Proceedings of the Eighth International Conference on
Parallel, Distributed, GPU and Cloud Computing for Engineering
Edited by: P. Iványi, J. Kruis and B.H.V. Topping
Civil-Comp Conferences, Volume 12, Paper 2.4
Civil-Comp Press, Edinburgh, United Kingdom, 2025
ISSN: 2753-3239, doi: 10.4203/ccc.12.2.4

GPU Parallelization for Analytical Hierarchical Tucker Representation Using Binary Trees

Z. Qiu[1,2], F. Magoulès[2] and D. Peláez[1]

[1] Institut des Sciences Moléculaires d'Orsay, Université Paris-Saclay, Orsay, Île-de France, France
[2] Laboratory of Mathematics in Interaction with Computer Science, MICS, CentraleSupélec, Université Paris-Saclay, Gif-sur-Yvette, Île-de-France, France

Abstract

In this contribution, we present a Distributed Data Parallel (DDP) approach for optimizing an analytical hierarchical Tucker in finite basis representation functional Tucker representation (HT-FBR) of high-order multivariate functions. We achieve up to a 10× speedup compared to the benchmark on a 6D proof-of-concept dataset and 2x speedup on a 12D Ethene trajectory dataset trained on 1, 2, 4 and 8 GPUs. This speedup is attributed to our implementation of an element-wise GPU parallelization algorithm for both forward and backward propagations, as well as a customized dataset and data loader configuration. We also show that the model trained on multi-GPU has less overfitting issue than the one trained on a single CPU/GPU. This paves the way to a large scale high-performance training schema and model parallelization on multi-GPU setting, especially the parallelism on levels of the nodes based on their binary tree dependency. On the other hand, improving the accuracy and reducing overfitting as a function of number of GPUs.

Keywords: functional Hierarchical Tucker format, finite basis representation, deep learning, multi-GPU parallelism, DDP, high-dimensional fitting

1 Introduction

A tensor is a multidimensional array of numerical data that naturally arises in fields such as solving partial differential equations (PDEs) like Poisson equation and Navier-Stockes equation, as epitomized by quantum dynamical calculations, and, more recently, the deep learning community etc. Let us assume a tensor $\mathbf{A} \in \mathbb{R}^{N_0 \times N_1 \times \ldots \times N_\mu \times \ldots \times N_{d-1}}$ where $N_\mu, \mu \in \{0, \cdots, d-1\}$ is a set of positive integers. The number of data-points grows exponentially with the number of dimensions, a phenomenon commonly referred to as the "curse of dimensionality." To address this exponential growth, tensor decomposition methods (Tucker, Canonic Polyadic, CP, etc.) have been introduced as a natural extension of matrix factorization, usually written in a so-called Sum-Of-Products form (SOP). Arguably, one of the most widely used SOP form is the Tucker form defined as below:

$$A_{i_0,\ldots,i_{d-1}} \approx A^{\text{Tucker}}_{i_0,\ldots,i_{d-1}} = \sum_{j_0}^{m_1} \cdots \sum_{j_{d-1}}^{m_{d-1}} C_{j_0,\ldots,j_{d-1}} \prod_{\mu=0}^{d-1} U_{i_\mu j_\mu} \tag{1}$$

where $C_{j_0,\ldots,j_{d-1}}$ is a core tensor element and the U_μ terms are collectively referred to as factors. In the context of quantum dynamics, it was shown that the factors could be expressed analytically using a set of auxiliary functions collectively referred to as Finite Basis Representation (FBR). This simple idea has led to the X-FBR (X=Tucker/SOP,[1] CP[2], HT[3, 4]) family of analytical representation of common tensor decomposition methods in the sum-of-products (SOP) form. Assume $t = \{\mu\}$ a singleton, U_t the factor matrix associated with node t, formally, a general representation of these basis functions can be written as a projection:

$$\sigma_t : \mathbb{R}^{n_t} \mapsto \mathbb{R}^{N_t+1} \text{ with } n_t \leq N_t + 1 \tag{2}$$

$$P : \mathbb{R}^{n_t \times k_t} \mapsto \mathbb{R}^{(N_t+1) \times k_t} \tag{3}$$

$$P(U_t)_j = \sigma_t(U_t)_j, j \in (0, \cdots, k_t - 1) \tag{4}$$

Here, σ_t denotes a projection operator that maps the n_t-dimensional column vectors of U_t to a space of dimension $N_t + 1$ spanned by a set of basis functions (e.g. polynomials) defined over the domain Ω_t. The nature of these depends on the topology of the problem.[1] The operator σ_t is applied column-wise to the factor matrix U_t associated with the child nodes. The Tucker form in Eq. 1 in FBR representation, therefore reads:

$$\tilde{A}^{\text{Tucker}} = \sum_{j_0=0}^{k_0} \cdots \sum_{j_{d-1}=0}^{k_{d-1}} C_{j_0,\ldots,j_{d-1}} \bigotimes_{\mu=0}^{d-1} P_\mu(U_\mu)_{j_\mu} \tag{5}$$

In our previous works [3, 4, 5], we extended this FBR approach to accommodate a hierarchical Tucker decomposition scheme, hence the name **HT-FBR**. Such an ap-

proach provides a recursive functional approximation to a multidimensional (binary) tree structure. From a formal perspective, it can be expressed as:[3, 4, 5]

$$\begin{aligned} P_{\Omega^d} A_H(\tilde{n}) &= \prod_{t \in T_I^p} P_t(\tilde{n}_t) \prod_{t \in T_I^p} \pi_t \cdots \prod_{t \in T_I^1} \pi_t \prod_{i=0}^{d-1} A^i \\ &= \langle \langle \langle P_0^\dagger(U_0)(\tilde{n}_0) \otimes P_1^\dagger(U_1)(\tilde{n}_1), B_0 \rangle, \\ &\langle P_2^\dagger(U_2)(\tilde{n}_2) \otimes P_3^\dagger(U_3)(\tilde{n}_0), B_1 \rangle, \cdots, B_{d-2} \rangle, B_{d-1} \rangle \end{aligned} \tag{6}$$

Furthermore, we proposed an element-wise parallelization algorithm, demonstrating that HT-FBR can be reformulated in a chain-of-operators form. [5] This structure allows us to optimize the entire process using a deep learning–inspired optimization strategy.[5]

Although previous works successfully approximate high-dimensional functions using only a dimension-dependent number of sampled points, several challenges still hinder further reduction of the loss in high-dimensional applications. First, the computational cost associated to the evaluation of increasingly high-dimensional functions. As a result, we are often forced to reduce the number of training epochs, which implicitly limits the ability to further optimize the model toward better solutions. Second, a mild overfitting resulting from the combination of high-dimensionality and the limited size of the training set, in spite of the model's efficiency in capturing the underlying structure. And third, limitations intrinsic to the use of a specific framework. Indeed, relying solely on existing features of a framework is no longer sufficient to meet the performance requirements of large-scale computation.

For all these reasons, a highly parallelized and customized algorithm that can fully leverage the capabilities of modern high-performance GPUs becomes essential. In this work, we present our approach to address these challenges. First, in section 2, we briefly introduce the theory underlying the HT-FBR form and its optimization. Then, in Section 3, we discuss the details of its implementation and outline our approach to training with multiple GPUs. Finally in 4, we present our comparisons of our GPU-based approach with CPU parallelization and some benchmarks.

2 Theory

A binary tree is a hierarchical data structure in which each node t has at most two child nodes and a single parent node. The leaf nodes, denoted by $T_{\mathfrak{L}}$, are those that have no child node. The interior nodes, denoted by $T_{\mathfrak{I}}$, are nodes that have at least one child and are not the root. In the original work of Hierarchical Tucker Decomposition [6], the author used a Tucker Tree $T_{\mathcal{I}}$ as an extension of a binary tree in which each node is associated with a subset of mode of dimensions, referred to as the **node mode** denoted as $t = \{0, 1, \cdots, \mu \leq d-1\}$. The node modes are recursively divided in a left-major order from the root to the leaves, such that each split roughly halves the parent's node mode. Upper Fig. 1 illustrates a Tucker tree corresponding to a four-

dimensional tensor, or equivalently, a four-dimensional multivariate function in the context of approximation of analytical functions in L^2.

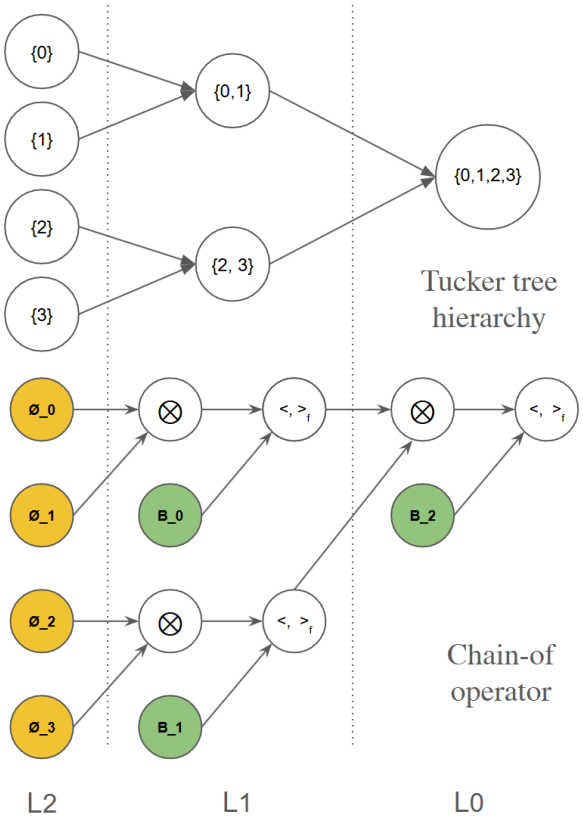

Figure 1: **Upper**: Example of Hierarchical Tucker tree associated to 4D function. **Down**: Chain rule format of HT-FBR.

In contrast to the original HTD approach that relies on the singular value decomposition (SVD) of a dense and matricizable tensor, in HT-FBR we use a functional schema as an analytical fitting of high-dimensional function without imposing any prior knowledge or structural assumption.

First, let $I_\mu \in L^2(\Omega_\mu)$ be a set of coordinates in $L^2(\Omega_\mu)$ on the μth dimension of the function $f \in L^2(\Omega_0 \times \cdots \times \Omega_\mu \times \cdots \Omega_{d-1})$, where Ω_μ is defined as follows:

$$\Omega_\mu \in [a_\mu \in \mathbb{R}, b_\mu \in \mathbb{R}] \quad \mu \in \{0, \ldots, d-1\} \tag{7}$$

The combined coordinate set $\mathcal{I}_t$ to the node t and its complete set $\mathcal{I}_{t'}$ is shown below:

$$\mathcal{I}_t := \times_{\mu \in t} \mathcal{I}_\mu \tag{8}$$

$$\mathcal{I}_{t'} := \times_{\mu \in t'} \mathcal{I}_\mu \tag{9}$$

We denote the factor associated with a leaf node t as $U_t \in \mathbb{R}^{\Omega_t \times k_t}$, which consists of the first k_t orthogonal basis in $L^2(\Omega_t)$, typically initialized randomly. The term k_t is often referred to "node rank" at node t as in HTD.

Let's define $\otimes_f$ an extension of the Kronecker product to L^2 space:

$$\begin{aligned} \otimes_f : \mathbb{R}^{\Omega_l \times k_l} \times \mathbb{R}^{\Omega_r \times k_r} &\mapsto \mathbb{R}^{(\Omega_l \times \Omega_r) \times (k_l \times k_r)} \\ (U_l \otimes_f U_r)(\phi, x)(\psi, y) &:= U_l(\phi, x) \cdot U_r(\psi, y) \end{aligned} \tag{10}$$

Assume $B_t \in \mathbb{R}^{(k_l \times k_r) \times k_t}, t \notin T_{\mathfrak{L}}$ a so-called "transfer tensor" as trainable parameter. Define an inner product $< ., . >_f$ corresponded:

$$\begin{aligned} p \in \mathbb{R}^{\Omega_t \times (k_l \times k_r)}, q \in \mathbb{R}^{(k_l \times k_r) \times k_t} \\ < p, B_t >_f := \sum_{i \in n_t} \sum_{j \in n_T} \int_{\Omega_t} p(x, i) B_t(i, j) d\mu(x) \end{aligned} \tag{11}$$

With the operators and factor/transfer tensors defined above, we now describe the leaves-to-root propagation using $\{U_{t \in T_{\mathfrak{I}}}, B_{t \notin T_{\mathfrak{I}}}\}$. Let U_l and U_r denote the factors (basis functions in the leaf nodes), or resulting representations (logits) associated with the two child nodes of an interior node t. Then, the truncation at node t is defined as:

$$\mathcal{I}_t = \mathcal{I}_l \times \mathcal{I}_r \tag{12}$$

$$U_t(\mathcal{I}_t) =< U_l(\mathcal{I}_l) \otimes_f U_r(\mathcal{I}_r), B_t >_f \tag{13}$$

Repeat Eq. 13 recursively from the leaf nodes to the root, we have the final representation of an approximation of f_A:

$$\begin{aligned} f_A(\mathcal{I}^d) \approx \langle \langle \langle P_0(U_0)(\mathcal{I}_0) \otimes_f P_1(U_1)(\mathcal{I}_1), B_0 \rangle_f, \\ \langle P_2(U_2)(\mathcal{I}_2) \otimes_f P_3(U_3)(\mathcal{I}_3), B_1 \rangle_f, \cdots, B_{d-2} \rangle_f, B_{d-1} \rangle_f \end{aligned} \tag{14}$$

We show the computation graph of a 4D sample in the down figure of 1. Such structure forms a chain-of-operators, enabling a neural network–like optimization of the transfer tensors B_t. Given a batched input of coordinates $\mathcal{I}$ and corresponding label values $f(\mathcal{I})$, the truncated output $f_A(\mathcal{I})$ by HT-FBR is evaluated using a loss function. The resulting gradient is then propagated backward through the tree structure to update B_t.

3 Implementation and parallelized optimization

In [4], we proposed a single-element passing rule that turns the evaluation of a HT-FBR structure into an embarrasingly parallel process. 14. In addition to this, an algorithm for computing the analytical gradient of parameters has been introduced in [5].

By integrating these two techniques, we have enabled GPU-based parallel forward and backward propagation via a Torch Script. To this end, we adopt the Nuwa framework from [7] to manage task dependencies between nodes. To improve data loading efficiency, we replace PyTorch's default data pipeline with a customized implementation using native Python code for both dataset creation and data loading process. However, we still employ PyTorch's sampler to perform randomized data sampling in the multi-GPU training. In the followingwe will denote our implementation as "nuwa" while the plain PyTorch implementation as "benchmark".

We adopt the same all-reduced strategy as used in [8]. The HT-FBR model is initialized and replicated across multiple GPUs, with each replica maintaining identical weights and hyperparameters. The training data is split into n mini-batches corresponding to the number of GPUs and distributed accordingly. Each replica performs a forward propagation on its local batch to compute logits, followed by a backward pass where gradients are computed and averaged across all devices to ensure synchronized updates usually referred to as Loosely synchronous parallelism. We present this DDP workflow in Fig. 2 where $n = 4$.

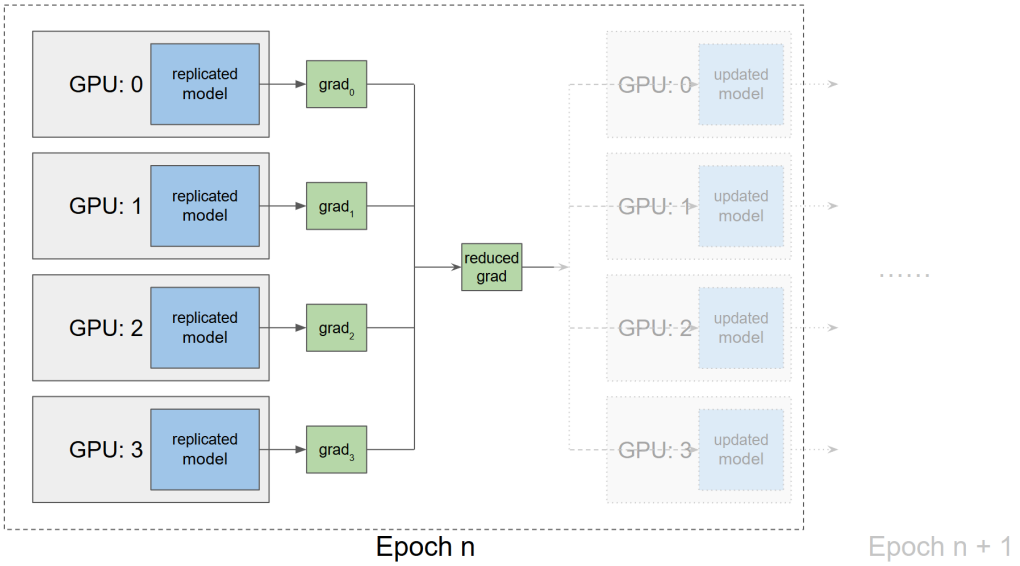

Figure 2: The DDP framework performs gradient reduction by averaging gradients across all participating devices, after which the synchronized gradients are used to update each replica of the distributed model.

4 Results and discussion

In this section, we present a comparative of the performance of various parallelization strategies for HT-FBR, concerning their convergence properties, computation time, and implementation efficiency compared to established benchmarks. To ensure con-

sistency and continuity with our previous works, we adopt the same dataset [9] as in [3][4] as benchmarks. In addition, we employ a 12-dimensional medium large Ethene system to intuitively demonstrate the parallel capability of our implementation. All CPU benchmarks were conducted using Python 3.10 and NumPy 1.26.4, with single-node CPU parallelization achieved using Numba 0.60.0. For GPU benchmarks and our implementation of the HT-FBR optimization, we used plain PyTorch 2.5.0 features with CUDA 11.8 and GCC 11.2.0. The figures presented below were generated using Matplotlib 3.9.2. The computations have been run on standard Ruche (local cluster) GPU nodes with Intel Xeon Gold 6230 20C @ 2.1GHz and Nvidia Tesla V100[10].

We adopted HONO PES dataset [9] as a proof-of-concept test that is widely adopted as a benchmark. Its analytical expression is defined as:

$$f_{\text{HONO}}(R_1, R_2, R_3, \theta_1, \theta_2, \tau) = \\ S_0 + e^{-D(R_1,R_2,R_3,\theta_1,\theta_2)} \times \sum_{ijklmn} c_{ijklm} Q_1^i Q_2^j Q_3^k Q_4^l Q_5^m \cos(nQ_6) \tag{15}$$

where $D = \sum_{i=1}^{3} d_i (R_i - R_i^{ref})^2 + \sum_{j=1}^{2} d_{j+3} (\theta_j - \theta_j^{ref})^2$ and $Q_{1/2/3} = 1 - e^{-0.7(R_{1/2/3} - R_{1/2/3}^{ref})}$, $Q_{4/5} = \theta_{1/2} - \theta_{1/2}^{ref}$, $Q_6 = \tau - \tau^{ref}$ with a set of known parameters R_1^{ref}, R_2^{ref}, R_3^{ref}, θ_1^{ref}, θ_2^{ref}, τ^{ref}.

The reference cloud of points are sampled on the isometric grid shows in Table 1. The total number of the samplings is 1e6. During optimization, both training set and test set consist of 10k randomly selected samplings. We use AdamW optimizer with learning rate 1e-1 and decays with a rate of 0.5 every 1,000 epochs, its minimum is set to be 1e-3. A full batch size is used during the training and k_t (node rank) for each node is set to be 10 except for the root is always be 1.

dim	n_μ	Ω_μ
0	10	[2.10, 3.25]
1	10	[1.30, 2.45]
2	10	[1.90, 2.60]
3	10	[-0.65, 0.25]
4	10	[-0.65, -0.10]
5	10	[0, π]

Table 1: Grid sampling on 6D f_{HONO}. 10 coordinates are equally taken on each dimension μ inside the domain Ω_μ

Fig 3 shows a comparison of the computation time required for our benchmark on 1, 2, 4 and 8 GPUs for a total of 100k epochs.The 8 GPUs configuration used two CPU nodes with each has 4 GPUs associated. We replaced Torch DataSet and DataLoader with our customized one, we used analytical gradient in the backward

instead of Torch Autograd. The other hyperparameters are kept identical in all the experiments. As shown below, our implementation gets a factor of 10 speedup when training on single GPU, it also achieves 5 times speedup on 8 GPUs when compared to the benchmark. On the other hand, the addition of a second node did not yield a significant improvement in computational efficiency.

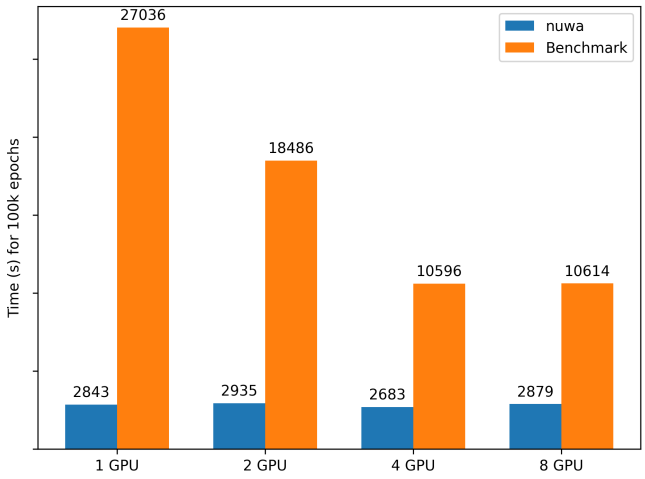

Figure 3: Comparison of computation time of nuwa and benchmark

Table 2 presents the results of models trained on both CPU and GPU. The CPU implementation is parallelized using two commonly used Python packages: NumPy and Numba, and it employs the Adam optimizer with an initial learning rate of 1e-1. While training losses converged to the 1e-4 level across all configurations, the use of DDP effectively mitigated overfitting, particularly when compared to training on a single CPU or GPU. As a comparison, a direct decomposition of HT-FBR with the same node ranks yields a global error of 7.5e-4 [3].

HONO global fitting CPU and GPU				
device	framework	train loss	test loss	time
1 CPU	nuwa	3.7e-4	7.2e-4	5h57m
1 GPU	**nuwa**	**6.1e-4**	**9.2e-4**	**47m18s**
1 GPU	benchmark	1.0e-3	1.3e-3	7h51m
2 GPU	nuwa	1.1e-3	1.3e-3	48m54s
2 GPU	**benchmark**	**4.9e-4**	**6.4e-4**	**5h8m**
4 GPU	**nuwa**	**5e-4**	**6.0e-4**	**44m42s**
4 GPU	benchmark	5.9e-4	6.3e-4	2h56m
8 GPU	**nuwa**	**5.3e-4**	**6.1e-4**	**47m58s**
8 GPU	benchmark	5.7e-4	6.2e-4	2h56m

Table 2: Comparison of nuwa with benchmark training on 1, 2, 4 and 8 GPUs. Training with 4 and 8 GPUs exhibits more stable behavior compared to configurations with fewer GPUs, with slightly reduced overfitting observed.

In addition, we assess the degree of overfitting by analyzing the gap between the training and testing losses throughout the training process. We observed that this absolute differences between the training and test losses are smaller when using four or eight GPUs compared to using only one or two GPUs as shown in Tab. 3.

Absolute difference training-test		
GPU	nuwa	benchmark
1	3.1e-4	3e-4
2	2e-4	1.5e-4
4	1e-4	4e-5
8	8e-5	5e -5

Table 3: The absolute differences between the training and test losses

To further evaluate the scalability and effectiveness of our method, we conducted the GPU optimization of a 12D HT-FBR model using a dataset consisting in 42,907 scattered data points. These have been generated in a series of Gaussian wavepacket trajectories (in the electronic ground state of ethene) using the so-called Direct Dynamics Variational Multiconfiguration Gaussian method (DD-vMCG).[11].

As shown in Table 4, GPU exhibit substantial performance improvement in terms of loss and performance over CPU in processing high-dimensional and large-scale datasets. Specifically, while a single CPU completed only 8,846 epochs in 70 hours, nuwa with a single GPU was able to complete 100,000 epochs in 15 hours. Moreover, this extended training led to a reduction in loss by an order of magnitude (1e-3 to 1e-4). Despite exceeding the 24-hour runtime limit, training with benchmark converged to an equivalent level of accuracy as observed in shorter runs. Experiments with 2, 4, and 8 GPUs yielded results consistent within the proof-of-concept case, where Nuwa achieved an approximate 2× speedup. also presents the computation time required to complete 100k epochs. Using the same total number of GPUs distributed across more nodes does not improve performance on the benchmark, although a slight improvement is observed for nuwa. On the other hand, with the dataset held constant, the use of more GPUs facilitated convergence to marginally lower loss values, while the overall training loss curve showed minimal variation.

Finally, a more detailed view of the training behavior is presented in Fig. 4. Nuwa and the benchmark exhibit nearly identical performance. Moreover, increasing the number of GPUs—while keeping the hyperparameter settings unchanged—does not affect the convergence behavior during training.

12D Ethene fitting CPU and GPU				
device	framework	train loss	epoch	time
1 CPU	nuwa	1.59e-3	8,846	70h
1 GPU	**nuwa**	**3.1e-4**	**100k**	**14h46m**
1 GPU	benchmark	3.3e-4	88,999	24h
2 GPU	**nuwa**	**3.1e-4**	**100k**	**6h57m**
2 GPU	benchmark	3.1e-4	100k	16h19m
4 GPU	**nuwa**	**2.8e-4**	**100k**	**4h30m**
4 GPU	benchmark	2.9e-4	100k	8h47m
8 GPU	**nuwa**	**2.8e-4**	**100k**	**4h**
8 GPU	benchmark	2.9e-4	100k	8h39m

Table 4: Benchmark comparison of nuwa. The training loss remains in the same level of accuracy for nuwa and benchmark. Training with either a single CPU or a single GPU failed to reach 100k epochs within the time constraints imposed by the computing cluster. An approximately 2x speedup is been observed on 4 and 8 GPUs by using nuwa.

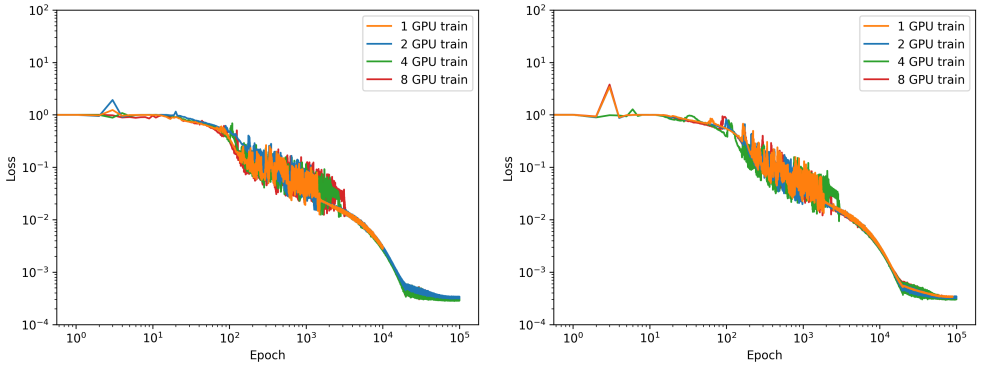

Figure 4: Upper and down figure present the training loss curve using nuwa and benchmark. Both of them exhibit same dynamic.

5 Conclusions

In this work, we have shown how HT-FBR analytical tree structures allow for Deep Learning-inspired optimization schemes.The HT-FBR training routine has been implemented in a multi-GPU environment. As main result, we have observed that the use of more GPUs with Distributed Data Parallel (DDP) not only reduces the training loss but also significantly mitigates overfitting. To further improve computational efficiency, we adopted two strategies that are (1) the use of analytical gradients during backpropagation, and (2) a customized data pipeline that replaces PyTorch's built-in data loading mechanism. We observe a speedup ranging from 2× to 10× on both the proof-of-concept and real-world datasets, while maintaining comparable levels of

accuracy. As part of our future work, we plan to implement multi-GPU model parallelism in conjunction with our current DDP approach to further reduce GPU memory usage and enable testing on higher-dimensional multivariate functions.

References

[1] Ramón Panadés-Barrueta and Daniel Peláez. "Low-rank sum-of-products finite-basis-representation (SOP-FBR) of potential energy surfaces". In: *The Journal of Chemical Physics* 153 (Nov. 2020), p. 234110. DOI: 10.1063/5.0027143.

[2] Nataša Nadoveza et al. "Analytical high-dimensional operators in Canonical Polyadic Finite Basis Representation (CP-FBR)". In: *The Journal of Chemical Physics* 158 (Feb. 2023). DOI: 10.1063/5.0139224.

[3] Zixing Qiu, Frédéric Magoulès, and Daniel Peláez. "Analytical Hierarchical Tucker Representation using Binary Trees". In: *P. Ivanyi, J. Kruis, B.H.V. Topping, (Editors), "Proceedings of the Seventeenth International Conference on Civil, Structural and Environmental Engineering Computing"* CCC 6.13.4 (2023). DOI: 10.4203/ccc.6.13.4. URL: https://doi.org/10.4203/ccc.6.13.4.

[4] Z. Qiu, F. Magoulès, and D. Peláez. "Single-entry computation of analytical hierarchical (binary) tree structures". In: *Advances in Engineering Software* 203 (2025), p. 103873. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2025.103873. URL: https://www.sciencedirect.com/science/article/pii/S0965997825000110.

[5] Z. Qiu, F. Magoulès, and D. Peláez. "Multidimensional regression through single-entry evaluation of analytical hierarchical tree structures using clouds of data-points (submitted)". In: *The Journal of Chemical Physics* (2025).

[6] Lars Grasedyck. "Hierarchical Singular Value Decomposition of Tensors". In: *SIAM Journal on Matrix Analysis and Applications* 31.4 (2010), pp. 2029–2054. DOI: 10.1137/090764189. eprint: https://doi.org/10.1137/090764189. URL: https://doi.org/10.1137/090764189.

[7] Zixing Qiu. *Nuwa0.0.1*. https://github.com/raulniconico/Nuwa0.0.1. Accessed: 2025-04-09. 2022.

[8] PyTorch Team. *torch.nn.parallel.DistributedDataParallel - PyTorch Documentation*. https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html. Accessed: 2025-04-07. 2024.

[9] Falk Richter et al. “A study of the mode-selective trans–cis isomerization in HONO using ab initio methodology”. In: *The Journal of Chemical Physics* 120.3 (Jan. 2004), pp. 1306–1317. ISSN: 0021-9606. DOI: 10.1063/1.1632471. eprint: https://pubs.aip.org/aip/jcp/article-pdf/120/3/1306/10857625/1306_1_online.pdf. URL: https://doi.org/10.1063/1.1632471.

[10] *Moulon mesocentre*. http://https://tex.stackexchange.com/questions/3587/how-can-i-use-bibtex-to-cite-a-web-page. Accessed: 2024-01-01.

[11] Sandra Gómez, Eryn Spinlove, and Graham Worth. “Benchmarking non-adiabatic quantum dynamics using the molecular Tully models”. In: *Phys. Chem. Chem. Phys.* 26 (3 2024), pp. 1829–1844. DOI: 10.1039/D3CP03964A. URL: http://dx.doi.org/10.1039/D3CP03964A.