Porting your Pytorch application to Voyager Gaudi Architecture

Javier Hernandez-Nicolau
Computational Data Scientist

Advanced HPC-CI Webinar Series. April 8th 2025

SAN DIEGO SUPERCOMPUTER CENTER

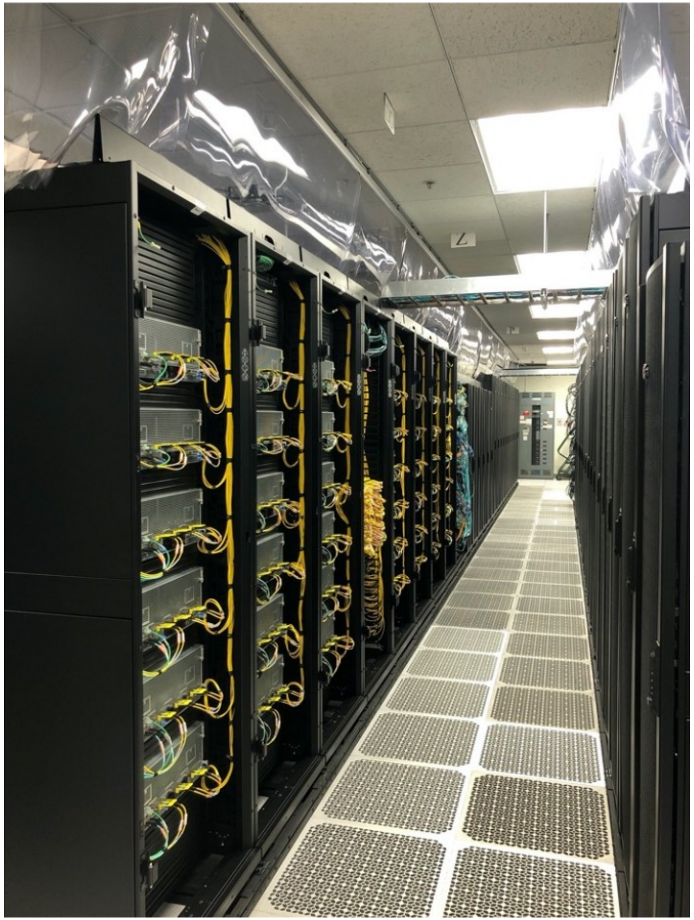
UC San Diego
SCHOOL OF COMPUTING, INFORMATION AND DATA SCIENCES

Outline

- Introduction to the Voyager system
- Containers and Kubernetes
- Running a pod on Voyager
- Porting a Pytorch application
- Parallel runs with MPI
- Jupyter Notebooks
- Import a model from Huggingface

Voyager: an HPC system with an innovative architecture dedicated to AI

- Envisioned as a system to facilitate exploration of a **new architectures** in support of **AI** in research and engineering.
- Its main component is the **Intel Gaudi** accelerators. They are specifically **designed** for **AI** applications and its optimized software allows users accelerate and scale their models.
- Voyager is a NSF category II HPC system. Test phase + production.
- During the 3-year **test phase**, the Voyager team has engaged with the research community. A **large number** of Deep Learning **models** from multiple fields have been deployed and analyzed on Voyager allowing us to learn more about this unique architecture.
- In the **allocation** (production) phase, Voyager will open its doors to a broader community.



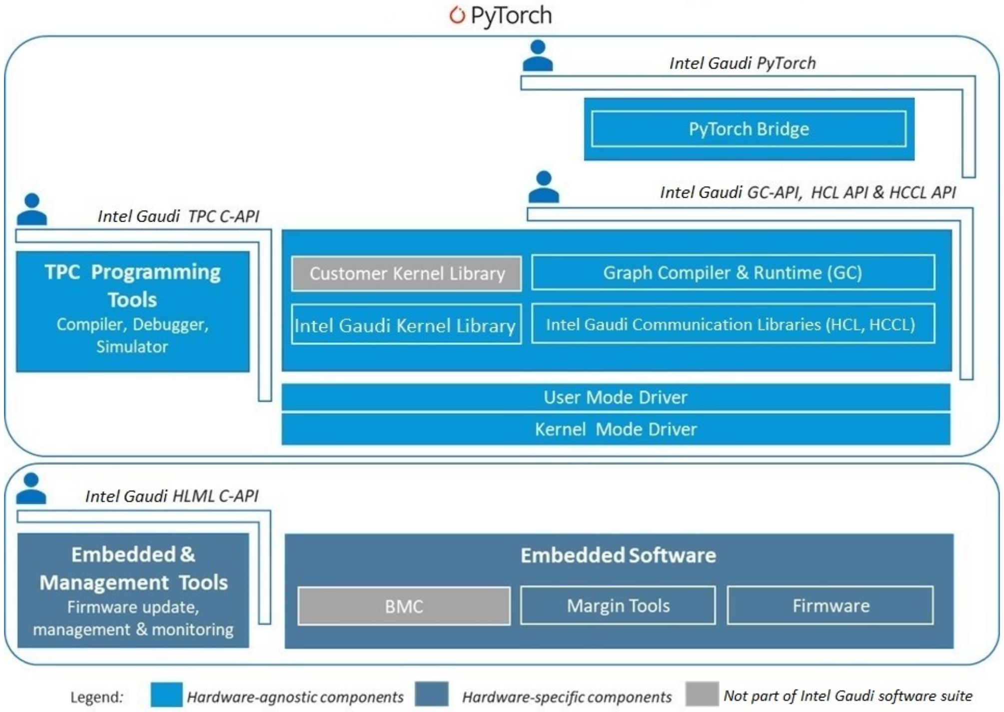
Voyager system: architecture

- 42 **Training nodes**, each with 8 **Intel Gaudi** cards specialized for AI. All-to-all networks between processors in a node.
- 36 Intel x86 processors **compute nodes** for general purpose (e.g. data processing…)
- 400 Gbe interconnect using RDMA over converged **Ethernet**
- Storage:
  - Gaudi nodes: 6.4Tb **local** NVME
  - **Ceph**: with 3PB connected via 25Gbe
  - **Home**: 324 TB connected to compute via 25Gbe

System component	Configuration
Supermicro X12 Gaudi Training Nodes	
CPU Type	Intel Xeon Gold 6336
Intel Gaudi processors	336
Nodes	42
Training processors/Node	8
Host X86 processors/node	2
Memory capacity	512 GB DDR4 DRAM
Memory/training processor	32 GB HDM2
Local storage	6.4 TB local NVMe
Compute nodes	
CPU Type	Intelx86
Nodes	36
X86 preocessors/node	2
Memory capacity	384 GB
Local NVMe	3.2 TB
Storage System	
Ceph	3PB
Home	324 TB

Software stack overview: Synapse AI

- At a very high level, users need very **little code modification** to run their AI models.
- **Pytorch** is **natively** integrated to Synapse AI and allows to run many popular AI applications in computer vision and generative AI (LLMs, Diffusion models).
- **Advanced users** can write their own customized kernel.
- Synapse AI is periodically **updated** including more features.



https://docs.habana.ai/

Scientific applications already running on (or being ported to) Voyager

Field/Project	AI architecture
Cosmology: Super resolution of galactic simulations	Diffusion
Biomedical text analytics	LLM
Cardiac image analysis	U-Net
High energy physics	Graph NN
Molecular simulations	NN
Data driven weather prediction	U-Net
Human microbiome research	Categorical VAE
Dose prediction in cervical brachytherapy	U-Net
Ontology-driven finetuning of LLM for epilepsy	LLM
Research accessibility via visual representation	Diffusion

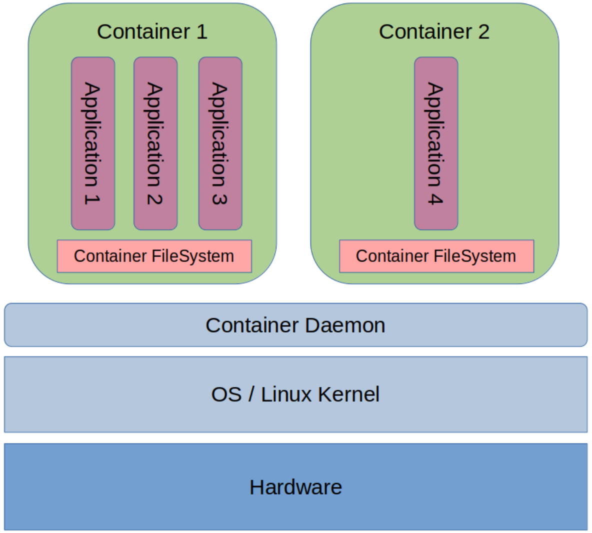
How to access Voyager

- Feel free to contact us at consult@sdsc.edu
- **Allocation** time can be requested via NAIRR. Several projects are already running.
- Starting June 1st, Voyager will get in to **production phase**. Allocations can be requested via ACCESS.
- Voyager can be accessed with **ssh**:
  - ssh to login.voyager.sdsc.edu
  - On Windows use any ssh app. For linux/Mac use the terminal
- **Tutorials** and reference models can be found at: https://github.com/javierhndev/Voyager-Reference-Models

Containers and Kubernetes

What is a container?

- A container is a way to **package applications** with all the necessary dependencies and configurations.
- In a container you can install your **own** software/libraries or specific versions.
- They are **portable** and can be easily shared with others. Also containers simplify **reproducibility**.
- Is this a Virtual Machine? No!
  - **VM**: Virtualization at hardware level. Isolation of machines
  - **Containers**: Virtualization at OS level. Isolation of processes.



Kubernetes

- Containers on Voyager are deployed via **Kubernetes** (K8S).
- Kubenetes is an **orchestration** system to deploy and manage containers.
- Kubernetes **objects** describe the containerized applications, software, resources (memory, cpus…), volumes, policies….
- K8S objects are described by a **configuration file** in json or yaml format.
- The smallest object in K8S is a **pod** and includes one (or more) containers.
- Each container must pick an **image** which includes all the **necessary software** to run the application.
- Images are **externally hosted**. The most famous repository being DockerHub. However images for Voyager and Intel Gaudi can be obtained from vault.habana.ai.

kubernetes

Manage applications in Kubernetes

- The kubectl command can be used to create, **manage** and delete kubernetes objects.
- It can be loaded on Voyager with the *module load kubernetes/voyager* command
- To **create** an object:
  - kubectl create -f mypod.yaml
- To **delete** the object:
  - kubectl delete -f mypod.yaml
- **List** all pods:
  - kubectl get pods
- Get the pod **logs**:
  - kubectl logs podname

Hello world pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  restartPolicy: Never
  containers:
    - name: gaudi-container
      image: vault.habana.ai/gaudi-docker/1.15.1/ubuntu22.04/habanalabs/pytorch-installer-2.2.0:latest
      resources:
        limits:
          memory: 32G
          cpu: 12
          habana.ai/gaudi: 1
      command: ["/bin/sh","-c"]
      args:
        - echo 'Hello World!';
```

Pod name

What if pod crashes?

Container image to use

Gaudi cards

Commands to be executed

Run the hello world pod

- Launch a single pod with:
  - Kubectl create -f podfile.yaml
- Check their status:
  - Kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
test	0/1	Completed	0	57s

- Check the ouput log with:
  - kubectl logs podname

A more complex yaml file

```
apiVersion: v1
kind: Pod
metadata:
  name: mnist
spec:
  restartPolicy: Never
  volumes:
    - name: scratch
      emptyDir: {}
    - name: workdir
      hostPath:
        path: /home/javierhn/webinar_adv_comp_2025
        type: Directory
  containers:
    - name: gaudi-container
      image: vault.habana.ai/gaudi-docker/1.15.1/ubuntu22.04/habanalabs/pytorch-
installer-2.2.0:latest
      …..
      …..  (continue next page)
```

Folders container can access

Scratch (purged)

Folder from /home FS

A more complex yaml file (continue)

```
containers:
  - name: gaudi-container
    image: vault.habana.ai/gaudi-docker/1.15.1/ubuntu22.04/habanalabs/pytorch-installer-2.2.0:latest
    volumeMounts:
     - mountPath: /scratch
       name: scratch
     - mountPath: /workdir
       name: workdir
    resources:
     limits:
       memory: 32G
       cpu: 12
       habana.ai/gaudi: 1
       hugepages-2Mi: 95000Mi
     requests:
       memory: 32G
       cpu: 12
       habana.ai/gaudi: 1
       hugepages-2Mi: 95000Mi
    command: ["/bin/sh","-c"]
    args:
     - hl-smi;
       cd /workdir;
        #pip install apackage;
       python run_mnist_onecard.py --epochs 20 --target-accuracy 0.90;
```

Mounts folders in the container

Defines name for folders

Gaudi card info

Install dependencies

Porting a Pytorch application to the Intel Gaudi architecture

Porting a Pytorch application to Intel Gaudi architecture

- Import the Intel Gaudi Pytorch framework:
  - `import habana_frameworks.torch.core as htcore`

- Target the Gaudi device:
  - `device = torch.device("hpu")`

- A `mark_step()` must be added after every `loss.backward()` and `optimizer.step()` during training and after loss calculation when testing/validating:
  - `htcore.mark_step()`

- See example (CNN with MNIST dataset) in repo.

Porting a Pytorch application to Intel Gaudi architecture

- More **complicated** model? CUDA/GPU API calls?
  - GPU Migration Toolkit
- **Deepspeed**?
  - Use Intel Gaudi's forked version
- Distributed training (DDP) with **NCCL**?
  - Use HCCL equivalent
- You can always contact SDSC support via consult@sdsc.edu

MPI parallelization on Voyager

Distributed Parallell training with MPI on Voyager

- Voyager is able to **scale up** your application to multiple Gaudi cards (and nodes!).
- For distributed training on (or less than) 8 cards you can use the Kubernetes **pod** object with the `mpirun` command.
- Running in **multiple nodes** (16 cards or more) requires a Kubeflow MPIJob.
- Check out our tutorials on https://github.com/javierhndev/Voyager-Reference-Models

Jupyter notebooks

Jupyter notebooks on Voyager

- Launch a **pod** with Jupyter:
  - pip install jupyter;
  - jupyter notebook –alow-root;

```
To access the server, open this file in a browser:
    file:///root/.local/share/jupyter/runtime/jpserver-48-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/tree?token=87c14bf9148425dcc3a1dc5d6cd5b9b
    http://127.0.0.1:8888/tree?token=87c14bf9148425dcc3a1dc5d6cd5b9b
```

- **Check** the Jupyter links with
  - kubectl logs yourpod

- Port forward the pod:
  - kubectl port-forward podname 9888:8888

- Create an **ssh tunnel** from the client (your pc):
  - ssh -N -f -L 8888:localhost:9888 login.voyager.sdsc.edu

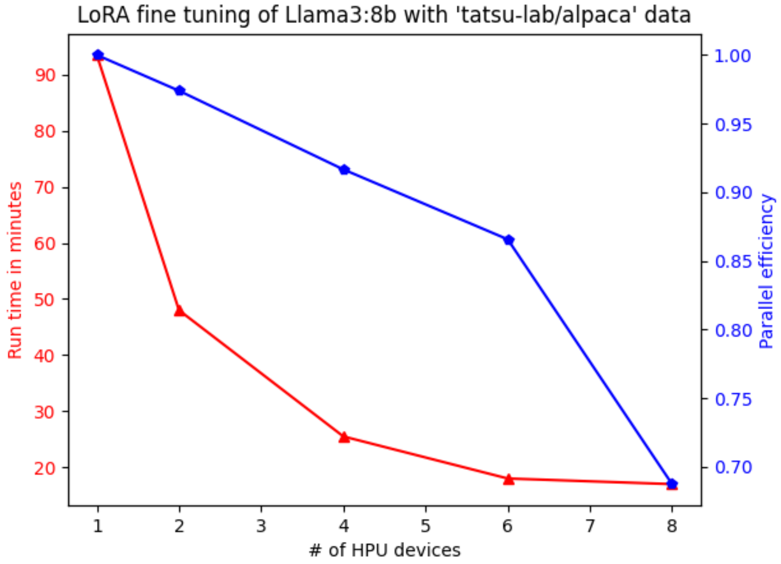
- Copy the Jupyter link to **your browser**.

Huggingface models on Voyager

Optimum-habana

Hugging Face

- **LLMs** and **Diffusion** models from **Huggingface** can be run on Voyager with the **optimum-habana** package.
- Optimum-habana (https://github.com/huggingface/optimum-habana) is built on top of **Hugginface Transformers/diffusers** and allows the models to take advantage of the **Gaudi** accelerators.



- **LLM:** finetuning of Llama3:8B, Llama3.1:8B, Gemma2:27B, Deepseek-R1-DistillQwen-32B
- Text Generation Inference (TGI)
- **Diffusers:** Stable Diffusion 1.5, Stable Diffusion XL1, CommonCanvas-XL-C
- **LMM:** qwen2-VL-7B, llama3.2-11B, llava-1.6-Mixtral-7B

Optimum-habana

Hugging Face

```
from optimum.habana.diffusers import GaudiDDIMScheduler, GaudiStableDiffusionPipeline

model_name='/dataset/stable-diffusion-v1-5'

scheduler = GaudiDDIMScheduler.from_pretrained(model_name, subfolder="scheduler")

pipeline = GaudiStableDiffusionPipeline.from_pretrained(
    model_name,
    scheduler=scheduler,
    use_habana=True,
    use_hpu_graphs=True,
    gaudi_config="Habana/stable-diffusion",
)

outputs = pipeline(
    ["A panda eating a taco"],
    num_images_per_prompt=8,
    batch_size=4,
)
image_save_dir="."
for i, image in enumerate(outputs.images):
    image.save( f"image_{i+1}.png")
```

Useful links

- Intel Gaudi documentation:
  - https://docs.habana.ai/
- Pytorch on Intel Gaudi:
  - https://docs.habana.ai/en/latest/PyTorch/index.html
- Reference models/Tutorials:
  - Intel-Gaudi:https://github.com/HabanaAI/Model-References
  - Voyager: https://github.com/javierhndev/Voyager-Reference-Models
- Optimum-habana:
  - https://github.com/huggingface/optimum-habana

Summary

- **Intel Gaudi** accelerators, designed for **AI**, are the main component of Voyager.
- Applications in Voyager are run in **containers** and are managed by **Kubernetes**. kubectl is the main command.
- Users can port their **Pytorch** applications with **little effort**.
- **Jupyter** notebook is **available** on Voyager
- To run LLMs or Diffusers from **Huggingface**, users need **optimum-habana** for Intel Gaudi.