# Load Tests for AMD CPU 3DvCache

## Introduction

We have a golden opportunity (thanks to our customer) to finally test the processing speed versus the L3 data access speed of CPUs for SQL Server workload.

For this reason, we will set up temporary LAB with two identical servers with different CPUs as follows:

[AMD EPYC™ 9184X](https://www.techpowerup.com/cpu-specs/epyc-9184x.c3255#:~:text=EPYC%209184X%20has%20768%20MB%20of%20L3%20cache,at%20AMD%2C%20but%20at%20the%20foundry%20of%20TSMC.)   16 Core Base clock 3.55GHz TB 4.2/3.85 All core boost L3 768MB

[AMD Workload Affinity](https://www.amd.com/en/products/specifications/server-processor.html): CAE|CFD|FEA , EDA , HPC

[AMD EPYC™ 9174F](https://www.techpowerup.com/cpu-specs/epyc-9174f.c2918)   16 Core Base clock 4.1GHz TB 4.4/ 4.15 All core boost L3 256MB

Both CPUs have same core (ZEN 4 Genoa) architecture, L1 64KB and L2 1MB per core and 8 CCD blocks according to [Link](https://hothardware.com/reviews/amd-genoa-data-center-cpu-launch#:~:text=In%20other%20words%2C%20an%20eight%20CCD%20configuration%20%28e.g.,to%20GMI7%20links%2C%20leaving%20the%20other%20four%20unused.)

[AMD Workload Affinity](https://www.amd.com/en/products/specifications/server-processor.html): EDA, High-speed data mgmt (NR|RDBMS) , Per core CAE|CFD|FEA

**EDA (Electronic Design Automation)**: EDA involves using software tools to design electronic systems such as integrated circuits and printed circuit boards. These tools help with designing, simulating, and verifying the electronic parts and are crucial in developing technology that requires high-speed and complex circuitry. AMD EPYC processors enhance EDA workloads by improving simulation speeds and design verification processes due to their high core counts and advanced caching capabilities.

**High-Speed Data Management (Non-relational Databases or NR)**: This refers to managing large volumes of data at high speeds, often using non-relational databases. These systems are designed to handle different types of data formats and structures at scale, which is vital for applications like big data analytics and IoT. AMD processors provide the necessary computational power and memory efficiency required for these demanding tasks.

**CAE (Computer-Aided Engineering)**: CAE includes software tools used for engineering tasks like simulation, analysis, and manufacturing. Subtypes like CFD (Computational Fluid Dynamics) and FEA (Finite Element Analysis) involve simulating fluid movements and structural stresses, respectively. AMD's processors are beneficial for these tasks due to their ability to process large datasets and perform complex calculations efficiently.

**CFD (Computational Fluid Dynamics)** and **FEA (Finite Element Analysis)**: Both are specialized areas under CAE. CFD focuses on fluid dynamics and thermal simulation, while FEA is used for structural analysis, dealing with stress, vibration, and mechanical strength of materials under various conditions. The high performance of AMD EPYC processors, especially in terms of parallel processing capabilities, significantly reduces the time for these simulations.

The main objective of this test is to provide hard data for discussion about CPU processing speed versus data access speed. Here’s how to consider each factor for the respective CPU:

[AMD EPYC™ 9174F](https://www.techpowerup.com/cpu-specs/epyc-9174f.c2918)

Pros: Faster processing of individual instructions, which can be beneficial for quick transaction processing thanks to higher base frequency 4.1GHz and Turbo Boost 4.4GHz SingleCore / 4.1GHz All Core

Cons: Less L3 cache may lead to more frequent RAM access, which could become a bottleneck, especially if transactions frequently access the same data.

[AMD EPYC™ 9184X](https://www.techpowerup.com/cpu-specs/epyc-9184x.c3255#:~:text=EPYC%209184X%20has%20768%20MB%20of%20L3%20cache,at%20AMD%2C%20but%20at%20the%20foundry%20of%20TSMC.)

Pros: More data can be kept close to the CPU, thanks to the triple size (256MB < 768MB) of the L3 cache, reducing latency from RAM accesses. This can be particularly beneficial in OLTP, where transactions often access small, frequently used data sets. In OLAP scenarios, the CPU will need fewer roundtrips to RAM to finish the large read operation.

Cons: Lower clock speed means individual instructions are processed more slowly, which could potentially slow down transaction processing, although this might be mitigated by the reduced need for RAM access.

### OLTP and OLAP SQL Workload Considerations:

* **OLTP (Online Transaction Processing):**
  + **Core and Thread Count:** OLTP systems benefit from more cores and threads as they handle numerous short transactions that can be processed in parallel.
  + **Clock Speed:** Important for small single-thread queries but not as crucial as the number of cores in mixed workload scenarios (usuall these days). A balanced approach between speed and number of cores is ideal.
  + **Cache Size:** Larger cache can be beneficial for frequently accessed data. CPU spends less time idling while waiting for data to be fetched from RAM, which leads to better utilization of the CPU’s processing power.
* **OLAP (Online Analytical Processing):**
  + **Core and Thread Count:** High core and thread count is crucial for handling complex queries across large datasets.
  + **Clock Speed:** While important, the ability to handle parallel tasks is more critical, hence, a focus on multi-core performance.
  + **Cache Size:** A larger cache is beneficial for storing parts of the large datasets typically processed in OLAP tasks. It could offset slower memory modules used on the server as we need fewer roundtrips to RAM from the CPU.

**Our Expectations:**

* L3 cache can positively impact OLTP performance, but typically not to the same extent as in core frequency (which affects increased IPC). This is because OLTP workloads involve a large number of small, independent transactions.
* With data more readily available in the cache, the CPU can process transactions more efficiently.
* A larger cache helps in reducing latency by keeping frequently accessed data closer to the CPU, which can be beneficial for OLTP systems with repetitive queries.
* In the context of CPU Clock Speed, having a high CPU clock speed is still valuable, as it allows for quicker processing of each transaction. However, the advantage gained from higher clock speeds can be diminished if the CPU is frequently waiting on memory access. A larger cache mitigates this bottleneck, ensuring that the CPU’s potential is more fully utilized.

## Testing Methodology

We want to run a series of tests in a regulated environment to collect as many data points as possible on various load tests.

Every test will run a specific number of executions (around 10,000) to provide stable average values, removing the possible effect of monitoring tools (observer overhead).

We will run the tests on different CPU configuration options for both processors as follows:

1. Turbo Boost **Disabled**
2. Turbo Boot **Enabled**
3. Turbo Boot **Enabled AVX 512 Enabled**

Other BIOS Settings that will have the same configuration all tests:

AMD SMT **Disabled**

NPS **4**

L3 to NUMA **Enabled**

No NUMA spanning

Hi-Performance Power profile

SVM Mode **Disabled**

OS Power plan will be set to High Performance. SQL Server 2022 Developer will not run any trace flags and will be configured with MAXDOP 8 and CTFP 15.

Tests are specifically designed to show L3 cache misses on 74F CPU. So we expect to see a performance drop when we work with tables larger than L3 per CCD on this CPU. 84X CPU in comparison should not show this performance drop since 56MB table is still inside the capacity of L3 cache of this processor.

Our tests minimize CPU computation activity (except 1.C test specifically designed to test this) to decrease its impact on query elapsed time in favor of the data load part where cache size importance may show.

Monitoring tools will be bounded via resource governor to a different NUMA node than workload threads. We will also collect specific wait stats per query execution for further analysis.

**1. Single-Threaded Workload Test:**

**A.** Single thread access of 1 session on one bounded core (using Resource Governor) at a time, accessing a small table 28MB & 56MB that can fit into the cache of 84X CPU. To confirm the impact of CPU Frequency.

* Expected Result > Low Latch Latency, Core frequency bound workload should be faster on 74F CPU

The following code logic will also be applied for 56MB/224MB/448MB table size with its own procedure call. Sizes have been specifically chosen to not overlap cache size and provide some buffer for internal memory/cache requirements.

/\*28MB Test\*/

CREATE TABLE Test28MB (Id INT IDENTITY, Payload CHAR(7996)); /\*1 row 8KB\*/

GO

/\*3584 rows should be sufficient to create 28MB (3584 \* 8KB = ~28MB)\*/

INSERT Test28MB

VALUES ('A'); /\*Repeat 3584 times\*/

GO 3584

CREATE PROC dbo.Test28MB\_Proc

@TestRun NVARCHAR(128)

AS

BEGIN

SET NOCOUNT ON

DECLARE @ProcName NVARCHAR(128) = 'Test28MB\_Proc'

DECLARE @StartTime DATETIME2

DECLARE @Wait\_Time\_ms BIGINT

DECLARE @Signal\_WTime\_ms BIGINT

DECLARE @LastInsertedID INT;

DECLARE @Before AS TABLE (

[session\_id] [SMALLINT] NOT NULL,

[wait\_type] [NVARCHAR](256) NOT NULL,

[waiting\_tasks\_count] [BIGINT] NOT NULL,

[wait\_time\_ms] [BIGINT] NOT NULL,

[max\_wait\_time\_ms] [BIGINT] NOT NULL,

[signal\_wait\_time\_ms] [BIGINT] NOT NULL

)

DECLARE @After AS TABLE (

[session\_id] [SMALLINT] NOT NULL,

[wait\_type] [NVARCHAR](256) NOT NULL,

[waiting\_tasks\_count] [BIGINT] NOT NULL,

[wait\_time\_ms] [BIGINT] NOT NULL,

[max\_wait\_time\_ms] [BIGINT] NOT NULL,

[signal\_wait\_time\_ms] [BIGINT] NOT NULL

)

/\* Capture initial wait stats \*/

INSERT @Before

SELECT \*

FROM sys.dm\_exec\_session\_wait\_stats WHERE session\_id = @@SPID

SET @StartTime = SYSDATETIME()

/\* Actual Workload\*/

SELECT \* FROM Test28MB

WHERE ID < 0 /\*to touch the data but not send anything to client (prevent ASYNC\_NETWORK\_IO)\*/

OPTION (MAXDOP 1) ;

/\*Collect Metrics\*/

DECLARE @ElapsedTime\_us BIGINT = (SELECT DATEDIFF(MICROSECOND,@StartTime,SYSDATETIME()) )

INSERT @After

SELECT

session\_id,

wait\_type,

waiting\_tasks\_count,

wait\_time\_ms,

max\_wait\_time\_ms,

signal\_wait\_time\_ms

FROM sys.dm\_exec\_session\_wait\_stats

WHERE session\_id = @@SPID

/\*Calculate wait times\*/

SELECT @Wait\_Time\_ms = SUM(COALESCE(a.wait\_time\_ms, b.wait\_time\_ms) - ISNULL(b.wait\_time\_ms, 0)),

@Signal\_WTime\_ms = SUM(COALESCE(a.signal\_wait\_time\_ms, b.signal\_wait\_time\_ms) - ISNULL(b.signal\_wait\_time\_ms, 0))

FROM @Before AS b

RIGHT JOIN @After AS a

ON a.wait\_type = b.wait\_type

/\*Insert overall test run results\*/

INSERT SMT\_Collector.LoadTests.ProctestRuns

SELECT @TestRun,

@ProcName,

@@SPID,

(@ElapsedTime\_us) AS ElapsedTime\_us ,

(@Wait\_Time\_ms - @Signal\_WTime\_ms) \*1000 AS Res\_Wtime\_us,

@Signal\_WTime\_ms \*1000 AS Signal\_WTime\_us,

@ElapsedTime\_us - (@Wait\_Time\_ms \*1000) AS CPU\_Time\_us,

SYSDATETIME()

/\*Capture the last inserted ID\*/

SET @LastInsertedID = SCOPE\_IDENTITY();

/\*Insert detailed wait stats\*/

INSERT SMT\_Collector.LoadTests.ProctestRuns\_Waits

SELECT @LastInsertedID

, @@SPID

, COALESCE (a.wait\_type , b.wait\_type) AS Wait\_type

, COALESCE(a.waiting\_tasks\_count, b.waiting\_tasks\_count) - ISNULL(b.waiting\_tasks\_count, 0) AS waiting\_tasks\_count

, COALESCE(a.wait\_time\_ms, b.wait\_time\_ms) - ISNULL(b.wait\_time\_ms, 0) AS wait\_time\_ms

, COALESCE(a.max\_wait\_time\_ms, b.max\_wait\_time\_ms) AS max\_wait\_time\_ms

, COALESCE(a.signal\_wait\_time\_ms, b.signal\_wait\_time\_ms) - ISNULL(b.signal\_wait\_time\_ms, 0) AS signal\_wait\_time\_ms

,'Diff: ' +CAST(COALESCE(a.max\_wait\_time\_ms, b.max\_wait\_time\_ms) - ISNULL(b.max\_wait\_time\_ms, 0)AS nvarchar(256))

+' Aft:' + CAST(COALESCE(a.max\_wait\_time\_ms, 0 ) AS nvarchar(256))

+' Bfr:'+ CAST(COALESCE (b.max\_wait\_time\_ms,0) AS nvarchar(256)) max\_wait\_diff\_m\_descr

,SYSDATETIME()

FROM @Before AS b

RIGHT JOIN @After AS a

ON a.wait\_type = b.wait\_type

END

GO  
/\*Running the tests\*/

EXEC dbo.Test28MB\_Proc 'DemotestRun1';

GO 10000

**B.** Single thread access of 16 concurrent sessions on all cores at the time accessing a small table 28MB & 56MB that can fit into the cache of 84X CPU. No Resource Governor binding. To confirm the impact of CPU Frequency.

Expected Result > Low Latch Latency, Core frequency bound workload, Context switches may grow

Same code as 1.A executed from SQLQueryStress on 16 threads and 1000 times per thread

Switches at higher rates, more signal waits, 74F should dominate with its high turbo boost frequencies (Turbo boost frequency difference 300MHz)

**C.** Compute Power where data are not needed. Here we just compute the power for 15 minutes, then compute final score and average power computations per millisecond. Bounded on specific core using Resource Governor.

Expected Result > Higher frequency core will provide faster results.

-- Check and drop #Waits table if it exists

IF OBJECT\_ID('tempdb..#Waits') IS NOT NULL

BEGIN

DROP TABLE #Waits;

END

-- Check and drop #Score table if it exists

IF OBJECT\_ID('tempdb..#Score') IS NOT NULL

BEGIN

DROP TABLE #Score;

END

-- Single core test

SET NOCOUNT ON

DECLARE @Before AS TABLE (

[session\_id] [smallint] NOT NULL,

[wait\_type] [nvarchar](256) NOT NULL,

[waiting\_tasks\_count] [bigint] NOT NULL,

[wait\_time\_ms] [bigint] NOT NULL,

[max\_wait\_time\_ms] [bigint] NOT NULL,

[signal\_wait\_time\_ms] [bigint] NOT NULL

)

DECLARE @After AS TABLE (

[session\_id] [smallint] NOT NULL,

[wait\_type] [nvarchar](256) NOT NULL,

[waiting\_tasks\_count] [bigint] NOT NULL,

[wait\_time\_ms] [bigint] NOT NULL,

[max\_wait\_time\_ms] [bigint] NOT NULL,

[signal\_wait\_time\_ms] [bigint] NOT NULL

)

INSERT @Before

SELECT \*

FROM sys.dm\_exec\_session\_wait\_stats WHERE session\_id = @@SPID

/\* Actual Workload\*/

DECLARE @T DATETIME, @F BIGINT = 0, @P BIGINT,@SecondsRun BIGINT = 900;

SET @T = GETDATE();

SET @T = DATEADD(SECOND,@SecondsRun,@T) /\*One Minute Test\*/

WHILE @T>GETDATE()

BEGIN

SET @P = POWER(2,30)

SET @F=@F+1

END

SELECT @F / 1000000 Score\_Milions , @F/ (@SecondsRun\*1000) AvgPerfPerms INTO #Score

INSERT @After

SELECT \*

FROM sys.dm\_exec\_session\_wait\_stats WHERE session\_id = @@SPID

SELECT COALESCE (a.wait\_type , b.wait\_type) AS Wait\_type

, COALESCE(a.waiting\_tasks\_count, b.waiting\_tasks\_count) - ISNULL(b.waiting\_tasks\_count, 0) AS waiting\_tasks\_count

, COALESCE(a.wait\_time\_ms, b.wait\_time\_ms) - ISNULL(b.wait\_time\_ms, 0) AS wait\_time\_ms

, COALESCE(a.max\_wait\_time\_ms, b.max\_wait\_time\_ms) AS max\_wait\_time\_ms

,'Diff: ' +CAST(COALESCE(a.max\_wait\_time\_ms, b.max\_wait\_time\_ms) - ISNULL(b.max\_wait\_time\_ms, 0)AS nvarchar(256))

+' Aft:' + CAST(a.max\_wait\_time\_ms AS nvarchar(256))

+' Bfr:'+ CAST(b.max\_wait\_time\_ms AS nvarchar(256)) max\_wait\_diff\_m\_descr

, COALESCE(a.signal\_wait\_time\_ms, b.signal\_wait\_time\_ms) - ISNULL(b.signal\_wait\_time\_ms, 0) AS signal\_wait\_time\_ms

INTO #Waits

FROM @Before AS b

RIGHT JOIN @After AS a

ON a.wait\_type = b.wait\_type

SELECT \* FROM #Waits AS W ORDER BY W.wait\_time\_ms DESC

SELECT \* FROM #Score AS S

**D.** Single thread access of 1 session on one bounded core (using Resource Governor) at a time, accessing a static set of data larger than CPU L3 cache from the tables of size 244MB and 448MB.

Expected Result > Higher cache latency due to L3 cache misses, 84X should perform better if we will be able to cause many RAM roundtrips.

Same code as 1.A just with bigger tables (224MB & 448MB).

**E. Loop Join.** This test has been contributed by Joe Chang ([li](https://www.linkedin.com/in/joe-chang-174a451/)) and focuses on nested loops join between smaller and bigger tables using randomized patterns. The catch is that table A4-A6 have different sizes and some may not fit into L3 of the CPU. So we should see more RAM round trips as code will start to work with bigger tables.

Table MB pages rows rang

A6 128 16384 1,654,784 103424

A5 386 49152 4,964,352 310272

A4 1536 196608 19,857,408 1241088

--0xCC18E089BC035491 --A4

DECLARE @I int, @R int SELECT @I = 1+1241088\*RAND(CHECKSUM(NEWID())); SELECT @R=MAX(b.TI) FROM dbo.A4 a INNER LOOP JOIN dbo.A b ON b.AID = a.AID WHERE a.GID = @I OPTION(MAXDOP 1)

GO 10000

--0xF550DF6E596250B8 --A5

DECLARE @I int, @R int SELECT @I = 1+310272\*RAND(CHECKSUM(NEWID())); SELECT @R=MAX(b.TI) FROM dbo.A5 a INNER LOOP JOIN dbo.A b ON b.AID = a.AID WHERE a.GID = @I OPTION(MAXDOP 1)

GO 10000

--0xE36ADC079C2D052C -- A6

DECLARE @I int, @R int SELECT @I = 1+103424\*RAND(CHECKSUM(NEWID())); SELECT @R=MAX(b.TI) FROM dbo.A6 a INNER LOOP JOIN dbo.A b ON b.AID = a.AID WHERE a.GID = @I OPTION(MAXDOP 1)

GO 10000

Expected Result > Higher cache latency due to L3 cache misses, 84X should perform better if we will be able to cause many RAM roundtrips.

2. **Multi-Threaded Workload Test:**

A. Single parallel query executed 1000 times on a moderately sized table to see if lower frequency and bigger L3 cache outperforms faster cores waiting on RAM in parallel operations.

On 74F we expect RAM roundtrip overhead in this test causing lower performance against 84X CPU

CREATE PROC dbo.Test448MB\_Proc\_Multi

@TestRun NVARCHAR(128) , @MAXDOP CHAR(2) = '16'

AS

BEGIN

SET NOCOUNT ON

DECLARE @ProcName NVARCHAR(128) = 'Test448MB\_Proc\_Multi'

DECLARE @StartTime DATETIME2

DECLARE @Wait\_Time\_ms BIGINT

DECLARE @Signal\_WTime\_ms BIGINT

DECLARE @LastInsertedID INT;

DECLARE @Before AS TABLE (

[session\_id] [SMALLINT] NOT NULL,

[wait\_type] [NVARCHAR](256) NOT NULL,

[waiting\_tasks\_count] [BIGINT] NOT NULL,

[wait\_time\_ms] [BIGINT] NOT NULL,

[max\_wait\_time\_ms] [BIGINT] NOT NULL,

[signal\_wait\_time\_ms] [BIGINT] NOT NULL

)

DECLARE @After AS TABLE (

[session\_id] [SMALLINT] NOT NULL,

[wait\_type] [NVARCHAR](256) NOT NULL,

[waiting\_tasks\_count] [BIGINT] NOT NULL,

[wait\_time\_ms] [BIGINT] NOT NULL,

[max\_wait\_time\_ms] [BIGINT] NOT NULL,

[signal\_wait\_time\_ms] [BIGINT] NOT NULL

)

/\* Capture initial wait stats \*/

INSERT @Before

SELECT \*

FROM sys.dm\_exec\_session\_wait\_stats WHERE session\_id = @@SPID

SET @StartTime = SYSDATETIME()

/\* Actual Workload\*/

EXECUTE('SELECT \* FROM Test448MB

WHERE ID < 0 /\*to touch the data but not send anything to client (prevent ASYNC\_NETWORK\_IO)\*/

OPTION (MAXDOP '+@MAXDOP+') ;' )

/\*Collect Metrics\*/

DECLARE @ElapsedTime\_us BIGINT = (SELECT DATEDIFF(MICROSECOND,@StartTime,SYSDATETIME()) ElapsedTime\_us)

INSERT @After

SELECT

session\_id,

wait\_type,

waiting\_tasks\_count,

wait\_time\_ms,

max\_wait\_time\_ms,

signal\_wait\_time\_ms

FROM sys.dm\_exec\_session\_wait\_stats

WHERE session\_id = @@SPID

/\*Calculate wait times\*/

SELECT @Wait\_Time\_ms = SUM(COALESCE(a.wait\_time\_ms, b.wait\_time\_ms) - ISNULL(b.wait\_time\_ms, 0)),

@Signal\_WTime\_ms = SUM(COALESCE(a.signal\_wait\_time\_ms, b.signal\_wait\_time\_ms) - ISNULL(b.signal\_wait\_time\_ms, 0))

FROM @Before AS b

RIGHT JOIN @After AS a

ON a.wait\_type = b.wait\_type

/\*Insert overall test run results\*/

INSERT SMT\_Collector.LoadTests.ProctestRuns

SELECT @TestRun,

@ProcName,

@@SPID,

(@ElapsedTime\_us) AS ElapsedTime\_us ,

(@Wait\_Time\_ms - @Signal\_WTime\_ms) \*1000 AS Res\_Wtime\_us,

@Signal\_WTime\_ms \*1000 AS Signal\_WTime\_us,

(@ElapsedTime\_us /\*Total code runtime\*/

- (

(@Wait\_Time\_ms /@MAXDOP) /\*Average wait per thread\*/

\*1000 /\*convert to microsec\*/

)

)

\* @MAXDOP /\*Account for all thread CPU util\*/AS CPU\_Time\_us,

SYSDATETIME()

/\*Capture the last inserted ID\*/

SET @LastInsertedID = SCOPE\_IDENTITY();

/\*Insert detailed wait stats\*/

INSERT SMT\_Collector.LoadTests.ProctestRuns\_Waits

SELECT @LastInsertedID

, @@SPID

, COALESCE (a.wait\_type , b.wait\_type) AS Wait\_type

, COALESCE(a.waiting\_tasks\_count, b.waiting\_tasks\_count) - ISNULL(b.waiting\_tasks\_count, 0) AS waiting\_tasks\_count

, COALESCE(a.wait\_time\_ms, b.wait\_time\_ms) - ISNULL(b.wait\_time\_ms, 0) AS wait\_time\_ms

, COALESCE(a.max\_wait\_time\_ms, b.max\_wait\_time\_ms) AS max\_wait\_time\_ms

, COALESCE(a.signal\_wait\_time\_ms, b.signal\_wait\_time\_ms) - ISNULL(b.signal\_wait\_time\_ms, 0) AS signal\_wait\_time\_ms

,'Diff: ' +CAST(COALESCE(a.max\_wait\_time\_ms, b.max\_wait\_time\_ms) - ISNULL(b.max\_wait\_time\_ms, 0)AS nvarchar(256))

+' Aft:' + CAST(COALESCE(a.max\_wait\_time\_ms, 0 ) AS nvarchar(256))

+' Bfr:'+ CAST(COALESCE (b.max\_wait\_time\_ms,0) AS nvarchar(256)) max\_wait\_diff\_m\_descr

,SYSDATETIME()

FROM @Before AS b

RIGHT JOIN @After AS a

ON a.wait\_type = b.wait\_type

END

GO

/\*Parallelism Test using SQLQueryStress\*/

Test448MB\_Proc\_Multi 'ParallelTest\_MAXDOP8' ,8

3 **Mixed Workload Test using HAMMER DB:**

* Simulate a real-world mixed workload that includes a combination of read and write operations, and different types of queries. This would provide insights into how the CPUs perform under typical varied database usage.

## Monitoring

Most of the monitoring duties will be done via [Woodler SMT](https://www.woodler.eu/smt). For mixed workload tests we will use [loadtest](https://www.woodler.eu/load-tests) solution.

For synthetic tests 1,2 and 3 we will use combination of sys.procedure\_stats & build in monitoring from procedure itself which allows us to examine specific executions and related wait statistics.

Base metrics we want to review:

CPU Clock Performance Metrics:

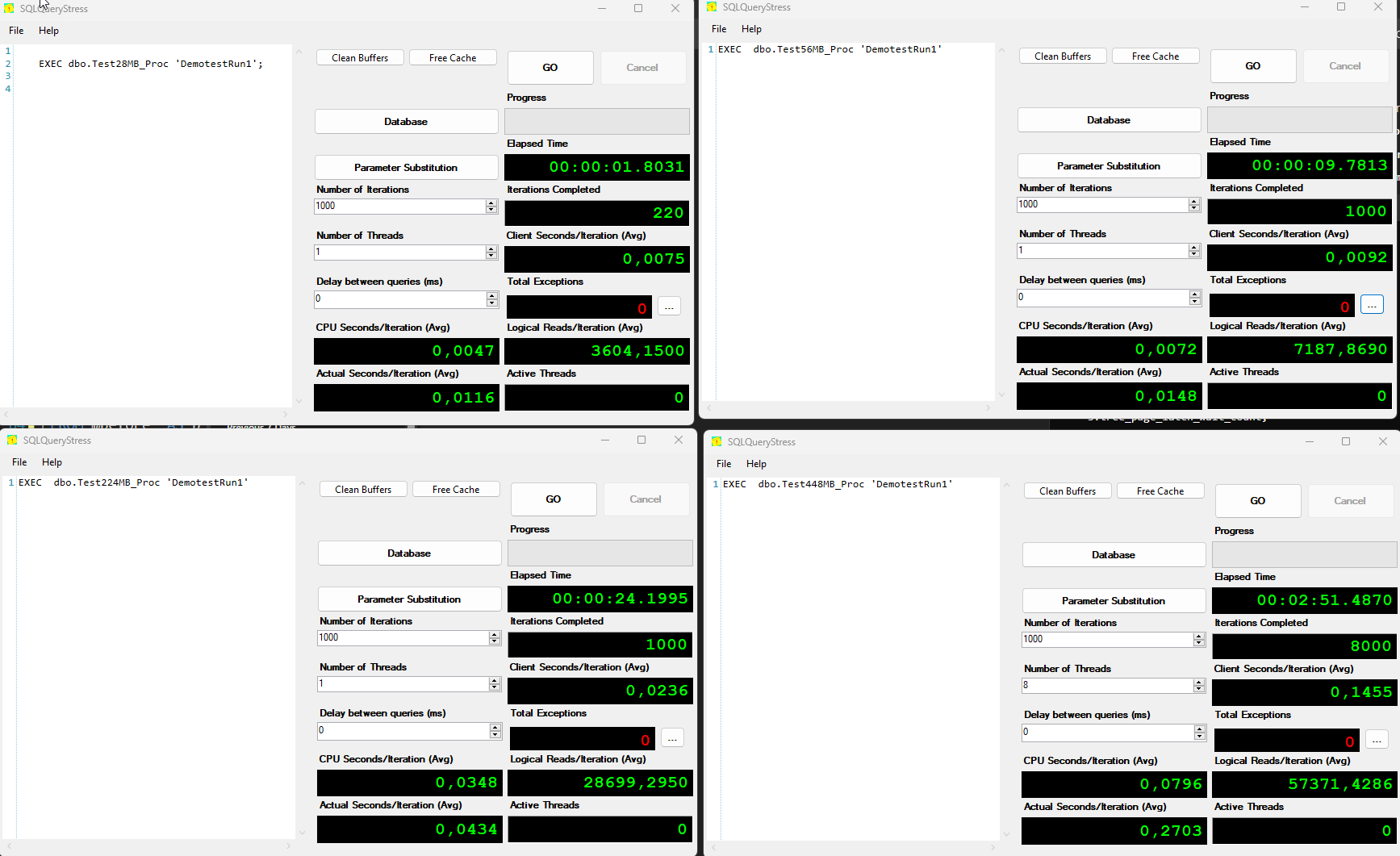
* **Signal Thread Waits and Worker Time:** Measures the efficiency of single-threaded operations.
* **Batch Requests per Second:** This can give an indication of how many transactions the server is handling, which is useful for understanding throughput.
* **Query Elapsed Time:** Provides direct insight into how long queries take to execute, which is a key measure of CPU performance. (avg and totals)
* **CPU Utilization:** Overall CPU usage can indicate whether the CPU is a bottleneck.
* **Context Switches per Second:** High context switching can indicate that the CPU is being overutilized or that threads are not being executed efficiently.
* **CPU Queue Length:** Indicates if there are more threads than the CPU can handle simultaneously, which can be an indirect measure of whether the CPU's clock speed or cache size is a bottleneck

#### CPU Cache and RAM Pressure Metrics:

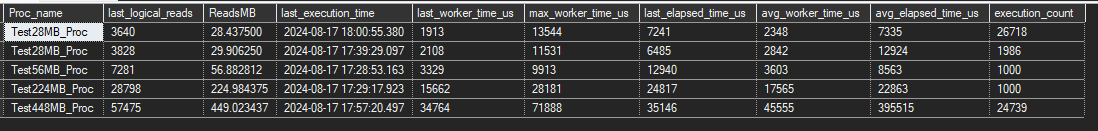
* **Latches Duration and Index Average Latch Time:** These are useful for understanding the time spent waiting on memory, which can be a bottleneck if the cache is insufficient.
* **Resource Semaphore Waits:** Indicates pressure on memory resources, which can be a sign of insufficient cache or high RAM dependency.
* **Execution Times and Resource Wait Statistics:** Focusing specifically on CPU-related wait types can give insights into how the CPU is handling the load and where it might be bottlenecked.
* **Using AMD** [**uProof**](https://www.amd.com/en/developer/uprof.html)
  + **l1** – L1 cache related metrics (DC access and IC Fetch miss ratio)
  + **l2** – L2D and L2I cache related access/hit/miss metrics
  + **l3** – L3 cache metrics like L3 Access, L3 Miss, and Average Miss latency
  + **dc** – advanced caching metrics such as DC refills by source (supported only on AMD “Zen3” and AMD “Zen4” processors)
  + **memory** – approximate memory read and write bandwidths in GB/s for all the channels
  + **swpfdc** – software prefetch data cache from various nodes and CCX (supported only on AMD “Zen3” and AMD “Zen4” processors)
  + **hwpfdc** – hardware prefetch data cache from various nodes and CCX (supported only on AMD “Zen3” and AMD “Zen4” processors)
  + **pipeline\_util** – top-down metrics to visualize the bottlenecks in the CPU pipeline (supported only on AMD “Zen4” processors)

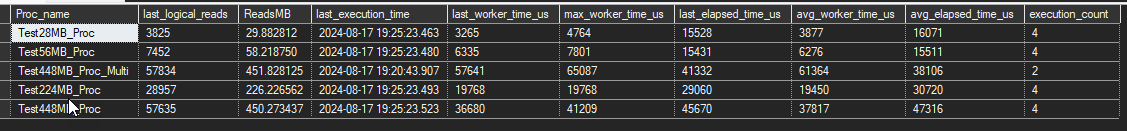
## Results Evaluation - TBD

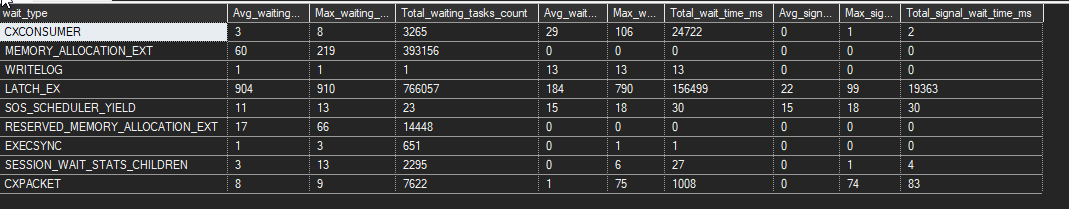
Combination of graph results from SMT and other collected data in PowerBI reporting.

For test generation we will have also partial results from SQLQueryStress Tool.   


For tests 1, 2 and 3 we will also have tabular data like this:

From sys.procedure stats we will have output like this:

And wait statistics:

Test queries cumulative statistics.

A screen shot of a black screen

Description automatically generated All that information will also be available per query execution to provide details in the Powerbi report if needed.

With hard data we should be able to provide clearer differences in CPU behavior under same circumstances and describe benefits of each one.