Experience Sharing from Team National Tsing Hua University for 2023 APAC HPC-AI Competition

Zhan-Yi Lin *, Chen-An Pai*, Shih-Hsun Wei *, Hao-Tien Yu *, Ming-Chun Tsai *, Shih-Jou Cheng *,
Chih-Yu Hsieh *, Tung Yu Hsieh *, Wei-Chih Huang *, Ssu-Cheng Lai *, Wei-Po Lin *,
Chan-Yu Mou †, Jerry Chou †
Department of Computer Science, National Tsing Hua Univeristy, Taiwan
* contributed equally to this work as first authors
† corresponding author

***Abstract*—APAC HPC-AI Competition is one of the major international student competitions for developing young talents with interdisciplinary skills in the field of High Performance Computing (HPC) and AI (Artificial Intelligence). The goal of the competition is to optimize the performance of HPC and AI applications, such as the Climate Prediction Model MPAS and the Large Language Model Bloom for this year's competition. This paper summarizes the experience and performance optimization techniques we learned from the competition. As the champion of the competition in 2023, we hope our work can inspire more students to participate the competition and share the light on how to prepare for the competition and what skills are required for winning the competition.**

***Index Terms*—Student Competition, Performance Optimization, AI**

I. Introduction

The 2023 High-Performance Computing and Artificial Intelligence (HPC-AI) Asia-Pacific Competition was organized by the HPC-AI Advisory Council and the National Supercomputing Centre (NSCC) of Singapore. The competition attracted 22 outstanding teams from 11 countries and regions. The competition spanned from May 19 to November 16, 2023. The HPCAI competition for this year announced two distinct challenges: the Climate Prediction Model MPAS [1] and the Large Language Model Bloom [2]. The first challenge, MPAS, is restricted to be run on CPUs, while the second challenge, Bloom, is restricted to be run on GPUs. The combination of scientific computing with AI challenges aligns well with the spirit of HPCAI for developing young talents with interdisciplinary skills.

The goal of this competition is to accelerate the applications for the given performance objectives. The primary performance objective of MPAS is to enhance simulation speed, while Bloom focuses on minimizing the generation time per token. Code modification is not allowed in the competition, so we have to achieve performance optimization by the means of finding the best resource allocation, system environment setting, acceleration library, and application configuration, etc.

Jerry Chou and Chan-Yu Mou, as the team advisor and student coach respectively, jointly guided the team to complete this study and writing. All authors approve of the content of the manuscript and agree to be held accountable for the work.

The performance results have to be collected from the two supercomputers provided by the competition: GADI [3] at the National Computational Infrastructure(NCI) in Australia, and ASPIRE-2A [4] at the National Supercomputing Centre (NSCC) in Singapore. But in order to perform more in-depth performance studies and collect more experimental results, we also conducted experiments on two supercomputers in Taiwan, naming Taiwania-2 [5] and Taiwania-3 [5]. By comparing the results from different supercomputer systems, we can also have more understanding on how the hardware specifications and system environments could affect the performance improvement of different optimization techniques.

In conclusion, we achieved a performance speedup of 1.34 times for MPAS and 11.4 times for Bloom using the following performance tuning and optimization techniques.

- Software Environment Configuration: The software environment configuration encompasses the selection of versions and the building configuration of the compilers, libraries and applications themselves.
- Leveraging GPU Shared Memory: Maximizing Shared Memory Utilization for Direct Inference Acceleration.
- Load Balancing Optimization: Achieving Effective Workload Distribution among nodes.

The rest of the paper is structured as follows. Section II describes the system environments and hardware specifications of the supercomputers used in our study. Section III introduces the applications and performance objectives for our optimization. Section IV and Section V are the result in which we achieved.

II. Experiment Environment

We use four supercomputers as the computing environments for our performance study, naming the GADI system at the National Computational Infrastructure(NCI) in Australia, the ASPIRE-2A system at the National Supercomputing Centre (NSCC) in Singapore, and the Taiwania-2 and Taiwania-3 systems at the National Center for High-performance Computing (NCHC) in Taiwan. GADI and ASPIRE-2A are the designated supercomputers for the competition. Taiwania-2 and Taiwania-3 are the additional supercomputers we used for conducting more experiment runs and collecting more in-depth performance measurement results for tuning and analysis.

TABLE I
THE HARDWARE CONFIGURATION OF THE SUPERCOMPUTERS USED IN OUR PERFORMANCE STUDY.

Type	GADI	ASPIRE-2A	Taiwania-2	Taiwania-3
CPU model	Intel Xeon Platinum 8274	CPU AMD EPYC 7713	Intel Xeon Gold 6154	Intel Xeon Platinum 8280
CPU architecture	Cascade Lake	Millan	Skylake	Cascade Lake
CPU frequency	3.2 GHz	2 GHz	3.0 GHz	2.4 GHz
Cores per CPU	24	32	18	28
Numbers of CPU per node	2	2	2	2
RAM per node	192GB	512 GB	192GB	192GB
GPUs per node	4 × NVIDIA V100	4 × NVIDIA A100	8 × NVIDIA V100	Not Available
Network	200 Gb/s	100 Gb/s	100 Gb/s	100 Gb/s
Operating System	Rocky Linux	RHEL8	CentOS	CentOS
Tested Application	Bloom, MPAS	Bloom	Bloom	MPAS

Table I summarizes the hardware specifications of the computing systems. It is clear that ASPIRE-2A has the strongest computing power equipped with 4 A100 GPUs and 64 cores per node. On the other hand, GADI has the highest interconnect network bandwidth at 200 Gb/s, while the other systems only have 100 GB/s network bandwidth. Taiwania-2 has a old CPU mode, but it has the most number of GPUs per node at 8. Taiwania-3 has no GPU, so we can only run MPAS on it, while Bloom was run on GADI, ASPIRE-2A and Taiwania-3. By conducting experiments across multiple systems, we are able to verify our experiment results in different computing environments, and observe the performance impact from hardware specifications and resource configurations.

For the system environment, we installed the same software on different supercomputers for each application. According to the competition rule, we are allowed to select for the choice of MPAS version among the four available versions 5, 6, 7, and 8. We opted for the seventh and eighth version, which stand out as the latest version and most stable iteration matches the benchmark data among the four, and we could choose the version of supporting libraries and compilers for optimizing the application performance as well. On the other hand, there are three versions of the Bloom models. The first is the original BLOOM-176B model trained by Bigscience. Then, there are two models quantized by the Microsoft research team: bloom-deepspeed-inference-int8 and bloom-deepspeed-inference-fp16. Ultimately, we decided to use bloom-deepspeed-inference-int8 over the others because it offers the best combination of performance and feasibility on the Gadi supercomputer.

Table II and Table III lists the required software installed for achieving the best performance for MPAS and Bloom, respectively. But, we also installed other versions of compilers, MPI libraries and acceleration packages for performance comparison.

TABLE II
THE SOFTWARE INSTALLED FOR MPAS.

Libraries	Versions
C Compiler	ICC 2021.10.0
MPI Library	OpenMPI 4.1.5
HDF5	1.8.18
PNETCDF	1.12.3
NETCDF C	4.8.1
NETCDF Fortran	4.6.1
Parallel I/O	2.5.10
Metis	5.1.0

TABLE III
THE SOFTWARE INSTALLED FOR BLOOM.

Libraries	Versions
C Compiler	GCC 10.3.0
GPU Compiler	CUDA nvcc 11.6
python	3.8
NCCL	2.18
Pytorch	1.12.1
Transformer	4.26.1
Deepspeed	0.7.6
accelerate	0.16.0
gunicorn	20.1.0
uvicorn	0.19.0
jinja2	3.1.2
pydantic	1.10.2
huggingface hub	0.12.1
grpcio-tools	1.50.0
bitsandbytes	0.41.1

III. DESCRIPTION OF APPLICATIONS

This section briefly introduces the applications designated by the competition, and the corresponding rules for running them during the competition.

A. MPAS-Atmosphere

For the competition, we were asked to run the atmospheric component of MPAS, which is a climate model designed to simulate and study the behavior of the Earth's atmosphere. The simulation is based on unstructured centroidal Voronoi (hexagonal) meshes using C-grid staggering and selective grid refinement. This unstructured voronoi grid is good for scaling on parallel computers. The task is to run the 10km benchmark provided from MPAS with `config_dt` (per step size) set to 60 and `config_run_duration` set to 16 minutes. The computing task in the competition is to run the application on 32 nodes with 48 CPUs core per node, and versions of MPAS we used is 7.3 and 8.0.1.

B. Bloom

The other application designated by the competition is BLOOM, which is a multilingual language model with 176 billion parameters, making it one of the world's largest

TABLE IV
COMPARISON OF SIMULATION SPEEDS FOR DIFFERENT COMPILER COMBINATIONS ON THE 32 NODES MPAS TESTCASE.

Combinations	Simulation Speed
GCC OMPI	20.34
GCC IMPI	20.48
ICC OMPI	25.7474
ICC IMPI	23.5069

open multilingual language model. It has the capability to generate text in 46 natural languages and 13 programming languages.The task is to run an int8 model type with deepspeed inference scripts to optimize throughput performance,aiming for the lowest throughput per token, including tokenization time (measured in milliseconds). The computing task in the competition is to run the Bloom inference requests on 2 nodes with 4 GPUs per node and BLOOM-176B model is downloaded from huggingface hub.

IV. MPAS-ATMOSPHERE RESULT

In this section, we present our successful optimization strategies and achievements. We explore the foundational layer of software installation and its dynamic performance optimizations.

In our study, a key metric for evaluation is the simulation speed, a parameter dedicated to scoring. We focus our optimization efforts on enhancing this simulation speed and present the outcomes accordingly.

$$\text{Simulation Speed} = \frac{\text{Computing Time}}{\text{Simulation Time Integration}}$$

The computing time represents the duration MPAS spends in simulation, excluding the time for data preparation tasks such as partitioning or I/O. The simulation time integration refers to the actual time span simulated which can be configured using `config_run_duration`. In this competition, it is set to 4 minutes for the 8-node testcase, 8 minutes for the 16-node testcase, and 16 minutes for the 32-node testcase.

A. Software Installation & Configuration

1) C and MPI Compiler selection: For our first approach, we've tested out different C and MPI compiler combination. The result shows that ICC 2021.10.0 with OpenMPI 4.1.5 is the best suite for MPAS, which outperformed other combinations when running MPAS on the 32 nodes. Additionally, performance testing was also conducted on the Taiwania 3. Fig. 1 illustrates a performance comparison between the two clusters. The network speed difference shown in Table. I is the main reason Gadi faster than Taiwania 3.

2) OpenMPI and UCX Configuration: Next, we focused on optimizing both UCX (Unified Communication X) [6] and OpenMPI to enhance performance. UCX exposes a set of abstract communication primitives that utilize the best of available hardware resources and offloads. MPI libraries adopt UCX as their underlying communication protocol.

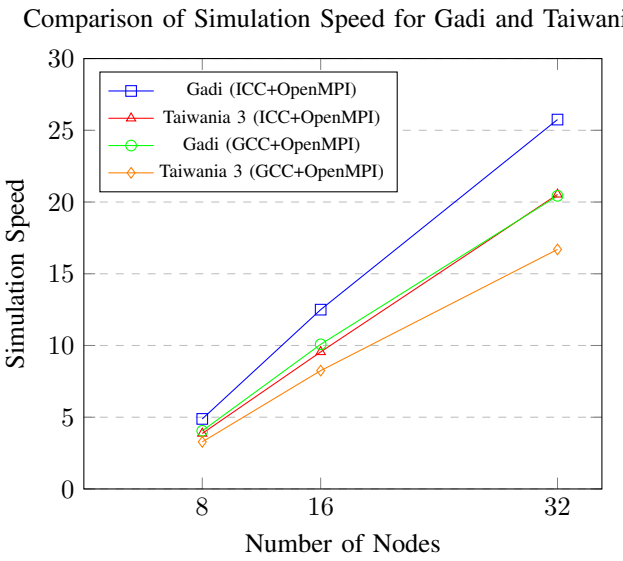

Fig. 1. Comparison of simulation speed between Gadi and Taiwania 3 with ICC and GCC compilers and OpenMPI on testcases with different nodes.

TABLE V
METIS CONFIGURATION COMPARISON

Type	Default	Optimize
Objtype	Edge Cuts (cut)	Communication (vol)
Edge cuts	350158	347823
Communication volume	354760	352425
Balance	1.015	1.004

For UCX optimization, we deactivated all debugging flags to improve performance, and enhanced it further by enabling UD (Unreliable Datagram) and RC (Reliable Connection) support. We also integrated KNEM for shared memory-based, high-performance intra-node MPI communication.

For OpenMPI 4.1.5 optimization, we enable HCOLL [7] for optimized collective communication and romio for advanced I/O handling in MPI. Additionally, we enabled Lustre support for MPI, coupled with Lustre + UFS for ROMIO filesystem types [8]. We also activated sparse group support and selected the `mellanox/optimized` configuration by setting the `--with-platform` flag.

3) Compile Time Flag: We utilize the `-march` flag for CPU-specific optimizations and the `-ax` flag to create code versions for multiple architectures during compilation. Consequently, these two flags gave reduce the overall integration time by 4 seconds which is about 10% performance improvement.

B. Dynamic Performance Optimizations

1) Graph Load-Balancing: Fig. 2 is our profiling result from IPM, in this figure we found that the load-balancing didn't seem well on MPAS. It can be cause by the way Metis [9] partition the mesh.

After researching, we change the load balancing tolerance in Metis and select `objtype` (the mesh optimization mode) to minimize edge cuts and communication time.

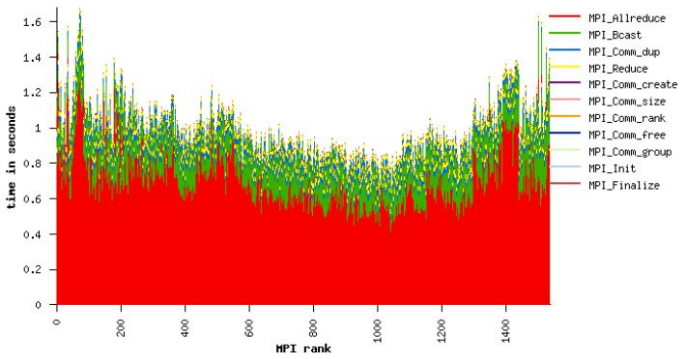

Fig. 2. Load balancing of MPI functions according to simulation time (without I/O)

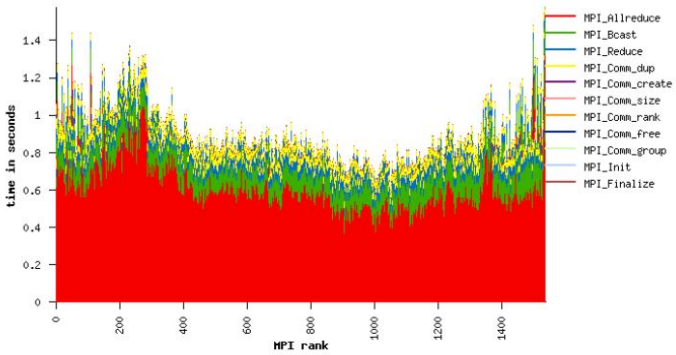

Fig. 3. Load balancing of MPI functions according to simulation time (without I/O) after our approach

After all, the balancing issue was improved. As shown in Fig. 3, the peak has dropped from 1.6 to 1.4 with a similar lower bound. We believe the load balancing issue may have even larger impact in some longer or larger simulations.

2) I/O optimization: Due to the longer integration step occurring every simulated 10 minutes, we suspect that there may have I/O operations causing delays. Thus, we've tried different I/O optimization attempts to solve this issue. One of the most effective method is to set the ROMIO [8] environment variables to `romio_cb_read` or `romio_cb_write`, which let us enable or disable the usage of collecting buffering. This configuration resulted in an approximated 50% reduction in I/O time.

C. Final results on Gadi and Taiwania 3

After all the approaches were done, the best results is show in Fig. 4 with a baseline.

V. Bloom Result

The task of this application is to minimize the value of metric "generation time per token" under the framework of Pytorch and Deepspeed. To simplify the meaning, we can describe it as :

$$\Phi_{token} = \frac{t_{gen} + t_{tokenize}}{n_{gen}}$$

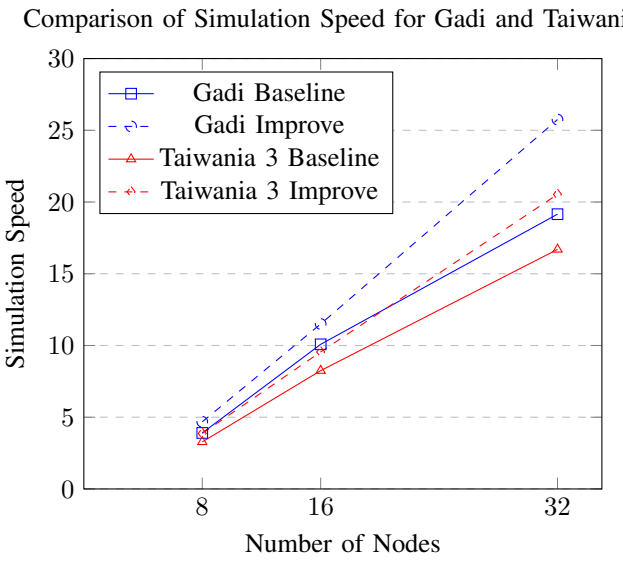

Fig. 4. Simulation speed comparison between baseline and final result in Gadi and Taiwania 3.

TABLE VI
THE INPUT SENTENCES FOR BLOOM INFERENCE

Input_sentences
"DeepSpeed is a machine learning framework"
"He is working on"
"He got all"
"Everyone is happy and I can"
"The new movie that got Oscar this year"
"In the far far distance from our galaxy,"
"Peace is the only way"

In this formula, t_{gen} denotes the generation time required, $t_{tokenize}$ signfies the tokenizing time required and n_{gen} represents the number of new token generated. Since the $t_{tokenize}$ is far smaller than t_{gen}, we can rewrite the equation as :

$$\Phi_{token} = \frac{t_{gen}}{n_{gen}}$$

To minimize the metric value for the BLOOM inference process, we can enhance the token inlet to generate more tokens, reduce the execution time, or implement both strategies.

A. Batch size

Our first strategy is to increase the total token inlet. By using Pytorch deep learning framework, we can easily transfer data from hosts to devices in pursue of better parallel ability.

Table VI lists all sentences that will be fed as the input tokens. We found that the sentences will be duplicated several times in the list until the content in the list exceeds the batch_size. Then, we will extract the first batch_size sentences and feed them into the language model.

By experiments, we have found that the metric value decreases as we increasing the batch_size as shown in Fig. 5.

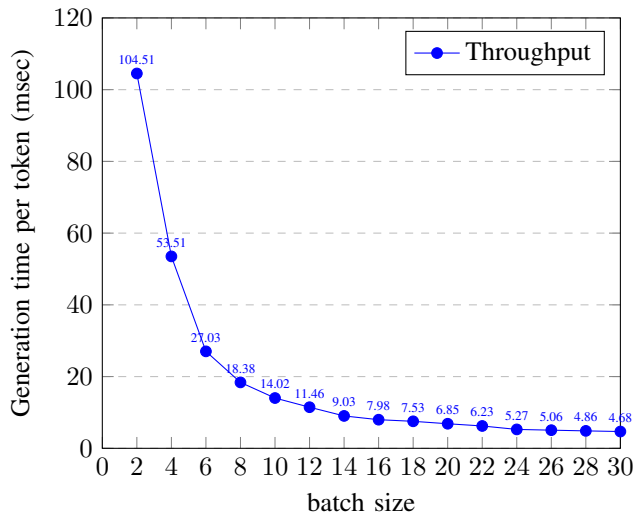

Fig. 5. Generation time per token to batch size in two ASPIRE-2A A100x4 nodes

B. Deepspeed Optimization

Here comes another strategy. The framework we choose for Bloom is Deepspeed. Developed by Microsoft, Deepspeed is an open-source deep learning optimization library specifically crafted to enhance the training efficiency and scalability of large deep learning models.

Deepspeed leverages tensor parallelism, a technique that distributes tensors across multiple GPUs. This method effectively breaks down a large model into numerous smaller segments, allocating them across various GPUs. Such a strategy enables models, which would typically exceed the capacity of a single GPU's memory, to be accommodated within multiple GPUs.

Moreover, as each GPU is assigned a segment of the overall computation, all GPUs are engaged in processing activities concurrently. Once they complete their individual computations, the GPUs exchange results through intercommunication, facilitating their progression to the next computational layer. By this approach, all the GPUs can work simultaneously, resulting in a better overall performance.

C. NCCL Optimization

Communication is another bottleneck of the execution time, here are some NCCL [10] environment variables that we explored the best NCCL environment variables for running deepspeed inference on the Gadi supercomputer.

1) NCCL_NET_GDR_READ: We set this variable to 1 (default 0 in non-NVlink based platform) to use GPU Direct RDMA to send data to the NIC directly.

2) NCCL_IB_QPR_PER_CONNECTION: We set this variable to "SYS" to always enable GPU Direct RDMA across nodes.

3) NCCL_NTHREADS: By default, V100 on Gadi use 256 threads only. We set this variable to 512 to enlarge the CUDA threads per CUDA blocks. Increasing this variable can increase the GPU clock rate.

4) NCCL_BUFFSIZE: This variable control the size of buffer used by NCCL when communicating between pairs of GPUs. By experiment, we have found that setting buffer size to 8Mb gives the best performance on Gadi.

D. Comparison between Gadi and Taiwania2

As depicted in Fig. 6., despite of the batch size, Gadi consistently outperforms Taiwania 2. The speedup achieved on Gadi is more than 2 times faster.

We conducted Nsight System profiling, and observed that during the inference process, a significant portion of GPU kernel usage is dedicated to the ncclKernel allreduce operation, which is responsible for combining results. This highlights the communication demands between GPUs when employing Tensor Parallelism.

NCCL tests were conducted to evaluate the interconnection capabilities of both Gadi and Taiwania 2. The results indicate that Gadi has an impressive interconnection bandwidth of approximately 130 gigabytes per second, while Taiwania 2 has only about 12 gigabytes per second. This difference in interconnection bandwidth could indeed be one of the reasons why Gadi has a better performance.

This insight highlights the importance of communication in large model inference.

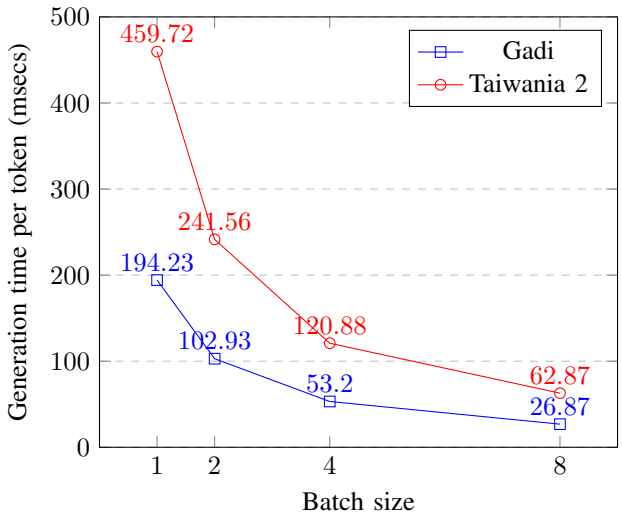

Fig. 6. Comparison of generation time per token for Gadi and Taiwania 2 across different batch sizes.

E. Unsuccessful Attempts

We've also tried to use some optimization tools provided by NVIDIA, but failed to achieve any significant improvement in our results, mostly due to memory issue.

Compared to deepspeed, which was the only method we successfully ran BLOOM-176B inference on Gadi and ASPIRE-2A, we concluded that it's because deepspeed provides one optimization – Mixture of Quantization, which loads INT8 data and converts it to FP16 later when needed. This helps reduce the model size.

The following summarizes the optimization tools we've tried, why we tried, and why it failed.

1) FasterTransformer: [11] We've tried FasterTransformer since it claims that it provides some optimizations such as layer fusion, caching, support of multi-node, matmul kernel auto-tunning, etc. However, it didn't have directly support of INT8 quantization for our BLOOM-176B model, which we thought would be unlikely to succeed. Therefore we turned to other tools after successfully building the required image.

2) TensorRT: [12] We've also tried TensorRT, which we thought can make use of the TensortRT inference engine to improve the performance. It first requires to convert BLOOM to ONNX format. With the former conclusion, any method without INT8 quantization might fail. Unfortunately, We are not allow to use other model (since the INT8 model provided seems to be only for deepspeed. Hence the only way we can do is to convert BLOOM-176B to ONNX first then do quantization. However, BLOOM-176B is such a large model that it's lots of work for us to transform the entire model. We make a pause after converting the first layer of BLOOM-176B.

3) TensorRTLLM: [13] Our last attempt was TensorRTLLM, which was just released recently. We thought it was the most suitable tool for us to use as its name showed. Not only it incorporates the desirable optimizations, such as kernel fusion, Tensor engine, ... some formally provided by FasterTransformer and TensorRT, it also provides python API and direct support of BLOOM-176B. Though finding such powerful tool, we still fail to build TensorRT engine due to memory issue, as we encountered when using TensorRT.

To sum up, almost all the attempts failed due to insufficient memory, which is unsolvable. Therefore, although they provide many desirable optimizations, these tools are not feasible in our cases.

VI. Conclusion

This paper has detailed our participation in the 2023 High-Performance Computing and Artificial Intelligence (HPC-AI) Asia-Pacific Competition, a challenging and enriching experience that tested our skills and knowledge in the realms of high-performance computing and AI. Our team's efforts were focused on two main challenges: optimizing the Climate Prediction Model MPAS to enhance simulation speed on CPUs, and accelerating the Large Language Model Bloom to minimize generation time per token on GPUs.

Our experiments, conducted on four different supercomputer systems, including GADI in Australia and ASPIRE-2A in Singapore, as well as Taiwania-2 and Taiwania-3 in Taiwan, allowed us to gather comprehensive data and compare results across different platforms.

In MPAS, we achieved a speedup of 1.34 times. Given the constraint of being unable to modify the source code, we optimized all aspects of the software environment and load balancing. While the numerical reduction in simulation time may not be a lot, we believe that these configurations will yield more substantial benefits when applied to simulations with larger time scales.

On the other hand, we significantly improved the performance of the quantized bloom-deepspeed-inference-int8 model, making its inference process 11.4 times faster than the original model's batch inference on the Gadi supercomputer. We observed that the primary bottleneck is the limited GPU RAM. As we are unable to alter the model or the inference framework, we believe that using A100 GPUs and finding better methods to achieve higher parallelism will help us overcome more challenges.

Finally, our participation in the 2023 HPC-AI competition has been a profound learning experience, providing us with the opportunity to apply our skills in a real-world context.

References

[1] Los Alamos National Security and the University Corporation for Atmospheric Research (UCAR), "The model for prediction across scales," 2023. [Online]. Available: https://github.com/MPAS-Dev/MPAS-Model

[2] jeffra, "microsoft/bloom-deepspeed-inference-int8," 2022. [Online]. Available: https://huggingface.co/microsoft/bloom-deepspeed-inference-int8

[3] National Computational Infrastructure (NCI), "Hpc systems - nci," 2020. [Online]. Available: https://nci.org.au/our-systems/hpc-systems

[4] National Supercomputing Centre Singapore (NSCC), "Aspier 2a," 2022. [Online]. Available: https://www.nscc.sg/aspire-2a/

[5] National Center for High-Performance Computing (NCHC), "Supercomputer," 2017. [Online]. Available: https://www.nchc.org.tw/Page?itemid=58&mid=109

[6] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss, Y. Shahar, S. Potluri, D. Rossetti, D. Becker, D. Poole, C. Lamb, S. Kumar, C. Stunkel, G. Bosilca, and A. Bouteiller, "Ucx: An open source framework for hpc network apis and beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, 2015, pp. 40–43.

[7] NVIDIA, "Hcoll - nvidia docs," 2023. [Online]. Available: https://docs.nvidia.com/networking/display/hpcxv216/hcoll

[8] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective i/o in romio," pp. 182–189, Feb 1999.

[9] *METIS 5.0 Manual*, Online, https://www.lrz.de/services/software/mathematik/metis/metis_5_0.pdf.

[10] S. Jeaugey, "Nccl 2.0," in *GPU Technology Conference (GTC)*, vol. 2, 2017.

[11] NVIDIA, "Fastertransformer," 2023. [Online]. Available: https://github.com/NVIDIA/FasterTransformer.git

[12] NVIDIA, "Nvidia tensorrt - developer guide," 2023. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/pdf/TensorRT-Developer-Guide.pdf

[13] ——, "Tensorrt-llm's documentation," 2023. [Online]. Available: https://nvidia.github.io/TensorRT-LLM/