Karlsruher Institut für Technologie

Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung
Prof. Dr. rer. nat. Wolfgang Karl

Bachelorarbeit

Studienarbeit
von

Nico Rudolf

an der Fakultät für Informatik

Tag der Anmeldung: 1. Mai 2018
Tag der Fertigstellung: 1. September 2018

Aufgabensteller:
Prof. Dr. rer. nat. Wolfgang Karl

Betreuer:
M. Sc. Thomas Becker

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

Karlsruhe, den 01.09.2018

Nico Rudolf

Zusammenfassung

Diese Arbeit beschäftigt sich..

Inhaltsverzeichnis

1 Motivation **1**
1.1 Zielsetzung . . . 2
1.2 State of the Art . . . 2

2 Material und Methoden **3**
2.1 Testplattform . . . 3
2.1.1 HPC Cluster Server . . . 3
2.1.2 Multicore Computer . . . 3
2.2 Metriken als Symptome . . . 3
2.2.1 Hardware Perfomance Counter . . . 3
2.2.2 Perfomance Application Programming Interface . . . 3
2.3 Physikalische Fehlerinjektion . . . 3
2.3.1 Simulation von defekten CPU-Cores . . . 4
2.3.2 Simulation von unzureichender Spannungsversorgung . . . 4
2.3.3 Manipulation der Taktfrequenz . . . 4
2.3.4 Manipulation der Speicherfrequenz . . . 4
2.4 Softwarebasiere Fehlerinjektion . . . 4
2.4.1 Fehlerinjektion in POSIX Bibliothek . . . 4
2.4.2 Fehlerinjektion im Linux Kernel . . . 4
2.4.3 Fehlerinjektion zur Laufzeit . . . 4
2.5 Benchmarks . . . 4
2.5.1 Rodinia Benchmark Suite . . . 5
2.5.2 Eigene Benchmarks . . . 5

3 Durchführung der Messungen **7**
3.1 Implementierung der Messungen . . . 7
3.2 Analyse . . . 7
3.3 Messergebnisse . . . 7

4 Evaluation **9**
4.1 Vergleich der Ausführungsmetriken . . . 9
4.2 Indizierung von Fehlerklassen . . . 9
4.3 Anwendbarkeit der symptombasierten Fehlererkennung . . . 9

5 Anwendung der symptombasierten Fehlererkennung 11
5.1 Schematische Implementierung 11
5.2 Anwendungsfall 11

6 Zusammenfassung und Ausblick 13

Tabellenverzeichnis

Abbildungsverzeichnis

1 Motivation

Mit der stetig wachsenden Komponentenzahl in modernen Rechensystemen geht eine Steigerung der Ausfallwahrscheinlichkeit und Fehleranfälligkeit der Komponenten einher. Eine Möglichkeit die Verlässlichkeit von Rechensystemen zu steigern, ist die Vermeidung von Fehlern während der Berechnung. Klassische Methoden der Fehlertoleranz wie Checkpoints oder Redundanz sind allerdings in modernen Systemen aufgrund der Vielzahl der Komponenten ineffizient und zu teuer. Deshalb ist ein alternativer Ansatz der Verzicht auf die Fehlervermeidung und auf die Erkennung von Fehlern zu setzen. Man unterscheidet hierbei zwischen transienten Fehlern (soft errors) und intransienten Fehlern (hard errors). Soft errors sind von zufälliger und temporärer Natur, das bedeutet, dass ein soft error während einer Berechnung zu einem fehlerhaftem Ergebnis führen kann, bei Wiederholung der Berechnung wird dieser Fehler wahrscheinlich jedoch nicht auftreten. Fehler die dauerhaft existieren, werden als hard errors bezeichnet. Beispiele für hard errors sind Hardware Ausfälle, denn ein Rechensystem würde unter diesen Umständen dauerhaft falsche Ergebnisse liefern. Zurückzuführen sind transiente Fehler auf kosmische Partikel oder elektromagnetische Störfelder.

Diese verursachen unter anderem ungewollte Bitflips in den Registern des Prozessors oder in den Speicherzellen des Hauptspeichers. Das Systeme immer häufiger von transienten Fehlern betroffen sind, ist im wesentlichen auf die niedrigeren Versorgungsspannungen zurückzuführen. Hintergrund dafür ist der Wunsch nach energieeffizienten Produkten aber auch die zunehmende Integrationsdichte.

Neben der Verfälschung von Berechnungsergebnissen und dem Hervorrufen von kritischen Systemfehlern, die das System zum Absturz bringen, kann ein transienter Fehler auch folgenlos bleiben. Um Computersysteme vor solchen Fehlern zu schützen, gibt es mehrere Ansätze, wie die Redundanz. Zum Einen gibt es die temporale Rendundanz, bei der eine Berechnung auf dem gleichen System mehrmals ausgeführt wird. Das berechnete Ergebnis wird dann verglichen und falsche Ergebnisse werden erkannt. Allerdings bedeutet das, dass die Kosten (Zeit) sich mindestens verdoppeln. Räumliche Redundanz wird durch die parallele Ausführung auf mehreren (identischen) Rechensystemen realisiert. Beispielsweise benutzt das Space Shuttle fünf Computersysteme, um so per Mehrheitsentscheid bestimmen zu können. Die (Hardware-) Kosten steigen hierbei extrem an, allerdings ist es im Gegensatz zur temporalen Redundanz möglich, nicht nur soft errors sondern auch hard errors erkannt werden. Eine Alternative zur teuren Redundanz, stellt eine leichtgewichtige Methode zur Erkennung von aufgetretenen Fehlern,

wie die Symptom-basierte Fehlererkennung, dar. Die Grundidee ist die Erkennung von Fehlern mittels Abweichungen vom üblichen Ausführungsverhalten. Das Verhalten wird mittels Metriken, sogenannten Performance Counter, wie z.B. die Anzahl an TLB Misses, charakterisiert. Zur Festlegung des normalen Ausführungsverhalten wird eine Datenbank mit Werten verschiedener Ausführungsmetriken von korrekten Ausführungen angelegt. Daraufhin werden Ausführungen gezielt mit Fehlern injiziert um die Metriken auf signifikante Abweichungen zu untersuchen. Zur Laufzeit werden dann die aktuellen Werte mit der Datenbank verglichen. Sofern die Abweichungen einer oder mehrerer Metriken einen festgesetzten Grenzwert überschreiten könnte dies als Indikator für diesen Fehler gelten. Ziel dieser Bachelorarbeit ist die Untersuchung der Anwendbarkeit der Symptombasierten Fehlererkennung. Dabei wird gezeigt ob das Auftreten von Fehlern oder sogar die aufgetretene Fehler(-art) bestimmt werden können.

1.1 Zielsetzung

Ziel dieser Bachelorarbeit ist die Untersuchung der Anwendbarkeit der Symptombasierten Fehlererkennung. Die Grundidee ist die Erkennung von Fehlern mittels Abweichungen vom üblichen Ausführungsverhalten. Das Verhalten wird mittels Metriken, sogenannten Performance Counter, wie z.B. die Anzahl an TLB Misses, charakterisiert. Zur Festlegung des normalen Ausführungsverhalten wird eine Datenbank mit Werten verschiedener Ausführungsmetriken von korrekten Ausführungen angelegt. Daraufhin werden Ausführungen gezielt mit Fehlern injiziert um die Metriken auf signifikante Abweichungen zu untersuchen. Zur Laufzeit werden dann die aktuellen Werte mit der Datenbank verglichen. Sofern die Abweichungen einer oder mehrerer Metriken einen festgesetzten Grenzwert überschreiten könnte dies als Indikator für diesen Fehler gelten. Dabei wird gezeigt ob das Auftreten von Fehlern oder sogar die aufgetretene Fehler(-art) bestimmt werden können.

1.2 State of the Art

noch hinzuzufügen , änderung

2 Material und Methoden

2.1 Testplattform

2.1.1 HPC Cluster Server

2.1.2 Multicore Computer

TEST

2.2 Metriken als Symptome

TEST

2.2.1 Hardware Perfomance Counter

TEST

2.2.2 Perfomance Application Programming Interface

TEST

2.3 Physikalische Fehlerinjektion

TEST

2.3.1 Simulation von defekten CPU-Cores

TEST

2.3.2 Simulation von unzureichender Spannungsversorgung

TEST

2.3.3 Manipulation der Taktfrequenz

TEST

2.3.4 Manipulation der Speicherfrequenz

TEST

2.4 Softwarebasiere Fehlerinjektion

2.4.1 Fehlerinjektion in POSIX Bibliothek

TEST

2.4.2 Fehlerinjektion im Linux Kernel

TEST

2.4.3 Fehlerinjektion zur Laufzeit

TEST

2.5 Benchmarks

TEST

2.5.1 Rodinia Benchmark Suite

TEST

2.5.2 Eigene Benchmarks

TEST

3 Durchführung der Messungen

TEST

3.1 Implementierung der Messungen

TEST

3.2 Analyse

Test

3.3 Messergebnisse

TEST

4 Evaluation

TEST

4.1 Vergleich der Ausführungsmetriken

TEST

4.2 Indizierung von Fehlerklassen

TEST

4.3 Anwendbarkeit der symptombasierten Fehlererkennung

TEST

5 Anwendung der symptombasierten Fehlererkennung

TEST

5.1 Schematische Implementierung

TEST

5.2 Anwendungsfall

TEST

6 Zusammenfassung und Ausblick

TEST

Literaturverzeichnis