# Context Switch Overheads for Linux on ARM Platforms


Francis David fdavid@uiuc.edu
Jeffrey Carlyle jcarlyle@uiuc.edu
Roy Campbell rhc@uiuc.edu

http://choices.cs.uiuc.edu

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

# Outline

- What is a Context Switch?
  - Overheads
  - Context switching in Linux
- Interrupt Handling Overheads
- ARM Experimentation Platform
- Context Switching
  - Experiment Setup
  - Results
- Interrupt Handling
  - Experiment Setup
  - Results

# What is a Context Switch?

- Storing current processor state and restoring another
- Mechanism used for multi-tasking
- User-Kernel transition is only a processor mode switch and is not a context switch

# Sources of Overhead

- Time spent in saving and restoring processor state
- Pollution of processor caches
- Switching between different processes
  - Virtual memory maps need to be switched
  - Synchronization of memory caches
- Paging

# Context Switching in Linux

- Context switch can be implemented in userspace or in kernelspace
- New 2.6 kernels use Native POSIX Threading Library (NPTL)
- NPTL uses one-to-one mapping of userspace threads to kernel threads
- Our experiments use kernel 2.6.20

# Outline

- What is a Context Switch?
  - Overheads
  - Context switching in Linux
- Interrupt Handling Overheads
- ARM Experimentation Platform
- Context Switching
  - Experiment Setup
  - Results
- Interrupt Handling
  - Experiment Setup
  - Results

# Interrupt Handling

- Interruption of normal program flow
- Virtual memory maps not switched
- Also causes overheads
  - Save and restore of processor state
  - Perturbation of processor caches

# Outline

- What is a Context Switch?
  - Overheads
  - Context switching in Linux
- Interrupt Handling Overheads
- ARM Experimentation Platform
- Context Switching
  - Experiment Setup
  - Results
- Interrupt Handling
  - Experiment Setup
  - Results

# ARM Experimentation Platform

- Processor Core: ARM9 @ 120 MHz
- SoC: Texas Instruments OMAP1610
- Split (Harvard) Cache
  - Instruction: 16 KB, 4-Way
  - Data: 8KB, 4-Way
- Virtually Tagged Caches
- Address Translation Cache (TLB)
  - Instruction: 64 entries
  - Data: 64 entries
- Measurement clock resolution: 0.16 microsecond

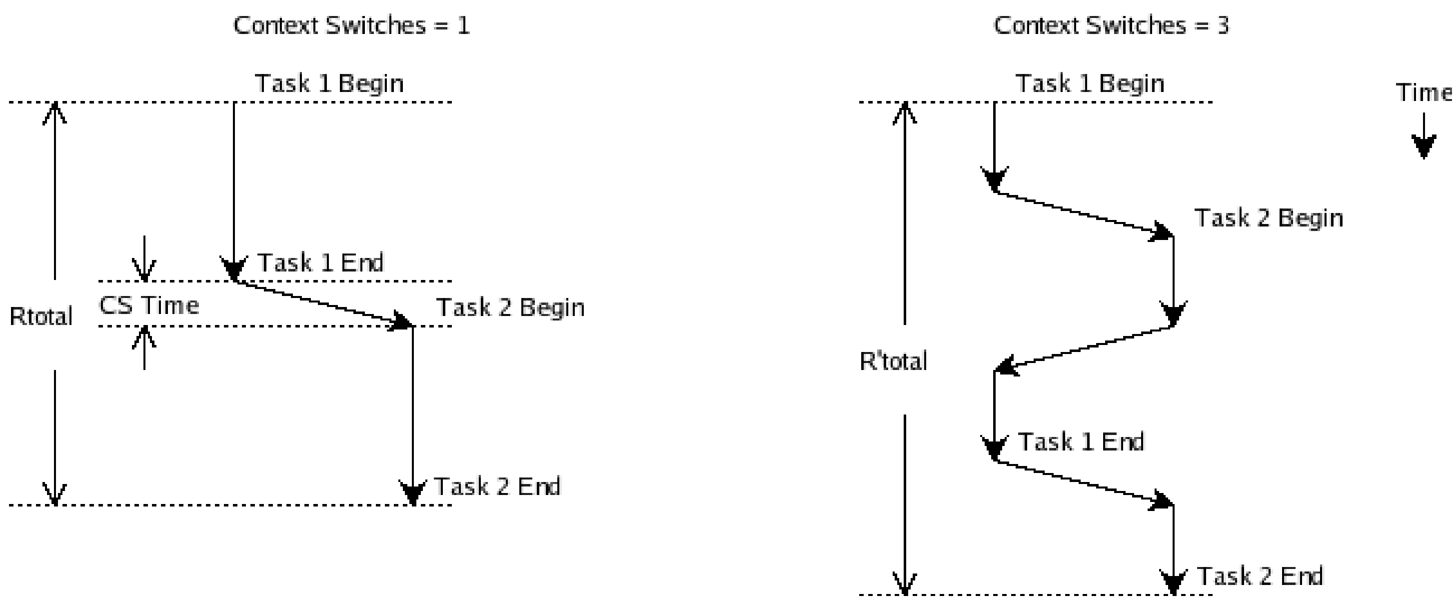
# Outline

- What is a Context Switch?
  - Overheads
  - Context switching in Linux
- Interrupt Handling Overheads
- ARM Experimentation Platform
- Context Switching
  - Experiment Setup
  - Results
- Interrupt Handling
  - Experiment Setup
  - Results

# Context Switching Measurement

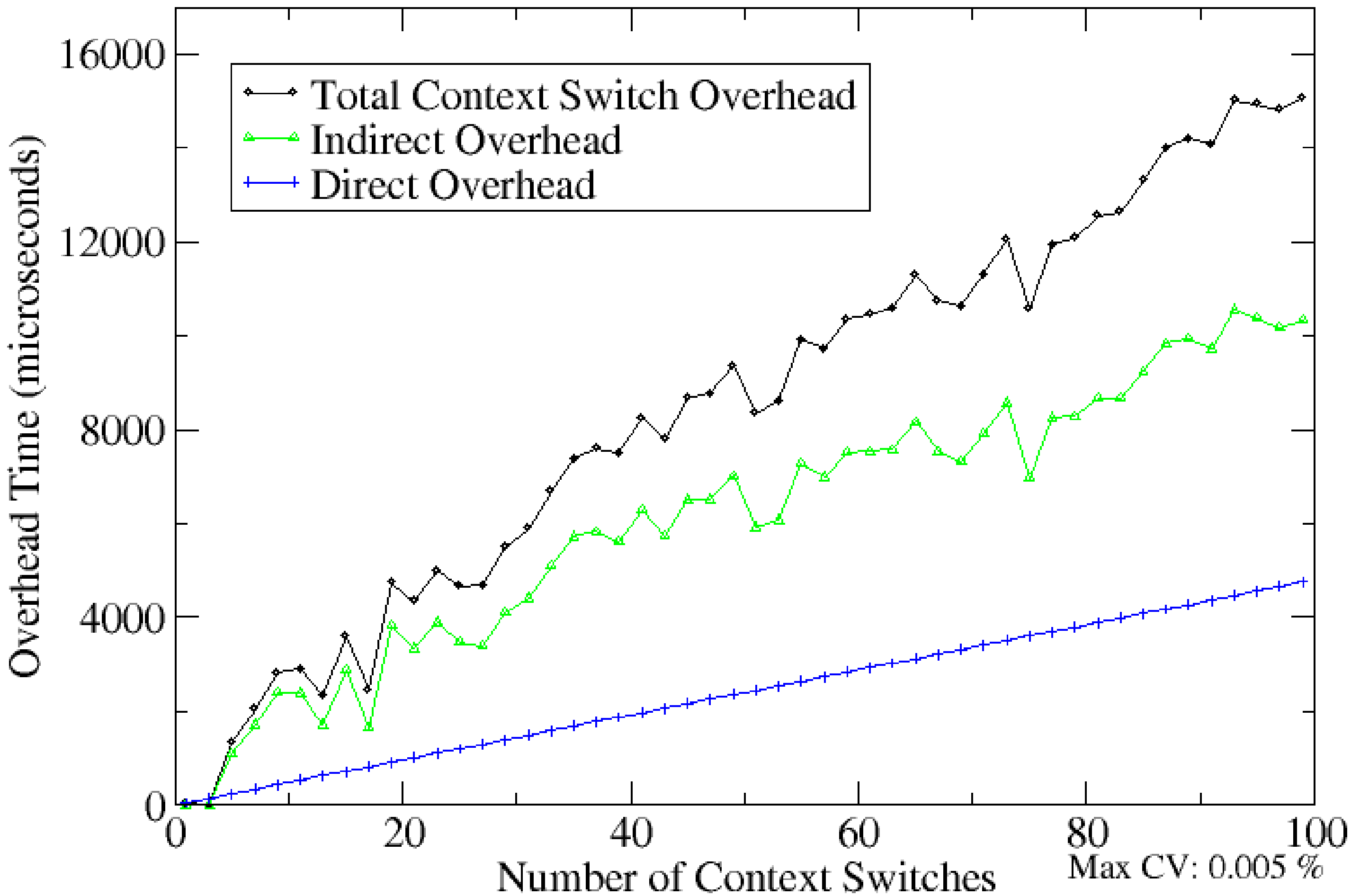
- Modified Kernel
  - Controlled environment
  - No interrupts
  - No system processes
- Pair of tasks using cooperative multitasking
  - Bubble sort (Duration: 3.6 seconds)
  - Deflate compression (Duration: 3.5 seconds)
  - AES encryption (Duration: 3.3 seconds)
  - CRC computation (Duration: 3.3 seconds)
- MMU switched and caches flushed

# Context Switching Measurement

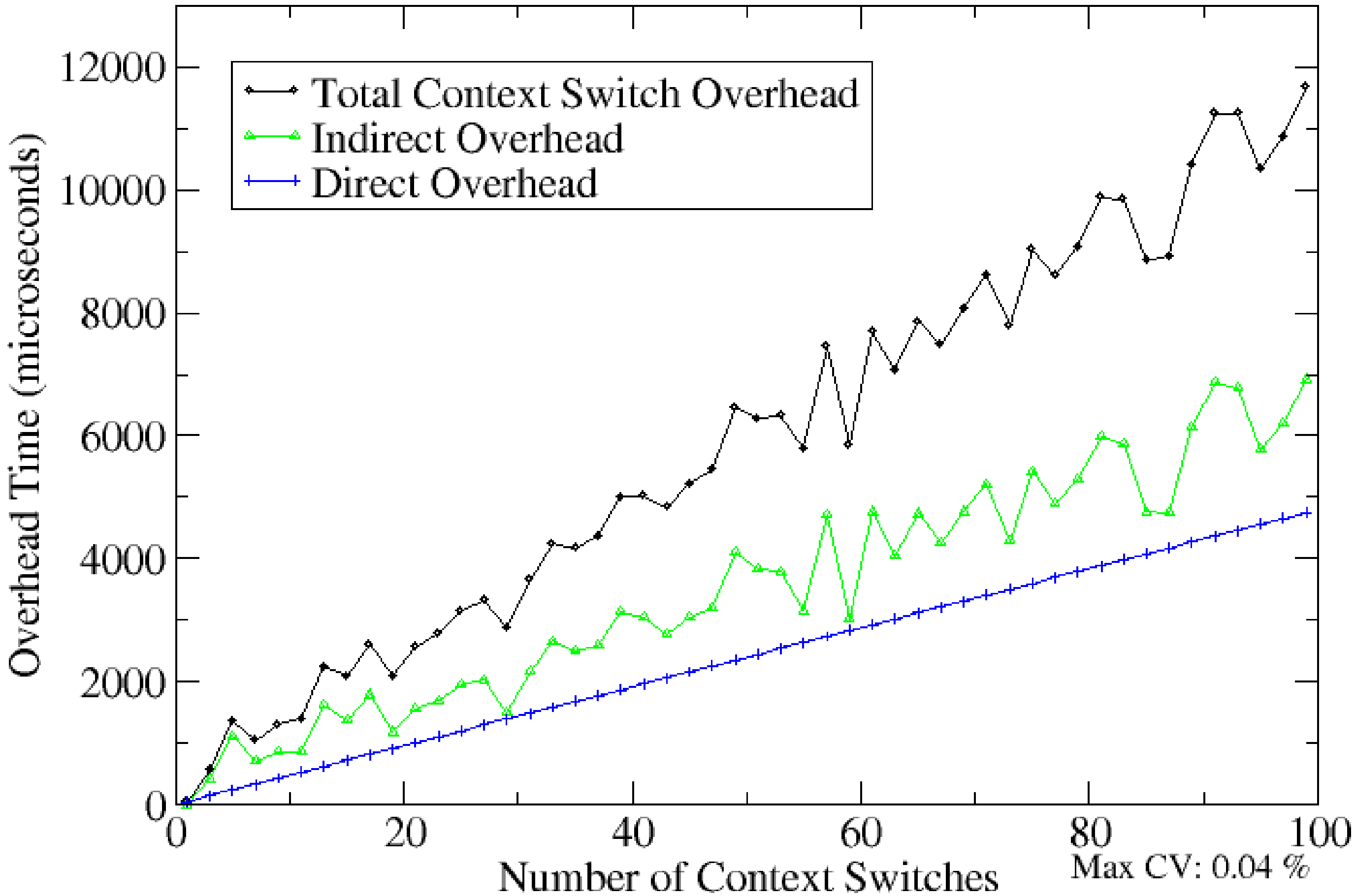



For *n* context switches with direct overhead *C*, the total indirect overhead $I = R'_{total} - (R_{total} - C) - n \times C$

# Sort and Deflate Compression



# AES and CRC

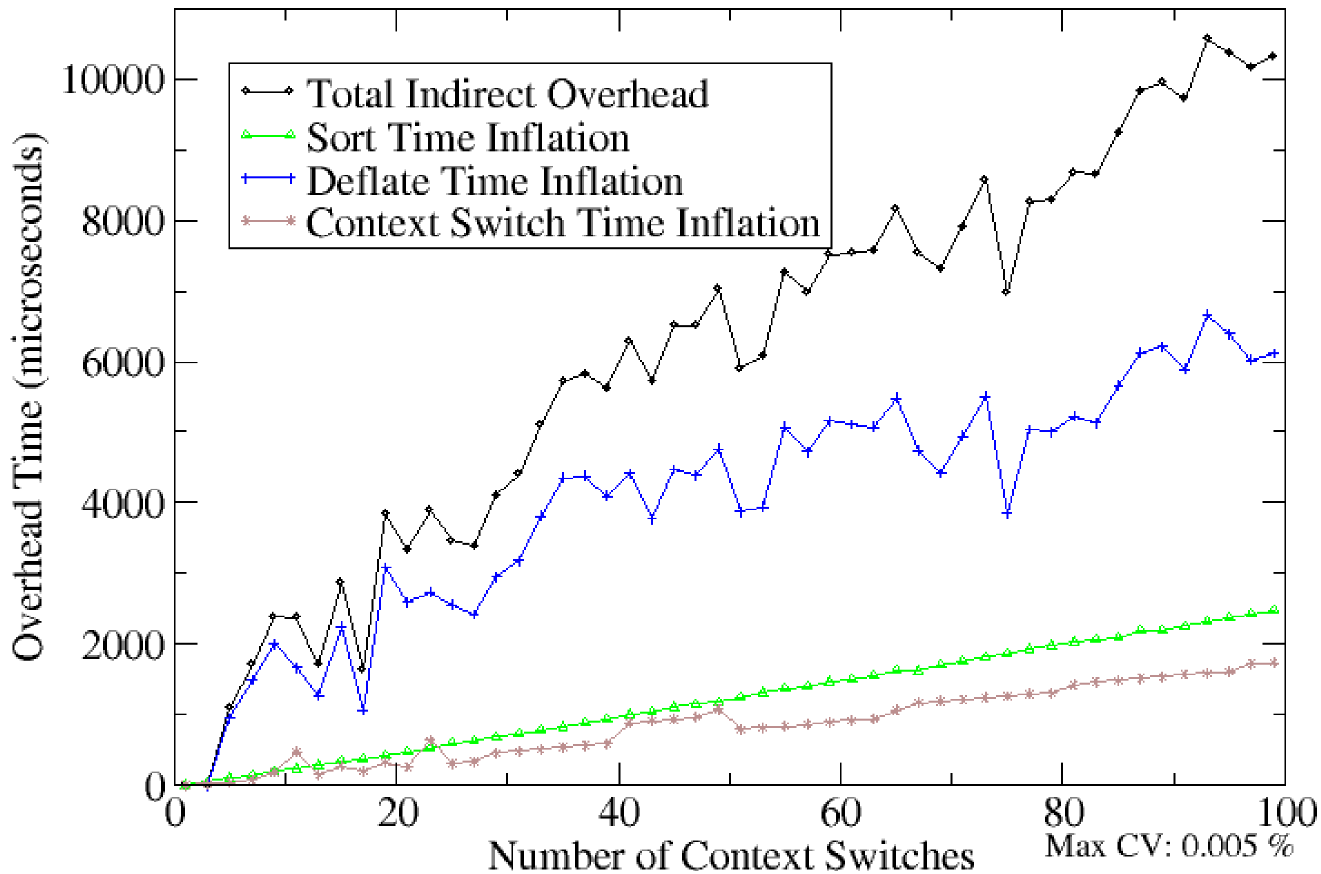


Total Context Switch Overhead
Indirect Overhead
Direct Overhead
Overhead Time (microseconds)
0
2000
4000
6000
8000
10000
12000
0
20
40
60
80
100
Number of Context Switches
Max CV: 0.04 %

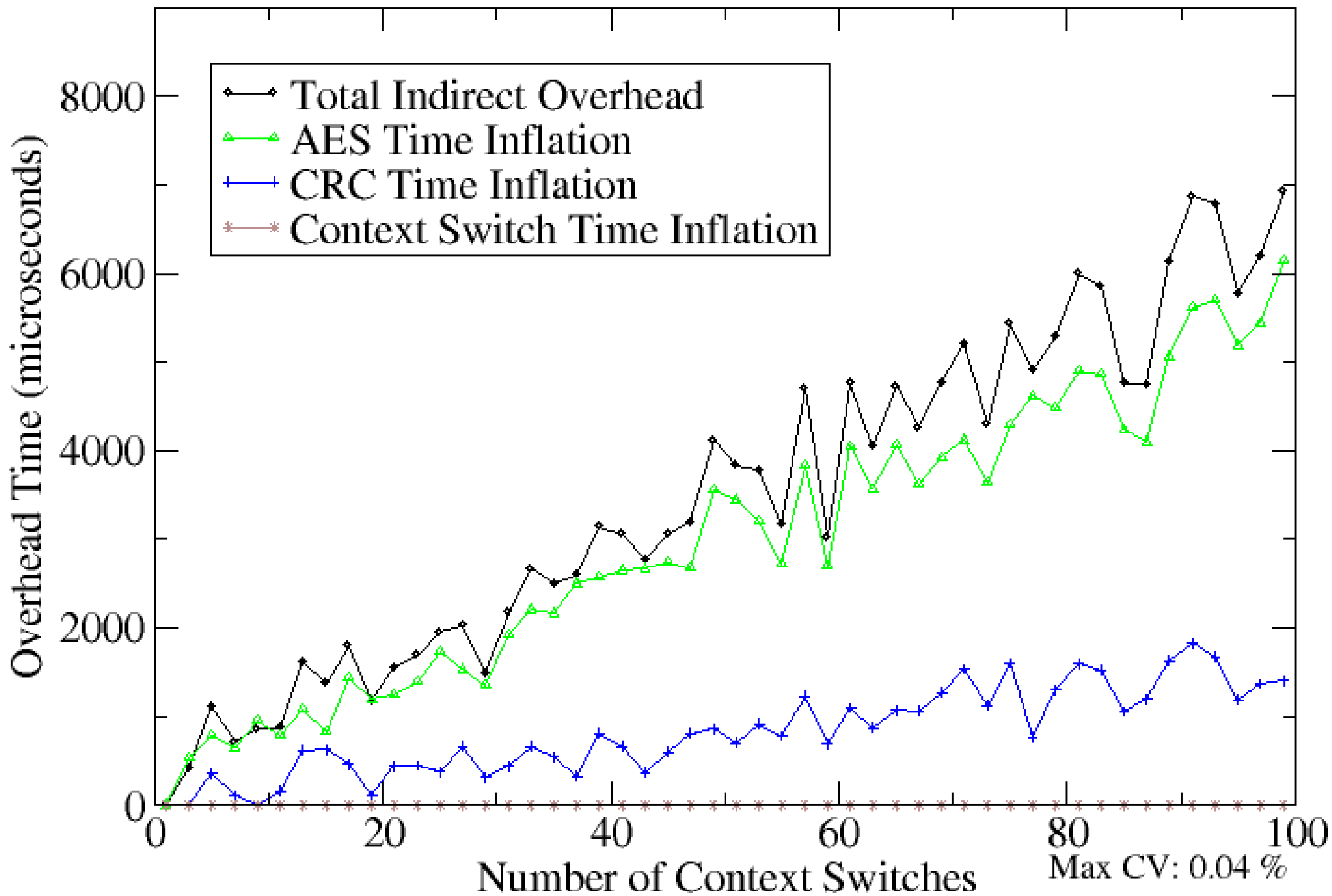
# Analysis

- Direct Overhead: 48 microseconds
- For 99 context switches
  - Max observed total overhead = 0.25%
  - Max observed indirect overhead = 0.18%
  - Indirect overhead > direct overhead

# Indirect Overhead Breakdown Sort-Deflate



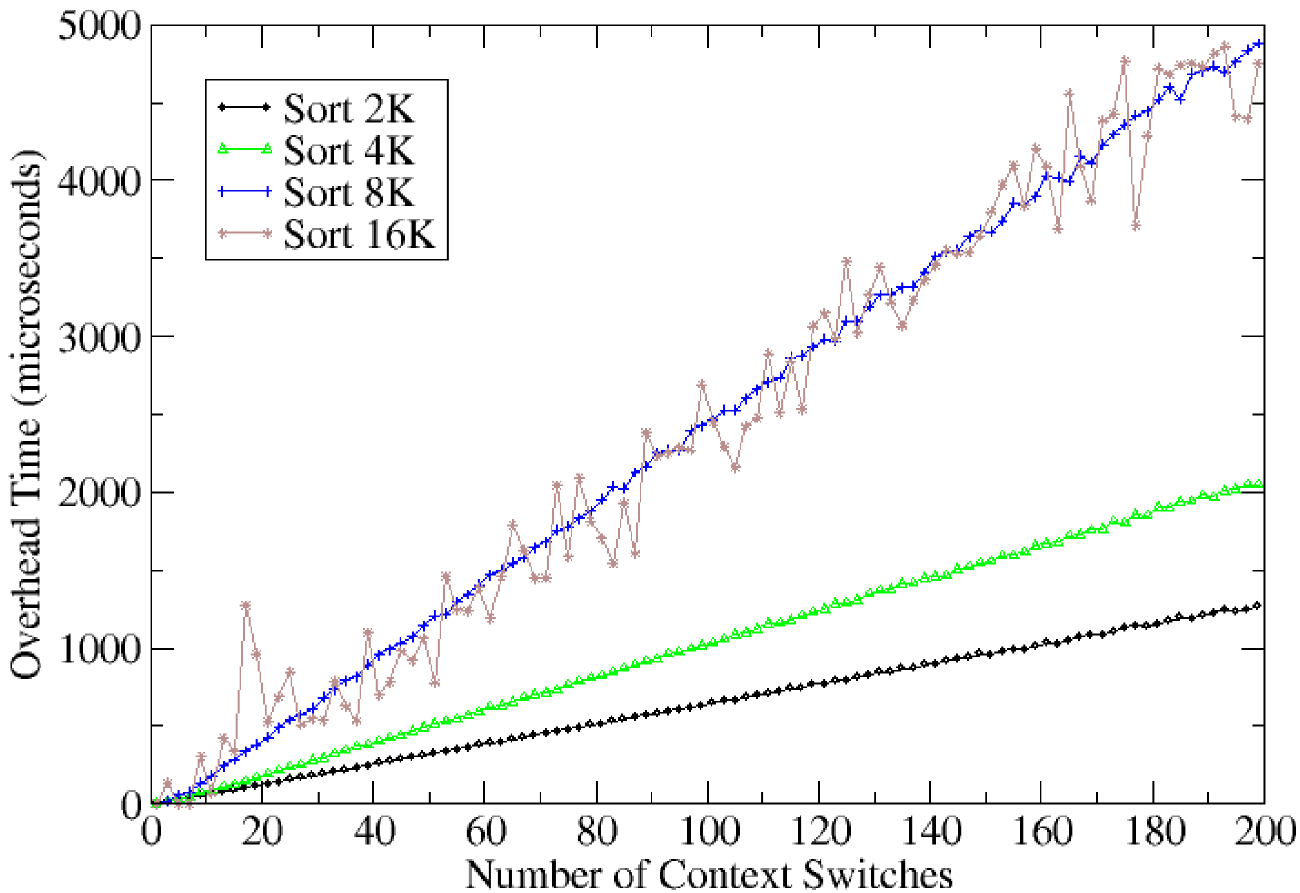
# Indirect Overhead Breakdown AES-CRC



# Analysis

- Execution Time Inflation due to Context Switching
  - AES & Deflate: Around 0.1%
  - Sort: Around 0.035%
  - CRC: Around 0.028%

# Varying Dataset Size with Sort



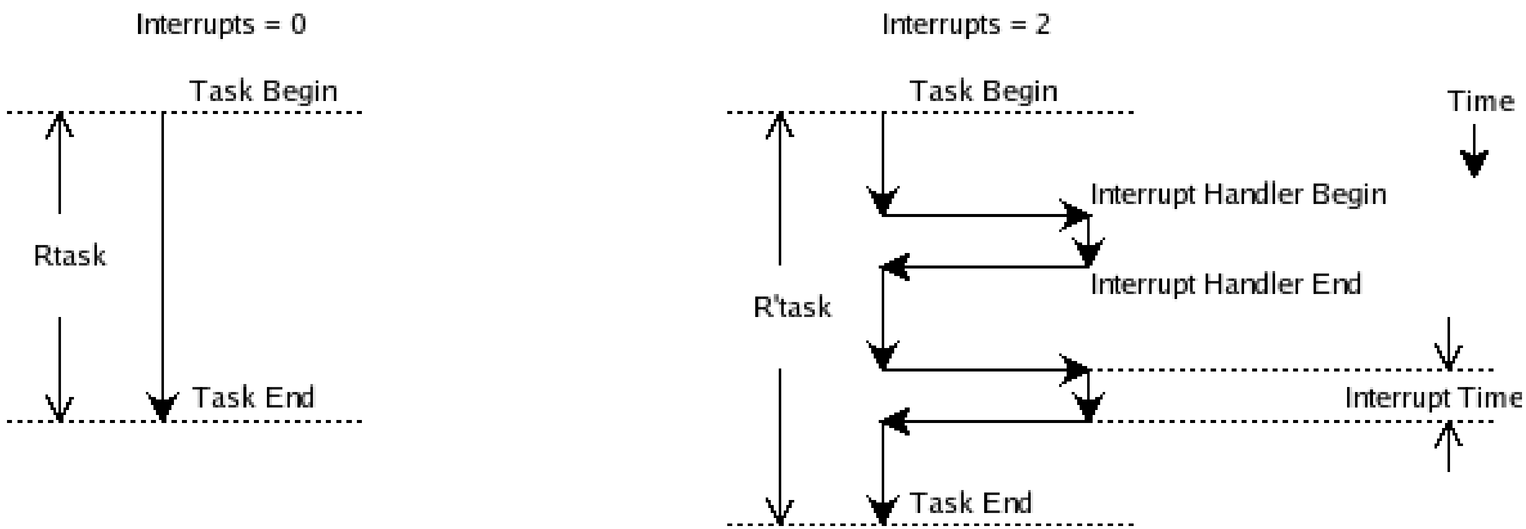
# Outline

- What is a Context Switch?
  - Overheads
  - Context switching in Linux
- Interrupt Handling Overheads
- ARM Experimentation Platform
- Context Switching
  - Experiment Setup
  - Results
- Interrupt Handling
  - Experiment Setup
  - Results

# Interrupt Handling Measurement

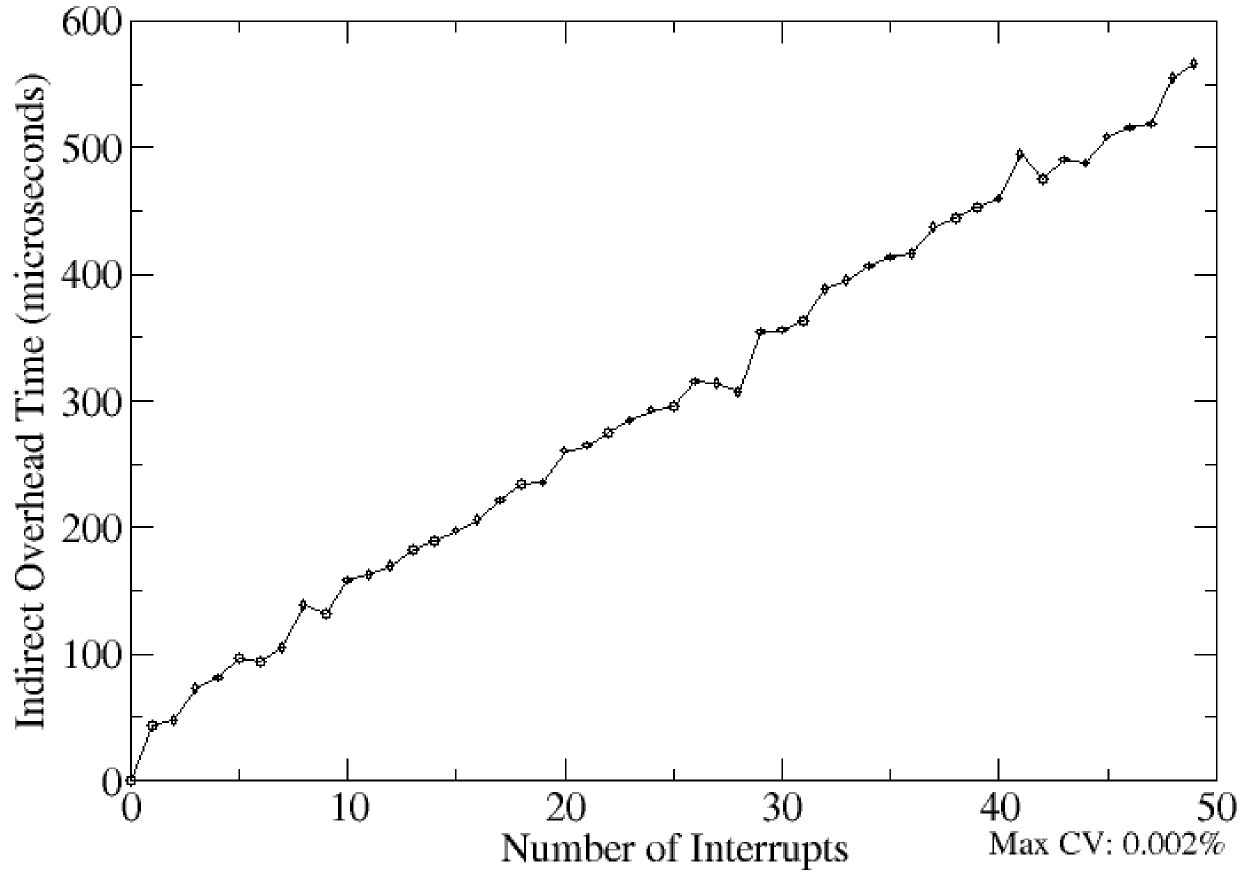
- Modified kernel for controlled environment
- Only one task – no context switching
- Only one interrupt enabled - timer
- Software triggered interrupt
- Study inflation in execution time of task with increasing number of interrupts

# Interrupt Handling Measurement




If the direct overhead in servicing the timer interrupt is *D*, the total indirect overhead is $I = R'_{task} - R_{task} - n \times D$

# Sort Indirect Overhead



# Analysis

- Indirect overhead for 49 interrupts
  - Sort – 0.01%
  - Deflate – 0.02%
  - AES – 0.09%
  - CRC – 0.05%

# Compare to Context Switching

Indirect Overheads for 49 interruptions

| | Context Switching* | Timer Interrupt* |
|---|---|---|
| Sort | 2500 | 600 |
| Deflate Compression | 7000 | 1300 |
| AES | 6000 | 3000 |
| CRC | 1500 | 1500 |

*Approximate, in microseconds

# Related Work

- Ousterhout: Measured round trip token passing time through pipe
- lmbench: Eliminated syscall overhead
- HP Labs: Relationship between caches and context switching
- University of Virginia: Effect on branch predictors is minimal
- Faster Context Switching
  - Processor Feature – Fast Address Space Switching
  - Physically tagged caches – ARM11

# Concluding Remarks

- Our context switch and interrupt measurements are performed at kernel level
  - Context Switch trend should reflect userspace switching closely
  - Interrupt handling effect should be identical for userspace tasks

- Code available on website
  http://choices.cs.uiuc.edu

# Context Switch Overheads for Linux on ARM Platforms

Francis David fdavid@uiuc.edu
Jeffrey Carlyle jcarlyle@uiuc.edu
Roy Campbell rhc@uiuc.edu

http://choices.cs.uiuc.edu

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

# Measurement Bug