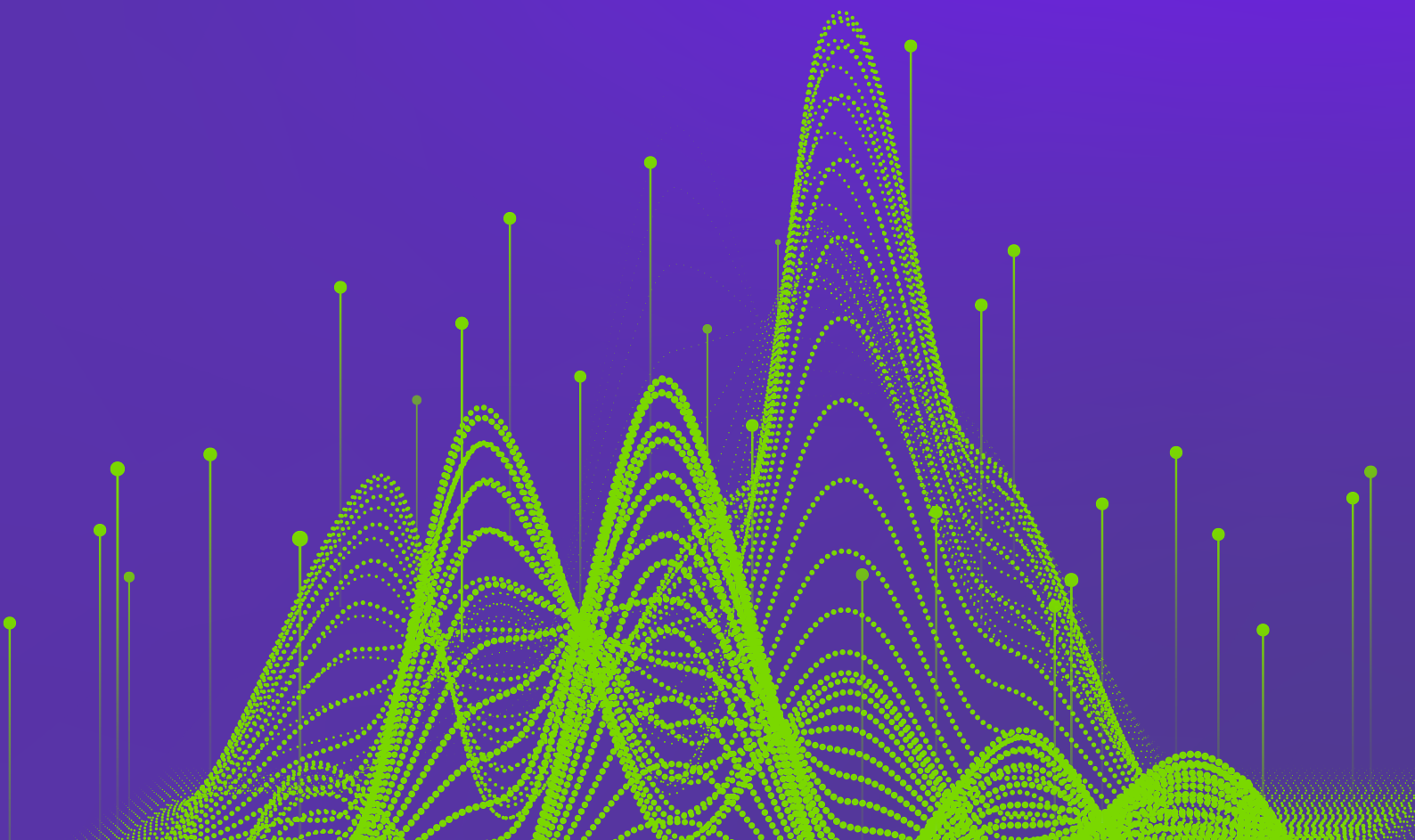
doitnow
HPC Services

WHITE PAPER

Next-Gen Velocity.

Benchmarking the Amazon EC2 Hpc8a Evolution

aws | AMD



Index.

01.

Introduction

Do IT Now deployed a HPC cluster in the cloud using AMD-based HPC-optimized instances on AWS, utilizing 10 nodes for each instance type, installed the necessary applications on the infrastructure and conducted benchmarks with reference models for each application. Do IT Now also compared results in terms of performance for each application and instance combination. During this analysis, Do IT Now also explored possible optimization patterns and parameters to tune the applications, the instances, and the overall HPC architecture in the cloud.

1.1 OBJECTIVE

The objective of this project is to provide the results of the benchmarks conducted in the cloud environment, showing the baseline performance results and any additional optimizations that the Do IT Now team applied to the infrastructure, the software and its parameters.

1.2 ABOUT CLOUD INSTANCES

The cloud instances provided are based on AMD processors, with the following characteristics:

- EC2 Hpc8a preview instances, featuring two 5th Gen AMD EPYC 9005 series 192 cores processor with up to 4.5 GHz all-core turbo frequency, 768 GiB RAM DDR5 and 300 Gbps networking (https://aws.amazon.com/it/ec2/instance-types/Hpc8a);

- EC2 Hpc7a instances, featuring two 4th Gen AMD EPYC 9004 series 192-core processors with up to 3.7 GHz all-core turbo frequency, DDR5 RAM and 300 Gbps networking. Hpc7a instances also offer 24-, 48-, 96-, 192-core 4th Gen AMD EPYC processors (https://aws.amazon.com/ec2/instance-types/Hpc7a).

Do IT Now uses two comparable instances to ensure a fair and accurate comparison of the results:

1.3 HPC8A.96XLARGE

Full specifications:

- **CPU:** AMD EPYC 9655 Processor @ 4.5 GHz
- **Cores:** 96 x 2 sockets, 1 thread per core, 192 total physical cores
- **RAM:** 768GiB
- **Network:** 2 x Elastic Fabric Adapter network interface, 300 Gbps bandwidth

1.4 HPC7A.96XLARGE

Full specifications:

- **CPU:** AMD EPYC 9R14 Processor @ 3.7 GHz
- **Cores:** 96 x 2 sockets, 1 thread per core, 192 total physical cores
- **RAM:** 768 GiB
- **Network:** 2 x Elastic Fabric Adapter network interface, 300 Gbps bandwidth

02.
About the software

The software used for the benchmarks is open source:.

OpenRadioss™ (v20251211) is a leading analysis solution to evaluate and optimize product performance for highly nonlinear problems under dynamic loadings. Used worldwide across all industry sectors, it improves the crashworthiness, safety, and manufacturability of complex designs.

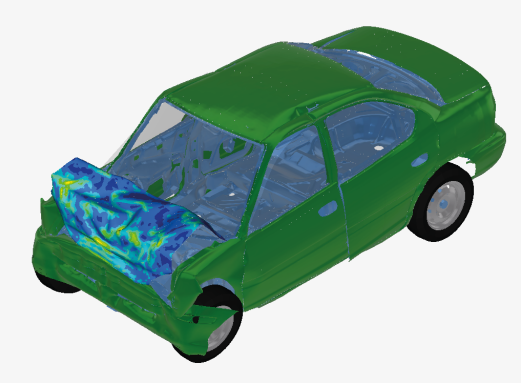
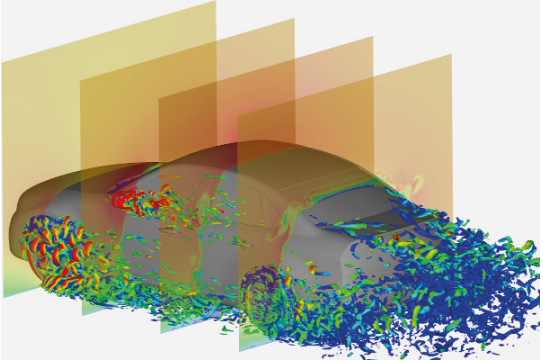


Figure 1. – OpenRadioss™ example output (source: OpenRadioss™.org)



OpenFOAM™ (v2512) is a leading general-purpose Computational Fluid Dynamics (CFD) solver that is capable of solving the most demanding industrial and scientific applications. Robust and scalable solver technology empowers users by providing unparalleled accuracy.

Figure 2 – OpenFOAM™ example output (source: openfoam.com)

2.1 ABOUT THE MODELS

All the models are well suited to benchmarking HPC performance in HPC clusters with a large number of CPUs.

2.1.1 OPENRADIOSS™

The model used for this software is the publicly available "Taurus 10 Million finite elements" model", recovered from the OpenRadioss™ public repository. This benchmark has a refined mesh with 10 million finite elements.

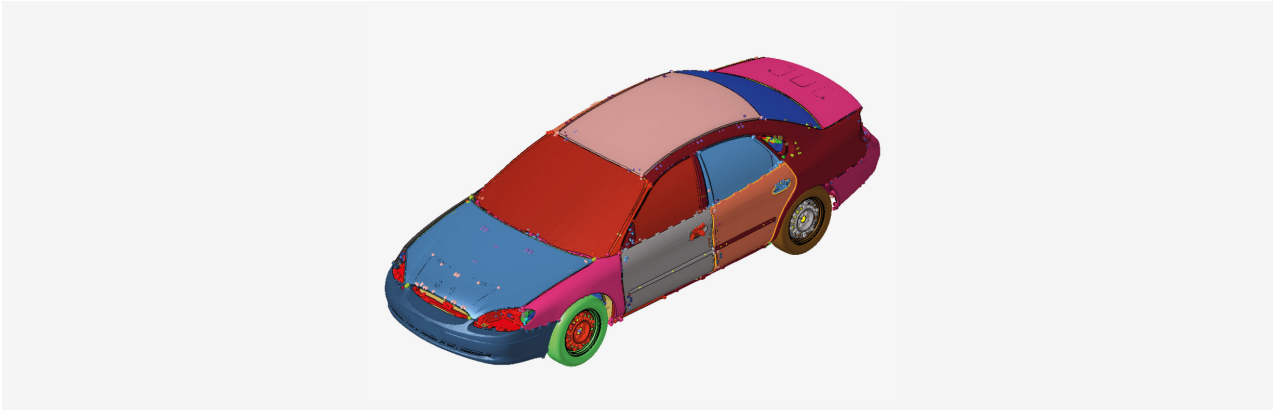


Figure 3 – Taurus 10M elements model (source: OpenRadioss™ project)

The model has three different settings, based on the simulation time:

- Full simulation, 120 milliseconds.
- Shorter simulation, 10 milliseconds: best suited for fast performance and scalability testing on a large number of nodes/CPUs.
- Very short simulation, 2 milliseconds: useful to test the HPC cluster functions.

2.1.2 OPENFOAM™

The model used for this software is the publicly available "Taurus 10 Million finite elements" model", recovered from the OpenRadioss™ public repository. This benchmark has a refined mesh with 10 million finite elements.

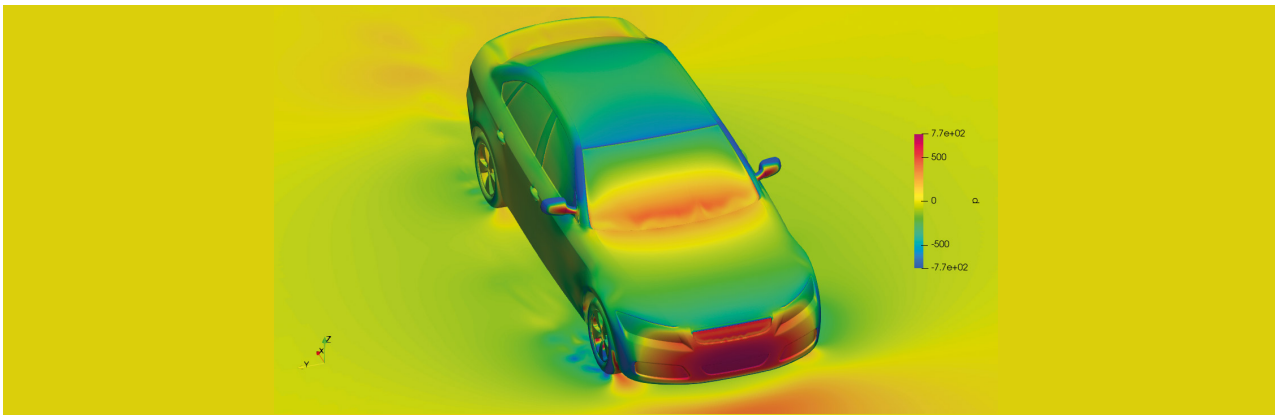


Figure 4 – Pressure at the surface result (source: develop.openfoam.com)

> **The model has three different settings, based on the mesh resolution, refined and adapted to the surface**
>
> - Coarse – 65M cells
> - Medium – 110M cells
> - Fine – 236M cells

For the benchmarks Do IT Now uses the Coarse and Fine test cases, to compare the result times with different configurations.

03.
Architecture design

The architecture is based on an AWS ParallelCluster standard deployment.

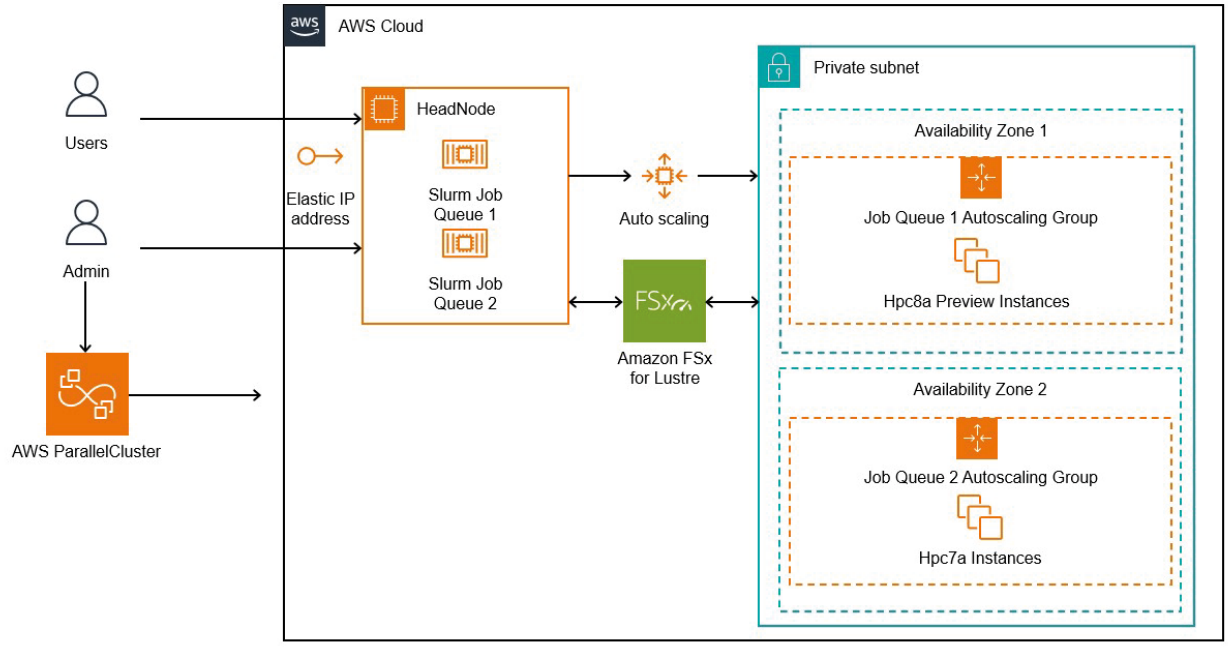


Figure 5 – AWS Architecture Diagram

A HeadNode is installed, providing centralized user authentication, job scheduling services and auto-scaling integration for two job queues. Each job queue is associated with a specific instance type in a separate Availability Zone.

An Amazon FSx for Lustre deployment provides a shared filesystem that will contain the applications, the user input and output data while also acting as a high-performance scratch filesystem for compute jobs.

All systems are using RedHat Enterprise Linux 8.10 (Ootpa).

On HPC instances, EFA networking (https://aws.amazon.com/hpc/efa/) is enabled.

04.
Installation

4.1 AWS PARALLELCLUSTER

The deployment of the architecture was performed using AWS ParallelCluster and choosing the appropriate dimensions, base AMI for instances and instance specifications.

4.2 SCHEDULER

The SLURM scheduler was also deployed using AWS ParallelCluster, which helped us configure the proper partitions (a.k.a. job queues) with the association of a specific instance type to each partition. All auto-scaling operations are directly controlled by SLURM, which takes care of deploying compute instances based on each job's resource requests.

4.3 STORAGE

FSx for Lustre storage was also deployed via AWS ParallelCluster, to ensure that the HeadNode and each compute instance is able to access the same data with the highest possible performance. Do IT Now considered the possibility of using EBS with NFS for the users home directories but deemed it non-necessary as this is not a production-ready environment, and wanted to contain overall costs by cutting what was considered to be non-essential features.

4.4 USER DATA REPOSITORY

User data was also stored in the FSx for Lustre filesystem, organized in "models" and "runs" per each solver.

05.
Benchmark execution

The different jobs directories are carefully prepared with all the required data to run the simulations in different configurations. Simulation results are produced within each job's directory.

5.1 BASELINING

As the provided instance is currently in a preview state, the hardware configuration cannot be altered. Therefore, scalability is evaluated by varying the number of nodes and cores allocated for each job.

5.2 MPI TUNING

The MPI library used is Intel MPI version 2021 Update 17, used in the AWS ParallelCluster system image. The MPI configurations are the same for all software used in these benchmarks.

5.3 MPI SETTINGS

The following environment variables are set at each job's run time:

```
1.    export I_MPI_OFI_LIBRARY_INTERNAL=0
2.    export I_MPI_FABRICS=shm:ofi
3.    export I_MPI_OFI_PROVIDER=efa
4.    export FI_EFA_SHM_AV_SIZE=256
5.    export I_MPI_PIN_DOMAIN=numa
6.    export I_MPI_DEBUG=5
7.    export I_MPI_MULTIRAIL=1
```

This is the baseline configuration used for tests:

- Make use of the AWS provided libfabric (I_MPI_OFI_LIBRARY_INTERNAL)
- Select the correct fabrics for optimal performance within the nodes and for node-to-node communications (I_MPI_FABRICS)
- Select the required fabric provider for AWS Elastic Fabric Adapter (I_MPI_OFI_PROVIDER)

- Defines the maximum number of entries in the Shared Memory (SHM) Address Vector (AV) for EFA (FI_EFA_SHM_AV_SIZE)
- Defines the logical scope for pinning MPI processes to CPU resources (I_MPI_PIN_DOMAIN)
- Enable debug mode with enough details to analyze the fabric configuration in detail (I_MPI_DEBUG)
- Activates multi-rail support to distribute traffic across available network adapters (I_MPI_MULTIRAIL)

5.4 SCALABILITY

To test scalability, the same simulations are run on different node configurations: starting from the baseline, the number of cores is divided by 8, 4 and 2 times the original amount. This brought our maximum core number per node to 192.

5.5 PERFORMANCE RESULTS

The following graphs represent the results obtained.
On the Y axis - recorded the time required to complete a job;
on the X axis - each data point represents the number of nodes and cores for that job (eg: 1 node – 24 cores / 4 nodes – 768 cores).

5.6 HPC8A OPENRADIOSS™

5.6.1 Hpc8a OpenRadioss™ - T10M model

The execution time of each OpenRadioss™ job is taken from the output files, using the reported time from the "engine" solver itself. This is because the "starter" process runs at the beginning of the job on a single node, preparing the input files for each single "engine" process that runs the simulation. The time for the "starter" process to prepare the input files increases linearly with the number of cores the simulation that runs. In the end, Do IT Now chose not to include "starter" timings in the benchmark elapsed time to provide a more precise, apples-to-apples comparison between different runs.

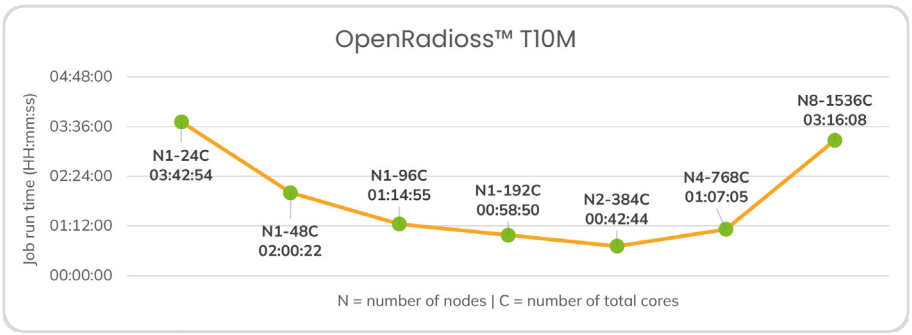


Figure 6 - Performance chart for OpenRadioss™ on Hpc8a instances

The OpenRadioss™ T10M benchmark demonstrates strong scaling up to 384 cores (2 nodes), which represents the optimal configuration with minimum execution time.

Beyond this point, scalability saturates; scaling to 768 cores (4 nodes) introduces a slight performance penalty, while extending to 1536 cores (8 nodes) results in severe degradation due to communication overhead dominating computational gains.

Within a single node (N1), the solver exhibits near-linear scaling when increasing the core count from 24 to 96. However, as the configuration approaches the full node capacity (192 cores), a reduction in parallel efficiency is observed. While the job time continues to decrease, the transition from 96 to 192 cores yields a less pronounced performance gain compared to lower core counts, likely due to increased memory bandwidth contention and resource sharing within the single compute instance.

5.7 Hpc8a OpenFOAM™

The execution time taken in consideration for OpenFOAM™ is the overall elapsed time between the start and the end of the whole computation. Like OpenRadioss™, OpenFOAM™ has a preparation phase to subdivide the simulation data for each solver processes.

5.7.1 Hpc8a OpenFOAM™ – Coarse model 65M

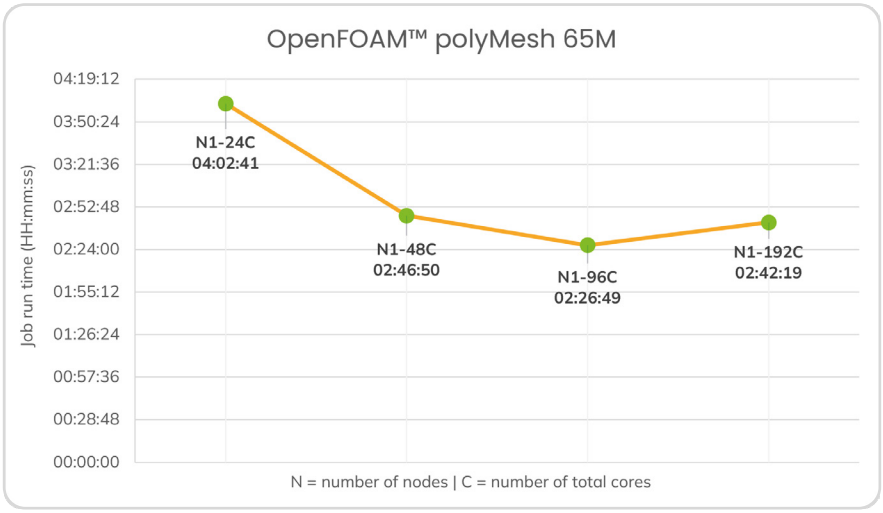


Figure 7 – Performance chart for OpenFOAM™ on Hpc8a instances / single node / Coarse model

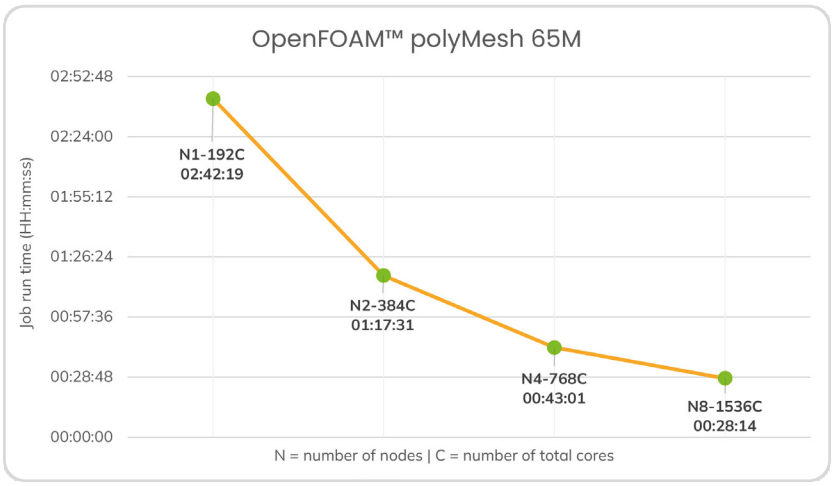


Figure 8 – Performance chart for OpenFOAM™ on Hpc8a instances / multi node scalability / Coarse model

The intra-node scaling analysis for OpenFOAM™ benchmark (65M cells) reveals a performance regression when fully populating the compute node. While runtime decreases up to 96 cores (02:26:49), increasing the core count to 192 results in a slower runtime (02:42:19). This behavior is characteristic of memory bandwidth saturation, where the memory channels cannot supply data fast enough to feed all physical cores simultaneously, causing CPU starvation.

However, scalability is immediately restored when transitioning to a multi-node configuration (N2-384C).

By adding nodes, the system gains aggregate memory bandwidth, relieving the local bottleneck and allowing the solver to scale efficiently down to 28 minutes on 8 nodes.

5.7.2 Hpc8a OpenFOAM™ – Fine model 236M

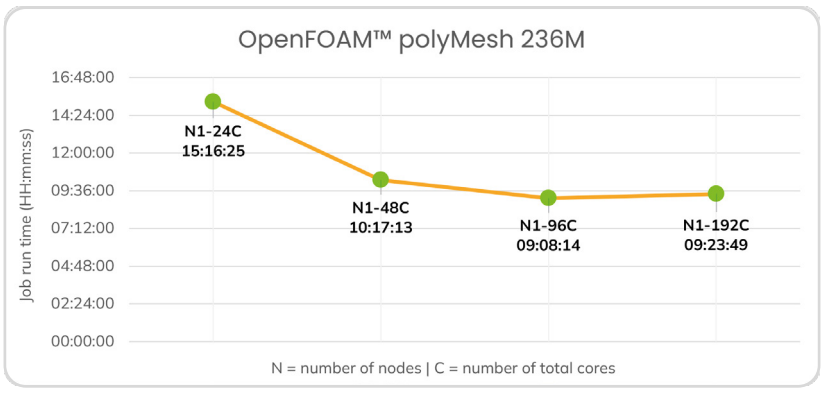


Figure 9 – Performance chart for OpenFOAM™ high-resolution model on Hpc8a instances / single node / Fine model

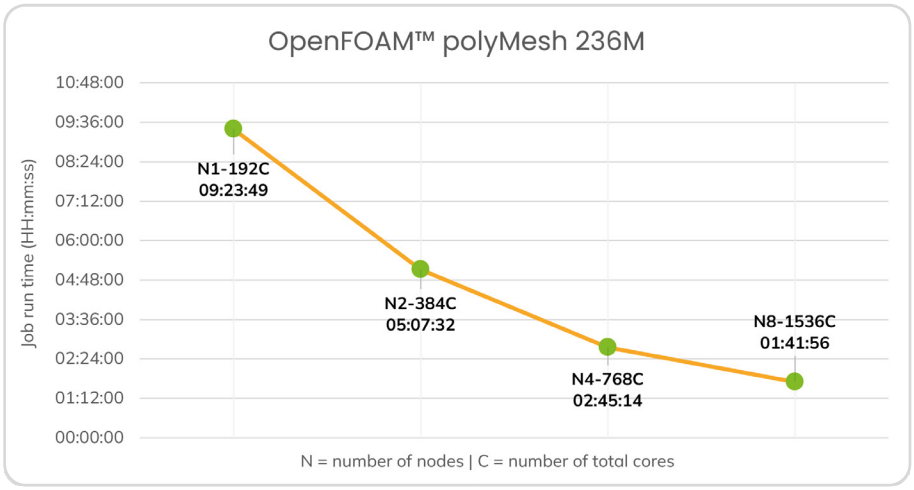


Figure 10 – Performance chart for OpenFOAM™ high-resolution model on Hpc8a instances / multi node scalability / Fine model

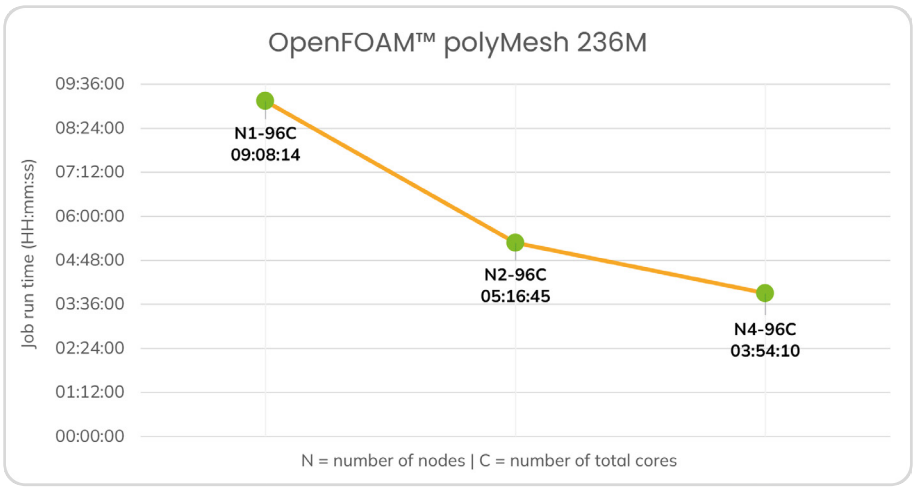


Figure 11 – Performance chart for OpenFOAM™ high-resolution model on Hpc8a instances / 96 core scalability / Fine model

We can observe similar behavior for the Fine 236M mesh configuration.
The transition from 96 to 192 cores within a single node again leads to increased runtime due to memory contention. Notably, the impact of bandwidth availability is highlighted by the N4-96C result (03:54:10), which significantly outperforms the N2-96C configuration (05:16:45). This confirms that distributing fewer cores across more nodes—thereby accessing more memory channels—is more efficient than packing cores into fewer nodes for this memory-bound solver.

06.
Performance comparison: Hpc8a vs Hpc7a

6.1 HPC8A VS HPC7A: OPENRADIOSS™ - T10M MODEL

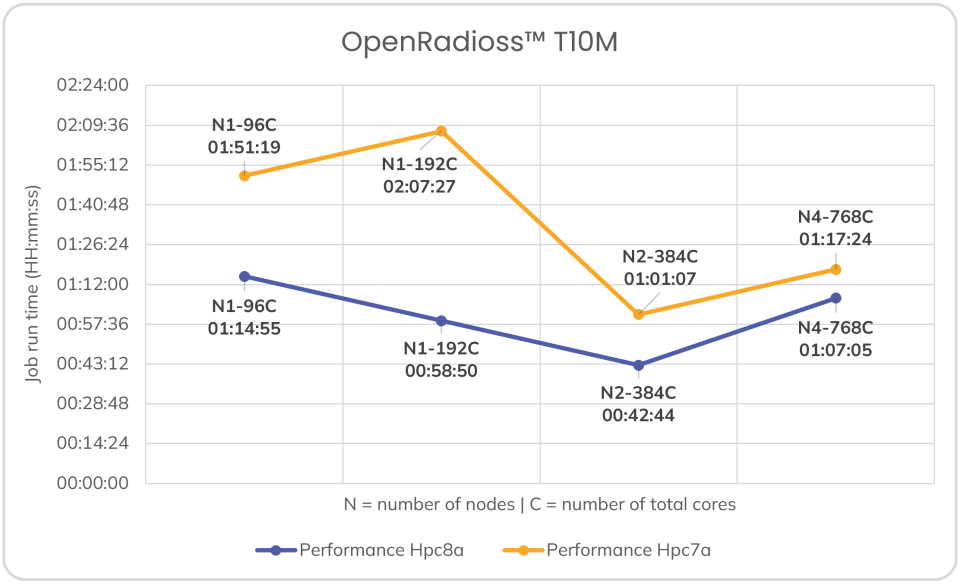


Figure 12 – Performance chart for OpenRadioss™ high-resolution model: Hpc8a vs Hpc7a instances

Number of node(s) - Number of cores	Hpc7a walltime (hh:mm:ss)	Hpc8a walltime (hh:mm:ss)	Run time reduction
N1-96C	01:51:19	01:14:55	- 32,7 %
N1-192C	02:07:27	00:58:50	- 53,8 %
N2-384C	01:01:07	00:42:44	- 30,1 %
N4-768C	01:17:24	01:07:05	- 13,3 %

6.2 HPC8A VS HPC7A: OPENFOAM™ - COARSE MODEL

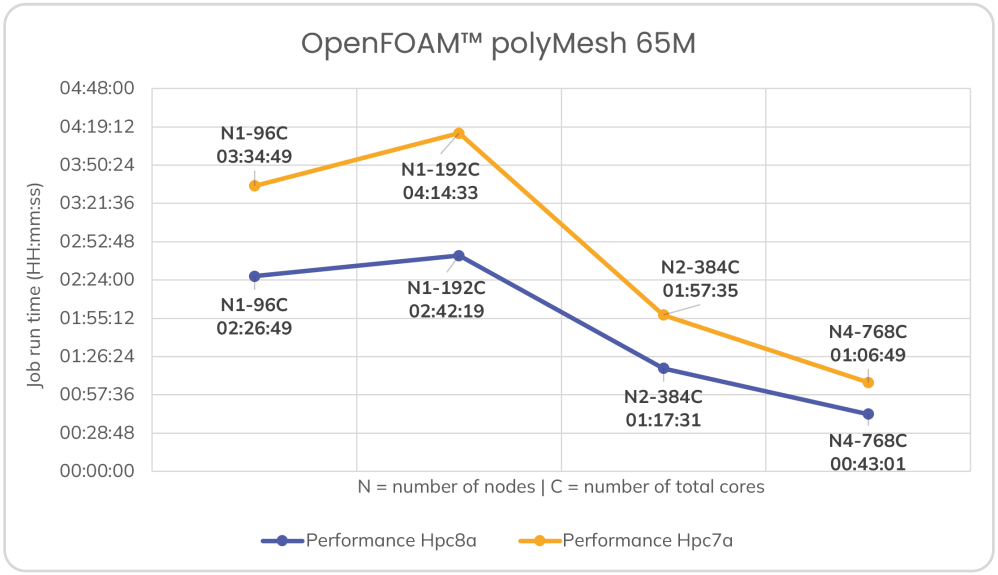


Figure 13 – Performance chart for OpenFOAM™ high-resolution model: Hpc8a vs Hpc7a instances

Number of node(s) - Number of cores	Hpc7a walltime (hh:mm:ss)	Hpc8a walltime (hh:mm:ss)	Run time reduction
N1-96C	03:34:49	02:26:49	- 31,7 %
N1-192C	4:14:33	02:42:19	- 36,2 %
N2-384C	01:57:35	01:17:31	- 34,1 %
N4-768C	01:06:49	00:43:01	- 35,6 %

6.3 HPC8A VS HPC7A: OPENFOAM™ - FINE MODEL

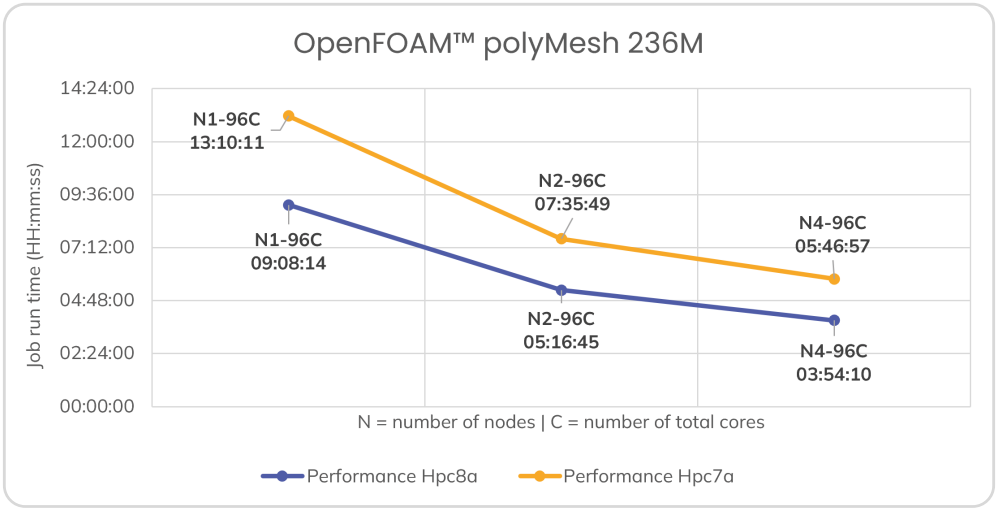


Figure 14 – Performance chart for OpenFOAM™ high-resolution model: Hpc8a vs Hpc7a instances

Number of node(s) - Number of cores	Hpc7a walltime (hh:mm:ss)	Hpc8a walltime (hh:mm:ss)	Run time reduction
N1-96C	13:10:11	09:08:14	- 30,6 %
N2-96C	07:35:49	05:16:45	- 30,5 %
N4-96C	05:46:57	03:54:10	- 32,5 %

07.
Conclusions

The benchmark campaign highlights that optimal HPC configuration is strictly solver-dependent, driven by the distinct computational characteristics of CFD and explicit FEA workloads.

Regarding node architecture, the high core density (192 cores per node) yielded different outcomes depending on the computational intensity of the solver. For OpenRadioss™, intra-node execution showed linear scaling, benefiting from the available core count. In contrast, OpenFOAM™ simulations exhibited performance regression at full node occupancy due to memory channel saturation. This data indicates that for memory-bandwidth-bound applications on this specific architecture, the ratio of memory bandwidth per core is a limiting factor, requiring core under-subscription to optimize execution time.

In terms of distributed scalability, the interconnect subsystem supported near-linear scaling for OpenFOAM™ up to 1536 cores. This suggests that network latency and bandwidth capabilities were sufficient to handle the MPI communication overhead for this solver. Conversely, the scaling limitations observed with OpenRadioss™ beyond 384 cores were consistent with fine-grained domain decomposition constraints rather than infrastructure bottlenecks.

In summary, the instance supports large-scale MPI workloads, though performance is dependent on configuration. The results demonstrate that while the network infrastructure facilitates distributed scaling, the high-density node topology requires specific process pinning and memory management strategies—such as relaxed domain binding and multirail transport—to mitigate local bandwidth contention.

7.1 HPC8A VS HPC7A COMPARISON

Looking at OpenFOAM™ results, both the Hpc7a and Hpc8a configurations demonstrate consistent scalability, with job run times decreasing significantly as the cluster size expands from a single node (96 cores) to four nodes (768 cores).

However, the Hpc8a instances exhibit a marked performance advantage over the previous generation. Across all tested scenarios, the Hpc8a technology consistently delivered faster execution times. Specifically, for the 65M mesh workload, the Hpc8a instances achieved a time reduction ranging from

approximately 31.7% to 36.2% compared to the Hpc7a. Similarly, for the heavier 236M mesh workload, the performance gain remained robust, with run time reductions between 30.5% and 32.5%. These results indicate that the Hpc8a architecture offers a substantial efficiency improvement, roughly cutting computation time by one-third compared to the Hpc7a baseline.

The OpenRadioss™ T10M benchmark results reveal a distinct scalability profile. While the Hpc8a architecture maintains a consistent performance lead over the Hpc7a across all configurations, the scalability efficiency diminishes beyond the two-node configuration.

The graphical trend suggests a potential scalability bottleneck or increased communication overhead for this specific workload at higher core counts. Despite this, the Hpc8a continues to outperform the previous generation, although the margin narrows to a 13.3% time reduction in this configuration.

7.2 CREDITS AND SPECIAL THANKS

Many thanks go to AWS for sponsoring this study and to everyone at Do IT Now involved in it.

Disclaimers:

Sponsored by Amazon Web Services and AMD

08.
Appendix

8.1 EXAMPLE SUBMISSION COMMAND FOR SLURM

```
sbatch -N<nodeNum> -n<coreNum> --exclusive --mem=0 <jobscript>
```

8.2 JOB SCRIPTS

8.2.1 Hpc8a OpenFOAM™ – Fine model 236M

```
#!/bin/bash


#SBATCH -J OR_1N-192C_LibFabDef
#SBATCH -N 1
#SBATCH -n 192
#SBATCH -p compute
#SBATCH --mem=0


NON="1"
CON="192"
JON="OR_${NON}N-${CON}C_LibFabDef"


rsync -av /fsx/apps/OpenRadioss/testcase/T10M_clean/
/fsx/apps/OpenRadioss/testcase/T10M_${JON}/
cd /fsx/apps/OpenRadioss/testcase/T10M_${JON}/


rm -v *tmp  *rst TAURUS_A05_FFB50A001 TAURUS_A05_FFB50T01
TAURUS_A05_FFB50_0001.out TAURUS_A05_FFB50_0000.out


RADIOSS_INP="TAURUS_A05_FFB50_0000.rad"
export
RAD_CFG_PATH=/fsx/apps/OpenRadioss/20251211/intelmpi/2021.16/hm_cfg_files/
export
```

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/fsx/apps/OpenRadioss/20251211/intelmpi/2021.16
/extlib/hm_reader/linux64
NDOMAINS=0


ulimit -s unlimited


NCPUS=${SLURM_NTASKS}


if [ $NDOMAINS -lt 1 ]; then
        NDOMAINS=${NCPUS}
        NTHREADS=1
else
        [ $((NCPUS % NDOMAINS)) -ne 0 ] && exit 1
        NTHREADS=$((NCPUS / NDOMAINS))
fi

NHOSTS=${SLURM_NNODES}

[ $((NDOMAINS % NHOSTS)) -ne 0 ] && exit 1
NDOMAINS_PER_NODE=$((NDOMAINS / NHOSTS))

NODEFILE=nodefile.txt

scontrol show hostnames "$SLURM_JOB_NODELIST" > ./hosts_file
for host in $(cat hosts_file); do
    printf $host'\n%.0s' {1..${SLURM_NTASKS}}
done >  ./${NODEFILE}

export FI_EFA_FORK_SAFE=1
#module load libfabric-aws/2.3.1amzn1.0

source /fsx/apps/intel/oneapi/setvars.sh

export I_MPI_OFI_LIBRARY_INTERNAL=0
export I_MPI_FABRICS=shm:ofi
export I_MPI_OFI_PROVIDER=efa
export I_MPI_DEBUG=5
export FI_EFA_SHM_AV_SIZE=256


if [[ "$NON" -eq 1 ]]; then
  export I_MPI_PIN_DOMAIN=numa
else
  export I_MPI_MULTIRAIL=1
fi


/fsx/apps/OpenRadioss/20251211/intelmpi/2021.16/exec/starter_linux64_ifx -i
${RADIOSS_INP} -nspmd ${NDOMAINS} -nt ${NDOMAINS} | tee
starter.${SLURM_JOBID}.out
```

```
if [[ "$NON" -eq 1 ]]; then
  mpiexec -n ${NDOMAINS} -genv I_MPI_PIN_DOMAIN=numa -genv I_MPI_FABRICS shm:ofi
-genv I_MPI_DEBUG=5 -genv I_MPI_OFI_LIBRARY_INTERNAL=0 -genv
I_MPI_OFI_PROVIDER=efa
/fsx/apps/OpenRadioss/20251211/intelmpi/2021.16/exec/engine_linux64_ifx_impi -i
${RADIOSS_INP//0000/0001} -nt ${NTHREADS}
else
  mpiexec -n ${NDOMAINS} -genv I_MPI_MULTIRAIL=1 -genv I_MPI_FABRICS shm:ofi
-genv I_MPI_DEBUG=5 -genv I_MPI_OFI_LIBRARY_INTERNAL=0 -genv
I_MPI_OFI_PROVIDER=efa
/fsx/apps/OpenRadioss/20251211/intelmpi/2021.16/exec/engine_linux64_ifx_impi -i
${RADIOSS_INP//0000/0001} -nt ${NTHREADS}
fi
```

8.2.2 OpenFOAM™

```
#!/bin/bash
### Script for running the open-closed coolings DrivAer with a static mesh
### Original authors: Charles Mockett, Hendrik Hetmann and Felix Kramer
(Upstream CFD GmbH), 2022-2023
### Modified setup for OpenFOAM HPC Challenge (OHC-1): Mark Wasserman (Huawei)
and Sergey Lesnik (Wikki GmbH), 2025


# Based on the geometry and setup of the AutoCFD4 Workshop:
# Hupertz, B., Chalupa, K., Krueger, L., Howard, K., Glueck, H.-D., Lewington,
N., . . . Shin, Y.-s. (2021). On the Aerodynamics of the Notchback Open Cooling
DrivAer: A Detailed Investigation of Wind Tunnel Data for Improved Correlation
and Reference. SAE Int. J. Adv. & Curr. Prac. in Mobility, 3(4), 1726-1747.
doi:https://doi.org/10.4271/2021-01-0958


### example for slurm submission
# adapt to your needs or ignore if not submitting to a queuing system
#SBATCH -J "OF_NUMA_1N-192C"
#SBATCH -N 1
#SBATCH -n 192

#SBATCH -p compute
#SBATCH --mem=0
##SBATCH --output=%x-%j.out
##SBATCH --error=%x-%j.err

source /fsx/apps/intel/oneapi/setvars.sh
source /fsx/apps/openfoam/v2512/intelmpi/OpenFOAM-v2512/etc/bashrc

### Solver
solverRANS="simpleFoam"
```

```
### Used directories
logDir="logFiles"


### Misc
fvSolution=fvSolution.fixedIter # fvSolution.fixedTol
controlDict=controlDict.noWrite # controlDict.withWrite
fileHandler=" "                 # "-fileHandler hostCollated "


# function to read parameters from caseDefinition...
function getInputParam ()
{
    varVal=`grep -w ${1} system/include/caseDefinition | cut -d ";" -f 1 | grep
-v '//'`
    varVal=`echo ${varVal:${#1}}`
    echo $varVal
}


# get input parameters
nProcs=`getInputParam nCores`
TIMESTAMP=$(date "+%Y-%m-%d_%H-%M-%S")


# copy controlDict/fvSolution files


cp system/${controlDict} system/controlDict
cp system/${fvSolution} system/fvSolution


#-----
export I_MPI_OFI_LIBRARY_INTERNAL=0
export I_MPI_FABRICS=shm:ofi
export I_MPI_OFI_PROVIDER=efa
export FI_EFA_SHM_AV_SIZE=256
export I_MPI_PIN_DOMAIN=numa
export I_MPI_DEBUG=5
#-----


# extra mpi flags

OMPI_FLAGS="-genv I_MPI_OFI_LIBRARY_INTERNAL=0 -genv I_MPI_FABRICS shm:ofi -genv
I_MPI_OFI_PROVIDER=efa -genv FI_EFA_SHM_AV_SIZE=256 -genv I_MPI_PIN_DOMAIN=numa
-genv I_MPI_DEBUG=5"
parEx="mpirun -np ${nProcs} ${OMPI_FLAGS}"
. ${WM_PROJECT_DIR:?}/bin/tools/RunFunctions

# ---- initialise & run simulation

mkdir $logDir
```

```
echo; echo "decomposePar"
time decomposePar -constant $fileHandler > $logDir/10_decomposePar.log 2>&1 ||
exit 1


echo; echo "restore 0 Directory"
restore0Dir -processor


echo; echo "renumberMesh"
time $parEx renumberMesh -constant -overwrite -parallel $fileHandler >
$logDir/20_renumberMesh.log 2>&1 || exit 1


echo; echo "potentialFoam"
time $parEx potentialFoam -initialiseUBCs -parallel $fileHandler >
$logDir/30_potentialFoam.log 2>&1 || exit 1


echo; echo "applyBoundaryLayer"
time $parEx applyBoundaryLayer -ybl "0.0450244" -parallel $fileHandler >
$logDir/40_applyBoundaryLayer.log 2>&1 || exit 1
echo; echo $solverRANS
time $parEx $solverRANS -parallel $fileHandler >
$logDir/50_$solverRANS.$TIMESTAMP.log 2>&1 || exit 1
echo "... done!"
```

8.3 PARALLELCLUSTER CONFIGURATION FILES

8.3.1 Hpc8a in eu-north-1

```
- Name: compute
  CapacityType: ONDEMAND
  ComputeResources:
      - Name: compute-resources
        InstanceType: Hpc8a.96xlarge
        MinCount: 0
        MaxCount: 10
        DisableSimultaneousMultithreading: true
        Efa:
           Enabled: true
   Networking:
        AssignPublicIp: false
        SubnetIds:
          - subnet-0xxxxxxxxx
        PlacementGroup:
          Enabled: true
   CustomActions:
        OnNodeConfigured:
          Script: s3://path/init.sh
```

8.3.2 Hpc7a in eu-north-1

```
- Name: compute-Hpc7a
  CapacityType: ONDEMAND
  ComputeResources:
      - Name: compute-Hpc7a
        InstanceType: Hpc7a.96xlarge
        MinCount: 0
        MaxCount: 10
        DisableSimultaneousMultithreading: true
        Efa:
          Enabled: true
   Networking:
        AssignPublicIp: false
        SubnetIds:
          - subnet-0xxxxxxxxxx
   PlacementGroup:
        Enabled: true
   CustomActions:
        OnNodeConfigured:
           Script: s3://path/init.sh
```

doitnow
HPC Services

DRIVING HPC INNOVATION FORWARD

www.doitnowgroup.com

Discuss your HPC needs today

Contact us at info@doitnowgroup.com

aws | AMD