Introduce Nvidia Modulus Symbolic (Modulus Sym)

Compare with traditional simulation and NVIDIA Modulus

Compared to traditional simulations, NVIDIA Modulus offers several benefits that leverage AI techniques to enhance and streamline the process of configuring, building, and training models for physical systems across various domains. Here's a breakdown of these benefits:

AI Toolkit for Physics: Modulus provides an AI toolkit that allows users to easily create and develop AI models for physical systems. This toolkit is versatile and can be applied to a wide range of domains, from engineering simulations to life sciences. The toolkit is accessible through simple Python APIs, making it user-friendly and accessible to a broad audience.

Customizable Models: Users can build upon and customize pretrained AI models available in the NVIDIA NGC catalog. This saves time and effort by allowing users to leverage state-of-the-art models as a starting point and then adapt them to their specific use cases and requirements.

Scalability with NVIDIA AI: Modulus takes advantage of NVIDIA's AI capabilities to scale training performance. This means that AI models developed using Modulus can be efficiently trained across different hardware setups, from single GPUs to multi-node clusters, ensuring optimal performance and faster training times.

Open-Source Design: Modulus is built on top of the PyTorch framework and is released under the Apache 2.0 open-source license. This design choice promotes transparency, collaboration, and community involvement, allowing users to contribute, modify, and extend the toolkit as needed.

Standardized AI Development: Modulus follows best practices for AI development specifically tailored to physics-based machine learning models. This approach ensures that AI models are developed with a strong focus on accuracy, reliability, and relevance to engineering applications.

Overall, NVIDIA Modulus aims to bridge the gap between traditional simulations and AI-driven approaches by providing a user-friendly toolkit that allows researchers, engineers, and scientists to harness the power of AI to enhance their understanding of physical systems. With its customizable models, scalability, open-source design, and emphasis on standardized development, Modulus offers a comprehensive solution for those seeking to integrate AI techniques into their domain-specific simulations and analyses.

Nvidia Modulus

NVIDIA Modulus brings together the world of physics, which involves special math equations (known as PDEs), along with information about boundaries and training data. This combination is used to create advanced, customizable, pretend models using deep learning. The platform takes away the complicated setup process for training these models, so you can use your expertise in a particular field to guide the creation of smart models and design improved ways for computers to understand things. There are also examples available that can help you get started on using the same ideas for new situations.

If you're a researcher aiming to create new AI methods for innovative engineering and scientific simulations, or if you're an engineer seeking to speed up design improvements and digital twin applications, the Modulus platform is a valuable resource for advancing your model development. Modulus provides a range of techniques to train neural network models based on physics principles. These approaches include models solely driven by physics, which integrate physics knowledge through physics-informed neural networks (PINNs), as well as models that blend physics concepts with real-world data, like the physics-oriented, data-driven designs seen in neural operators.

NVIDIA Modulus addresses two significant challenges:

1. **Parameterized Problems:** This involves defining geometry data with parameters, such as an object's dimensions, conductivity, and more. The neural network solver learns various combinations of these parameterized inputs, allowing for convenient adjustments to input values and subsequent simulations.
2. **Inverse Problems:** Modulus also proves invaluable in solving inverse problems. In these scenarios, the process begins with a set of observations, and the goal is to determine the underlying factors that led to those observations. The neural network solver aids in deriving the causal factors from the given observations.

Modulus architecture:

Modulus Core forms the foundational module, incorporating essential components of the framework necessary for creating Physics-ML models.

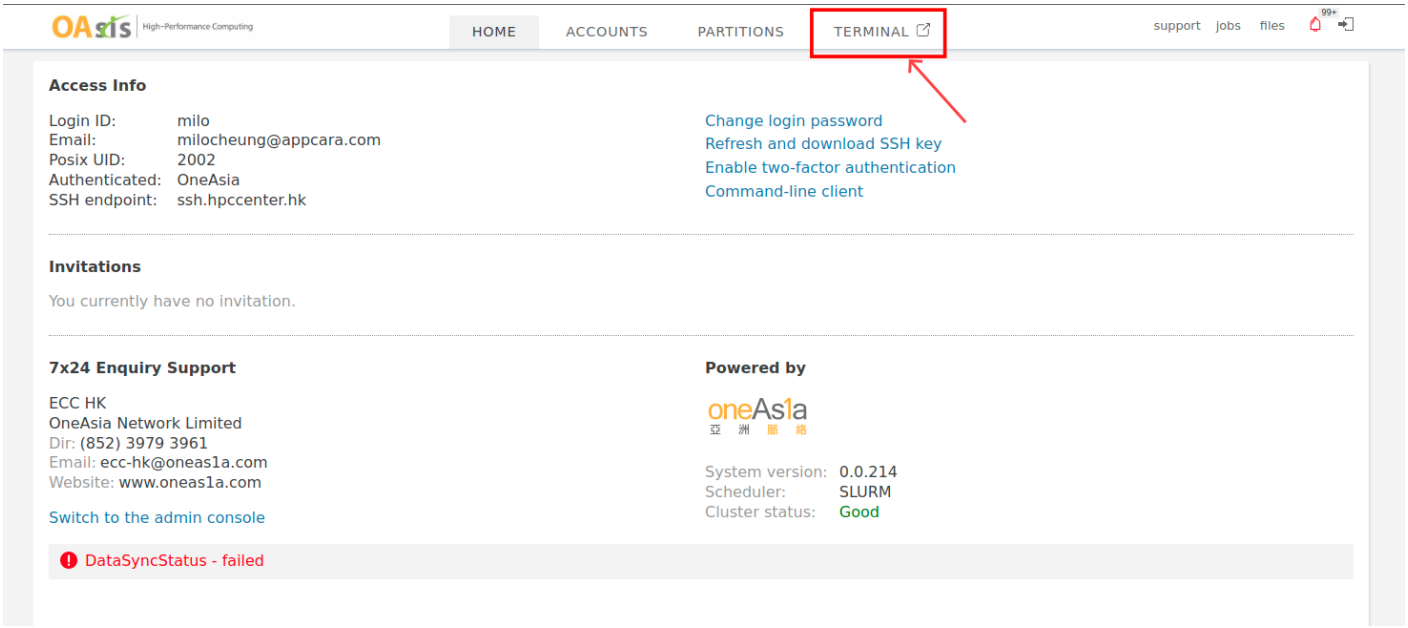
Modulus Sym introduces an abstraction layer designed to facilitate the utilization of PDE-based symbolic loss functions.

Modulus Launch provides optimized training recipes for data driven Physics-ML models.

Setting up Nvidia Modulus Sym in OAsis

In this guide, we'll use Modulus Sym as an illustrative example.

Login into OAsis, select "**TERMINAL**"



Execute the following commands:

```
# create job with gpu resource
srun -p gpu --cpus-per-task=4 --mem=16G --gres=gpu:3g.40gb:1 --pty bash

# load required modules
module load Anaconda3/2022.05
module load CUDA/11.8
module load GCCcore/11.3.0 git/2.36.0-nodocs
module load git-lfs/3.2.0

# create conda environment with python version 3.8
conda create -n modulus-symbolic python=3.8
source activate modulus-symbolic

# install required package in "modulus-symbolic" conda environment
conda install pip ipykernel
conda install gxx_linux-64
conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia
pip install blosc2==2.0.0 cython protobuf pyqtwebengine pyqt5 blosc2 cmake lit
pip install nvidia-modulus.sym tensorboard
```

After successfully setting up the conda environment, you can load the module and activate the environment effortlessly by following commands:

```
module load Anaconda3/2022.05
module load CUDA/11.8
```

```
source activate modulus-symbolic
```

Clone the source code and run an example.

```
cd $HOME
git clone https://github.com/NVIDIA/modulus-sym.git
cd $HOME/modulus-sym/examples/chip_2d/
python chip_2d.py

# To reduce the training time, you can adjust the max_steps parameter in the conf/config.yaml file to a lower value, such as 5000.
# This change will limit the number of training steps the model undergoes, thereby speeding up the training process.
```

Edit the conf/config.yaml file according to adjust the traning max_steps.

```
# Copyright (c) 2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

defaults :
  - modulus_default
  - arch:
      - fully_connected
  - scheduler: tf_exponential_lr
  - optimizer: adam
  - loss: sum
  - _self_

scheduler:
  decay_rate: 0.95
  decay_steps: 350

training:
  rec_results_freq : 1000
  rec_constraint_freq: 10000
  max_steps : 5000

batch_size:
  inlet: 64
  outlet: 64
  no_slip: 2800
  interior_lr: 1000
  interior_hr: 1000
  integral_continuity: 512
  num_integral_continuity: 4
~
~
~
```

Finished Training .

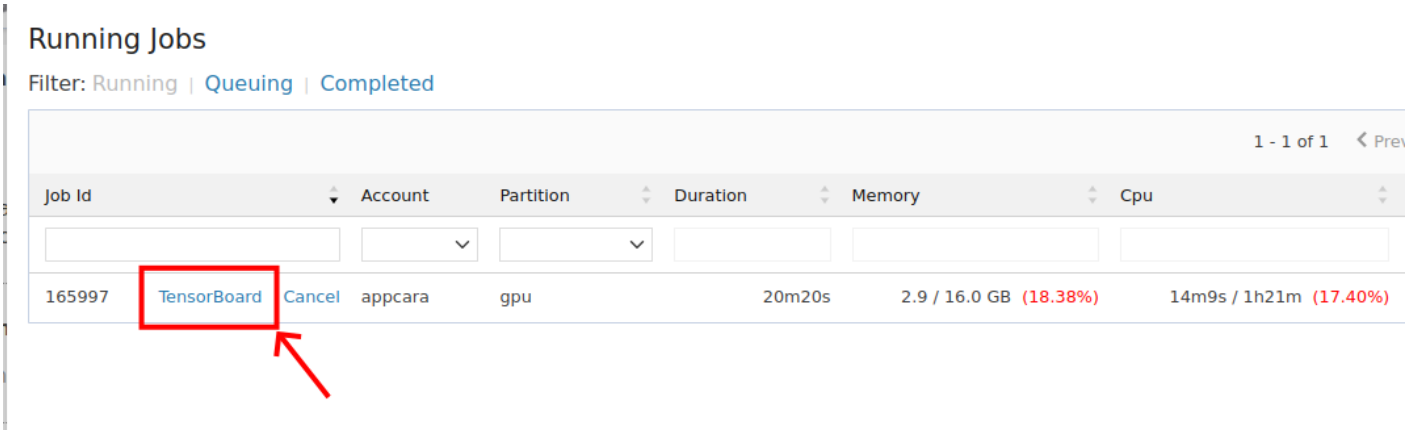
```
  warnings.warn(
[10:46:54] - Installed PyTorch version 2.0.1 is not TorchScript supported in Modulus. Version 1.14.0a0+410ce96 is officially supporte
[10:46:54] - attempting to restore from: outputs/chip_2d
[10:46:54] - optimizer checkpoint not found
[10:46:54] - model flow_network.0.pth not found
[10:46:59] - [step:          0] record constraint batch time:  1.030e-01s
[10:46:59] - [step:          0] saved checkpoint to outputs/chip_2d
[10:46:59] - [step:          0] loss:  5.002e+00
[10:47:01] - Attempting cuda graph building, this may take a bit...
[10:47:04] - [step:        100] loss:  1.822e+00, time/iteration:  4.802e+01 ms
[10:47:08] - [step:        200] loss:  1.245e+00, time/iteration:  3.893e+01 ms
[10:47:12] - [step:        300] loss:  6.752e-01, time/iteration:  3.897e+01 ms
[10:47:16] - [step:        400] loss:  6.896e-01, time/iteration:  3.890e+01 ms
[10:47:20] - [step:        500] loss:  5.911e-01, time/iteration:  3.888e+01 ms
[10:47:24] - [step:        600] loss:  8.701e-01, time/iteration:  3.896e+01 ms
[10:47:27] - [step:        700] loss:  4.957e-01, time/iteration:  3.892e+01 ms
[10:47:31] - [step:        800] loss:  3.339e-01, time/iteration:  3.895e+01 ms
[10:47:35] - [step:        900] loss:  4.792e-01, time/iteration:  3.892e+01 ms
[10:47:40] - [step:       1000] saved checkpoint to outputs/chip_2d
[10:47:40] - [step:       1000] loss:  3.356e-01, time/iteration:  4.399e+01 ms
[10:47:44] - [step:       1100] loss:  3.315e-01, time/iteration:  3.888e+01 ms
[10:47:47] - [step:       1200] loss:  3.620e-01, time/iteration:  3.886e+01 ms
[10:47:51] - [step:       1300] loss:  3.732e-01, time/iteration:  3.892e+01 ms
[10:47:55] - [step:       1400] loss:  3.488e-01, time/iteration:  3.886e+01 ms
[10:47:59] - [step:       1500] loss:  3.198e-01, time/iteration:  3.888e+01 ms
[10:48:03] - [step:       1600] loss:  2.903e-01, time/iteration:  3.891e+01 ms
[10:48:07] - [step:       1700] loss:  2.630e-01, time/iteration:  3.886e+01 ms
[10:48:11] - [step:       1800] loss:  1.777e-01, time/iteration:  3.886e+01 ms
[10:48:15] - [step:       1900] loss:  2.352e-01, time/iteration:  3.890e+01 ms
[10:48:19] - [step:       2000] saved checkpoint to outputs/chip_2d
[10:48:19] - [step:       2000] loss:  2.151e-01, time/iteration:  4.356e+01 ms
[10:48:23] - [step:       2100] loss:  2.050e-01, time/iteration:  3.886e+01 ms
[10:48:27] - [step:       2200] loss:  1.759e-01, time/iteration:  3.885e+01 ms
[10:48:31] - [step:       2300] loss:  1.156e-01, time/iteration:  3.880e+01 ms
[10:48:35] - [step:       2400] loss:  2.028e-01, time/iteration:  3.881e+01 ms
[10:48:38] - [step:       2500] loss:  1.773e-01, time/iteration:  3.885e+01 ms
[10:48:42] - [step:       2600] loss:  1.522e-01, time/iteration:  3.889e+01 ms
[10:48:46] - [step:       2700] loss:  1.645e-01, time/iteration:  3.887e+01 ms
[10:48:50] - [step:       2800] loss:  1.187e-01, time/iteration:  3.888e+01 ms
[10:48:54] - [step:       2900] loss:  1.912e-01, time/iteration:  3.893e+01 ms
[10:48:58] - [step:       3000] saved checkpoint to outputs/chip_2d
[10:48:58] - [step:       3000] loss:  1.812e-01, time/iteration:  4.335e+01 ms
[10:49:02] - [step:       3100] loss:  1.301e-01, time/iteration:  3.916e+01 ms
[10:49:06] - [step:       3200] loss:  1.117e-01, time/iteration:  3.883e+01 ms
[10:49:10] - [step:       3300] loss:  1.295e-01, time/iteration:  3.889e+01 ms
[10:49:14] - [step:       3400] loss:  1.715e-01, time/iteration:  3.890e+01 ms
[10:49:18] - [step:       3500] loss:  1.010e-01, time/iteration:  3.888e+01 ms
[10:49:22] - [step:       3600] loss:  1.053e-01, time/iteration:  3.888e+01 ms
[10:49:26] - [step:       3700] loss:  1.870e-01, time/iteration:  3.887e+01 ms
[10:49:29] - [step:       3800] loss:  8.633e-02, time/iteration:  3.895e+01 ms
[10:49:33] - [step:       3900] loss:  9.028e-02, time/iteration:  3.888e+01 ms
[10:49:38] - [step:       4000] saved checkpoint to outputs/chip_2d
[10:49:38] - [step:       4000] loss:  1.011e-01, time/iteration:  4.372e+01 ms
[10:49:42] - [step:       4100] loss:  1.181e-01, time/iteration:  3.886e+01 ms
[10:49:45] - [step:       4200] loss:  1.019e-01, time/iteration:  3.884e+01 ms
[10:49:49] - [step:       4300] loss:  5.567e-02, time/iteration:  3.890e+01 ms
[10:49:53] - [step:       4400] loss:  7.803e-02, time/iteration:  3.890e+01 ms
[10:49:57] - [step:       4500] loss:  8.845e-02, time/iteration:  3.888e+01 ms
[10:50:01] - [step:       4600] loss:  8.558e-02, time/iteration:  3.890e+01 ms
[10:50:05] - [step:       4700] loss:  1.845e-01, time/iteration:  3.888e+01 ms
[10:50:09] - [step:       4800] loss:  9.646e-02, time/iteration:  3.892e+01 ms
[10:50:13] - [step:       4900] loss:  8.078e-02, time/iteration:  3.893e+01 ms
[10:50:17] - [step:       5000] saved checkpoint to outputs/chip_2d
[10:50:17] - [step:       5000] loss:  5.511e-02, time/iteration:  4.340e+01 ms
[10:50:17] - [step:       5000] reached maximum training steps, finished training!
(modulus-symbolic) [milc@oaklbhpcgpu001 chip_2d]$
```

Examine the trained model using the **TensorBoard** Web UI.

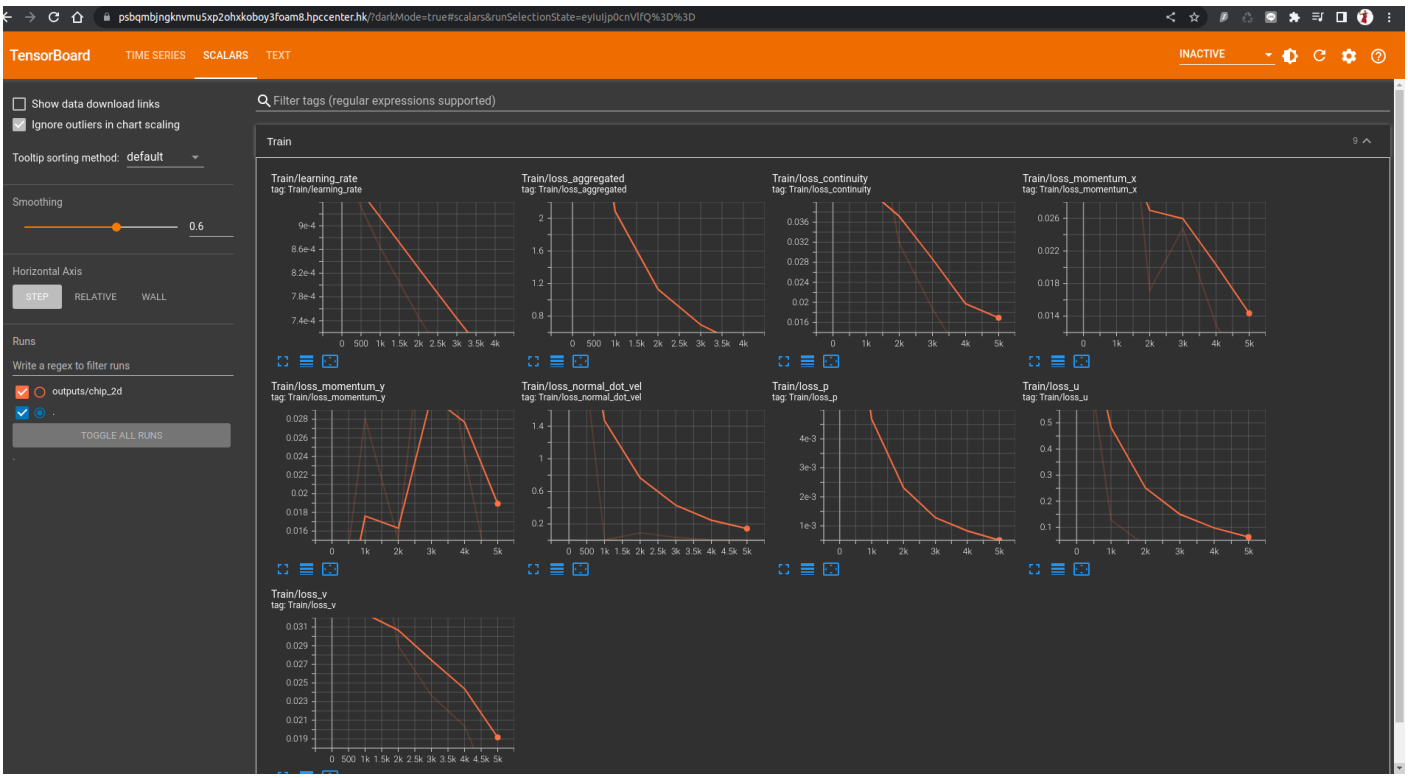
```
# type following command to bind hostname, port to tensorboard
host=$(hostname)
port=$(hc acquire-port -j $SLURM_JOB_ID -u web --host $host -l TensorBoard)

# you can edit logdir to other directory
tensorboard --logdir . --host $host --port $port
```

Click the "TensorBoard" button to access the web-based user interface.



Navigate to the "SCALARS" tab to delve into comprehensive training insights.



This tutorial elucidates how to set up and utilize Nvidia Modulus Sym within the OAsis platform, showcasing the potential of merging physics and AI to achieve exceptional model performance.

Revision #12
Created 7 August 2023 06:17:00 by Milo Cheung
Updated 17 August 2023 09:59:28 by Milo Cheung