Intel® AVX-512 Accelerates PostgreSQL 18 CRC32C Checksums

Real-World Performance Gains and Cost Savings from Hardware-Accelerated CRC32C Calculations

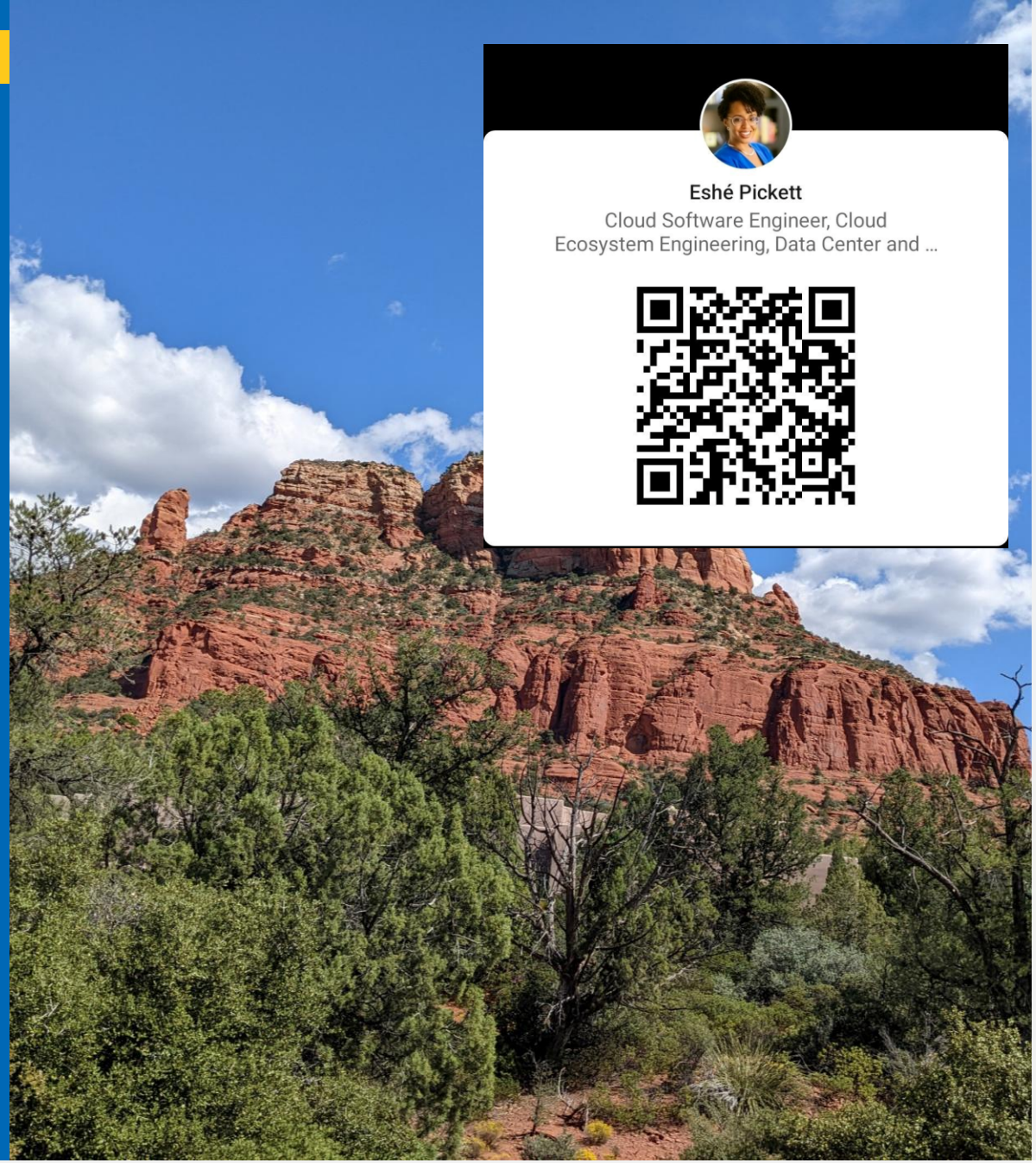
Eshé N Pickett
eshe.pickett@intel.com

intel

- Undergrad/Grad – Computer Science
- System Administration
- (Backend Silicon) Design Automation
- Cloud Infrastructure
- Software Architecture
- Cloud Software Development

LinkedIn: @eshepickett

About Me



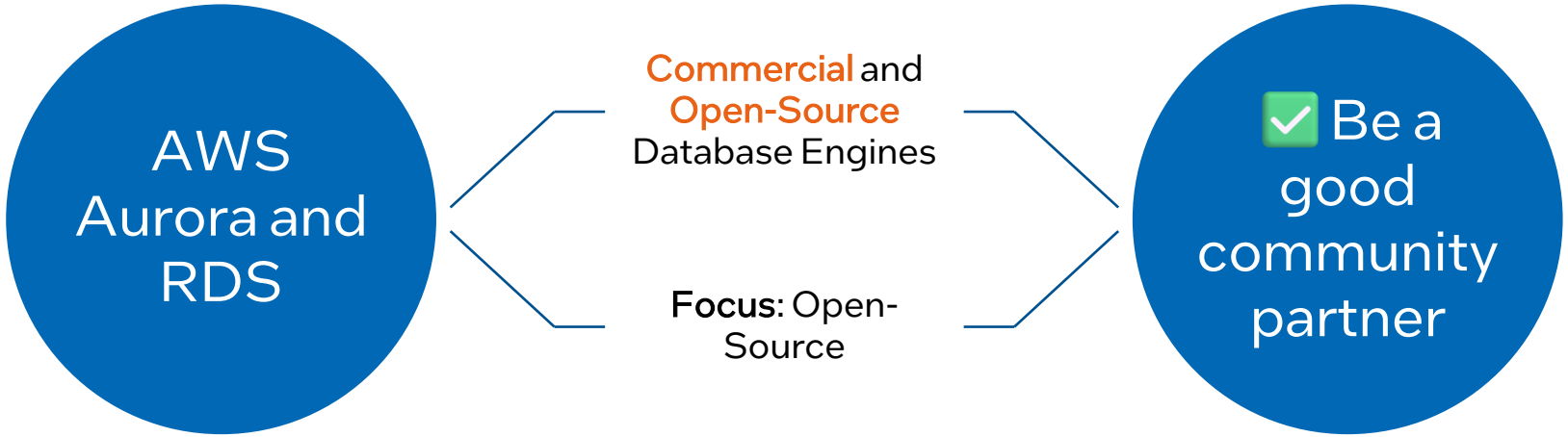
Session Agenda

- Problem Context
- Intel SIMD Architecture & AVX-512
- PostgreSQL CRC32C Implementation
- Benchmark Methodology & Configuration
- Real-world Results & Cost Analysis
- Community Impact & Future Directions

Problem Context

We’ll Spend 5 Minutes Here

Problem Context



Intel SIMD Architecture & AVX-512 Capabilities

We'll Spend 10 Minutes Here

Single Instruction, Multiple Data (SIMD)

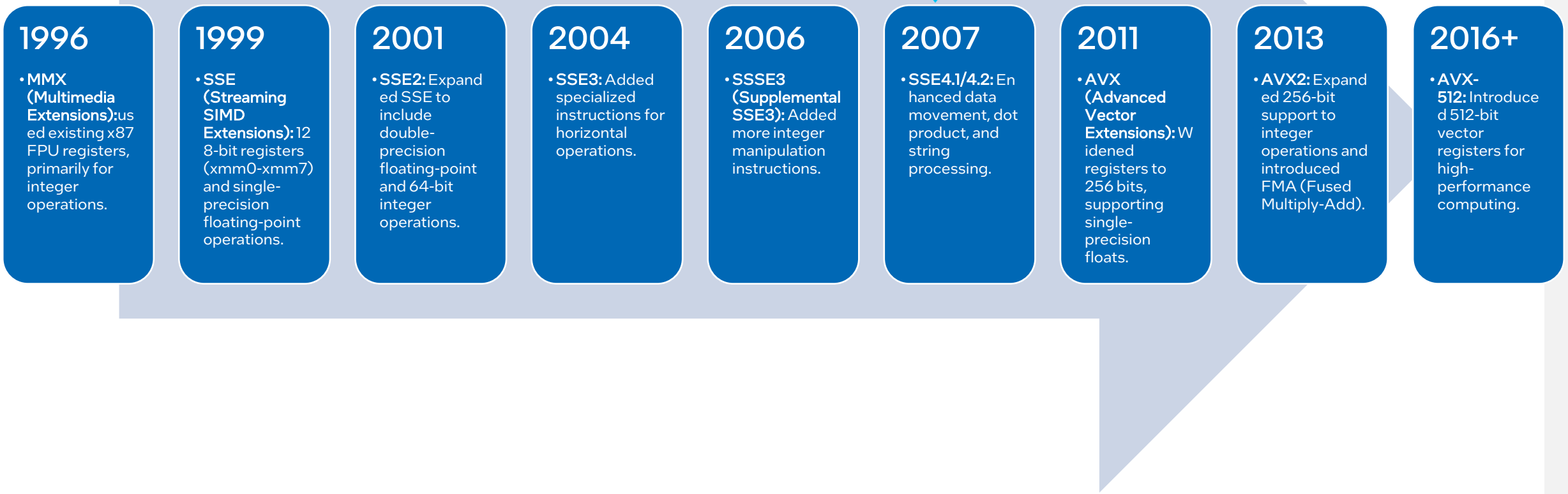
- SIMD **boosts CPU performance** by applying the same operations **across multiple data lanes.**
- More lanes ***usually*** mean better performance
  - As long as the **code aligns with the processor's instruction set**.

Single Instruction Multiple Data Made Easy with Intel® Implicit SPMD Program Compiler

https://www.intel.com/content/www/us/en/developer/articles/technical/simd-made-easy-with-intel-ispc.html

Intel SIMD Intrinsics Timeline

Introduction of **CRC32C** and **string comparison** instructions!



Intel® Processor Architecture SIMD Instructions (Historical)

https://www.intel.sg/content/dam/www/public/apac/xa/en/pdfs/ssg/Intel_Processor_Architecture_SIMD_Instructions.pdf

Recent Extensions

Advanced Matrix Extensions (AMX) 2021

Introduces tile registers for matrix operations (up to 8KB per tile)

Optimized for AI/ML workloads with INT8 and BF16 operations

Provides acceleration for matrix multiplication operations

Recent Extensions

Advanced Performance Extensions (APX) 2023/2024

Doubles the number of general-purpose registers from **16** to **32**

Adds new conditional forms of existing instructions

Enhances register utilization and reduces register pressure

Note: Not primarily a SIMD extension, but it improves overall performance

Intel® Intrinsics Guide

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

Recent Extensions

Intel Advanced Vector Extension 10.2 (AVX10.2) 2024/2025

Unifies AVX-512 capabilities across Intel product lines

Provides consistent 256-bit and 512-bit vector support

Includes new media and AI acceleration instructions

Key Take away: Simplifies software development with unified instruction set

Intel® Advanced Vector Extensions 10.2 (Intel® AVX10.2) Architecture Specification

https://www.intel.com/content/www/us/en/content-details/828965/intel-advanced-vector-extensions-10-2-intel-avx10-2-architecture-specification.html

AVX Evolution Recap

Intel® AVX	Intel® AVX2
128/256-bit FP 16 registers NDS (and AVX128) Improved blend MASKMOV Implicit unaligned	Float16 128/256-bit FP FMA 256-bit int PERMD Gather

Intel® AVX-512
128/256/512-bit FP/Int 32 vector registers 8 mask registers 512-bit embedded rounding Embedded broadcast Scalar/SSE/AVX "promotions" Native media additions HPC additions Transcendental support Gather/Scatter Flag-based enumeration Intel® Xeon P-core only

Intel® AVX10.1 (pre-enabling)
128/256/512-bit FP/Int 32 vector registers 8 mask registers 512-bit embedded rounding Embedded broadcast Scalar/SSE/AVX "promotions" Native media additions HPC additions Transcendental support Gather/Scatter **Version-based** enumeration Intel® Xeon P-core only

Intel® AVX10.2
New data movement, transforms and type instructions 128/256/512-bit FP/Int 32 vector registers 8 mask registers 512-bit embedded rounding Embedded broadcast Scalar/SSE/AVX "promotions" Native media additions HPC additions Transcendental support Gather/Scatter **Version-based** enumeration **Supported on P-cores, E-cores**

Bringing the Next Generation of Intel® CPU Performance Optimizations to GCC* 15

https://www.intel.com/content/www/us/en/developer/articles/technical/next-gen-performance-gcc-15.html

Options for Vectorization

Option	Description	Pro	Con
Compiler Switch	Enable auto-vectorization via the compiler	Easy	Unreliable
C99 'restrict' keyword	Do not allow aliased parameters	Developer Controlled	Requires Subject Matter Expertise
Use The IVDEP Pragma	Ignore assumed vector dependencies	Developer Controlled	Intel Specific - Not supported by all compilers
OpenMP Pragma (opm_simd)	Mandatory vectorization	Developer Controlled	Requires Subject Matter Expertise
Use Intrinsics	Use specific SIMD instructions	Developer Controlled	Not Portable
Code Explicitly	Manually vectorize key sections	Developer Controlled	Requires Subject Matter Expertise

[Performance: SIMD, Vectorization and Performance Tuning | James Reinders, former Intel Director]

[S2024 #06 - Vectorized Query Execution Using SIMD (CMU Advanced Database Systems)]

Simple Array Summation Example: Scalar (Non-Vectorized)

```c
#include <immintrin.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

// Scalar version - processes one element at a time
float sum_scalar(float* array, int size) {
    float sum = 0.0f;
    printf("Scalar processing: 1 element per cycle\n");

    for (int i = 0; i < size; i++) {
        sum += array[i];  // Single addition per cycle
    }

    return sum;
}
```

Simple Array Summation Example: AVX-512 Vectorized

```
19 // AVX-512 version - processes 16 elements simultaneously
20 float sum_avx512(float* array, int size) {
21     __m512 vec_sum = _mm512_setzero_ps();  // Initialize 16 zeros
22     float final_sum = 0.0f;
23
24     printf("AVX-512 processing: 16 elements per cycle\n");
25     printf("Using 512-bit registers (16 x 32-bit floats)\n");
26
27     // Process 16 elements simultaneously
28     int vectorized_size = (size / 16) * 16;
29     int cycles = vectorized_size / 16;
30
31     printf("Vectorized cycles needed: %d (vs %d scalar cycles)\n", cycles, size);
32
33     for (int i = 0; i < vectorized_size; i += 16) {
34         // Load 16 floats from memory into 512-bit register
35         __m512 vec_data = _mm512_loadu_ps(&array[i]);
36
37         // Add 16 pairs simultaneously in one instruction
38         vec_sum = _mm512_add_ps(vec_sum, vec_data);
39     }
40
41     // Horizontal sum: combine all 16 elements in vec_sum into single value
42     final_sum = _mm512_reduce_add_ps(vec_sum);
43
44     // Handle remaining elements (if size not divisible by 16)
45     for (int i = vectorized_size; i < size; i++) {
46         final_sum += array[i];
47     }
48
49     return final_sum;
50 }
```

Simple Array Summation: Key AVX-512 Intrinsics Used

```
52 // Demonstrate the key AVX-512 intrinsics used
53 void demonstrate_intrinsics() {
54     printf("\n=== AVX-512 Intrinsics Demonstration ===\n");
55
56     // Create sample data
57     float data[16] = {1.0f, 2.0f, 3.0f, 4.0f, 5.0f, 6.0f, 7.0f, 8.0f,
58                       9.0f, 10.0f, 11.0f, 12.0f, 13.0f, 14.0f, 15.0f, 16.0f};
59
60     printf("Input data: ");
61     for (int i = 0; i < 16; i++) {
62         printf("%.0f ", data[i]);
63     }
64     printf("\n");
65
66     // Key intrinsics used in the optimization:
67
68     // 1. _mm512_loadu_ps - Load 16 floats from unaligned memory
69     __m512 vec_data = _mm512_loadu_ps(data);
70     printf("1. _mm512_loadu_ps: Loaded 16 floats into 512-bit register\n");
71
72     // 2. _mm512_setzero_ps - Initialize register with zeros
73     __m512 vec_zero = _mm512_setzero_ps();
74     printf("2. _mm512_setzero_ps: Initialized register with 16 zeros\n");
75
76     // 3. _mm512_add_ps - Add 16 pairs of floats simultaneously
77     __m512 vec_result = _mm512_add_ps(vec_data, vec_data);  // Double each value
78     printf("3. _mm512_add_ps: Added 16 pairs simultaneously (doubled values)\n");
79
80     // 4. _mm512_reduce_add_ps - Horizontal sum of all 16 elements
```

Simple Array Summation: Test Output

```
=== AVX-512 Intrinsics Demonstration ===
Input data: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1. _mm512_loadu_ps: Loaded 16 floats into 512-bit register
2. _mm512_setzero_ps: Initialized register with 16 zeros
3. _mm512_add_ps: Added 16 pairs simultaneously (doubled values)
4. _mm512_reduce_add_ps: Horizontal sum = 136
Expected sum: 136 (1+2+...+16 = 136)
```

Simple Array Summation: Test Output

```
=== Performance Comparison ===
Initializing array with 1000000 elements...

--- Scalar Version ---
Scalar processing: 1 element per cycle

--- AVX-512 Version ---
AVX-512 processing: 16 elements per cycle
Using 512-bit registers (16 x 32-bit floats)
Vectorized cycles needed: 62500 (vs 1000000 scalar cycles)

=== Results ===
Scalar result:  4999413760000.00 (0.000843 seconds)
AVX-512 result: 4999998996480.00 (0.000349 seconds)
Results match:  NO
Speedup:        2.42x
Theoretical max speedup: 16x (16 elements per instruction)
```

PostgreSQL CRC32C Implementation Deep-dive

We'll spend 15 Minutes Here

Motivation

What tools in our “toolbox” do we have, and what paths can they benefit?

What instructions were currently in use? (SSE4.2)

Modernize existing SSE4.2 implementations using new Instruction set

Improve the performance of the CRC32C algorithm.

The SSE4.2 implementation of CRC32C processed **8 bytes** at a time in a loop	The AVX-512 implementation processes **64 bytes** at a time

CRC32C Performance pgsql-hackers Thread

https://www.postgresql.org/message-id/flat/PH8PR11MB82869FF741DFA4E9A029FF13FBF72@PH8PR11MB8286.namprd11.prod.outlook.com

What Benefits from CRC32C Improvement?

- Replication slot state includes a CRC32C over checksummed portions
- Backup
  - Backup/restore/related tooling paths that stream or materialize src/common/blkreftable.c structures can see CPU reductions.
- Two-phase commit (2PC) workloads (prepared transactions)
  - Can be affected, depending on how much CRC'd data is involved.
- Write-Ahead Logging (WAL)
  - Fairly large contiguous buffers (e.g., big FPIs, big records)
  - AVX-512 can reduce CPU time per record.

Intel® Architecture Instruction Set Extensions Programming Reference

https://www.intel.com/content/www/us/en/content-details/915637/intel-architecture-instruction-set-extensions-programming-reference.html

PostgreSQL 18 Key Changes

- New Intel® AVX-512 Path
- Smarter Runtime Checks
- Broader Hardware Support

Implementation Details

- Detection at Runtime and Build Time
- Function Dispatch
- Intel® AVX-512 Algorithm
- What's Involved?
  - CPUID
  - VPCLMULQDQ
  - ZMM registers
  - XGETBV

Intel® AVX-512 Algorithm

Derived from the MIT-licensed fast-crc32 project.

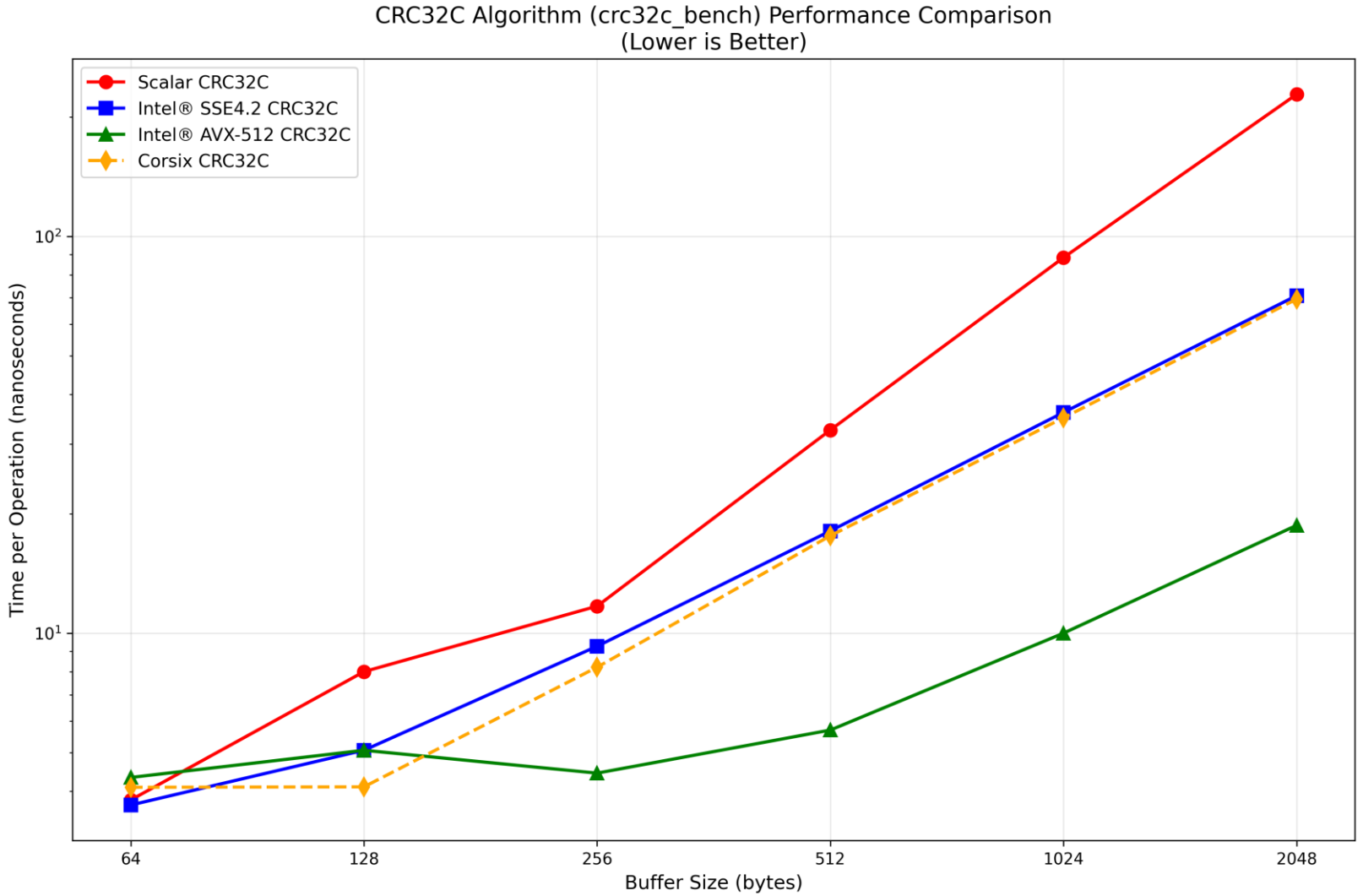
Processes 64 bytes at a time

load data:	Carry-less Multiplication:	Intrinsics for reductions:
_mm512_loadu_si512	_mm512_clmulepi64_epi128	_mm512_ternarylogic_epi64 ▪and others

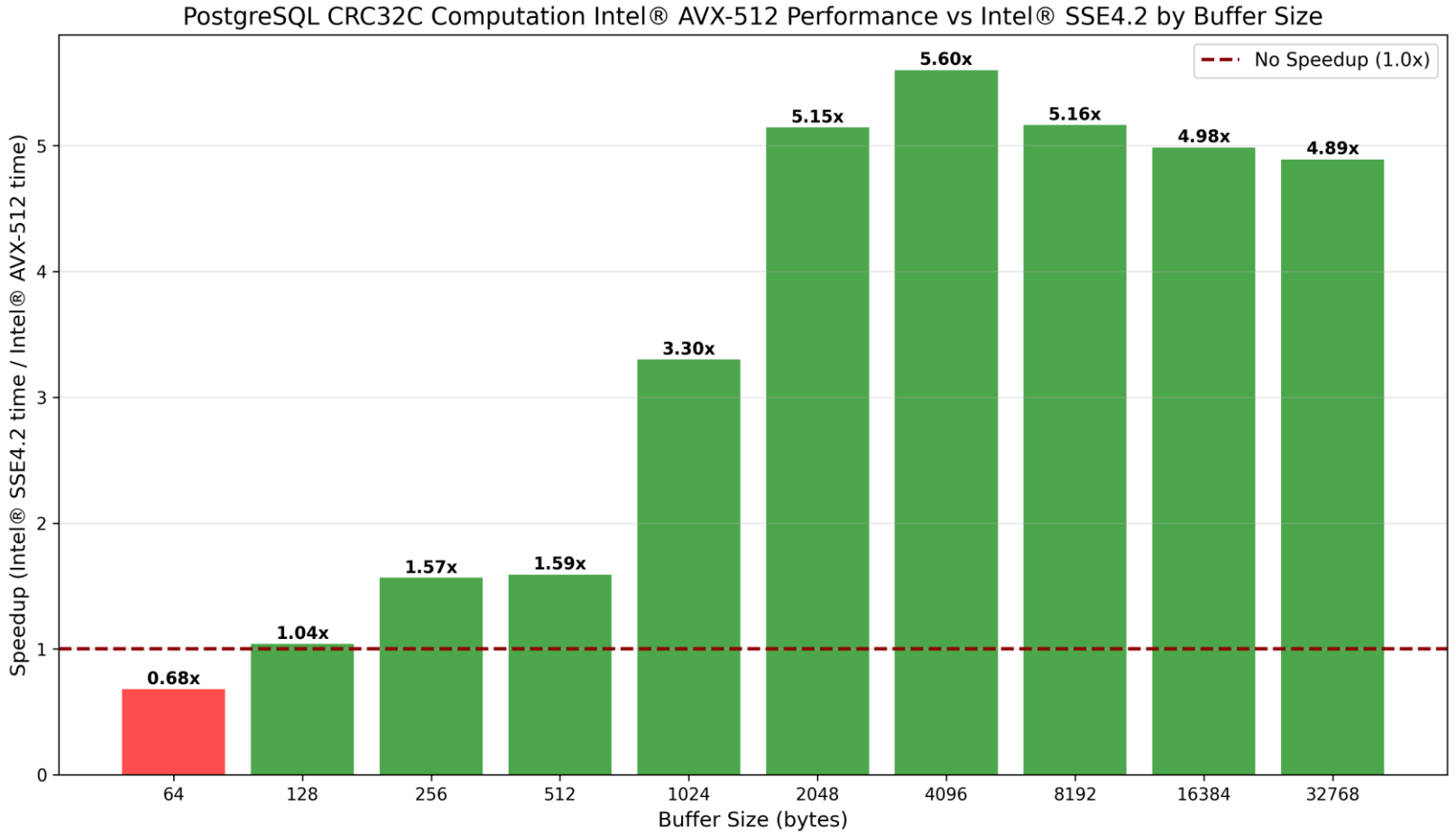
Code Example (Excerpt)

```
pg_crc32c
pg_comp_crc32c_avx512(pg_crc32c crc, const void *data, size_t length)
{
    // ...alignment and setup...
    while (len >= 64) {
        // Load 64 bytes
        __m512i x0 = _mm512_loadu_si512(buf);
        // Polynomial folding with carryless multiplication
        __m512i y0 = clmul_lo(x0, k);
        x0 = clmul_hi(x0, k);
        x0 = _mm512_ternarylogic_epi64(x0, y0, ...);
        // Advance to next chunk
        buf += 64;
        len -= 64;
    }
    // Reduce to 32 bits and finish
    // ...
    return crc;
}
```

Algorithm-Level Performance



PostgreSQL test_crc32c Extension



Why it Matters

Performance

Modernization

Reliability

Implications

Runtime check now required for builds that target SSE 4.2

In practice: WAL (arguably the most critical case)

The final computation with the 20-byte WAL header is inlined and unrolled when targeting (since commit e2809e3)

Benchmark Methodology & Configuration Details

We'll Spend 10 Minutes Here

High-Level Configuration

- Intel® 4th Generation Xeon® Scalable processor
- PostgreSQL 18 RC1
  - Compiled with Intel AVX-512
  - Compiled with forced SSE4.2 (same system)
- Minimal requirements:
  - Intel processor with Intel SSE4.2 support (Core i7/i5 2nd gen+, Xeon 5500+)
  - PCLMUL instruction support
  - Operating System: Linux variant with kernel 6.x or later

Methodology: Core Principles

Consistent

Repeatable

Automated

Consistent

Multiple Runs

- Rule out transient system effects

Observed Performance

- Leverage Performance Monitoring Tools
  - Intel: EMON

Repeatable

Configuration Management

- Revision Controlled

Apply Benchmarking Best Practices

- Performance Settings
- CPU and NUMA pinning for client and server

Automated

Build Scripts

- Revision Controlled

Validation for expected results

- Flag Outliers
- Multiple Runs

Real-world Results & Cost Analysis

We'll Spend 7 Minutes Here.

Microbenchmarks vs End-to-End

Purpose: Determine performance in a real-world scenario

- Microbenchmark gains are a small window
- Full picture requires end-to-end testing (as gains can be obscured by full-system performance)

Identified Workload: pg_basebackup

- Reasoning
  - A large database backup requires a substantial number of CRC32C computations
  - Backup operations are performed regularly (value)

PostgreSQL Backup (pg_basebackup) Methodology

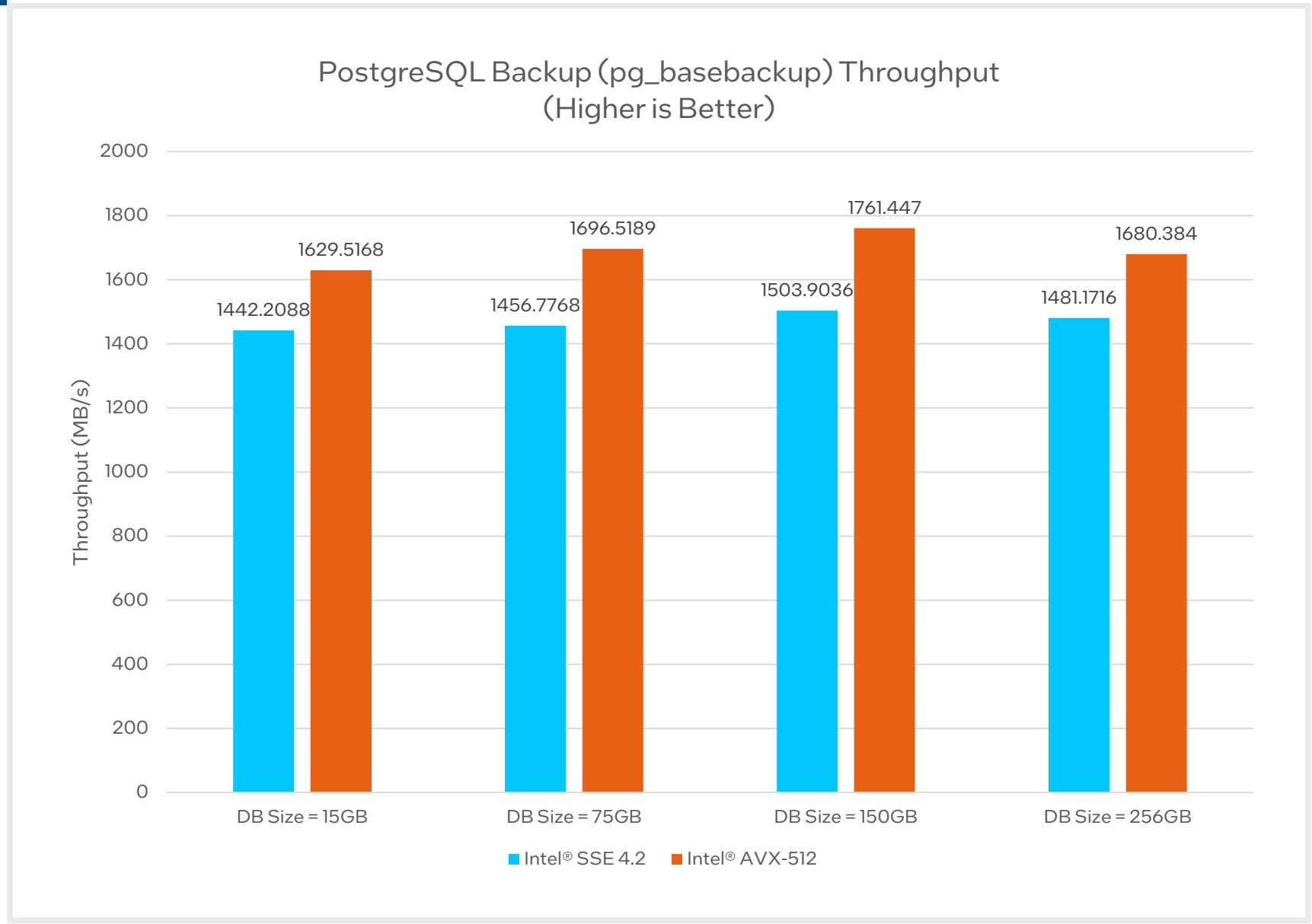
Progressively larger database sizes

- Small → Enterprise

Observations

- PostgreSQL processes database backups in 32KB chunks (most cases)
- pg_basebackup workload utilizes the default page size of 8KB (which processes four pages at a time)
  - Consistent computational workload

Performance of Intel SSE4.2 vs. Intel AVX-512



Real-world Implications for Time Savings

The Intel® AVX-512 execution shows that when compared to Intel® SSE4.2, the **performance improves on average by 15%**

Database Size	Intel® SSE4.2 Time	Intel® AVX-512 Time	Time Saved
15.0 GB	10.4 sec	9.2 sec	**1.2 sec**
75.0 GB	51.4 sec	44.2 sec	**7.2 sec**
150.0 GB	99.5 sec	85.2 sec	**14.3 sec**
256.3 GB	173.0	152.6 sec	**20.4 sec**

Translated to Production Benefits

- **Enterprise backups (256GB)**: Save **20+ seconds** per backup
- **Daily backups**: 20 seconds × 365 days = **2+ hours annually**
- **Multiple databases**: Savings multiply across database instances

Community Impact, Future Directions & Resources

We'll spend a Few Minutes Here

Acknowledgements

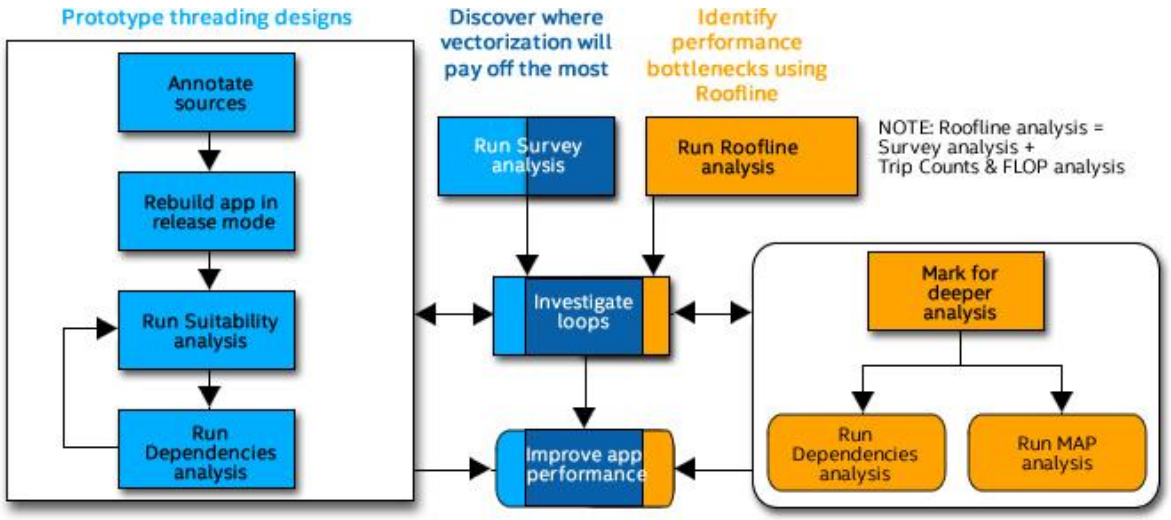
This patch was co-authored and reviewed by contributors from Intel and the PostgreSQL community.

- Credits to:
  - John Naylor (AWS)
  - Nathan Bossart (AWS)
  - Raghuveer Devulapalli (Intel)
  - Paul Amonson (Intel)
  - Matthew Sterrett (Intel)
  - Kelly McKeighan (Intel)
  - Akash Shankaran (Intel)

Future Directions

- **Intel AVX10.2:** Updates to all community contributions to implement the new instructions
- Pending Contributions

Contribution	Description	PostgreSQL Version	Impact
checksum	Enable auto-vectorization for checksum calculations	PG 19 > unreleased target Sept-2026: See Roadmap)	This gives about a 2x speedup in a synthetic benchmark for pg_checksum *> Full Blog Publication TBD*
NUM_XLOGINSERT_LOCKS	Increase NUM_XLOGINSERT_LOCKS	⏳ under review	Enables more concurrent inserters, reducing XLogInsertLock stalls
XLog Reservation	Lock-free XLog Reservation from WAL	⏳ under review	Enables WAL space reservation via lock-free atomic fetch-add
LWLock acquisition	Optimize shared LWLock acquisition for high-core-count systems	⏳ under review	Merges read and update operations for LW_SHARED lock state into single atomic operation.



Understand Vectorization with Intel® Advisor

Identify loops that will benefit most from vectorization

Identify what is blocking effective vectorization

Explore the benefit of alternative data reorganizations

Increase the confidence that vectorization is safe

Vectorization Resources for Intel® Advisor Users

https://www.intel.com/content/www/us/en/developer/articles/technical/advisor-vectorization-resources.html

Read the Blog

Intel® AVX-512 Accelerates PostgreSQL CRC32C Checksums

Other Intel Sessions

Accelerating PostgreSQL Backups with Intel® QuickAssist Technology