Unleashing the Potential of Mixed Precision in AI-Accelerated CFD Simulation on Intel CPU/GPU Architectures

Kamil Halbiniak[1][0000−0001−9116−8981], Krzysztof Rojek[1][0000−0002−2635−7345], Sergio Iserte[2][0000−0003−3654−7924], and Roman Wyrzykowski[1][0000−0003−1724−1786]

[1] Institute of Computer and Information Sciences, Częstochowa, Poland
{khalbiniak, krojek, roman}@icis.pcz.pl
[2] Barcelona Supercomputing Center, Spain
sergio.iserte@bsc.es

Abstract. CFD has emerged as an indispensable tool for comprehending and refining fluid flow phenomena within engineering domains. The recent integration of CFD with AI has unveiled novel avenues for expedited simulations and computing precision. This research paper delves into the accuracy of amalgamating CFD with AI and assesses its performance across modern server-class Intel CPU/GPU architectures such as the 4th generation of Intel Xeon Scalable CPUs (codename Sapphire Rapids) and Intel Data Center Max GPUs (or Ponte Vecchio).
Our investigation focuses on exploring the potential of mixed-precision techniques with diverse number formats, namely, *FP32*, *FP16*, and *BF16*, to accelerate CFD computations through AI-based methods. Particular emphasis is given to validating outcomes to ensure their applicability across a CFD motorBike simulation.
This research explores the performance/accuracy trade-off for both AI training and simulations, including OpenFOAM solver and interference with the trained model, across various data types available on Intel CPUs/GPUs. We aim to provide a thorough understanding of how different number formats impact the performance and accuracy of the DNN-based model in various application scenarios running on modern HPC architectures.

Keywords: HPC · CFD · AI/ML · DNN · mixed precision · CPU/GPU · Intel architectures

1 Introduction

Computational Fluid Dynamics (CFD) has emerged as a cornerstone in understanding and optimizing fluid flow phenomena across various engineering disciplines. In recent years, the intersection of CFD with artificial intelligence (AI) has sparked new possibilities, promising both accelerated simulations and enhanced accuracy. This paper addresses a crucial amalgamation of topics at the

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

forefront of this intersection—floating-point number formats, mixed precision in AI, and the utilization of Intel CPU and GPU architectures in CFD simulations.

In this paper, our focus centers on unraveling the intricacies of AI-accelerated CFD simulations by concentrating specifically on the Intel CPU and GPU architectures. Notably, we deliberately abstain from exploring the extensively studied NVIDIA GPU platform due to the wealth of existing literature in this domain. By directing our attention to the Intel ecosystem, we aim to contribute a distinctive perspective on mixed-precision computations, recognizing the diverse range of processors and graphics units integral to many computational environments.

Our study leverages the capabilities of OpenFOAM, a versatile and open-source CFD software, to conduct a comprehensive simulation of steady flow around a motorcycle and rider. OpenFOAM's flexibility and robust numerical algorithms make it an ideal tool for capturing the intricate fluid dynamics involved in complex scenarios. By utilizing OpenFOAM, we aim to not only investigate the impact of mixed-precision computations and Intel CPU/GPU architectures on the simulation accuracy and efficiency but also to showcase the practical application of these advancements in a real-world CFD scenario.

Our paper includes a list of contributions that we have made. These contributions are outlined below:

- *Utilizing different number formats for improved performance and validated accuracy.* Investigating the impact of floating-point data formats — float32 (*FP32*), float16 (*FP16*), and bfloat16 (*BF16*) —in CFD AI acceleration is a central focus of this work. We systematically explore the performance gains and trade-offs associated with each datatype. Additionally, we emphasize the crucial aspect of validating the accuracy of results obtained using these datatypes to ensure their applicability across diverse engineering scenarios.
- *AI acceleration of OpenFoam CFD simulation.* The adaptation of AI-accelerated techniques in CFD simulations, particularly through the integration of machine learning models into OpenFOAM, represents a significant stride towards more efficient and adaptive fluid flow predictions. Our paper demonstrates how AI can catalyze accelerating OpenFOAM simulations, leading to quicker turnaround times without compromising accuracy.
- *Verification on Intel CPU and GPU Platforms.* In pursuit of a holistic understanding, our paper extends beyond theoretical considerations. We delve into the practical implementation of our findings on Intel CPU and GPU architectures, offering a comparative analysis of performance and accuracy. By verifying the results on these widely used platforms, we bridge the gap between theoretical advancements and real-world applicability.

2 Related work

In the realm of CFD simulations, the choice of data formats significantly influences both the accuracy and efficiency of computations. This work delves into the fundamental aspects of floating-point data formats, with a particular focus on *FP32*, *FP16*, and *BF16*. Additionally, we explore the application of mixed

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

precision in deep learning techniques to accelerate CFD simulations, shedding light on the trade-offs between precision and computational efficiency in the context of AI-driven fluid dynamics simulations.

In recent years, there has been a notable surge in research exploring the integration of AI techniques within CFD simulations. These efforts aim to enhance the efficiency and precision of CFD simulations [16], thereby enabling the handling of more intricate and realistic problems [26, 10].

A prevalent approach involves leveraging machine learning algorithms to model fluid behavior. For instance, neural networks have been successfully employed to simulate turbulence [2], forecast drag and lift forces on aircraft, and optimize the design of turbulent flow control devices [20]. Additionally, researchers have delved into utilizing AI techniques for optimizing CFD simulation parameters and settings [15]. Methods such as genetic algorithms have been utilized to determine the optimal mesh size and solver configurations for specific simulations, with the ability to dynamically adjust these parameters based on simulation outcomes [3].

Furthermore, investigations have explored employing AI techniques to analyze and interpret CFD simulation results [27]. Clustering algorithms have been effective in grouping similar flow patterns [21], while classification algorithms have been instrumental in identifying and categorizing various flow types.

To the best of our knowledge, there is a lack of literature exploring the usage of the newest Intel server-class GPUs (as well as CPUs) for CFD simulations incorporating AI techniques. In particular, in our previous works [16, 17], we used NVIDIA V100 GPUs, while Graphcore IPU accelerator was employed in paper [18]. Despite the growing interest in leveraging AI for CFD acceleration and the increasing utilization of NVIDIA and AMD GPUs, there appears to be a gap in research addressing the optimization and performance assessment of Intel GPUs in this domain. The same conclusion is pertinent for the newest Intel data center CPUs, introducing AMX accelerators.

3 Floating-point data formats and mixed precision in AI-accelerated CFD simulation

Precision in deep learning models is a critical factor influencing performance, memory utilization, and overall computational efficiency. This section provides an in-depth analysis of different floating-point data formats, namely *FP32*, *FP16*, and *BF16*, and their implications for mixed precision deep learning. By understanding the nuances of these formats, we aim to unravel the potential advantages and challenges associated with employing mixed precision techniques in the context of accelerating AI algorithms for CFD simulations.

The *FP32* format was the backbone of deep learning for a long time [14]. It offers a high range of representable values, making it suitable for a wide array of AI computations, particularly in problems where accuracy is crucial. On the other hand, *FP32* requires relatively a lot of memory and computing resources. In contrast, lower precision formats such as *FP16* and *BF16* sacrifice precision

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

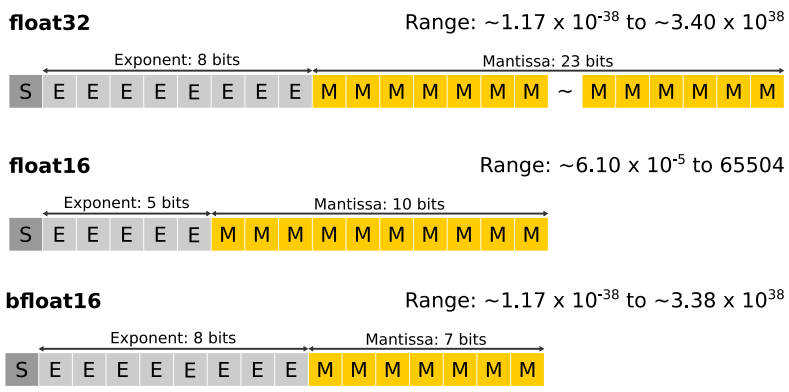

Fig. 1: Comparison of *FP32*, *FP16* and *BF16* data formats (ranges for normalized values) [12]

in favor of reduced memory usage and computational demands [12], making them an attractive choice for large-scale AI computations (e.g., training of large models). Nonetheless, reduced precision may lead to numerical instability in certain computations. Therefore, the optimal choice of floating-point precision depends on the specific application and the trade-off between precision needs, memory constraints, and computational efficiency.

Figure 1 illustrates the comparison of *FP32*, *FP16* and *BF16* data formats. The *FP32* uses 32 bits to represent a floating-point number. In this format, a single bit is allocated for the sign, 8 bits for the exponent, and 23 bits for the mantissa. Half-precision *FP16* format utilizes 16 bits to represent floating-point values. Within these 16 bits, a single bit is reserved for sign, 5 bits for exponent, and 10 bits for mantissa [12]. This format offers a reduced precision compared to *FP32*, making it more memory-efficient and computationally faster. However, it comes with the cost of the smaller range of representable values [12]. The *BF16* also uses 16 bits while balancing precision and efficiency. It provides the approximate dynamic range of *FP32* format by retaining eight exponent bits but supports only a 7-bit mantissa rather than the 23-bit [4, 12]. The *BF16* is a replacement for the *FP16* format. It allows for fast conversion to and from *FP32*. Unlike *FP16*, which usually requires special techniques, conversion from *FP32* to the *BF16* is performed by truncating the mantissa field [12]. This difference can be seen in the training process, where the use of *FP16* forces adding a loss scaling process to preserve small gradient values [14]. Another advantage of *BF16* is the hardware cost. By having three fewer mantissa bits, the *BF16* multiplier takes up about half of the hardware area (number of transistors) against the *FP16* unit [25]. The *BF16* data format is implemented in modern Intel CPUs and GPUs, Google's TPU, and NVIDIA GPUs [24].

Mixed-precision is a technique used to optimize the performance of numerical computations using a combination of lower and higher-precision data formats. It has become a powerful optimization to accelerate AI computations, especially

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

deep learning training and inference processes [12]. Employing a combination of 16-bit and 32-bit floating-point data formats in a model during training aims to run it faster and reduce memory utilization. Additionally, accelerating the computations and reducing the execution time with lower-precision number formats allows decreasing energy consumption. By preserving certain parts of the model in the *FP32* format for numeric stability, the model will have a lower step time and train equally as well in terms of the evaluation metrics (e.g., accuracy) [24]. Modern deep learning frameworks and libraries, such as TensorFlow or PyTorch, provide tools to facilitate the implementation of mixed precision in both model training and inference. Although mixed precision can be run on most hardware, it will only speed up the computations on the devices that support mixing 16-bit and 32-bit data formats [24].

4 Intel CPU and GPU architectures in deep learning mixed-precision computation

As the landscape of deep learning accelerates, the choice of hardware architectures plays a pivotal role in achieving optimal performance [19]. Focusing on modern Intel's CPU and GPU architectures, this section investigates their role in facilitating mixed-precision computations for deep learning tasks. In particular, we investigate how the features and capabilities of the 4th Generation of Intel Xeon Scalable CPUs (codename Sapphire Rapids) [6] and Intel Data Center Max GPUs (codename Ponte Vecchio) [9] can be used in practice to accelerate AI computations.

The Intel Xeon Sapphire Rapids processors revolutionize a landscape of AI computations on general-purpose processors by introducing a built-in Intel Advanced Matrix Extension (Intel AMX) accelerator. Intel AMX is a dedicated hardware block of the processor core that helps optimize and accelerate deep learning training and inferencing workloads relying on matrix operations [6]. It allows running AI computations directly on the CPU instead of offloading them to a discrete accelerator (e.g., GPU). The Intel AMX architecture consists of two components [6]: (i) tiles consisting of two-dimensional registers (each 1KB in size) that store large chunks of data; (ii) Tile Matrix Multiplication (TMUL) which is an accelerator engine attached to the tile that performs AI matrix multiplication.

The AMX accelerator supports *INT8* (8-bit integer) and *BF16* data formats. While the first is a data type used for inferencing, the second can be used for both training and inference. Using the AMX, Intel Sapphire Rapids-based processors can quickly pivot between optimizing the AI workloads and general computing. In practice, the programmers can code AI computations to take advantage of the AMX instruction set and implement non-AI functionality based on the processor instruction set architecture [6].

Intel Data Center GPU Max is a series of general-purpose discrete GPUs, designed for breakthrough performance in data-intensive computing models used in AI and HPC [9]. The GPUs are available as PCIe cards and OpenCompute

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

Accelerator Modules to offer remedies for servers with high GPU density. Intel Data Center GPUs are based on the X^e HPC architecture with a compute-focused, programable, and scalable element named the X^e-core [9]. Each core consists of 512-bit wide vector engines, 4096-bit wide matrix engines called Intel Xe Matrix eXtensions (Intel XMX), L1 data cache and shared local memory [9]. The vector engines of X^e-core support *FP64*, *FP32* and *FP16* vector computations. Simultaneously, the XMX is engineered to accelerate AI computations and provides support for *TF32* (19-bit tensor float format), *BF16*, *FP16* and *INT8* data formats.

The Intel Data Center GPUs are built upon the X^e HPC Stacks, which are made up of various tiles stacked on top of each other within a single package. The Xe HPC Stack includes of the X^e-core Tile (Compute Tile), L2 Cache Tile, Base Tile (PCIe paths, media engine, etc.), High Memory Bandwidth Tile, Xe Link Tile for scale-up and scale-out, and the Embedded Multi-Die Interconnect Bridge for communication between X^e HPC Stacks [9]. These components all reside within a Multi Tile Package. The top-of-the-line Intel Data Center Max 1550 GPU contains two X^e HPC Stacks with 64 X^e-cores, 204MB of L2 cache and 64GB HBM2e each. This Intel GPU offers the peak performance of 832 TFLOP for *BF16* and 52 TFLOPS for both *FP64* and *FP32* computations [9].

To leverage Intel CPUs/GPUs architectures in deep learning mixed-precision computation, the programmers may use the TensorFlow framework together with Intel Extension for TensorFlow (ITEX in short). ITEX is a heterogeneous, high-performance, deep-learning extension plugin based on the TensorFlow Pluggable Device interface [7]. It is designed to optimize the performance of TensorFlow-based applications on Intel computing architectures. The ITEX provides a feature called Advanced Auto Mixed Precision (AMP in short), which allows users to enable mixed-precision computations. The AMP is similar to stock TensorFlow Auto Mixed Precision but offers better usage and performance on Intel CPUs and GPUs [8]. Listing 1.1 shows the snippet of code that enables mixed-precision computing on Intel CPUs and GPUs with ITEX. The presented code sets the global policy leading to the mixed-precision computations based on *BF16* data format. As a result, it allows AI computations to be performed using AMX and XMX instruction sets on Intel Xeon Sapphire Rapids CPUs and Intel Data Centers Max GPUs, respectively. The programmer can easily change the mixed-precision policy to *FP16* data format by replacing `itex.BFLOAT16` with `itex.FLOAT16`.

Beyond specifying the data format used in mixed-precision computing, the AMP also allows controlling operators that can be converted to lower-precision data types. TensorFlow provides *Allow*, *Deny*, *Clear*, and *Infer* list of operators to classify operators based on the operation's numerical safety [5, 7]. The numerical safety corresponds to how the accuracy of the model is affected by using lower precision. The exact lists could be found in `auto_mixed_precision_lists.h` file in the TensorFlow GitHub repository [23]. Listing 1.2 presents the snippet of code corresponding to the usage of the AMP to add and remove operators from the TensorFlow lists. This presented code adds the `Conv2D` operator to

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

the *Deny* list which holds operations considered to be numerically dangerous in lower precision [5]. Simultaneously, the `Conv2D` operator is removed from the list of operators considered as numerically safe for execution (*Allow* list). The modification of the remaining lists is analogous. It is important to note that it is not allowed for the same operator to be in more than one list [24].

```
import intel_extension_for_tensorflow as itex

amp_options = itex.AutoMixedPrecisionOptions()
amp_options.data_type = itex.BFLOAT16

graph_options = itex.GraphOptions(
    auto_mixed_precision_options=amp_options)
graph_options.auto_mixed_precision = itex.ON

config = itex.ConfigProto(graph_options=graph_options)
itex.set_config(config)
```

Listing 1.1: Activation of Advanced Automatic Mixed Precision on Intel CPUs and GPUs using Intel Extension for TensorFlow

```
import intel_extension_for_tensorflow as itex

amp_options = itex.AutoMixedPrecisionOptions()
amp_options.denylist_add= "Conv2D"
amp_options.allowlist_remove = "Conv2D"
```

Listing 1.2: Modification of TensorFlow lists of operators using advanced Automatic Mixed Precision of Intel Extension for TensorFlow

5 AI-accelerated CFD simulation

5.1 CFD simulation of steady flow around a motorcycle and rider

Applying the basic approaches discussed in the preceding sections, we shift our focus to a practical application — CFD simulation of steady flow around a motorcycle and rider. This job uses OpenFOAM [1] to calculate the steady flow. The *simpleFoam* solver is used for this case, performing steady-state, incompressible Reynolds-Averaged Navier-Stokes calculations (RANS) over the mesh. RANS is a widely used approach in CFD for simulating turbulent flows. In fluid dynamics, the Navier-Stokes equations describe the motion of fluid, taking into account viscosity, pressure, and velocity.

In this study, we aim to harness the power of AI to streamline and enhance the evaluation of crucial fluid dynamic parameters, specifically velocity U and pressure p, within the framework of RANS. By integrating AI methodologies into the evaluation process, we seek to improve the accuracy and efficiency of estimating U and p fields.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

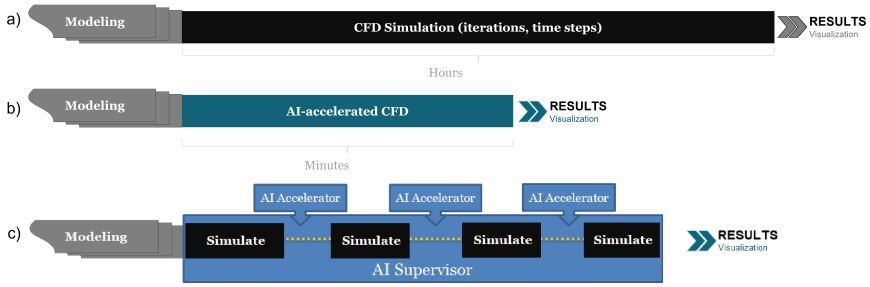

Fig. 2: The idea of integrating the AI module with CFD simulation, where a) presents standalone CFD simulation, b) idea of time reduction by combining CFD simulation with AI, c) the mechanism of management of the CFD simulation by AI supervisor.

5.2 Integration of AI acceleration with CFD solver

This study exploits the methodology proposed in our previous work [17] aimed at enhancing CFD simulations by integrating two main components: the AI supervisor and the AI accelerator. During the inference stage, the AI accelerator uses the previously trained AI model to predict the simulation results based on a relatively small number of iterations generated by the CFD solver. The AI supervisor dynamically switches between traditional CFD simulation and AI predictions, while the AI accelerator module expedites the process by extrapolating simulation results. Initially, the traditional CFD solver runs for a predetermined number of iterations to establish initial data points, subsequently utilized by a machine learning model to forecast fluid flow dynamics. Upon generating output, the AI supervisor directs the CFD solver to resume simulation based on the predicted data, iteratively alternating between CFD and AI components until convergence is reached. The convergence threshold hinges on factors such as the complexity of simulated flow dynamics and the quality of training data. The idea of our method is presented in Fig. 2.

The AI supervisor's role in this methodology is pivotal, discerning data patterns within the simulation to gauge if a steady state has been attained. By analyzing CFD simulation output, it determines whether to invoke the AI accelerator or conclude the simulation. The specifics regarding the AI model, including its architecture and training methodology, are presented in our prior publications [17, 11]. The model primarily comprises convolutional layers, forming the backbone of its architecture. These convolutional layers play a central role in extracting features from the input data, enabling the AI model to learn and predict fluid flow dynamics effectively. All the operators within the model can be safely converted and used with lower precision.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

5.3 Training dataset

Machine learning algorithms, trained on vast datasets generated through RANS simulations, are employed to infer and optimize the estimation of U and p parameters. These datasets are carefully constructed, with each simulation representing various operational conditions, such as different motorbike speeds, alongside additional parameters like ambient temperature, and turbulence intensity. In our case, 50 simulations of a motorbike at different speeds were conducted, each comprising a window of 5 consecutive iterations. They are spaced at intervals of 20 timesteps, with steady-state conditions achieved after 500 iterations. To ensure robustness and diversity, $500/20 = 25$ samples are extracted from each simulation, resulting in the generation of $20 * 25 = 1250$ samples in total.

6 Experimental results

6.1 Testing Platforms

In the tests, we use the following computing platforms:

1. a server with the two 4-th generation 56-core Intel Xeon Platinum 8480 CPUs and 512 GB DDR5 main memory;
2. a server with two 32-core Intel Xeon Platinum 8462Y CPUs, 1024 GB DDR5 main memory and four Intel Data Center Max 1550 GPUs.

Both servers are empowered with TensorFlow 2.13.0, Intel Extension for TensorFlow* 2.13.0.1 and Intel oneAPI Base Toolkit 2023.2.0. The computing platform with Intel Xeon 8480 CPUs has installed Python 3.9.16, while the server with Intel Max GPUs uses Python 3.11.5.

6.2 Accuracy and performance results

Table 1 shows the execution times T_{CPU} and T_{GPU} obtained for the AI training workloads running on two Intel Xeon 8480 CPUs and a single Intel Data Center Max 1550 GPU. In the benchmarks, we focus on pure *FP32* and the mixed-precision training with *BF16* and *FP16* formats, respectively. In the case of

Table 1: Execution times (in seconds) and speedups achieved for AI training with different data formats running on two Intel Xeon 8480 CPU and a single Intel Data Center Max 1550 GPU

Data format	Intel Xeon 8480 CPUs		Intel Max 1550 GPU		S_{GPU}
	T_{CPU}	S_{FP32}	T_{GPU}	S_{FP32}	
FP32	2326	1	408	1	5.7x
BF16	1544	1.51x	238	1.71x	6.48x
FP16	—	—	430	0.95x	—

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

Intel GPU, the AI workloads are executed using a single Compute Tile from both available in the accelerator. This is because the TensorFlow environment, by default, treats every Intel Max 1550 GPU tile as a TensorFlow individual device. To maximize the performance of training, different batch sizes for Intel Xeon CPUs and Intel Max GPU were evaluated.

Intel Extension for TensorFlow employs OpenMP to parallelize the computations across cores/threads of Intel CPUs. To maximize the performance of the AI training, we also investigate different setups of OpenMP environment variables. Among the considered environment variables are [23]:

- `OMP_NUM_THREADS` which sets the number of OpenMP threads;
- `KMP_AFFINITY` which bind OpenMP threads to physical cores;
- `KMP_BLOCKTIME` which sets the time (in milliseconds) that a thread should wait, after completing the execution of a parallel region, before sleeping.

When benchmarking Intel CPUs, these variables were set as follows:

- `OMP_NUM_THREADS`=112;
- `KMP_AFFINITY`=fine,compact,1,0;
- `KMP_BLOCKTIME`=0.

The performance results presented in Table 1 are obtained for the training process with batch sizes equal to 64 and 8 for Intel Xeon 8480 CPUs and Intel Data Center Max 1550 GPU, respectively. Besides the execution times, the table presents also speedup S_{FP32} obtained against *FP32* data format. The last column of Table 1 illustrates the speedup $S_{GPU} = T_{CPU}/T_{GPU}$ obtained for Intel Max 1550 GPU against two Intel Xeon 8480 CPUs. The performance results in Table 1 indicate significant benefits of using mixed-precision training based on *BF16* number format. The speedup achieved against the pure *FP32* AI computations is about 1.5 and 1.7 times for Intel Xeon CPUs and Intel Max GPU, respectively. The Advanced Auto Mixed Precision feature does not support mixed-precision *FP16* AI computations on Intel processors [23]. Thus, we are not able to measure the performance of mixed-precision training for this data format. At the same time, the utilization of *FP16* during training on Intel Max GPU yields even longer execution time than *FP32* data format. The explanation for unexpected low performance may be the cost of casting tensors from *FP32* to *FP16*, or vice versa. In some cases (especially for operations on huge tensors), the casting cost surpasses the performance benefit of using low precision [5].

Based on Table 1, we conclude that Intel Data Center Max 1550 GPU outperforms two Intel Xeon 8480 CPUs. Already for *FP32* computations, a single tile of Intel GPU allows executing the training process 5.7 times faster. Even greater performance gain is notable for *FP16*, where the Intel GPU allows accelerating the AI computations about 6.5 times against two Intel CPUs.

Fig. 3 displays the mean square error (MSE) for CFD simulations using various configurations of the CFD solver in conjunction with the AI accelerator. The MSE serves as a crucial metric indicating the deviation between predicted and actual values during the convergence process toward the steady state. This figure

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

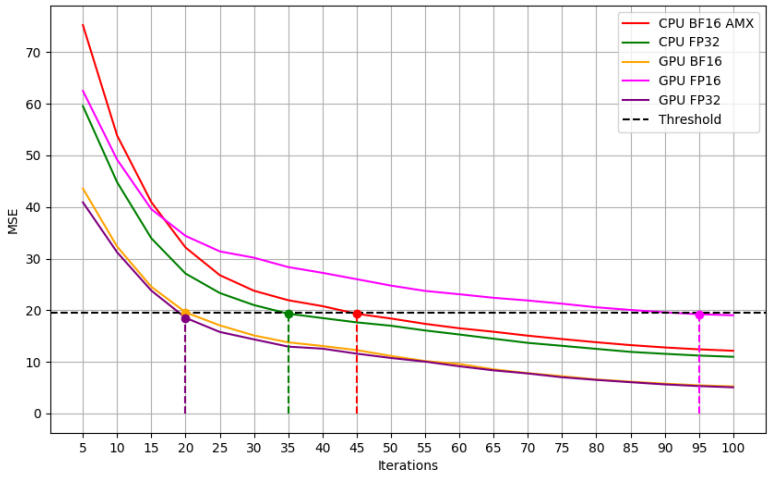

Fig. 3: MSE comparison and the number of iterations required to execute by the CFD solver to achieve the steady state with the AI accelerator.

provides insights into the number of iterations required by each data precision to meet the designated MSE threshold, thereby reaching the desired accuracy and correlation levels.

Each configuration aims to achieve a specific MSE threshold predetermined by our AI supervisor. This threshold is selected to ensure that the result accuracy surpasses 97%, and the Pearson correlation coefficient exceeds 0.98 compared to the CFD standalone version.

Table 2 compares accuracy metrics for various configurations of the CFD solver integrated with the AI accelerator executed on both CPU and GPU platforms. It outlines the number of iterations required for convergence, MSE, Pearson correlation coefficient, and accuracy percentages for each configuration. Notably, it highlights the trade-offs between precision levels provided with various data formats (*FP16*, *BF16*, and *FP32*) for the studied platforms (CPU and GPU) in terms of computational model accuracy.

The last row of Table 2 represents the speedup determined as the ratio of iterations for the standalone CFD solver and AI-accelerated simulation. This way of calculating the speedup is justified by the fact that the execution time for performing CFD solver iterations is much longer than the execution time of AI predictions.

Fig. 4 illustrates the number of iterations required by the CFD solver in conjunction with an AI accelerator to achieve a steady state during application execution. It is important to underline that the CFD solver's computational load significantly influences the overall execution time of the application.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

Table 2: Accuracy comparison and the number of iterations required to execute by the CFD solver to achieve the steady state with the AI accelerator

	CPU *BF16 AMX*	CPU *FP32*	GPU *BF16*	GPU *FP16*	GPU *FP32*
Iterations	45	35	20	95	20
MSE	19.32	19.34	19.61	19.24	18.54
Pearson	0.983	0.983	0.983	0.986	0.984
Accuracy[%]	96.9	96.9	96.9	96.9	97.0
Speedup	11.1	14.3	25.0	5.3	25.0

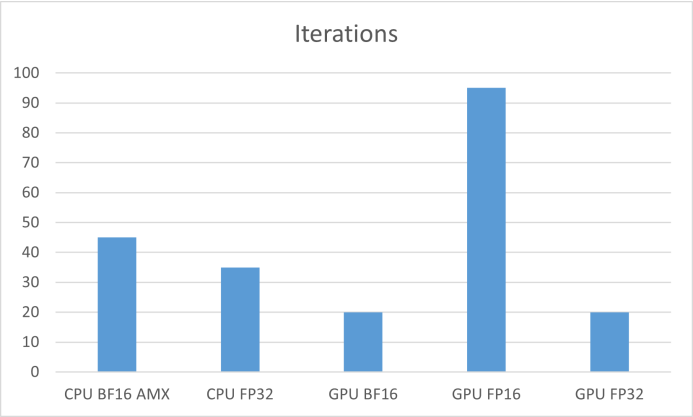

Fig. 4: The number of iterations required to execute by the CFD solver to achieve the steady state with the AI accelerator.

Utilizing the *FP32* precision is the most efficient approach for the CPU. Despite *FP32* requiring a longer inference time compared to *BF16*, when using the trained model, it effectively reduces the number of CFD solver iterations. This reduction in iterations contributes to faster convergence towards the steady state, outweighing the slower inference speed.

When employing the GPU, *BF16* and *FP32* emerge as the preferred precision choice. With *BF16*, the steady state can be reached within the same number of iterations as *FP32*, making it a favorable option. On the contrary, *FP16* for the GPU requires 95 iterations of the solver, making it the least efficient configuration due to its extended computational time.

7 Conclusion

In this paper, we investigate the benefits of using modern server-class Intel CPU and GPU architectures for AI-accelerated simulations. This study focuses on unveiling the potential of using mixed-precision arithmetic based on *FP32*, *FP16*, and *BF16* data formats to accelerate CFD computations through AI techniques. We aim to not only assess the impact of the mixed-precision approach on the

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

performance of computations, but also on the simulation accuracy based on the motorBike use-case scenario.

The performance results achieved in the paper show the performance benefits of using mixed-precision on both Intel Xeon 8480 CPUs and Intel Data Center Max 1550 GPUs for model training. The utilization of *BF16* format allows accelerating the training process about 1.5 and 1.7 times against pure *FP32* computations for Intel CPUs and GPU, respectively. At the same time, the *FP16* mixed-precision training workloads give even longer execution time than in the case of *FP32* data format. An additional conclusion resulting from the benchmarks is that a single tile of Intel Data Center Max 1550 GPU outperforms two Intel Xeon 8480 CPUs during the training of DNN models. In fact, Intel Max GPU allows executing the training process 5.7 and 6.5 times faster against two Intel Xeon CPUs, for *FP32* and *BF16* data formats, respectively.

The results achieved for the whole simulation underscore the nuanced balance between computational performance and model accuracy, as configurations diverge across precision levels (*FP16*, *BF16*, and *FP32*) and platforms (CPU and GPU). Specifically, for the CPU, the most efficient configuration emerges as *FP32* due to its ability to achieve convergence with fewer iterations of the CFD solver compared to other precision types. For the GPU, the optimal configurations are *BF16* and *BF32* that provide a comparable performance.

The presented analysis focuses on the convergence behavior of the CFD solver accelerated with AI. The MSE error serves as a crucial indicator of convergence, with each configuration aiming to meet a specific threshold determined by the AI supervisor. This threshold is carefully selected to ensure the result accuracy of about 97% and the Pearson correlation coefficient surpassing 0.98 compared to the standalone CFD version.

In our future work, we plan to investigate the differences observed in the experiments (see Figure 2 and Figure 4) when implementing the simulation on Intel CPU and GPU. These differences occur for both the *FP32* and *BF16* data formats, and are reflected in the increased number of iterations on CPU compared to GPU. We guess that the reason is the aggressive performance optimizations [22] performed by TensorFlow. This optimization is forced on the CPU by its oneDNN option, which is set by default for Intel CPUs (and can be turned off). This option is not used for Intel GPUs. The side effect of using TensorFlow with oneDNN optimizations are changes in the execution order of operations and greater sensitivity to floating-point round-off errors [22]. Consequently, this improves the performance at the cost of the reduced accuracy, which leads to an increased number of iterations during the inference stage.

Another direction of future work includes investigating the usage of mixed-precision on the newest NVIDIA H100 and H200 GPUs as providing innovative hardware features [13] compared to previous Volta and Ampere GPUs. We also plan to incorporate more application examples to show the benefits of using the mixed-precision approach for CFD simulations.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

Acknowledgements We extend our sincere gratitude to byteLAKE. Their support and collaboration have been instrumental in our research and development endeavors. The researcher from BSC is involved in the project The European PILOT which has received funding from the European High-Performance Computing Joint Undertaking under grant agreement No. 101034126 and No. PCI2021-122090-2A under the MCIN/AEI and the EU NextGenerationEU/PRTR.

References

1. OpenFOAM. https://www.openfoam.com, [Online; accessed 23-February-2024]
2. Berkooz, G., Holmes, P., Lumley, J.L.: The proper orthogonal decomposition in the analysis of turbulent flows. Annual Review of Fluid Mechanics **25**(1), 539–575 (1993)
3. Bhatt, D., Zhang, B., Zuckerman, D.: Steady-state simulations using weighted ensemble path sampling. The Journal of Chemical Physics **133**(1) (2010)
4. Dörrich, M., Fan, M., Kist, A.M.: Impact of mixed precision techniques on training and inference efficiency of deep neural networks. IEEE Access **11**, 57627–57634 (2023)
5. He, X., Sun, J., Chen, H., Li, D.: Campo: Cost-Aware performance optimization for Mixed-Precision neural network training. In: 2022 USENIX Annual Technical Conference (USENIX ATC 22). pp. 505–518. USENIX Association, Carlsbad, CA (2022), https://www.usenix.org/conference/atc22/presentation/he
6. Intel: Accelerate Artificial Intelligence (AI) Workloads with Intel Advanced Matrix Extensions (Intel AMX). https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-12/accelerate-ai-with-amx-sb.pdf (2022)
7. Intel: An Easy Introduction to Intel Extension for TensorFlow. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-intel-extension-for-tensorflow.html (2022)
8. Intel: Intel Extension for TensorFlow: Advanced Auto Mixed Precision. https://intel.github.io/intel-extension-for-tensorflow (2022)
9. Intel: Intel Data Center GPU Max Series Technical Overview. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-data-center-gpu-max-series-overview.html (2023)
10. Iserte, S., Carratala, P., Arnau, R., Barreda, P., Basiero, L., Matínez-Cuenca, R., Climent, J., Chiva, S.: Modeling of Wastewater Treatment Processes with HydroSludge. Water Environment Research **93**(12), 3049–3063 (2021)
11. Iserte, S., Macías, A., Martínez-Cuenca, R., Chiva, S., Paredes, R., Quintana-Ortí, E.S.: Accelerating urban scale simulations leveraging local spatial 3D structure. Journal of Computational Science **62**, 101741 (2022)
12. Kalamkar, D., et al.: A study of bfloat16 for deep learning training (2019), https://arxiv.org/abs/1905.12322
13. Luo, W., et al.: Benchmarking and Dissecting the Nvidia Hopper GPU Architecture (2024), https://arxiv.org/abs/2402.13499v1
14. Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., Wu, H.: Mixed precision training (2018), https://arxiv.org/abs/1710.03740
15. Rojek, K., Wyrzykowski, R.: Performance and scalability analysis of AI-accelerated CFD simulations across various computing platforms. In: Singer, J., Elkhatib, Y., Heras, D., Diehl, P., Brown, N., Ilic, A. (eds.) Euro-Par 2022: Parallel Processing Workshops. pp. 223–234. Springer International Publishing, Cham (2023)

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15

16. Rojek, K., Wyrzykowski, R., Gepner, P.: AI-Accelerated CFD Simulation Based on OpenFOAM and CPU/GPU Computing. In: Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M.A. (eds.) Computational Science – ICCS 2021. pp. 373–385. Springer International Publishing, Cham (2021)
17. Rojek, K., Wyrzykowski, R., Gepner, P.: Chemical Mixing Simulations with Integrated AI Accelerator. In: Mikyška, J., de Mulatier, C., Paszynski, M., Krzhizhanovskaya, V.V., Dongarra, J.J., Sloot, P.M. (eds.) Computational Science – ICCS 2023. pp. 494–508. Springer Nature Switzerland, Cham (2023)
18. Rościszewski P. and Krzywaniak A. and Iserte S. and Rojek K. and Gepner P.: Optimizing throughput of Seq2Seq model training on the IPU platform for AI-accelerated CFD simulations. Future Generation Computer Systems **147**, 149–162 (2023)
19. Silvano, C., et al.: A Survey on Deep Learning Hardware Accelerators for Heterogeneous HPC Platforms (2023), https://arxiv.org/abs/2306.15552
20. Srivastava, S., Damodaran, M., Khoo, B.C.: Machine Learning Surrogates for Predicting Response of an Aero-structural-sloshing System (2019), https://arxiv.org/pdf/1911.10043
21. Sun, P., Gao, L., Han, S.: Identification of overlapping and non-overlapping community structure by fuzzy clustering in complex networks. Inf. Sci. **181**, 1060–1071 (2011)
22. TensorFlow: What's new in TensorFlow 2.9? https://blog.tensorflow.org/2022/05/whats-new-in-tensorflow-29.html (2022)
23. TensorFlow: TensforFlow Official GitHub Repository. https://github.com/ tensorflow/tensorflow (2024)
24. TensorFlow: TensorFlow Guide: Mixed Precision. https://www.tensorflow.org/guide/mixed_precision (2024)
25. Verheyde, A.: BFloat16 Deep Dive: ARM Brings BF16 Deep Learning Data Format to ARMv8-A. https://www.tomshardware.com/news/bfloat16-deep-dive-arm-bf16-support-armv8-a,40305.html (2019)
26. Vinuesa, R., Brunton, S.L.: The Potential of Machine Learning to Enhance Computational Fluid Dynamics (2021), https://arxiv.org/pdf/2110.02085
27. Zhang, S., Wang, R., Zhang, X.: Identification of overlapping community structure in complex networks using fuzzy c-means clustering. Physica A. **374**, 483–490 (2007)

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_15