TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Institut für Technische Informatik , Lehrstuhl für VLSI-EDA

Potential dynamisch rekonfigurierbarer Ressourcen beim Cache Mapping

Josia Mädler // Donnerstag, 21.06.2018

Betreuer: Dipl.-Inf. Paul R. Genßler

Inhalt

Hinführung zum Thema Cache Mapping
- Was ist Cache Mapping?

Hard- und Software zur Realisierung des dynamischen Cache Mappings
- LEON 3
- Prototypische Implementierung
- Integration der Rekonfigurierbaren Schaltkreise
- Encodierung: Cartesion Genetic Programming
- Rekonfigurations-Prozess

Benchmark Ergebnisse und Interpretation
- Evolution der Konfigurationen
- Diagramme

Zusammenfassung
- Bedeutung
- Ausblick

Hinführung
zum Cache Mapping

Hinführung
Was ist Cache Mapping?

- L1-Caches sind sehr viel kleiner
- Starkes anwachsen der Datenmengen
- Viele Programme gleichzeitig
- Gruppe aus Paderborn mit LEON3

Evolvable Caches: Optimization of Reconfigurable Cache Mappings for a LEON3/Linux-based Multi-Core Processor

Nam Ho, Paul Kaufmann and Marco Platzner
Department of Computer Science, Paderborn University, Germany
Email: namh@mail.upb.de, paul.kaufmann@gmail.com, platzner@upb.de

***Abstract*—Reconfigurable hardware technology can help to improve the performance of processor caches by, for instance, tailoring and adapting associativity, block sizes, and replacement strategies to a particular application. A fundamentally different approach in using reconfigurability for better caches, and topic of this work, is cache adaptation by optimizing the memory-to-cache-index mapping function. This idea has been investigated previously but we present and evaluate for the first time a complete hardware implementation of such a system. Using a LEON3 multi-core processor running a standard Linux OS we optimize cache mappings for 12 applications from the MiBench suite with the result that miss rates can be reduced by up to 67% compared to the conventional cache mapping.**

I. Introduction

Conventional caches use a consolidated block of memory address bits to index a cache set. Mathematically, this corresponds to the computation of the modulo power-of-2 function. This mapping scheme is used in contemporary caches due to hardware design simplicity and good performance for sequences of consecutive addresses. Optimizing the conventional cache mapping function has been investigated previously. Permutation-based [1], [2] and XOR-based [3] mappings have been used to reduce the number of cache misses. Recently, non-modulo cache mapping functions have been investigated in GPUs [4], [5], [6]. There, non-modulo indexing schemes can help achieving uniform accesses to the data cache that is shared by thousands of threads.

A new cache scheme operating with reconfigurable mapping functions has been proposed in our previous work [7]. We proposed realizing arbitrarily Boolean functions for computing the cache index sets by adding small reconfigurable fabrics to the CPU. Fabrics' configurations are evolved and optimized by a local search (LS) metaheuristic. The paper evaluated the approach for single-core OS-less system configuration. More recently, more complex cache address translation functions have been investigated in [8], [9].

In this paper, we are presenting for the first time a fully working hardware implementation of a processor that is able to freely redefine its memory-to-cache address functions and reconfigure them at any point of time. We have extended the caches and their snooping mechanisms of a Gaisler LEON3 SPARC multi-core architecture, realized universal cache and hardware event sensors that are incorporating smoothly into the standard Linux performance measurement infrastructure and extended the Linux kernel to handle cache mapping reconfigurations during task switches. To that end, we show in the experiments that a multi-core processor benefits greatly from reconfigurable cache mapping functions and that the optimized cache mapping functions can reduce cache misses by up to 67%.

II. Reconfigurable Mapping Cache Architecture

A. Cache Organization

The reconfigurable cache subsystem of the LEON3 processor is presented in Fig. 1. It consists of a Physically Indexed Physically Tagged (PIPT) L1 data (L1:D) cache, a Virtually Indexed Virtually Tagged (VIVT) L1 instruction (L1:I) cache, the reconfiguration controller (RC), and the reconfigurable circuit blocks.

In the original LEON3 implementation the L1:D caches were virtually addressed. The virtual indexes were aligned among the L1:D caches of different cores to prevent synonyms. Holding on to this scheme while using reconfigurable cache mappings with non-aligned indexes would require larger and wider back pointer tables for synonym detection. To avoid this overhead, we configured the L1:D caches to be physically addressed. In such a configuration, the Data Cache (D) Translation Look-aside Buffer (DTLB) needs to be consulted on each data access, adding latency to the processor pipeline (cf. Fig. 1). On the other hand, the implementation of back pointer tables can be skipped. The integration of reconfigurable mappings into the L1:D caches is shown at the bottom-right of Fig. 1. There are reconfigurable circuit blocks placed in between the DTLB and the cache memory, and the cache memory and the memory bus. The first reconfigurable block maps processor memory requests to cache indexes while the second reconfigurable block snoops for write requests on the memory bus and checks, if a local cache block needs to be invalidated.

The L1:I cache is read-only and does not need to snoop on the memory bus for write invalidate requests. L1:I virtual addressing mode can therefore be extended by reconfigurable cache mappings without any additional measures.

B. Reconfigurable Mapping Circuits

The memory-to-cache address mapping functions are encoded using the Cartesian Genetic Programming model (CGP) [10]. CGP is well suited to represent combinational

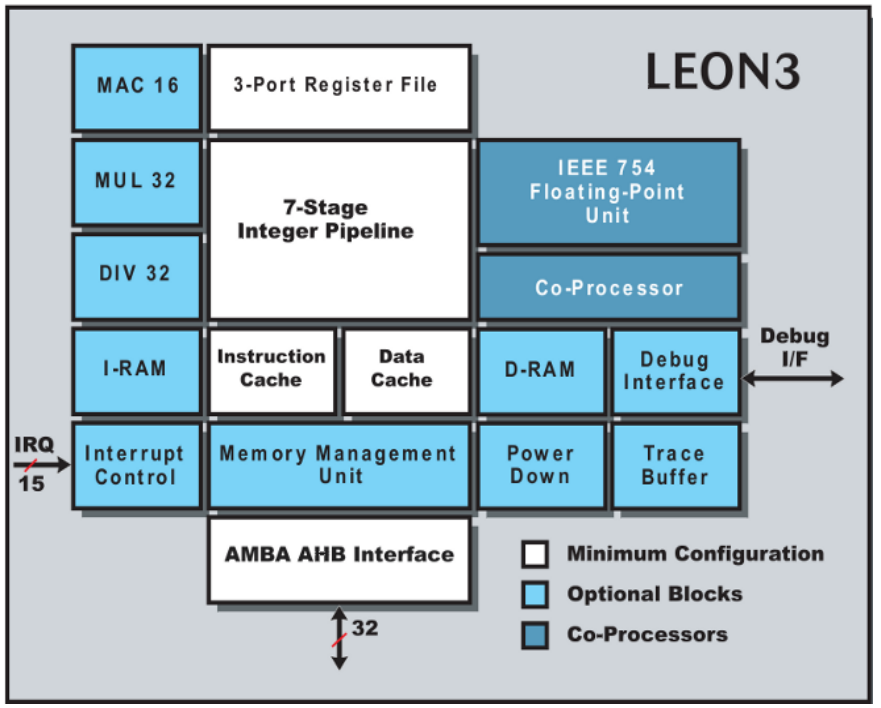
978-1-5386-2656-6/17/$31.00 © 2017 IEEE 215 FPT 2017

Quelle: [7]

Hard- und Software
Realisierung des dynamischen Cache Mappings

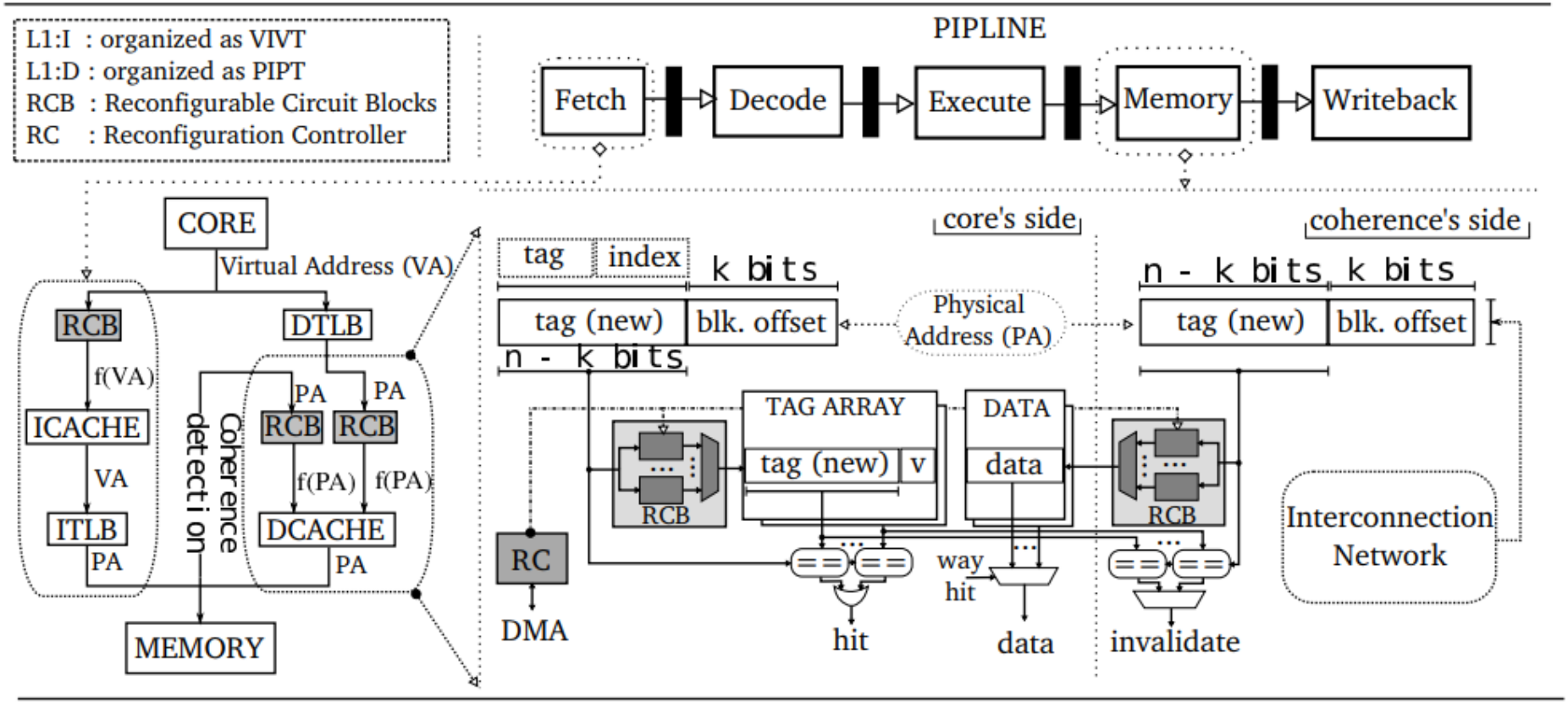
Hard- und Software
Der LEON3

- SPARC V8 Architektur
- Open Source
- VHDL
- Immer wieder bei Forschungen verwendet
- Unterstützt Linux
- Prototypische Implementierung:
  - Adressbreite 32-Bit
  - 8KB Datencache
  - 4KB Befehlscache
  - 50MHz



Quelle: [ZQ1]

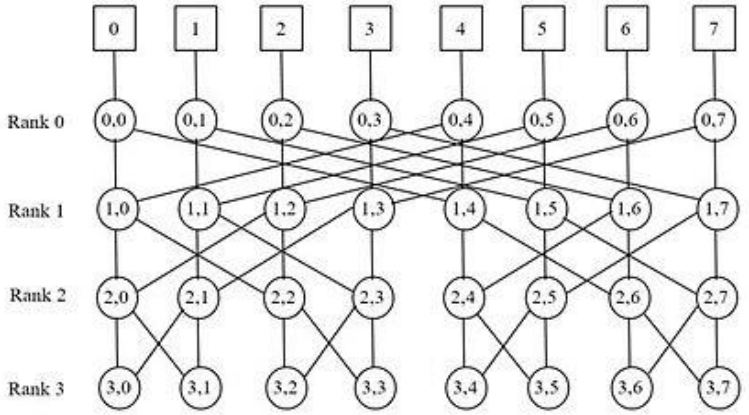
Hard- und Software
Integration der rekonfigurierbaren Hardware



Quelle: [7]

Hard- und Software
Cartesion Genetic Programming

- Biologisch inspirierte Methode
- Geeignet für neurale Netzwerke, elektrische Schaltkreise und Computer Programme
- Azyklische gerichtete Graphen
- String von Binär Werten (Genotyp)
- 101 111 001 …
- Repräsentiert ein Butterfly Netzwerk



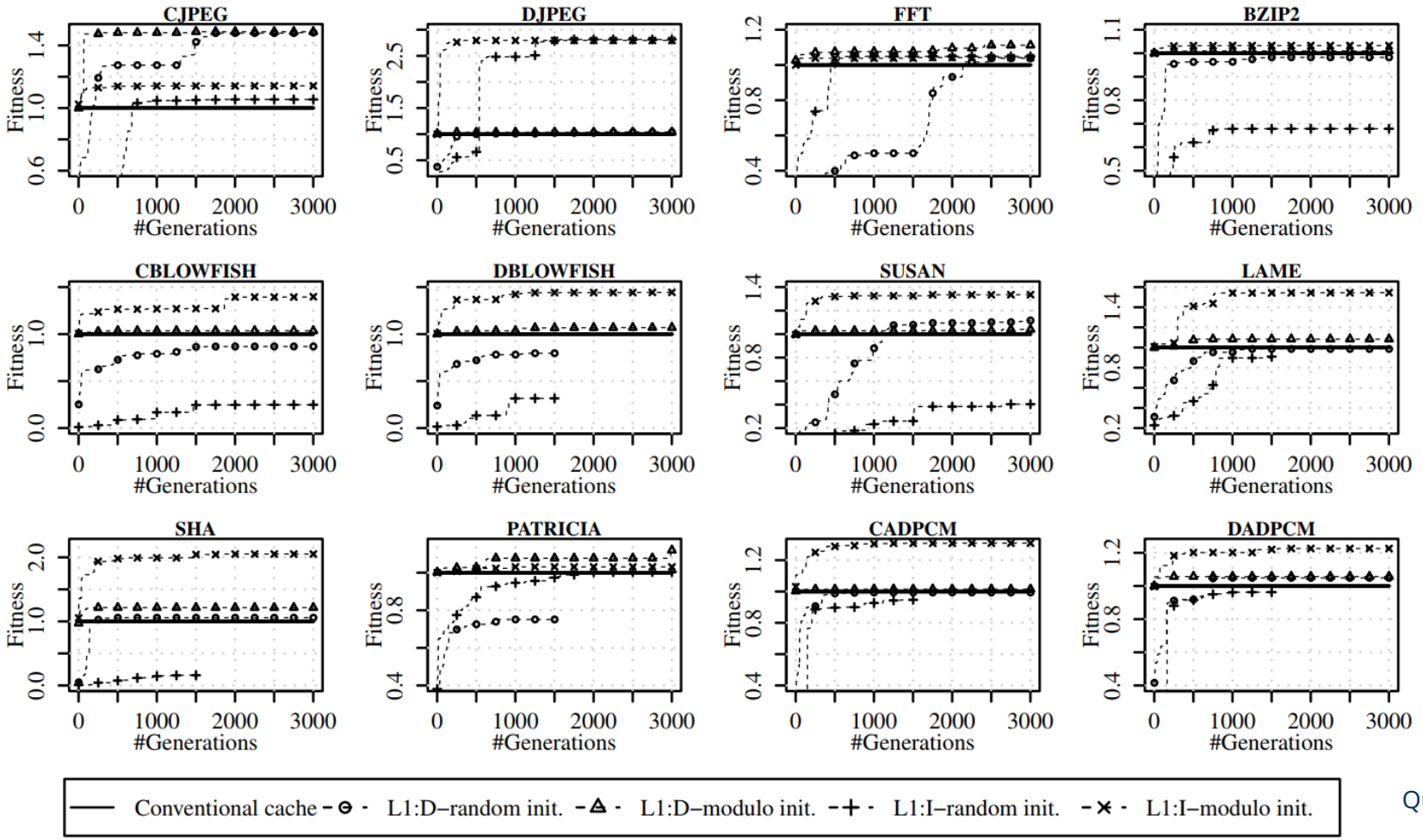
Quelle: [15]

Hard- und Software
Rekonfigurations-/Trainingsprozess

- Rekonfiguration durch zusätzliche Controller
- Ausgangspunkt:
  - Zufällige Konfiguration
  - Modulo Konfiguration
- Lernen durch 1+1 Hill Climber
- Bei jeder Iteration werden einzelne Bits des Genotyps geändert
- Maximal 3000 Iterationen
- Monitoring:
  - PMU selten vorhanden
  - Eigene PMU Lösung (siehe Literatur)
  - Direkt in Hardware

Benchmark
Ergebnisse und Interpretation

Benchmark
Evolution der Cache Mapping Konfiguration



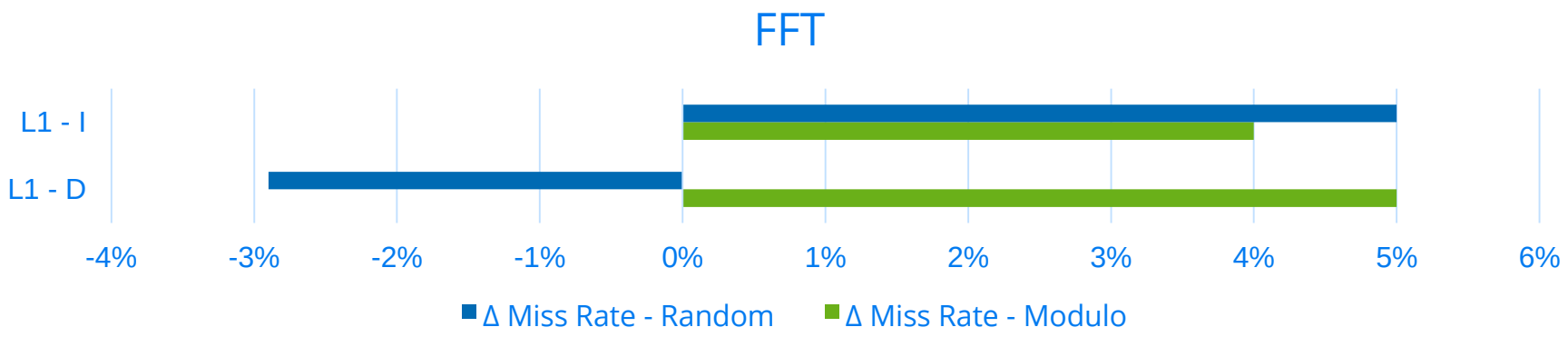
Quelle: [7]

Benchmark
Tabelle

Application	Init	L1:D [%]			L1:I [%]		
		Trn. Q2	Test Q2	Test max	Trn. Q2	Test Q2	Test max
CJPEG	rand	32.5	35.16	38.20	5.2	-0.62	6.25
	mod	32.7	29.13	37.71	12.4	5.47	10.47
DJPEG	rand	3.6	3.34	5.38	64.5	67.74	73.34
	mod	3.4	-11.85	-1.06	64.2	67.56	72.85
FFT	rand	3.7	-2.91	11.00	4.64	4.68	5.62
	mod	10.08	5.01	9.10	4.03	4.00	5.88
BZIP2	rand	-1.8		-	-47.34	-	-
	mod	0.94	0.97	1.54	3.16	1.61	3.33
CBFISH	rand	-15.2		-	-303.1	-	-
	mod	3.4	3.46	4.54	28.47	16.48	18.66
DBFISH	rand	-25.6		-	-216.1	-	-
	mod	6.5	4.84	7.23	30.7	19.64	24.42
SUSAN	rand	10.4	-7.89	8.84	-147.9	-	-
	mod	3.8	-22.50	3.37	25.12	7.30	23.00
LAME	rand	-1.47		-	-9.88	-	-
	mod	7.68	-7.07	-2.69	35.27	20.23	28.95
SHA	rand	5.46	-20.89	-3.35	-525.24	-	-
	mod	17.68	-3.80	-2.65	51.24	45.10	46.74
PATRICIA	rand	-33.15	-	-	0.35	3.30	3.30
	mod	10.66	6.35	6.50	3.00	3.30	3.30
CADPCM	rand	-0.65		-	-5.61	-	-
	mod	1.45	-0.07	0.19	23.58	15.43	19.67
DADPCM	rand	4.63	-77.67	-74.30	-3.87	-	-
	mod	5.38	3.04	4.22	16.28	12.22	19.70

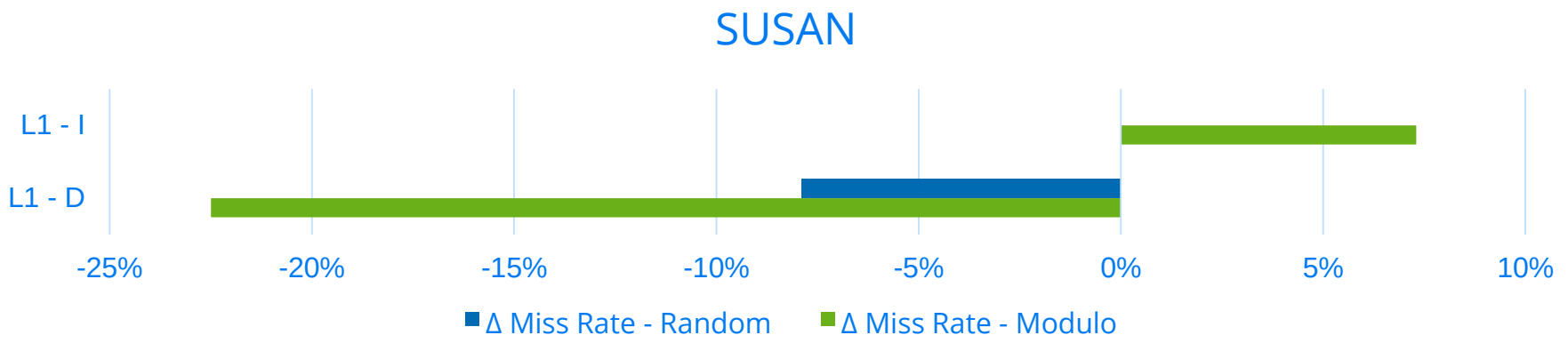
Quelle: [7]

Benchmark
Diagramme



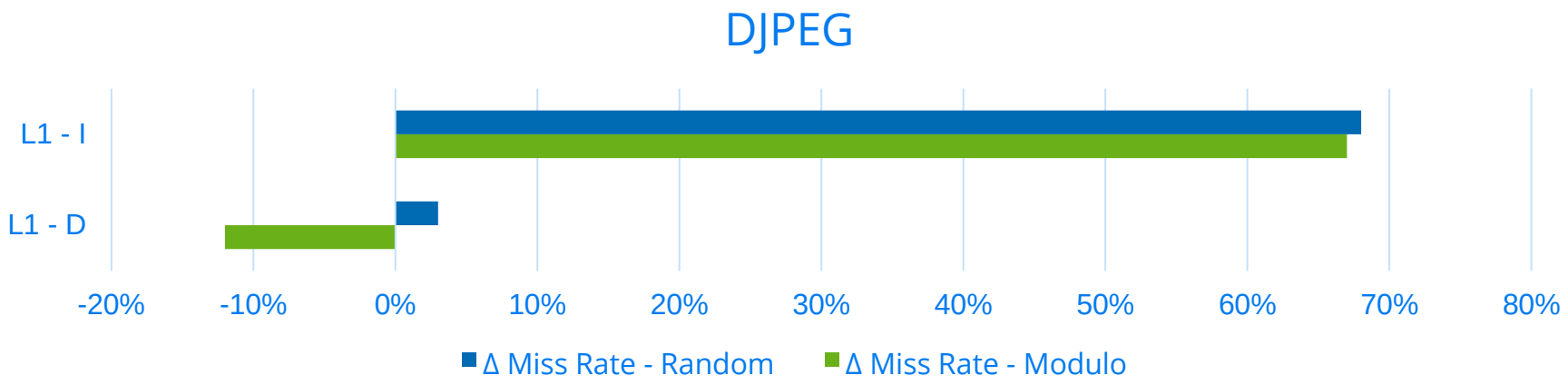
- Durchschnittliches Ergebnis
- FTT:
  - Schnelle Fourier Transformation
  - Typisch bei der Verarbeitung Digitaler Signale

Benchmark
Diagramme



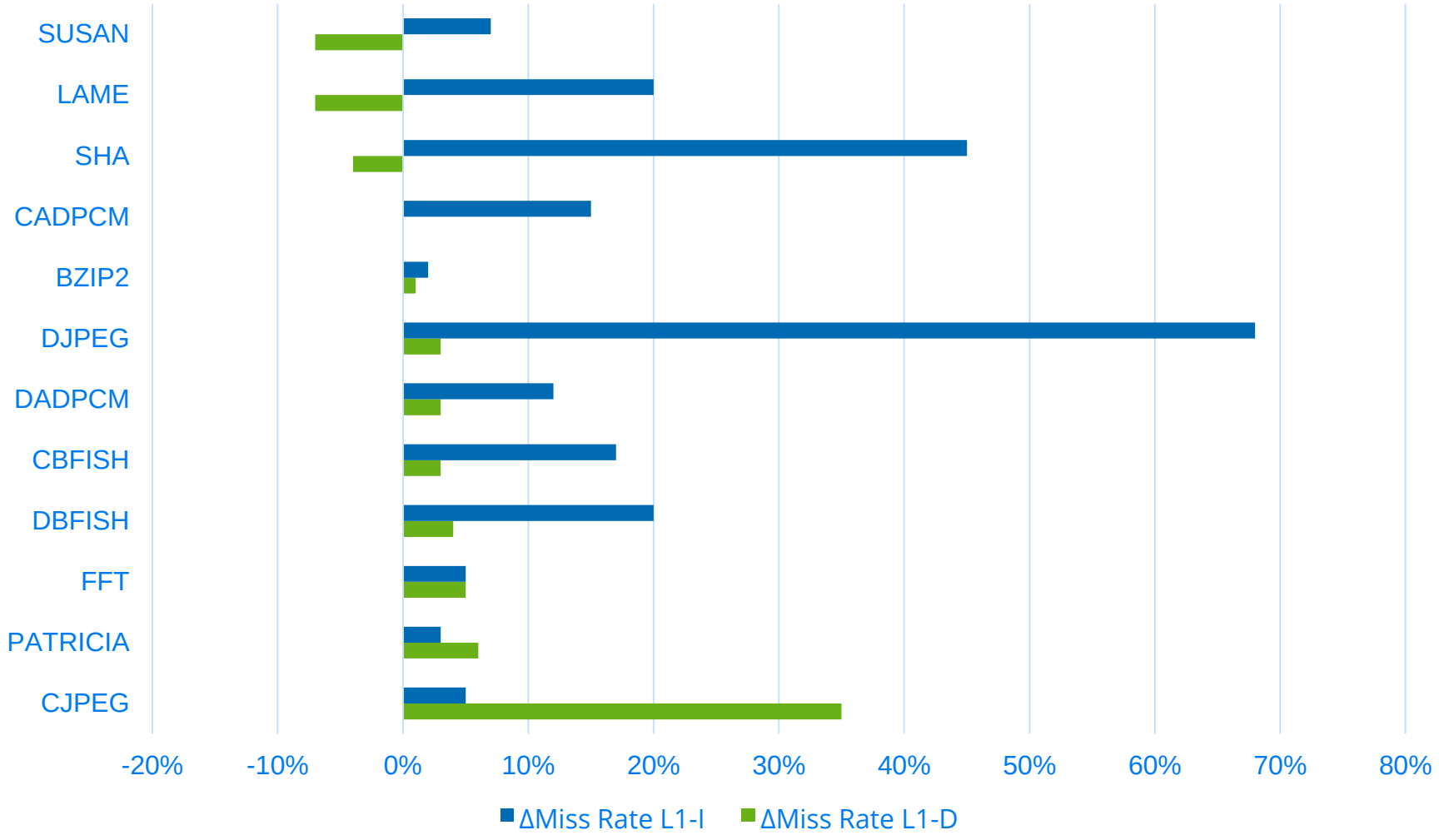
- Schlechtestes L1 – D Ergebnis
- SUSAN:
  - Bilderkennungsalgorithmus für Hirn MRT-Bilder
  - Spiegelt gut reale Probleme wieder

Benchmark
Diagramme



- Bestes Ergebnis
- DJPEG:
  - Dekomprimierung von Bildern
  - Typisch bei eingebetteten Bildern

Benchmark
Diagramm - Beste Ergebnisse



Zusammenfassung
Bedeutung und Ausblick

Zusammenfassung
Bedeutung

- Verbesserungen der Cache Auslastung
- Datencache geringer als beim Befehlscache:
  - Programme für Datencache optimiert
  - Zu groß gewählter Datencache?
- Befehlscache mit deutlichen Verbesserungen
- Vergleich der Ergebniskonfigurationen
  - Ähnlichkeit: eine bessere Konfiguration für alle
  - Unterschiedlich: Einbettung rekonfigurierbarer Hardware vor allem beim Befehlscache
- Wechsel von Cache Mapping Konfigurationen im laufenden Betrieb möglich

Zusammenfassung
Ausblick

- Offene Punkte der Forschung:
  - Einschätzung des Aufwands für die Integration
  - Untersuchung der Ergebniskonfigurationen
- Verwaltung der Wechsel der Cache Mapping Konfigurationen:
  - Durch Betriebssystem
  - Durch Prozessor
  - Durch API für Programmierer
  - Kombinationen

- Mehr Verständnis über das Verhalten von Caches in komplexen Systemen

Quellen
Quell- und Literaturverzeichnis

Literatur

[1]H. Abbasitabar, H. R. Zarandi und R. Salamat. “Susceptibility Analysis of LEON3 Embedded Processor against Multiple Event Transients and Upsets”. In: 2012 IEEE 15th International Conference on Computational Science and Engineering. Dez. 2012, S. 548–553. DOI: 10.1109/ICCSE.2012.8

[2]M. Cekleov und M. Dubois. “Virtual-address caches. Part 1: problems and solutions in uniprocessors”. In: IEEE Micro 17.5 (Sep. 1997), S. 64–71. ISSN: 0272-1732. DOI: 10.1109/40.621215.

[3]Cobham Gaisler. LEON3 Processor. URL: https://www.gaisler.com/index.php/products/processors/leon3 (besucht am 29. 05. 2018).

[4]Kyrre Glette u. a. “A Comparison of Evolvable Hardware Architectures for Classification Tasks”. In: Evolvable Systems: From Biology to Hardware. Hrsg. von Gregory S. Hornby, Lukáš Sekanina und Pauline C. Haddow. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, S. 22–33. ISBN: 978-3-540-85857-7.

[5]M. R. Guthaus u. a. “MiBench: A free, commercially representative embedded benchmark suite”. In: Proceedings of the Fourth Annual IEEE International Workshop on Workload Charac-terization. WWC-4 (Cat. No.01EX538). Dez. 2001, S. 3–14. DOI: 10.1109/WWC.2001.990739.

[6]N. Ho, P. Kaufmann und M. Platzner. “A hardware/software infrastructure for performance monitoring on LEON3 multicore platforms”. In: 2014 24th International Conference on Field Programmable Logic and Applications (FPL). Sep. 2014, S. 1–4. DOI: 10.1109/FPL.2014.6927437.

[7]N. Ho, P. Kaufmann und M. Platzner. “Evolvable caches: Optimization of reconfigurable cache mappings for a LEON3/Linux-based multi-core processor”. In: 2017 International Conference on Field Programmable Technology (ICFPT). Dez. 2017, S. 215–218. DOI: 10.1109/FPT.2017.828014

Literatur

[8]Jagdeep Singh. Slide. 2013. URL: https://pt.slideshare.net/jagdeepmatharu/cartesian-genetic-programming/5?smtNoRedir=1 (besucht am 27. 05. 2018).

[9]Julian F. Miller. “Cartesian Genetic Programming”. In: Cartesian Genetic Programming. Hrsg. von Julian F. Miller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 17–34. ISBN: 978-3-642-17310-3. DOI: 10.1007/978- 3- 642-17310- 3_URL: https://doi.org/10.1007/978-3-642-17310-3

[10]Julian Miller und Andrew Turner. “Cartesian Genetic Programming”. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. GECCO Companion ’15. Madrid, Spain: ACM, 2015, S. 179–198. ISBN: 978-1-4503-3488-4. DOI : 10.1145/2739482.2756571. URL : http://doi.acm.org/10.1145/2739482.275657

[11]S. Penolazzi, L. Bolognino und A. Hemani. “Energy and Performance Model of a SPARC Leon3 Processor”. In: 2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools. Aug. 2009, S. 651–656. DOI: 10.1109/DSD.2009.14

[12]SPARC International Inc. The SPARC Architecture Manual V8. 1991,1992.

[13]T. Vladimirova und Xiaofeng Wu. “On-Board Partial Run-Time Reconfiguration for PicoSatellite Constellations”. In: First NASA/ESA Conference on Adaptive Hardware and Systems (AHS’06). Juni 2006, S. 262–269. DOI: 10.1109/AHS.2006.5

[14]wikipedia.org. Hill climbing. 2018. URL: https://en.wikipedia.org/wiki/Hill_climbing(besucht am 28. 05. 2018).

[15]wikipedia.org. Picture. 2016. URL: https://en.wikipedia.org/wiki/Butterfly_network#/media/File:Butterfly_Network.jpg (besucht am 25. 05. 2018).

Zusatzquellen für Präsentation

- [ZQ1] http://www.rte.se/blog/blogg-modesty-corex/leon3-32-bit-processor-core/1.5 (12.06.2018)

Fragen?
Danke für Ihre Aufmerksamkeit!