X86 vs. ARM64:
An Investigation of Factors Influencing Serverless Performance

Xinghan Chen, Ling-Hong Hung, Robert Cordingly, Wes Lloyd
kirito20@uw.edu

School of Engineering and Technology
University of Washington Tacoma

December 11, 2023

24th ACM/IFIP International Middleware Conference
MIDDLEWARE 2023

Outline

Background and Motivation

- Research Goals
- Methodology
- Results
- Conclusions

Research Goals

- **In this project, for executing serverless FaaS functions on x86 and ARM64 processors, we investigate differences in:**
    - **(RQ-1):** CPU utilization metrics
    - **(RQ-2):** Performance
    - **(RQ_3):** Performance variance
    - **(RQ-4):** Cost

Outline

- Background and Motivation
- Research Goals
- Methodology
- Results
- Conclusions

X86 vs. ARM64

Computing architecture

Switch to ARM64:

- Power efficiency
- Low cost

Stay on X86:

- No migration cost
- Widely supported
- Performance optimization
- Rely on platform specific abilities

?

Outline

- Background and Motivation
- Research Goals
- ➤ Methodology
- Results
- Conclusions

Workloads

AWS Lambda us-west-2 (Oregon),
memory size: 3008MB(3GB) with 2 vCPU cores,
5GB ephemeral disk for I/O related tests

	Short Name	Function Name	Description
cpuUser	linpack	python_linpack	Solve linear equations: Ax = b
	chacha20	openssl_encrypt_chacha20	Repeatedly perform openssl encryption of 8MB file n times
	sqlite	python_sqlite_dump	Execute n random SELECT queries on a 10*1000 SQLite database
	video-processing	ffmpeg_sebs_220_gif	Convert PNG to GIF n times
	json_dumps	python_json_dumps	JSON deserialization using a downloaded JSON-encoded string dataset
	graph-pagerank	python_sebs_501_pagerank	PageRank implementation with igraph.
	graph-mst	python_sebs_502_mst	Minimum spanning tree (MST) implementation with igraph.
	float	python_float_operation	Perform sin, cos, sqrt ops
	chameleon	python_chameleon	Create HTML table of n rows and M columns
	graph-bfs	python_sebs_503_bfs	Breadth-first search (BFS) implementation with igraph.
F	primenumber	sysbench_cpu_prime	Prime number generator
cpuKernel	thread	sysbench_thread	Create thread, put locks and release thread
	filehandle	python_fopen	Open and close file handles
	socket	python_socket	Open and close socket n times
Memory	readmemory	sysbench_memory	N sequential reads of 1GB memory block
	readwritememory	python_malloc_write	Allowcate 1MByte of memory, write 0x42 into it and release
I/O	readdisk	fio_disk_io_random_read	Test random read speed on a 1GB block
	compression	python_sebs_311_compression	Create a .gz file for a file

Supporting Tools - SAAF

We utilize the Serverless Application Analytics Framework to collect metrics from serverless functions.

Metrics such as CPU timing accounting, runtime, latency, and more can collected by the Analyzer function and used to make routing decisions by the Proxies.

SAAF and our other tools are is available here:
https://github.com/wlloyduw/SAAF

Outline

- Background and Motivation
- Research Goals
- Methodology
- Results
- Conclusions

Research Question 1

How do Linux CPU utilization measurements compare for serverless functions run on x86 (Intel) vs. ARM64 (Graviton2) processors?

We investigate changes in CPU user mode time, CPU kernel mode time, and CPU idle time.

# ARM64 vs. x86 CPU Utilization Comparison
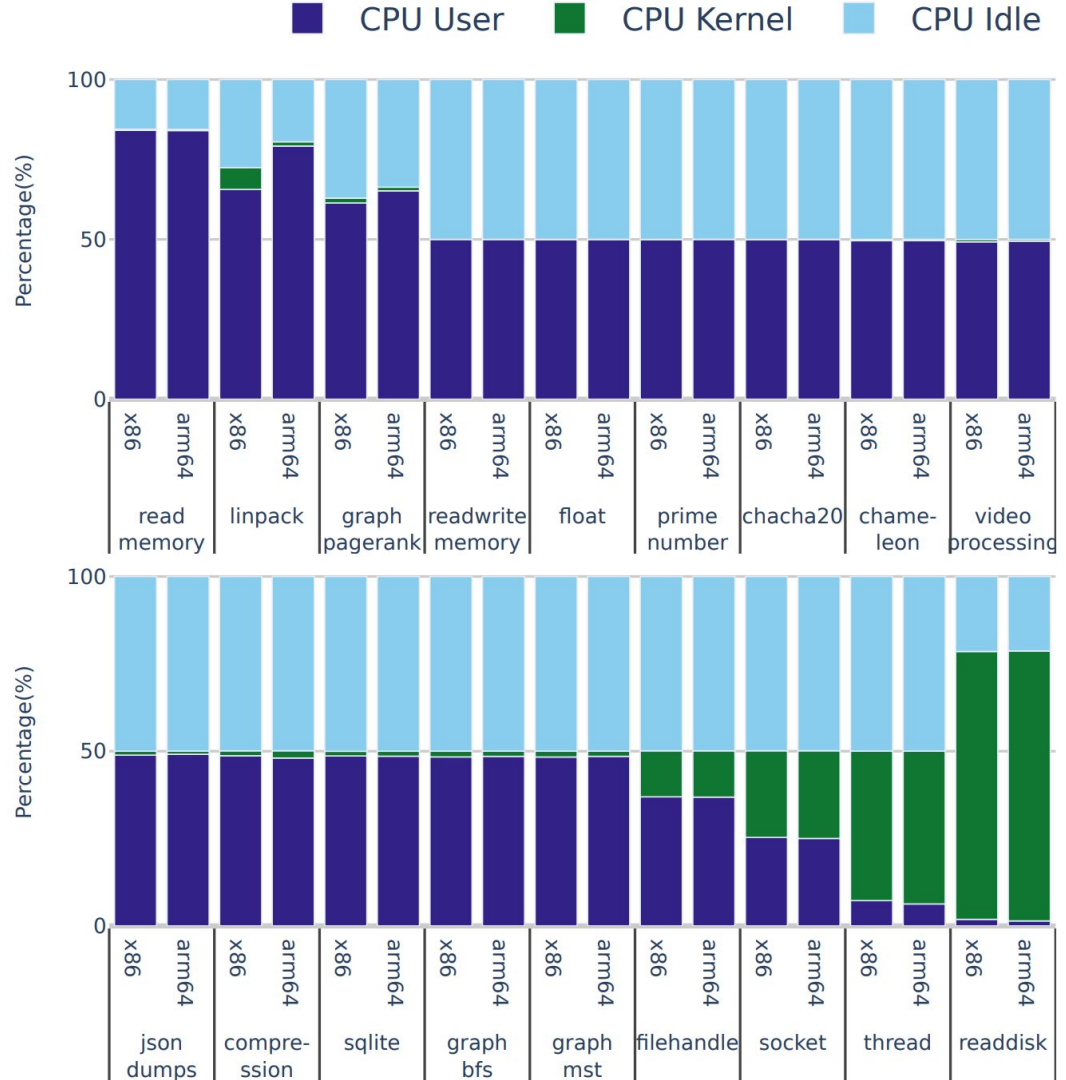
Research Question 2

How does serverless function runtime compare on x86 (Intel) vs. ARM64 (Graviton2) processors?

Using runtime on x86 processors as a baseline, we identify functions with faster runtime on ARM, similar runtime on ARM, and slower runtime on ARM. In addition, we investigate x86 vs. ARM64 runtime implications when scaling up the work performed by function instances

ARM64 Function Performance Difference vs. x86
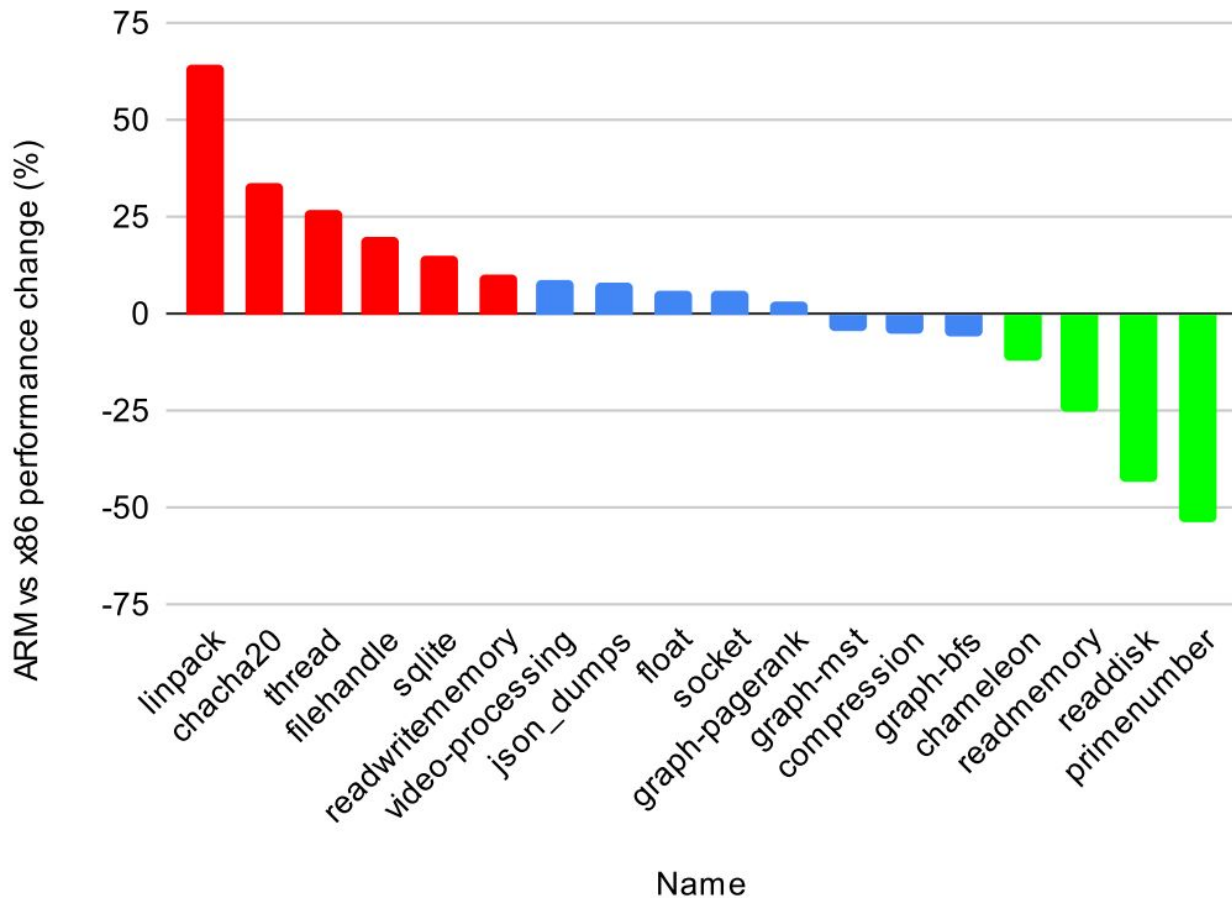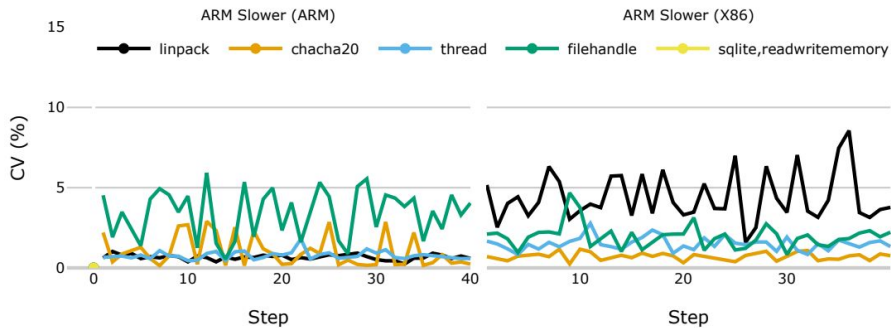
Research Question 3

What is the difference in performance variance of serverless functions executed on x86 (Intel) vs. ARM64 (Graviton2) processors?

We calculate and analyze the coefficient of variation of function runtime while scaling the work of function instances using forty distinct steps to increase runtime.

Average CV (%) of function runtime
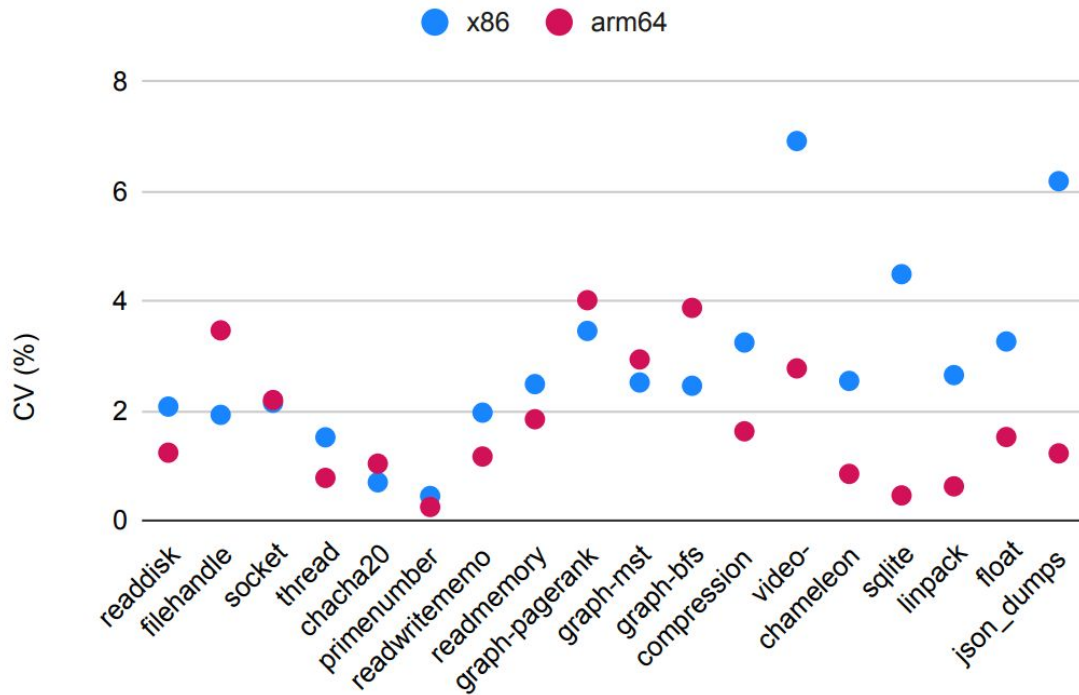
Function runtime: change in CV(%) over 40 steps
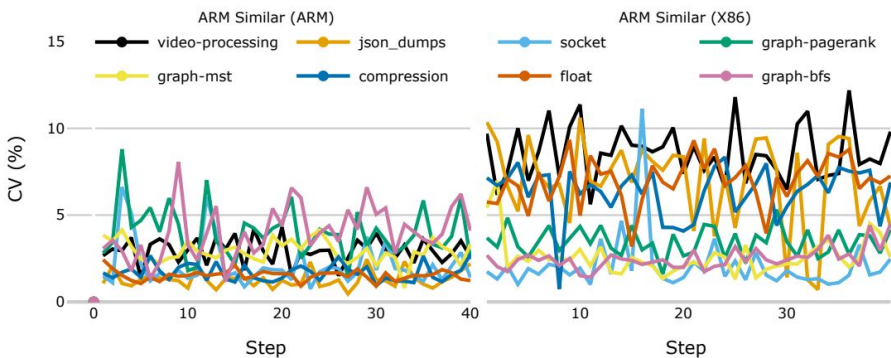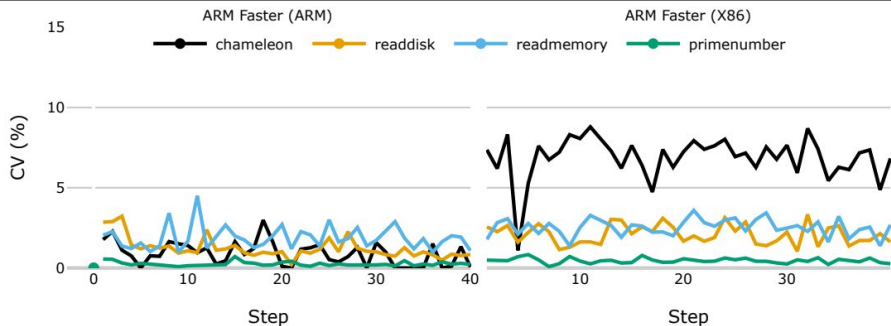
Research Question 4

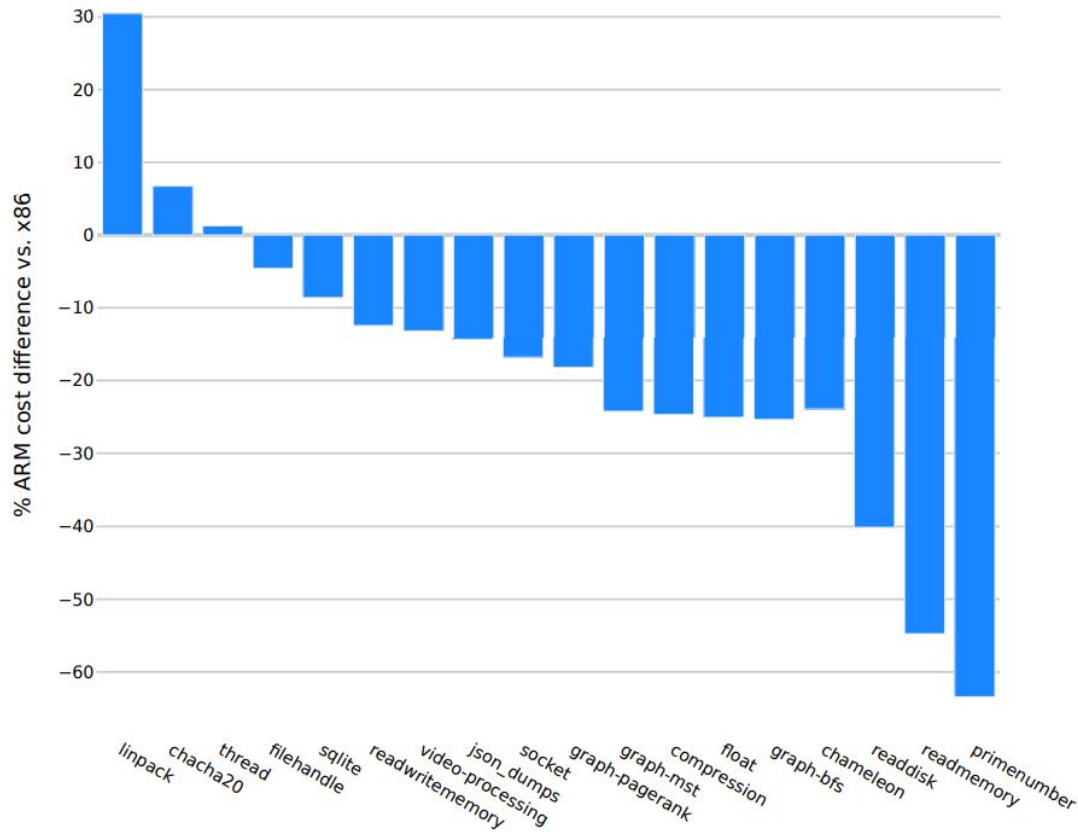What is the cost difference in hosting serverless functions on x86 (Intel) vs. ARM64 (Graviton2) processors?

We compare the overall hosting costs of 18 distinct functions while scaling function runtime across forty steps.

Estimated cost of 400k function calls: x86 vs. ARM
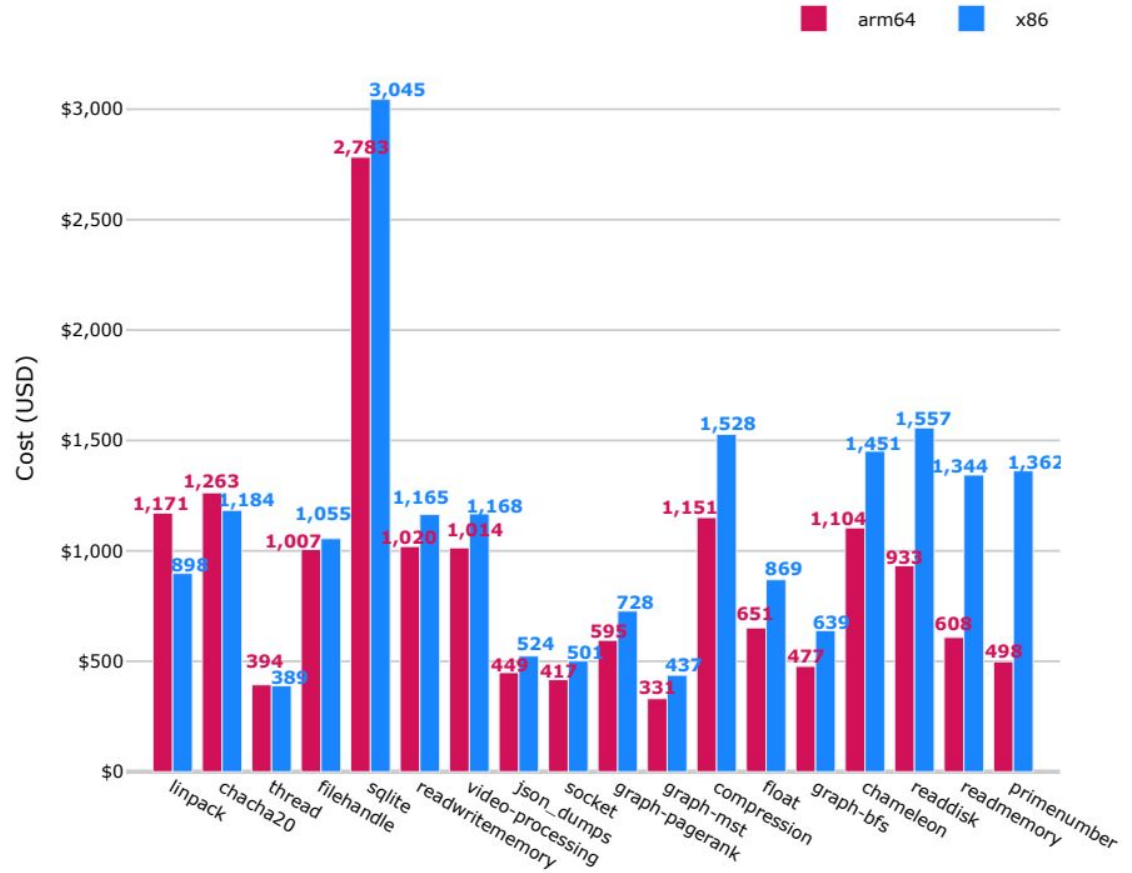
ARM cost difference (+/-%) relative to x86

Outline

- Background and Motivation
- Research Goals
- Methodology
- Results

Conclusions

Conclusion Summary

- We executed experiments using 18 functions on AWS to compare X86 vs. ARM64 FaaS

- **(RQ-1 - CPU Utilization):** While most functions had similar CPU utilization profiles across both architectures, some functions on ARM64 had higher CPU kernel mode utilization. These differences may help detect where x86 vs. ARM64 performance differences are likely occur.

- **(RQ-2 - Performance):** ARM64 can provide performance advantages for serverless workloads. ARM64 provided faster runtime than x86 for 7 of 18 functions. Four functions were more than 10% faster. Runtime improvements appeared highly dependent on the nature of the workload.
  →**Average function runtime increased by 2.86% (18 functions x 40 timesteps).**

Conclusion Summary - 2

- **(RQ-3 - Performance Variance):** Functions run on x86 on AWS Lambda, exhibit more than twice the runtime variance vs. ARM64 making x86 less reliable for consistent performance.

- **(RQ-4 - Cost):** ARM64 offers cost savings on AWS Lambda (15 of 18 tested serverless functions). Some of the cost savings are attributed to the 20% cost discount offered by the cloud provider for ARM64 processors.
  → **Average execution costs decreased by 18.4% (18 functions x 40 timesteps)**

Thank You!