprozessorsysteme vorgenommen. Es zeigt sich, dass in Abhängigkeit von der Anzahl der kontinuierlich durchgeführten Zugriffe auf den Hauptspeicher oder auf Peripherie-

geräte von einem linearen Zugriffsverhalten ausgegangen werden kann. Grund hierfür ist, dass PC-Systeme grundsätzlich nach dem Prinzip der Fairness arbeiten.

Da mit zunehmender Anzahl der beteiligten Komponenten nicht jede mögliche Kombination der Beeinflussungen separat untersucht werden kann, wird in dieser Arbeit vorgeschlagen, die einzelnen Beeinflussungen der Komponenten zunächst voneinander getrennt zu betrachten. Nur jene Konstellationen, bei denen Einflüsse auf die Zugriffszeit vorhandenen sind, müssen anschließend in Kombination untersucht werden.

Aufgrund der Komplexität und der oft nicht verfügbaren Spezifikationen der einzelnen Komponenten der Multiprozessorsysteme werden die einzelnen Parameter, die für die Beschreibung maximaler Zugriffszeiten benötigt werden, mittels Messung ermittelt. Dabei müssen sich alle beteiligten Komponenten wie im Worst-Case verhalten. Parallele Zugriffe von CPUs lassen sich mit relativ geringem Aufwand bewerkstelligen; die Bestimmung des Einflusses von Peripheriegeräten ist dagegen mit höherem Aufwand verbunden. Insbesondere kann hier nicht mit absoluter Sicherheit gesagt werden, dass sich in einem speziellen Szenario das betrachtete Peripheriegerät wie im Worst-Case verhält. Hier kann nur durch Erhöhung der Anzahl der durchgeführten Messungen die Wahrscheinlichkeit erhöht werden, den Worst-Case bestimmt zu haben. Da in dieser Arbeit vorgeschlagen wird, maximale Zugriffszeiten mit der Messung eines bzw. weniger aufeinanderfolgender Zugriffe zu ermitteln und eine größere Anzahl aufeinanderfolgender Zugriffe mit einer Geradengleichung zu approximieren, erhöht sich nochmals die Wahrscheinlichkeit, die jeweils maximal mögliche Zugriffszeit bestimmt zu haben.

Im folgenden Kapitel wird beschrieben, wie diese Erkenntnisse für die Konstruktion eines Realzeitbetriebssystems genutzt werden können. Ziel hierbei ist es, dass Realzeitsoftware so wenig wie möglich durch parallel stattfindende Datentransfers beeinflusst wird. Die verbleibenden Beeinflussungen sollen dabei minimal sein. Parallel zu den Realzeittasks soll ein Standardbetriebssystem mit kompletter Treiber- und Softwareunterstützung ausgeführt werden können.

5 Eine Softwarearchitektur für Realzeitsysteme

In diesem Kapitel wird beschrieben, wie die in Kapitel 4 erlangten Erkenntnisse für die Konstruktion eines Realzeitbetriebssystems herangezogen werden können. Ziel hierbei ist es, sowohl Standard-, als auch Realzeitapplikationen parallel auf einem Multiprozessorsystem ausführen zu können. Dabei sollen auftretende Latenzzeiten minimal sein. Gleichzeitig soll Realzeitsoftware mit der maximal möglichen Geschwindigkeit ausgeführt werden können, die das eingesetzte Multiprozessorsystem ermöglicht.

Hierzu werden in Abschnitt 5.1 die auftretenden Problematiken bei der Verwendung aktueller Realzeitbetriebssysteme für Multiprozessorsysteme beschrieben. Dies motiviert für die in Abschnitt 5.2 diskutierte Softwarearchitektur zur Minimierung hardwarebedingter Laufzeitschwankungen. In Abschnitt 5.3 werden Heuristiken vorgestellt, die für den Realzeitbetrieb benötigt werden. Es wird erläutert, wie die Realzeittasks am sinnvollsten auf die einzelnen Prozessoren verteilt werden sollten, wie Ressourcenbelegungen zwischen den Realzeittasks synchronisiert werden können, welche Schedulingverfahren eingesetzt werden sollten und was bei der Programmerstellung zu beachten ist. Anschließend wird in Abschnitt 5.4 erklärt, wie maximale Ausführungszeiten von Realzeitsoftware ermittelt werden. Diese Zeiten werden in Verbindung mit dem verwendeten Schedulingverfahren für den Realzeitnachweis benötigt. In Abschnitt 5.5 erfolgt abschließend eine Zusammenfassung.

5.1 Motivation

Wird ein Standard- und ein Realzeitbetriebssystem parallel auf PC-Hardware ausgeführt, ergibt sich dadurch der Vorteil, dass das Realzeitbetriebssystem für *harte* Realzeitaufgaben verwendet werden kann. Auf dem Standardbetriebssystem können alle Aufgaben durchgeführt werden, die nur *weichen* Realzeitbedingungen unterliegen. Insbesondere kann das Standardbetriebssystem *(GPOS, General Purpose Operating System)* zur Konfiguration der Realzeittasks und für Visualisierungsaufgaben eingesetzt werden. Unter dem RTOS *(Real-Time Operating System)* werden alle Tasks mit harten Realzeitanforderungen ausgeführt.

Ein weiterer Vorteil bei der Verwendung einer Kombination aus GPOS und RTOS ist, dass im Allgemeinen für das GPOS eine größere Anzahl an Hardwaretreibern zur Verfügung steht. Somit können alle Tasks des GPOS auf die entsprechenden Peripheriegeräte zugreifen. Für das RTOS ist hier ein Zugriff nur eingeschränkt möglich. Gegebenenfalls muss mit höheren Latenzzeiten gerechnet werden, falls die Funktionalität des GPOS verwendet wird.

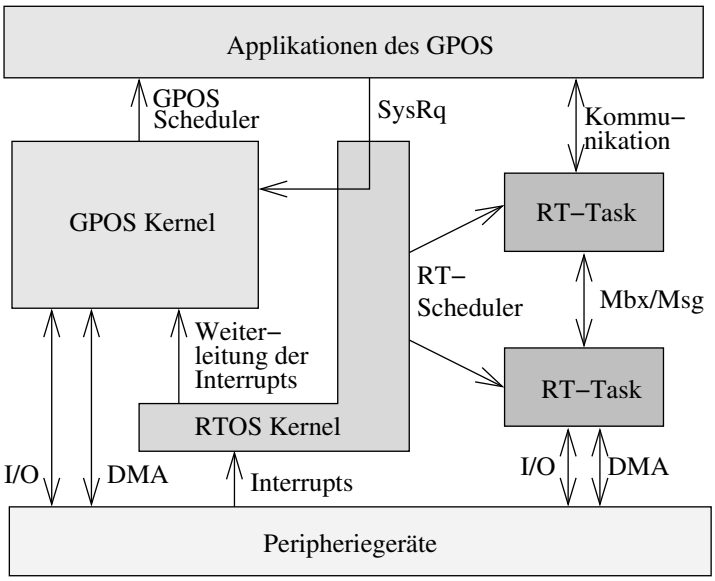


Abbildung 5.1: GPOS mit Realzeiterweiterung

Werden Multiprozessorsysteme auf PC-Basis eingesetzt, bietet sich als GPOS ein Betriebssystem der Windows-Familie oder Linux an. Für diese existieren Erweiterungen, die das jeweilige Betriebssystem um die Realzeitfunktionalität ausbauen. Eine Diskussion der verschiedenen Realzeiterweiterungen für PC-Systeme ist in [11] zu finden. Allen Konzepten, die PC-basierte Multiprozessorsysteme um *harte* Realzeitfunktionalität erweitern und die *ohne* Zusatzhardware auskommen, ist im Allgemeinen folgendes gemeinsam:

- Die *Interruptverwaltung* wird der Kontrolle des RTOS unterstellt. Auftretende Interrupts werden von den Handlern des RTOS bearbeitet, dem GPOS wird ein Interrupt ggf. weitergeleitet.
- Dem GPOS wird die Berechtigung genommen, selbstständig das Auftreten von Interrupts zu sperren. Statt dessen teilt das GPOS dem RTOS mit, dass während der Ausführung kritischer Codeabschnitte keine Interrupts an das GPOS geleitet werden sollen. Die Interruptverwaltung unterliegt dabei vollständig der Kontrolle des RTOS.
- Der Zeitgeber des PC-Systems wird auf eine höhere Auflösung eingestellt. Mit dieser Zeitauflösung arbeitet auch das RTOS.
- Vom RTOS werden Mechanismen bereitgestellt, die für den Realzeitbetrieb benötigt werden. Insbesondere sind dies RT-Scheduler, Warteschlangen für Nachrichten und Mailboxes.
- Es werden Kommunikationsmöglichkeiten mit dem GPOS bereitgestellt.

Der prinzipielle Aufbau der betrachteten Architektur ist in Abbildung 5.1 dargestellt: Interrupts werden von der Peripherie ausgelöst und vom RTOS bearbeitet. In Abhängigkeit von der Unterbrechungsanforderung wird der Interrupt von einer ISR des RTOS bzw. von einer RT-Task

bedient. Hat der Interrupt für das RTOS keine Relevanz und/oder soll das GPOS auf den Interrupt reagieren, so wird er an dieses weitergeleitet.

Standardanwendungen und weiche Realzeittasks werden vom GPOS ausgeführt. Über System-Requests können diese mit dem GPOS bzw. mit dem RTOS kommunizieren. Über spezielle Kommunikationsmechanismen, wie beispielsweise FIFOs, können die Realzeitapplikationen des RTOS und die Tasks des GPOS Daten austauschen. Der Nachrichtenaustausch zwischen den RT-Tasks erfolgt ebenfalls über geeignete Kommunikationsmechanismen, wie beispielsweise Mailboxes.

Dem GPOS und den Realzeitapplikationen ist es erlaubt, zu jedem Zeitpunkt auf Peripheriegeräte zuzugreifen. Falls es der Schutzmechanismus des GPOS erlaubt, darf dieses auch von dessen Tasks durchgeführt werden. Selbiges gilt für die Tasks des RTOS: Hier hängt es vom Design des RTOS ab, ob die RT-Tasks auf Peripheriekomponenten zugreifen dürfen, oder ob dies über das RTOS geschehen muss. Da es den Betriebssystemen erlaubt ist, auf Peripheriegeräte zuzugreifen, kann durch einen entsprechenden Zugriff ein Peripheriegerät ggf. dazu angeregt werden, einen DMA-Transfer zu initiieren.

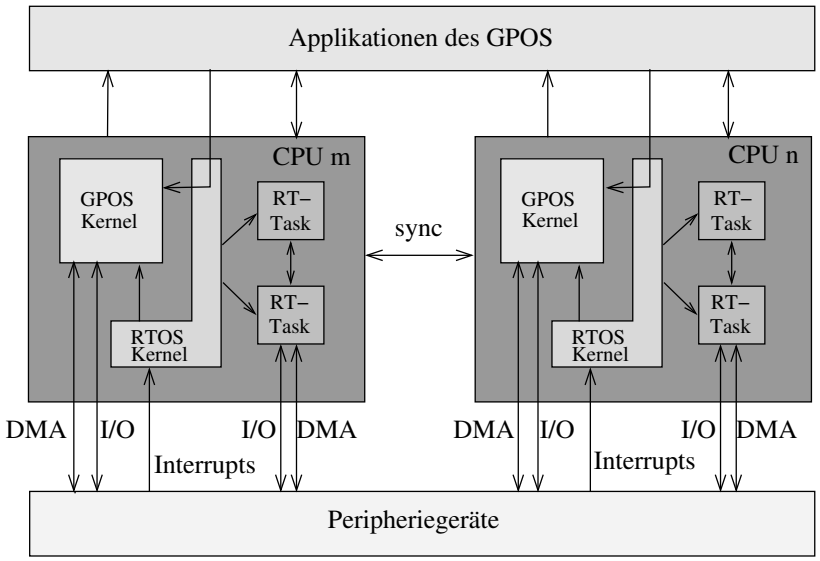


Abbildung 5.2: GPOS mit RTOS auf Multiprozessorsystemen

Werden GPOS und RTOS auf einem Multiprozessorsystem ausgeführt, so entspricht das Szenario im Allgemeinen dem in Abbildung 5.2 dargestellten Fall. Auf jeder CPU des Multiprozessorsystems werden sowohl das RTOS, als auch das GPOS ausgeführt. Somit werden auf allen Prozessoren abwechselnd Standard- und Realzeittasks aufgerufen. Interrupts werden von jeder CPU bearbeitet. Im Allgemeinen werden hier die Interrupts in Abhängigkeit von der erzeugten Last den einzelnen Prozessoren zugewiesen und ggf. auf jedem Prozessor an das GPOS weitergeleitet.

Zugriffe auf Peripheriegeräte können vom GPOS und vom RTOS von jeder CPU aus durchgeführt werden. Somit kann prinzipiell jede CPU zu jedem Zeitpunkt Peripheriegeräte adressieren

und ggf. DMA-Transfers anstoßen. Weiterhin ist es Peripheriegeräten möglich, selbstständig DMA-Transfers durchzuführen. Das Ende der DMA-Operation wird anschließend einer beliebigen CPU mitgeteilt. Hierfür wird ein Interrupt verwendet.

Zusätzlich synchronisieren sich die Prozessoren des Multiprozessorsystems. Insbesondere werden IPIs zwischen den jeweiligen Instanzen des GPOS und des RTOS verschickt. Zum Austausch von Nachrichten wird normalerweise ein globaler Speicherbereich verwendet, auf den alle Prozessoren zugreifen.

Die beschriebene Architektur führt zu den im Folgenden aufgeführten Einschränkungen für Realzeitsoftware:

- Das GPOS wird auf jeder CPU des Multiprozessorsystems ausgeführt. Falls eine Realzeittask auf einem Prozessor fortgesetzt wird, kann nicht definiert werden, in welchem Zustand sich die jeweilige CPU befindet. Insbesondere können bei der Wiederaufnahme einer Realzeittask Inhalte der Caches durch das GPOS verdrängt worden sein. Selbiges gilt für den Zustand der *Translation Lookaside Buffers (TLB)* und für die Sprungvorhersage. Falls sich der Prozessor beim Wechsel vom GPOS zum RTOS in einem niederprivilegierten Modus befand, können aufgrund dieses Kontextwechsels zusätzliche Latenzzeiten entstehen.

- Interrupts werden an jede CPU geleitet. Da beim Auftreten eines Interrupts zunächst nicht unterschieden werden kann, um welchen Interrupt es sich handelt, wird hierbei im allgemeinen Fall die Ausführung einer Realzeittask unterbrochen und die entsprechende ISR des RTOS aufgerufen. Hier entscheidet sich, ob der Interrupt sofort bearbeitet werden muss, oder ob er ggf. später vom GPOS behandelt werden kann. Da jedoch viele Interrupts nur für das GPOS von Interesse sind — beispielsweise sind hier die Interrupts von Tastatur und Maus zu nennen — wird die Realzeittask zusätzlichen Latenzzeiten unterworfen. Im ungünstigsten Fall werden durch das Auftreten eines Interrupts Verdrängungen in den Caches und TLBs verursacht, die sich zusätzlich negativ auf die Ausführungszeit von Realzeitsoftware auswirken.

- Jeder Prozessor darf I/O-Zugriffe durchführen und somit DMA-Transfers der Peripheriegeräte anstoßen. Dieses kann sowohl durch das GPOS, als auch durch das RTOS geschehen. Zusätzlich können die Peripheriegeräte prinzipiell zu jedem Zeitpunkt autonom Datentransfers durchführen. Muss parallel zu diesen Transfers eine Realzeittask auf den Hauptspeicher oder auf Peripheriegeräte zugreifen, kommt es zu teilweise erheblichen Verzögerungen beim Zugriff auf die jeweilige Ressource. Die Bestimmung der Größenordnung dieser Beeinflussungen wurde ausführlich in Kapitel 4 behandelt.

Im folgenden Abschnitt wird beschrieben, wie obige Problematiken gehandhabt werden können. Ziel hierbei ist es, Standard- und Realzeittasks parallel ausführen zu können und trotzdem minimale Ausführungs- und Reaktionszeiten zu ermöglichen.

5.2 Minimierung hardwarebedingter Laufzeitschwankungen

In diesem Abschnitt werden die entwickelten Maßnahmen zur Minimierung maximaler Lauf- und Zugriffszeiten von Realzeittasks bei der Verwendung eines PC-basierten Multiprozessorsystems erläutert. Diese Maßnahmen sind in Abbildung 5.3 illustriert und werden in den folgenden Unterabschnitten diskutiert.

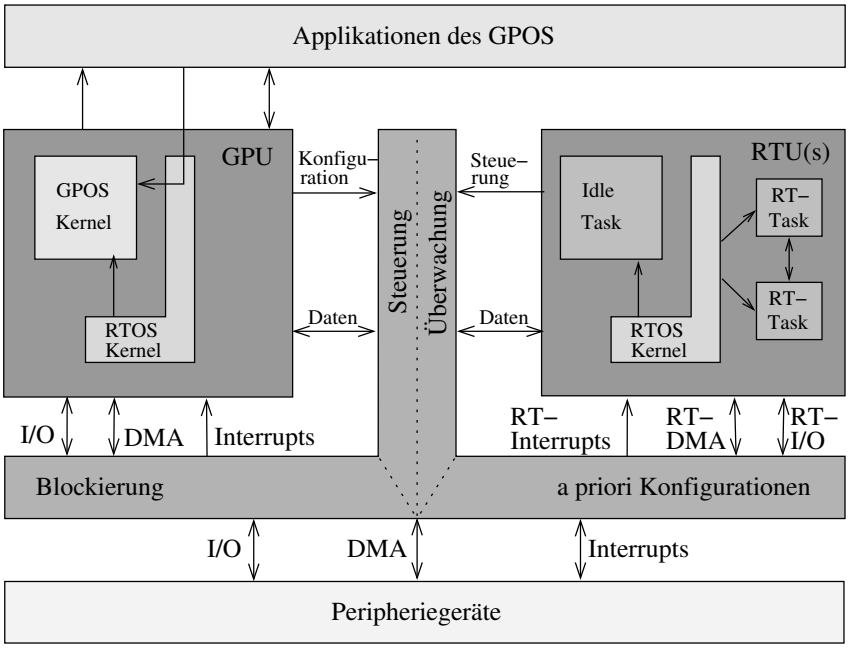


Abbildung 5.3: Softwarearchitektur

5.2.1 Trennung von Standard- und Realzeitbetriebssystem

Die Prozessoren des Multiprozessorsystems werden in eine *GPU (General Purpose Unit)* und eine oder mehrere *RTUs (Real-Time Units)* eingeteilt. Wie in Abbildung 5.3 dargestellt, werden die Applikationen des GPOS auf der GPU ausgeführt. Alle weiteren Prozessoren stehen exklusiv für harte Realzeitapplikationen zur Verfügung.

Dieses Konzept bietet den Vorteil, dass Steuerungs- und Visualisierungsaufgaben ausschließlich auf der GPU ausgeführt werden. Für diese Aufgabe, sowie ggf. für weitere Applikationen, wie beispielsweise Kommunikationsaufgaben, die keinen harten Realzeitbedingungen unterliegen, ist die hierfür benötigte Performanz einer einzelnen x86-CPU im Allgemeinen ausreichend.

Alle weiteren Prozessoren stehen uneingeschränkt für den Realzeitbetrieb zur Verfügung. Hiermit wird erreicht, dass die Leistungsfähigkeit moderner Prozessoren exklusiv für Realzeitaufgaben verwendet werden kann, ohne diese mit Standardapplikationen teilen zu müssen. Dadurch

können die RTUs permanent in der höchsten Privilegierungsstufe betrieben werden, womit ein Kontextwechsel vom GPOS zum RTOS und die damit verbundene Latenzzeit entfällt.

Die vorgestellte Architektur kann beispielsweise dadurch realisiert werden, dass auf den RTUs eine Idle-Task gestartet wird, die eine höhere Priorität als das GPOS besitzt. Somit kann auf den entsprechenden Prozessoren das GPOS nicht mehr ausgeführt werden. Damit das GPOS funktionsfähig bleibt, müssen *vor* dem Start der Idle-Task auf den RTUs alle Tasks des GPOS auf die GPU migriert werden. Zusätzlich müssen alle Synchronisationsmechanismen des GPOS mit den CPUs des Multiprozessorsystems und alle Task-Migrationsmechanismen deaktiviert werden. Insbesondere dürfen keine IPIs des GPOS zwischen den einzelnen CPUs versandt werden, die beispielsweise auf allen CPUs einen TLB-Flush erzwingen.

Zusätzlich werden die Interrupts des GPOS von jenen getrennt, die für Realzeitapplikationen benötigt werden. Dies wird im folgenden Abschnitt beschrieben.

5.2.2 Interruptzuordnung

Bei der beschriebenen Softwarearchitektur werden Interrupts, die ausschließlich vom GPOS bedient werden müssen, exklusiv an die GPU geschickt. Dies wird durch eine entsprechende Programmierung der I/O-APICs erreicht. Interrupts, die an die GPU geleitet werden, sind insbesondere die Interrupts des Maus- und Tastaturcontrollers und die Interrupts von Grafik- und Soundkarte. Falls auf Massenspeicher oder Netzwerkadapter ausschließlich vom GPOS aus zugegriffen wird, müssen auch die Interrupts dieser Geräte an die GPU geliefert werden.

Interrupts, auf die eine ISR des RTOS reagieren muss, werden an eine RTU geschickt. Dabei sollte der Interrupt an diejenige RTU geliefert werden, auf der auch eine ggf. dazugehörende Realzeittask ausgeführt wird. Beispielsweise wird der Interrupt eines Prozesssignaladapters exklusiv an die RTU geliefert, auf der die entsprechende Realzeittask ausgeführt wird, die dieses Peripheriegerät bedient.

Für Realzeitsysteme resultieren hieraus die im Folgenden aufgeführten Vorteile:

- Die Ausführung von Realzeittasks wird nicht durch Anforderungen unterbrochen, die nicht für das Realzeitsystem relevant sind. Da bei Auftreten eines Interrupts vom Prozessor nicht unterschieden wird, um welche Art von Unterbrechung es sich handelt, wird *immer* das laufende Programm unterbrochen und die entsprechende ISR aufgerufen. Erst durch diesen Aufruf wird vom Betriebssystem erkannt, um welchen Interrupt es sich handelt.
- Da eine Task nicht durch einen Interrupt unterbrochen wird, der Code des GPOS ausführt, können Code und Daten des RTOS besser im Cache gehalten werden. Insbesondere entstehen keine Verdrängungen in den Caches und TLBs, die durch den Aufruf des GPOS verursacht werden. Diese Thematik wird nochmals in Abschnitt 5.2.6 aufgegriffen.
- Die Interruptzuordnung kann vor dem Start der Realzeitapplikationen durchgeführt werden. Deshalb resultieren aus dieser Maßnahme keine zusätzlichen Latenzzeiten.

Mit der Aufteilung der Prozessoren in GPU und RTUs, sowie mit der Verteilung der Interrupts auf die entsprechenden Prozessoren, wird erreicht, dass das GPOS auf keiner RTU ausgeführt werden kann. Daraus resultiert unmittelbar eine geringere Reaktionszeit der Realzeittasks. Allerdings ergeben sich beim Zugriff auf gemeinsam genutzte Ressourcen des Multiprozessorsystems Beeinflussungen, die teilweise nicht unerheblich sind. Wie diese von der vorgestellten Softwarearchitektur gehandhabt werden, wird im folgenden Abschnitt erläutert.

5.2.3 Ressourcenzugriffe des Standardbetriebssystems

Die Größenordnung der resultierenden Latenzzeiten bei paralleler Last wurde bereits ausführlich in Kapitel 4 untersucht. Dabei wurde festgestellt, dass sich unter anderem Zugriffe auf den Hauptspeicher bei SMP-Systemen bzw. Zugriffe auf Hauptspeicherbereiche bei NUMA-Systemen, sowie Zugriffe auf die gleichen PCI-Busse gegenseitig beeinflussen. Zusätzlich wirken sich DMA-Transfers der Peripheriegeräte auf das Laufzeitverhalten von Realzeitsoftware aus, wobei diese Transfers von den Peripheriegeräten autonom durchgeführt werden können. Diese Thematik wird im folgenden Abschnitt 5.2.4 behandelt.

Die Auswirkungen paralleler Ressourcenzugriffe des GPOS bzw. der GPU lassen sich in parallele Zugriffe auf den Hauptspeicher und in parallele Zugriffe auf Peripheriekomponenten separieren. Beide sollen im Folgenden diskutiert werden:

Hauptspeicherzugriffe des Standardbetriebssystems

Prinzipiell kann davon ausgegangen werden, dass das GPOS während der Ausführung der Standardapplikationen *immer* auf den Hauptspeicher zugreifen muss. Im Gegensatz zu den Tasks einer Realzeitapplikation sind die Applikationen des GPOS im Allgemeinen sehr speicherintensiv. Falls nicht generell mit parallelen Hauptspeicherzugriffen der GPU gerechnet werden soll — in diesem Fall würde sich eine höhere maximale Zugriffszeit auf den Hauptspeicher ergeben — können die folgenden Maßnahmen durchgeführt werden:

1. Am naheliegendsten ist, die Ausführung des GPOS zu den Zeitpunkten, an denen eine Realzeitapplikation auf den Hauptspeicher zugreift, zu stoppen. Zu welchen Zeitpunkten dies geschieht, wird in [14] untersucht. Allerdings wird für die Deaktivierung des GPOS auch eine gewisse Zeit benötigt, da hier ein IPI an die GPU geschickt werden muss. Zusätzlich muss in die Realzeitapplikation Code eingefügt werden, der das GPOS deaktiviert und nach den Hauptspeicherzugriffen wieder aktiviert. Ein sinnvoller Einsatz in Realzeitsystemen ist folglich nur dann möglich, falls von einer Realzeittask eine große Anzahl aufeinanderfolgender Hauptspeicherzugriffe durchgeführt werden muss. Dabei wird durch diese Maßnahme die Leistungsfähigkeit des GPOS verringert.

2. Die Realzeitapplikation könnte vollständig im Cache einer RTU fixiert werden. Wie dies zu bewerkstelligen ist, kann ebenfalls in [14] nachgelesen werden. Falls die Realzeitapplikation nicht auf den Hauptspeicher zugreifen muss, kommt es ebenfalls zu keinen Beeinflussungen. Voraussetzung ist, dass das GPOS keine Verdrängungen im Cache der je-

weiligen RTU verursacht, wie sie beispielsweise bei der Verwendung von gemeinsam genutzten globalen Variablen auftreten.

3. Es kann ein NUMA-Multiprozessorsystem verwendet werden, bei dem das GPOS *ausschließlich* auf den im Knoten der GPU vorhandenen Speicher zugreift. Somit treten nur auf diesem Knoten gegenseitige Beeinflussungen auf. Auf den anderen Knoten können nur Beeinflussungen der Realzeitapplikationen untereinander auftreten. Diese Thematik wird bei der in Abschnitt 5.2.6 beschriebenen Codeanordnung behandelt.

Hier hängt es vom vorhandenen Multiprozessorsystem und von den Anforderungen der Realzeittasks an den Hauptspeicher ab, welche Maßnahme verwendet werden soll. Falls ein NUMA-System eingesetzt werden kann, ist sicherlich eine Kombination der Methoden 2 und 3 am sinnvollsten.

Peripheriezugriffe des Standardbetriebssystems

Gegenseitige Beeinflussungen zwischen GPOS und RTOS treten dann auf, falls auf Peripheriegeräte zugegriffen wird, die über einen *gemeinsam verwendeten* PCI- oder PCI-X-Bus angeschlossen sind. Diese Beeinflussungen können vermieden werden, falls die Peripheriegeräte auf mehrere PCI-Busse aufgeteilt werden, wobei diese Busse voneinander unabhängig sein müssen. Somit können die Peripheriegeräte, die vom GPOS verwendet werden, an dem einen PCI-Bus betrieben werden. Die von den Realzeitapplikationen benötigten Peripheriegeräte müssen dann an einem anderen PCI-Bus angeschlossen sein.

Falls dies nicht möglich ist — in einem klassischen North-/South-Bridge-System existiert beispielsweise nur ein zentraler PCI-Bus, über den ggf. weitere PCI-Busse über eine Bridge angebunden sind — kann das GPOS zu bestimmten Zeiten daran gehindert werden, auf Peripheriegeräte zuzugreifen. Dies lässt sich mit relativ geringem Aufwand realisieren, wobei insbesondere die Performanz des GPOS im Allgemeinen nicht wesentlich unter der durchgeführten Maßnahme leidet. Insbesondere muss hier nur eine globale Variable gesetzt werden, die dem GPOS anzeigt, dass es während bestimmter Abschnitte nicht auf Peripheriegeräte zugreifen darf. Die verwendete Methodik besitzt dabei die im Folgenden aufgeführten Eigenschaften, die auch in Abbildung 5.3 illustriert sind:

1. Vor dem Start der Realzeitapplikation werden die Adressräume konfiguriert, auf denen es zu gegenseitigen Beeinflussungen zwischen dem GPOS und dem RTOS kommen könnte. Insbesondere werden hier die entsprechenden Portadressen der Peripheriegeräte angegeben, sowie jene Adressen, auf die mittels Memory-Mapped-I/O zugegriffen wird.

2. Der Zugriff des GPOS auf Peripheriegeräte, der unmittelbar zu Beeinflussungen mit einer Realzeitapplikation führen könnte, wird dem GPOS für die Dauer der Zugriffe des RTOS verwehrt:

   - Vor jedem *Portzugriff* überprüft das GPOS, ob das RTOS diesen Zugriff eventuell gesperrt hat. Falls die Zugriffe des GPOS blockiert sind, wird die momentan ausgeführte Task ausgesetzt und eine andere Task fortgeführt. Da Portzugriffe des GPOS

im Allgemeinen über eine API durchgeführt werden, kann die entsprechende Funktionalität dem GPOS mit geringem Aufwand hinzugefügt werden.

- Über *Memory-Mapped-I/O* durchgeführte Peripheriezugriffe können ebenfalls zurückgestellt werden. Hierzu existieren zwei Methoden, mit denen dies erreicht werden kann: Die Page-Tables der GPU können für die Dauer einer Sperrung umkonfiguriert werden, so dass auf der GPU eine Schutzverletzung ausgelöst wird, sobald diese auf ein Peripheriegerät zugreift. Dieser Fall muss anschließend in der entsprechenden Exception gesondert behandelt werden. Alternativ kann auch — falls es das verwendete Betriebssystem unterstützt — die entsprechende Funktionalität hinter der API angepasst werden, so dass vor einem Peripheriezugriff in Äquivalenz zu Portzugriffen der Speicherbereich der Zieladresse überprüft wird. Letztere Methode ist besser geeignet, da hierbei geringere Latenzzeiten zur Laufzeit entstehen.

Falls Peripheriezugriffe des GPOS jeweils über eine Modifikation der Funktionalität hinter der API blockiert werden, ist die zusätzlich benötigte Ausführungszeit der Realzeitapplikationen gering. Diese entspricht der Zeit, die für das Ändern einer globalen Variablen benötigt wird. Falls jedoch die Page-Tables der GPU modifiziert werden, muss mit höheren Reaktionszeiten gerechnet werden, da hier der GPU ein IPI geschickt werden muss. Dieser zeigt der GPU insbesondere die modifizierten Page-Tables an.

3. Die Größenordnung, mit der die Tasks des GPOS durch die durchgeführten Maßnahmen verlangsamt werden, hängt von der I/O-Last der jeweiligen Applikation ab. Dabei werden Tasks der GPOS, die viele Peripheriezugriffe ausführen, stärker beeinflusst als jene, die nicht auf die Peripherie zugreifen. Weitere Parameter sind die Häufigkeit und die Dauer, mit der das RTOS exklusiv auf bestimmte Peripheriekomponenten zugreifen muss.

Neben den Beeinflussungen durch die GPU treten auch Latenzzeiten auf, die durch parallel stattfindende DMA-Transfers der Peripheriegeräte verursacht werden. Wie diese gehandhabt werden sollten, beschreibt der folgende Abschnitt.

5.2.4 Beeinflussungen durch Peripheriegeräte

Beeinflussungen durch Peripheriegeräte treten insbesondere dann auf, wenn eine Realzeittask auf einen Peripheriebus zugreifen muss, an dem gerade ein Gerät einen Burst-Transfer durchführt. Zusätzlich treten weitere Latenzzeiten beim Zugriff auf den Hauptspeicher auf, falls hier ein Peripheriegerät parallel zu diesem Zugriff ebenfalls einen Zugriff auf den Hauptspeicher ausführt.

Diese Beeinflussungen lassen sich eliminieren, falls die im Folgenden aufgeführten Maßnahmen durchgeführt werden:

- Peripheriegeräte sollten in Abhängigkeit davon, ob sie für Realzeitapplikationen benötigt werden oder nicht, an unterschiedliche PCI-Busse angeschlossen werden. Somit kann ein Zugriff einer Realzeittask auf ein Peripheriegerät nicht durch Geräte behindert werden, die vom GPOS bedient werden.

- Peripheriegeräte, die das GPOS bedient, dürfen nur auf den vom GPOS verwendeten Speicherbereich zugreifen. Insbesondere bei der Verwendung eines NUMA-Systems resultiert hieraus der Vorteil, dass Speicherzugriffe einer RTU auf ihren eigenen Speicherbereich nicht durch parallel durchgeführte DMA-Transfers beeinflusst werden. Voraussetzung hierfür ist, dass das GPOS ausschließlich auf den, auf dem Knoten der GPU enthaltenen Hauptspeicherbereich zugreift.

- Durch Löschen des Master Enable Bits (vgl. S. 79) können Peripheriegeräte daran gehindert werden, selbstständig einen DMA-Transfer durchzuführen. Werden diese Bits während des Zugriffs einer Realzeittask auf den entsprechenden Peripheriekomponenten gelöscht, werden diese Zugriffe nicht durch parallele Anforderungen an den PCI-Arbiter beeinflusst. Prinzipiell kann diese Methode auch bei Hauptspeicherzugriffen einer RTU eingesetzt werden. Hier hängt es von den Anforderungen des Realzeitsystems an die Peripherie ab, ob diese Methode sinnvoll eingesetzt werden kann. Insbesondere ist dies der Fall, falls sehr viele aufeinanderfolgende Zugriffe einer Realzeittask auf ein Peripheriegerät durchgeführt werden müssen. Für sporadische Zugriffe vieler Realzeittasks kann diese Methode dagegen nicht sinnvoll eingesetzt werden.

Die Zeit, die zum Löschen der Master Enable Bits benötigt wird, hängt von der Anzahl der zu deaktivierenden Geräte ab. Pro Gerät muss hierbei ein Konfigurationszugriff durchgeführt werden. Wie diese Zeiten ermittelt werden können, wurde in Abschnitt 4.3.3 beschrieben. Die benötigten Zugriffszeiten lassen sich verkürzen, wenn alle zu deaktivierenden Geräte an den gleichen PCI-Bus angeschlossen werden, der beispielsweise über eine PCI-to-PCI Bridge an das System angebunden ist. In diesem Fall muss nur das Master Enable Bit der Bridge gelöscht werden. Alle PCI-Geräte hinter dieser Bridge können dann nicht mehr auf den Hauptspeicher oder auf PCI-Geräte jenseits der Bridge zugreifen.

Nachdem die Realzeittask den Zugriff auf den PCI-Bus durchgeführt hat, müssen die Master Enable Bits der zuvor deaktivierten Geräte wieder gesetzt werden. Da hierbei keine parallele Last vorhanden ist, muss mit keinen Latenzzeiten durch PCI-Geräte gerechnet werden.

5.2.5 Speicherverwaltung

Für die vorgeschlagene Softwarearchitektur wurde eine Speicherverwaltung implementiert, die insbesondere für die im folgenden Abschnitt beschriebene Codeanordnung benötigt wird. Hierbei wird der obere Teil des Hauptspeichers exklusiv den RTUs zur Verfügung gestellt. Dem GPOS ist die Existenz des reservierten Speichers nicht bekannt. Für Realzeitsysteme bietet dies die folgenden Vorteile:

- Auf NUMA-Systemen kann mit Hilfe dieser Speicherverwaltung der Speicherbereich der RTUs ausschließlich für diese reserviert werden. Zugriffe des GPOS erfolgen nur auf den Speicherbereich der GPU.

- Im Speicherbereich der RTUs muss mit keinen Snooping-Effekten und Cacheverdrängungen gerechnet werden, die durch das GPOS verursacht werden.

- Es steht großer, *physikalisch zusammenhängender* Speicherbereich zur Verfügung. Dieser lässt sich für Anordnungen von Code und Daten im Cache, sowie für vom RTOS bediente Gerätetreiber nutzen.

Dies wird realisiert, indem die verkleinerte Größe des Hauptspeichers dem GPOS beim Bootvorgang mitgeteilt wird. Damit der Speicher der RTUs initialisiert werden kann, wird dieser Speicherbereich anschließend in den virtuellen Adressraum des GPOS eingeblendet. Das hierbei verwendete Verfahren entspricht demjenigen, mit dem auch der Adressraum von Peripheriegeräten — auf die mittels Memory-Mapped-I/O zugegriffen wird — in den Adressraum des GPOS eingeblendet wird.

5.2.6 Codeanordnung

Mit Hilfe der Anordnung des Binärcodes im physikalischen Hauptspeicher kann erreicht werden, dass bestimmte Code- und Datenbereiche auf die gleichen *Sets* der in den CPUs vorhandenen Caches abgebildet werden. Hiermit wird ermöglicht, dass bestimmte Teile des Codes des RTOS im Cache einer RTU fixiert werden können. Falls die einzelnen Realzeittasks nicht zu umfangreich sind, ist es mit dieser Methode möglich, das komplette RTOS und dessen Applikationen im Cache der RTUs anzuordnen: Um Code und Daten permanent in einem Cache einer CPU zu halten, muss jeweils der selbe Prozentsatz in Cache und Hauptspeicher reserviert werden. Falls beispielsweise der Cache einer CPU 1024 KByte groß ist und das RTOS hier 768 KByte belegt, müssen auch 75% des Hauptspeichers, auf den diese CPU zugreift, hierfür reserviert werden. Auf Singleprozessormaschinen macht dies im Allgemeinen wenig Sinn, da hier die Verschwendung an Hauptspeicher zu groß wäre. Da die RTUs aber generell auf ihrem eigenen Speicherbereich arbeiten, muss folglich nur von diesem Bereich der entsprechende Prozentsatz reserviert werden.

Für Realzeitapplikationen resultiert hieraus der Vorteil, dass die Reaktionszeiten der Realzeittasks erheblich verringert werden können. Die Größenordnungen, in den sich die jeweiligen Zugriffszeiten von Cache- und Hauptspeicherzugriffen bewegen, wurden in Abschnitt 4.2.1 bestimmt. Falls Code und Daten des RTOS vollständig im Cache fixiert sind, resultiert hieraus der weitere Vorteil, dass das RTOS nicht durch parallele Speichertransfers des GPOS, sowie durch DMA-Transfers der Peripheriegeräte, beeinflusst wird.

Die zur Anordnung von Code und Daten entwickelten Methodiken werden ausführlich in der Dissertation von A. von Bülow [14] behandelt. An dieser Stelle soll deshalb hierauf nicht weiter eingegangen werden.

5.3 Realzeitbetrieb

In diesem Abschnitt wird beschrieben, welche Methoden für den Realzeitbetrieb angewendet werden können. Hierzu wird im folgenden Unterabschnitt erläutert, wie die einzelnen Tasks und ISRs unter Berücksichtigung der PC-Architektur auf den RTUs verteilt werden sollten. Auf

die Synchronisation der Ressourcenbelegungen, auf die Auswahl geeigneter Schedulingverfahren und auf Methoden zur Programmerstellung wird in den darauffolgenden Unterabschnitten eingegangen.

5.3.1 Heuristiken für die Interrupt- und Taskzuordnung

Die Tasks und ISRs des Realzeitsystems müssen verschiedenen RTUs zugeordnet werden. Dabei wird zunächst davon ausgegangen, dass die Tasks *statisch* während der Initialisierungsphase auf die einzelnen RTUs verteilt werden. Ein nachträglicher Wechsel der Tasks von einer RTU auf eine andere erfolgt nicht (vergleiche hierzu auch Abschnitt 5.3.3).

Um Tasks und ISRs den jeweiligen Prozessoren zuordnen zu können, müssen wichtige Parameter des Realzeitsystems und der eingesetzten Hardware bekannt sein. Die Parameter des Realzeitsystems resultieren insbesondere aus den Anforderungen, die das Realzeitsystem erfüllen muss. Von besonderem Interesse sind

- die Anzahl der benötigten Realzeittasks,
- der Bedarf an Speicherplatz,
- die Ressourcenanforderungen der Realzeittasks und
- die gegenseitige Abhängigkeit der Realzeittasks bzgl. Ressourcenzugriffen, zeitlicher Abfolge, etc.

Weitere Parameter, die insbesondere für den Realzeitnachweis (vgl. Abschnitt 5.4) benötigt werden, sind Periode bzw. Ereignisfunktion, sowie die Deadline einer Realzeittask. Die aufgeführten Parameter hängen vom jeweiligen Szenario ab und sind im Allgemeinen nicht veränderbar.

Die Realzeittasks werden auf einem PC-basierten Multiprozessorsystem ausgeführt. Die relevanten Eigenschaften dieses Multiprozessorsystems müssen ebenfalls bekannt sein, um die Tasks den jeweiligen CPUs zuordnen zu können. Insbesondere sind dies:

- Die eingesetzte Architektur. Hier wird zwischen SMP- und NUMA-Multiprozessorsystemen unterschieden.
- Die Anzahl der vorhandenen Prozessoren.
- Die Anzahl der verfügbaren, voneinander unabhängigen Peripheriebusse.[1)]
- Die Größe des vorhandenen Hauptspeichers und die Größe der Caches der eingesetzten Prozessoren.
- Bei NUMA-Systemen muss Folgendes zusätzlich bekannt sein:
  - Die Anzahl der Peripheriebusse und Peripheriegeräte pro Knoten.
  - Die Größe des verfügbaren Hauptspeichers pro Knoten.

[1)] Zwei PCI-Busse sind voneinander unabhängig, falls diese *nicht* über eine PCI-to-PCI Bridge miteinander verbunden sind.

Ziel bei der Zuordnung der einzelnen Tasks ist es, Beeinflussungen durch parallele Datentransfers so weit als möglich zu vermeiden. In Abhängigkeit von der Anzahl parallel möglicher Datenübertragungen bei Zugriffen auf Peripheriegeräte oder den Hauptspeicher resultieren unterschiedliche maximale Zugriffszeiten. Falls bestimmte Beeinflussungen nicht auftreten können, kann bei der Bestimmung der WCET der einzelnen Codefragmente von geringeren maximalen Zugriffszeiten ausgegangen werden. Somit verringert sich zum einen die Reaktionszeit der Realzeittasks, zum anderen erhöht sich auch die Anzahl der Tasks, die auf einer RTU ausgeführt werden können. Die einzelnen Anhaltspunkte für die Zuordnung der Tasks und ISRs werden im Folgenden beschrieben.

Kriterien für die Zuordnung der Interruptserviceroutinen

Die ISRs des RTOS sollten von der RTU bearbeitet werden, auf der auch die zugehörigen Realzeittasks ausgeführt werden. Somit müssen keine IPIs zwischen den RTUs verschickt werden, die die jeweilige Task anstoßen. Bei NUMA-Systemen sollte die ISR von der RTU ausgeführt werden, auf deren Knoten sich auch das zugehörige Peripheriegerät befindet. Hieraus resultieren ggf. geringfügig verringerte Latenzzeiten.

Allgemeingültige Kriterien für die Taskzuordnung

Bei der Taskzuordnung gilt insbesondere, dass die jeweiligen Tasks des Realzeitsystems in Abhängigkeit ihrer Deadline, Rechenzeit und Ereignisfunktion bzw. Periode schritthaltend ausführt werden müssen. Weiterhin müssen die für den Realzeitbetrieb benötigten Parallelitäten eingehalten werden, falls beispielsweise mehrere Prozessoren *gleichzeitig* eine Aufgabe bearbeiten sollen.

Zusätzlich existieren die im Folgenden aufgeführten Kriterien, die sowohl für SMP-, als auch für NUMA-Systeme herangezogen werden können:

Zuordnung anhand des Speicherbedarfs: Die Tasks können so auf den RTUs verteilt werden, dass Tasks, die eine geringe Codegröße haben, im Cache einer RTU fixiert werden. Insbesondere eignet sich dies für jene Tasks, die eine kurze Deadline besitzen. Tasks, die mehr Speicherplatz benötigen und öfters auf den Hauptspeicher zugreifen, werden von einer anderen RTU ausgeführt. Selbiges gilt insbesondere auch für ISRs.

Zuordnung anhand der adressierten Peripheriegeräte: Falls mehrere Tasks des Realzeitbetriebssystems auf Peripheriegeräte zugreifen müssen, kann die Zuordnung anhand der PCI-Busse erfolgen, an die die zugehörigen Peripheriegeräte angebunden sind. Somit werden parallele Zugriffe anderer CPUs auf den PCI-Bus vermieden, wodurch mit geringeren Latenzzeiten beim Zugriff gerechnet werden kann (vgl. Abschnitt 5.3.2). Insbesondere entfallen damit auch weitere Synchronisierungsmaßnahmen.

Zusätzliche Kriterien für NUMA-Systeme

Der Vorteil von NUMA-Systemen ist, dass hier bei geeigneter Taskzuordnung im Vergleich zu SMP-Systemen weniger Beeinflussungen auftreten. Bei der Zuordnung der Tasks auf die einzelnen RTUs können die folgenden Aspekte *zusätzlich* berücksichtigt werden:

- Code und Daten der Realzeittasks sollten im Speicher des Knotens abgelegt werden, dessen RTUs die entsprechenden Tasks ausführen. Prozessoren anderer Knoten sollten auf diesen Speicherbereich nicht zugreifen. Falls möglich, sollte dies auch für die DMA-Transfers der Peripheriegeräte gelten. Globale Speicherbereiche (Shared-Memory) könnten dann in einem Speicherbereich abgelegt werden, der gegenseitigen Beeinflussungen unterworfen ist, beispielsweise auf dem Knoten der GPU. Somit würden beim Zugriff auf den lokalen Speicherbereich geringere Latenzzeiten entstehen.
- Die Realzeittasks, die Peripheriegeräte bedienen, können auf den Knoten ausgeführt werden, auf den diese Peripheriegeräte im Zuge von DMA-Transfers ihre Daten ablegen. Da der Einfluss paralleler DMA-Transfers einiger weniger Peripheriegeräte im Vergleich zum Einfluss der Zugriffe anderer Prozessoren prinzipiell geringer ist, resultieren hieraus kürzere Latenzzeiten. Voraussetzung hierfür ist, dass andere RTUs keine Zugriffe auf diesen Speicherbereich durchführen.

5.3.2 Synchronisierung der Ressourcenbelegungen

Greifen Prozessoren oder Peripheriegeräte auf einen PCI-Bus zu, treten in Abhängigkeit von der jeweiligen Aktivität Latenzzeiten beim Zugriff auf. Die Größenordnung dieser Latenzzeiten wurde bereits in Kapitel 4 bestimmt.

Falls nicht generell mit höheren Latenzzeiten beim Zugriff auf den PCI-Bus gerechnet werden soll, müssen die Zugriffe auf diesen PCI-Bus synchronisiert werden, damit immer nur ein Teilnehmer zu bestimmten Zeiten Zugriff auf den Bus bekommt. Die anderen Geräte sollten während dieser Perioden nicht auf den Bus zugreifen. Durch diese Maßnahme wird die Funktionalität des PCI-Arbiters umgangen, da nun die Zuteilung des PCI-Busses vom RTOS durchgeführt wird.

Beim Zugriff auf einen PCI-Bus müssen zum einen die zugreifenden Prozessoren synchronisiert werden, zum anderen dürfen die angeschlossenen Peripheriegeräte ebenfalls keine Datenübertragungen selbstständig durchführen. In den folgenden Abschnitten werden Prozessoren und Peripheriegeräte getrennt betrachtet:

5.3.2.1 Synchronisation der Prozessoren

Falls mehrere CPUs auf einen PCI-Bus zugreifen möchten, treten bei der Host-to-PCI Bridge Latenzzeiten auf. Diese Bridge kann immer nur einer CPU Zugriff zum Bus gewähren; die anderen Prozessoren müssen warten, bis sie an der Reihe sind.

Wie die GPU daran gehindert werden kann, während bestimmter Zeitabschnitte auf einen PCI-Bus zuzugreifen, wurde bereits in Abschnitt 5.2.3 erläutert. Für die RTUs existieren zwei mögliche Zugriffsszenarien:

- Alle Realzeittasks, die auf einen bestimmten PCI-Bus zugreifen, werden nur von *einer* RTU bearbeitet. In diesem Fall müssen keine weiteren Synchronisierungsmaßnahmen durchgeführt werden, da aus Sicht der Hardware immer nur die eine RTU zu einem gegebenen Zeitpunkt auf den PCI-Bus zugreift. Diese Zugriffe werden nicht durch die Zugriffe anderer RTUs beeinflusst.
- Die Realzeittasks, die auf einen bestimmten PCI-Bus zugreifen, werden von *mehreren* RTUs ausgeführt. Falls möglich, sollte hier der gleichzeitige Zugriff durch geeignete Synchronisierung vermieden werden. Falls dies nicht möglich ist, muss mit verlängerten maximalen Zugriffszeiten gerechnet werden. Im Allgemeinen ist letzteres Szenario der Fall.

5.3.2.2 Synchronisation mit Peripheriegeräten

Peripheriegeräte sollten in Abhängigkeit davon, ob sie für Realzeitaufgaben benötigt werden oder nicht, auf verschiedene PCI-Busse aufgeteilt werden. Da dies nicht immer möglich ist, können durch Löschen der Master Enable Bits die entsprechenden Peripheriegeräte daran gehindert werden, selbstständig Datentransfers zu initiieren (vgl. hierzu auch Abschnitt 5.2.4).

Die Datentransfers der für Realzeitaufgaben benötigten Peripheriegeräte lassen sich in zwei Gruppen einteilen, die im Folgenden diskutiert werden sollen. Grundsätzliches Ziel der Maßnahmen ist es, dass Zugriffe auf Peripheriegeräte, auf die im hohen Maße von Realzeittasks zugegriffen wird, möglichst geringen parallelen Beeinflussungen ausgesetzt sind.

1. DMA-Transfers werden von Realzeittasks angestoßen

Hierunter fallen alle Peripheriegeräte, deren DMA-Transfers von Software initiiert werden und die *sofort* nach dem Senden des entsprechenden Befehls mit der Datenübertragung beginnen. Beispiel hierfür ist die Datenübertragung eines IDE/ATA-Controllers. Nach einer Leseanforderung werden die Daten in den Hauptspeicher geschrieben. Mit Hilfe eines Interrupts wird der Software mitgeteilt, dass die Daten übertragen wurden.

Falls möglich, sollte ein Zugriff einer Realzeittask auf ein anderes Peripheriegerät, das ebenfalls an den PCI-Bus angeschlossen ist, während des DMA-Transfers vermieden werden. Dies könnte beispielsweise durch geeignete Synchronisation geschehen. Falls dies nicht möglich ist, muss ebenfalls mit den entsprechend verlängerten Zugriffszeiten gerechnet werden.

2. DMA-Transfers erfolgen zu beliebigen Zeitpunkten

Falls ein Peripheriegerät zu einem beliebigen Zeitpunkt einen Datentransfer durchführt, wie beispielsweise ein FireWire- oder ein Ethernet-Controller, wird empfohlen, dieses an einem

separaten PCI-Bus zu betreiben. Falls wenige Konfigurationszugriffe einer Realzeittask auf das Peripheriegerät während des Betriebs erforderlich sind, kann dieses Gerät auch zusammen mit den Peripheriegeräten des GPOS betrieben werden. Dabei müssen die Einflüsse dieser Geräte bei der Abschätzung der maximalen Zugriffszeiten berücksichtigt werden.

5.3.3 Auswahl geeigneter Schedulingverfahren

Das zu verwendende Verfahren hängt im hohen Maße von der Anwendung ab, die mit dem Realzeitsystem bearbeitet werden soll. Tasks können statisch einer CPU zugeordnet werden. Es existieren allerdings auch Scheduler, die in Abhängigkeit von der Auslastung der einzelnen CPUs die Tasks auf die CPUs des Multiprozessorsystems verteilen. Dabei ist ein Wechsel der CPU prinzipiell zu jedem Zeitpunkt möglich.

Letzteres Verfahren bietet den Vorteil, dass hier die Auslastung der RTUs im Allgemeinen größer ist als bei statischer Taskzuordnung. Allerdings ist die Komplexität des Realzeitnachweises hier um einiges höher.

Statische Taskzuordnung ist im Hinblick auf die in dieser Arbeit erlangten Erkenntnisse für den Einsatz in PC-basierten Multiprozessorsystemen besser geeignet. Die Gründe hierfür sind:

- Die Realzeittasks können besser in den Caches der Prozessoren fixiert werden.
- Gegenseitige Beeinflussungen der Realzeittasks bei Zugriffen auf den Hauptspeicher und auf Peripheriegeräte können a priori vermieden werden.
- ISRs und Realzeittasks können einer RTU zugeordnet werden.

Welches Schedulingverfahren zum Zuge kommt hängt letztendlich davon ab, welche Verfahren vom jeweils verwendeten Realzeitbetriebssystem angeboten werden. Schwankungen in der Laufzeit von Realzeitsoftware, die durch die Hardware bedingt werden, können hiermit nicht weiter reduziert werden.

5.3.4 Programmerstellung

Ziel beim Design moderner Prozessoren ist es, die Ausführungszeit von Software so gering wie möglich zu halten. Wie die Untersuchungen in Kapitel 4 gezeigt haben, spielt es eine große Rolle, ob die von der CPU benötigten Daten im Cache vorhanden sind, oder ob diese aus dem relativ langsamen Hauptspeicher geladen werden müssen. Im letzteren Fall verlängert sich die Ausführungszeit von Software erheblich.

Deshalb versuchen die Hersteller von Mikroprozessoren mit Hilfe von *Prefetchmechanismen*, Speicherbereiche, auf die wahrscheinlich in der zukünftigen Programmausführung zugegriffen wird, in den Cache der CPU auf Verdacht zu laden. Dabei ist im Allgemeinen sowohl für Code-, als auch für Datenzugriffe ein eigener Prefetchmechanismus vorhanden. Mit Hilfe der `prefetch`-Assemblerinstruktion kann dem Prozessor zusätzlich ein Hinweis auf zukünftig benötigte Speicherbereiche gegeben werden. Hierbei werden die benötigten Daten parallel

zur Programmausführung geladen; allerdings ist das Laden der entsprechenden Daten aus dem Hauptspeicher für die CPU nicht bindend.

Eine CPU kann dazu gezwungen werden, bestimmte Datenbereiche in ihren Cache zu laden, indem aufsteigend immer ein Datum einer Cacheline in ein Scratch-Register gelesen wird. Diese Methode wird als *Block-Prefetching* bezeichnet und ist die schnellste Methode, Daten aus dem Hauptspeicher in den Cache zu laden. Hierbei "erkennt" eine CPU einen Burst-Transfer und überträgt die Daten mit der maximal möglichen Rate [1]. Auf der anderen Seite können auch bestimmte Bereiche aus dem Cache wieder in den Hauptspeicher mit einem Burst-Transfer zurückgeschrieben werden. Hierzu muss die `movntq`-Assemblerinstruktion verwendet werden, die unter Umgehung der Caches ein Datum direkt in den Hauptspeicher schreibt.

Falls in einer Realzeitapplikation auf große Datenbereiche zugegriffen werden muss, was beispielsweise bei (Video-)Streaming-Applikationen der Fall ist, wird deshalb empfohlen, diese Daten blockweise zu verarbeiten. Diese Art der Datenverarbeitung ist in [1] beschrieben und ist für die Prozessoren der x86-Architektur die schnellst mögliche. Hierbei wird die Datenverarbeitung in die folgenden Schritte eingeteilt:

1. Die benötigten Datenblöcke werden in den Cache der CPU geladen. Dies geschieht unter Verwendung einer Schleife, die jeweils eine Cacheline liest. Der Unroll-Faktor der Schleife sollte dabei mindestens 2 betragen.

2. Die Daten werden verarbeitet und in einem Bereich des Caches abgelegt, der keine ungesicherten Daten enthält. Somit können die Daten zügig verarbeitet werden, da ausschließlich auf den Daten im Cache gearbeitet wird und somit Datentransfers anderer Prozessoren und Peripheriegeräte keinen Einfluss auf die Laufzeit haben.

3. Nachdem die Rechenoperationen abgeschlossen wurden, können die Daten in den Hauptspeicher zurückgeschrieben werden. Hierzu werden die Daten vom Cache in ein Prozessorregister geladen und anschließend unter Verwendung der `movntq`-Instruktion im Hauptspeicher abgelegt.

Für die Bestimmung der Ausführungszeit von Realzeitsoftware bedeutet dies, dass in Abhängigkeit davon, ob von der CPU ein Speicher-Burst durchgeführt wird oder nicht, mit unterschiedlichen Parametern gerechnet werden muss. In den Beispielen aus Kapitel 4 wurden Speicher-Bursts vermieden, indem die Cachelines nicht aufsteigend adressiert wurden, sondern ausschließlich Cachelines des selben Sets modifiziert wurden. Um jedoch die maximal benötigte Ausführungszeit nicht zu pessimistisch abzuschätzen, müssen die Parameter der tatsächlich verwendeten Übertragungsmethode ermittelt werden.

5.4 Bestimmung von Ausführungszeiten und Realzeitnachweis

In Kapitel 4 wurde beschrieben, in welchem Ausmaß sich die Ausführungszeiten von Realzeitsoftware beim Zugriff auf gemeinsam genutzte Betriebsmittel verändern. Hier hängt es vom je-

weiligen Belastungsszenario ab, in welcher Größenordnung sich die vorhandenen Latenzzeiten bewegen. In diesem Abschnitt wird erläutert, wie aus den bekannten maximalen Zugriffszeiten unter bestimmten Belastungsszenarien die WCET von Realzeitsoftware ermittelt werden kann. Die WCET-Bestimmung bildet die Grundlage für die Durchführung eines Realzeitnachweises.

5.4.1 Konfigurationszustände

In den Abschnitten 5.2 und 5.3 wurde erläutert, mit welchen statischen und dynamischen Maßnahmen die gegenseitigen Beeinflussungen bei Zugriffen auf gemeinsam verwendete Ressourcen reduziert werden können. Ziel hierbei ist es, die Zugriffe einer Realzeittask so wenig wie möglich durch parallel stattfindende Datentransfers zu behindern.

In Abhängigkeit von der Anzahl der noch vorhandenen Beeinflussungen lässt sich somit für Zugriffe auf gemeinsam genutzte Betriebsmittel ein *Konfigurationszustand* angeben. Dieser Konfigurationszustand ist abhängig von den jeweiligen Datentransfers, die parallel zu den Zugriffen ausgeführt werden können. Befindet sich die Soft- und Hardware in einem bestimmten Konfigurationszustand, stellen sich für diesen Zustand spezifische maximale Zugriffszeiten ein.

Der Konfigurationszustand kann *statisch* konfiguriert werden, indem beispielsweise alle Peripheriegeräte nur an einem PCI-Bus angeschlossen werden und nur auf bestimmte Speicherbereiche zugreifen. Zusätzlich greifen beispielsweise während des Realzeitbetriebs bestimmte RTUs nicht auf gewisse Speicherbereiche zu. Zur Laufzeit entstehen durch die statische Konfiguration *keine* zusätzlichen Latenzzeiten.

Falls der Konfigurationszustand *dynamisch* geändert wird, ist der Wechsel von einem Zustand zu einem anderen mit gewissen Latenzzeiten verbunden. Falls beispielsweise während der Ausführung eines bestimmten Codeabschnitts die Peripheriegeräte eines PCI-Busses keine Datentransfers anstoßen sollen, wird für die Deaktivierung dieser PCI-Geräte eine gewisse Zeit benötigt. Allerdings resultiert aus dieser Maßnahme ein Konfigurationszustand, in dem beim Zugriff auf den PCI-Bus geringere maximale Zugriffszeiten auftreten. Der Wechsel eines Konfigurationszustandes zur Laufzeit ist folglich sinnvoll, falls eine bestimmte minimale Anzahl aufeinanderfolgender Zugriffe auf ein Betriebsmittel überschritten wird.

Um zu entscheiden, ob ein Konfigurationszustand gewechselt werden soll, muss die Anzahl der durchzuführenden Zugriffe bekannt sein. Hierzu wird der Realzeitcode in *Codesequenzen* eingeteilt. Die Anzahl der aufeinanderfolgend durchgeführten Zugriffe in dieser Codesequenz wird anschließend ermittelt:

- Die Anzahl der aufeinanderfolgend durchgeführten Zugriffe auf den *Hauptspeicher* kann ermittelt werden, indem die Wechselwirkungen zwischen Cache und Hauptspeicher untersucht werden. Insbesondere erhält man diese Zahl aus den in [14] entwickelten Methoden zur Anordnung und Fixierung des Codes der Realzeitapplikationen in den Caches der RTUs.
- Die Anzahl der durchgeführten Zugriffe auf Peripheriegeräte lässt sich beispielsweise mittels Codeannotationen handhaben. Da die Zugriffe auf Peripheriegeräte normalerwei-

se direkt in Software programmiert werden, ist hier die jeweils durchgeführte Anzahl aufeinanderfolgender Zugriffe bei der Programmerstellung bekannt.

Es sei angemerkt, dass eine Codesequenz einen oder mehrere *Basic Blocks* beinhalten kann. Wie Basic Blocks besitzt eine Codesequenz einen Eintrittspunkt und einen Austrittspunkt. Im Gegensatz zu Basic Blocks darf eine Codesequenz jedoch Sprünge und Schleifen enthalten. Die Grenzen einer Codesequenz resultieren aus den Programmabschnitten des Realzeitcodes, in denen aufeinanderfolgende Zugriffe auf den Hauptspeicher oder auf Peripheriegeräte durchgeführt werden.

Sobald die Anzahl der jeweils durchgeführten Zugriffe bekannt ist, muss beim dynamischen Wechsel des Konfigurationszustandes entschieden werden, ob hier ein Wechsel sinnvoll ist. Diese Entscheidung hängt von den folgenden Aspekten ab:

- Falls alle Realzeitbedingungen eingehalten werden können, besteht kein Grund, den Konfigurationszustand zu ändern.
- Falls möglichst geringe Schwankungen in der Ausführungszeit gewünscht werden, sollte eine Änderung des Konfigurationszustandes in Betracht gezogen werden. Insbesondere beim Zugriff über einen PCI-Bus resultieren aus der Deaktivierung der PCI-Geräte deutlich geringere Differenzen zwischen maximaler und minimaler Zugriffszeit.
- Weiterhin hängt es von der Anzahl der durchgeführten Zugriffe ab, ob der Konfigurationszustand gewechselt werden sollte. Falls viele Zugriffe durchgeführt werden müssen, resultieren hieraus ggf. erheblich verringerte maximale Zugriffszeiten. Falls nur wenige Zugriffe durchgeführt werden müssen, ist ein Wechsel des Konfigurationszustandes im Allgemeinen nicht sinnvoll.

Im letzteren Aspekt ist ein Wechsel von einem Konfigurationszustand A zu einem Zustand B insbesondere dann sinnvoll, wenn hieraus eine geringere Ausführungszeit der Realzeitsoftware resultiert. Insbesondere muss für eine bestimmte Anzahl x durchgeführter Zugriffe auf eine Ressource folgende Gleichung gelten, wobei $t_{WCET,CS_{AB}}$ die Zeit beschreibt, die maximal für den Wechsel des Konfigurationszustandes benötigt wird:

$$t_{WCAT,A}(x) - t_{WCAT,B}(x) > t_{WCET,CS_{AB}}; \qquad x \in \mathbb{N} \tag{5.1}$$

5.4.2 Bestimmung maximaler Ausführungszeiten

Soll die WCET einer bestimmten Codesequenz ermittelt werden, müssen die im Folgenden aufgeführten Parameter bekannt sein:

- Die Anzahl a_m der Hauptspeicherzugriffe, die während der Ausführung des Codeabschnitts erfolgen.
- Die Anzahl a_p der durchgeführten Peripheriezugriffe.
- Der Konfigurationszustand S der Hardware, der jeweils während der Zugriffe vorliegt.

- Die WCET der Codesequenz c_{WCET}, falls keine Zugriffe auf den Hauptspeicher oder auf Peripheriegeräte während der Ausführung des Codes erfolgen. Insbesondere werden hier alle Aspekte der Semantik des Codes und die Einflüsse der Pipelines und der Sprungvorhersage berücksichtigt. Verfügbare Methoden der WCET-Analyse können herangezogen werden, um diesen Wert zu ermitteln.

Insbesondere kann c_{WCET} mit Hilfe von Simulationen ermittelt werden, wobei hier die Hardwareeigenschaften einer CPU nachgebildet werden. Alternativ kann auch dieser Wert mittels Messung bestimmt werden: Die Untersuchungen in Abschnitt 4.2 haben gezeigt, dass Code, der sich in den Caches der Prozessoren befindet, nur geringen Laufzeitschwankungen unterworfen ist. Somit kann c_{WCET} mit relativ geringem Aufwand ermittelt werden. Um hier die Ausführungszeiten *ohne* Zugriffe auf den Hauptspeicher zu bestimmen, müssen sich Code und Daten der Codesequenz vollständig im Cache befinden. Zugriffe auf Peripheriegeräte dürfen ebenfalls nicht erfolgen. Diese müssen durch leere Instruktionen ersetzt werden. Hierzu sollte bei der Programmerstellung ein geeigneter Betriebsmodus vorgesehen werden.

Für einen Konfigurationszustand S müssen die jeweiligen Parameter der Hardware bekannt sein, die mit den in Kapitel 4 beschriebenen Methoden ermittelt werden sollten. Insbesondere sind dies die Beeinflussungen im Chipsatz (t_{CS}, m_{CS}) und die Beeinflussungen durch Peripheriegeräte (τ_{PCI},μ_{PCI}), die an dem gleichen PCI-Bus angeschlossen sind, auf dem der jeweilige Peripheriezugriff erfolgt (vgl. Seite 81). Zusätzlich sollte der Parameter P bekannt sein, der die Latenzzeiten des jeweiligen Peripheriegeräts für einen bestimmten Zugriff beschreibt (vgl. Seite 88). Insbesondere wird dieser Parameter benötigt, wenn sich die Zugriffszeiten auf verschiedene Register unterschiedlich verhalten.

Für Zugriffe in *SMP-Systemen* und für *lokale* Zugriffe in *NUMA-Systemen* sind die Beeinflussungen t_{CS} und m_{CS} äquivalent zu den in Abschnitt 4.3.1 in Gleichung 4.8 definierten Parametern $t_{WCAT,1}$ und m_{WCAT}:

$$t_{CS} = t_{WCAT,1} \tag{5.2}$$

$$m_{CS} = m_{WCAT} \tag{5.3}$$

Bei Zugriffen auf *entfernte* Knoten eines *NUMA-Systems* setzen sich die Beeinflussungen im Chipsatz aus den Parametern t_U, τ_l, τ_n und τ_r bzw. aus den Parametern m_U, μ_l, μ_n und μ_r zusammen — vergleiche hierzu auch Gleichung 4.10 auf Seite 60. Hinzu kommen noch die Korrelationsfaktoren k_{lnr} und l_{lnr}. Analog zu den Gleichungen 4.11 und 4.12 gilt deshalb:

$$t_{CS} = t_U \tau_l \tau_n \tau_r k_{lnr} \tag{5.4}$$

$$m_{CS} = m_U \mu_l \mu_n \mu_r l_{lnr} \tag{5.5}$$

Die WCET t_{WCET} einer gegebenen Codesequenz setzt sich aus der Zeit c_{WCET} und den Zeiten t_m und t_p zusammen, die für Hauptspeicher- und Peripheriezugriffe benötigt werden:

$$t_{WCET} = c_{WCET} + t_m + t_p \tag{5.6}$$

In Abhängigkeit von der Architektur des Multiprozessorsystems gilt hierbei für t_m und t_p:

$$t_m = t_{CS,m,S} + m_{CS,m,S}(a_m - 1) \quad (5.7)$$

$$t_p = Pt_{CS,p,S}\tau_{PCI,p,S}k_{CS,PCI} + Pm_{CS,p,S}\mu_{PCI,p,S}l_{CS,PCI}(a_p - 1) \quad (5.8)$$

Hierbei geben die Parameter m und p an, ob es sich um Hauptspeicher- oder um Peripheriezugriffe handelt. S definiert den jeweiligen Konfigurationszustand. Falls bei der Ausführung einer Codesequenz keine Zugriffe auf den Hauptspeicher oder auf Peripheriegeräte erfolgen, muss t_m bzw. t_p in Gleichung 5.6 gleich Null gesetzt werden.

5.4.3 Realzeitnachweis

Sobald die Ausführungszeiten der einzelnen Codesequenzen bekannt sind, kann die maximale Ausführungszeit bzw. die maximale Rechenzeit der Realzeittasks ermittelt werden. Dabei ist zu beachten, dass der Wechsel eines Konfigurationszustandes auch eine eigene Codesequenz ist.

Die Ausführungszeiten der Realzeittasks ergeben sich aus den WCETs der einzelnen Codesequenzen, die in Abhängigkeit von der Semantik der Realzeitapplikation die maximalen Rechenzeiten liefern. Mit dem gewählten Schedulingverfahren und dem verwendeten Ressourcenzuteilungsprotokoll kann anschließend ein Realzeitnachweis geführt werden. Wie dieser durchzuführen ist, wird beispielsweise in [32] beschrieben.

5.5 Zusammenfassung

In diesem Kapitel wird erläutert, wie maximale Zugriffszeiten auf den Hauptspeicher und auf Peripheriegeräte in Realzeitsystemen vermindert werden können. Ziel ist es, gegenseitige Beeinflussungen beim Zugriff auf gemeinsam genutzte Betriebsmittel so weit als möglich zu vermeiden. Die WCATs müssen anschließend für das verbleibende Belastungsszenario ermittelt werden. In Verbindung mit der maximalen Ausführungszeit des betrachteten Codeabschnitts ohne Zugriffe auf gemeinsam genutzte Betriebsmittel resultiert die WCET dieser Codesequenz.

Vorteil dieses Verfahrens ist, dass die WCET nicht durch Messung der Ausführungszeit bestimmt wird. Statt dessen beschränken sich die Untersuchungen auf die Bestimmung der Ausführungszeiten eines oder mehrerer *Zugriffe* auf gemeinsam genutzte Betriebsmittel. Die WCET des Codes ohne vorhandene Ressourcenzugriffe kann mit anderen Verfahren bestimmt werden.

Falls der betrachtete Code keine Zugriffe auf außerhalb der CPU liegende Ressourcen durchführt, ergeben sich sehr geringe Laufzeitunterschiede, so dass diese Zeiten ebenfalls mittels Messung bestimmt werden können. Bei der WCET-Bestimmung resultiert aus dem gewählten Verfahren eine genauere Abschätzung der maximalen Ausführungszeit im Vergleich zu der Methode, die Ausführungszeit des gesamten Codeabschnitts ausschließlich mittels Messung zu bestimmen. Grund hierfür ist, dass bei der Messung der Ausführungszeit des Codeabschnitts — inklusive Hauptspeicher und Peripheriezugriffe — nicht bei jedem Zugriff die Zugriffszeit

um das maximal mögliche Maß verlängert wird. Die Messungen müssten hier unverhältnismäßig oft wiederholt werden, was praktisch nicht durchführbar ist. Bei der Durchführung weniger Messungen wird die WCET zu optimistisch abgeschätzt.

Gegenseitige Beeinflussungen lassen sich vermeiden, indem möglichst jede CPU auf ihre eigenen Betriebsmittel zugreift. Insbesondere sind dies die lokal vorhandenen Caches; bei NUMA-Systemen kommen noch lokaler Hauptspeicher und lokale Peripheriegeräte hinzu. Einige Maßnahmen können statisch vorgenommen werden und verursachen zur Laufzeit keine zusätzlichen Kosten. Für dynamische Konfigurationsmaßnahmen wird eine gewisse Zeit benötigt. Hier entscheidet die Anzahl der durchzuführenden Zugriffe, sowie das vorhandene Szenario der Realzeittasks, ob eine Anwendung der jeweiligen Maßnahme sinnvoll ist.

6 Anwendungsbeispiele

Im folgenden Kapitel werden Anwendungsbeispiele beschrieben, die die in dieser Arbeit vorgestellten Methoden verdeutlichen sollen. Insbesondere wird mit den in Kapitel 5 vorgeschlagenen Konzepten demonstriert, wie maximale Ausführungszeiten von Realzeitsoftware, die Zugriffe auf gemeinsam genutzte Ressourcen durchführt, verringert werden können. Zum Einsatz kommen hierbei ein Dual-Athlon und ein Quad-Opteron Multiprozessorsystem. Als Realzeitbetriebssystem wird Linux mit der Erweiterung RTAI (Real-Time Application Interface) verwendet, das um die in Kapitel 5 beschriebene Funktionalität ergänzt wurde.

6.1 Video-Streaming

Zunächst soll der Einfluss parallel arbeitender Prozessoren und Peripheriegeräte auf die Hauptspeicherzugriffe einer Realzeittask demonstriert werden. Hierzu wird auf dem Dual-Athlon System ein Kamerabild, das beispielsweise von einer FireWire-Kamera aufgenommen wurde, vom YUV422 in das Y-Format umgewandelt. Durch diese Umwandlung werden die Farbinformationen des Bildes entfernt. Das resultierende Grauwertbild kann anschließend für weitere Bildverarbeitungsoperationen verwendet werden.

In diesem Beispiel soll nur der Umwandlungsprozess betrachtet werden. Für andere Bildverarbeitungsoperationen resultieren andere c_{WCET}, die verwendete Methodik ändert sich dadurch nicht.

Während des Lese- und Konvertierungsvorgangs kann davon ausgegangen werden, dass keine Verdrängungen in den Caches und TLBs auftreten. Dies wird mit den in Abschnitt 5.2 beschriebenen Maßnahmen konfiguriert. Durch Messung mit Hilfe der in Abschnitt 4.3.1 beschriebenen Methodiken werden die relevanten Parameter für einen Burst-Datentransfer[1)] ermittelt. Diese Werte sind in Tabelle 6.1 aufgeführt. Die zugrundeliegenden Messwerte sind in Abbildung 6.1 grafisch dargestellt.

	$t_{WCAT,1}$ [μs]	m_{WCAT} [μs]
keine parallelen Zugriffe	0,66	0,044
CPU-Zugriffe	0,73	0,147
CPU- und Peripheriezugriffe	0,98	0,157

Tabelle 6.1: Einfluss der parallelen Belastung bei Hauptspeicherzugriffen

[1)] Bei den in Abschnitt 4.3.1 ermittelten Parametern greift der betrachtete Prozessor nicht auf aufeinanderfolgende Cachelines zu.

Bei den Messwerten fällt auf, dass die betrachtete Athlon-CPU im ungestörten Fall schnell auf den Hauptspeicher zugreifen kann. Treten jedoch beim Zugriff gegenseitige Beeinflussungen auf, können die Vorteile einer Burst-Übertragung nicht ausgeschöpft werden. Zusätzlich wird bei den in Tabelle 6.1 aufgeführten Parametern die Zeit für den ersten Zugriff und damit der Offset der jeweiligen Geraden etwas erhöht, um diese besser an die gemessenen Werte annähern zu können. Hieraus resultiert eine etwas optimistischere Abschätzung der maximalen Zugriffszeiten.

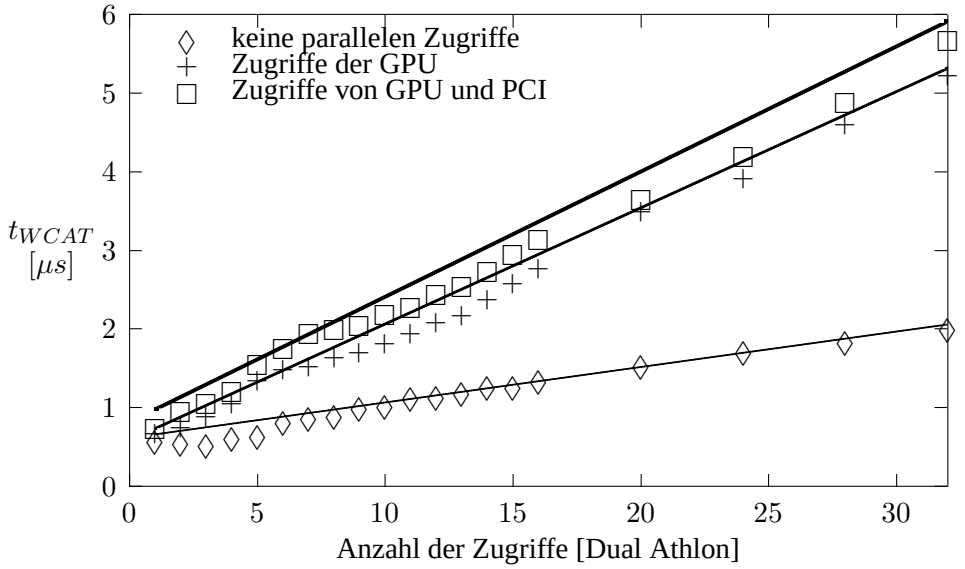


Abbildung 6.1: Zugriffe auf den Hauptspeicher

Es kann davon ausgegangen werden, dass sich der zur Umwandlung der Bilddaten benötigte Code bereits im Instruction-Cache des Prozessors befindet. Zugriffe des Grafikadapters auf den Hauptspeicher treten im Realzeitbetrieb nicht auf.

Das Bild hat eine Breite von 320 Bildpunkten und eine Höhe von 240 Bildpunkten. Das YUV-Bild hat somit eine Größe von 150 KByte. Für das Grauwertbild werden 75 KByte benötigt. Die Bilddaten werden in Blöcken zu je 200 Cachelines à 64 Byte in den Cache eingelesen. Nach der Konvertierung werden die Bilddaten in Blöcken zu je 100 Cachelines in den Hauptspeicher zurückgeschrieben. Die benötigte Zeit c_{WCET} für die Konvertierung eines Blocks beträgt $0{,}73\mu s$. Um ein komplettes Bild zu wandeln, muss die beschriebene Codesequenz zwölf mal durchlaufen werden. Dabei werden insgesamt 3600 Hauptspeicherzugriffe durchgeführt.

Falls die Zugriffe auf den Hauptspeicher nicht durch andere Transfers beeinflusst werden, kann $t_{WCAT,1} = 0{,}66\mu s$ und $m_{WCAT} = 0{,}044\mu s$ angesetzt werden. In diesem Fall erhält man mit den Gleichungen 4.8 und 5.6:

$$\begin{aligned} t_{WCET} &= 12 \cdot (t_{WCAT,1} + 199 \cdot m_{WCAT} + c_{WCET} + t_{WCAT,1} + 99 \cdot m_{WCAT}) \\ &= 12 \cdot (0{,}66\mu s + 199 \cdot 0{,}044\mu s + 0{,}73\mu s + 0{,}66\mu s + 99 \cdot 0{,}044\mu s) = 181{,}94\mu s \end{aligned}$$

Die gemessenen Ausführungszeiten, die für die Wandlung des Bildes benötigt werden, sind in Abbildung 6.2 dargestellt. Die gemessene maximale Ausführungszeit beträgt hier $165{,}82\mu s$.

Die Abweichung von der theoretisch hergeleiteten von der gemessenen Ausführungszeit kann mit $9{,}7\%$ angegeben werden.

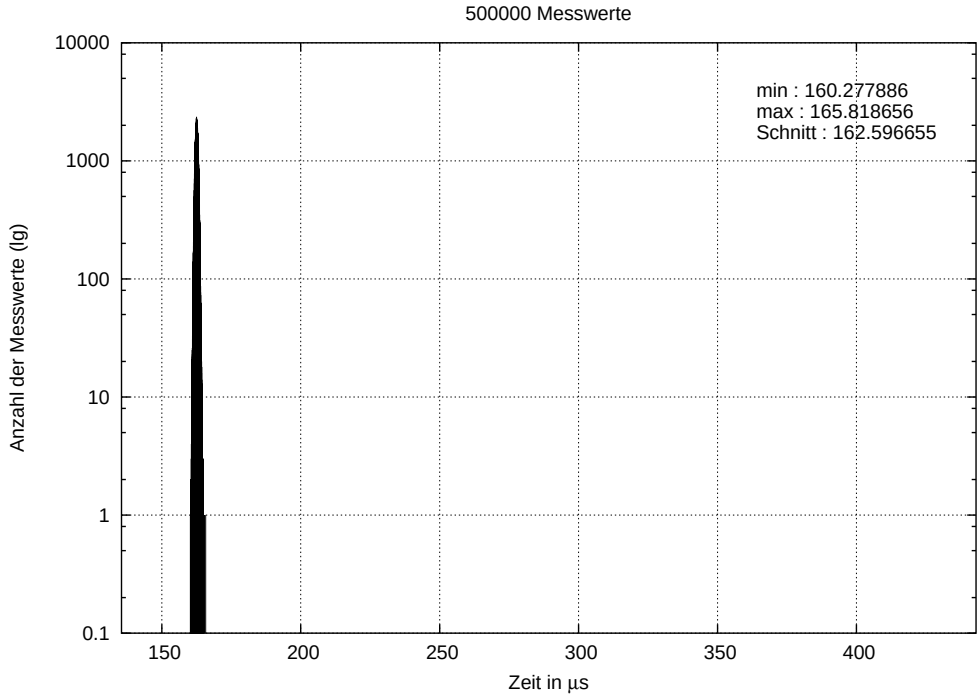


Abbildung 6.2: Bildkonvertierung ohne Beeinflussungen

Im Fall paralleler Zugriffe durch die GPU und damit durch das Standardbetriebssystem wird die maximale Ausführungszeit für den Algorithmus mit $t_{WCET} = 551{,}95\mu s$ ermittelt. Falls zusätzlich Zugriffe durch PCI-Geräte auf den Hauptspeicher erfolgen, resultiert hieraus eine maximal mögliche Zugriffszeit von $t_{WCET} = 593{,}71\mu s$. Die Ermittlung dieser Ausführungszeiten anhand von Messungen ist nur schwer möglich: Da während der Messung der Ausführungszeit eines Konvertierungsvorgangs insgesamt 3600 Zugriffe von der Realzeittask auf den Hauptspeicher erfolgen, ist es sehr unwahrscheinlich, dass alle diese 3600 Zugriffe um das jeweils maximal mögliche Maß verzögert ausgeführt werden. Trotzdem kann dies nicht vollständig ausgeschlossen werden. Wird dennoch die Ausführungszeit des entsprechenden Codeabschnitts unter paralleler Last mittels Messungen bestimmt, resultiert hieraus eine Unterabschätzung der Ausführungszeiten.

In Abbildung 6.3 sind die gemessenen Ausführungszeiten des Konvertierungsalgorithmus bei paralleler Belastung durch das GPOS aufgetragen. Während der Messungen wird auf dem GPOS ein Übersetzungsvorgang und ein Bildverarbeitungsprogramm ausgeführt. Zusätzlich erfolgen Festplatten- und Netzzugriffe. Ziel dieser Maßnahme ist es, Last zu erzeugen, wie sie im normalen Realzeitbetrieb auch auftreten kann.

Die gemessene maximale Ausführungszeit beträgt hier $417{,}42\mu s$. Mit der errechneten maximalen Ausführungszeit von $t_{WCET} = 593{,}71\mu s$ würde hieraus eine Überabschätzung von $42{,}2\%$ folgen, falls dieser gemessene Wert der tatsächliche Worst-Case wäre. Da aber nicht garantiert werden kann, dass unter bestimmten Konstellationen nicht noch höhere Ausführungszeiten auftreten können, darf dieser gemessene Wert nur in weichen Realzeitsystemen verwendet werden. Im Gegensatz hierzu kann der errechnete Wert als obere Grenze der Ausführungszeit betrachtet werden.

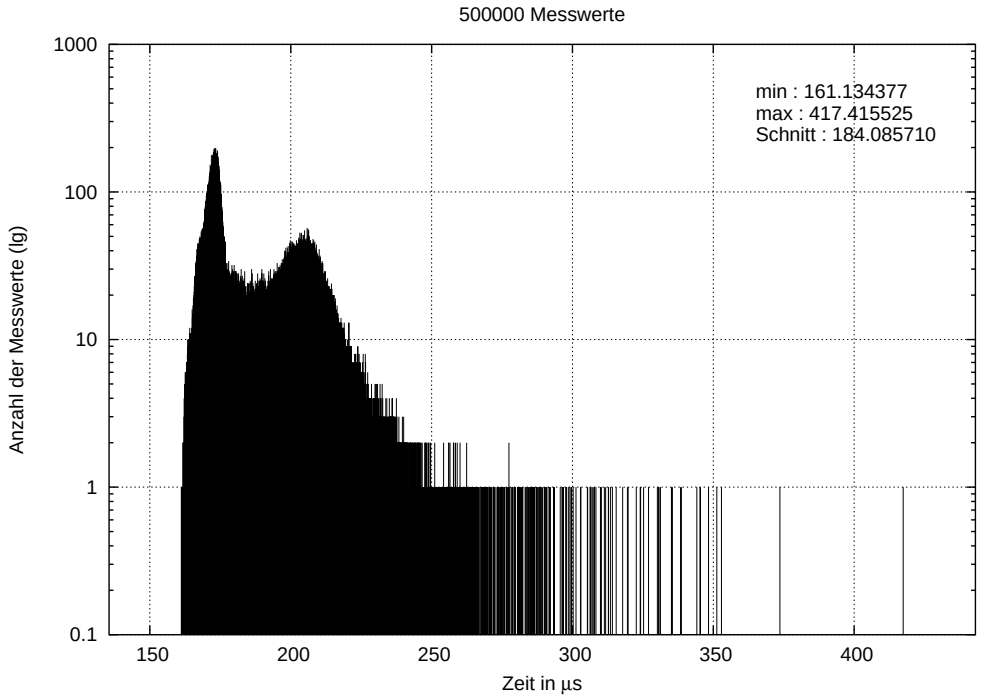


Abbildung 6.3: Gemessene Ausführungszeiten bei der Bildkonvertierung mit Beeinflussungen durch die GPU und durch PCI-Geräte

Bei diesem Messergebnis fällt zusätzlich auf, dass die durchschnittliche Zugriffszeit nur gering von den gemessenen Ausführungszeiten im ungestörten Fall abweicht. Hier resultiert aufgrund paralleler Datenübertragungen eine Erhöhung der durchschnittlichen Zugriffszeit um ca. $20\mu s$. Die gemessenen maximalen Ausführungszeiten werden jedoch bei parallelen Datenübertragungen um einen Faktor von $2{,}5$ verlängert.

Falls sich während des Realzeitbetriebs die Datentransfers der GPU und der Peripheriegeräte nicht auf Hauptspeicherzugriffe der RTU auswirken sollen, können diese daran gehindert werden, Übertragungen durchzuführen. Die maximal benötigte Zeit zum Wechsel in diesen Konfigurationszustand beträgt ca. $15\mu s$. Zusätzlich können in der Realzeitapplikation eventuell Schreib- und Lesezugriffe auf den Hauptspeicher gebündelt ausgeführt werden, indem beispielsweise die Daten des Vorgängerbildes unmittelbar vor dem Einlesen des nächsten Bildes in den Hauptspeicher geschrieben werden. Mit einer Bildfrequenz von 30fps würde hiermit

	Lesezugriffe		Schreibzugriffe	
parallele Last	$t_{WCAT,1,l}$	$m_{WCAT,l}$	$t_{WCAT,1,s}$	$m_{WCAT,s}$
keine parallelen Zugriffe	$0{,}71\mu s$	$0{,}63\mu s$	$0{,}47\mu s$	$0{,}44\mu s$
Peripheriegeräte aktiv	$1{,}50\mu s$	$0{,}96\mu s$	$1{,}27\mu s$	$0{,}96\mu s$

Tabelle 6.2: Benötigte Zugriffszeiten auf einen AD/DA-Wandler

die Performanz des GPOS um ca. $0{,}5\%$ verringert werden. Voraussetzung wäre natürlich, dass keine weiteren Realzeittasks das GPOS aussetzen.

6.2 AD/DA-Wandlung

In einem weiteren Beispiel soll eine AD/DA-Wandlerkarte an ein Dual-Athlon Multiprozessorsystem angeschlossen werden. Hierbei kommt eine National Instruments NI PCI-1640E Wandlerkarte zum Einsatz. Diese Karte bietet 16 analoge Eingangskanäle, 2 analoge Ausgangskanäle und 8 digitale Ein-/Ausgabekanäle.

In dem betrachteten Multiprozessorsystem werden die North- und die South-Bridge über einen PCI-66 Bus miteinander verbunden. Ein PCI-33 Bus ist über eine PCI-to-PCI Bridge an den PCI-66 Bus angebunden.

Um ein möglichst schnelles Deaktivieren der vom GPOS verwendeten PCI-Geräte zu ermöglichen, werden diese alle an den PCI-33 Bus angeschlossen. Die vom Realzeitsystem verwendete Wandlerkarte wird dagegen an den PCI-66 Bus angebunden. Zusätzlich wird darauf geachtet, dass die Interruptvektoren der Wandlerkarte nicht mit einem anderen Gerät gemeinsam genutzt werden. Neben der Wandlerkarte befindet sich zusätzlich noch ein IDE-Controller an dem PCI-66 Bus, der auch für Realzeitapplikationen genutzt werden könnte. Über den PCI-33 Bus werden insbesondere ein weiterer IDE-Controller und ein Netzwerkadapter vom GPOS bedient.

Im Folgenden soll der Fall betrachtet werden, dass ein analoger Wert eingelesen wird. Hierbei führt der verwendete Treiber zunächst vier Schreibzugriffe auf die Wandlerkarte durch, um die AD-Wandlung anzustoßen. Die für die Wandlung benötigte Zeit beträgt maximal $t_W = 2\mu s$. Während der Wandlung überprüft der Treiber mittels Polling, ob die AD-Wandlung abgeschlossen wurde. Für jeden Pollvorgang wird ein Schreib- und ein Lesezugriff auf die Karte durchgeführt. Zwischen den einzelnen Abfragen wird nicht gewartet. Liegt der gewandelte Wert vor, erfolgt abschließend ein Lesezugriff.

Die relevanten Zugriffszeiten sind in Tabelle 6.2 aufgeführt. Aufgrund vorhandener Zwischenspeicher in der Host-Bridge kann die Ausführungszeit für Schreibzugriffe nur indirekt über die Messung der Ausführungszeit eines Schreib- und eines Lesezugriffs bestimmt werden. Hierbei erzwingt der Lesezugriff, dass der zwischengespeicherte Schreibzugriff auf das Peripheriegerät ausgeführt wird. Zusätzlich wird überprüft, ob die Schreib- und Lesezugriffe auf unterschiedliche Register der Wandlerkarte jeweils gleich lange dauern. Letzteres ist bei der betrachteten Karte der Fall.

Falls sich der benötigte Code komplett im Cache der RTU befindet, gilt für die Ausführungszeit $c_{WCET} = 0{,}11\mu s$. Da c_{WCET} im Vergleich zu den benötigten Zugriffszeiten gering ist, kann davon ausgegangen werden, dass die Peripheriezugriffe aufeinanderfolgend ausgeführt werden. Für die WCET des betrachteten Codeabschnitts gilt:

$$\begin{aligned} t_{WCET} &= c_{WCET} + t_{WCAT,1,s} + 3 \cdot m_{WCAT,s} \\ &\quad + t_W + m_{WCAT,s} + m_{WCAT,l} + \\ &\quad + m_{WCAT,l} \end{aligned}$$

Hierbei erfolgen zunächst die vier Konfigurationszugriffe. Anschließend wird die Wandlung durchgeführt, die maximal die Zeit t_W benötigt. Parallel hierzu wird permanent der Zustand der Wandlung durch die CPU abgefragt; pro Abfrage erfolgt dabei ein Schreib- und ein Lesezugriff. Im Worst-Case ist bei einer Zustandsabfrage die AD-Wandlung gerade noch nicht abgeschlossen. In jedem Fall erfolgt jedoch mindestens eine Zustandsabfrage. Abschließend wird der gewandelte Wert eingelesen.

Die Peripheriezugriffe der GPU können mit vernachlässigbarer Latenzzeit vermieden werden. Hierbei muss lediglich eine globale Variable gesetzt werden. Da jedoch für die Deaktivierung der Peripheriegeräte eine gewisse Zeit benötigt wird und die Durchführung dieser Maßnahme nicht immer gewünscht ist, wird im Folgenden die Ausführungszeit mit aktiven Peripheriegeräten mit der Ausführungszeit ohne parallelen Beeinflussungen verglichen.

Im ungestörten Fall ohne paralleler Zugriffe der GPU oder von Peripheriegeräten ist $t_{WCET} = 5{,}60\mu s$. Die gemessene Ausführungszeit liegt hier bei $4{,}84\mu s$ mit einer durchschnittlichen Zugriffszeit von $3{,}84\mu s$. Abbildung 6.4 illustriert die gemessenen Zugriffszeiten. Die beiden Häufungen resultieren aus verschiedenen Wandlungszeiten des AD/DA-Wandlers.

Werden parallel zu den Zugriffen die Peripheriegeräte nicht deaktiviert, so resultiert hieraus die maximale Ausführungszeit von $t_{WCET} = 9{,}14\mu s$. Im Vergleich hierzu beträgt die gemessene maximale Ausführungszeit im Realzeitbetrieb $7{,}57\mu s$. Die gemessenen Zugriffszeiten sind in Abbildung 6.5 dargestellt.

Grund für die Abweichung der berechneten von den tatsächlichen Ausführungszeiten ist, dass die Zwischenspeicher parallel zur Ausführung des Codes geleert werden können. Zusätzlich resultiert aufgrund der benötigten Ausführungszeit für eine Zustandsabfrage eine Überabschätzung der WCET: Da bei einer Zustandsabfrage das Ergebnis der AD-Wandlung gerade noch nicht vorliegen kann, ist hier eine Abweichung der berechneten von der tatsächlichen Ausführungszeit bis zur Höhe der Ausführungszeit für eine Zustandsabfrage möglich. Im ungestörten Fall beträgt diese $1{,}07\mu s$, mit paralleler Last durch PCI-Geräte $1{,}92\mu s$. Die gemessenen Abweichungen betragen dabei $0{,}76\mu s$ für den ungestörten Fall und $1{,}57\mu s$, falls parallele Last vorhanden ist.

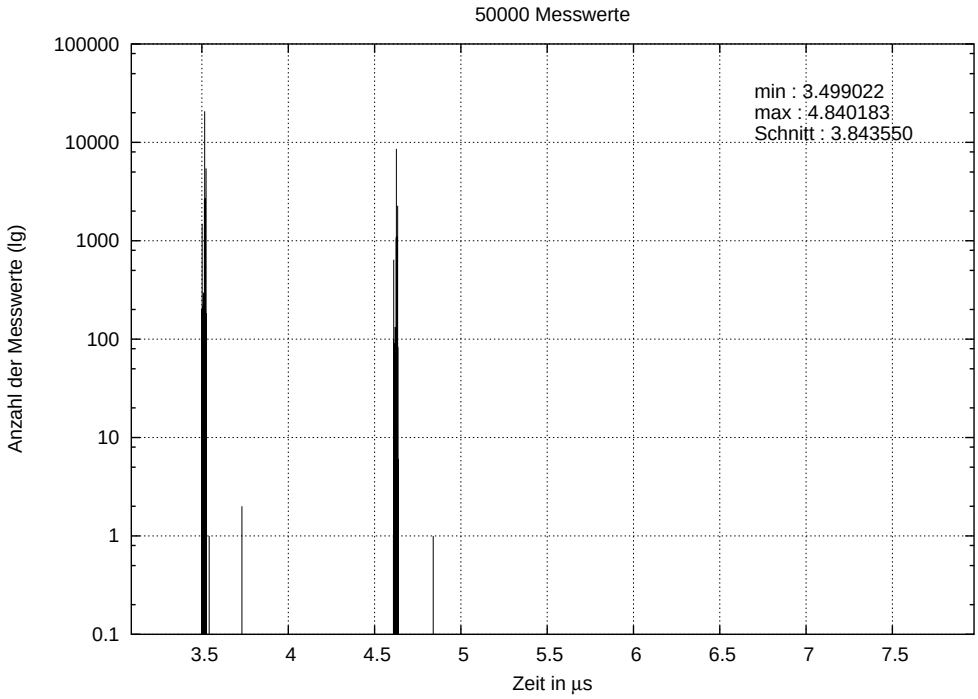


Abbildung 6.4: AD-Wandlung ohne paralleler Last

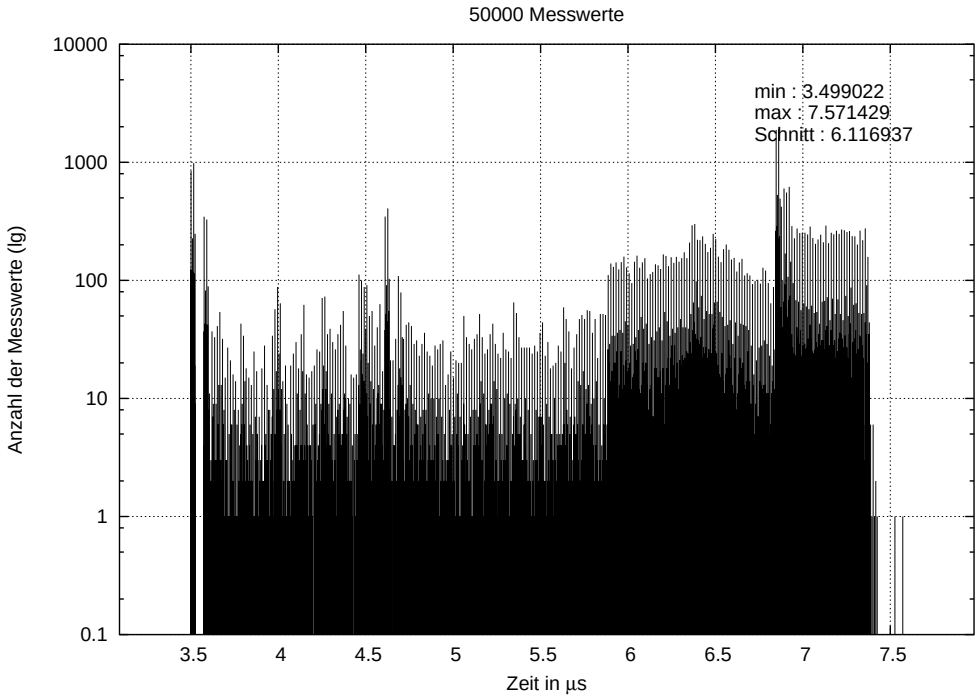


Abbildung 6.5: AD-Wandlung mit PCI-Aktivität

6.3 Speicherung von Messdaten

In einer weiteren Anwendung sollen auf dem Quad-Opteron Multiprozessorsystem Daten, wie beispielsweise Messwerte, auf einen Massenspeicher geschrieben werden. Hierbei wird ein IDE-Controller im PIO-Modus adressiert.

Das Multiprozessorsystem besitzt einen PCI-Bus, an dem nur die vom GPOS verwendete Grafikkarte angeschlossen ist. Weitere Steckplätze sind an diesem Bus nicht vorhanden. Zusätzlich existieren ein PCI-X-Bus mit drei Steckplätzen und zwei PCI-X-Busse mit jeweils einem Steckplatz. Über den Chipsatz sind an alle drei PCI-X-Busse keine weiteren Peripheriegeräte angebunden. Um gegenseitige Beeinflussungen durch andere Datentransfers beim Zugriff auf den Massenspeicher zu vermeiden, wird dieser an einen PCI-X-Bus mit nur einem Steckplatz angeschlossen. Da es sich bei dem zugehörenden Controller um ein PCI-Gerät handelt, wird der PCI-X-Bus und der Controller im zu PCI kompatiblen Modus mit 66 MHz betrieben.

Sollen auf diesem Multiprozessorsystem die in den vorhergehenden Abschnitten beschriebene Bildverarbeitung und die AD/DA-Wandlung ebenfalls ausgeführt werden, so muss die Wandlerkarte an dem zweiten PCI-X-Bus mit einem Slot angeschlossen werden. Grund hierfür ist, dass ebenfalls mehrere Zugriffe von einer Realzeittask auf dieses Gerät erfolgen müssen. Eine FireWire-Karte, über die die Bilder für die Bildverarbeitung geliefert werden, könnte dagegen an dem verbleibenden PCI-X-Bus, auch zusammen mit Geräten des GPOS, betrieben werden. Dies ist sinnvoll, da zum Betrieb von FireWire nur wenige Hardwarezugriffe (zwei Lese- und zwei Schreibzugriffe pro Frame) erfolgen müssen.

Die zugehörige Realzeittask zur Ansteuerung des IDE-Controllers wird auf dem Knoten betrieben, an dem sich auch der PCI-X-Bus befindet. Vor der Übertragung der Daten müssen zunächst einige Konfigurationen durchgeführt werden. Hierbei werden maximal 10 Lese- und 9 Schreibzugriffe ausgeführt. Die Untersuchungen ergeben, dass für alle Lesezugriffe eine maximale Zugriffszeit von $t_l = 1{,}08\mu s$ und für alle Schreibzugriffe eine Zugriffszeit von $t_s = 0{,}72\mu s$ angesetzt werden kann. Dabei werden zwischen diesen Zugriffen teilweise Wartezyklen durchlaufen, so dass hier zur Vereinfachung bei der Bestimmung der WCET von einzelnen Zugriffen ausgegangen wird. Sonstige Laufzeitunterschiede für Zugriffe auf unterschiedliche Ports können nicht festgestellt werden.

Die Daten selbst werden wortweise übertragen. Um einen kompletten Block zu beschreiben, müssen 256 Zugriffe durchgeführt werden. Für den ersten Zugriff wird hier ebenfalls eine maximale Zugriffszeit von $t_{WCAT,1} = 0{,}72\mu s$ gemessen. Für alle folgenden Zugriffe wird $m_{WCAT} = 0{,}29\mu s$ ermittelt. Die maximale Laufzeit des Codes ohne Speicher und Peripheriezugriffe kann mit $c_{WCET} = 13{,}67\mu s$ angegeben werden. Die zu schreibenden Daten befinden sich im Cache der CPU, so dass keine Zugriffe auf den Hauptspeicher während der Datenübertragung erfolgen.

Die maximale Ausführungszeit eines Schreibvorgangs lässt sich folglich angeben mit

$$\begin{aligned} t_{WCET} &= c_{WCET} + 10 \cdot t_l + 9 \cdot t_s + t_{WCAT,1} + 255 \cdot m_{WCAT} \\ &= 13{,}67\mu s + 10 \cdot 1{,}08\mu s + 9 \cdot 0{,}72\mu s + 0{,}72\mu s + 255 \cdot 0{,}29\mu s = 105{,}62\mu s. \end{aligned}$$

Die gemessene maximale Ausführungszeit wird mit $102{,}13\mu s$ bestimmt. Die durchschnittliche Zugriffszeit beträgt $100{,}03\mu s$. Dabei wird während der Messungen Last von anderen Peripheriegeräten erzeugt. Das Messergebnis ist in Abbildung 6.6 grafisch dargestellt. Wird dagegen der DMA-Controller an dem PCI-X-Bus mit drei Steckplätzen zusammen mit weiteren Peripheriegeräten betrieben, sind aufgrund paralleler Beeinflussungen und aufgrund eines verringerten Bustaktes erheblich höhere maximale Ausführungszeiten dieses Codeabschnitts zu erwarten. Wird der IDE-Controller zusammen mit einer Netzwerkkarte und einem SCSI-Controller an diesem PCI-X-Bus verwendet, folgt mit den Werten aus Tabelle 4.15 auf Seite 83 eine maximale Ausführungszeit von ca. einer Millisekunde.

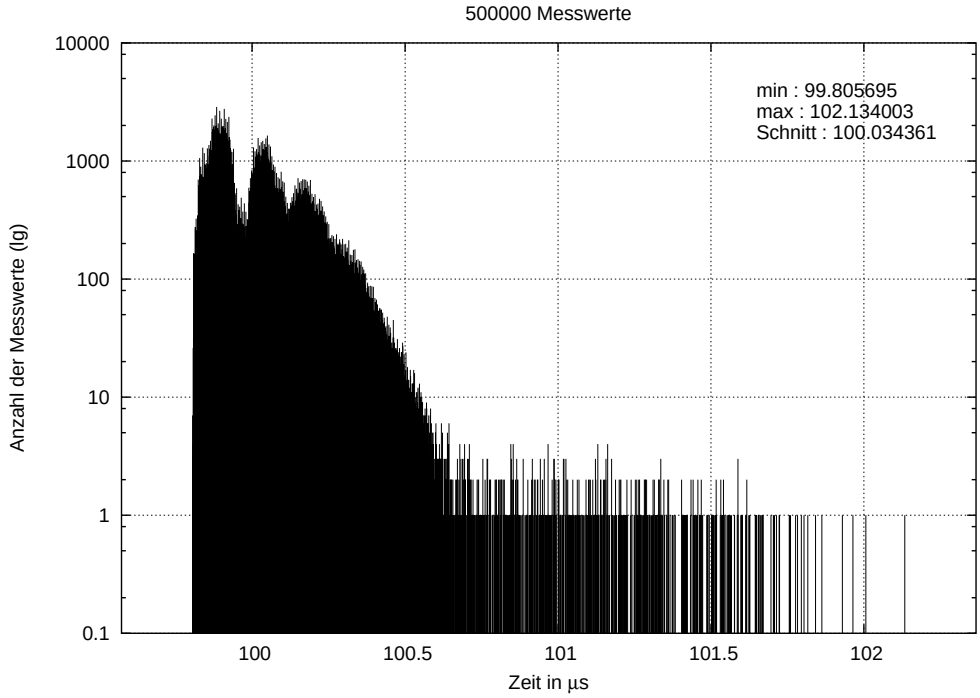


Abbildung 6.6: Übertragung eines Blocks im PIO-Modus

7 Zusammenfassung

Im Vergleich zu Einprozessormaschinen bieten Multiprozessorsysteme den Vorteil, dass die auf den CPUs vorhandenen Caches exklusiv für Realzeitapplikationen verwendbar sind. Sollen auf dem Rechensystem sowohl Standard-, als auch Realzeittasks ausgeführt werden, können diese Tasks auf den CPUs des Multiprozessorsystems verteilt werden: die Standardapplikationen werden von einer CPU ausgeführt und die Realzeittasks werden auf die restlichen CPUs verteilt. Somit können die relativ schlanken Realzeittasks im Cache der Prozessoren gehalten werden und werden nicht beim Aufruf der Standardapplikationen aus diesem verdrängt. Befinden sich Code und Daten im Cache, sind die Ausführungszeiten von Software nur minimalen Laufzeitschwankungen unterworfen.

Beim Zugriff auf den Hauptspeicher oder auf Peripheriegeräte treten jedoch beim Einsatz eines Multiprozessorsystems im Vergleich zu einer Einprozessormaschine ungleich höhere Latenzzeiten auf. Falls mehrere Prozessoren oder Peripheriegeräte gleichzeitig auf eine gemeinsam genutzte Ressource zugreifen, werden die Zugriffszeiten um ein Vielfaches der ungestörten Zugriffszeiten verlängert.

In dieser Arbeit liegt der Schwerpunkt auf der Untersuchung der auftretenden Latenzzeiten bei Zugriffen eines Prozessors auf außerhalb der CPU liegende Ressourcen. Im Allgemeinen sind dies Zugriffe auf den Hauptspeicher oder auf Peripheriegeräte. Hierfür wird zunächst die in dieser Arbeit entwickelte Messmethodik erläutert, die es erlaubt, gezielt definierte Datenströme in einem Multiprozessorsystem zu erzeugen. Diese Messmethodik ermöglicht es, den Einfluss der Datenströme zu bestimmen, die von parallel arbeitenden Prozessoren und Peripheriegeräten initiiert werden. Somit können die möglichen Einflussfaktoren separiert und die Ursache sowie die Größenordnung auftretender Latenzzeiten ermittelt werden.

Anschließend werden die Laufzeiten von Software untersucht, deren Code und Daten sich zum einen vollständig im Cache eines Prozessors befinden, zum anderen aus dem Hauptspeicher geladen werden müssen. Dabei wird nachgewiesen, dass Software, die keine Zugriffe auf den Hauptspeicher ausführt, nur minimalen Laufzeitschwankungen unterworfen ist. Daraus folgt, dass sich die Einflüsse vorhandener Beschleunigungsmechanismen, wie die Sprungvorhersage und die Pipelines, meist vernachlässigen lassen. Müssen jedoch Zugriffe auf den Hauptspeicher durchgeführt werden, verlängert sich die Zugriffszeit im erheblichen Maße. Da die Ausführung von Software parallel zum Nachladen des Codes bzw. der Daten aus dem Hauptspeicher geschieht, hängt die tatsächliche Verlängerung der Laufzeit von der Struktur des verwendeten Codes ab.

In den weiteren Abschnitten werden die Zugriffszeiten auf außerhalb der CPUs liegende Ressourcen für SMP- und NUMA-Systeme mittels Messungen untersucht. Dabei wird erläutert,

welche Messungen durchgeführt werden müssen und wie diese durchzuführen sind. Da die Spezifikationen der verwendeten Komponenten oft unvollständig sind und Multiprozessorsysteme auf PC-Basis an vielen Stellen konfigurierbar sind, ist die Messung der jeweils maximalen Zugriffszeit die einzige Methode, um verwertbare Ergebnisse zu erhalten.

Bei Zugriffen auf den Hauptspeicher und auf Peripheriegeräte ist das Verhalten der auftretenden Zugriffszeiten in Abhängigkeit von der Anzahl aufeinanderfolgend durchgeführter Zugriffe linear. Falls keine parallelen Datenströme vorhanden sind, liegen die Zugriffszeiten im Allgemeinen auf einer Geraden. Falls andere Prozessoren oder Peripheriegeräte parallele Datenübertragungen initiieren, können die jeweiligen Zugriffszeiten mittels einer Geraden approximiert werden. Somit lassen sich zur Abschätzung der maximalen Laufzeit einer bestimmten Anzahl aufeinanderfolgender Ressourcenzugriffe eines Prozessors die Parameter der jeweiligen Geraden verwenden. Hierzu müssen die Laufzeiten einiger weniger aufeinanderfolgender Zugriffe unter der jeweiligen Lastsituation bestimmt werden. Dabei sind so viele Messungen durchzuführen, bis die Gerade zur oberen Abschrankung der Zugriffszeiten ermittelt werden kann. Im realen Betrieb ergeben sich geringere Zugriffszeiten, wobei mit zunehmender Anzahl aufeinanderfolgender Zugriffe die Wahrscheinlichkeit sinkt, bei jedem Zugriff worst-case Bedingungen zu erhalten. Somit ist durch die verwendete Methodik unter allen Umständen eine obere Abschrankung der Zugriffszeiten gewährleistet.

Die Untersuchung der Peripheriebusse ergibt, dass die vorgeschlagenen Methoden übertragbar sind. Wie bei den Beeinflussungen im Chipsatz kann auch bei Peripheriebussen grundsätzlich von fairen Arbitrierungsverfahren ausgegangen werden. Somit lassen sich auch hier Offset und Steigung einer Geraden zur Abschrankung der jeweiligen maximalen Zugriffszeit ermitteln. Beim Zugriff auf ein Peripheriegerät werden durch dieses Gerät weitere Latenzzeiten verursacht. Hier müssen die Latenzzeiten für jedes Peripheriegerät gesondert untersucht werden.

Weiterhin wird in dieser Arbeit eine Softwarearchitektur vorgestellt, die den Umgang mit den ermittelten Latenzzeiten im Realzeitbetrieb beschreibt bzw. ermöglicht. Ziel hierbei ist es, die maximale Ausführungszeit von Realzeitsoftware sicher abzugrenzen, zum anderen aber auch die Leistungsfähigkeit moderner Hardware auszunutzen. Insbesondere können mit Hilfe dieser Softwarearchitektur das Standardbetriebssystem und von diesem bediente Peripheriegeräte zu bestimmten Zeiten daran gehindert werden, Ressourcenzugriffe durchzuführen. Diese Softwarearchitektur bildet damit die Grundlage für die in dieser Arbeit durchgeführten Messungen der jeweiligen Ausführungs- und Zugriffszeiten.

In Abhängigkeit davon, ob bei einem Ressourcenzugriff einer Realzeittask parallele Beeinflussungen vorhanden sein können oder nicht, lassen sich verschiedene Konfigurationszustände definieren, die die jeweils mögliche parallele Last beschreiben. Designziel für den Realzeitbetrieb ist, dass möglichst wenig parallele Beeinflussungen auftreten. Dies kann zum einen statisch durch eine geeignete Konfiguration der Hardware erreicht werden, zum anderen kann ggf. auch zur Laufzeit von einem Konfigurationszustand in einen anderen gewechselt werden. Im letzteren Fall müssen allerdings die Zeitstrafen für diesen Wechsel berücksichtigt werden. Falls bestimmte Beeinflussungen nicht vermieden werden können, muss zur Abschätzung der WCET mit den zugehörigen schlechteren Parametern gerechnet werden. Die WCET resultiert dabei aus der Summe der Zugriffszeiten auf gemeinsam genutzte Ressourcen und der maximalen

Ausführungszeit des Codes, falls keine Zugriffe auf außerhalb der CPU liegende Komponenten erfolgen.

Letztendlich stellt sich die Frage, ob ein PC-basiertes Multiprozessorsystem unter harten Realzeitbedingungen betrieben werden kann oder nicht. Prinzipiell ist solch ein System für den harten Realzeitbetrieb geeignet — die durchgeführten Untersuchungen belegen dies. Da PCs allerdings nicht für dieses Einsatzgebiet konstruiert werden, sind auf alle Fälle umfangreiche Tests notwendig, um die Realzeiteigenschaften einer vorhandenen Hardware zu überprüfen. Insbesondere muss getestet werden, ob sich das zeitliche Verhalten im erwarteten Maß bewegt, oder ob Fehler in Hard- oder Software höhere Ausführungszeiten bedingen. Hier sollte mit den Herstellern der einzelnen Komponenten zusammengearbeitet werden, damit sichergestellt ist, dass nur “ausgereifte” Hardware in ein Realzeitsystem eingebaut wird.

Literaturverzeichnis

[1] AMD, One AMD Place, Sunnyvale, CA 94088, USA: *AMD Athlon Processor x86 Code Optimization Guide*, February 2002.

[2] AMD, One AMD Place, Sunnyvale, CA 94088, USA: *Whitepaper: The AMD-760 MPX Platform for the AMD Athlon MP Processor*, January 2002.

[3] ANDERSON, DON: *FireWire System Architecture*. Addison–Wesley Publishing Company, Reading, Massachusetts, 2nd edition, 1999.

[4] ARNAUD, A. and I. PUAUT: *Towards a Predictable and High Performance Use of Instruction Caches in Hard Real–Time Systems*. In *Proceedings of the work-in-progress session of the 15th Euromicro Conference on Real-Time Systems*, pages 61–64, Porto, Portugal, July 2003.

[5] ATANASSOV, PAVEL, RAIMUND KIRNER, and PETER PUSCHNER: *Using Real Hardware to Create an Accurate Timing Model for Execution-Time Analysis*. In *Proceedings of the IEEE International Workshop on Real-Time Embedded Systems*, December 2001.

[6] BATE, I. and R. REUTEMANN: *Worst-Case Execution Time Analysis for Dynamic Branch Predictors*. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, June 2004.

[7] BODIN, F. and I. PUAUT: *A WCET-Oriented Static Branch Prediction Scheme for Real-Time Systems*. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Balearic Islands, Spain, July 2005.

[8] BOVET, DANIEL P. and MARCO CESATI: *Understanding the Linux Kernel*. O'Reilly, Sebastopol, 2nd edition, 2003.

[9] BUCAR, DEJAN: *Reducing Interrupt Latency using the Cache*. Master's thesis, Royal Institute of Technology, Sweden, 2001.

[10] BUDRUK, RAVI, DON ANDERSON, and TOM SHANLEY: *PCI Express System Architecture*. Addison–Wesley Publishing Company, Boston, 2003.

[11] BURMBERGER, GREGOR: *PC–basierte Systemarchitekturen für zeitkritische technische Prozesse*. Doktorarbeit, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, Februar 2002.

[12] BURNS, ALAN and STEWART EDGAR: *Predicting Computation Time for Advanced Processor Architectures*. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, June 19–21 2000.

[13] BÜLOW VON, A., J. STOHR, G. FÄRBER, P. MÜLLER, and J. B. SCHRAML: *Using the RECOMS Architecture for Controlling a Radio Telescope*. Technical report, Institute for Real–Time Computer Systems, Technische Universität München, May 2004.

[14] BÜLOW VON, ALEXANDER: *Optimale Cache-Nutzung für Realzeitsoftware auf Multiprozessorsystemen*. Doktorarbeit, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, Dezember 2005.

[15] BÜLOW VON, ALEXANDER, JÜRGEN STOHR, and GEORG FÄRBER: *Towards an Efficient Use of Caches in State of the Art Processors for Real–Time Systems*. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems – Work in Progress Session*, Catania, Italy, June 2004.

[16] DIAPM, Dipartimento di Ingegneria Aerospaziale Politecnico di Milano: *A Hard Real Time Support for LINUX*, 2002.

[17] FÄRBER, GEORG: *Prozessrechentechnik*. Springer, Berlin, 3. Auflage, 1994.

[18] GOEBL, MATTHIAS: *Einflüsse der Bussysteme moderner PCs auf das Laufzeitverhalten von Realzeitsoftware*. Diplomarbeit, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, 2003.

[19] GONZALEZ-SCHILLER, CHRISTIAN: *Realzeiteigenschaften von IDE Festplatten*. Diplomarbeit, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, 2004.

[20] GÖTZ, SIEGFRIED: *Implementierung einer alternativen Speicherverwaltung für Realzeitsysteme*. Diplomarbeit, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, 2003.

[21] HAHN, JOOSUN, RHAN HA, SANG LYUL MIN, and JANE W.-S. LIU: *Analysis of Worst-Case DMA Response Time in a Fixed-Priority Bus Arbitration Protocol*. Journal of Real-Time Systems, 23:209–238, 2002.

[22] HECKMANN, REINHOLD, MARC LANGENBACH, STEPHAN THESING, and REINHARD WILHELM: *The Influence of Processor Architecture on the Design and the Results of WCET Tools*. In *Proceedings of the IEEE*, volume 91, July 2003.

[23] HENNESSY, JOHN L. and DAVID A. PATTERSON: *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann Publishers, San Francisco, 3rd edition, 2003.

[24] HEURSCH, ARND C., DIRK GRAMBOW, ALEXANDER HORSTKOTTE, and HELMUT RZEHAK: *Steps Towards a Fully Preemptable Linux Kernel*. In *Proceedings of the 27th IFAC/IFIP/IEEE Workshop on Real-Time Programming, WRTP'03*, Lagow, Poland, May 2003.

[25] HEURSCH, ARND C., ALEXANDER KOENEN, WITOLD JAWORSKI, and HELMUT RZEHAK: *Improving Conditions for Executing Soft Real-Time Tasks Timely in Linux*. In *Proceedings of the 5th Real-Time Linux Workshop*, Valencia, Spain, November 2003.

[26] HOPFNER, THOMAS, JÜRGEN STOHR, WOLFRAM FAUL und GEORG FÄRBER: *RTCPU – Realzeitanwendungen auf Dual-Prozessor PC Architekturen*. it+ti — Informationstechnik und Technische Informatik, 43(6):291, Dezember 2001.

[27] HUANG, TAI-YI, JANE W.-S. LIU, and DAVID HULL: *A Method for Bounding the Effect of DMA I/O Interference on Program Execution Time*. In *Proceedings Real-Time Systems Symposium*, pages 275–285, Washington, D. C., USA, December 1996.

[28] INTEL, P.O. Box 7641, Mr. Prospect IL 60056-7641: *IA-32 Intel Architecture, Software Developer's Manual, Volume 1-3*, 2001.

[29] LIEDTKE, JOCHEN, HERMANN HÄRTIG, and MICHAEL HOHMUTH: *OS–Controlled Cache Predictability for Real–Time Systems*. In *Proceedings of the Third IEEE Real-Time Technology and Applications Symposium (RTAS'97)*, Montreal, Canada, June 9–11 1997.

[30] LI, JINGYUAN: *Analysis of the Real-Time Performance of a NUMA-Based Multiprocessor System*. Master's thesis, Institute for Real–Time Computer Systems, Technische Universität München, 2004.

[31] LINEO, INC.: *RTAI Programming Guide 1.0*, September 2000.

[32] LIU, JANE W. S.: *Real–Time Systems*. Prentice Hall, New Jersey, 2000.

[33] MEHNERT, FRANK, MICHAEL HOHMUTH, and HERMANN HÄRTIG: *Cost and benefit of separate address spaces in real-time operating systems*. In *Proceedings of the 23th IEEE Real-Time Systems Symposium*, Austin, TX, (USA), December 2002.

[34] MESSMER, HANS-PETER und KLAUS DEMBOWSKI: *PC-Hardwarebuch*. Addison–Wesley Publishing Company, München, 7. Auflage, 2003.

[35] MOLL, LAURENT and MARK SHAND: *Systems Performance Measurement on PCI Pamette*. In POCEK, KENNETH L. and JEFFREY ARNOLD (editors): *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 125–133, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[36] MÜLLER, FRANK: *Timing Analysis for Instruction Caches*. Journal of Real-Time Systems, 18:217–247, 2000.

[37] NOZYNSKI, ANDRZEJ: *Einsatz von FireWire auf der RTCPU–SMP–Architektur*. Diplomarbeit, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, 2002.

[38] PATTERSON, DAVID A. and JOHN L. HENNESSY: *Computer Organization and Design*. Morgan Kaufmann Publishers, San Francisco, 2nd edition, 1998.

[39] PETTERS, STEFAN M.: *Worst Case Execution Time Estimation for Advanced Processor Architectures*. PhD thesis, Institute for Real–Time Computer Systems, Technische Universität München, September 2002.

[40] RUBINI, ALESSANDRO and JONATHAN CORBET: *Linux Device Drivers*. O'Reilly, Sebastopol, 2nd edition, 2001.

[41] SCHNEIDER, J. and CH. FERDINAND: *Pipeline Behavior Prediction for Superscalar Processors by Abstract Interpretation*. In *Proceedings of the ACM SIGPLAN 1999 Workshop on Languages, Compilers, and Tools for Embedded Systems*, pages 35–44, May 1999.

[42] SCHÖNBERG, SEBASTIAN: *Using PCI-Bus Systems in Real-Time Environments*. PhD thesis, Department of Computer Science, Institute for System Architecture, Technische Universität Dresden, June 2002.

[43] SCHÖNBERG, SEBASTIAN: *Impact of PCI-Bus Load on Applications in a PC Architecture*. In *Proceedings of 24th IEEE International Real–Time Systems Symposium*, Cancun, Mexico, December 2003.

[44] SHANLEY, TOM: *PCI-X System Architecture*. Addison–Wesley Publishing Company, Boston, 2001.

[45] SHANLEY, TOM and DON ANDERSON: *PCI System Architecture*. Addison–Wesley Publishing Company, 3rd edition, 1995.

[46] SRINIVASAN, B., S. PATHER, R. HILL, F. ANSARI, and D. NIEHAUS: *A Firm Real–Time System Implementation Using Commercial Off–The–Shelf Hardware and Free Software*. In *Proceedings of Real–Time Technology and Applications Symposium, Denver*, June 1998.

[47] STAPPERT, F. and P. ALTENBERND: *Complete Worst-Case Execution Time Analysis of Straight-Line Hard Real-Time Programs*. Journal of System Architecture, 46:339–355, 2000.

[48] STOHR, J., A. VON BÜLOW und M. GOEBL: *Einflüsse des PCI–Busses auf das Laufzeitverhalten von Realzeitsoftware*. Technischer Bericht, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, Dezember 2003.

[49] STOHR, JÜRGEN: *Untersuchungen zur Eignung der Intel SMP Architektur für Realzeitsysteme*. Diplomarbeit, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, 2002.

[50] STOHR, JÜRGEN, ALEXANDER VON BÜLOW, and GEORG FÄRBER: *Controlling the Influence of PCI DMA Transfers on Worst Case Execution Times of Real–Time Software*. In *Proceedings of the 4th International Workshop on Worst Case Execution Time Analysis in conjunction with the 16th Euromicro Conference on Real–Time Systems*, Catania, Italy, June 2004.

[51] STOHR, JÜRGEN, ALEXANDER VON BÜLOW, and GEORG FÄRBER: *Using State of the Art Multiprocessor Systems as Real-Time Systems – The RECOMS Software Architec-*

ture. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems – Work in Progress Session*, Catania, Italy, June 2004.

[52] STOHR, JÜRGEN, ALEXANDER VON BÜLOW, and GEORG FÄRBER: *Bounding Worst-Case Access Times in Modern Multiprocessor Systems*. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Balearic Islands, Spain, July 2005.

[53] TIETZE, U. und CH. SCHENK: *Halbleiterschaltungstechnik*. Springer, Berlin, 12. Auflage, 2002.

[54] TRODDEN, J. and D. ANDERSON: *HyperTransport System Architecture*. Addison–Wesley Publishing Company, Boston, 2003.

[55] WALDSCHMIDT, KLAUS: *Parallelrechner*. B. G. Teubner, Stuttgart, 1995.

[56] WEGENER, JOACHIM and FRANK MÜLLER: *A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints*. In *Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium (RTAS'98)*, pages 179–188. IEEE, June 1998.

[57] YODAIKEN, VICTOR: *The RTLinux Manifesto*. In *Proceedings of the 5th Linux Expo*, Raleigh, NC, March 1999.

[58] YU, LANG: *Anbindung einer AD/DA Wandlerkarte an ein Quad-Opteron Multiprozessorsystem*. Diplomarbeit, Lehrstuhl für Realzeit–Computersysteme, Technische Universität München, 2004.