Parallel - 360° Projector

Keshav Bhandari & Apan Qasem

***Abstract*-Distortion variants operations like convolution, pooling and interpolation in DNN(Deep Neural Network) create challenges in omni directional computer vision applications. There are several extensions for these operations. One of the straightforward approach is to carry out these operations in spherical domain. The key ideas are 3 folds, project, operate and back-project. We offer a parallelable 360-projector to achieve this task. We contrast its performance against serial implementation. We base this parallel implementation on pytorch cuda extensions and can be easily differentiable. We can easily extend this implementation to perform task like parameter learning, which is useful for fine tuning.**[1]

Introduction

Computer vision applications in omni-directional videos/images are relatively newer field. Majority of computer vision applications exploit perspective images/videos. These perspective images/videos have a limited field of view, whereas omni-directional videos/images provide 360°/omni-directional field of view. Compared to perspective data, these omnidirectional data provides whole new application spectrum.

The real bottleneck in omnidirectional based computer vision is transferability of existing DNN architecture. Omnidirectional data contains huge distortion compared to perspective data. This distortion result from projection(equirectangular[2]) from spherical domain to Euclidean domain. Most of the DNN architectures for computer vision are tailored for perspective videos/images, where distortion is not an issue. Meanwhile, operations like convolution(most sensitive), pooling, interpolation in DNN are distortion sensitive, for which existing architectures are not distortion invariant. There are many ways to deal with this

$$\begin{aligned} x &= \rho \sin\phi \cos\theta \\ y &= \rho \sin\phi \sin\theta \\ z &= \rho \cos\phi \end{aligned} \qquad \begin{aligned} \phi^2 &= x^2 + y^2 + z^2 \\ \theta &= \arctan\frac{y}{x} \\ \phi &= \arccos\frac{z}{\sqrt{\rho}} \end{aligned} \qquad \text{Eq. 1}$$

Spherical to Rectangular — Rectangular to Spherical

issue, like spherical convolution(1–3), convolution in frequency domain, kernel transformation techniques(4, 5) and so on. One of the straightforward way is to perform convolution in tangential plane. Instead of convolving over fine-grid, coarse-grid convolution performs similar with negligible performance loss which can be improved via techniques like transformer network. In this work our goal is to implement a parallel ***Projector-360°*** to perform this operation and contrast its speed-up with serial implementation. We define Projector as a function to map(shown in Eq-1, where θ, ϕ are polar and azimuthal angle respectively) equirectangular plane to $(|\Phi| \times |\Theta|)$ fov(field of view), perform convolution, pooling or interpolation and project back to equirectangular plane. Basically, these stages are 3 folds, project, operate and back-project. These fov-maps are made continuous by adding pads from neighboring projections. Here, Φ represents a discrete set of azimuthal angle within range $(-\pi, +\pi) or (-180°, +180°)$, similarly Θ represents a discrete set of polar angle within range $(-\frac{\pi}{2}, +\frac{\pi}{2}) or (-90°, +90°)$.

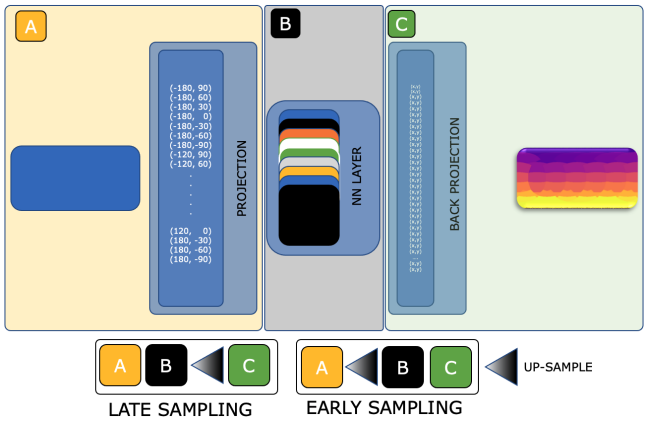


Fig. 1. Showing Projector 360°steps. Major stages are project(A), operate(B) and back-project(C). Project stage performs multiple tangential projections, Operate stage apply operations like convolution, pooling or interpolation and finally back project stage perform reverse projection from sphere to plan. Sampling is necessary for better performance. Sampling can be done before operation or after operation.

Experiment

Parallel-360°Projector has 3 primary stages, as shown in Fig-1. Stage - A, performs Spherical to Rectangular projection. Number of tangential projections in this stage is defined by the choice of sets of polar angles(Θ) and azimuthal angle(Φ) taken at a discrete interval from range (-90°,+90°) and (-180°,+180°) respectively. For example, in this experiment we choose two sets. First set $\Theta = \{-90, -60, ..., +90\}$ with interval 30°, $\Phi = \{-180, -120, ..., +180\}$ such that $|\Theta| = |\Phi| = 7$. In this way we have 49 different tangential projections. Similarly, we chose second set to have $|\Theta| = |\Phi| = 4$, which leads to 16 tangential projections. The former projections are finer than the later and have lesser distortion. However, it requires 3 times more computation compared to the later one.

Stage-B will apply operations(convolutions, pooling, or interpolation) on these tangential planes. Now the output of these operations are projected back to equirectangular plane

[1] https://github.com/keshavsbhandari/360projector

[2] Examining other projection technique is out of the scope for this project

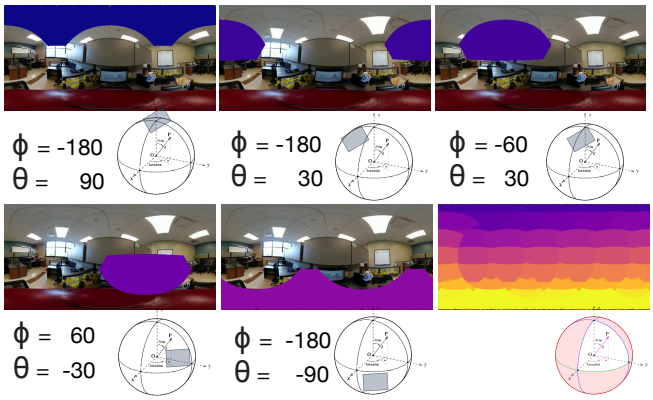


Fig. 2. Visualizing operations via Projector-360°. In this visualization we choose 9 tangential projections. We map different color to different tangential projections shown in bottom right figure to visualize mapping. Similarly, we can apply operations like convolution, pooling and interpolation and map back into the plane.

in stage-C. The mapping process from plane to tangent must sample enough points such that when we do back-projection to the plane, we don't want any sparsity. To fix this, we can increase the sampling rate. If we increase our sampling rate after stage-A, we call it early sampling. And if we increase sampling by scaling the output of stage-B we call it late sampling. Early sampling is compute intensive, as it has to do $(|\Theta| \times |\Phi|)$-operations(convolution is really expensive). An alternative approach, late-sampling can achieve similar performance with lesser computation, which makes our implementation much faster.

Projector-360° is based on pypi-package called py360Convert(6). We take this as a serial implementation baseline. We perform linear speed-up comparison with the baseline. Similarly, we also made a comparison between early sampling vs late-sampling technique. We also perform initialization speed-up between early and late sampling version.

Results. Our best pytorch based implementation(Fig-3, see numpy_vs_torch_late) is around 15 times faster than the serial implementation. Our best implementation torch_late implements late sampling techniques and huge sampling rate. We can analyse the effects of sampling from qualitative results presented in Supplementary Material section. We provide an optical flow estimation on 360 videos using (7, 8). We basically wrap pre-trained convolution layers with projector-360°. The results are impressive. Since these experiments are from ongoing work, we haven't provided experiment details regarding those implementations.

In Fig-3, we made initialization comparison as well. Initialization is done one time and should not be important. However, in typical DNN we have multiple layers that need transformation to deal with this distortion via projector-360°. We don't want our model to take 10 minutes to load. We load-unload model multiple times, and this is practically unuseful. Our implementation easily solves this problem by having significantly lesser initialization time.

Platform. We run our experiment on Ubuntu 18.04. It has Intel(R) Core(TM) i7-8700K CPU, and one 1080-Ti Nvidia

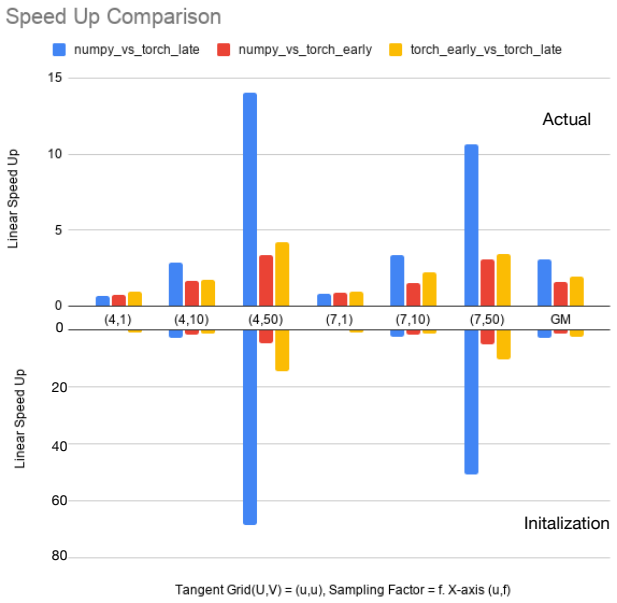


Fig. 3. Comparing speedup with serial implementation and among parallel versions. Top - Comparing actual project, map and back-project part. Bottom- Comparing initialization time.

GPU. We use torch 1.7 , numpy-1.19.0, cuda-11.0 and python 3.7. Further platform detail is presented in Supplementary material section.

Conclusion

Projector-360°is a simple but important tool for working with spherical data. While there are tons of custom implementation, our pytorch-cuda extension based implementation is fairly simple, faster and flexible, which we can extend for parameter learning as well. This is a work in progress and there are still enough room for improvements, which we will visit in future works.

Bibliography

1. Marc Eder and Jan-Michael Frahm. Convolutions on Spherical Images. *arXiv e-prints*, art. arXiv:1905.08409, May 2019.
2. Taco S. Cohen, Mario Geiger, Jonas Koehler, and Max Welling. Spherical CNNs. *arXiv e-prints*, art. arXiv:1801.10130, January 2018.
3. Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning SO(3) Equivariant Representations with Spherical CNNs. *arXiv e-prints*, art. arXiv:1711.06721, November 2017.
4. Yu-Chuan Su and Kristen Grauman. Learning Spherical Convolution for Fast Features from 360° Imagery. *arXiv e-prints*, art. arXiv:1708.00919, August 2017.
5. Yu-Chuan Su and Kristen Grauman. Kernel Transformer Networks for Compact Spherical Convolution. *arXiv e-prints*, art. arXiv:1812.03115, December 2018.
6. https://github.com/sunset1995. py360convert 0.1.0. December 2020.
7. Zachary Teed and Jia Deng. RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. *arXiv e-prints*, art. arXiv:2003.12039, March 2020.
8. Keshav Bhandari, Ziliang Zong, and Yan Yan. Revisiting Optical Flow Estimation in 360 Videos. *arXiv e-prints*, art. arXiv:2010.08045, October 2020.

Supplementary Material

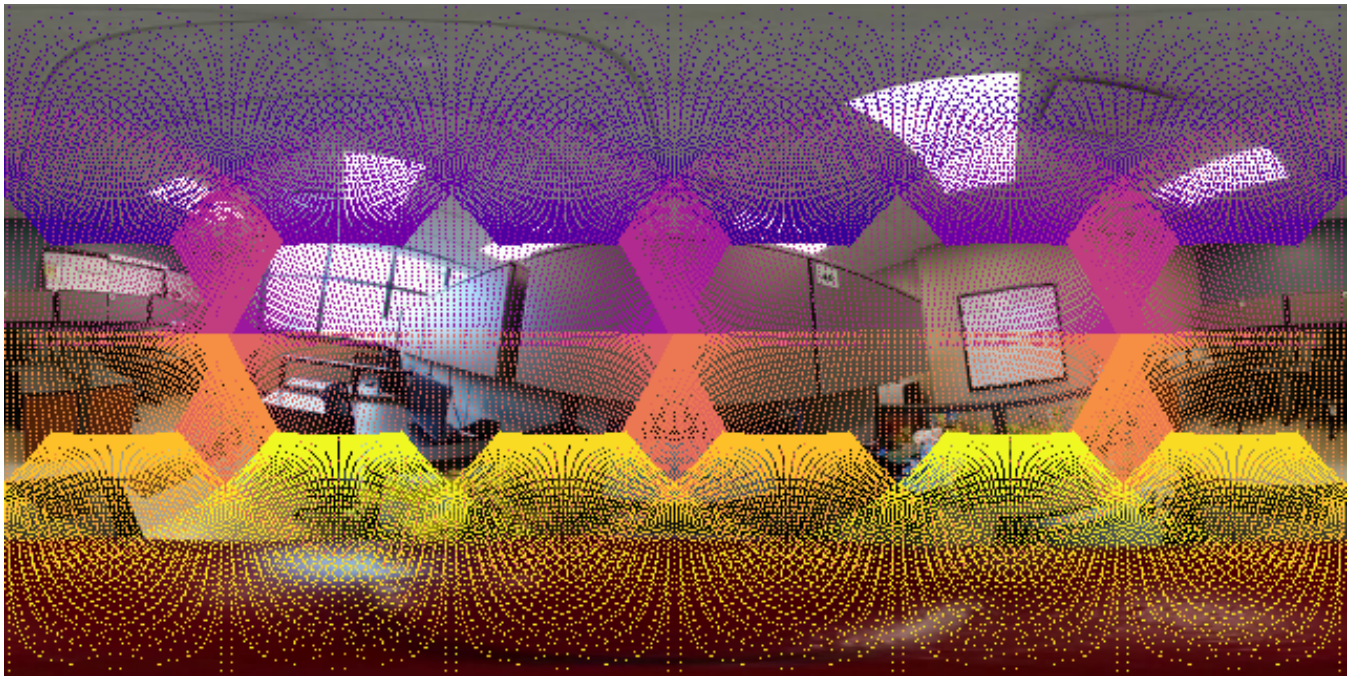


Fig. 4. Convolution mapping using a $(\Theta, \Phi) = (4, 4)$ grid to produce 16 tangential planes with a sampling rate of 1 shows sparse convolution map.

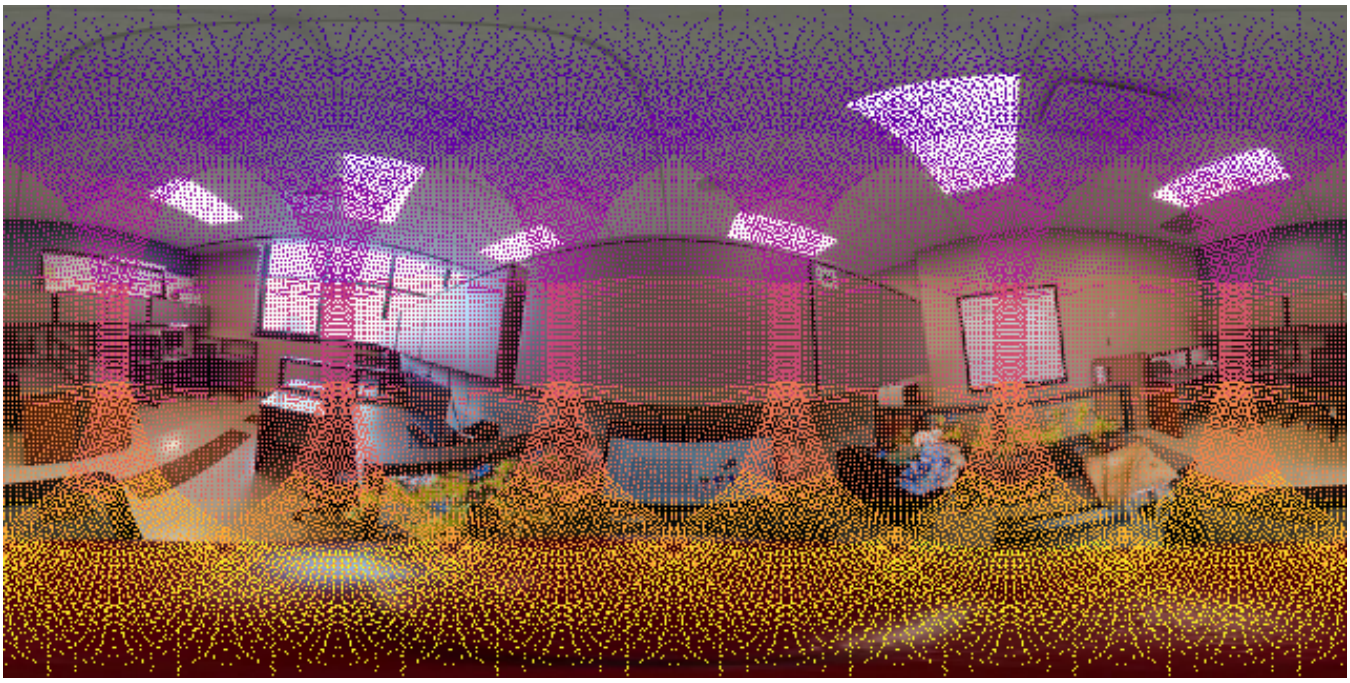
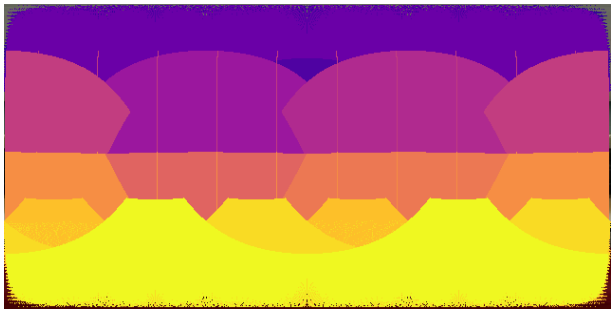
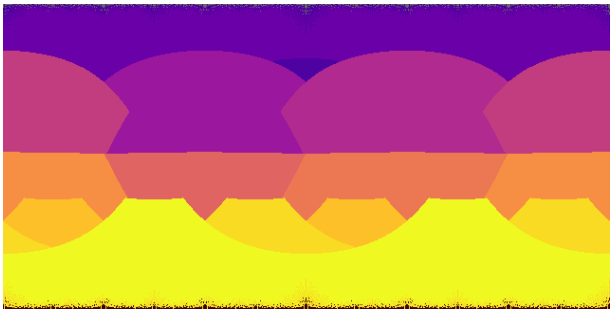


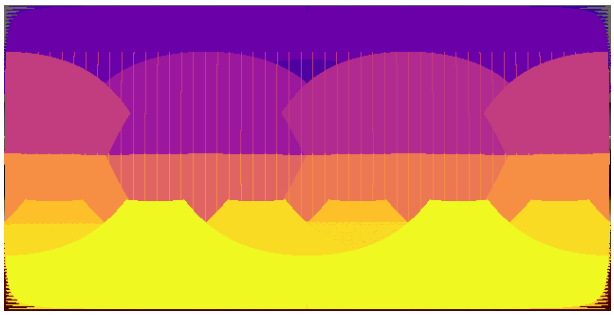
Fig. 5. Convolution mapping using a $(\Theta, \Phi) = (7, 7)$ grid to produce 49 tangential planes with a sampling rate of 1 still shows sparse convolution map.



UV-GRID (4,4), sampling_factor = 10, late



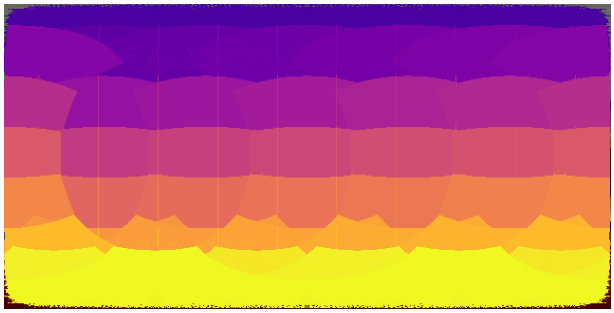
UV-GRID (4,4), sampling_factor = 10, early



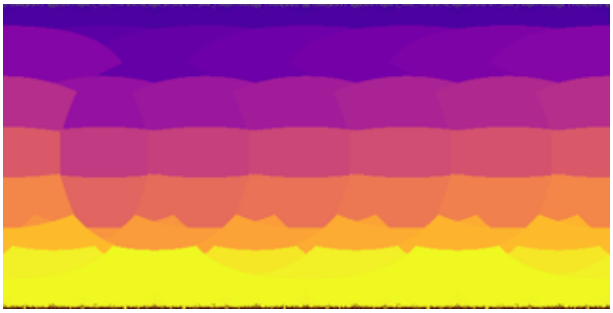
UV-GRID (4,4), sampling_factor = 50, late

UV-GRID (4,4), sampling_factor = 50, early

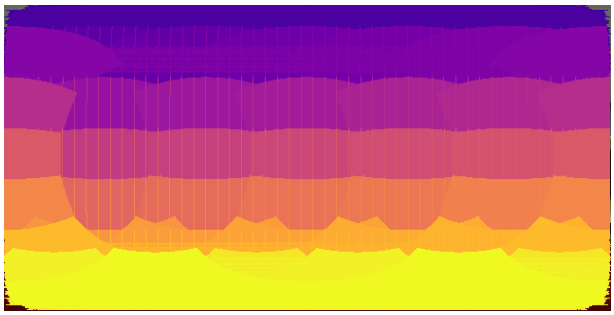
Fig. 6. Convolution mapping with 4 by 4 uv-grid is satisfactory. 4 by 4 uv-grid mapping with a sampling factor of 50 is comparitively better.



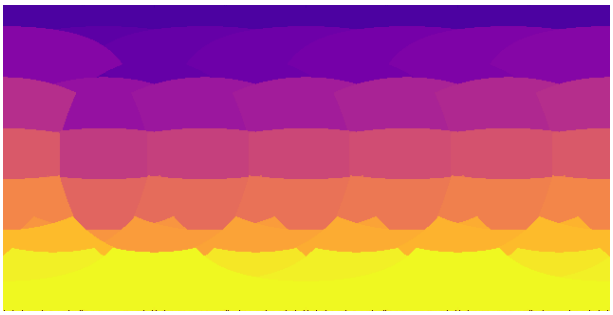
UV-GRID (7,7), sampling_factor = 10, late



UV-GRID (7,7), sampling_factor = 10, early



UV-GRID (7,7), sampling_factor = 50, late



UV-GRID (7,7), sampling_factor = 50, early

Fig. 7. Convolution mapping with 7 by 7 uv-grid is better than 4 by 4 uv-grid mapping. This mapping with 50 sampling factor is nearly perfect.

Fig. 8. Using (7, 8) with Projector360°shows significant improvement without further fine tuning. Left figure shows cube-map based flow, which shows significant flow discontinuity. Middle and the rights are stage-2 and stage-3 of (8) with (7) as back bone. We base this result without fine-tuning, which is really impressive.

```
# modules: projector360.nn
from projector360.utils import projector
import torch.nn as nn

class Conv360(nn.Module):
def __init__(self, *args, **kwargs):
      super(Conv360, self).__init__()
      self.proj_args = kwargs.pop('projector_args')
      self.conv360 = projector(
                                        nn.Conv2d(**kwargs),
                                        **self.proj_args
              )
def initialize(self, weight, bias):
      self.conv360.weight.data = weight
      self.conv360.bias.data = bias

def forward(self, x):
      return self.conv360(x)
...
```

```
# modules:perspectiveModel.py
from torch.nn import Conv2d
from torch.nn import MaxPool2d
from torch.nn.functional import interpolate
...
```

```
# modules:sphericalModel.py
from projector360.nn import Conv360 as Conv2d
from projector360.nn import MaxPool360 as MaxPool2d
from projector360.nn import Interpolate as interpolate
...
```

Fig. 9. Above code snippet shows a higher level aim of Projector-360°. This is a work in progress. Once we finalize the code, we push it as version-1 and will be available via pypi.

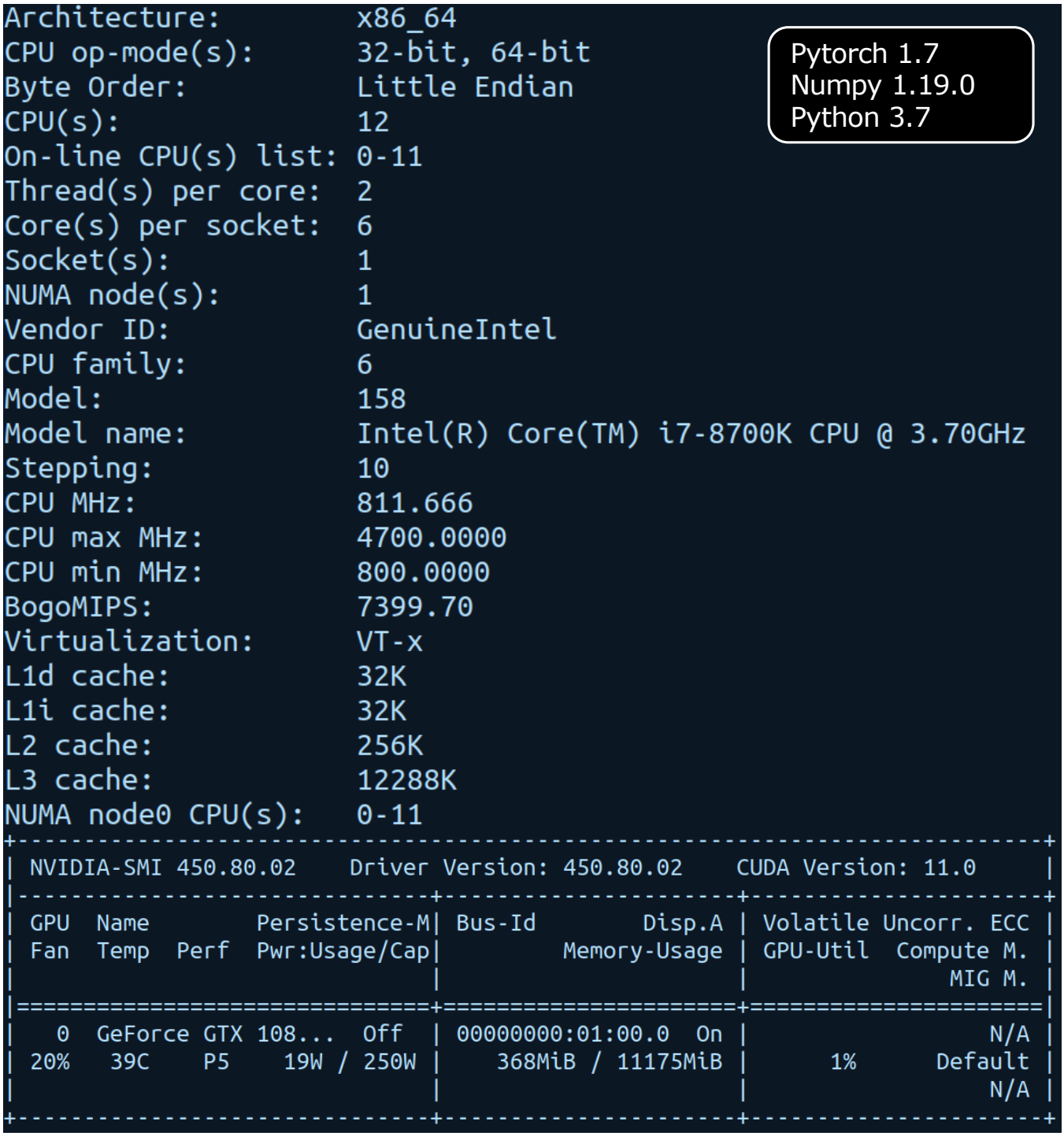


Fig. 10. Detail Platform Description.