TECHNISCHE UNIVERSITÄT DRESDEN

DIPLOMVERTEIDIGUNGSVORTRAG

Eine generische Software-MMU für den Befehlssatzsimulator Jahris

Michael Jahn

Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur
Institut für Technische Informatik
Fakultät Informatik

Dresden, April 2, 2014

Inhalt

1 Einführung

Jahris

- Befehlssatzsimulator, geschrieben in Java.
- Portabel, schnell, retargierbar.
- Übersetzung in interpretierbare Zwischenrepräsentation.
- Erzeugung von Java Bytecode aus der Zwischenrepräsentation.

Memory Management Unit

- Ermöglicht Multitasking durch Adressraumorganisation.
- Virtueller Adressraum wird seitenweise auf physischen Adressraum abgebildet.
- Überprüfung von Zugriffsrechten.
- Konfiguration der Adressräume durch das Betriebssystem in mehrstufigen Seitentabellen (page tables).

1 Einführung

Translation Lookaside Buffers (TLBs)

- Cachen Zuordnungen von virtuellen zu physischem Speicher.
- TLB-Lookup für jeden einzelnen Speicherzugriff.
- Fehlende TLB-Einträge müssen nachgeladen werden.
- Nachladen mittels Hardware (page walk der page tables) oder durch spezielle Instruktionen (Software).
- Seitenfehler (page faults) werden vom Betriebssystem behandelt.

2 Entwurfsentscheidungen

Wie wurden Entwurfsentscheidungen getroffen?

- Abwägen von Pro- und Kontra.
- Höhere Performance rechtfertigt nicht jede Einschränkung der Wiederverwendbarkeit und vice versa.

Entwurfsentscheidungen

1. Hohe Performance bei realistischen Anwendungen.
2. Gute Wiederverwendbarkeit.
3. Gute Anpassbarkeit an neue Architekturen.

Bitweise Adressierung

- Originale Implementierung.
- Generische Modellierung von Byte- (z.B. 4, 7, oder 9 Bit) und Wortformaten (z.B. 3 Byte).
- Emulation von unaligned Speicherzugriffen.
- Komplex.
- Speicherpfad ist kritisch für die Simulationsperformance.

3 Speichersystem
Byteweise Adressierung

- Neue Implementierung.
- Standard Datentypen der JVM (8Bit/Byte, 4 bzw. 8 Byte/Word).
- Einsparung von Bitmasken und Shifts.
- Alignment Checks in den ***BusDevice***s.
- Dedizierter Kanals für Instruktionen.

3 Speichersystem
FastRAM

- Basiert auf sun.misc.Unsafe.
- Keine Bounds-Checks.
- Allokation eines Memory-Mappings im Adressraum des Java Prozesses.
- Unsafe-Operationen auf diesem Mapping.
- On-Demand Speicherallokation durch das Betriebssystem.
- Unaligned Speicherzugriffe sind problemlos.
- Benötigt ***CheckingAddressWindow*** falls einziges Gerät auf dem Speicherbus.
- Alternative mit Bounds-Checking: ***FastRAM_ByteBuffer***.

3 Speichersystem
PagingDelegator

- Zuordnung von Speicheradresse zu ***BusDevice***.
- Mehrstufige Lookup-Table (angelehnt an Seitentabellen).
- 1MB Eintrag in der 1. Stufe ist entweder eine Seitentabelle, ein ***BusDevice*** oder unmapped.
- 1kB Eintrag in der 2. Stufe analog.
- 4B Eintrag in der 3. Stufe ist ein ***BusDevice*** oder unmapped.
- Impliziert Bounds-Checking (***FastRAM***).
- Vorteil: >1MB ***BusDevice***s im 1. Level.

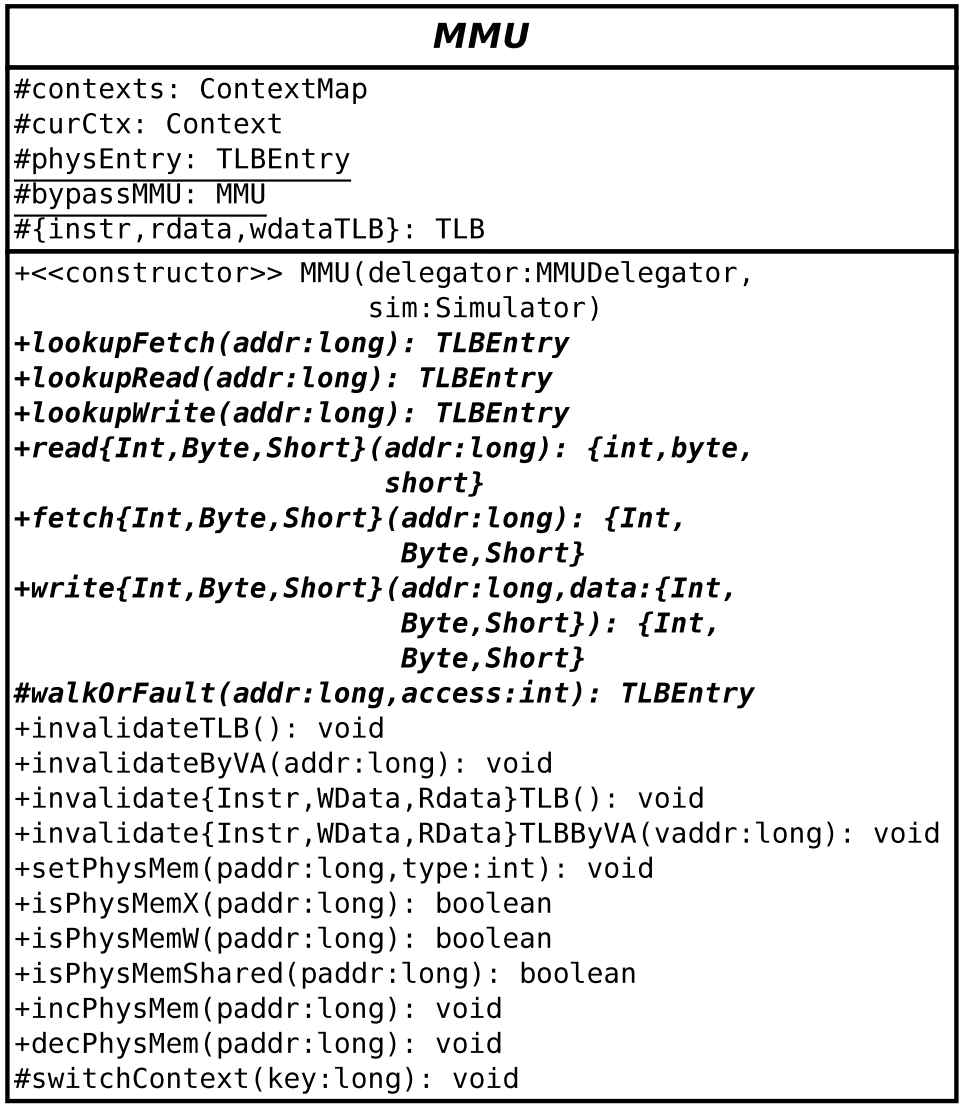
4 Memory Management Unit

Grundsätzlich

- Generische Software-MMU.
- Architekturspezifisch: Seitentabellen und -fehler, Konfigurationsregister, Sonderverhalten.
- Generisch: *TLB*s, *TLBEntry*s und *MMU*.
- Anbindung der *MMU* mittels *PagingMMUDelegator*.
- *BypassMMU* mit statischen *TLBEntry*.
- Ziel: Komplexe bzw. optimierte Algorithmen generisch.

MMU

- Verwaltung der TLBs (read, write, instruction fetch).
- Hinzufügen und Invalidieren von TLB Einträgen.
- Implementierung sämtlicher Zugriffsmethoden zur Feststellung der Zugriffsmodalitäten.
- Verwaltet Datenstrukturen zu physischen Speicherseiten zur Referenzzählung (Shared-Memory Detektion) und Typisierung (W, X)


MMU
#contexts: ContextMap
#curCtx: Context
#physEntry: TLBEntry
#bypassMMU: MMU
#{instr,rdata,wdataTLB}: TLB
+<<constructor>> MMU(delegator:MMUDelegator, sim:Simulator)
+lookupFetch(addr:long): TLBEntry
+lookupRead(addr:long): TLBEntry
+lookupWrite(addr:long): TLBEntry
+read{Int,Byte,Short}(addr:long): {int,byte, short}
+fetch{Int,Byte,Short}(addr:long): {Int, Byte,Short}
+write{Int,Byte,Short}(addr:long,data:{Int, Byte,Short}): {Int, Byte,Short}
#walkOrFault(addr:long,access:int): TLBEntry
+invalidateTLB(): void
+invalidateByVA(addr:long): void
+invalidate{Instr,WData,Rdata}TLB(): void
+invalidate{Instr,WData,RData}TLBByVA(vaddr:long): void
+setPhysMem(paddr:long,type:int): void
+isPhysMemX(paddr:long): boolean
+isPhysMemW(paddr:long): boolean
+isPhysMemShared(paddr:long): boolean
+incPhysMem(paddr:long): void
+decPhysMem(paddr:long): void
#switchContext(key:long): void

4 Memory Management Unit

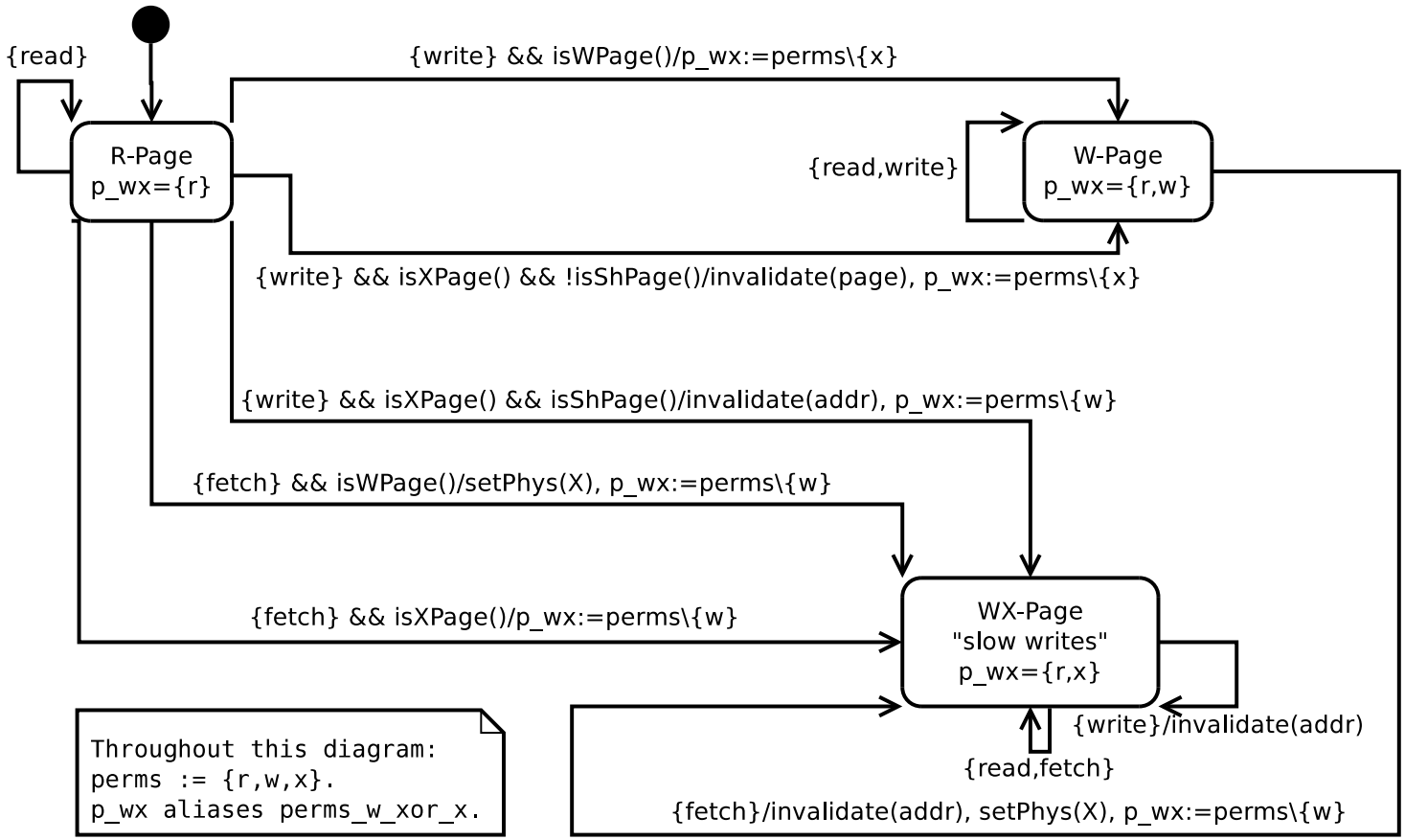
TLBEntry und *TLB*

- *TLBEntry* muss architekturspezifisch implementiert werden.
- *ARMTLBEntry* erweitert *TLBEntry* um data/prefetch_abort Exceptions und implementiert den Konstruktor.
- *TLB*s sind architekturunabhängig.
- *IndexTLB*s und *PagingTLB*s.
- *CompactPagingTLB* ist der schnellste TLB Mechanismus.
- Zweistufige Lookup-Table.
- 1. Stufe ist im *CompactPagingTLB* (1MB).
- 2. Stufe wird referenziert aus *TLBEntry* der 1. Stufe (1kB).

TLBEntry
+<<constructor>> TLBEntry(context:Context, dev:BusDevice,adjust_va2dev:long, adjust_va2phys:long, perms:long,paddr_start:long, va_norm_start:long, va_norm_end:long) +<<constructor>> TLBEntry(context:Context) +<<constructor>> TLBEntry(level:TLBEntry[]) +match(addr:long): boolean +translate_phys(addr:long): long +translate_physmask(addr:long): long +translate_dev(addr:long): long +check(addr:long,access:int): void +read{Int,Byte,Short}(addr:long,access:int): {Int, Byte,Short} +fetch{Int,Byte,Short}(addr:long,access:int): {Int, Byte,Short} +write{Int,Byte,Short}(addr:long,data:{Int, Byte,Short},access:int): void ***#checkFault(addr:long,access:int): void***

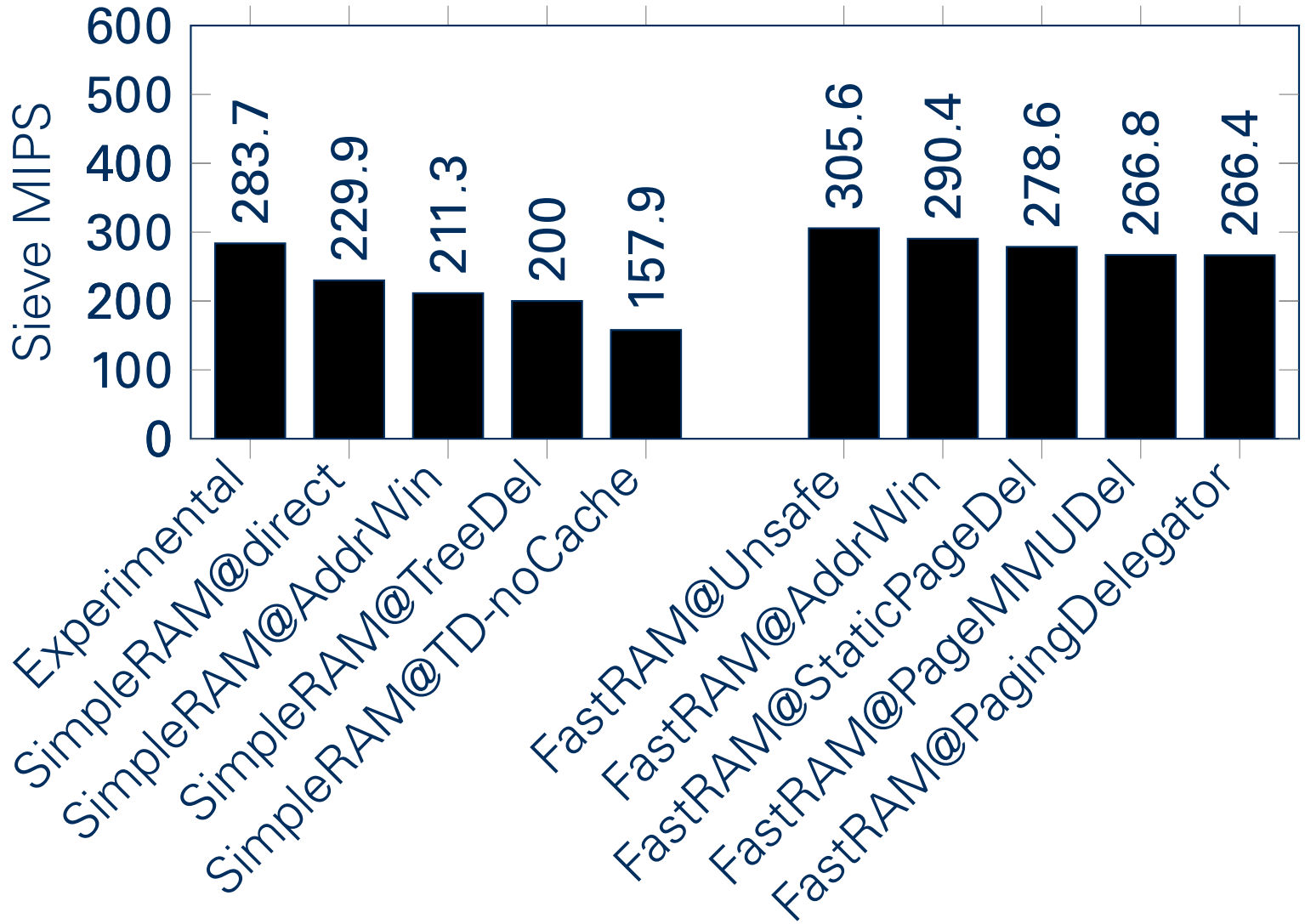
4 Memory Management Unit

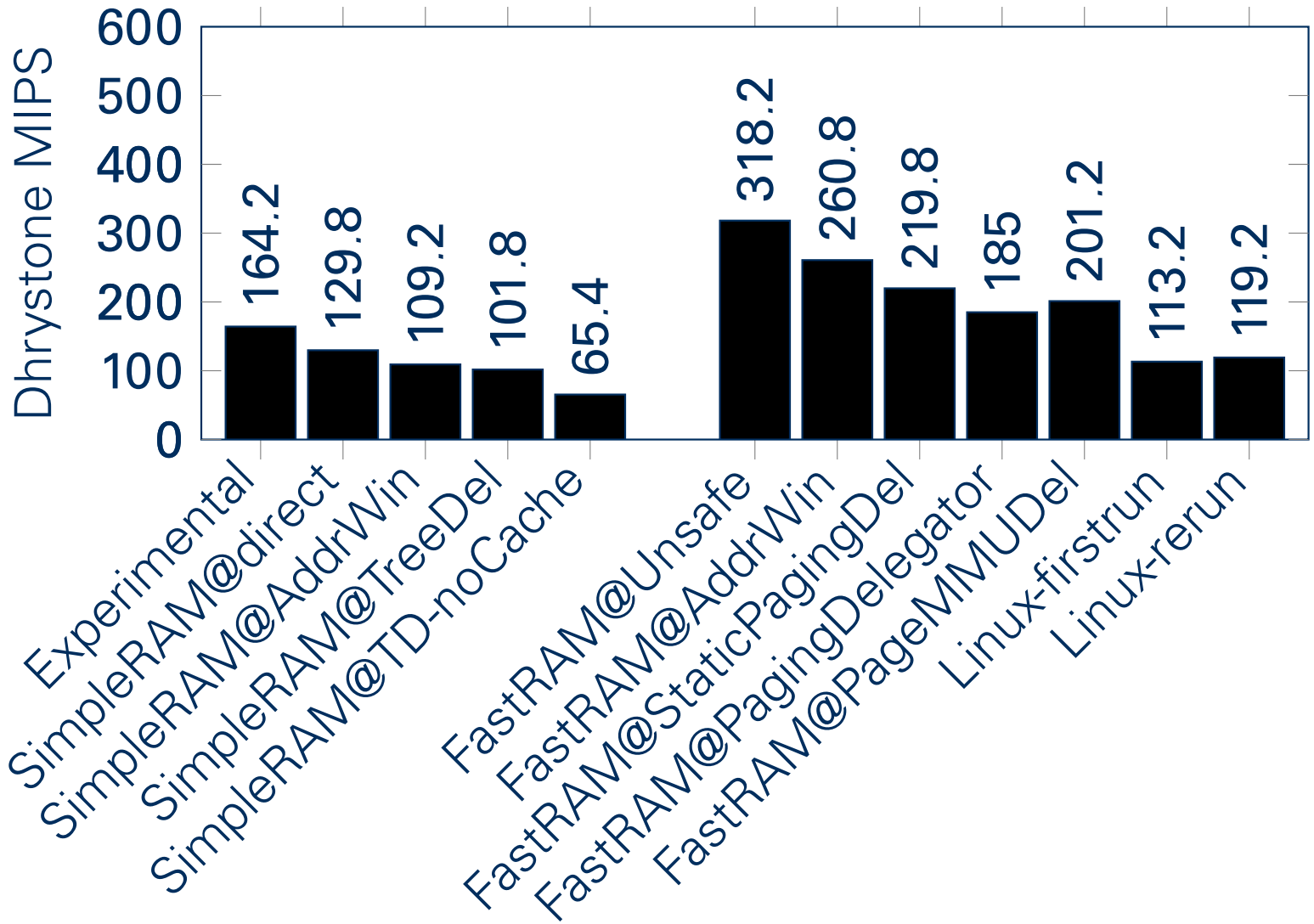
Transitionen eines *TLBEntry*



4 Memory Management Unit
Optimierungen

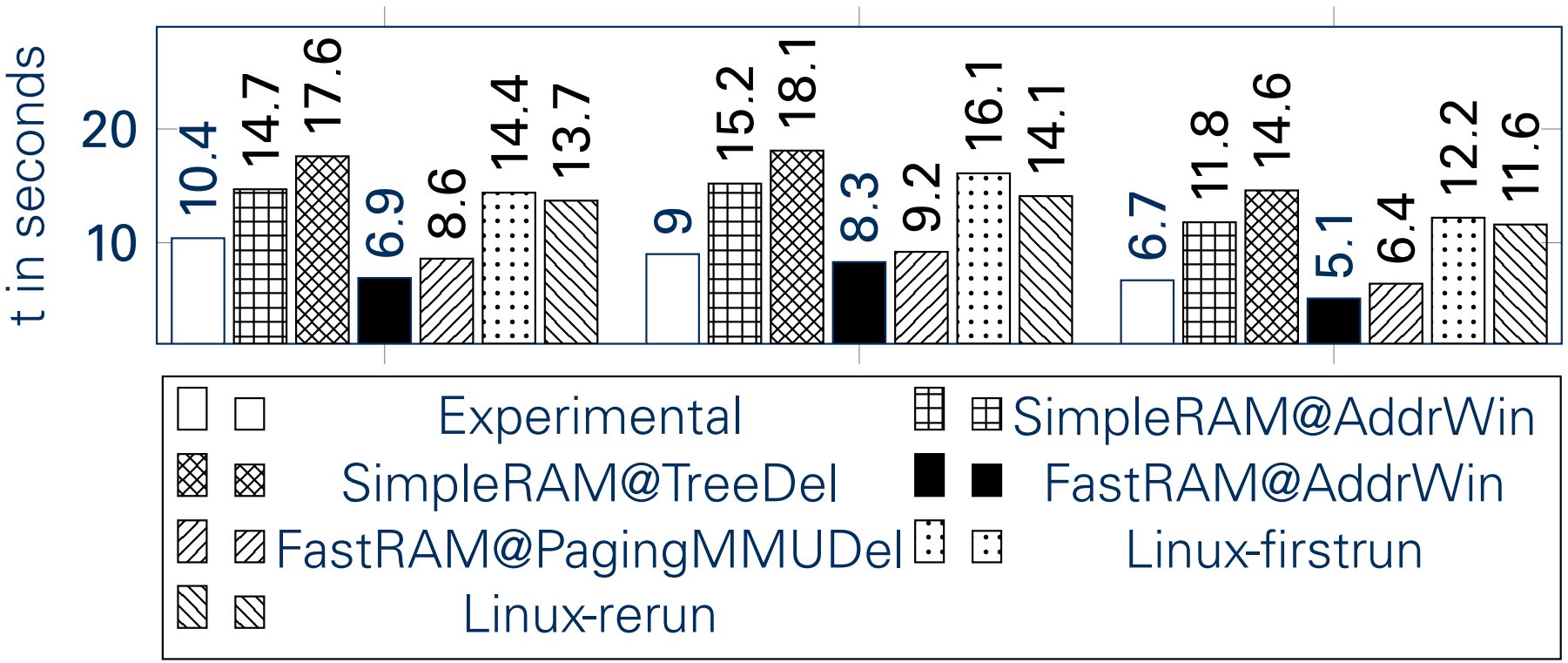
- Direkte Übersetzung von virtuellen Gastadressen in *BusDevice*-Adressen bzw. virtuelle Hostadressen.
- Prüfung aller Zugriffsrechte (r/w/x-s/u) mit einem Vergleich.
- ARM: Vorausberechnete Masken für Zugriffsprivilegien (User/Superuser?).
- *UpdateMMUException* wechselt run-Methode des *Simulator*s.
- *static*.


Sieve MIPS
600
500
400
300
200
100
0
283.7
229.9
211.3
200
157.9
305.6
290.4
278.6
266.8
266.4
Experimental
SimpleRAM@direct
SimpleRAM@AddrWin
SimpleRAM@TreeDel
SimpleRAM@TD-noCache
FastRAM@Unsafe
FastRAM@AddrWin
FastRAM@StaticPageDel
FastRAM@PageMMUDel
FastRAM@PagingDelegator


Dhrystone MIPS
600
500
400
300
200
100
0
164.2
129.8
109.2
101.8
65.4
318.2
260.8
219.8
185
201.2
113.2
119.2
Experimental
SimpleRAM@direct
SimpleRAM@AddrWin
SimpleRAM@TreeDel
SimpleRAM@TD-noCache
FastRAM@Unsafe
FastRAM@AddrWin
FastRAM@StaticPagingDel
FastRAM@PagingDelegator
FastRAM@PageMMUDel
Linux-firstrun
Linux-rerun

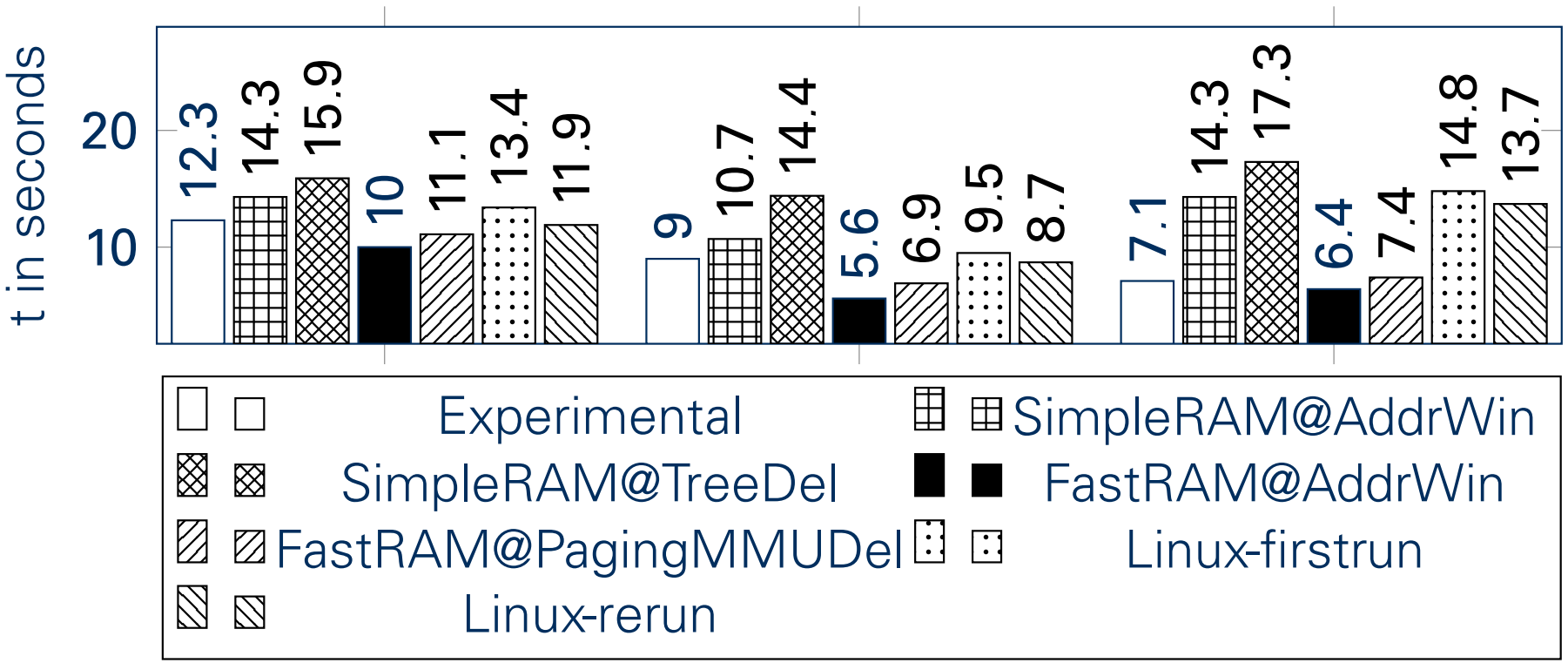
5 Ergebnisse
EEMBC Automotive I.

- V.l.n.r.: canrdr (CAN Remote Req/Resp.), idctrn (Inv. Disc. Cosine Transform), rspeed (Road Speed Calc.)



5 Ergebnisse
EEMBC Automotive II.

- V.l.n.r.: tblook (Table Lookup), cacheb (Cache Buster), aifftr (Fast Fourier Transform).



6 Abschluss

Vorschläge für weitere Entwicklung

- Weitere Architekturen.
- Inlining der TLB-Lookups.
- Mehrfach-Load/Store Befehlen.

6 Abschluss
Linux Demonstration

- Funktionsprüfung durch Ausführung eines ARM Linux.
- Zusätzlich notwendige Geräte: IC, UART, Timer.
- Umfangreiche Erweiterungen des Interruptsystems und des ARM Architekturmodells.
- ATAGS Bootloader für die Simulationskonsole.

Vielen Dank für Ihre Aufmerksamkeit.

Haben Sie Fragen?

6 Abschluss
Literatur I


ARM Limited.
ARM926EJ-S Technical Reference Manual, 2008.


ARM Limited.
ARM® Architecture Reference Manual ARM® v7-A and ARM® v7-R edition Errata markup, 2010.


The Embedded Microprocessor
Benchmark Consortium EEMBC.
Autobench 1.1 software benchmark data book.


Michael Jahn.
A generic software mmu für the jahris instruction set simulator, 2014.

6 Abschluss
Literatur II


Marco Kaufmann.
Erschließung von just-in-time-compilierungstechniken in der realisierung eines retargierbaren architektursimulators, 2009.


William Stallings.
Operating Systems: Internals and Design Princ., 7/E.
Pearson, 2012.


Reinhold P Weicker.
Dhrystone: a synthetic systems programming benchmark.
Communications of the ACM, 27(10):1013–1030, 1984.

Referenzzählung physischer Speicherseiten I.

- Problem: Speicherseiten die sowohl geschrieben als auch ausgeführt werden (z.B. 1MB Kernel Seiten).
- Wenn 1. Zugriff nach Invalidierung der TLBs ein Schreibzugriff: Annahme einer Schreibseite und Invalidierung des Simulationscaches falls die Seite vorher eine Execute-Seite ist.
- Aber: 1MB Seite leert den gesamten direct-mapped Simulationscache.
- Referenzzählung der Speicherseiten liefert Informationen zur Auswahl eines günstigen *TLBEntry*-Zustands.

Referenzzählung physischer Speicherseiten II.

- *Context* enthält Kopie der zu einem Adressraum zugehörigen *TLBEntry*s zur Referenzzählung.
- *Context*e wird nicht mit den TLBs der *MMU* invalidiert und bleibt nach Kontextswitch erhalten.
- *ContextMap* implementiert eine kapazitätsbeschränkte HashMap (LRU) für *Context*e.
- Verdrängung aus der *ContextMap* entfernt alle Referenzen.
- Falls die Anzahl der Referenzen auf eine physische Seite > 1 ist handelt es sich um Shared-Memory.

Wieviel Code ist generisch?

Klasse	Codezeilen
MMU	552
TLBEntry	310
CompactPagingTLB	175
MMUDelegator	188
Summe Generisch	**1225**
ARMTLBEntry	285
ARMMMU	623
Summe Spezifisch	**908**

- *ARMTLBEntry*-Fabrikmethode 210 Zeilen.
- *ARMMMU* mit Methoden zum Auslesen der Seitentabellen (98), ***SwitchOffMMU*** (83), und *ARMFault* (58).