2022 APAC HPC-AI

Team NTHU-1 Presentation

Hao-Lung, Hsiao Hsin-Ping, Peng
Pin-Syuan, Lee Chun-Mu, Weng
Hsin-Cheng, Tu Jing-Yu, Yang

Dept. of Computer Science, Nat'l Tsing Hua U.

October 24, 2022

1. High Performance Computing with Quantum ESPRESSO
   - Parameters
   - Single Node
   - Multiple Nodes
   - Additional Supplement
2. Communications Performance with UCX
   - Introduction
   - Optimized Configurations
   - Conclusion
   - Running on DGX-A100
3. Deep-Learning-based DNA Sequence fast decoding
   - Why do we choose this model?
   - Model structure
   - Tuning
     - Batch size
   - Result

Section 1

High Performance Computing with Quantum ESPRESSO

1 High Performance Computing with Quantum ESPRESSO
- Parameters
- Single Node
- Multiple Nodes
- Additional Supplement

2 Communications Performance with UCX

3 Deep-Learning-based DNA Sequence fast decoding

Parameters

`-np`, `-diag`, `OMP_NUM_THREADS`

Number of Processes (`-np`)

- Each node has 48 processes
- Cannot greater than 1536, must be an integer in $[1, 48]$ or $48x | 1 \leq x \leq 32$
- Has effects on `-npool` & diagonalization

Diagonalization (`-ndiag`)

- Organized in a square 2D grid
- `-ndiag` needs to be n^2, where n is an integer $\leq \frac{-\mathtt{np}}{-\mathtt{npool}}$

Number of OpenMP Threads (`OMP_NUM_THREADS`)

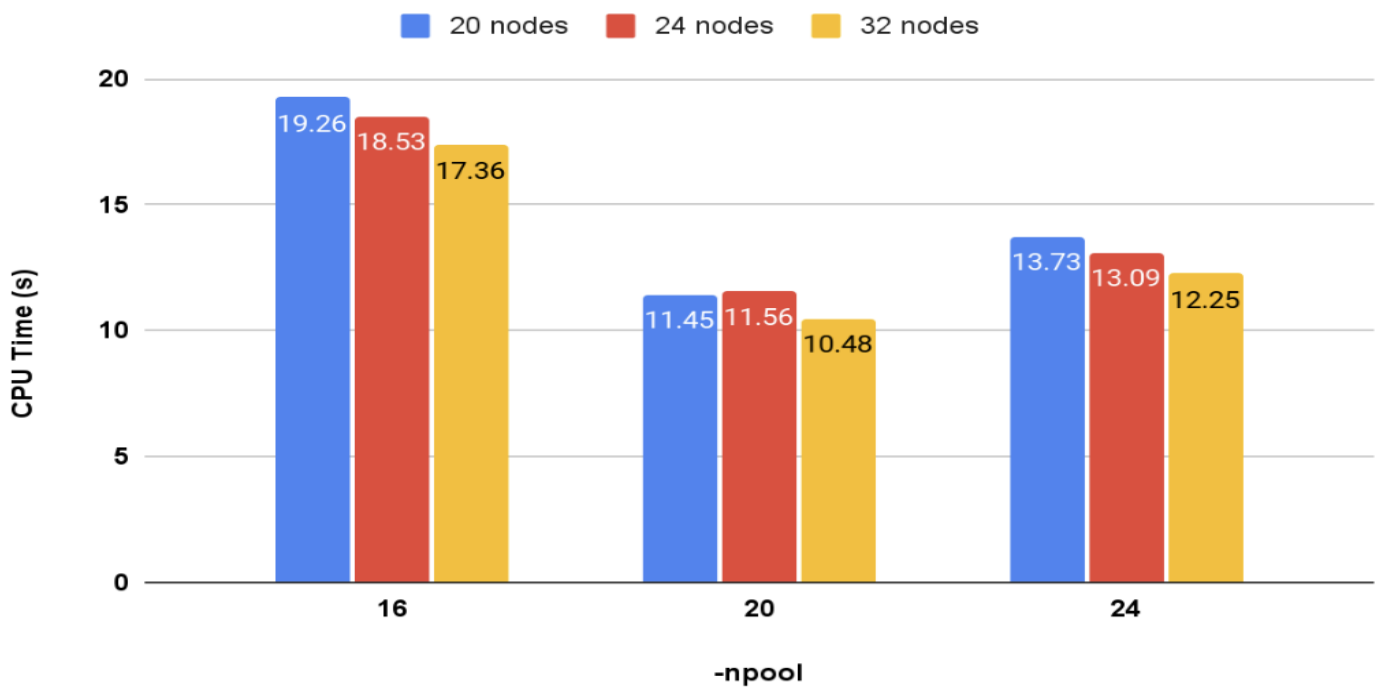
- `OMP_NUM_THREADS` ↑, CPU Time ↑ (fixed -npool & `NCPUS`)
- Set `OMP_NUM_THREADS` to 1 and start to tune the other parameters

Parameters

-npool

Number of Pools (-npool)

- Must be a divisor of the total number of processes (-np)
- K-points will be subpartitioned into several pools
- The More -npool $\not\Rightarrow$ The Better CPU Time



Single Node Performance of Gadi module

Average of 5 Times

# CPUs (`np`)	# pools (`npool`)	# linear algebra groups (`ndiag`)	CPU time [s]
48	24	4	1m53.138s
48	24	1	1m53.540s
40	20	4	1m52.794s
40	20	1	1m52.941s

Single Node Performance of Intel Compiler + Intel MPI

Average of 5 Times

# CPUs (`np`)	# pools (`npool`)	# linear algebra groups (`ndiag`)	CPU time [s]
48	24	4	1min58.916s
48	24	1	1min59.004s
40	20	4	1min56.944s
40	20	1	1min57.084s

Summary

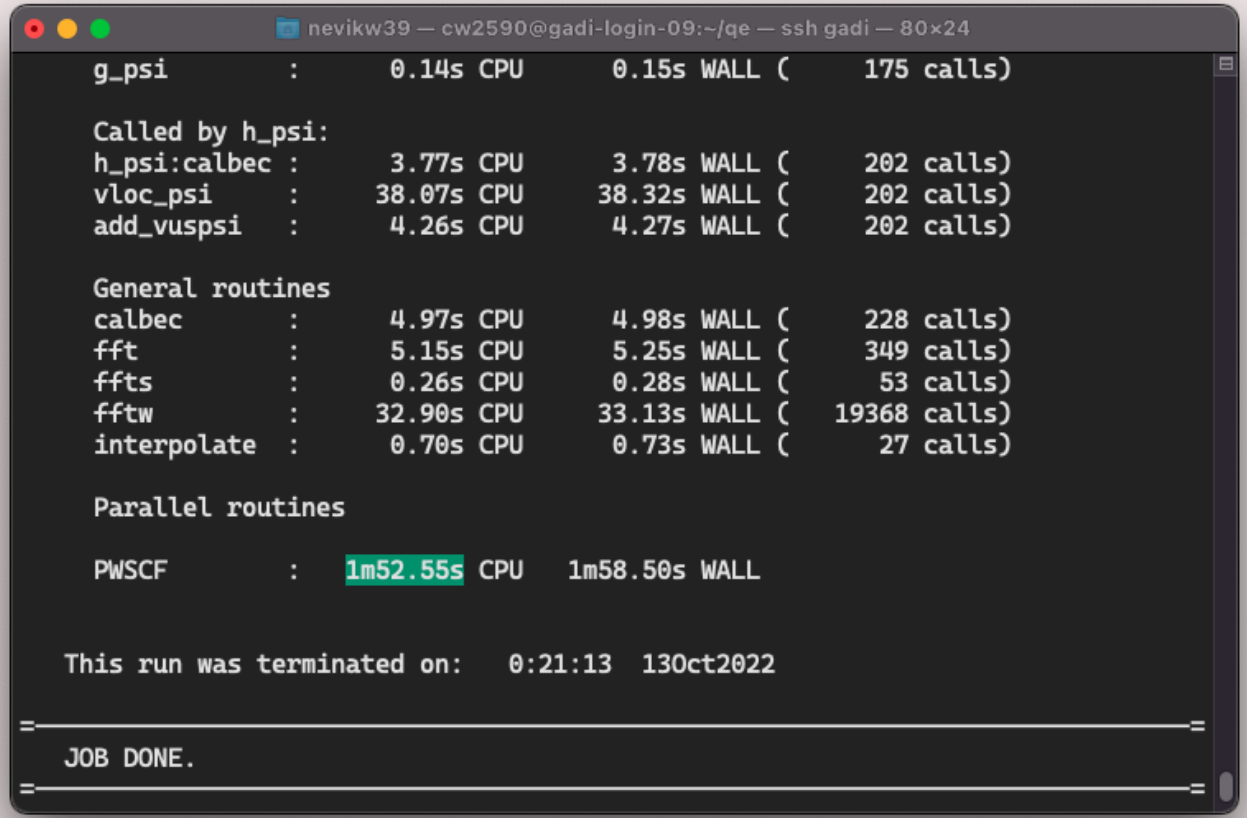
Script

```
#!/bin/bash
#PBS -l walltime=00:10:00
#PBS -l ncpus=40
#PBS -l mem=190GB
#PBS -l software=qe
#PBS -l wd
#PBS -P jx00
#PBS -N QE-single

module load qe
export OMP_NUM_THREADS=1
mpirun -np 40 pw.x -npool 20 -ndiag 4 -inp CeO2.in
```

Summary (cont.)

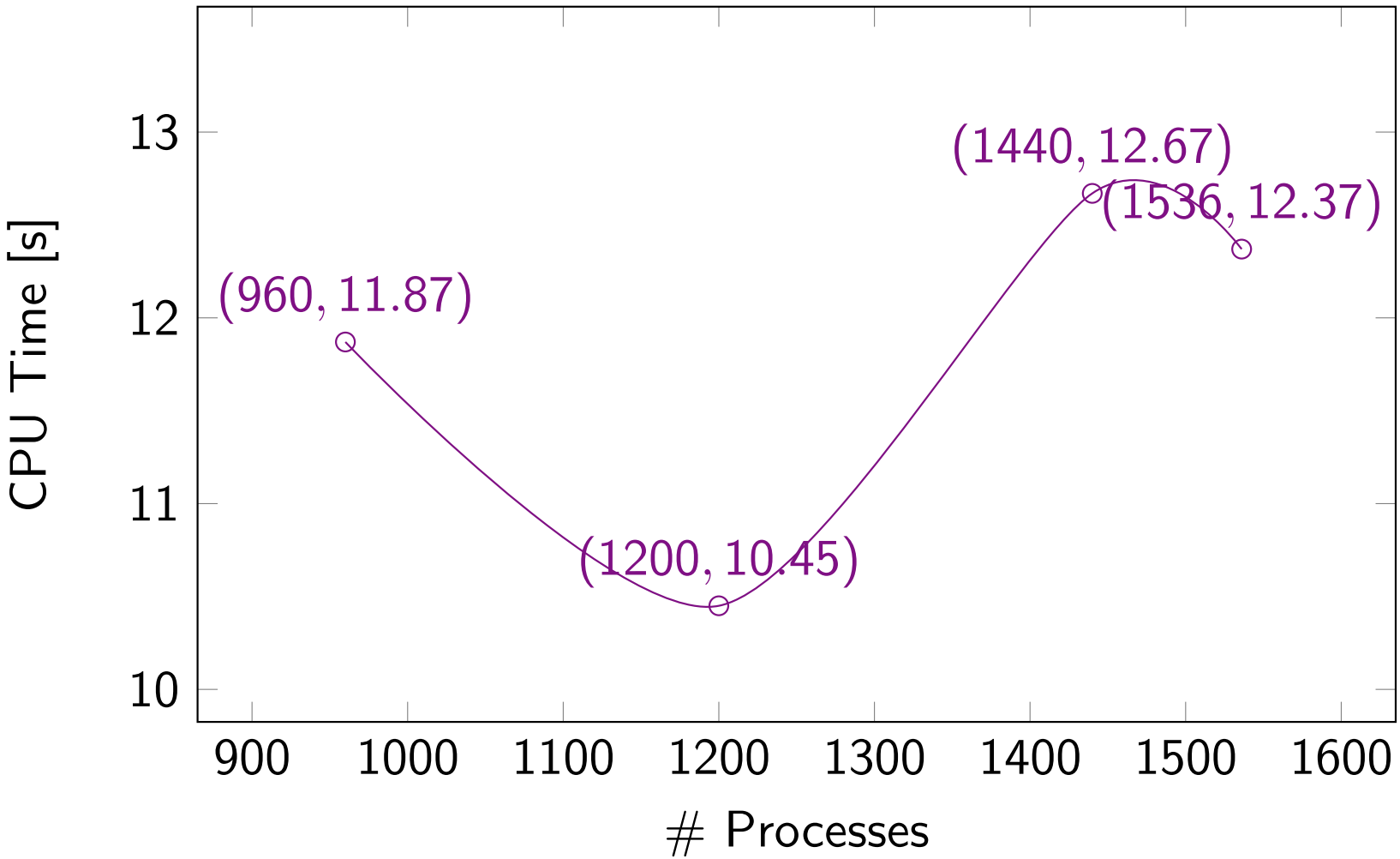
Result



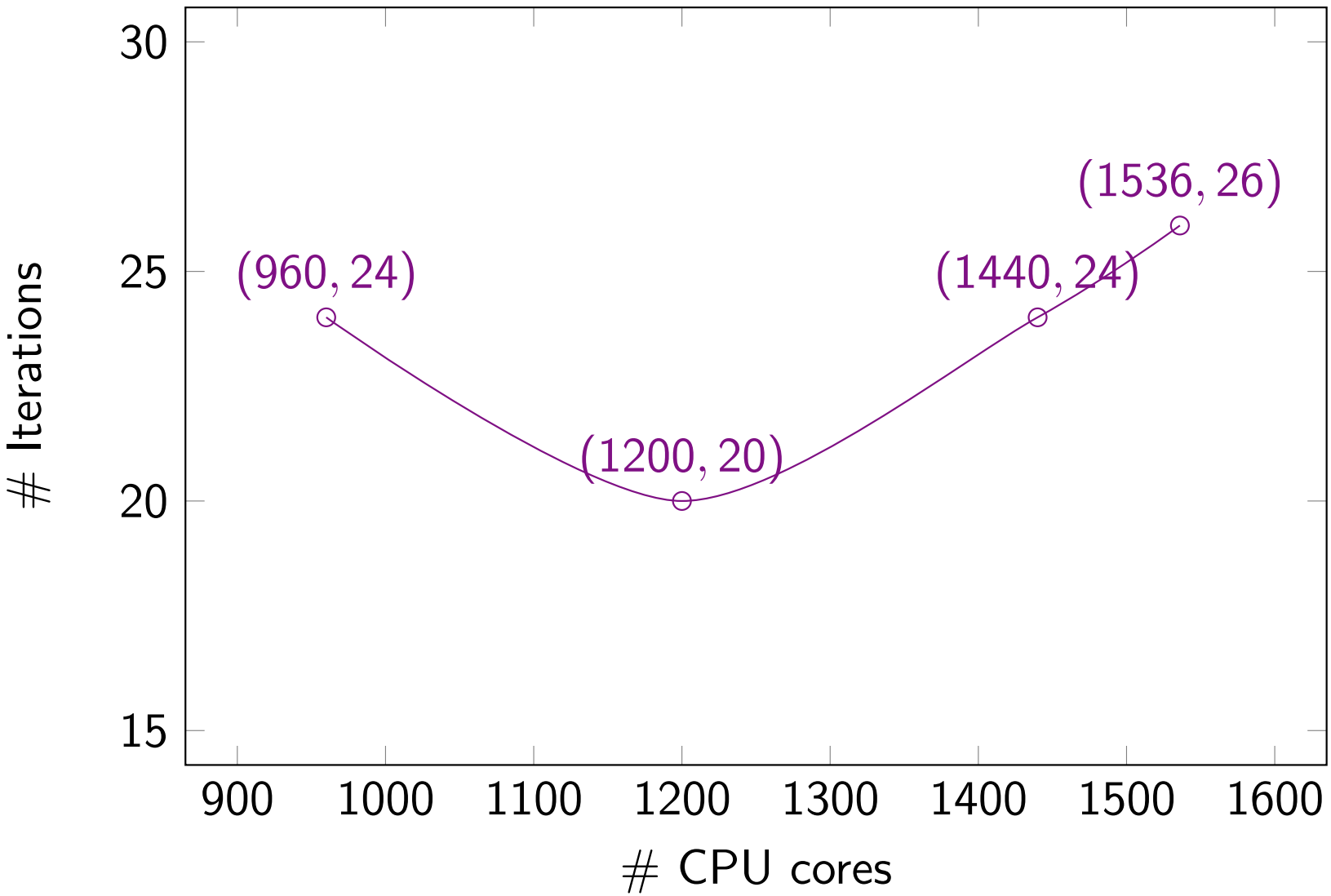
CPU Time vs. # Processes

`npools` were 20, 20, 20, 24 resp.; `ndiags` were left as default

The More -np $\not\Rightarrow$ The Better!!

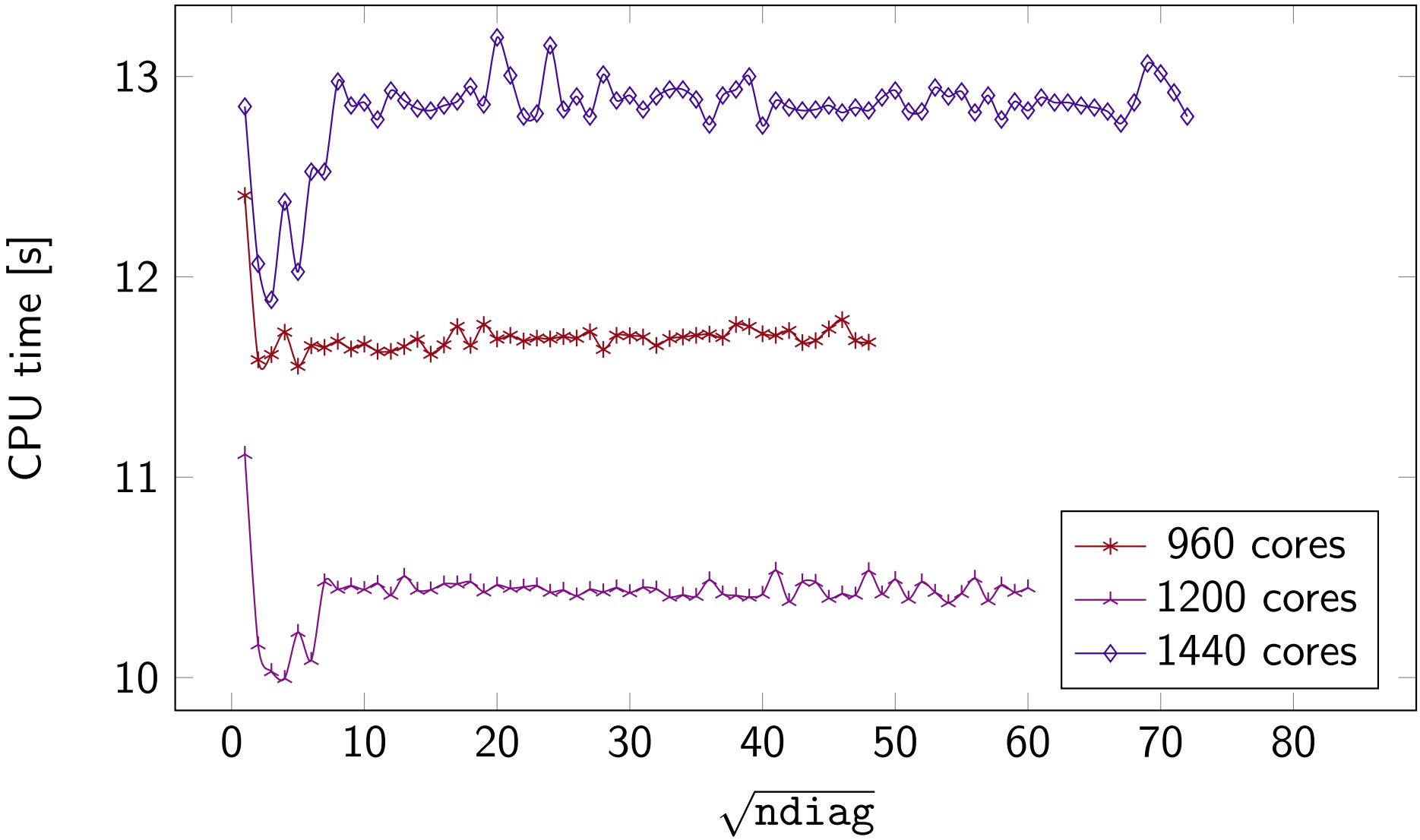


Iterations vs. # CPU cores



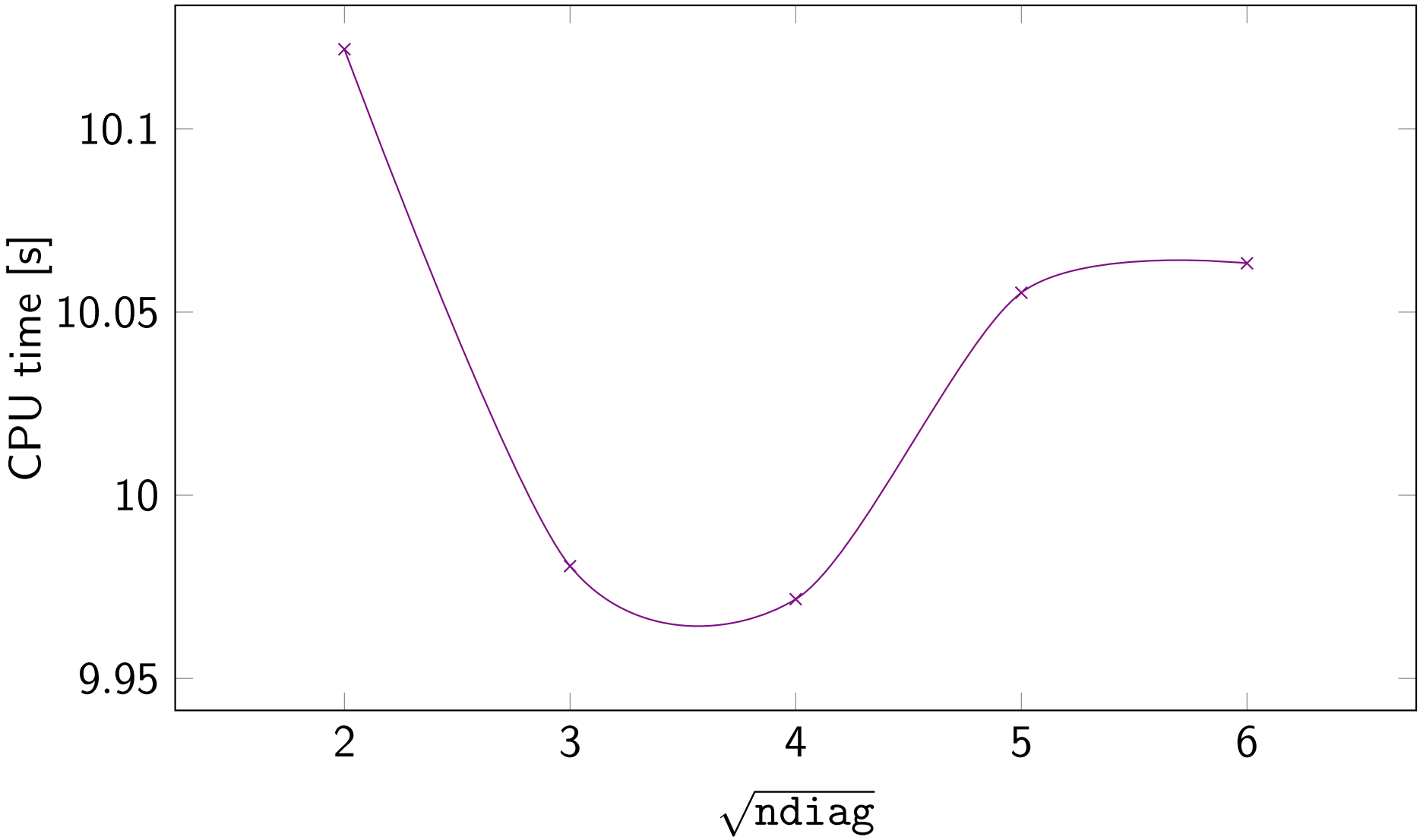
CPU time vs. ndiag of different CPU cores

Average of 5 Times



CPU time vs. ndiag of 1200 CPU cores

A closer, deeper insight



Conclusion

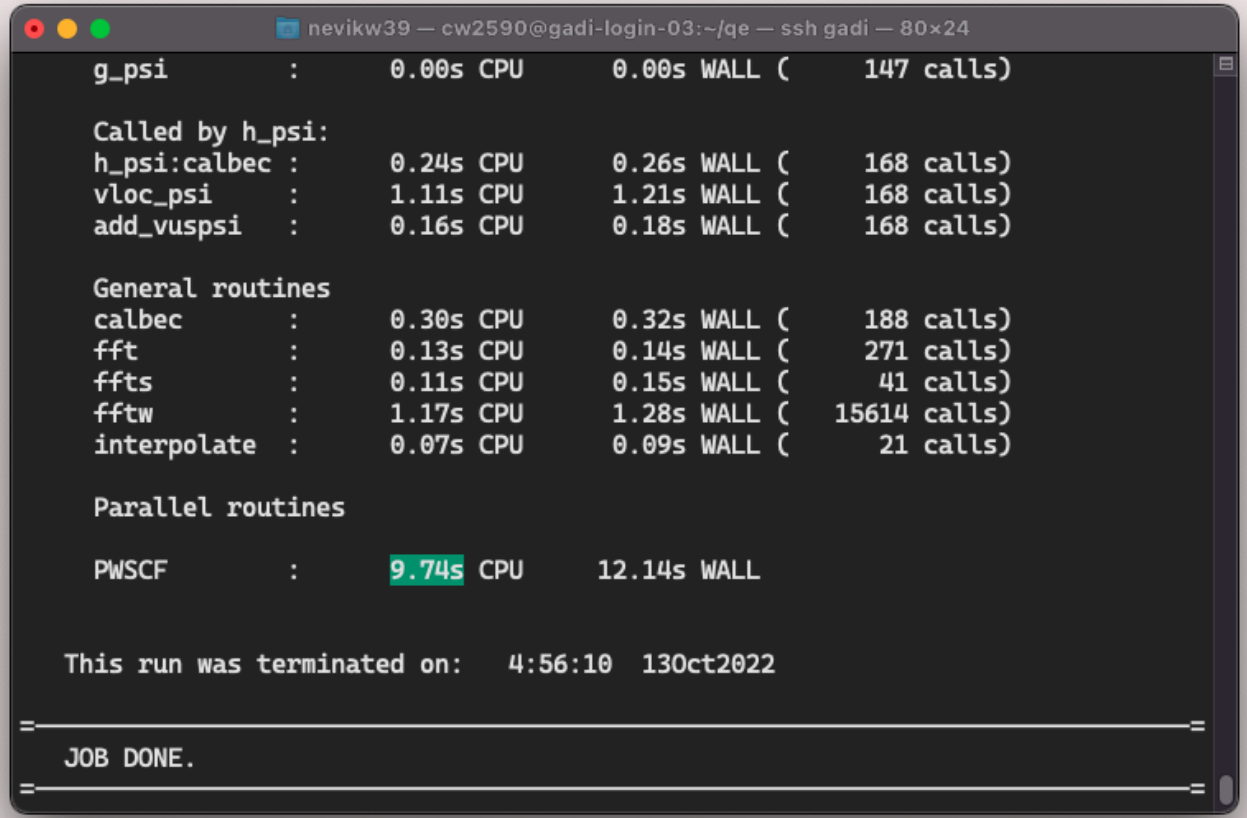
Script

```
#!/bin/bash
#PBS -l walltime=00:10:00
#PBS -l ncpus=1200
#PBS -l mem=760GB
#PBS -l software=qe
#PBS -l wd
#PBS -P jx00
#PBS -N QE-multi

module load qe
export OMP_NUM_THREADS=1
mpirun -np 1200 pw.x -npool 20 -ndiag 16 -inp CeO2.in
```

Conclusion (cont.)

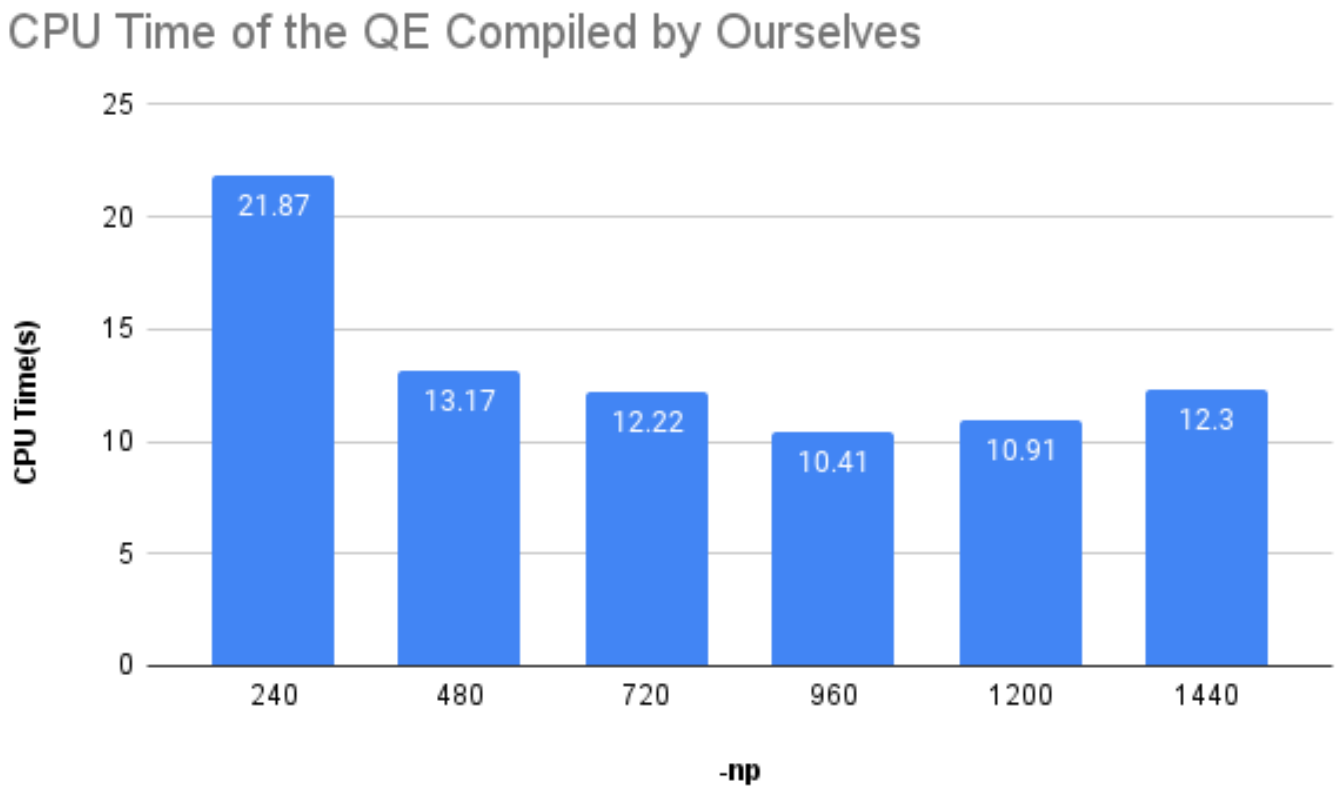
Result



Additional Supplement

Compare to QE compiled by ourselves

- QE compiled by Intel Compiler with IntelMPI will have the better performance when -np equals to 960 instead of 1200 like the QE module on Gadi



Section 2

Communications Performance with UCX

1 High Performance Computing with Quantum ESPRESSO

2 Communications Performance with UCX
- Introduction
- Optimized Configurations
- Conclusion
- Running on DGX-A100

3 Deep-Learning-based DNA Sequence fast decoding

Problem

The benchmark is running on 16 GPUs, on each of which is a worker there with left and right dataframe distributed throughout all GPUs.
It is our task that each worker merges its left and right dataframe, which means that a worker need communicate with others at scale.

Objectives

Bandwidth of a worker

W.L.O.G., let's consider the worker with rank 0.

Define that $\text{bw} = \frac{\sum_{i=1}^{15} \textit{size of Dataframe transferred}_i}{\sum_{i=1}^{15} \text{Wall time}_i}$.

$$\text{Bandwidth} = \frac{\text{bw}_{left} + \text{bw}_{right}}{2}$$

Throughput

$$\text{Throughput} = \frac{\#\text{ chunks} \times \text{Data processed}}{\text{Wall time}}$$

Baseline

Average of 10 iterations, small data set (i.e., each chunk with 10^6 rows), running on 16 GPUs over 4 Gadi Volta nodes.

Bandwidth

- 507.21 MiB/s

Throughput

- 4.27 GiB/s

Enable Hardware Tag Matching

Avg. of 10 iterations, small data set, Gadi Volta nodes

Config

```
export UCX_RC_MLX5_TM_ENABLE=y
export UCX_DC_MLX5_TM_ENABLE=y
```

Bandwidth

- 521.87 MiB/s
- 102.8% speedup

Throughput

- 4.36 GiB/s
- 102.1% speedup

Enable various optimizations intended for homogeneous environment

Avg. of 10 iterations, small data set, Gadi Volta nodes

Config

```
export UCX_UNIFIED_MODE=y
```

Bandwidth

- 511.68 MiB/s
- 100.8% speedup

Throughput

- 4.39 GiB/s
- 102.8% speedup

Increase the amount of buffers added every time the receive / send memory pool grows

Avg. of 10 iterations, small data set, Gadi Volta nodes

Config

```
export UCX_TCP_RX_BUFS_GROW=16
export UCX_TCP_TX_BUFS_GROW=16
```

Bandwidth

- 513.73 MiB/s
- 101.2% speedup

Throughput

- 4.68 GiB/s
- 109.6% speedup

Use **mutex** instead of **spinlock** for multithreading support in UCP

Avg. of 10 iterations, small data set, Gadi Volta nodes

Config

```
export UCX_USE_MT_MUTEX=y
```

Bandwidth

- 509.99 MiB/s
- 100.5% speedup

Throughput

- 4.71 GiB/s
- 110.3% speedup

Enable UCX-Py non-blocking mode

Avg. of 10 iterations, small data set, Gadi Volta nodes

Config

```
export UCXPY_NON_BLOCKING_MODE=1
```

Bandwidth

- 590.62 MiB/s
- 116.4% speedup

Throughput

- 4.96 GiB/s
- 116.1% speedup

Set *Rendezvous* protocol to use *Active Messages* scheme

Avg. of 10 iterations, small data set, Gadi Volta nodes

Config

```
export UCX_RNDV_SCHEME=am
```

Bandwidth

- 546.51 MiB/s
- 107.7% speedup

Throughput

- 5.54 GiB/s
- 129.7% speedup

Miscellanies

- UCX_IB_GPU_DIRECT_RDMA
- UCX_RNDV_THRESH
- UCX_TCP_TX_SEG_SIZE, UCX_TCP_RX_SEG_SIZE

Optimal Combination of Configurations

Config

```
export UCX_RC_TM_ENABLE=y
export UCX_DC_TM_ENABLE=y

export UCX_USE_MT_MUTEX=y

export UCXPY_NON_BLOCKING_MODE=1

export UCX_RNDV_SCHEME=am

export UCX_IB_GPU_DIRECT_RDMA=y
export UCX_RNDV_THRESH=1024
export UCX_TCP_TX_SEG_SIZE=64k
export UCX_TCP_RX_SEG_SIZE=512k
```

Overall Bandwidth Result

Avg. of 100 iterations on 16 GPUs over 4 Gadi Volta nodes

Small Data Set 1.06 GiB/s,
214.0% speedup in comparison to baseline (507.21 MiB/s)

Large Data Set 1.15 GiB/s,
290.1% speedup in comparison to baseline (405.89 MiB/s)

Overall Throughput Result

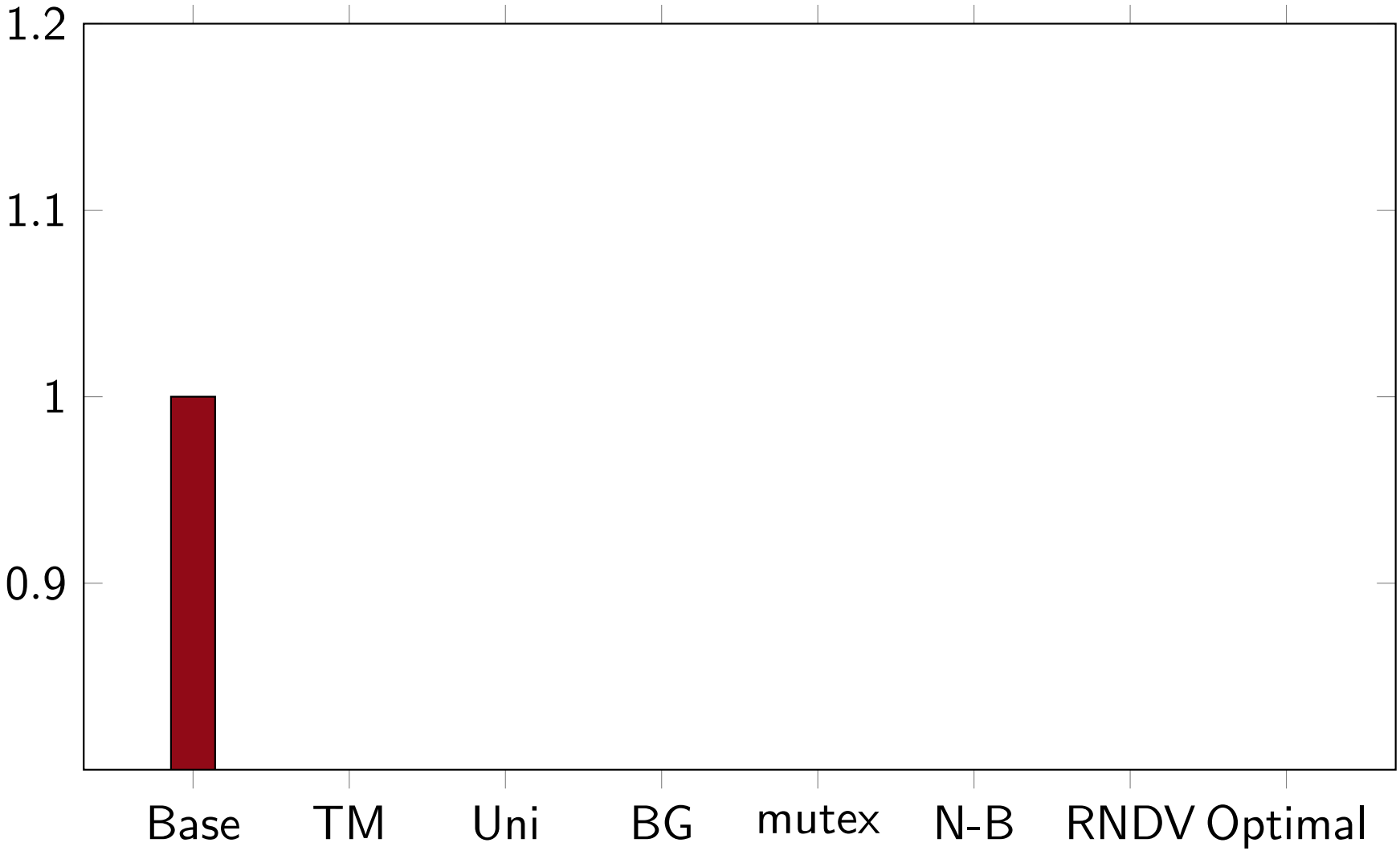
Avg. of 100 iterations on 16 GPUs over 4 Gadi Volta nodes

Small Data Set 9.28 GiB/s,
217.3% speedup in comparison to baseline (4.27 GiB/s)

Large Data Set 12.37 GiB/s,
281.1% speedup in comparison to baseline (4.40 GiB/s)

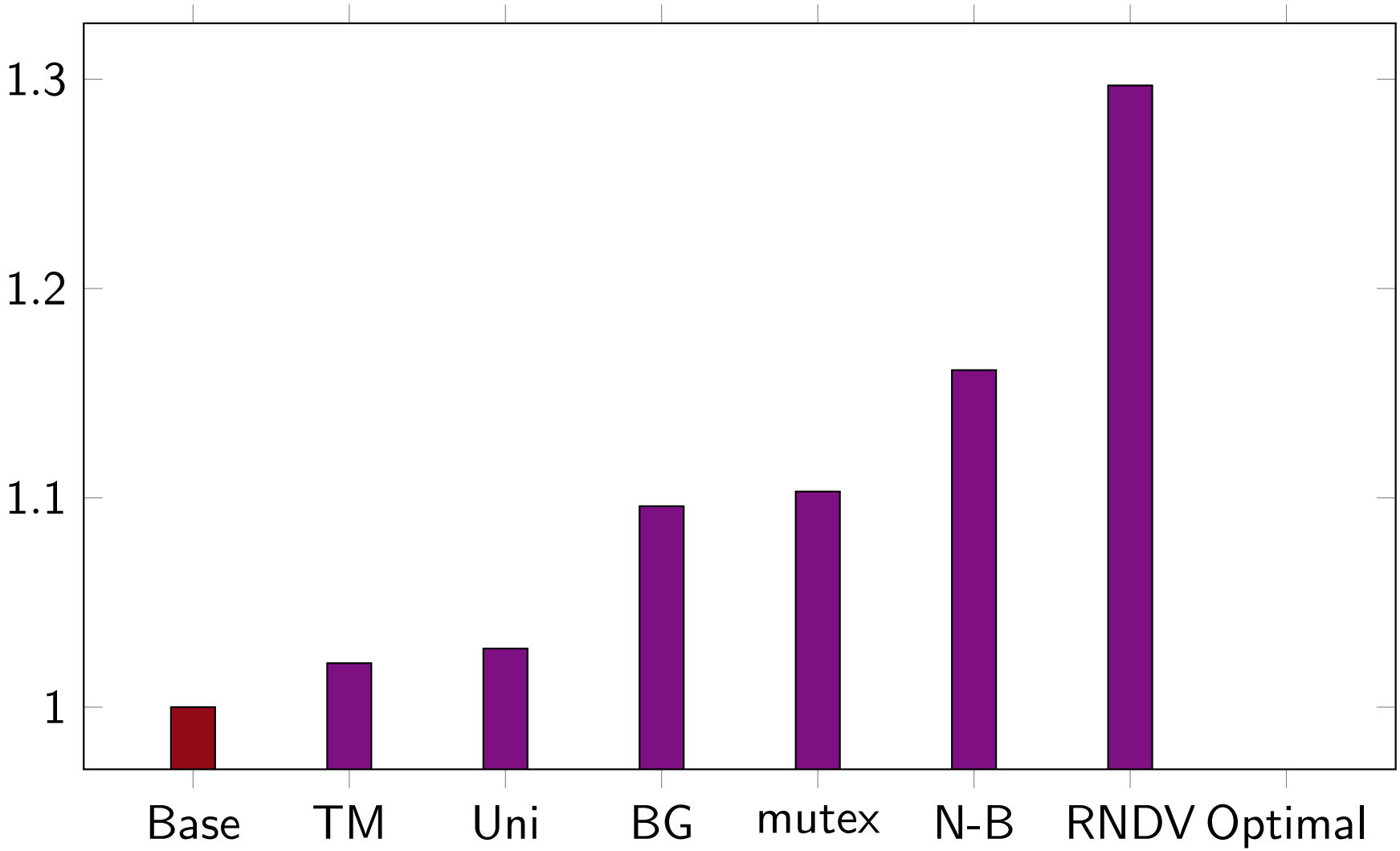
Bar graph of throughput speedup

Avg. of 10 iterations, small data set, Gadi Volta nodes



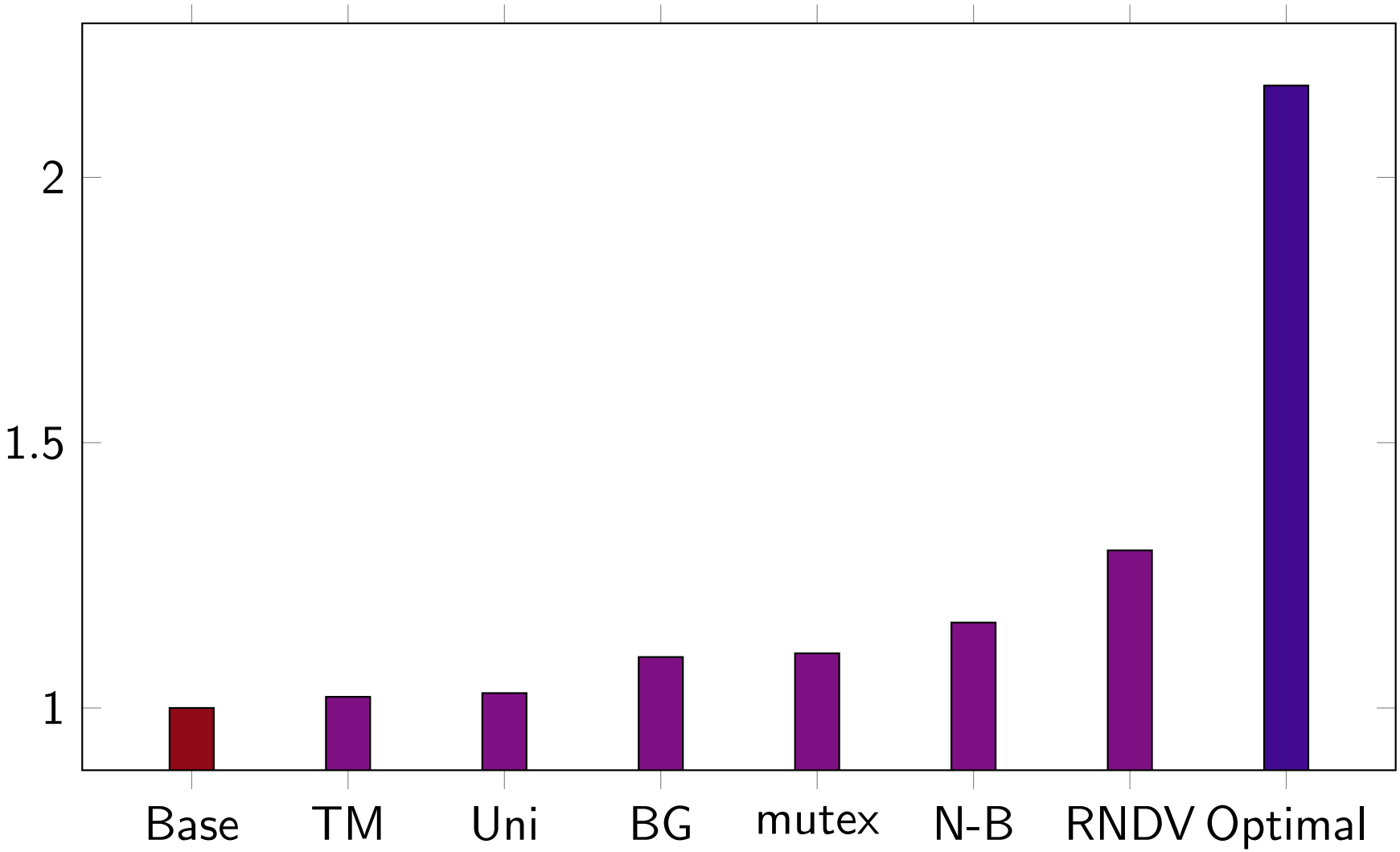
Bar graph of throughput speedup

Avg. of 10 iterations, small data set, Gadi Volta nodes



Bar graph of throughput speedup

Avg. of 10 iterations, small data set, Gadi Volta nodes



Overall Throughput Result on DGX-A100 nodes

Avg. of 100 iterations on 16 GPUs over 2 Gadi DGX-A100 nodes

Table: Average throughput of 100 times (With DGX-A100s)

Chunk size	**Default**	**Optimized Configurations**
10^6	16.66 GiB/s	16.69 GiB/s
2.5×10^7	88.29 GiB/s	34.89 GiB/s[1] 89.00 GiB/s[2]

Section 3

Deep-Learning-based DNA Sequence fast decoding

1 High Performance Computing with QUANTUM ESPRESSO

2 Communications Performance with UCX

3 Deep-Learning-based DNA Sequence fast decoding
- Why do we choose this model?
- Model structure
- Tuning
- Result

More accurate, more efficiency

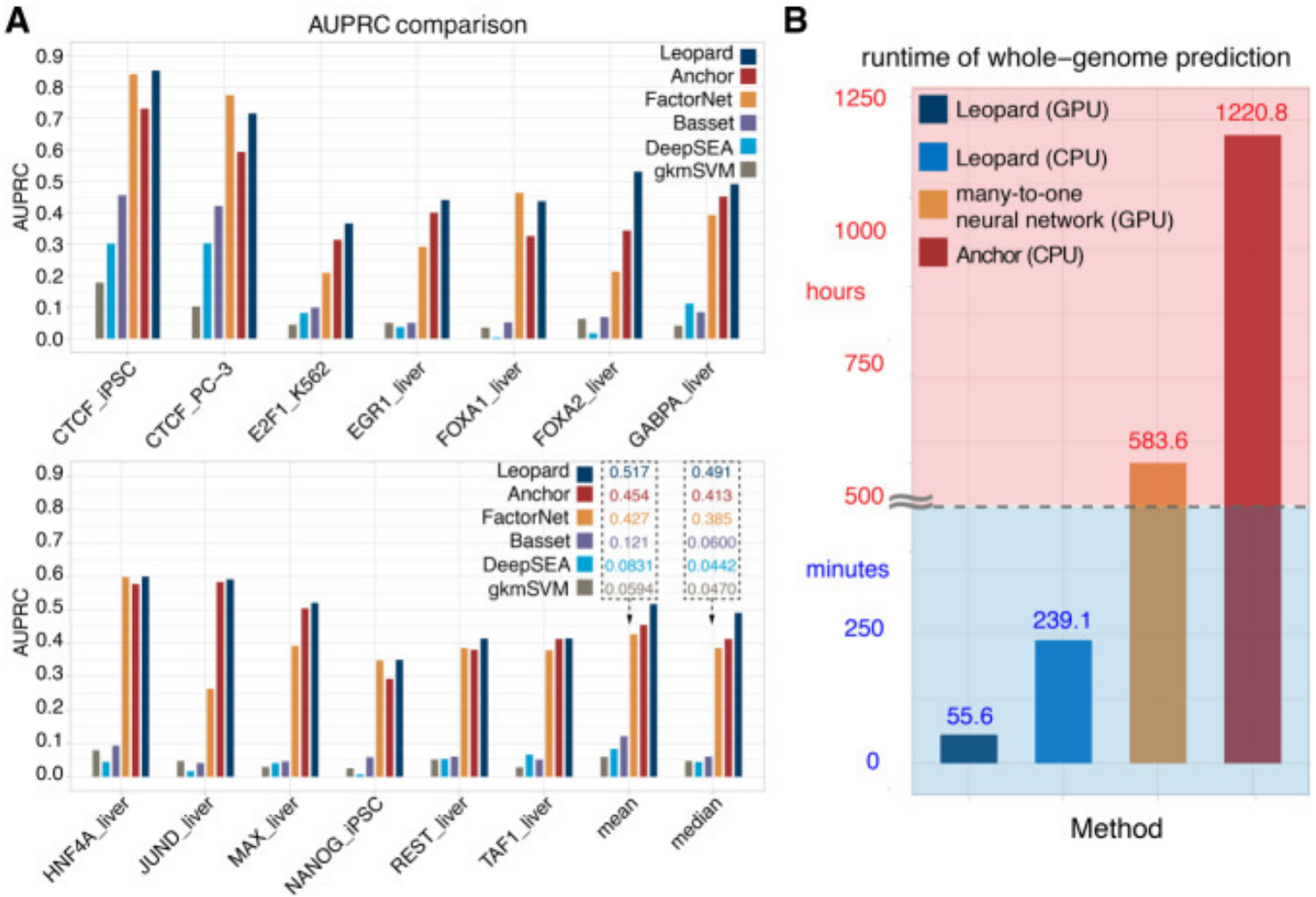


Figure: Model performance, adapted from the original paper

Main components

- Input
- Encoder (CCP blocks)
- Decoder (UCC blocks)
- Output

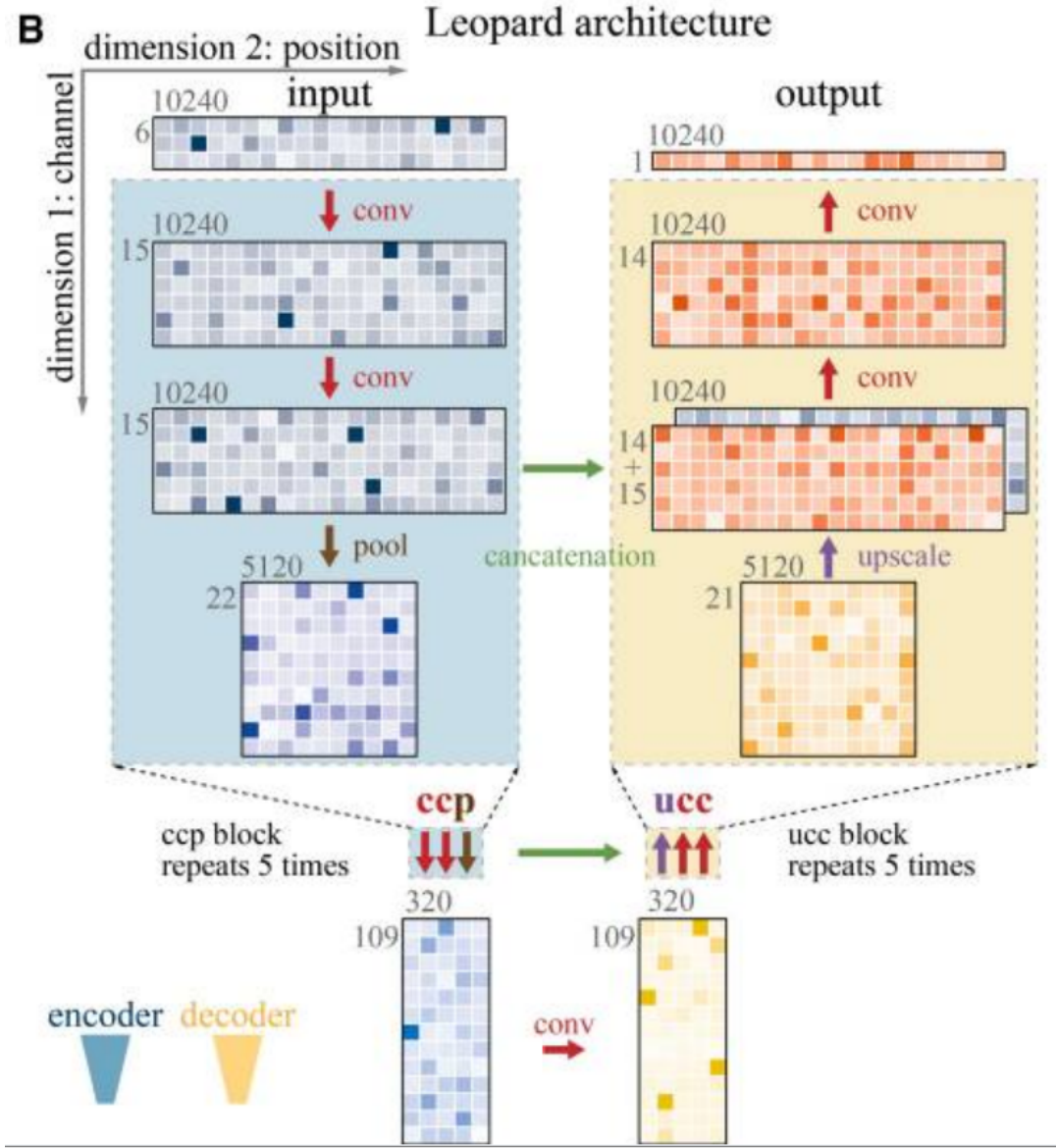


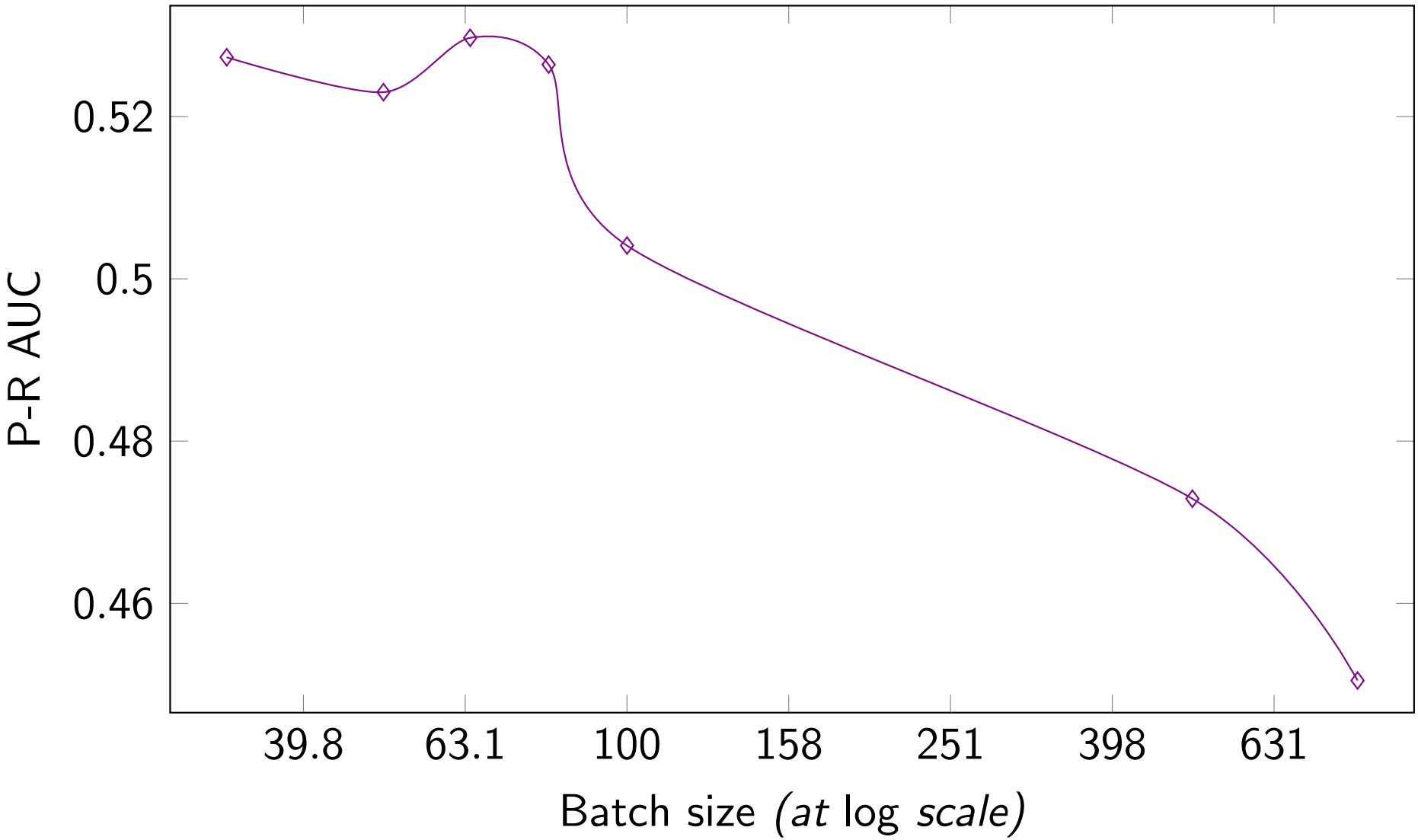
Figure: Model structure, adapted from the original paper

Brief summary

Some facts

- 61 layers
- 475969 parameters

Maximum P-R AUC ever reached of various batch sizes

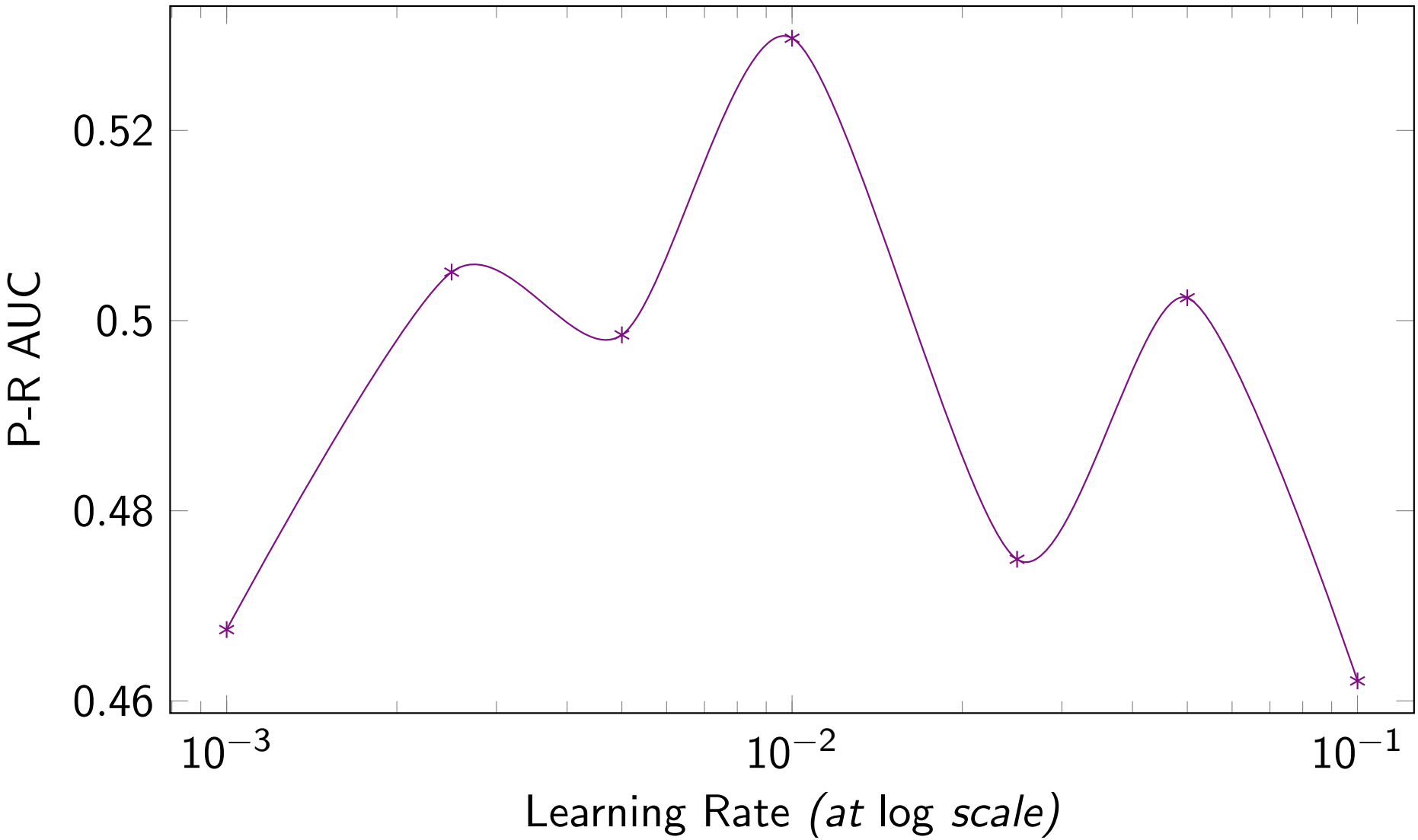


Batch sizes

- We tried batch size 32, 50, 80 and the default 100, yet their results were not optimal.

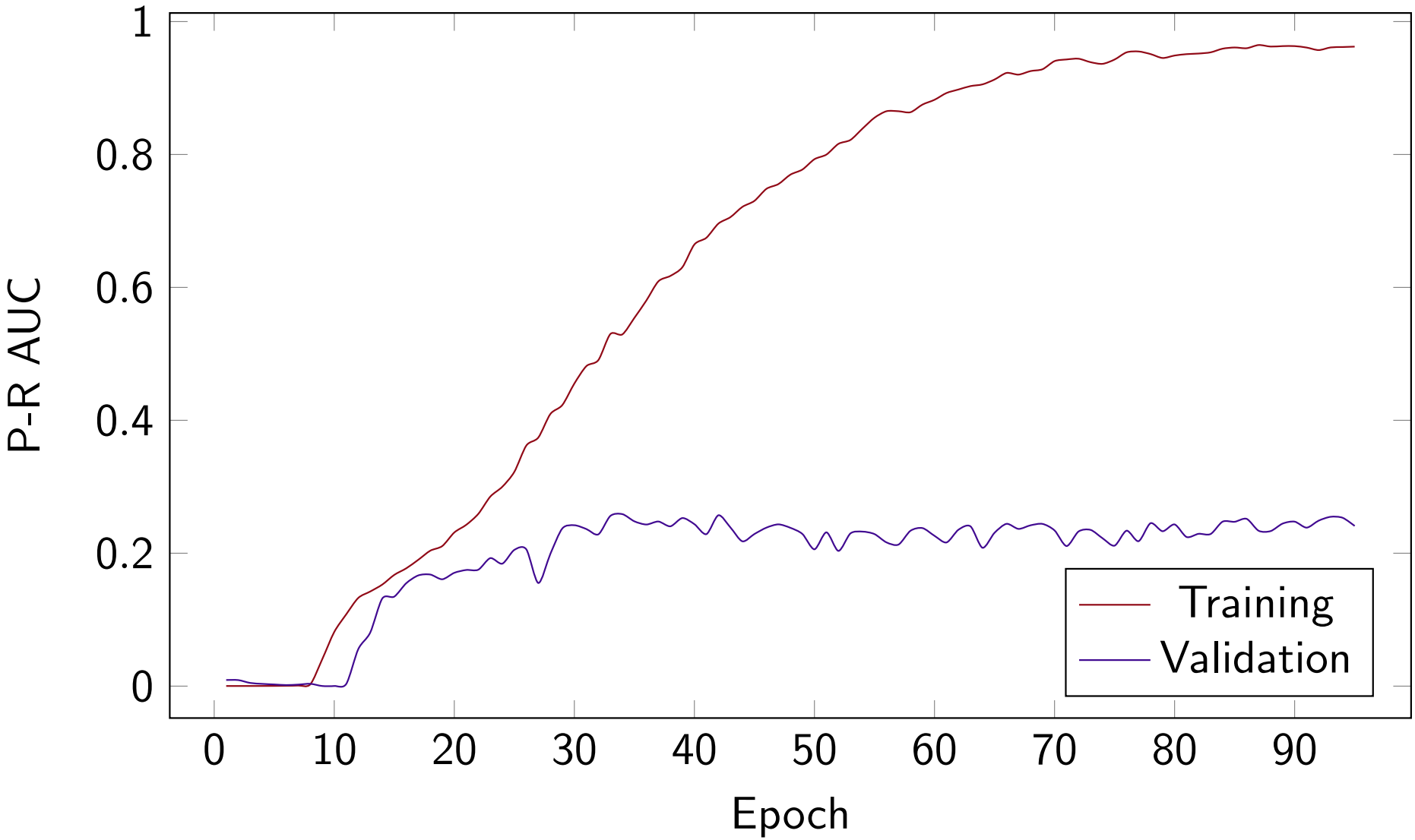
Batch size 64

Maximum P-R AUC of several learning rates



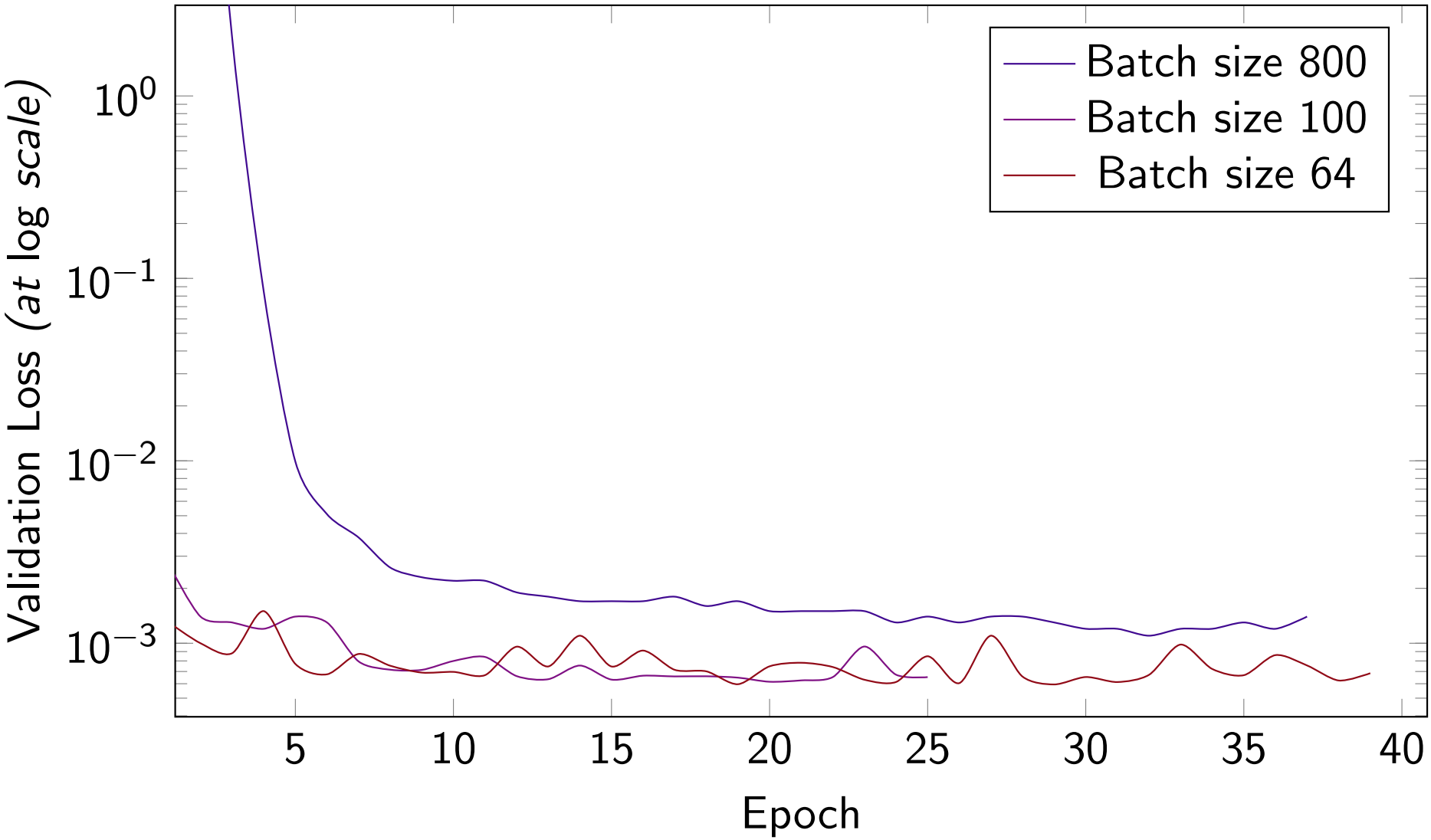
Batch size 500

Overfitting



Batch size 800

Validation loss cannot converge to less than 10^{-3} (Note that y-axis is at log scale)

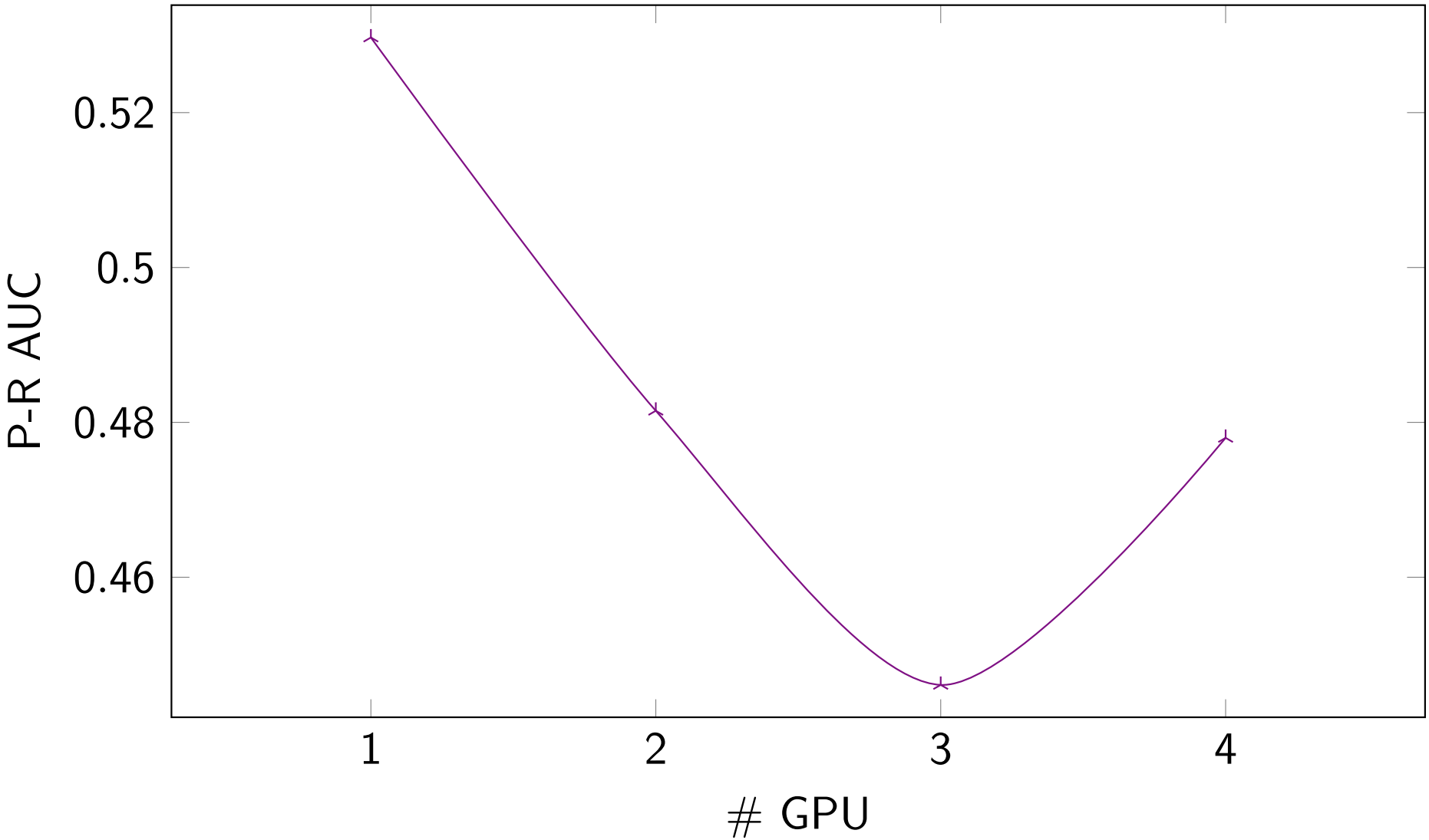


Batch size 1600, 2000, 2500

- On A100 only since memory usage would be over about 31GiB!
- Loss could hardly converge, similar with batch size 800

Parallelization via Multiple GPUs

Performance always counts more!



Volta and Ampère GPUs

- NVIDIA Volta V100: Main GPUs on Gadi
- NVIDIA Ampère A100: Faster and more memory

Hyperparameters and environments

Batch Size	64
Learning Rate	0.01
# GPU	1

Table: Performance of Single GPU

GPU Type	Loss *(Binary Crossentropy)*	P-R AUC	Dice coeff.	Binary Int. of Union	Training Time [s]
V100	7.2768×10^{-4}	0.5297	0.3705	0.3625	3170.19
A100	7.3153×10^{-4}	0.5288	0.4109	0.3701	941.63