Karlsruher Institut für Technologie

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

Symptombasierte Fehlererkennung in heterogenen Systemen

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

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

In diesem Kapitel werden die Materialien und Methoden vorgestellt mit denen die Messungen durchgeführt werden.

2.1 Testplattform

2.1.1 HPC Cluster Server

2.1.2 Multicore Computer

TEST

2.2 Metriken als Symptome

Als Metriken werden in dieser Bachelorarbeit definierte Aktivitäten wie beispielsweise Cache Zugriffe interpretiert. Diese Aktivitäten werden mithilfe von Hardware Perfomance Countern dokumentiert und als Symptome behandelt. Das Perfomance Application Programming Interface (kurz: PAPI) dient dem leichteren Auslesen dieser Perfomance Counter.

2.2.1 Hardware Perfomance Counter

Hardware Perfomance Counter sind spezielle Mehrzweckregister die in modernen Prozessoren eingebaut sind. Diese Register werden von der on-chip Performance Monitoring Unit (kurz: PMU) verwaltet und beschrieben. [1] Das Ziel der PMU ist das Zählen und somit Dokumentieren von Events, wie z.B. Zyklen, Cache Misses oder Floating Point Operations, die im folgenden als Symptome definieren.

Diese Events können in Hardware Events (CPU-Cycles, Instructions, Cache Zugriffe etc,) und Software Events (Page Faults, Context Switche,..) unterschieden werden.

Im Allgemeinen besteht die Hardware zum Zählen der Events aus zwei Komponenten:

Zum einen die Perfomance Event Select Registers

Diese geben an welche Events überwacht und wie gemessen wird. Und Event Counters, die als Zielregister für die Werte fungieren.

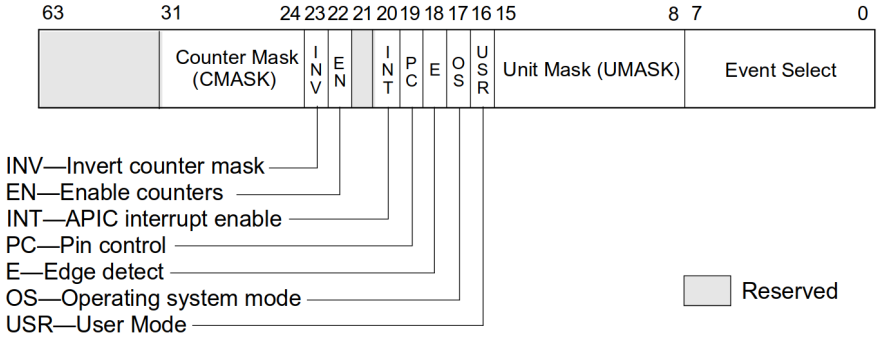

Abbildung 2.1: Intel IA32PERFEVTSELxMSRs (https://software.intel.com/sites/default/files/managed/3 sdm-vol-1-2abcd-3abcd.pdf)

Man unterscheidet gemäß Intel zwischen drei Typen von Perfomance Countern:

- Fixed function counters
- General purpose perfomance counters
- Precise-event based sampling

Entwicklung der Perfomance Counter

Ein Pentium III Prozessor (1999) besitzt zwei Hardware Perfomance Counter, wohingegen ein aktueller Prozessor wie der Intel Xeon E5-2620 (2014) 634 Perfomance Counter zur Verfügung stellt, wie der Tabelle im Anhang entnommen werden kann. [2]

Perfomance Counter werden üblicherweise benutzt um in Abhängigkeit einer Anwendung Hardware auf Engpässe, sogenannte Bottle Necks zu untersuchen. Bottle Necks können typischerweise zu geringer Speicher, geringer Netzwerkdurchsatz oder eine überlastete CPU sein. [3]

Ein Beispiel für die klassische Verwendung ist folgender Programmcode.

Bei einem unoptimierten Compiler, würden die benachbarten Datenelemente nie nacheinander zugegriffen werden. Messungen an der NCSA Illinois zeigen, dass dies zu einer höheren Anzahl von L2-Cache Misses (212 665 026) führt.

Wenn man allerdings die verschachtelten Schleifen umändert, dass die Matrizen row-by-row basiert durchgegangen werden würde. was der Intel ICC Compiler macht, dann

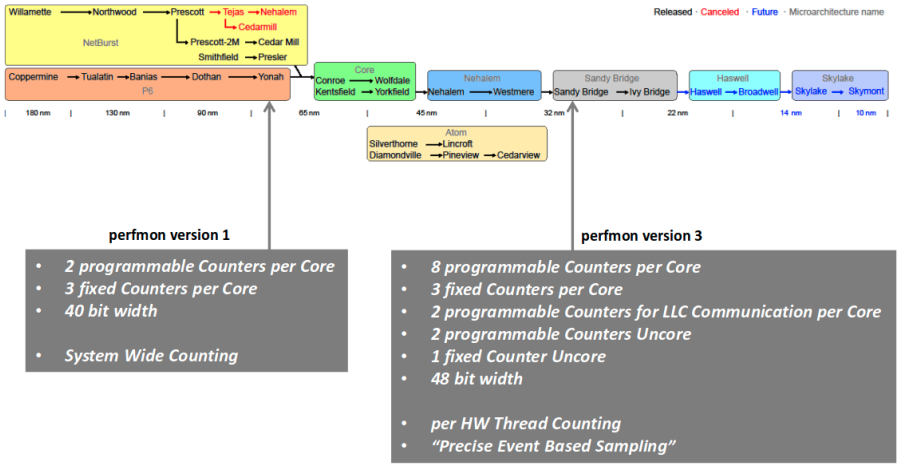

Abbildung 2.2: https://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring13/slides/11-perfcounters.pdf

```
for (j = 0; j < COLS; j++)
    for (i = 0; i < ROWS; i++)
        a[i][j] = b[i][j] + c[i]
```

Abbildung 2.3: http://perfsuite.ncsa.illinois.edu/publications/LJ135/x35.html

sinken um 11,89 Prozent die L2-Cache Misses auf 25 287 572. Gleichzeitig erfolgt die Berechnung 10-Mal so schnell. [4]

2.2.2 Perfomance Application Programming Interface

Die University of Tennessee entwickelt das Performance Application Programming Interface (kurz: PAPI) um einen leichteren Zugriff auf perfomance counter anzubieten. Das einheitliche PAPI-Interface bietet Entwicklern eine Abstraktion der nativen Interfaces und somit eine vereinfachte Entwicklung. PAPI unterstützt alle aktuellen Intel und AMD x86 sowie x8664 Prozessoren (CPU). Durch die Erweiterung PAPI CUDA wird der Zugriff auf perfomance counter von nVidia Grafikkarten (GPU) erweitert. Es wird zwischen high-level interface sowie low-level interface unterschieden. Das high-level interface bietet einen einfachen sowie schnellen Zugriff, wohingegen das low-level interface mehr Kontrolle und einen detailliertes Vorgehen erlaubt. In dieser Arbeit wird vor allem das low level interface benutzt. Informationen über die Implementierungsarbeit ist in Abschnitt 4 zu finden. Jede Plattform (CPU, GPU,..) besitzt sogenannte native events, die aus den unterstützen hardware Performance Countern abgeleitet werden. Die Namen dieser events

sind plattformabhängig. Durch generische preset events abstrahiert PAPI diese nativen events und ermöglicht dem Entwickler eine einheitliche Sicht ohne exakte Kenntnisse der native events zu fordern. Sofern ein preset event nicht verfügbar ist versucht PAPI dieses von anderen events abzuleiten, z.B. Total Number of L2 Cache Misses = L2 Cache Access - L2 Cache Hits. Die Konfiguration der Events wird durch EventSets ermöglicht. Diese bestehen aus einem oder mehreren Events und klassifizieren wie die events gemessen werden sollen, z.B. ob der code im user space oder kernel space ablaufen soll.

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 Messungen

TEST

3.1 Pretests

TEST

3.2 Analyse

Test

3.3 Durchführung der Messungen

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

[1] *Perfomance Monitoring Unit WWW Homepage*. – http://rts.lab.asu.edu/web_438/project_final/Talk%209%20Performance%20Monitoring%20Unit.pdf

[2] *Performance Counter - Non- Uniform Memory Access Seminar WWW Homepage*. 2014. – https://www.dcl.hpi.uni-potsdam.de/teaching/numasem/slides/NUMASem_Performance_counter.pdf

[3] *Know-how: Windows Perfomance Counter WWW Homepage*. 2005. – https://www.tecchannel.de/a/know-how-windows-performance-counter,402403

[4] *Using Perfomance Counters to Measure Application Characteristics WWW Homepage*. 1996. – http://perfsuite.ncsa.illinois.edu/publications/LJ135/x35.html