Performance Analysis and Benchmarking of a Temperature Downscaling Deep Learning Model

Karthick Panner Selvam
SEDAN, SnT
University of Luxembourg
Luxembourg
karthick.pannerselvam@uni.lu

Mats Brorsson
SEDAN, SnT
University of Luxembourg
Luxembourg
mats.brorsson@uni.lu

***Abstract*—We are presenting here a detailed analysis and performance characterization of a statistical temperature downscaling application used in the MAELSTROM EuroHPC project. This application uses a deep learning methodology to convert low-resolution atmospheric temperature states into high-resolution. We have performed in-depth profiling and roofline analysis at different levels (Operators, Training, Distributed Training, Inference) of the downscaling model on different hardware architectures (Nvidia V100 & A100 GPUs). Finally, we compare the training and inference cost of the downscaling model with various cloud providers. Our results identify the model bottlenecks which can be used to enhance the model architecture and determine hardware configuration for efficiently utilizing the HPC. Furthermore, we provide a comprehensive methodology for in-depth profiling and benchmarking of the deep learning models.**

***Index Terms*—Performance Analysis, Roofline Model, Weather Forecast, Deep Learning Benchmark**

I. Introduction

The use of AI and machine learning involves a considerable amount of computing resources in training the models and, to a lesser degree, in production when the models are used. It is essential to understand the computational needs of a model in training and inference to i) give it enough resources and ii) not give it more than needed so that resources are not wasted. In this paper, we present a thorough performance analysis and characterization of the MAELSTROM Temperature Downscaling (MTD) application which is a post-processing methodology to convert low-resolution atmospheric grid space into high-resolution grid space using a Convolution Neural Network (CNN). The statistical downscaling technique used in MTD is highly inspired by CNN-based image super-resolution because it takes grid-based input, learns spatial-temporal patterns from trained data, and converts the low-resolution atmospheric states to a high-resolution [1], [2]. The MTD application uses a U-Net [3] architecture to enhance the spatial resolution of atmospheric T2M (2 meters above surface air temperature) [4]–[6]. The approach we take to analyze and characterize the downscaling application is to combine modular-level deep learning benchmarking [7] with roofline analysis [8], where we study i) the operators that make up the model, ii) the inference network, and iii) the training of the network. We specifically concentrate on the convolutional operators of the U-net model architecture. These are studied with the help of roofline graphs to understand how close the application performance is to the architecture's empirical limits. The platforms we have chosen to study the application are Nvidia A100 and V100 GPUs. Improving the accuracy of the MTD model is not part of our research.

The contributions of our work include the following:

- An analysis of the main computational components of the MTD application and their computational need,
- Roofline characterizations of the convolutional operators of the MTD application on A100 and V100 GPUs,
- Benchmarking of inference and training performance on A100 and V100 and provided methodology to effectively to utilise A100 GPU for MTD model inference
- A cost-analysis on some of the major cloud providers,
- A methodology to capture performance data for the above-mentioned performance measurements.

To identify the bottleneck of convolution operators, we used roofline analysis by varying convolution operator parameters such as kernel, strides, and filters. Our benchmark and profiling techniques shows tensor core utilization across multiple GPUs and energy utilization for training. Using our performance metrics results, developers can enhance the MTD model architecture and adjust the hardware configuration to utilize the underlying hardware effectively. For example, fixing the batch size, selecting the optimal number of GPUs for training, and finding cost-effective cloud instances for inference and training.

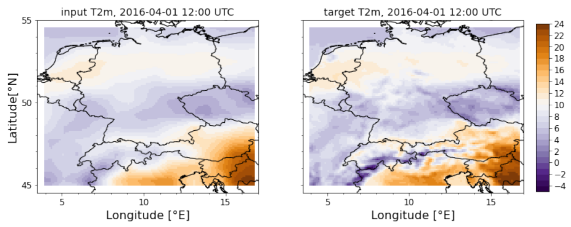

input T2m, 2016-04-01 12:00 UTC
target T2m, 2016-04-01 12:00 UTC
Latitude[°N]
55
50
45
5
10
15
Longitude [°E]
24
22
20
18
16
14
12
10
8
6
4
2
0
−2
−4

Fig. 1: MAELSTROM Temperature Downscaling Model Input and Target Output of T2m (2 meters above surface air temperature) values plotted using cartopy-matlplotlib of Grid Space [96 x 128] in the zonal and meridian directions.

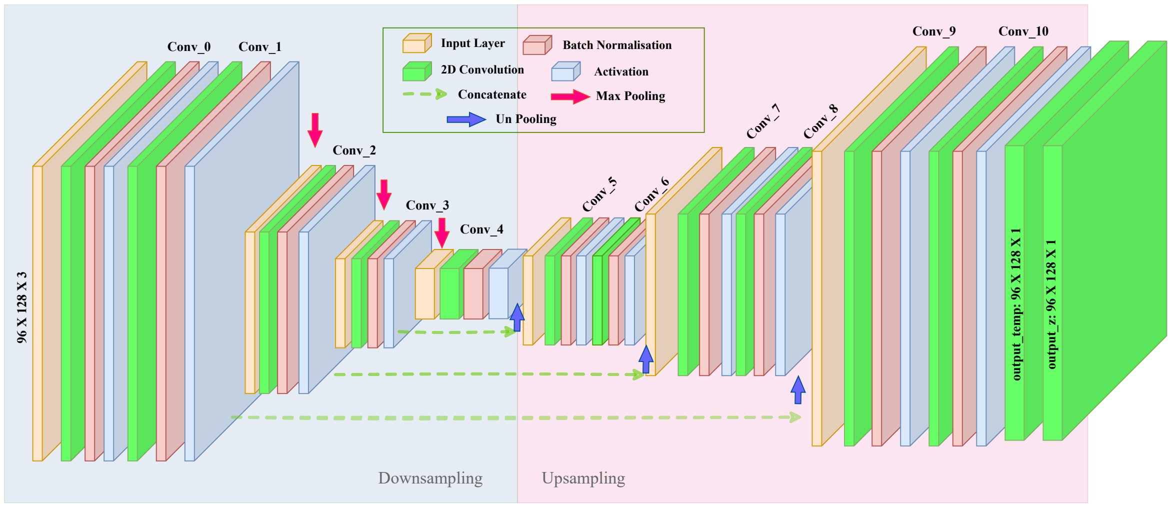


Fig. 2: MAELSTROM Temperature Downscaling Model Architecture

.

II. Background

A. The MAELSTROM Temperature Downscaling Application

The MTD model uses a U-Net architecture to convert low-resolution grid space atmospheric 2m temperature states into a high resolution [3]. For training the MTD model, we used data provided by the European Centre for Medium-Range Weather Forecasts (ECMWF) in the form of the HRES[1] dataset using the Bi-linear interpolation technique. The target T2M (2 meters above surface air temperature) is the direct output of the ECMWF HRES dataset; that is why input T2M looks like smooth temperature fields, whereas target T2M is sharpened because the original data is captured in complex terrain, as shown in Figure 1.

The MTD model architecture is illustrated in Figure 2. The most compute-intensive parts are the 2D convolution layers (Conv2D). The MTD model accepts input of $96 \times 128 \times 3$ grid space, data variables are (Low-resolution T2M, elevation and High-resolution elevation) and generates two outputs, each with the shape of $96 \times 128 \times 1$ (High-resolution T2M and elevation). MTD's model contains a total of 3525650 trainable parameters and 3360 non-trainable parameters with 7.5GiB training data and 0.95 GiB of validation data. The current dataset contains only the central part of Europe. More coarsened HRES data across Europe on different periods will be added to improve the precision, so the MTD models will require retraining with new data to increase the accuracy of enhancing the spatial resolution of T2M.

B. Hardware aspects

We have chosen to base most of our analysis on measurements on the Nvidia V100 and A100 GPUs. These are computational engines built specifically for AI and machine learning applications. Table I summarizes the main characteristics of these platforms. Both A100 and V100 use a Tensor Cores (TC) to accelerate the computing of matrix-matrix multiplication, and it is widely used in deep learning model training. A100 TC supports single precision (FP32), half-precision (FP16), BFLOAT16 and INT8. Whereas V100 only supports half precision tensor core.

[1]https://www.ecmwf.int/en/forecasts/datasets/set-i

TABLE I: Nvidia V100 and A100 main specifications.

	V100	A100
FP64	7.8 TFlop/s	9.7 TFlop/s
FP32	15.7 TFlop/s	19.5 TFlop/s
Tensor Cores	125 TFlop/s	312 TFlop/s
GPU Memory	16 GB	40 GB

C. Benchmarking Deep Learning Models

Various deep learning benchmark suites are available like MLPerf [9], Dawnbench [10], and Deepbench [11] [12]. Most suites support high-level profiling aspects, like wallclock performance metric, accuracy, training, and validation loss. Ben-Nun et al. [7]. proposed a modular level approach to benchmark a deep learning model on various levels, including distributed training, but it lacks to identify the bottlenecks of operators and cloud costs. However, low-level profiling is vital to identify the bottlenecks of model architecture efficiency. Researchers used roofline analysis to identify the bottlenecks of deep learning applications, but it is limited to the operator-level investigation [13], [14]. Yang et al. [8] proposed hierarchical roofline analysis to identify bottlenecks in convolution operator and model training; however, this methodology lacks to extend it to distributed training.

We propose a comprehensive and in-depth profiling methodology for benchmarking and performance analysis of deep learning models to fill the gap in the previous research work. We combine the modular level approach of deep learning benchmark and roofline technique [7], [8] with cost analysis and dissect the model into five levels: Operator, Training, Distributed training, Inference, and Cloud cost comparison.

We performed deep profiling at each level and identified the performance bottlenecks of the model for future enhancement. Furthermore, we provided a methodology to use A100 GPU for inference effectively.

III. Methodology

We separated the experiments into multiple levels: Operator, Training, Distributed training, Inference, and cloud compute cost comparison. For the operator level, we conduct the experiment on the MTD model's convolution operators. We perform roofline analysis on NVIDIA V100 and A100 GPUs for Conv2D operators by the varying batch size, strides, kernel, and filters to identify the bottlenecks of the operators. Using the knowledge gained from the operator level, we train the MTD model with two precision modes, single precision, and mixed precision, on single V100 and A100 GPU, and we measure Tensor core, GPU utilization, GPU power, memory usage, average epoch time, training and validation loss. Then, we use HPC clusters for distributed training to determine whether increasing GPUs reduces the training time for the MTD model. For each node, we increase GPU gradually from 1 to 4 for training. We used TensorFlow mirrored strategy API for distributed training because it synchronous training across multiple GPUs on a single node.

At the inference level, we calculate the one full forward propagation time of the MTD model trained with single and mixed precision. Then we used the DLProf[2] tool to extract the TC utilization, GPU utilization, and wallclock time for inference. We used roofline analysis to find the computational bottleneck of the kernel. Then we compare inference with V100 and A100 GPUs, and we suggest a methodology to use NVIDIA Multi-instance GPU (MIG[3]) technology for efficient inference on A100 GPU. Finally, we compare training and inference cost with various cloud vendors.

A. Systems used

1) Hardware: We used two HPC systems for our research: JUWELS[4] Booster and JUWELS cluster. Juwels Booster contains 936 nodes, and each node contains 4 x A100 Nvidia GPUs with 40 GB HBM connected via NVLink3 with AMD EPYC 7402 processor and 512GB memory. Juwels cluster contains 56 computing nodes, each containing 4× NVIDIA V100 GPU and 16 GB HBM with 2× Intel Xeon Gold 6148 processor. We used single-node JUWELS Booster and JUWELS Cluster extensively across all levels.

2) Software framework: Throughout the experiments, we used TensorFlow version 2, Python 3.9.6, CUDA 11.6, and cuDNN v8.3.1. We used the following precision modes on all levels: single precision and half-precision for operator levels and mixed precision (It enables TC on GPU to utilize FP32 and FP16 for accelerated computing) for training level.

[2]https://docs.nvidia.com/deeplearning/frameworks/dlprof-user-guide/
[3]https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html
[4]https://www.fz-juelich.de/en/ias/jsc/systems/supercomputers/juwels

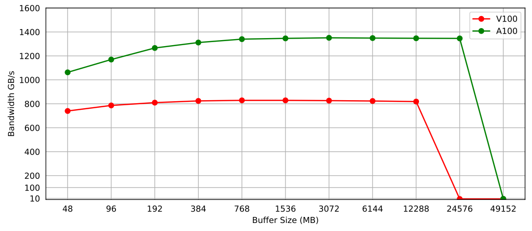


Fig. 3: High Bandwidth Memory (HBM) Peak Throughput of V100 and A100 GPUs

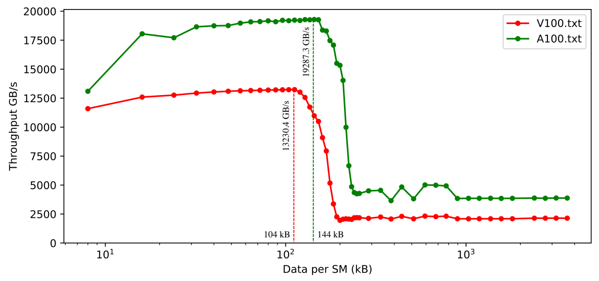


Fig. 4: L1 and L2 cache Peak Throughput of V100 and A100 GPUs.

B. Roofline characterization

The roofline model is a visually-intuitive way to understand the kernel performance and identify the bottlenecks of kernel execution on a given machine [8]. We perform roofline analysis on V100 and A100 GPUs of Conv2D operators because they are the most compute-intensive operations in the MTD application. The roofline model is an intuitive model based on bounds of data transfer and computes capacity and can be expressed as in equation 1. Where P is the achievable performance (GFLOP/s), π is the peak performance (GFLOP/s), β is the peak bandwidth (GB/s), and AI is the arithmetic intensity expressed as floating point operations by data movement (FLOPs per Bytes). To plot the roofline model for MTD Conv2D operators on GPUs, we need two components:

- *Machine Characterization*: In terms of peak GFLOP/s for (single precision, half-precision, and TC) and Peak bandwidth (GB/s) of L1 cache, L2 cache, and High Bandwidth Memory (HBM).
- *Application Characterization*: Conv2D kernel Arithmetic Intensity (AI) and Conv2D kernel throughput GFLOP/s of (single precision, half-precision, and TC).

$$\mathrm{P} \leq \min \begin{cases} \pi \\ \beta \times AI \end{cases} \tag{1}$$

1) Machine Characterization: We used the Empirical Roofline Tool [8] (ERT) and Cuda-Bench [15] tool to collect machine characterization metrics empirically. We collected single and half-precision values for GPUs using ERT. V100 peak single precision is 15.5 TFLOP/s, and half-precision is 29.6 TFLOP/s. A100 peak single precision is

18.5 TFLOP/s, and half-precision is 58.8 TFLOP/s. We used Cutlass GEMMs (General Matrix Multiplications) to calculate Peak TC TFLOP/s. GEMMs are defined in Equation 2.

$$D = \alpha \mathrm{AB} + \beta C \tag{2}$$

where A and B are input matrices, and C is already existing matrix and overwritten by output matrix D. α and β are scalar constants. For matrix A is M x K, B is M x N and C and D are N x K matrix. We used $M = N = K = 16384$ matrix to get the peak TC results for V100 is 101.2 TFLOP/s, and A100 is 292.2 TFLOP/s. To calculate the Peak HBM throughput, we used a stream-kernel technique from Cuda-Bench by increasing the buffer size from 48 MB to 49152 MB, which guarantees to capture the device memory within. The V100 we used has a 16GB memory limit, whereas the A100 has a 40 GB memory limit. As a result, V100 can reach a peak HBM bandwidth of 818.5 GB/s on a buffer size of 12288 MB, and A100 reaches a peak HBM bandwidth of 1346 GB/s on 24576 MB, as shown in Figure 3.

To read the L1 and L2 cache bandwidth, we launch one thread block per Stream Multiprocessor (SM). Every thread block reads the same buffer continuously, and we vary the buffer size to measure peak L1 and L2 bandwidth, as shown in Figure 4. As a result, V100 L1 bandwidth is 13230.4 GB/s, and L2 bandwidth is 2142.8 GB/s. And A100 L1 bandwidth is 19287.3 GB/s and L2 is 3877.6 GB/s. We use these values to plot the roofline model of L1, L2, HBM, Single and half-precision, and Tensor Core ceilings as shown in Figure 5

2) Application Characterization: We use the Nsight[5] Compute CLI tool to collect the kernel performance and bandwidth metrics for roofline analysis. We use PyCUDA[6] marker API to specify the specific region of the code to extract the metrics. We used the following command to extract kernel metrics.

*ncu -profile-from-start off –metrics **[metrics]** python app.py*

We used Equation (3) to calculate the kernel execution time K_t from the ncu output. We calculate single precision and half-precision and TC FLOPs using Equation (4), (5), and (6), respectively. We calculate L1 and L2 cache from this metrics $\{l1tex, lts\}\_\_t\_bytes.sum$ and device memory from this metrics $dram\_\_bytes.sum$

$$K_t = \frac{elapsed\_avg}{avg\_per\_second} \tag{3}$$

$$FP32\ FLOPs = \_fadd\_ + (2 * \_ffma\_) + \_fmul\_ \tag{4}$$

$$FP16\ FLOPs = \_hadd\_ + (2 * \_hfma\_) + \_hmul\_ \tag{5}$$

$$TC\ FLOPs = \_pipe\_tensor.sum * 512 \tag{6}$$

[5]https://docs.nvidia.com/nsight-compute/NsightComputeCli/index.html
[6]https://documen.tician.de/pycuda/

IV. Experiments and Results

A. Operator Level

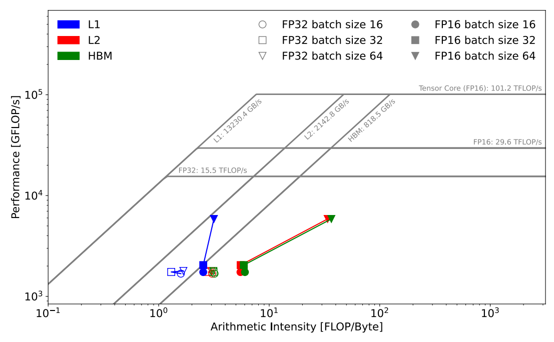
We use roofline analysis to identify the computational bottlenecks and data movement of different level cache (L1 and L2) and HBM of convolution operators. We focused only on the computational bottleneck of convolution operators rather than the measuring accuracy of the model while varying operator configuration. We use V100 and A100 GPUs for roofline analysis. First, we collected the GPU kernel metrics using the nsight-compute-cli tool and used PyCUDA marker API to specify the code region to extract metrics. Then, we pre-process the metrics to get the Arithmetic intensity of L1, L2, HBM, and performance GFLOP/s as mentioned in Section III-B2 and plot these results in the roofline chart as shown in Figure 5.

In the roofline chart, the horizontal ceiling represents peak GFLOP/s of Tensor Core, single precision, and half-precision. The color blue, red, and green represents kernel associated with L1, L2 cache, and HBM. If the kernel value is near the ceilings of (TC, FP32, and FP16) of the roofline chart, it shows the kernel has higher performance. Suppose the kernel values of blue, red, and green colors near the respective diagonal ceilings (L1, L2, and HBM) show good data locality. If any color overlaps, it shows poor data locality.

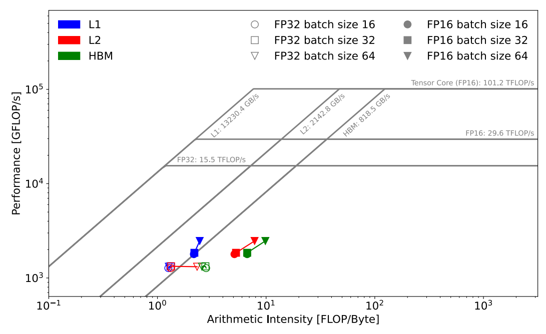
For this experiment, we used the MTD model's highest FLOP operator, which is Conv2D:(tensor shape: 128x96x112, batch size:32, filters: 56, kernel: 3x3, strides:3) as base configuration. We vary the batch size, kernel size, strides, and filters of the above-mentioned Conv2d operator with single and half-precision modes. The roofline analyses for the mentioned Conv2D operators are performed in both forward and backward passes. However, we can use and change only batch size configuration for further experiments of MTD's model. Still, we perform roofline analysis for varying strides, kernels, and filters to identify bottlenecks. Due to the page limit, we only included the roofline chart for batch size configuration and only backward passes of varying kernel size. For the remaining configurations, we summarised them in their appropriate sections.

Batch Size: For this experiment, we used the above-mentioned Conv2D operator for roofline analysis by varying batch sizes from 16, 32, and 64. As shown in Figure 5, the forward pass of the V100 GPU shows all batch sizes of single precision and half-precision L2 and HBM overlap, which means poor L2 cache data locality, but there is a gap between L1 and L2 that indicates there is less L1 cache misses. Half precision mode outperformed single precision for both (V100 and A100) forward pass and backward pass. In half-precision, batch sizes 16 and 32 perform less than batch size 64. V100 forward pass results are the same as A100 forward pass, but A100 gives better data locality of L1, L2, and HBM for single and half-precision. In A100 backward pass increasing batch size from 16 to 64, performance is not linearly increasing.

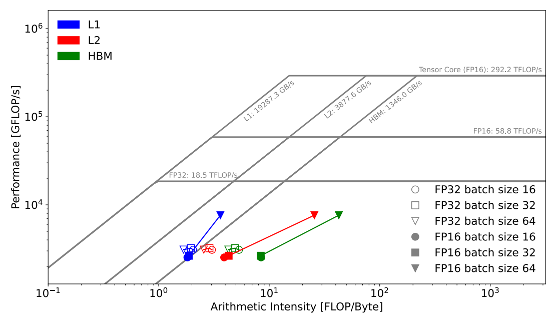
Strides: For this experiment, we use the base Conv2D operator and vary the strides from 2, 3, 4, and 5. The backward



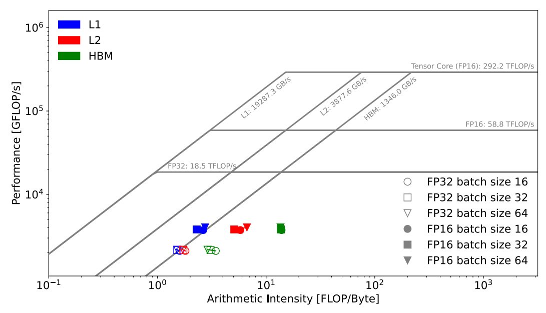
(a) V100 Forward Pass



(b) V100 Backward Pass



(c) A100 Forward Pass



(d) A100 Backward Pass

Fig. 5: Batch Size: Roofline Analysis of V100 and A100 GPUs Forward and Backward pass of MTD Convolutional Operator Conv2D:(tensor shape: 128x96x112, filters: 56, kernel: 3x3, strides:3) by varying batch size of 16, 32 and 64.

pass of V100, for single and half-precision stride size 2, gives more performance than others. There is more L1 cache misses on single precision. In half-precision stride sizes, 2 and 3 give good L1 cache data locality. Half-precision of A100 GPU, there is a good gap between L1, L2, and HBM. Therefore, it gives good data locality than V100. Strides 4 and 5 perform more than strides 2 and 3 in half-precision of A100 GPU.

Kernel: Now we vary the kernel size from 3x3, 5x5, and 7x7 in the base Conv2d operator. As shown in Figure 6, V100 backward pass single and half-precision give good data locality, and performance increases linearly with increasing the kernel size, but A100 backward pass kernel 5x5 gives better performance than 7x7 in half-precision, but single precision performance increase linearly like V100 and it gives poor L1 cache data locality on single precision.

Filters: We use the base Conv2D operator for this experiment and vary the filter sizes to 56, 112, and 224. The backward pass A100 GPU half-precision performance increases linearly with the filters, whereas in single precision, filter size 112 performs better than filter size 224. Furthermore, in V100 backward pass, both single and half-precision performance increases linearly with filter size. As a result, both A100 and V100 backward pass give good data locality in half-precision, and some L1 caches miss on single precision. On the other hand, V100 forward pass has poor data locality and linear performance on single precision. Both A100 and V100 forward pass half-precision of filters 112, and 224 give the same level of performance but comparatively very high than filter 56.

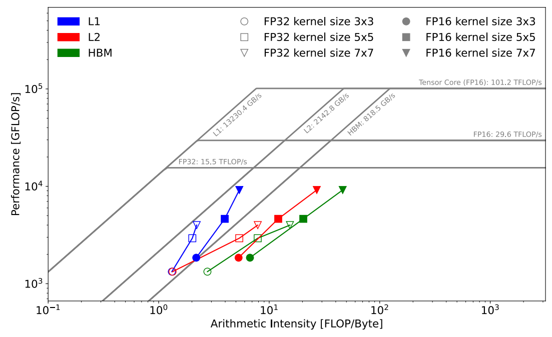
***Summary*:**

- Increasing the batch size from 32 to 64 gives a significant performance boost.
- In the backward pass of A100, half-precision strides 4 and 5 give more performance than 2 and 3. whereas V100 backward pass (half and single precision) stride 2 provides higher performance than with 4 and 5.
- The V100 backward pass kernel size $(3\times3, 5\times5, 7\times7)$ performance increases linearly in both half and single precision. Whereas half-precision of A100 backward pass of kernel size 5×5 gives more performance than 7×7.
- In A100, half-precision backward pass performance increases linearly with the number of filters (56, 112, 224). In single precision, 112 filters give more performance than 224 filters.

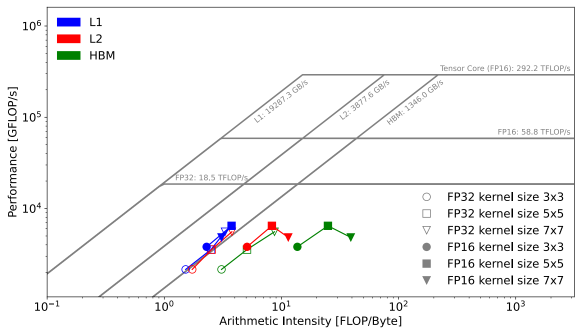
Comparatively, A100 offers better data locality of L1, L2, and HBM than V100. And more L1 cache miss on single precision than half precision. We can use batch size 64 and mixed precision rather than 32 batch size and single precision for training. Bottleneck findings of strides, kernel, and filters help the developer modify the MTD's model architecture to utilize the hardware effectively.

B. Training Level

In single GPU training, we trained the MTD model with two precision modes, single precision and mixed precision. As we already know, half-precision performs more than single precision for Conv2D operators from the operator level.



(a) V100 Backward Pass



(b) A100 Backward Pass

Fig. 6: Roofline Analysis of V100 and A100 GPUs Backward pass of MTD Convolutional Operator by varying kernel size.

TABLE II: Single GPU Training with Mixed Precision and Single-Precision Mode on A100 and V100 GPUs.

GPU	**V100**				**A100**			
Batch Size	**32**		**64**		**32**		**64**	
Precision	**Single**	**Mixed**	**Single**	**Mixed**	**Single**	**Mixed**	**Single**	**Mixed**
Data loading time (s)	8.87	17.29	13.42	12.02	5.08	5.14	5.05	5.58
Total run-time (s)	2847	1892	2756	1779	1530	1319	1450	1212
Total training time (s)	2805	1844	2706	1734	1507	1296	1427	1188
Avg. epoch time (s)	40.07	26.34	38.66	24.78	21.53	18.52	20.39	16.98
Training loss	0.0532	0.0535	0.0554	0.0551	0.0532	0.0536	0.0552	0.0556
Validation loss	0.0585	0.0586	0.0599	0.0591	0.0584	0.0601	0.0588	0.0594
Avg. GPU Usage %	90.53	81.86	91.38	83.55	22.38	19.27	22.46	22.02
Avg. Mem. Usage Rate %	63.24	44.41	66.13	47.52	14.06	7.25	14.94	10.72
Avg. GPU Temp °C	59.92	57.5	58.49	57.74	50.96	48.45	50.9	47.85
Avg. GPU Power W	259.13	244.14	252.35	238.49	116.01	98.73	116.99	107.51

The mixed precision contains both single and half-precision but is automatically selected by the deep learning framework. Our assumption is mixed precision will perform more than single precision. We will verify it from training-level results. We trained in V100 and A100 GPUs with batch sizes (32, 64) and 70 epochs with Adam [16] optimizer. As results shown in Table II, single precision of V100 using 32 batch sizes consumes a higher total training time. Whereas A100 mixed precision of 64 batch size consumes less training time than others. The training and validation loss of batch sizes 32 and 64 is almost identical for single precision and mixed precision on V100 and A100 GPUs. Mixed precision gives good training time per epoch than single precision mode. We exclude the first epoch time for the average training time per epoch calculation because the first epoch always takes higher time in single precision and mixed precision on both V100 and A100. While increasing the batch size from 32 to 64, there is only slight performance improvement for V100 and A100 in both modes. The mixed precision of V100 gives more performance than single precision, whereas A100 gives only a slight marginal improvement. Because tensor cores are activated only half-precision in V100 but single and half-precision in A100. Mixed precision consumes less average GPU and memory usage than single precision mode. V100 consumes more additional GPU power than A100 GPU. While using mixed precision in V100 and A100, GPU temperature and power consumption were also reduced.

Summary: Mixed precision training consumes less GPU power and memory than single precision training on both GPUs. V100 mixed precision of batch size 64 utilizes Tensor Core at 89.5%, whereas A100 utilizes only 31.7% of TC.

C. *Distributed Training Level*

We conduct this experiment to determine whether increasing the number of GPUs for training the MTD model will affect the training time. We used two nodes for this experiment JUWELS Booster and JUWELS Cluster. Due to the inadequate training data volume on the MTD model, we used a maximum of 4 GPUs for distributed training, and we increased GPU gradually from 1 to 4. We used TensorFlow mirrored strategy API for distributed training as discussed in section III-A. We used only ten epochs and dynamic batch sizes (Batch size gradually increased with the number of GPUs). We already discussed in section IV-B that mixed precision gives more performance than single precision, so we used mixed precision for training. The loading data time is lesser in a single GPU than in multiple GPUs because data needs to transfer to multiple GPUs. These variations are higher in V100 than in A100. The average epoch time for training in V100 decreases significantly from using 1 GPU to 2 GPUs but later, using 3 and 4 GPUs does not give enough performance boost. Whereas in A100, using 1 GPU gives an average epoch time of 22.9 seconds, and using 2 GPUs gives 19.3, there is no substantial improvement in using multiple GPUs. Final training loss and validation loss are almost the same for using 1 and 2 GPUs for both V100 and A100, but further increasing GPUs, there

TABLE III: Inference Performance Metrics on A100 and V100

Device	V100		A100	
Precision	Single	Mixed	Single	Mixed
Wallclock time (s)	0.09	0.051	0.10	0.037
TC utilization %	None	58.7	16.1	6.4
GPU utilization %	8.1	3.2	2.6	1.1

is a drop in training and validation losses because we increase batch size gradually with GPUs, batch size implicit connection with gradient estimator, so increasing batch size above 64 causes low variance.

***Summary*:** Using four GPUs on V100 and A100 is not sufficiently utilizing all the GPUs in the node for training. V100 utilizes 81.8% of TC, whereas A100 only utilizes 26.9% of TC. Average GPU usage and memory usage decreases gradually with an increasing number of GPUs in V100. On average, A100 consumes less GPU energy than V100. Overall the MTD model does not sufficiently utilize the multiple GPUs for distributed training. We suggest using a single GPU for training the MTD model is sufficient.

D. Inference Level

The inference level calculates the time to complete the full forward propagation of the MTD model. Furthermore, we investigate the roofline analysis on the most time-consuming kernel during the inference. The MTD model network is explained in the section II. We compare the results with V100 and A100 GPUs. We used single and mixed precision MTD models for the inference benchmark. Both the model trained with batch size 64. As shown in Table III, the model is trained with mixed precision utilizing tensor core very well on V100 than the model trained with single precision. Whereas in A100, both single and mixed precision utilizes the tensor core because, as we already discussed in section II-B, A100 will utilize the tensor core on both single and half-precision, but V100 only uses tensor core on half precision.

The model's accuracy with the single and mixed precision modes is almost identical. In A100 mixed precision model took less time for inference than other variants. Inference on the mixed precision model performs more than a single precision-trained model on both GPUs. So we will use mixed precision trained model for inference roofline analysis. If we perform roofline analysis for all the kernels during the inference will not be very effective in studying the system because there are too many kernel launches during inference. So first, we used the NSight System CLI tool to extract the kernel statistics, we identified **conv2d_grouped_direct_kernel** and **implicit_convolve_sgemm_kernel** took considerable GPU computing time than others. We also performed roofline analysis on the most time-consuming kernels. Both A100 and V100 conv2d_grouped kernel utilize only 1% of the device's FP32 Peak performance. V100 implicit_convolve_sgemm utilizes 10% device's FP32 Peak performance, and in A100, it utilizes 13% device's FP32 Peak performance. As a result, the MTD model inference in A100 is not giving a bigger performance boost than V100 GPU.

***Summary*:** For inference, A100 mixed-precision trained models provide 2.7 times more performance than A100 single precision and 2.4 times more than V100 single precision. After roofline analysis of the MTD model inference, both V100 and A100 GPUs do not effectively utilize the device's peak performance. The A100 GPU does not provide a significant performance increase compared to the V100 GPU.

***Suggestion*:** We can utilize the Nividia MIG technology to partition GPU into multiple instances and deploy the model into any one of the instances for inference. This technology is applicable only starting from Ampere architecture. For A100, there are seven MIG profiles available, but we categorize them into 4 profiles

- **7_5GB**: split GPU into 7 instances. Each 5GB memory
- **3_10GB**: split GPU into 3 instances. Each 10GB memory
- **2_20GB**: split GPU into 2 instances. Each 20GB memory
- **1_40GB**: split GPU into 1 instance. Full 40 GB memory

If the model inference utilizes the device less than 25%, we use **7_5GB** profile. If usage is 25% to 50%, we can use **3_10GB** profile. Likewise, we can use other profiles for inference. For example, our MTD model utilizes only 1.1% of GPU for inference in A100, so we use **7_5GB** profile and split the A100 GPU into 7_5GB profile by using the following command ***sudo nvidia-smi mig -cgi 19 -C*** and deploy the MTD model into any of the instances or on many instances and use a load balancer[7] for the inference server to effectively utilize the underlying A100 GPU.

E. Cloud Compute Cost

Training: We compare the public cloud computing cost of V100 and A100 (1 GPU and 4 GPU) on three cloud vendors AWS, GCP, and Azure. As we already gathered training time from training and distributed training levels, V100 single and four GPUs' average epoch times are 32.8s and 22.3s. Similarly, A100, a single, and four GPUs are 22.88s and 18.94s, respectively. At least 70 epochs are needed to get good accuracy of the MTD model. To get a fair comparison, we calculated the computing cost for ten times training the model. So V100 single and four GPU total training time 6.38 hrs and 4.34 hrs. The total training time in A100 single and four GPU is 4.45 hrs and 3.69 hrs, respectively. As shown in Table IV, GCP gives the least computing cost for one A100 GPU than V100. AWS consumes higher GPU costs than other vendors because AWS doesn't have instances with exactly 4 A100 GPUs, so we calculated the cost for instance, with 8 A100 GPUs. A single A100 GPU from GCP is good for MTD model training.

Inference: As we discussed in section IV-D, both V100 and A100 GPUs give almost the same performance for the MTD model inference on test data. We checked MTD model inference for CPU to determine whether it will cost less than GPU. We used AWS c5.4xlarge instance, which has Intel(R) Xeon(R) Platinum 8275CL CPU @ 3.00GHz CPU total of 16 cores for the MTD model trained with mixed-precision

[7]https://www.envoyproxy.io/docs/envoy

TABLE IV: GPU Compute Cost Comparison for training with AWS, GCP, and AZURE

Cloud Provider	Instance Type	Device	GPUs	Per Hour USD	Total Training (hour)	Total USD
AWS	p4d.24xlarge	A100	8	32.77	3.69	120.92
	p3.2xlarge	V100	1	3.06	6.38	19.52
	p3.8xlarge		4	12.24	4.34	53.12
GCP	NVIDIA A100	A100	1	2.93	4.45	13.03
			4	11.74	3.69	43.32
	NVIDIA V100	V100	1	2.48	6.38	15.82
			4	9.92	4.34	43.05
AZURE	NC6s v3	V100	1	3.06	6.38	19.52
	NC24rs v3		4	13.46	4.34	58.41
	NC24ads v4	A100	1	3.67	4.45	16.33
	NC96ads v4		4	14.69	3.69	54.20

mode takes an average time of approximately 5.15 seconds for inference which is slower than A100 inference time. GPUs outperform CPUs in terms of performance and cost for MTD model inference. For example, per hour cost for a c5.4xlarge instance is \$0.68 and GCP single A100 GPU per hour cost is \$2.93. If we perform 10^4 times inference c5.4xlarge instance cost is \$9.7 whereas cost of GCP A100 is \$0.30. We suggest using A100 for inference to leverage Nividia MIG technology to utilize the hardware effectively.

V. Conclusions

We provided a comprehensive methodology for profiling the MAELSTROM Temperature Downscaling (MTD) application starting from the operator level and identifying the bottleneck of the convolutional operator using roofline analysis and using the parameters to train the model and analyze the training and distribute training level. We profiled the MTD model inference to identify the device utilization, and we suggested a methodology to use Multi-Instance GPU for inference. Finally, we compared computing costs for training and inference from cloud vendors. Our results will help to choose the hardware configuration but also helps to enhance the MTD model architecture because we detailed operator bottlenecks for varying batch sizes, kernels, strides, and filters using roofline analysis. To the author's knowledge, our methodologies and results will help other researchers and developers enhance and profile their deep learning models.

Acknowledgment

This work has been funded by EuroHPC JU under contract number 955513 and by the Luxembourg National Research Fund (FNR) under contract number 15092355.

References

[1] Y. Wang, "Single image super-resolution with u-net generative adversarial networks," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 4, 2021, pp. 1835–1840.

[2] J. Leinonen, D. Nerini, and A. Berne, "Stochastic super-resolution for downscaling time-evolving atmospheric fields with a generative adversarial network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 9, pp. 7211–7223, 2021.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[4] G. Bing, L. Michael, D. Peter, C. Matthew, N. Thomas, A. Markus, and B.-N. Tal, "Report on a survey of MAELSTROM applications and ML tools and architectures. Deliverable 2.1." MEAELSTROM EuroHPC project., Tech. Rep., 2021. [Online]. Available: https://www.maelstrom-eurohpc.eu/deliverables

[5] M. Brorsson, "Roadmap analysis of technologies relevant for ML solutions in W&C. Deliverable 3.2." MEAELSTROM EuroHPC project., Tech. Rep., 2021. [Online]. Available: https://www.maelstrom-eurohpc.eu/deliverables

[6] Y. Sha, D. J. G. Ii, G. West, and R. Stull, "Deep-Learning-Based Gridded Downscaling of Surface Meteorological Variables in Complex Terrain. Part I: Daily Maximum and Minimum 2-m Temperature," *Journal of Applied Meteorology and Climatology*, vol. 59, no. 12, pp. 2057–2073, Dec. 2020. [Online]. Available: https://journals.ametsoc.org/view/journals/apme/59/12/jamc-d-20-0057.1.xml

[7] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, "A modular benchmarking infrastructure for high-performance and reproducible deep learning," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019, pp. 66–77.

[8] C. Yang, Y. Wang, T. Kurth, S. Farrell, and S. Williams, "Hierarchical roofline performance analysis for deep learning applications," in *Intelligent Computing*, K. Arai, Ed. Cham: Springer International Publishing, 2021, pp. 473–491.

[9] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446–459.

[10] C. A. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. D. Bailis, K. Olukotun, C. Ré, and M. A. Zaharia, "Dawnbench : An end-to-end deep learning benchmark and competition," 2017.

[11] S. Narang and G. Diamos, "Deepbench: Benchmarking deep learning operations on different hardware (2017)," 2017. [Online]. Available: https://github.com/Baidu-Research/DeepBench

[12] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Jayarajan, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "Benchmarking and analyzing deep neural network training," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 88–100.

[13] Y. Wang, C. Yang, S. Farrell, Y. Zhang, T. Kurth, and S. Williams, "Time-based roofline for deep learning performance analysis," in *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*, 2020, pp. 10–19.

[14] N. Ding and S. Williams, "An instruction roofline model for gpus," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2019, pp. 7–18.

[15] D. Ernst, G. Hager, J. Thies, and G. Wellein, "Performance engineering for real and complex tall & skinny matrix multiplication kernels on gpus," *The International Journal of High Performance Computing Applications*, vol. 35, no. 1, pp. 5–19, 2021. [Online]. Available: https://doi.org/10.1177/1094342020965661

[16] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.