Umgebung für automatisierte Tests von Dateisystemen auf NAND-Flash

Pascal Pieper

18. November 2016

Inhaltsübersicht

Übersicht

Problemstellung

Computergestützte Raumfahrt

- Besteht aus vielen Teilsystemen
  - Fokus der Arbeit liegt auf Permanentspeicher
- Messergebnisse
- Instruktionen
- Computerprogramme

  → Vieles muss zwischengespeichert werden

Problemstellung

Negative Einflüsse auf Speicher

- Erschütterungen
- Harte Strahlung
- große Temperaturschwankungen
- Temperaturabführung schwer

Problemstellung

Abhilfe

- Strahlungsresistente oder robuste Speicher
  - Hohe Anschaffungskosten

Billige Speicher benutzen

- Fehleranfälligkeit durch Dateisystem ausgleichen
- Eine Logik optimiert Lebensdauer und Zuverlässigkeit

Nutzen

Günstiger Speicher im Weltraum!

Voraussetzungen

Software

- Angepasstes Dateisystem für Flash
  - Hohe Fehlererkennungsrate
  - Robust gegenüber Ausfällen
  - Redundanz
  - Abnutzung einzelner Stellen vermindern
- Steuerlogik
  - Ab- und Zuschalten von Speicherchips
  - Überwachung der Fehlerrate
  - Anpassung von Parametern des Dateisystems

Ausführung

Wie passendes Dateisystem aussuchen?

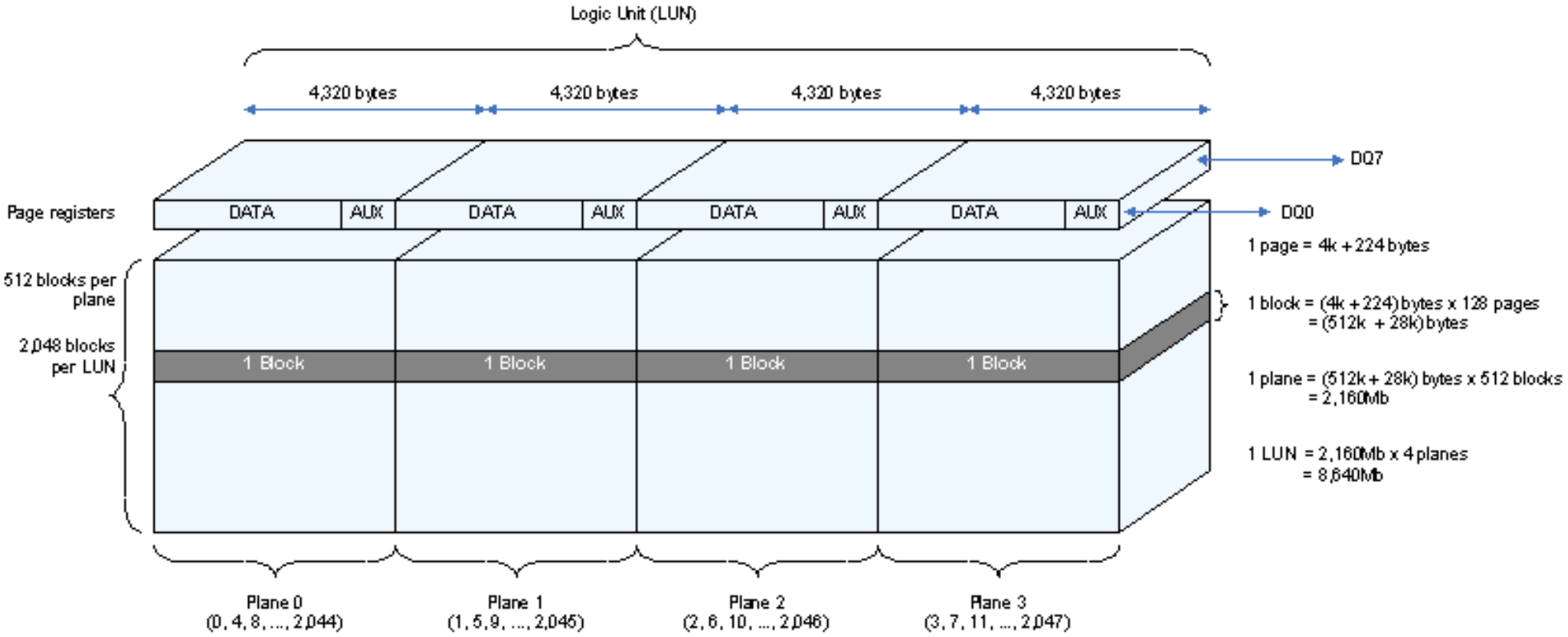
→ Testumgebung, die einheitliche Bewertung einführt

Übersicht

NAND-Flash

- Kein wahlfreier Zugriff
- Geringe Lebensdauer
- Ionen können leicht Informationen verändern
- Evtl. ab Werk defekte Blöcke

NAND-Flash



Aufbau eines Flashspeichers mit beispielhaften Werten (8 MiB)

Fehlerursachen

Unterstützte Fehlerursachen

- *Single Event Effects*
  - *Single Event Upsets*
  - *Single Event Latchups*
- Total Ionizing Dose
- Abnutzung
- *Bad Blocks* ab Werk

Übersicht

Dateisysteme

Strategien

- Fehlererkennende /-korrigierende Codes
- *Wear leveling*
- Redundanz durch RAID oder vergleichbaren Methoden
- *Journaling*
- Echtzeitfähigkeit

Dateisysteme

Viele mögliche Dateisysteme

- RTFFS
- NAMU
- RCFFS
- FAT
- YAFFS
- ...

Teilweise interessante Strategien!

Dateisysteme

Für Test ausgewählt

- FAT (file allocator table)
- YAFFS (yet another flash file system)

Dateisysteme

FAT

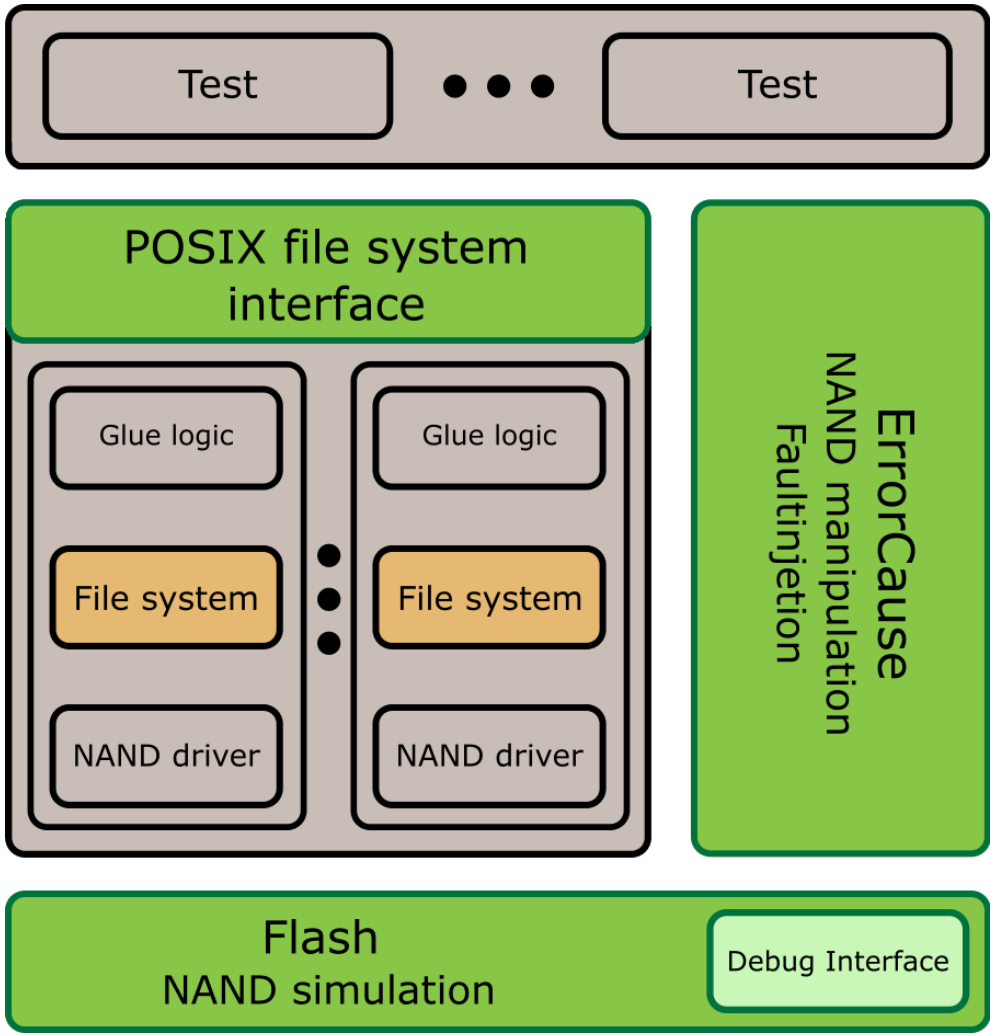
- 1981 für HDDs entwickelt
- Einzelne Tabelle weist Dateien zu Clusternummern
- \- Schlecht für NAND-Flash
- \+ Einfache Struktur, gute Basis für Vergleich

Dateisysteme

YAFFS

- 2002 speziell für NAND-Flash entwickelt
- Position von Objekten muss bei mount ermittelt werden
  - \- Kann nur einen Bitfehler pro Page korrigieren
  - \+ Sehr weit verbreitet, betreibt *wear leveling* und *journaling*

Übersicht



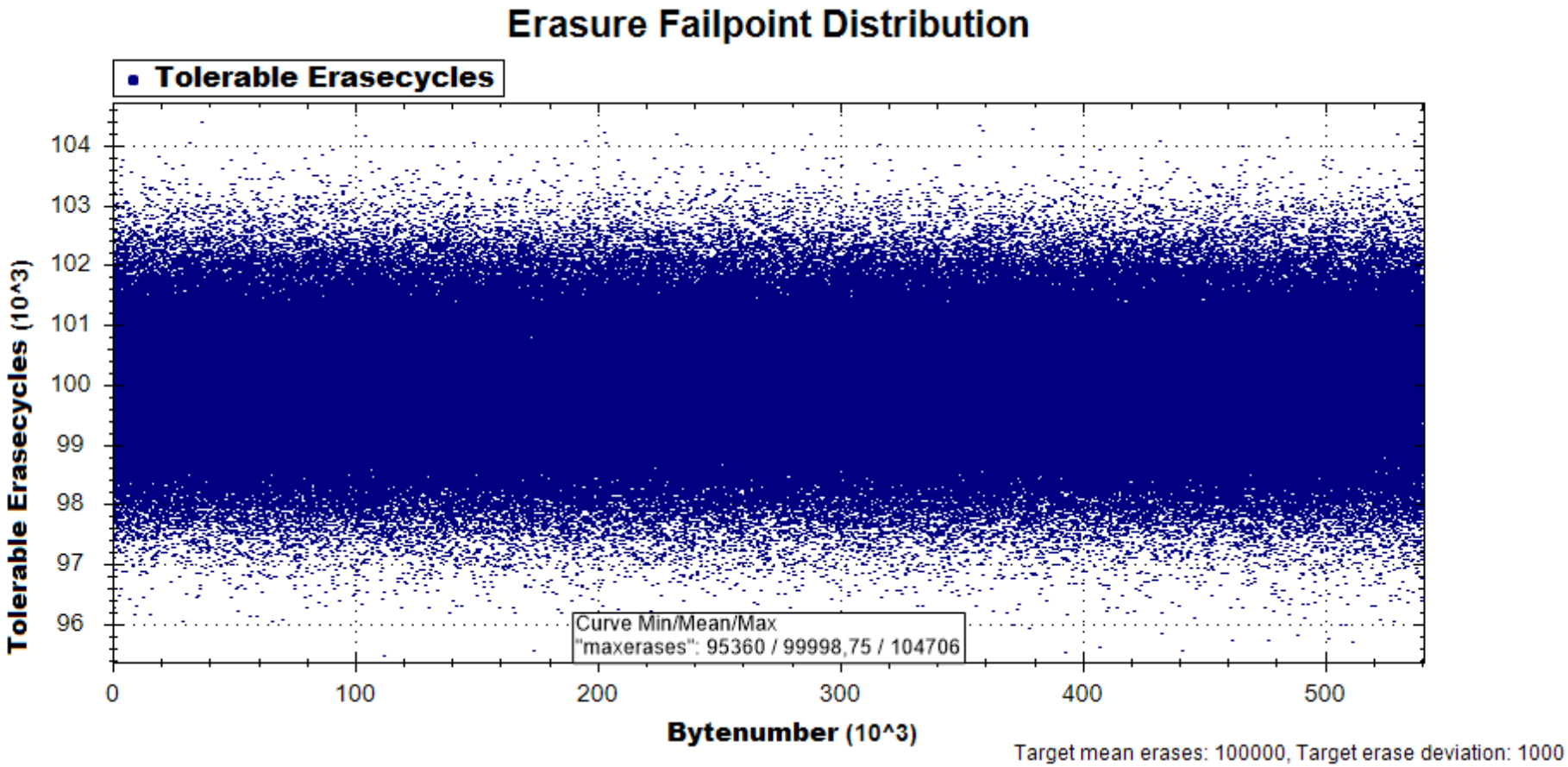
Aufbau Komponenten der Testumgebung und die Position der benutzerdefinierten Modulen

Simulationsumgebung - NAND-Simulation

NAND-Modell

- Klassen der NAND Struktur angepasst
- Zur Instanzierung bestimmbare Parameter für Ausfallpunkte
- Abnutzung (Anzahl Löschvorgänge), verkraftbare *TID*
- . . . und deren Verteilung (statisch, Normalverteilung)

Simulationsumgebung - NAND-Simulation



Beispielverteilung der vordefinierten Abnutzungspunkte

Simulationsumgebung - NAND-Simulation

Fehler im NAND-Modell

- Durch *DebugInterface* gesteuert
  - Kann *SEUs, SELs* injizieren und *TID* erhöhen
  - Kann auch Werte auslesen, ohne Abnutzung zu verändern

  $\rightarrow$ Serialisierung, Auswertung

Simulationsumgebung - Fehlerinjektion

Fehlerursache *ErrorCause*

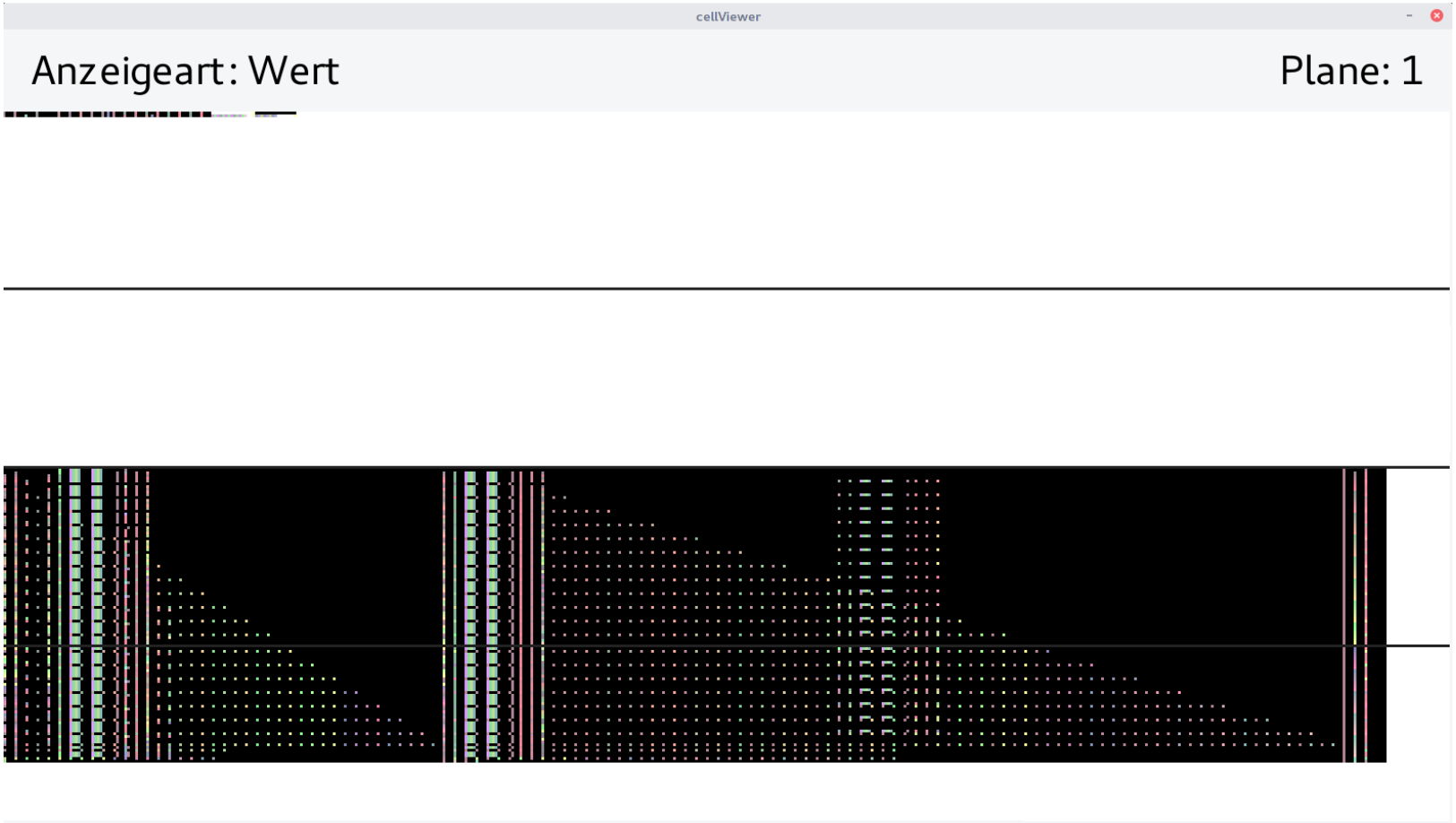
- Zunächst nur abstrakte Klasse
- Von Test gesteuert, manipuliert Speicher durch *DebugInterface*
- Verschiedene Implementationen
  - *set fabrication defect*
  - *increase TID*
  - *induce SEL*
  - *induce SEU*

Simulationsumgebung - Tests

Testklasse

- Abstrakte Klasse
- Besitzt Funktionen, die von Steuerungssystem aufgerufen werden
  - *Name* und *Beschreibung*
  - *initTest*
  - *startTest*
  - *serializeResultData*

Simulationsumgebung - Visualisierung



Ein Beispiel der Visualisierung während eines Tests

Implementierte Tests

Tests

- Abnutzung
- Bitkipper
- Ausfall einzelner Zellen
- Erhöhte *TID*
- Werksseitige *Bad Blocks*
- Dateigröße
- Anzahl Dateien

Ablauf - Abnutzung

Wiederholen, solange keine Fehler

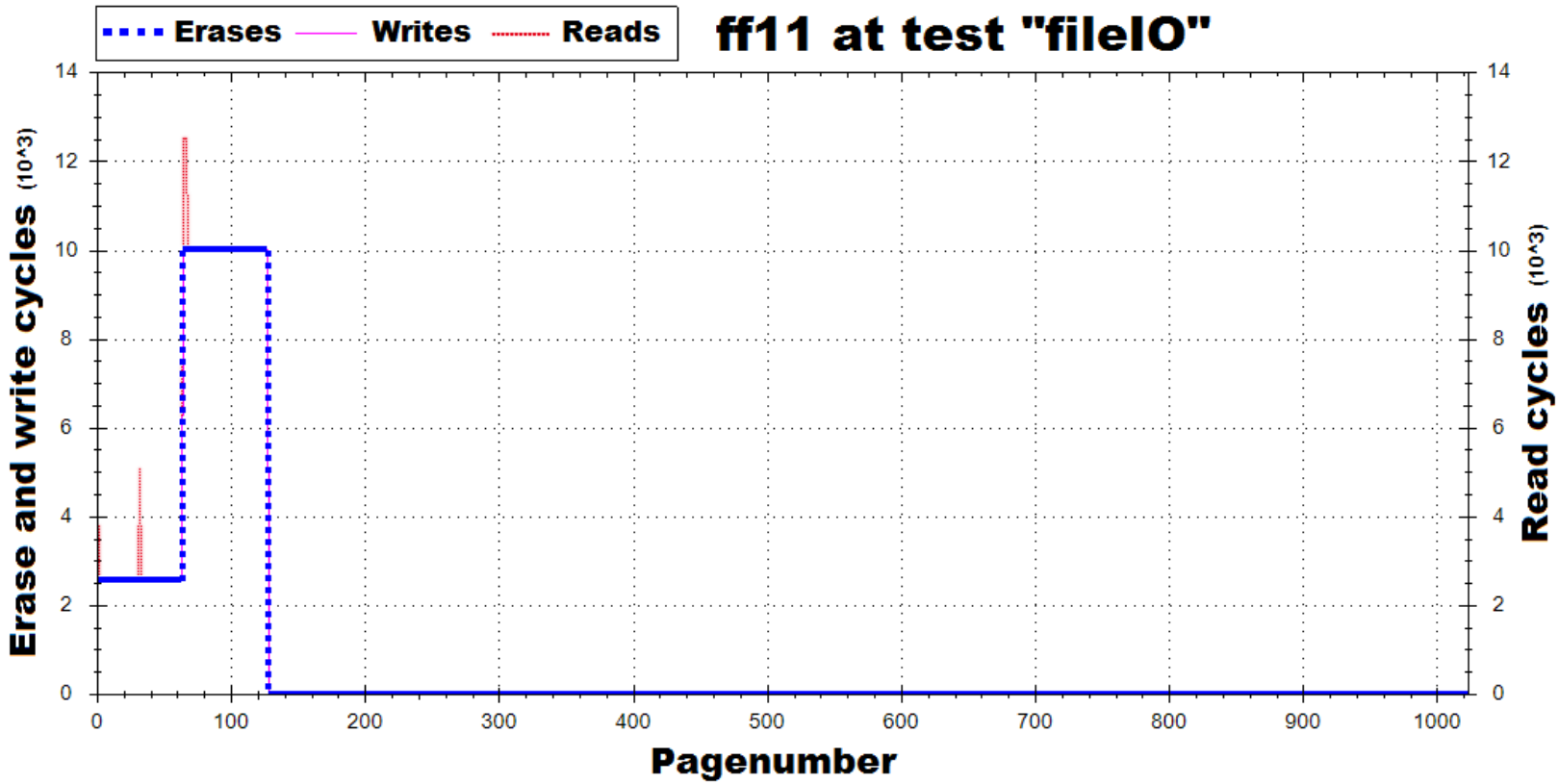
- Zufällige Daten schreiben
- Auslesen
- Verifizieren

Ablauf - Bad Blocks

- Generierung des Speichers mit defekten *bitlines*
- Dateisystem formatiert Speicher
- Dateisystem teilt freien Speicher mit
- Freier Speicher wird gefüllt und auf Fehler überprüft

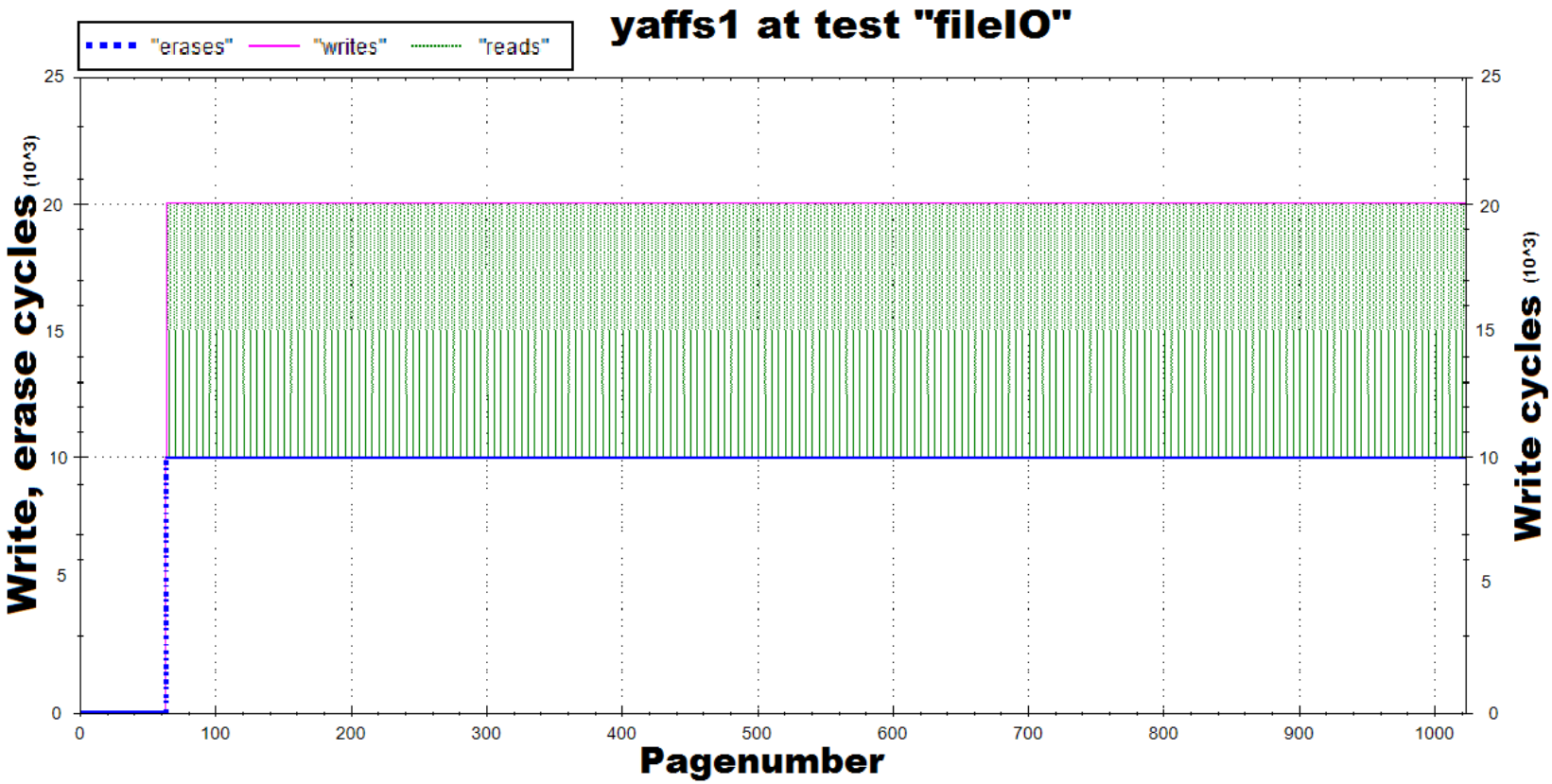
Übersicht

Abnutzung - FAT



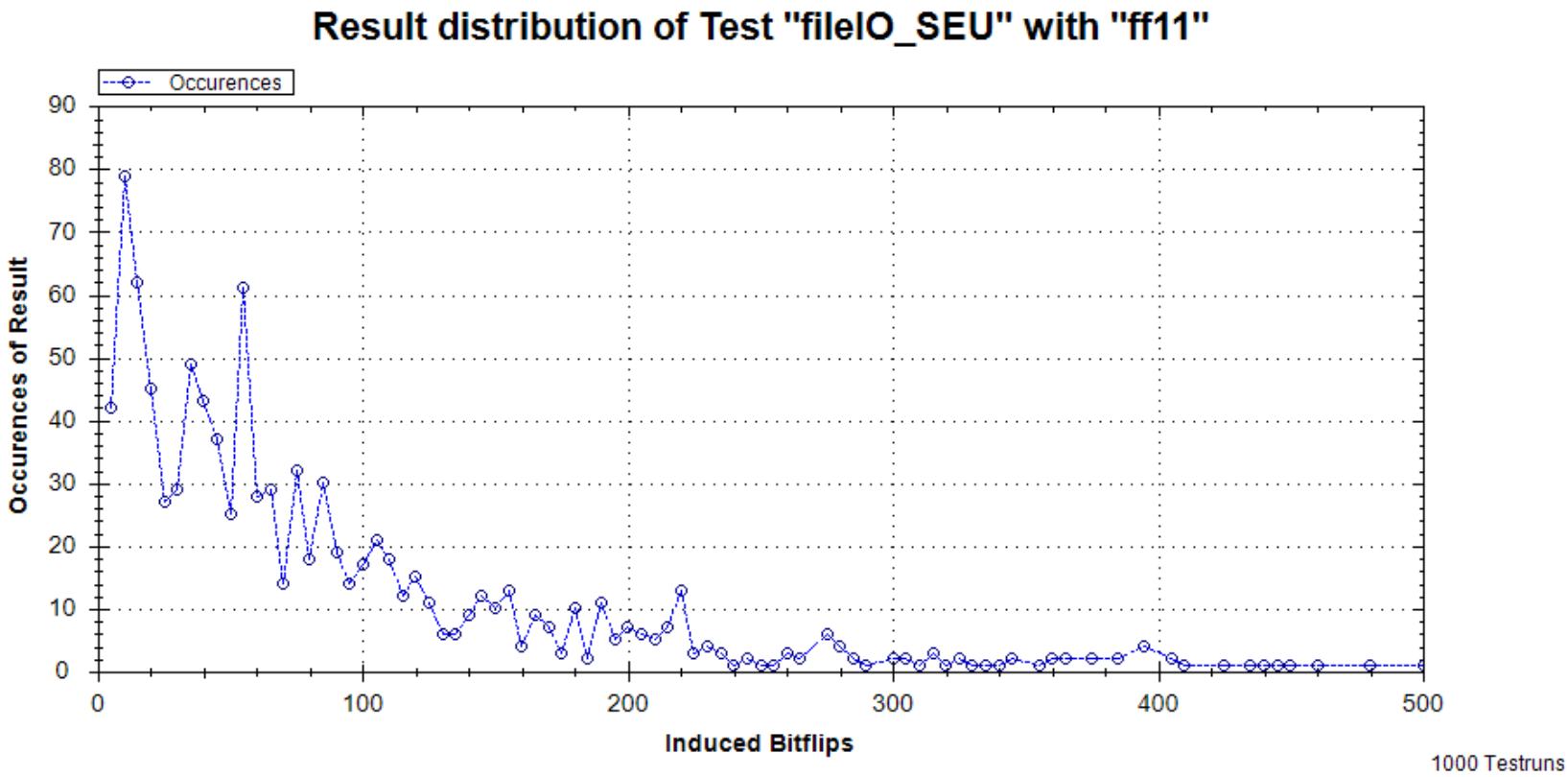
Visualisierter NAND-Speicher nach Integritätsverlust bei 2.500 Zugriffen

Abnutzung - YAFFS1



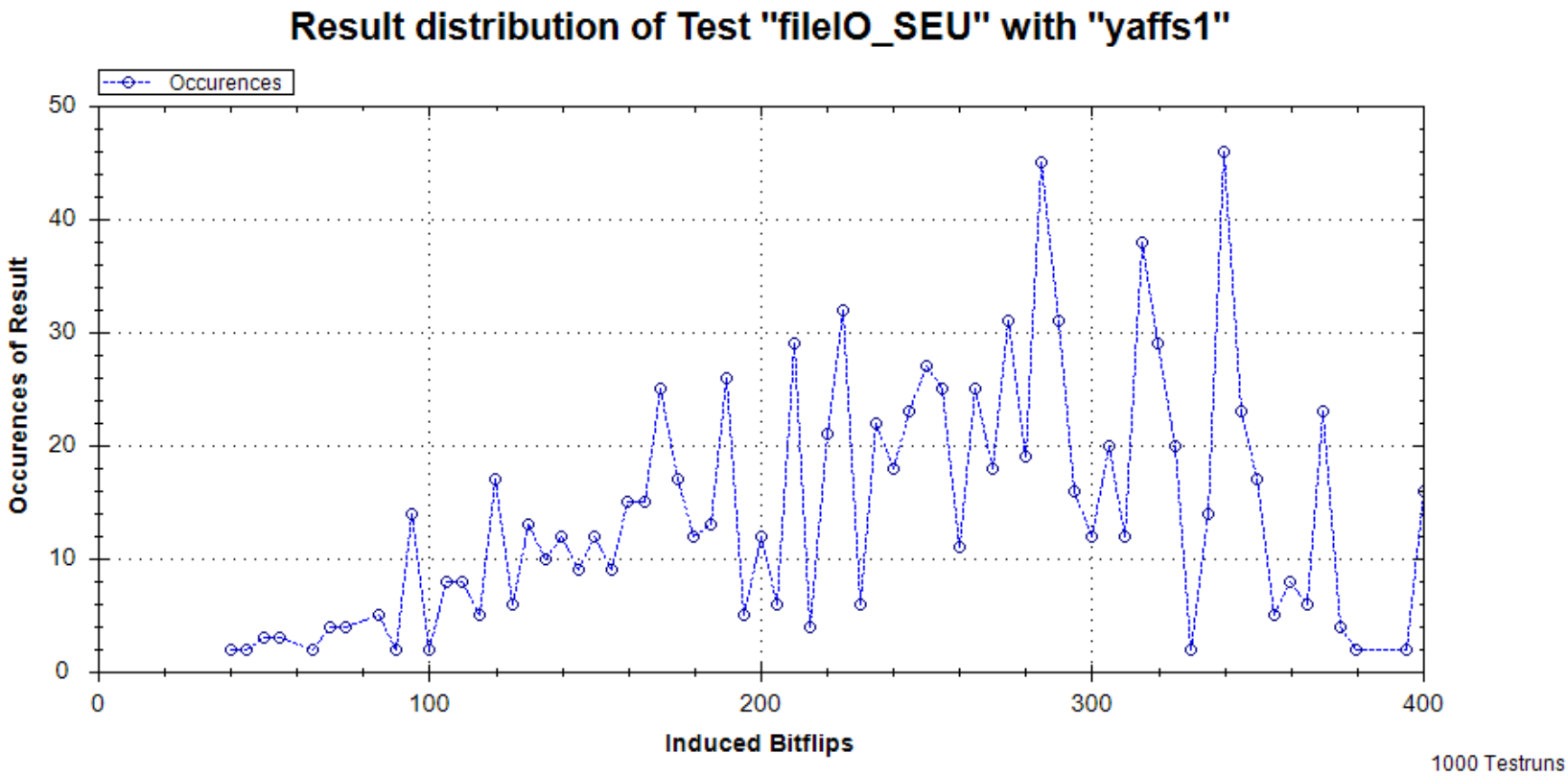
Visualisierter NAND-Speicher nach Integritätsverlust bei 1.920.190 Zugriffen

Bitkipper - FAT



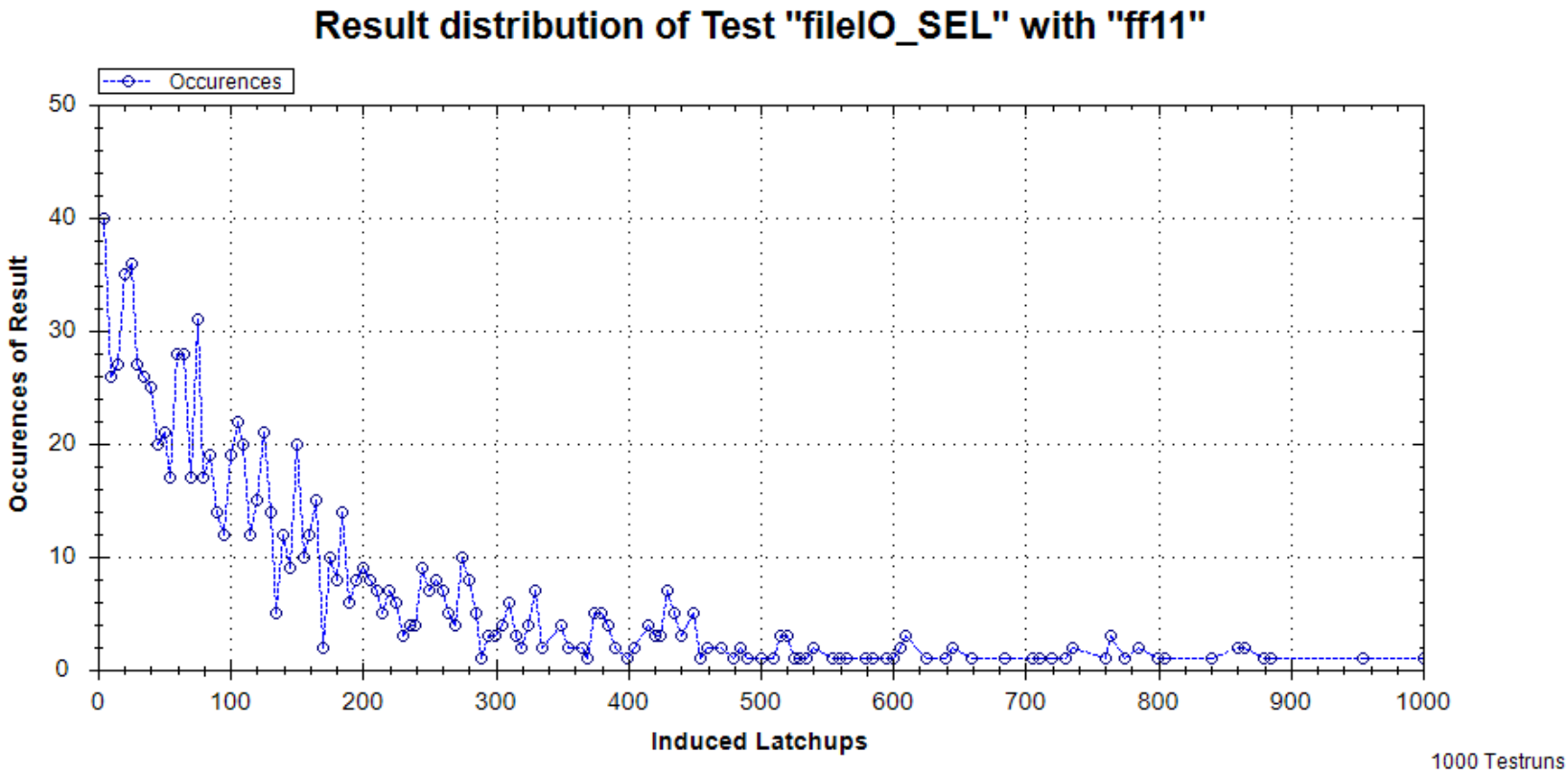
Verteilung der Ergebnisse nach 1000 Testläufen

Bitkipper - YAFFS1



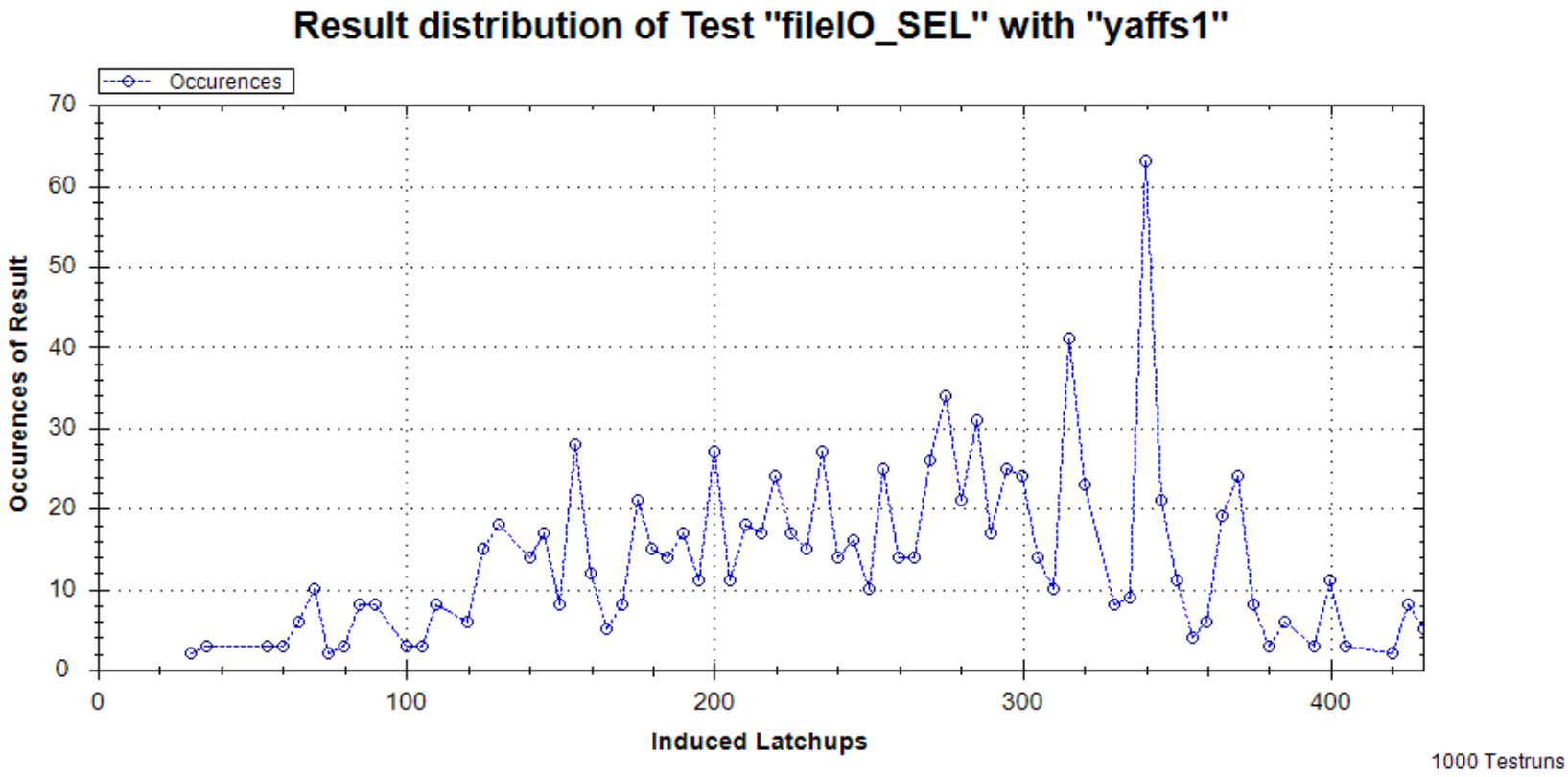
Verteilung der Ergebnisse nach 1000 Testläufen

Permanente Bitfehler - FAT



Verteilung der Ergebnisse nach 1000 Testläufen

Permanente Bitfehler - YAFFS1



Verteilung der Ergebnisse nach 1000 Testläufen

Maximale Dateigröße

FAT
480 KiB

YAFFS1
448 KiB

Erstellbare Dateien

FAT
512 Dateien

YAFFS1
447 Dateien

Auswertung

Vergleich getesteter Dateisysteme

- YAFFS1 deutlich fehlertoleranter und langlebiger
  - → Auf NAND abgestimmt
- FAT speichert effizienter

Übersicht

Zusammenfassung

- Simulationsumgebung entwickelt
  - Fehlerklassen strukturiert
  - Testszenarien implementiert
  - Zwei Beispieldateisysteme integriert
  - NAND-Flash Simulation
  - Visualisierung implementiert
- Dateisysteme verglichen
- Für Raumfahrtszenario empfohlene Dateisysteme modelliert

Ausblick

Optimierungsmöglichkeiten

- Tests und Fehlerklassen trennen
- Größere Testbibliothek
- Tests in Skriptsprache
- Externes Interface für Dateisysteme
- (Abstrakte) Zeit simulieren