TECHNISCHE UNIVERSITÄT DRESDEN

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

Belegverteidigung

Studie zur Erhöhung der Zuverlässigkeit der SHAP-Mikroarchitektur.

Alexander Simon, s1514488@inf.tu-dresden.de

Dresden, 18.09.2009

Gliederung

1. Aufgabenstellung
2. Einleitung
3. Unterscheidung zwischen Fehlerarten
4. Methoden der Fehlererkennung und -korrektur
5. Anwendung auf SHAP
6. Zusammenfassung

Aufgabenstellung

1. Literaturstudium zu Fehlererkennung und Reparatur von permanenten Hardwarefehlern.
2. Literaturstudium zu Ursachen, Erkennung und Korrektur von transienten Hardwarefehlern.
3. Identifikation von geeigneten Konzepten zur Fehlererkennung und -korrektur und deren Anwendung auf die verschiedenen Klassen von Hardware-Modulen.
4. Auswahl geeigneter Ansätze für die SHAP-Mikroarchitektur. Zusammenstellung verschiedener Szenarien zur Fehlererkennung und -korrektur.
5. Zusammenstellung und Dokumentation der Ergebnisse.

Einleitung

Zuverlässigkeit [1]

- verfügbar, echt, vertrauenswürdig, vertraulich

Zuverlässigkeitsgewinn [1]

- Fehlervorsorge, Fehlertoleranz, Fehlerbeseitigung, Fehlerprognose

Fehlerarten [7]

- permanent
  - → wenn die zugesicherte Eigenschaft oder die geforderte Funktion von der Betrachtungseinheit dauerhaft nicht erfüllt wird
- transient
  - → vorübergehendes Nichterfüllen einer Funktion
- intermittierend
  - → bedingtes Fehlverhalten einer Einheit

Unterscheidung zwischen Fehlerarten

Heuristische Methode [3]

- Fehlerzähler in Modulen
- ab bestimmtem Schwellwert wird ein permanenter Fehler angenommen
- einfache Umsetzung, aber Schwellwert muss gut dimensioniert sein

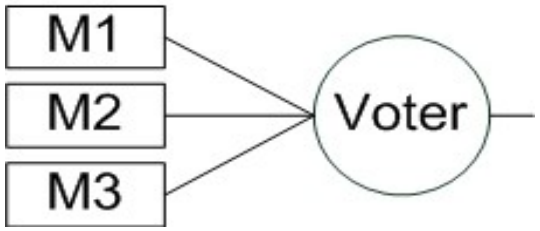
Bayessche Methode [10]

- basiert auf der Bayesschen Gleichung für bedingte Wahrscheinlichkeiten
- durch Beobachtung vorheriger Zustände wird die Wahrscheinlichkeit für das Auftreten eines permanenten Fehlers berechnet
- exaktere Unterscheidung des Fehlertyps möglich

Methoden zur Fehlererkennung und -korrektur

- redundante Systeme
- dynamische Verifikation
- Trace-Based Fehlerdiagnose
- ActiveRedundant-Simultaneous Multithreading
- Dataflow Verification
- Codierungstechniken

Redundante Systeme [6]



- statische Redundanz

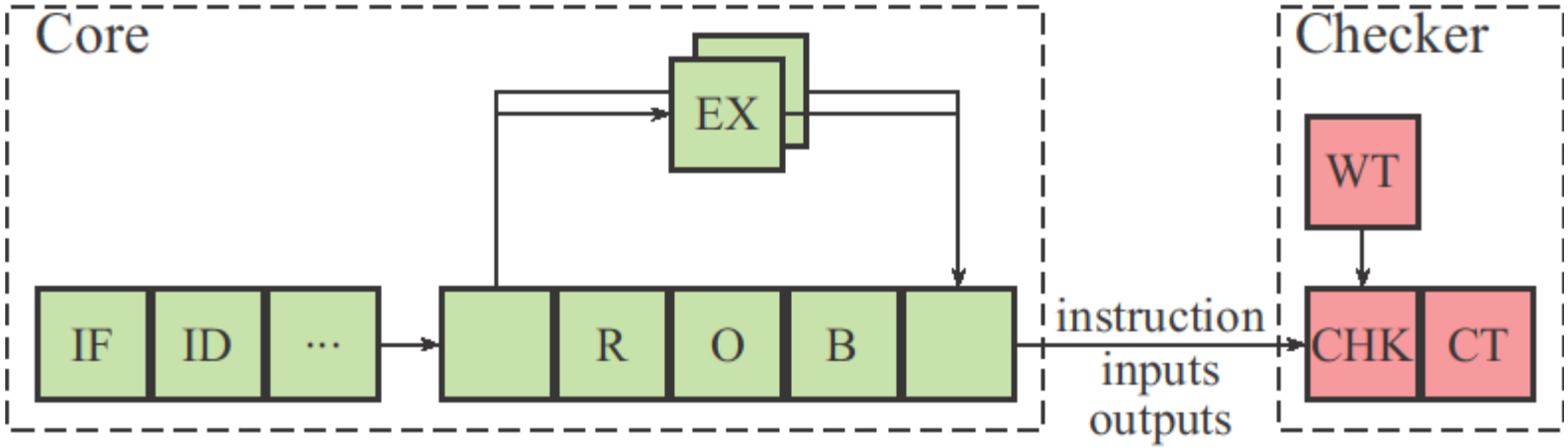
- dynamische Redundanz
  - ein Modul arbeitet und Reserveeinheiten stehen bereit
  - bei entdecktem Fehler wird Reservemodul aktiv

- hybride Redundanz
  - Kombination aus statischer und dynamischer Redundanz
  - bei Ausfall von fehlerhaften Modulen stehen Reserven zur Verfügung

- weitere Arten: self-purging Redundanz, Rekonfigurationsmöglichkeiten

Dynamische Verifikation (DIVA) [2]

- dient zur Detektion und Korrektur permanenter und transienter Fehler, als auch von Designfehlern
- Verifikation aller Ergebnisse mit einer Checkereinheit
- bei detektiertem Fehler wird Pipeline geleert und neu gestartet



Dynamische Verifikation

- Vorteile: - online Verifikation mit geringen Geschwindigkeitseinbußen
  - Fehlerkorrektur möglich
  - jede Berechnung wird am Ende korrekt abgespeichert
  - einfache Implementierung des Checkers
  - geringer Hardwareaufwand wenn Hazards gebilligt werden

- Nachteile: - Strukturhazards wenn Checker auf Speicher zugreifen will
  - keine Fehlererkennung im Checker

Trace-Based Fehlerdiagnose (TBFD) [4]

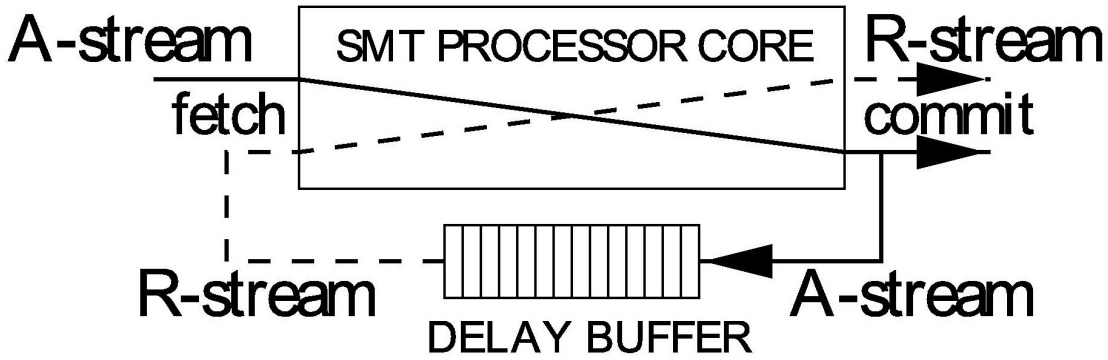
- Detektion permanenter Fehler, welche selten auftreten
- für Multi-Core-Systeme
- bei entdecktem Fehler wird das System auf sicheren Zustand zurückgesetzt und dieser wird auf anderen Kern geladen
- durch Vergleich beider Durchläufe kann das defekte Modul identifiziert werden
- Voraussetzung: Zustände speicher- und wiederherstellbar

- Vorteil: geringer HW-Aufwand

- Nachteil: 2 Cores beschäftigt bei der Fehlerdiagnose

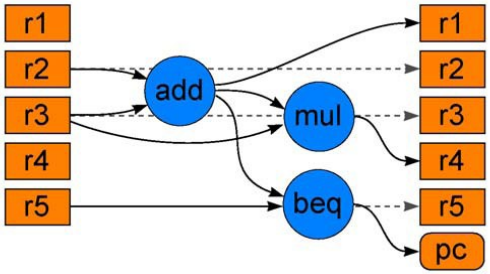
ActiveRedundant-Simultaneous Multithreading [9]

- nutzt zeitliche Redundanz, um transiente Fehler tolerieren zu können
- Thread wird nach bestimmter Zeit noch einmal ausgeführt
- bei unterschiedlichen Ergebnissen wird Thread erneut ausgeführt



Dynamic Dataflow Verification (DDFV) [8]

- Vergleich von dynamischen und statischen Datenflussgraphen
- Erkennung von permanenten, transienten Fehlern und Designfehlern
- Einteilung des Programmes in bekannte Blöcke
- Berechnung der statischen Signaturen durch Compiler
- Berechnung der dynamischen Signaturen während der Ausführung
- Vergleich mittels speziellen Befehlen



Codierungstechniken

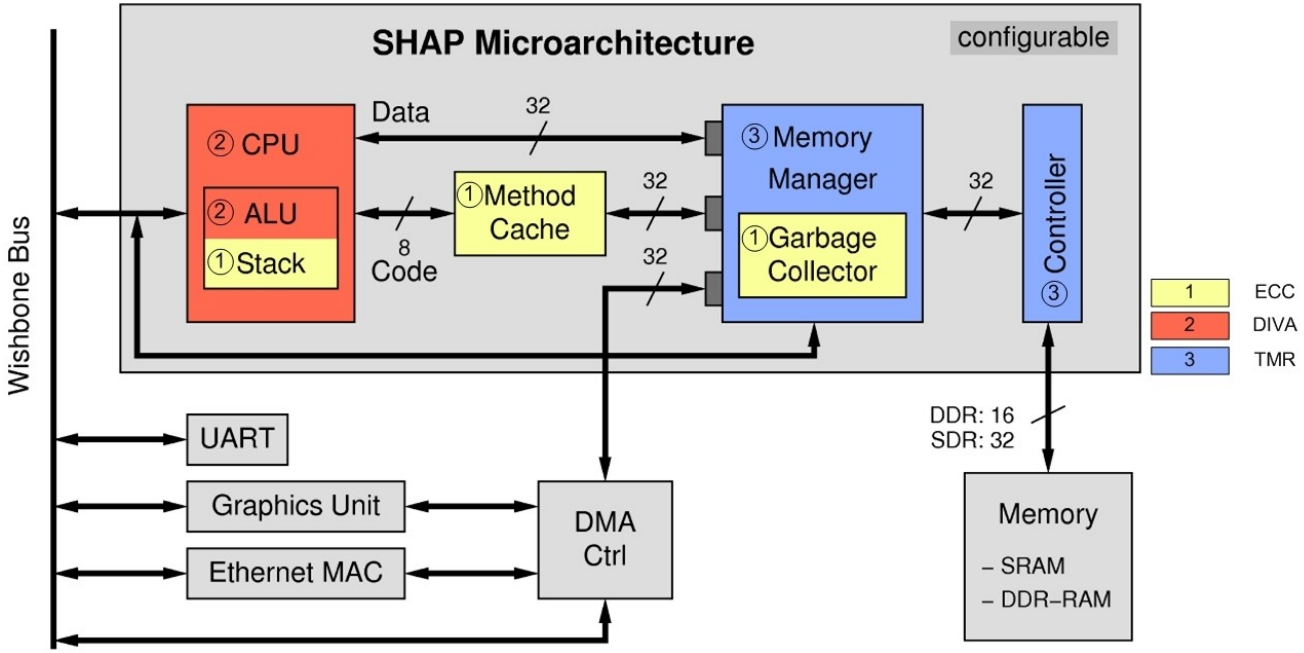
	Hammingdistanz	Anmerkungen
Paritätsprüfung	2	Keine Fehlerkorrektur möglich
Kreuzparität	4	Fehlerkorrektur möglich
CRC	-	Nutzdaten nicht direkt enthalten
Hamming-Code	3 oder 4	Nutzdaten direkt enthalten
Diamond-Code	d1*d2	Doppelte Codierung
RS-Code	n-k+1	Aufwendige Decodierung
Faltungscodes	je nach Struktur	Lange Codewörter
Residue-Code	-	Fehlerschutz in Funktionseinheiten
Crosstalk Avoidance Code	max. 3	Fehlerschutz auf Bussen

Modulzuordnung

	DIVA	TMR	AR-SMT	TBFD	DDFV	Codes
Pipeline	T,P,D	T,P	T	P	T,P,D	-
ALUs	T,P,D	T,P	T	P	-	T,P
FPUs	-	T,P	T	P	-	-
Multiplizierer	T,P,D	T,P	T	P	-	-
Dividierer	T,P,D	T,P	T	P	-	-
Speicher	-	-	-	-	-	T,P
Leitungen	-	-	-	-	-	T,P

Anwendung auf SHAP

Beispielkonfiguration



Anwendung auf SHAP

Änderungen, Ergebnisse

- DIVA: - zusätzliche Logik und Register für neue Pipelinestufen
  - Ausführungseinheiten für DIVA Checker
  - Fehlerzähler und Logik zur Umkonfiguration
  - → aus 4 werden 7 Pipelinestufen
  - → aber durch implementiertes Forwarding kaum Performanceverlust

- TMR: - Bestimmung von Modulen zur Umsetzung
  - Verdreifachung der Module und Implementierung eines Voters

Anwendung auf SHAP

Änderungen, Ergebnisse

- ECC: - Codierer/Decodierer für jeden Speicher
  → Anstieg der Fläche für Speicher um 21,7%

Speicher	Größe ohne Codierung	Größe mit Codierung
Microcode	2,25KB	3,25KB
Stack	8,25KB	9,75KB
ZPU	4,00KB	4,75KB

Zusammenfassung

- verschiedene Methoden zur Steigerung der Zuverlässigkeit wurden aufgezeigt
- bewertet wurde im Hinblick auf Performance und Flächennutzung
- Vergleich der Methoden zeigt, dass eine Kombination mehrerer Techniken sinnvoll ist
  - ➔ bspw. DIVA, TMR, Hamming-Codes, CAC

- mit einer folgenden Implementierung in die SHAP-Architektur könnten die Kosten ermittelt und bewertet werden

Literaturverzeichnis(1)

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell and Vytautas Magnus U. Fundamental concepts of dependability, 2000.

[2] Fred A. Bower, Daniel J. Sorin, and Sule Ozev. A mechanism for online diagnosis of hard faults in microprocessors. In MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture, pages 197–208, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Fred A. Bower, Daniel J. Sorin, and Sule Ozev. Online diagnosis of hard faults in microprocessors. ACM Trans. Archit. Code Optim., 4(2):8, 2007.

[4] Man-Lap Li, Pradeep Ramachandran, Swarup Kumar Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. Trace-based microarchitecture-level diagnosis of permanent hardware faults. In DSN, pages 22–31. IEEE Computer Society, 2008.

Literaturverzeichnis(2)

[5] M. Pizza, Lorenzo Strigini, Andrea Bondavalli, and Felicita Di Giandomenico. Optimal discrimination between transient and permanent faults. In HASE ’98: The 3rd IEEE International Symposium on High-Assurance Systems Engineering, pages 214–223, Washington, DC, USA, 1998. IEEE Computer Society.

[6] Stephen Y. H. Su and Richard J. Spillman. An overview of fault-tolerant digital system architecture. In AFIPS ’77: Proceedings of the June 13-16, 1977, national computer conference, pages 19–26, New York, NY, USA, 1977. ACM.

[7] Bundesamt für Sicherheit in der Informationstechnik. Definitionen und Metriken für die Hochverfügbarkeit. 2009.

Literaturverzeichnis(3)

[8] MEIXNER, A.; SORIN, D. J.: Error Detection Using Dynamic Dataflow Verification. In: PACT ’07: Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques. Washington, DC, USA: IEEE Computer Society, 2007.

[9] ROTENBERG, E.: AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors. In: FTCS ’99: Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing. Washington, DC, USA: IEEE Computer Society, 1999.

[10] PIZZA, M.; STRIGINI, L.; BONDAVALLI, A.; GIANDOMENICO, F. D.: Optimal Discriminationbetween Transient and Permanent Faults. In: HASE ’98: The 3rd IEEE International Symposium on High-Assurance Systems Engineering. Washington, DC, USA: IEEE Computer Society, 1998.

Backup-Folien

Hamming-Code

- (7,4) Hamming-Code → Codewortlänge = 7
  → Datenwortlänge =4
- Bildung der Paritätsbits:

$$P_1 = N_1 \oplus N_2 \oplus N_4$$
$$P_2 = N_1 \oplus N_3 \oplus N_4$$
$$P_3 = N_2 \oplus N_3 \oplus N_4$$

- resultierendes Codewort: $N_1 N_2 N_3 N_4 P_1 P_2 P_3$

Diamond-Code

- besteht aus 2 zusammengesetzten Blockcodes (a und b)
- Verknüpfung zu einer Matrix
- resultierendes Codewort:

$$
\begin{array}{cccc}
a_0 b_0 & & & \\
a_1 b_0 & a_0 b_1 & & \\
a_2 b_0 & a_1 b_1 & \ddots & \\
\vdots & \vdots & & a_0 b_q \\
a_p b_0 & \vdots & & \vdots \\
 & a_p b_1 & & \vdots \\
 & & \ddots & \vdots \\
 & & & a_p b_q
\end{array}
$$

- doppelte Codierung und Decodierung notwendig
- Vereinigung der Eigenschaften beider Teilcodes

Residue-Code

- Moduli sind bekannt: $m_1=3, m_2=5, m_3=2, m_4=7$
  $m_5=6, m_6=8$
- Wahlvorschrift: $m_1 * m_2 = m_3 * m_4 - 1$
  $m_5 \wedge m_6 \notin \{m_1, m_2, m_3, m_4\}$
- Beispiel:

$$\begin{aligned} X &= 5 &&= (2,0,1,5,5,5) \\ Y &= 10 &&= (1,0,0,3,4,2) \\ Z = X + Y &= 15 &&= (0,0,1,1,3,7) \end{aligned}$$

- Fehlerüberprüfung: $|Z|_{m_i} = |\,|m_1 * m_2|_{m_i} * |\gamma_Z|_{m_i}|_{m_i} + |\delta_Z|_{m_i}|_{m_i} \quad , \quad i \in \{5,6\}$

$$\gamma_Z = \left\lfloor \frac{Z}{m_1 * m_2} \right\rfloor$$

$$\delta_Z = |Z|_{m_1 * m_2}$$

Crosstalk-Avoidance-Codes

- zur Verringerung von Übersprechen auf benachbarten Leitungen
  - Verhinderung ungünstiger Bitwechsel durch Forbidden Transition Codes
  - Verringerung des Übersprechens durch Forbidden Pattern Codes
  - Vereinigung beider Methoden mit Overlapping Codes