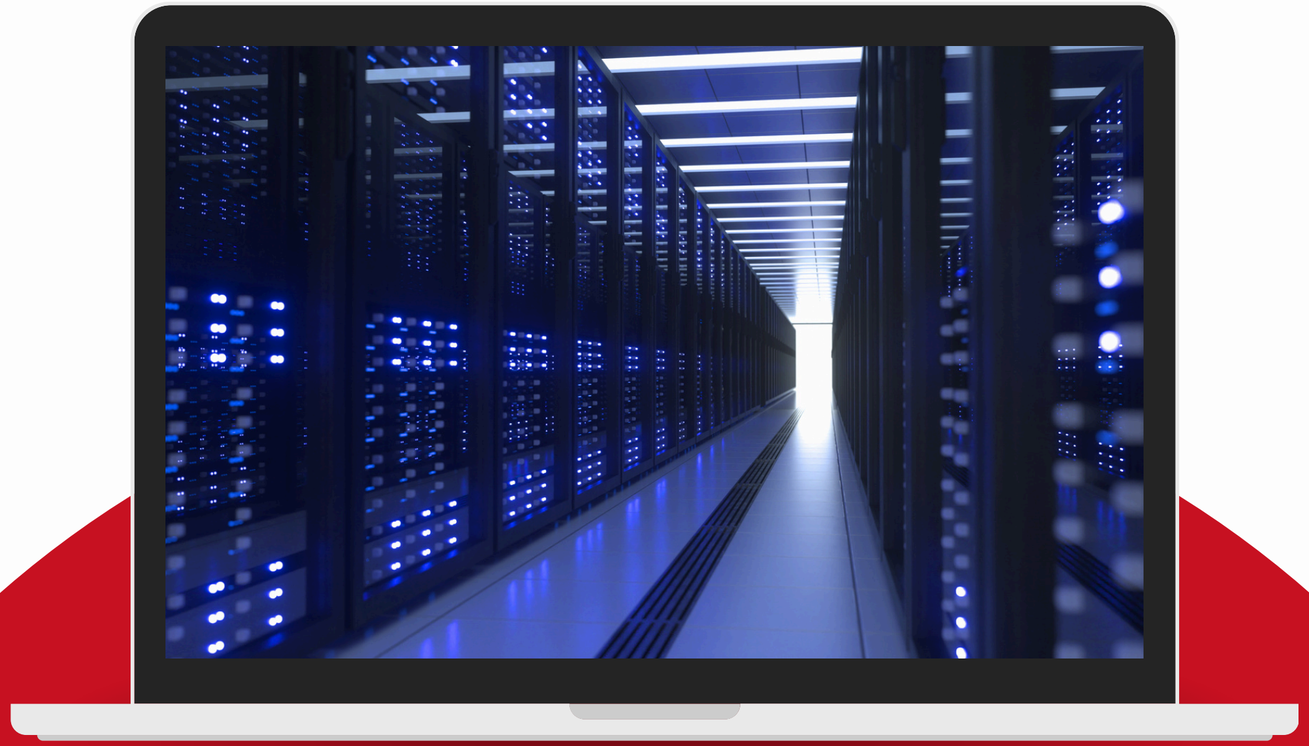
2025 APAC HPC-AI

Universiti Putra Malaysia
UPM Team 2



Our Team

Anis Humaira

Muhammad Aqil

Muhammad Akmal

Nurul Farizatul Aina

Universiti Putra Malaysia

Faculty of Computer Science and Information Technology (FCSIT)

In the 1980s, UPM started offering computer science courses to meet growing demand. By 1992, a dedicated department was established, offering various courses, including postgraduate programs.

In 1998, the Faculty of Computer Science and Information Technology was created to meet the increasing demand for computer education and keep up with technological advancements. Today, the faculty has four departments: Multimedia, Computer Science, Software Engineering & Information System, and Communication Technology & Networks.

Our Supervisors

MRS. Y.M. RAJA AZLINA RAJA MAHMOOD

ASSOC. PROF. DR. NOR ASILAH WATI ABDUL HAMID

Definitions

NWChem

NWChem is a theoritical computational chemistry software package which includes quantum chemical and molecular dynamics functionality. It was designed to run on high-performance parallel supercomputers as well as conventional workstation clusters.

DeepSeek

DeepSeek is known for creating the DeepSeek-R1 model which is considered competitive with other leading AI models from companies like OpenAI. We use SGLang framework as the engine to execute the DeepSeek-R1 model.

TASK 1
NWCHEM USING GADI

Our objective

Challenge:

We were tasked with running the NWChem input file with large, iterative matrix multiplications, and heavy operations within 5 minutes time constraint.

Objective:

To improve performance by accomplishing lower processing time used by CPUs and elapsed time.

Rules

- The results will be executed on **up to 4 standard CPU servers**.
  - Use **normalsr queue** in Gadi.

- Results with execution time beyond **300 seconds** are considered **INVALID.**

```
#!/bin/bash
#PBS -P ph60
#PBS -q normalsr
#PBS -l walltime=00:05:00
#PBS -j oe
#PBS -N w12opt
```

Performance Metrics

01

CPU time : It represents the total amount of computational work done by all processing cores.

02

Wall time : It measures the actual elapsed time from the moment the program starts to when it finishes.

Our optimization

1. **Implementation of fast storage**
2. **Application of OpenMP and MPI (Hybrid Parallelization)**
3. **Changes in input file**

Comparison Method

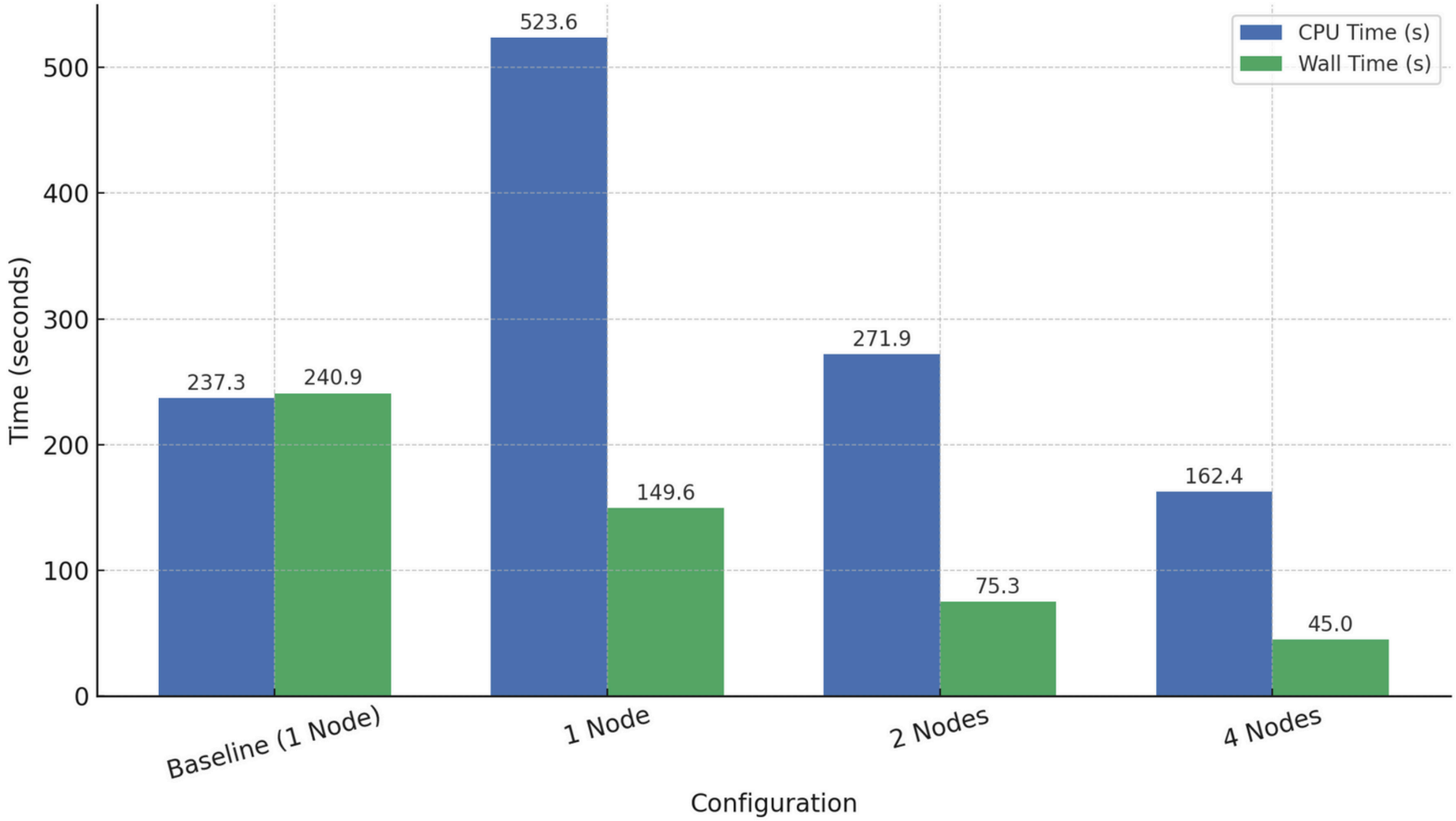
- **Compared across 3 different nodes (1, 2 and 4)**
- **Compared across 3 different memory stack directives**
  - memory stack **4000** mb heap **50** mb global **4000** mb
  - memory stack **8000** mb heap **100** mb global **8000** mb
  - memory stack **16000** mb heap **200** mb global **16000** mb

NWChem Results

Script	Configuration	No. of nodes	Number of CPUs	mpirun -np	Average CPU Time (s)	Average Wall Time (s)
baseline.pbs	Baseline	1	104 x 1 = 104	104	238.5	241.8
script8a.pbs script8b.pbs script8c.pbs	Optimized	1	104 x 1 = 104	26	523.6	149.6
script11a.pbs script11b.pbs script11c.pbs		2	104 x 2 = 208	52	271.9	75.3
script5a.pbs script5b.pbs script5c.pbs		4	104 x 4 = 416	104	162.4	45.0

Comparison Graph

Performance Scaling Across Different Nodes



Graph Analysis

Baseline vs 4 Nodes

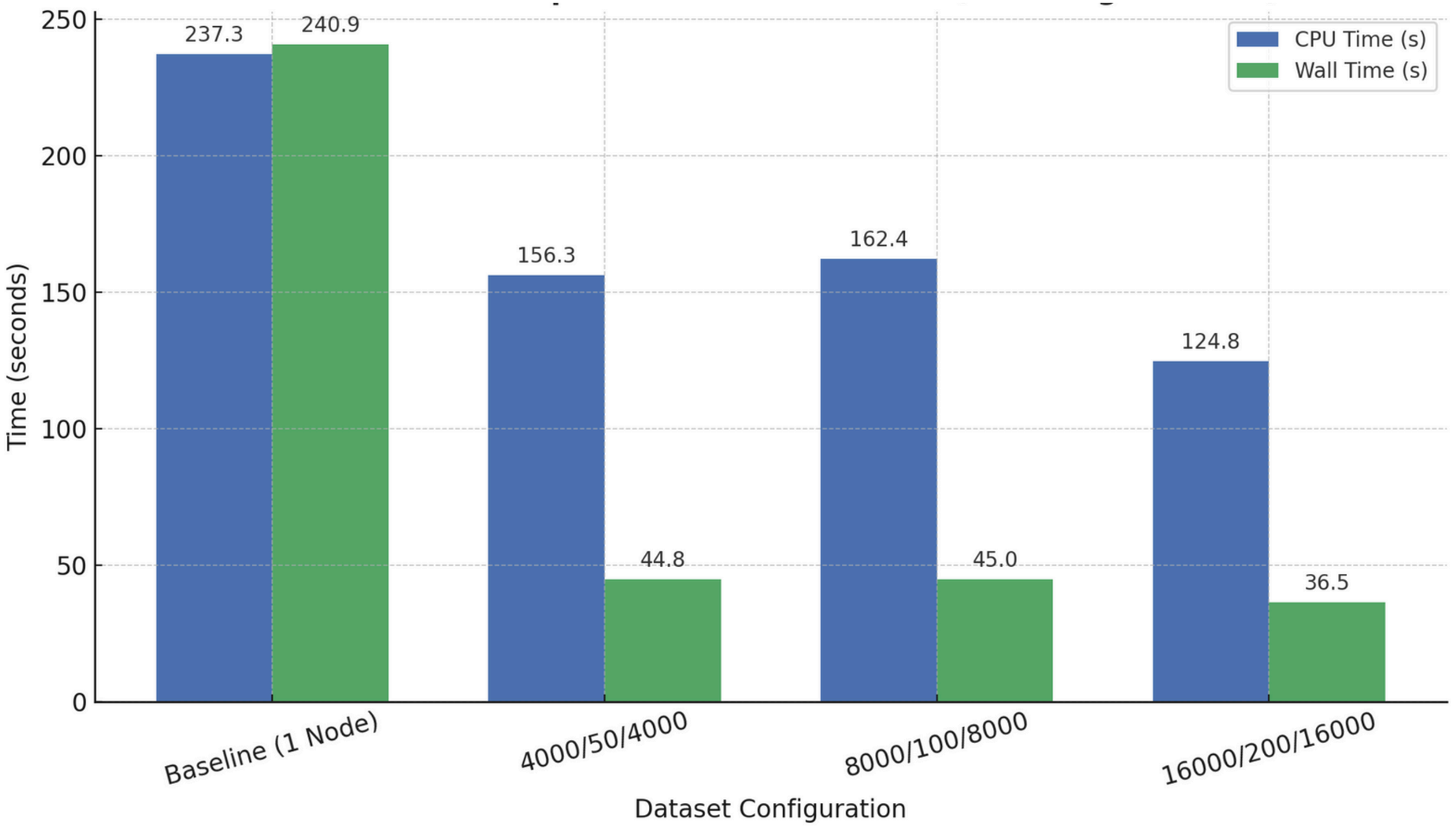
Metric	Baseline (1 Node)	4 Nodes	Speedup (×)	Improvement (%)
CPU Time (s)	237.3	162.4	1.46× faster	31.6% improvement
Wall Time (s)	240.9	45.0	5.36× faster	81.3% improvement

NWChem Results

Script	Configuration	No. of nodes	Memory Directives (stack/ heap/ global)	Average CPU Time (s)	Average Wall Time (s)
baseline.pbs	Baseline	1	8000 / 100 / 8000	238.5	241.8
script6a.pbs script6b.pbs script6c.pbs	Optimized	4	4000 / 50 / 5000	156.3	44.8
script5a.pbs script5b.pbs script5c.pbs		4	8000 / 100 / 8000	162.4	45.0
script4a.pbs script4b.pbs script4c.pbs		4	16000 / 200 / 16000	124.8	36.5

Comparison Graph

Comparative Performance of Configurations on different memory directives



Graph Analysis

Metric	Baseline	16000 / 200 / 16000	Speedup (×)	Improvement (%)
CPU Time (s)	237.3	124.8	1.90× faster	47.4% improvement
Wall Time (s)	240.9	36.5	6.60× faster	84.8% improvement

Optimization NWChem
1) Implementation of Fast Storage

Original input file

```
start w12_b3lyp_cc-pvtz_energy

memory stack 8000 mb heap 100 mb global 8000 mb noverify    # you can change this

permanent_dir .     # you can change this
scratch_dir /tmp    # you can change this
```

```
# Change to working directory
mkdir -p ${HOME}/scratch/${USER}/nwchem/run/${PBS_JOBID}
cd       ${HOME}/scratch/${USER}/nwchem/run/${PBS_JOBID}
```

Adjusted input file

```
start w12_b3lyp_cc-pvtz_energy
memory stack 8000 mb heap 100 mb global 8000 mb noverify
permanent_dir .
scratch_dir /scratch/${USER}/${PBS_JOBID}
```

```
# Create and move to scratch directory
mkdir -p /scratch/${USER}/${PBS_JOBID}
cd /scratch/${USER}/${PBS_JOBID}
```

Reason: To move the huge amount of temporary data in job

- Separating the slow and fast storage and directing the high-volume data to the parallel file system

Optimization NWChem
2) Hybrid Parallelization

Original script

```
export OMP_NUM_THREADS=1
```

```
time mpirun -np ${NCPUS:-104} \
```

Adjusted script

```
export OMP_NUM_THREADS=4
```

```
time mpirun -np 104 --map-by ppr:26:node:PE=4 --bind-to core \
```

- Better memory use, where threads share memory efficiently within a node

Optimization NWChem
2) Hybrid Parallelization

MPI and OpenMP Configuration

Parameter	Value	Description
OMP_NUM_THREADS	4	OpenMP threads per MPI process
--map-by	ppr:26:node:PE=4	26 MPI ranks per nodes, 4 cores per rank
--bind-to	core	Bind processes to cores

- The configuration uses hybrid MPI+OpenMP parallelization
- Total cores per MPI process: 4 (OMP_NUM_THREADS)
- Total MPI processes per node: 26

Optimization NWChem
3) Changes in Input File

Original input file

```
start w12_b3lyp_cc-pvtz_energy

memory stack 8000 mb heap 100 mb global 8000 mb noverify
```

```
# you can remove this entire block if you want
scf
  direct   # you can change this
  singlet
  rhf
  thresh 1e-7
  maxiter 100
  vectors input atomic output w12_scf_cc-pvtz.movecs
  noprint "final vectors analysis" "final vector symmetries"
end

task scf energy ignore
# /end thing you can remove

dft
  direct   # you can change this
  xc b3lyp
  grid fine
  iterations 100
  vectors input w12_scf_cc-pvtz.movecs   # remove this if you remove the block above
  noprint "final vectors analysis" "final vector symmetries"
end

task dft energy
```

Adjusted input file

```
start w12_b3lyp_cc-pvtz_energy
memory stack 16000 mb heap 200 mb global 16000 mb noverify
```

```
basis "ao basis" spherical noprint
  * library cc-pvtz
end

dft
  semidirect memsize 2000000000 filesize 0
  xc b3lyp
  grid fine
  iterations 100
  vectors input atomic
  noprint "final vectors analysis" "final vector symmetries"
end
task dft energy
```

- Removal of SCF energy block
- Changes in memory stack value

Challenges

- **Memory Management** – handling large molecular datasets and optimizing memory use is complex and resource-intensive.
- **Time-Consuming Process** – Frequent errors and repeated test runs increased total development time.

Conclusion

We successfully optimized NWChem performance on Gadi by applying:

- Input tuning for efficient workload balancing
- Fast storage configuration to reduce read/write delays
- MPI + OpenMP hybrid parallelism for better multi-node scalability
- Node scaling optimization to maximize CPU utilization

The optimized NWChem runs showed significant improvements in execution time and scalability, achieving faster Wall Time and better parallel efficiency as more cores were added.

TASK 2
DEEPSEEK-SGLANG
USING ASPIRE2P NSCC

Our objective

Challenge:

We were tasked with running the DeepSeek-R1 model on the SGLang framework using the ShareGPT dataset, with the goal of completing execution within just 7 minutes.

Objective:

To improve performance by accomplishing higher value of total token processing speed and other metrics.

Rules

- The jobs will be executed on **2 GPU servers with 8 GPUs per node** (16 H100 GPUs in total).
- Results with execution time beyond **420 seconds** are considered **INVALID**.

```
#!/bin/bash
#PBS -P 50000112
#PBS -l walltime=420
#PBS -l select=2:ncpus=112:ngpus=8:mpiprocs=2:mem=1880gb
#PBS -j oe
#PBS -m abe
##PBS -l other=hyperthread

module load cuda
```

Rules

- Benchmark prompts: **2000**
- Load format: **Dummy**
- Random seed: **2025**

```
-m sglang.bench_offline_throughput \
--model-path deepseek-ai/DeepSeek-R1 \
--dataset-path ${HOME}/scratch/ShareGPT_V3_unfiltered_cleaned_split.json \
--num-prompts 2000 --load-format dummy --seed 2025 --dtype bfloat16 \
```

Performance Metrics

01 **Total token throughput (tok/s): The most important metric –** combines input and output token processing speed

02 **Input token throughput (tok/s):** Speed of processing input tokens

03 **Output token throughput (tok/s):** Speed of generating output tokens

04 **Request throughput (req/s):** Number of requests processed per second

*Higher values indicate better performance.

Our Optimization

1. **NCCL communication tuning**
2. **CUDA efficiency settings**
3. **Warm-up execution**

*We combined these 3 optimizations into a script

DeepSeek Results

Baseline: sglang-baseline.sh (Original)

Number of nodes	Number of CPUs	Number of GPUs	Memory Used	Total token throughput (token/second)
2	112 x 2	8 x 2	1880 x 2	5839.65

```
:[1,0]<stdout>:====== Offline Throughput Benchmark Result =======
-[1,0]<stdout>:Backend:                                    engine
-[1,0]<stdout>:Successful requests:                        2000
-[1,0]<stdout>:Benchmark duration (s):                     173.88
-[1,0]<stdout>:Total input tokens:                         626729
-[1,0]<stdout>:Total generated tokens:                     388685
-[1,0]<stdout>:Last generation throughput (tok/s):         34.25
-[1,0]<stdout>:Request throughput (req/s):                 11.50
-[1,0]<stdout>:Input token throughput (tok/s):             3604.32
-[1,0]<stdout>:Output token throughput (tok/s):            2235.33
-[1,0]<stdout>:Total token throughput (tok/s):             5839.65
-[1,0]<stdout>:==================================================
```

DeepSeek Results

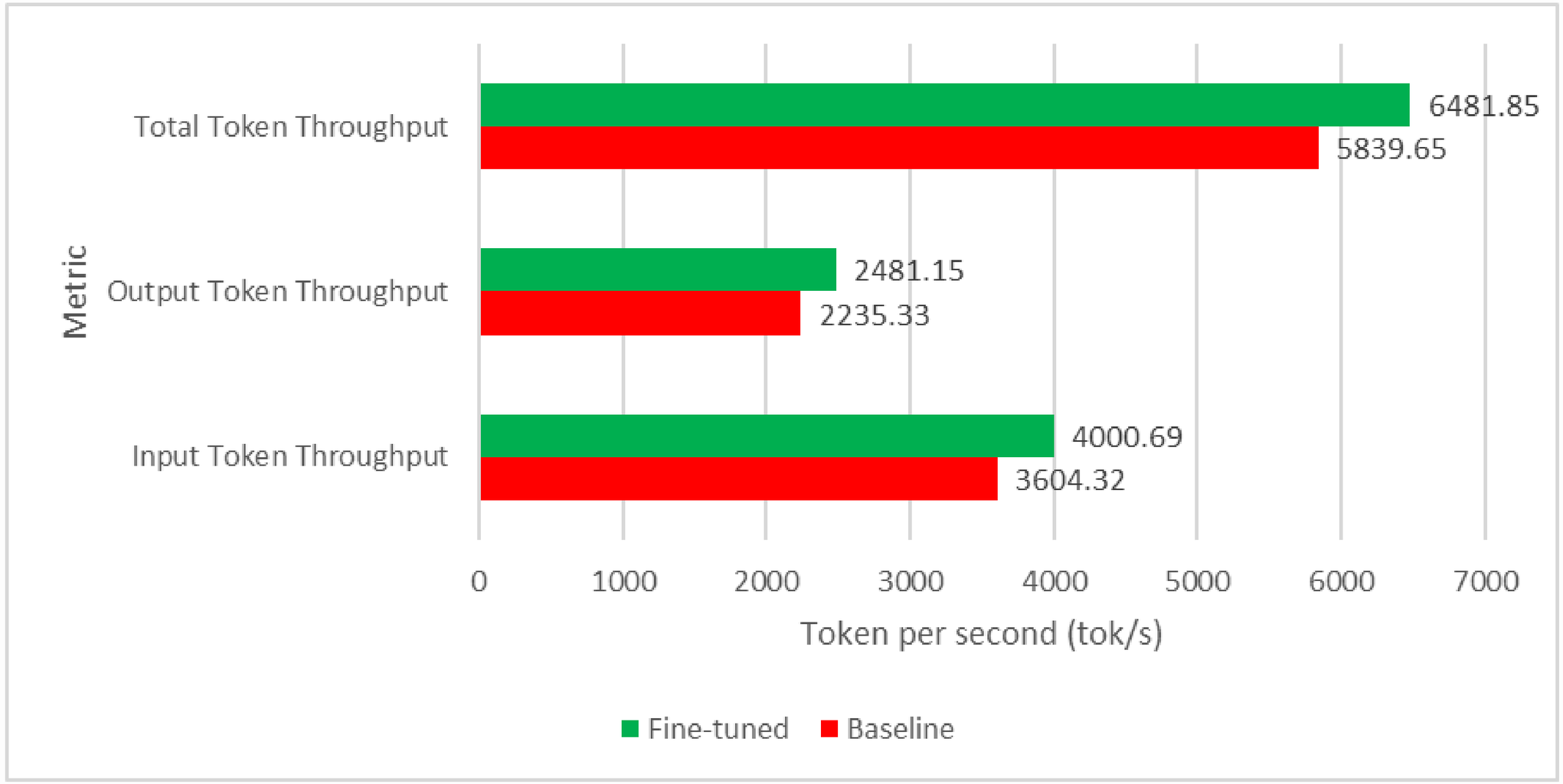
sglang-warmup.sh (Optimized)

Number of nodes	Number of CPUs	Number of GPUs	Memory Used	Total token throughput (token/second)
2	112 x 2	8 x 2	1880 x 2	6481.85

```
[1,0]<stdout>:====== Offline Throughput Benchmark Result =======
[1,0]<stdout>:Backend:                                    engine
[1,0]<stdout>:Successful requests:                        2000
[1,0]<stdout>:Benchmark duration (s):                     156.66
[1,0]<stdout>:Total input tokens:                         626729
[1,0]<stdout>:Total generated tokens:                     388685
[1,0]<stdout>:Last generation throughput (tok/s):         77.36
[1,0]<stdout>:Request throughput (req/s):                 12.77
[1,0]<stdout>:Input token throughput (tok/s):             4000.69
[1,0]<stdout>:Output token throughput (tok/s):            2481.15
[1,0]<stdout>:Total token throughput (tok/s):             6481.85
[1,0]<stdout>:================================================
```

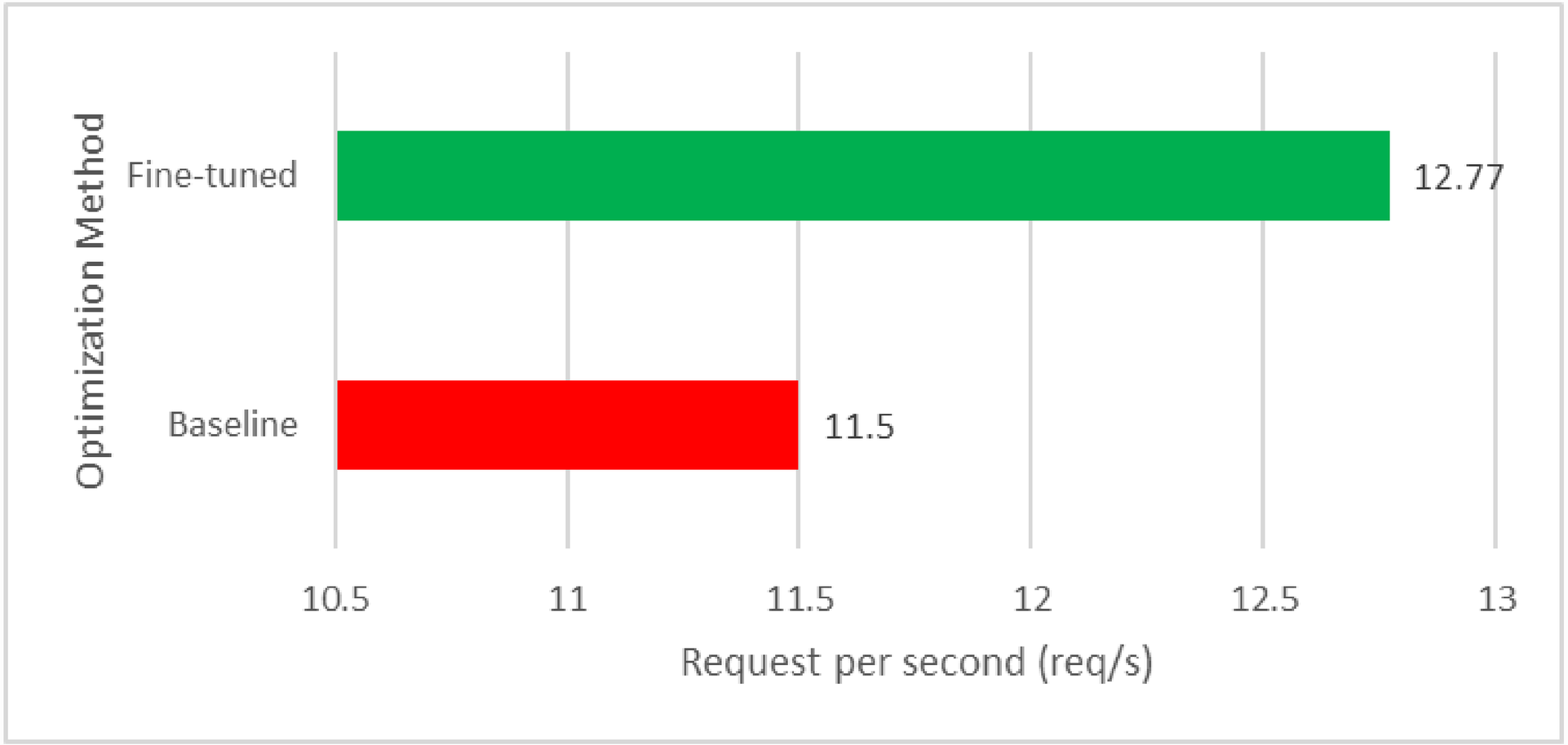
DeepSeek Results Comparison

Total, output, input token throughput



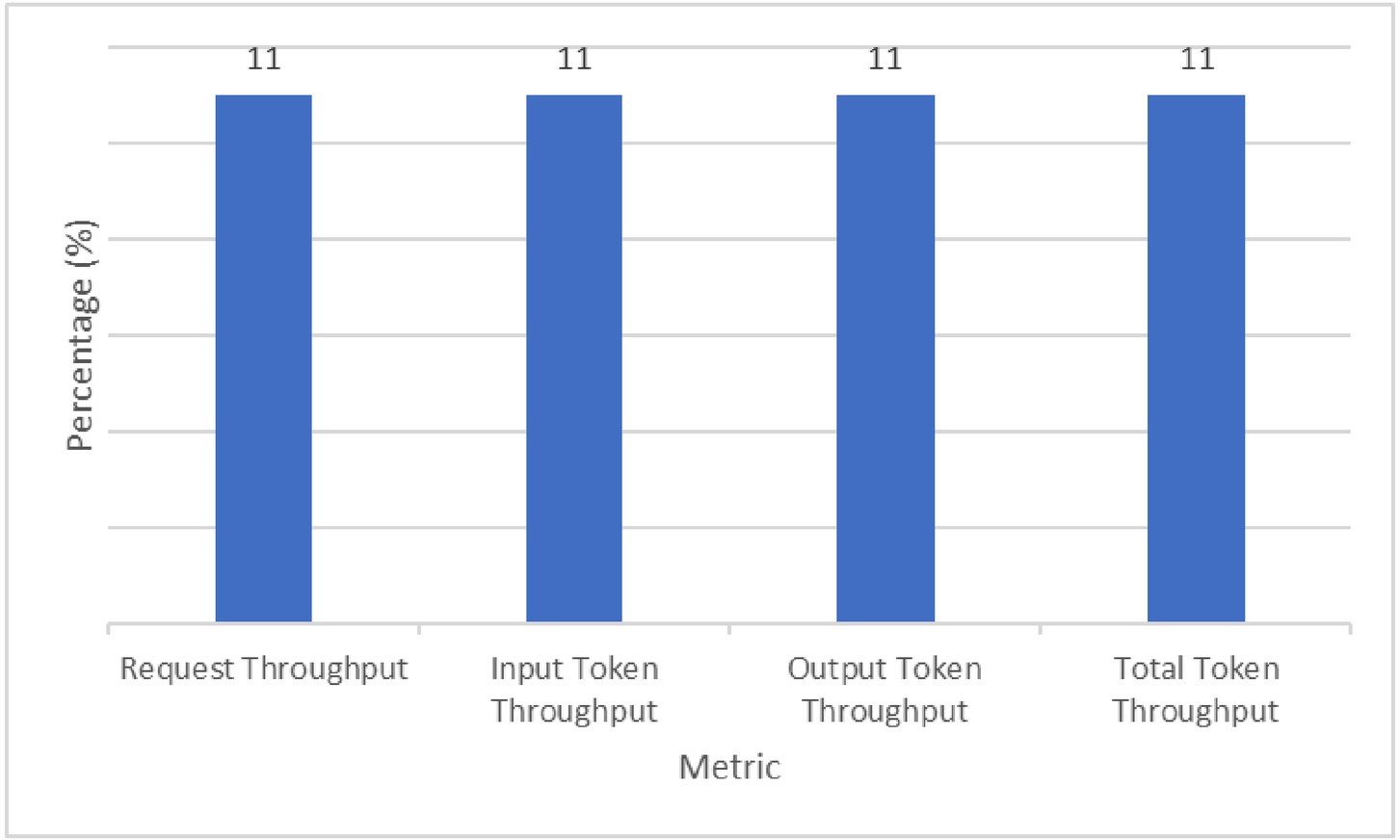
DeepSeek Results Comparison

Request throughput



DeepSeek Results Comparison

Percentage Improvement



Fine-tuning improved overall token throughput by **11%**

Script Explanation
Improved script:
NCCL communication tuning

Specifies mlx5 RDMA interface to use for communication

```
export NCCL_IB_HCA=mlx5
```

Control when to use GPU Direct RDMA between a NIC and a GPU

```
export NCCL_NET_GDR_LEVEL=PHB
```

Specifies the network interfaces NCCL should use: InfiniBand

```
export NCCL_SOCKET_IFNAME="ib0,bond0,eno1,eth0"
```

Script Explanation

Improved script: CUDA efficiency setting

Limits how many concurrent connections each CUDA device (GPU) can have

```
export CUDA_DEVICE_MAX_CONNECTIONS=1
```

Makes only GPUs 0–7 visible to the program.

```
export CUDA_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
```

Disable TF32 to only adhere to bfloat16 for more math accuracy

```
export NVIDIA_TF32_OVERRIDE=0
```

Script Explanation
Improved script

```
COMMON_FLAGS="\
  --model-path deepseek-ai/DeepSeek-R1 \
  --dataset-path ${HOME}/scratch/ShareGPT_V3_unfiltered_cleaned_split.json \
  --seed 2025 \
  --dtype bfloat16 \
  --trust-remote-code \
  --tp 16 --nnodes 2 \
  --dist-init-addr ${DIST_INIT_ADDR}:${DIST_INIT_PORT} \
  --node-rank \${OMPI_COMM_WORLD_RANK} \
  \
"
```

```
#-------- OPTIMIZATION: WARM UP ---------
/usr/mpi/gcc/openmpi-4.1.7a1/bin/mpirun \
  -hostfile ${PBS_NODEFILE} \
  -map-by ppr:1:node \
  -bind-to none \
  -x PATH \
  -x NCCL_DEBUG=INFO \
  -x DIST_INIT_ADDR=$(HEAD -N 1 $PBS_NODEFILE) \
  bash -c "time ${PYTHON} -m sglang.bench_offline_throughput \
    ${COMMON_FLAGS} \
    --num-prompts 64 \
    --load-format dummy \
  " || true
```

Challenges

NO	ISSUE	PROBLEM
1	Walltime exceeded	Execution finished, but no output
2	Batch jobs stuck in a queue	High cluster load during busy times

Possible Improvements

- **Profiling** - to identify bottlenecks and apply parallelism in the hotspot areas.

- **Dynamic Load Balancing** – distribute workload evenly across GPUs to prevent stragglers in multi-node setups.

Conclusion

We are able to optimize SGLang-based DeepSeek inference performance by applying:

- NCCL communication tuning
- CUDA efficiency setting
- MPI parallelism
- Warm up execution

The optimized DeepSeek model has shown an increasing improvements in throughput and scalability, achieving higher token generation speed and better GPU utilization. These optimizations allow the model to fully leverage the underlying multi-GPU (16×H100) infrastructure, delivering faster inference times while maintaining output consistency and accuracy.

THANK YOU!

UPM Team 2

Universiti Putra Malaysia