Behind the Report:

Comparing the Relative Performance of Dell™ PowerEdge™ Servers Powered by Intel® Xeon® 6 Processors

This methodology outlines the procedures that Prowess Consulting followed to determine which workload types are best targeted to each of Intel Xeon 6 processors with E-cores, P-cores, and AP-cores.

Summary

In this study, Prowess Consulting evaluated how Intel Xeon processors with Efficient-cores (E-cores), Performance-cores (P-cores), and Advanced Performance–cores (AP-cores) behave when running representative enterprise workloads on Dell PowerEdge servers. The goal of the testing was to determine how performance characteristics, scalability, and efficiency vary when workloads are intentionally matched to the processor core type best suited to their execution profile. By holding system software, operating system configuration, and test methodology constant across platforms, our testing isolated the impact of core architecture on workload behavior.

Testing Summary

We conducted testing on three PowerEdge server configurations, each optimized for Intel Xeon 6 processors with specific core types: a systems with E-cores, a system with P-cores, and a system with AP-cores. All systems ran Red Hat® Enterprise Linux® 10 and were configured with comparable baseline software, networking, and storage to ensure consistency. We executed each workload on a dedicated system to reflect realistic deployment scenarios and avoid crossworkload interference. For our testing, we captured performance, scalability, and system behavior using standard monitoring and logging tools, enabling repeatable measurement of throughput, efficiency, and aggregate work completed across the tested platforms.

Our engineers tested the following workloads:

- **WRK2** to evaluate concurrency and efficiency under sustained request load
- **ResNet®-50** to measure CPU-based AI inference performance across precision modes
- **Nanoscale Molecular Dynamics (NAMD)** to assess multithreaded, compute-intensive simulation throughput[1]

Table 1 | Server configuration

Model	Dell PowerEdge R770 (with E-Cores)	Dell PowerEdge R770 (with P-Cores)	Dell PowerEdge R770AP (with AP-cores)
CPU	2 x Intel Xeon 6740E processors (96 cores each)	2 x Intel Xeon 6767P processors (64 cores each)	2x Intel Xeon 6978P processors (120 cores each)
Memory	2,048 GB Micron® MTC40F2046S1RC64BD2, DDR5, 5,200 megatransfers per second (MT/s) RDIMM	2,048 GB Micron MTC40F2046S1RC64BD2, DDR5, 5,200 MT/s (max 6,400 MT/s) RDIMM	1,536 GB dual-rank SK hynix® HMCT04MHBRC470N, DDR5, 6,400 MT/s RDIMM
Power	PSU 1: Delta 1,500 W PSU 2: Delta 1,500 W	PSU 1: Delta 1,500 W PSU 2: Delta 1,500 W	PSU 1: Liteon 2,400 W PSU 2: Liteon 2,400 W
Networking	4-port, 25 Gb Broadcom® SFP 57504S OCP network interface controller (NIC)	4-port, 25 Gb Broadcom SFP 57504S OCP NIC	2-port, 100 Gb Mellanox® ConnectX®-6 Dx NIC 2 x 200 Gb Broadcom BCM57608 OCP Ethernet NIC—8C:84:74:9D:89:16
Storage	2 x 480 GB Micron EC NVM Express® (NVMe®) ISE 7450 RI M.2 80 (Dell™ Boot-Optimized Server Storage [BOSS] attach) 8 x 3.84 TB Micron MTFDDAK3T8TGA-1B (Dell™ PowerEdge RAID Controller [PERC] H365i attach) 4 x 1.92 TB KIOXIA® DC NVMe CD8 U.2 (CPU direct attach)	2 x 480 GB Micron EC NVMe ISE 7450 RI M.2 80 (Dell BOSS attach) 8 x 3.84 TB Micron MTFDDAK3T8TGA-1B (Dell PERC H365i attach) 4 x 1.92 TB KIOXIA DC NVMe CD8 U.2 1.92TB (CPU direct attach)	2 x 480 GB Micron EC NVMe ISE 7450 RI M.2 80 (Dell BOSS attach) 2 x 1.6 TB Samsung® DC NVMe PM9D5a MU U.2 (CPU direct attach)

Testing Procedures

For this testing, Prowess Consulting completed the following test steps.

Red Hat Enterprise Linux 10 Installation

For each workload, install Red Hat Enterprise Linux 10.

1. Download Red Hat Enterprise Linux 10, and then create a bootable USB using the downloaded files.
2. Boot to Red Hat Enterprise Linux 10.
3. Select **Install Red Hat Enterprise Linux 10.0**.
4. From the **Welcome to Red Hat Enterprise Linux 10.0** page, select **English**, and then click **Continue**.
5. From the **Root Account** page, select **Enable root account**, and then, in the **Root Password** and **Confirm** fields, enter a valid root account password.
6. Select the **Allow root SSH login with password** checkbox, and then click **Done** twice.
7. From the **Create User** page, use the following parameters:
   a. **Full name: user**
   b. **User name: user**
   c. **Password**: *<password>*
   d. **Confirm password**: *<password>*
   e. **Add administrative privileges**: Selected
   f. **Require a password to use this account**: Selected
8. Click **Done** twice.
9. From the **Connect to Red Hat** page, use the following parameters:
   a. **Authentication: Account**
   b. **User name**: *Red Hat developer account*
   c. **Password**: *Red Hat developer account password*
10. Click **Register**, and then click **Done**.

11. From the **Software Select** page, select **Server**, and then click **Done**.
12. From the **Installation Destination** page, select the boot disk, and then click **Done**.
13. From the **Installation Summary** page, click **Begin Installation**.
14. Once installation is complete, create a Secure Shell (SSH) connection to the server.
15. Run the following command to install updates:

```
dnf update -y
```

16. Run the following command to install additional prerequisites to support the benchmarking:

```
dnf install nmon dstat sysstat -y
```

WRK2

WRK2 is a benchmark that can provide a constant throughput load to a web server such as Apache HTTP Server™ or NGINX®. The purpose of running this benchmark with Intel Xeon 6 processors with E-cores, P-cores, and AP-cores is to show that, even though the processors with P-cores and AP-cores can functionally run web server workloads, processors with E-cores can maintain the workload as effectively while providing power efficiency that can be saved for workloads related to data and AI.

1. Run the following command to install Apache HTTP Server:

```
dnf install httpd -y
```

2. Run the following command to enable and start the Apache web server:

```
systemctl enable –now httpd && sudo systemctl start httpd
```

3. Run the following command to download **wrk2**:

```
git clone https://github.com/giltene/wrk2.git
```

4. To run the WRK2 benchmark, copy the file contents provided in **Appendix A** to a new shell script, taking care to update the variables as needed.

ResNet-50 Version 1.5

ResNet-50 version 1.5 is a neural network architecture that is designed for image classification. The purpose of running this benchmark against Intel Xeon 6 processors with E-cores, P-cores, and AP-cores is to show that, although the Intel Xeon 6 processors with E-cores struggled to maintain the workload, both systems with P-cores and those with AP-cores were able to do so. In this case, making use of Intel Xeon 6 processors with P-cores enables businesses to reserve their more-powerful Intel Xeon 6 processors with AP-cores for more demanding high-performance computing (HPC) workloads.

The following precision, batch size, and pretrained model were utilized:

Precision	Batch Size	Pretrained Model
INT8	1	bias_resnet50.pb
INT8	16	bias_resnet50.pb
FP32	1	resnet50_v1.pb
FP32	16	resnet50_v1.pb

1. Run the following command to confirm Python® 3 is available:

```
python --version
```

2. Run the following command to create a new Python virtual environment:

```
python -m venv resnet-env
```

3. Run the following command to activate the Python virtual environment:

```
source resnet-env/bin/activate
```

4. Run the following command to install the Intel® Extension for TensorFlow™:

```
pip install intel-tensorflow
```

5. Run the following command to clone the Intel® AI Reference Models repository:

```
git clone https://github.com/intel/ai-reference-models.git
```

6. Follow **Intel's instructions to prepare the ImageNet Dataset**.
7. Download the **pretrained models**.
8. To run the pretrained models, copy the file contents from **Appendix B** to a new shell script, taking care to update the variables as needed to run with the correct precision. For more details, reference the **detail's provided online**.

NAMD

NAMD allows for simulating biomolecular systems. NAMD provides benchmarks to simulate a workload on the system under test (SUT). We found that Intel Xeon 6 processors with AP-cores are best-designed to handle this type of HPC workload.

For our testing, we used the following benchmarks:

- ApoA1 (Apolipoprotein A1)
- STMV (satellite tobacco mosaic virus)
- F1ATPase (F1-ATPase)

We completed this testing with the following steps:

1. Run the following command to install prerequisites:

```
dnf groupinstall 'Development Tools' -y && dnf install ncurses-devel bison openssl-devel php ninja-build
environment-modules elfutils clang gtk3-devel alsa-utilsv-y
```

2. Run the following commands to download the Intel® oneAPI scripts:

```
wget https://registrationcenter-download.intel.com/akdlm/IRC_NAS/d640da34-77cc-4ab2-8019-ac5592f4ec19/
intel-oneapi-base-toolkit-2025.3.0.375_offline.sh

wget https://registrationcenter-download.intel.com/akdlm/IRC_NAS/66021d90-934d-41f4-bedf-b8c00bbe98bc/
intel-oneapi-hpc-toolkit-2025.3.0.381_offline.sh
```

3. Run the following command to install the Intel oneAPI Base Toolkit (Base Kit):

```
sh ./intel-oneapi-base-toolkit-2025.3.0.375_offline.sh -a --silent --cli --eula accept
```

4. Run the following command to install the Intel oneAPI HPC Toolkit:

```
sh ./intel-oneapi-hpc-toolkit-2025.3.0.381_offline.sh -a --silent --cli --eula accept
```

5. Download NAMD from **Linux-x86-64_multicore** and **Linux-x86_64_multicore-AVX512**. (**Note**: You will be required to register to download the workload.)
6. To run the NAMD benchmarks, copy the file contents from **Appendix C** to a new shell script, taking care to update the variables as needed.

Appendix

Appendix A—WRK2 Script

```
#!/bin/bash

# Define the optstring

optstring="rspt:o:"

# Define the variables to store the option values

# Parse the command-line arguments

while getopts "$optstring" opt; do

  case "$opt" in

    o) requested_output_path="$OPTARG" ;;
```

```
  esac
done

# Sanity check arguments
if [[ -z "$requested_output_path" ]]; then

  echo "Error: missing flags!  All below are required
-o directory to create and save output into
"
  exit 1
fi

ts=$(date +%s)
output_path=${requested_output_path}.${ts}

# Prepare files and output path
mkdir -p $output_path
sleep 2
# Run the tests
concurrency=(50)
threads=(10 50)
requests=(3k 5k 10k 20k 30k)
for c in ${concurrency[@]}; do
        for t in ${threads[@]}; do
                for r in ${requests[@]}; do
                        run_ts=$(date +%s)
taskset --cpu-list 1,3,5,7 wrk -c ${c} -t ${t} -R ${r} -d 150s --latency http://{}/ 2>&1 | tee $output_
path/${c}_${t}_${r}_$run_ts.txt
                done
        done
done
concurrency=(100 200 300 500 700 1000)
threads=(10 50 100)
requests=(3k 5k 10k 20k 30k)
for c in ${concurrency[@]}; do
    for t in ${threads[@]}; do
```

```
                    for r in ${requests[@]}; do
                            run_ts=$(date +%s)
                            taskset --cpu-list 1,3,5,7 wrk -c ${c} -t ${t} -R ${r} -d 150s --latency
http://{}/ 2>&1 | tee $output_path/${c}_${t}_${r}_$run_ts.txt
                    done
    done
done
```

Appendix B—ResNet-50 Script

```
#!/bin/bash
# Define the optstring
optstring="rspt:o:v:"
# Define the variables to store the option values
# Parse the command-line arguments
while getopts "$optstring" opt; do
  case "$opt" in
    o) requested_output_path="$OPTARG" ;;
    v) volume="$OPTARG" ;;
    esac
done

# Sanity check arguments
if [[ -z "$requested_output_path" || -z "$volume" ]]; then
  echo "Error: missing flags!  All below are required
        -o directory to create and save output into
        -v volume to be tested"
  exit 1
fi

# Extract the device name from the volume path
short_volume=$(basename $volume)

ts=$(date +%s)
output_path=${requested_output_path}.${ts}
## ResNet-50 v1.5 variables
export PRECISION=int8 or FP32
## FP32 Model
```

```
#export PRETRAINED_MODEL=<path>/resnet50/resnet50_v1.pb

## Int8 Model

#export PRETRAINED_MODEL=<path>/resnet50/bias_resnet50.pb

export BATCH_SIZE=1

export DATASET_DIR=<path>/resnet50/tf_records

#export RESNET_SCRIPT=./models_v2/tensorflow/resnet50v1_5/inference/cpu/inference_realtime_multi_instance.
sh

#export RESNET_SCRIPT=./models_v2/tensorflow/resnet50v1_5/inference/cpu/inference_throughput_multi_
instance.sh

# Prepare files and output path

mkdir -p $output_path

##cp $test_path/$testType* $output_path/

# start the monitors

nmon -F $output_path/nmon.out -s2 -c100000 -t &

# Initialize d_pid

d_pid=""

# Check if mapped volume or device and start dstat accordingly

if [[ $short_volume == md* ]]; then

        dstat -tam -C total --md -M total,$short_volume --output $output_path/dstat.out 2 &

        d_pid=$!

elif [[ $short_volume == sd* || $short_volume == nv* ]]; then

        dstat -tam -C total -dD total,$short_volume --output $output_path/dstat.out 2 &

        d_pid=$!

fi

iostat -k -t -o JSON -cdx $volume 2 > $output_path/iostat.out &

sleep 2

# Run the tests

for cpi in 4 8 12 16 20 24 30 32 36 40 ; do

        export CORES_PER_INSTANCE=$cpi

        for threads in 20 40 60 80 100 120 160 180 200 220 232 240 280 300 320 ; do

                sleep 15

                export OMP_NUM_THREADS=$threads

                        mkdir -p $output_path/$cpi/$threads

                        export OUTPUT_DIR=$output_path/$cpi/$threads

                        $RESNET_SCRIPT

        done
```

```
done
# Cleanup monitors
killall nmon
if [[ -n "$d_pid" ]]; then
    kill $d_pid
fi
killall iostat
echo "]}]}}" >> $output_path/iostat.out
```

Appendix C—NAMD Script

```
#!/bin/bash
# Define the optstring
optstring="rspt:o:v:"
# Define the variables to store the option values
# Parse the command-line arguments
while getopts "$optstring" opt; do
  case "$opt" in
    o) requested_output_path="$OPTARG" ;;
    v) volume="$OPTARG" ;;
    esac
done

# Sanity check arguments
if [[ -z "$requested_output_path" || -z "$volume" ]]; then
  echo "Error: missing flags!  All below are required
        -o directory to create and save output into
        -v volume to be tested"
  exit 1
fi

# Extract the device name from the volume path
short_volume=$(basename $volume)

ts=$(date +%s)
output_path=${requested_output_path}.${ts}
# Uncomment which NAMD script to run
```

```
#export NAMD_SCRIPT=<path>/f1atpase/f1atpase.namd

#export NAMD_SCRIPT=<path>/apoa1/apoa1.namd

#export NAMD_SCRIPT=<path>/stmv/stmv.namd

# Prepare files and output path

mkdir -p $output_path

# start the monitors

nmon -F $output_path/nmon.out -s2 -c100000 -t &

# Initialize d_pid

d_pid=""

# Check if mapped volume or device and start dstat accordingly

if [[ $short_volume == md* ]]; then

        dstat -tam -C total --md -M total,$short_volume --output $output_path/dstat.out 2 &

        d_pid=$!

elif [[ $short_volume == sd* || $short_volume == nv* ]]; then

        dstat -tam -C total -dD total,$short_volume --output $output_path/dstat.out 2 &

        d_pid=$!

fi

iostat -k -t -o JSON -cdx $volume 2 > $output_path/iostat.out &

sleep 2


turbostat -S -i 2 > $output_path/turbostat.out &

# Run the tests

for threads in 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128 136 144 152 160 168 176 184 192 200 208
216 224 232 236 ; do

        sleep 15

        export NUM_CPU_CORES=$threads

        echo "Threads: $threads"

        numactl -C $NUM_CPU_CORES ./namd3 +p$NUM_CPU_CORES +setcpuaffinity +maffinity +CmiSleepOnIdle
$NAMD_SCRIPT > $output_path/$threads.txt

done

# Cleanup monitors

killall nmon

if [[ -n "$d_pid" ]]; then

    kill $d_pid

fi

killall iostat
```

```
echo “]}]}}” >> $output_path/iostat.out

killall turbostat
```

Endnotes

[1] NAMD was developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. **www.ks.uiuc.edu/Research/namd/**.

0426/250111