Unleashing AI and HPC: Exploring SuperPod, HPC4, and Scalable Systems

Information Technology Services Office
The Hong Kong University of Science and Technology

Speaker: Kin Fai TSE
Manager (Research Computing), Ph. D.

2 Oct 2025

AI/HPC Facilities at HKUST

SuperPod

Nodes	56 (~45 Operative)
Total GPUs	448

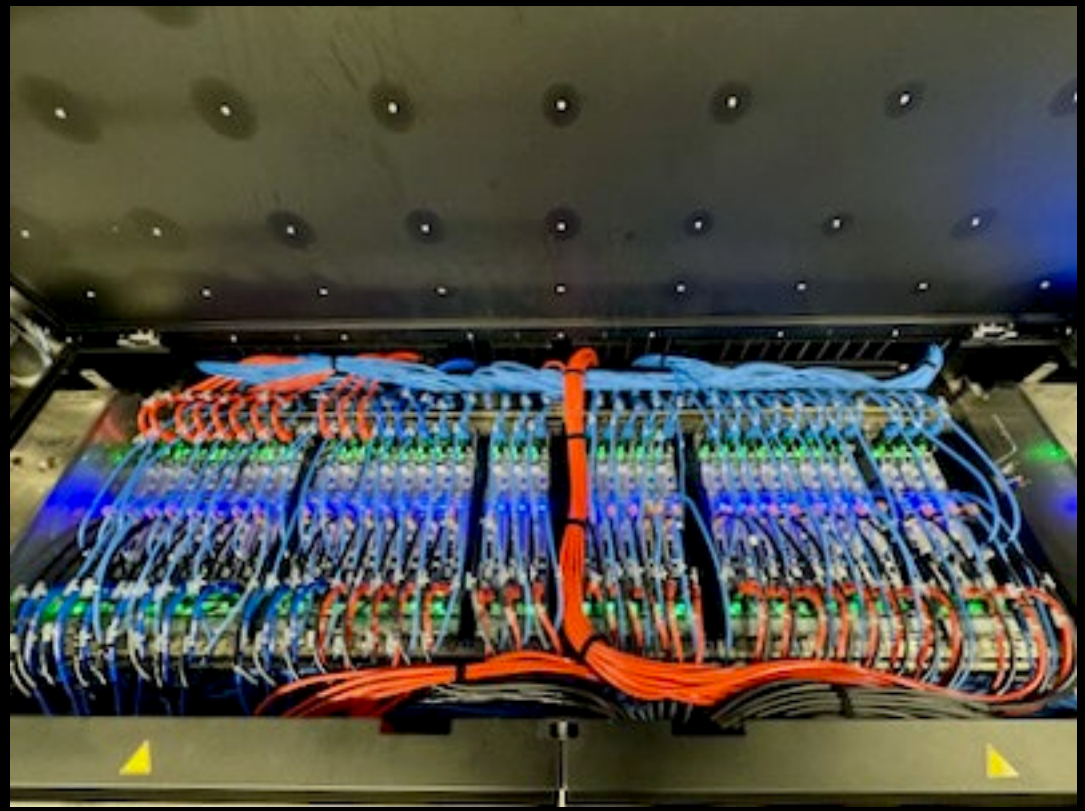
Per-node

GPU	8 x Nvidia H800 SXM 80GB
Memory	2 TB
CPU	2 x Intel 56 Core 112 Thread
Network	8 x 400Gbps InfiniBand
Tier 1 Storage	Lustre via auxillary 2 x 400Gbps InfiniBand
Tier 2 Storage	NFS via 2 x 100Gbps Ethernet

* Both facilities are billed to use

AI/HPC Facilities at HKUST

SuperPod



Nodes	>100
Total Cores	>25600 cores

Per-node

CPU	• 2 x Intel 64 Core • 2 x AMD 128 Core • GPU Server: 2 x Intel 32 Core
Memory	512GB – 1.5TB
GPU	• FP64: 4 x Nvidia A30 • 4 – 8 x Various Nvidia GPU
Network	1 x 200Gbps RoCEv2
Tier 1 Storage	NFS via 200Gbps RoCEv2
Tier 2 Storage	NFS via 200Gbps RoCEv2

* Both facilities are billed to use

Gaining Access to Computing Facilities

Main Method:

Engage in Research Opportunities with Professors

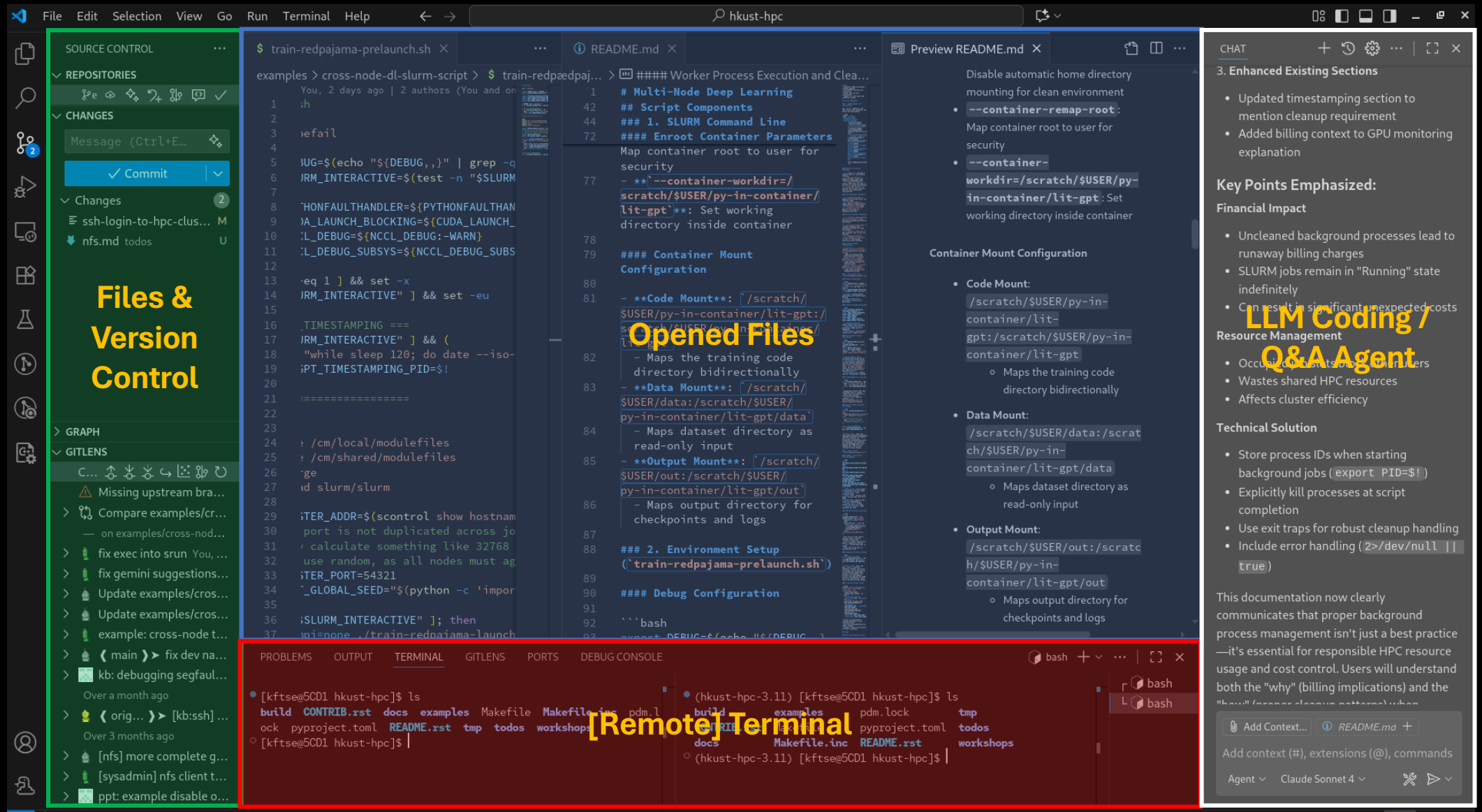
AI/HPC Basics
A Survival Guide

AI/HPC Survival Kit – 4 Simple Steps

Checklist

- ❑ **[Paid] Get an Intelligent IDE with LLM Integration**
  - ~10-20 US$ Monthly
    - Visual Studio Code with Copilot
    - Cursor
  - Code Generation: Claude-sonnet-4
  - Q&A (Reasoning): Gemini-2.5-pro
- ❑ **[Free] Set-up password-less login**
  - We support VPN / Campus Network + SSH Key Authentication
- ❑ **[Free] Get familiar with Git**
- ❑ **[Free] Use SLURM the Modern Way**

Modern IDE with LLM Integration



Set-up password-less login

<15 min set-up for **30+ hours saved per year!**

Instruction here
https://hkust-hpc-docs.readthedocs.io/latest/kb/ssh/ssh-login-to-hpc-cluster-without-usi-ImiEj9.html#login-to-hpc-cluster-without-using-password

* Assume at least 5 mins a day logging in several times

Git: Gear up for Serious Dev/Debugging

A long journey, and must be learnt through practicing

This is what I found the simplest guide

https://rogerdudler.github.io/git-guide/

SLURM – the Modern Way

Pitfall: **--nodes=2 --gpus=4**

- Ambiguous GPU resource: *Do you get 4 GPU or 8?*
- Unspecified resources: *How many cores per GPU do you get*?
- Ambiguous launch: *Are you running a main program per node, or per GPU*?
- Unconstraint topology: Is it ok to get *3 GPU on one node, and 1 GPU on another*?
- Unstated needs: *How long* do you need these for?

Answer: depends on *SLURM version*, and our *guess on what "average user" needs*

Common reason why jobs failed or being extremely slow

SLURM – the Modern Way

Correct: **--nodes=2 --gpus-per-node=4 --ntasks-per-node=4 --cpus-per-task=28 --time=3-0:0:0**

- GPU: 2 nodes * 4 gpus-per-node = *8 GPU*
- Resources: 2 nodes * 4 ntasks-per-node * *28 cpus-per-task* = *224 cores*
- Launch: *4 main program per node*, i.e. 1 per GPU
- *Requires identical topology* for each node
- Time: *3 days*

Pro Tips:

- Always *specify in `--X-per-Y`* if syntax available to avoid ambiguity
- *Fully specify* what you need
- *Do NOT specify memory* (we allocate proportional to GPU, or CPU if GPU not avail)

From Development to Production on SLURM

Interactive Development

`*salloc* <resource-params>`

- Pre-allocate a session

`*srun* --jobid <jobid> --pty bash`

- Run interactively into the session

`*srun* --overlap --jobid <jobid> launch-script.sh`:

- Fork a run inside an interactive session

You should connect from your PC throughout the whole job
You must interact with the job via srun

Production Batch Run

`*sbatch* <resource-params> sbatch-script.sh`

- Submit a batch job

You can disconnect your PC when queuing or running

You cannot interact with the job during run

Cancelling Jobs

Interactive

`scancel <jobid>`

- Cancel when you finish your development session
- Billable time: Throughout the time which the allocation is usable
  - Don’t leave an allocation and go to bed!

Production

`scancel <jobid>`

- Cancel when you no longer need the job
  - The job/input has some problem
  - The model is trained enough for now
  - etc
- Billable time: While the job is running
  - Not the time requested

`squeue` to check for job ids.

AI/HPC Survival Kit – 4 Simple Steps

Checklist

- ❑ [Quick] Set-up password-less login
- ❑ [Quick] Use SLURM the Modern Way
- ❑ [Medium] Get an Intelligent IDE with LLM Integration
- ❑ [Slow] Get familiar with Git
- ❑ [Lengthy] Work on your Research Project

AI/HPC Architecture

Specialized Features, and How to get the Best out of them

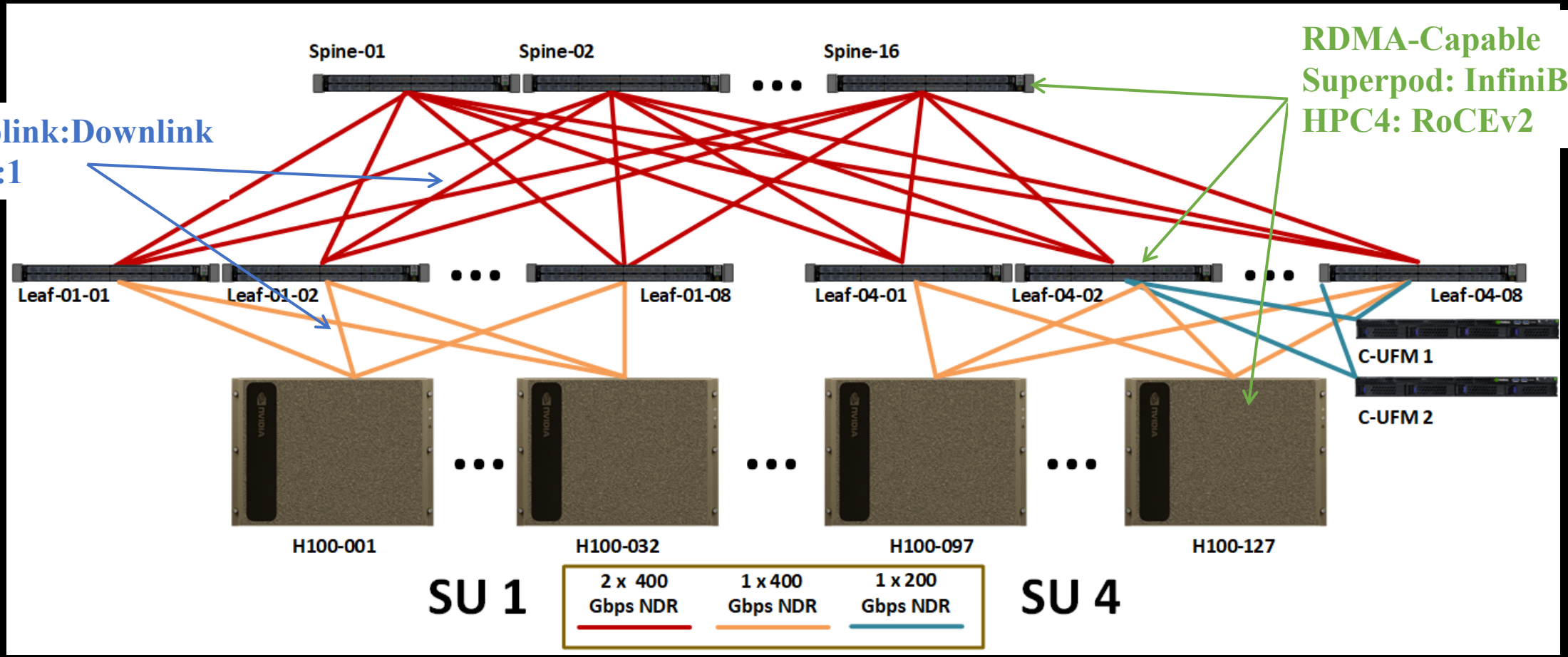
Important Aspects of an HPC

- Networking Topology / RDMA
- NUMA Domains
- GPU Accelerated Computing
- Parallel File System
- HPC Container Runtime

Design your program / workflow around what the platform is best at doing

Another common reason why jobs being extremely slow

Fat Tree Network Topology



https://docs.nvidia.com/dgx-superpod/reference-architecture-scalable-infrastructure-h100/latest/network-fabrics.html

Best Practice:
Use Established Async Communication Library

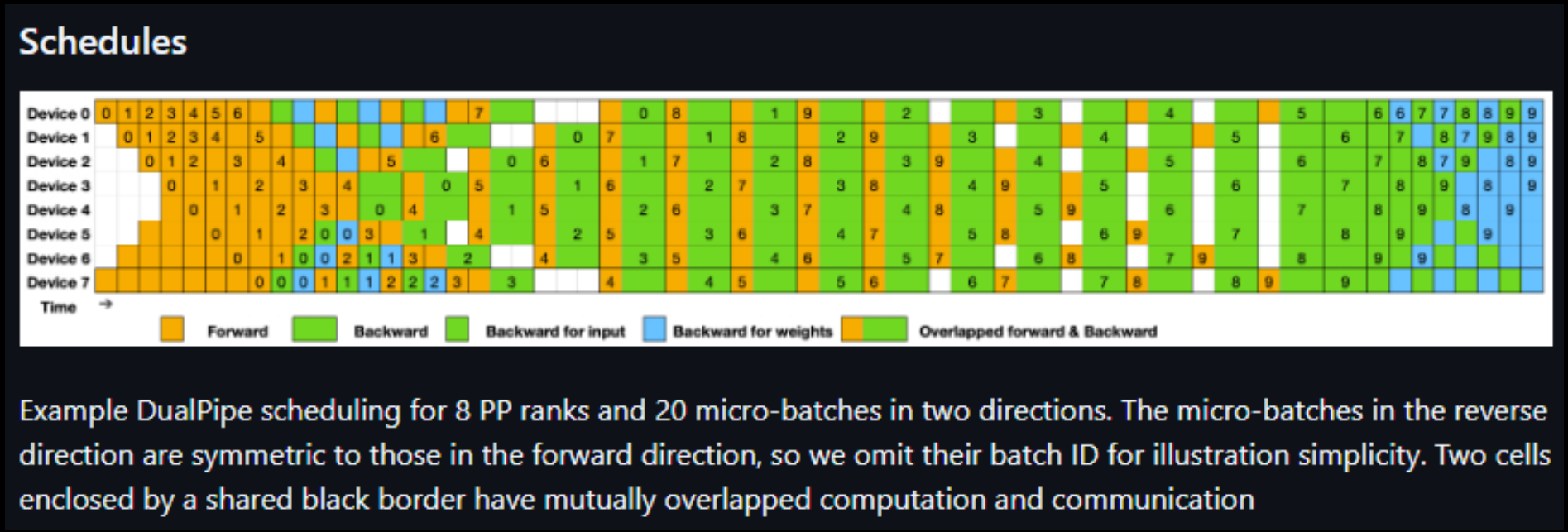
These work across platforms

- `torch.distributed` and distributed training library based on it
  https://docs.pytorch.org/docs/stable/distributed.html
- Message Passing Interface (MPI) for HPC application
  https://www.mpich.org/static/docs/latest/
  https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf
- NVIDIA Collective Communications Library (NCCL) for GPU applications
  https://developer.nvidia.com/nccl

Best Practice:
Stack Compute with Async Data Transfer

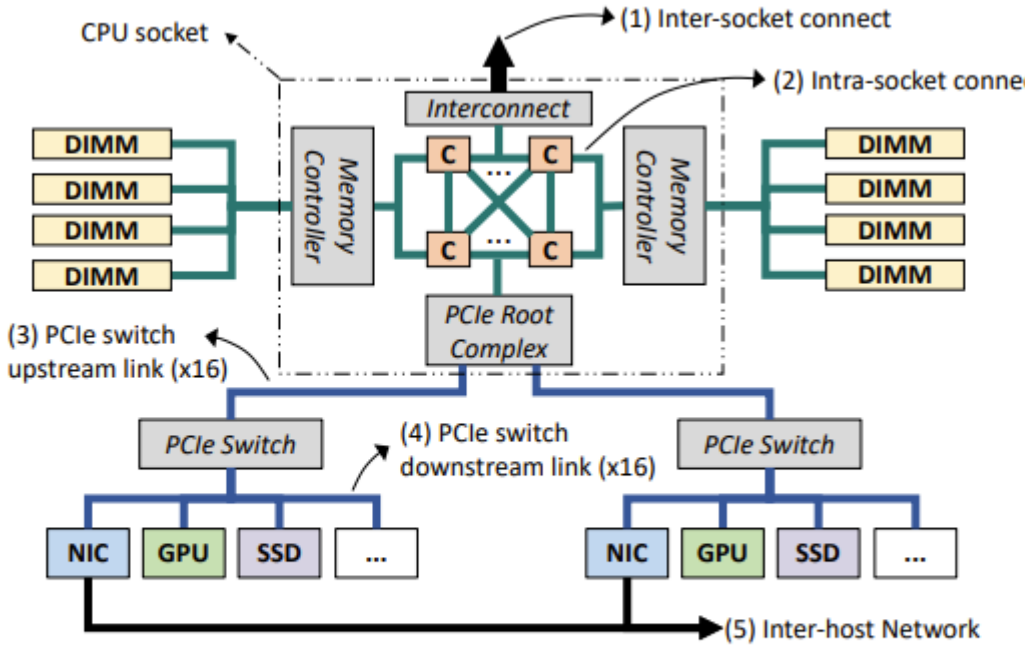
If viable, depends on algorithm

- Don’t over-optimize



https://github.com/deepseek-ai/DualPipe

NUMA Domains: Intra-node Connectivity



Configuration

Socket

NUMA IOMMU DDIO

Intra-Host Fabrics

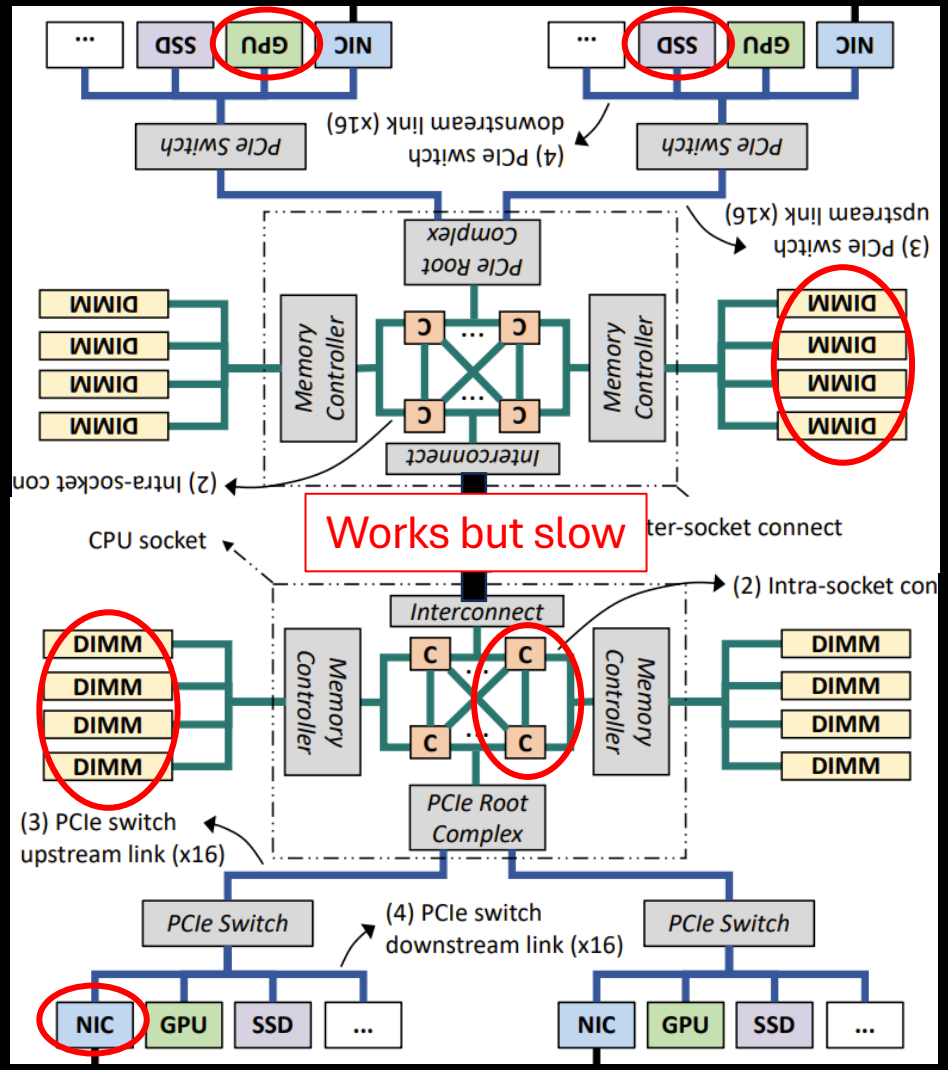
Request/Payload Size
Ordering Restriction
Access Control Services
Translation Services
Interrupt moderations

Item	Capacity	Basic Latency
(1)	20-72 GBps	130-220ns
(2)	100-200 GBps	2-110ns
(3)	~256 Gbps	30-120ns
(4)	~256 Gbps	30-120ns
(5)	~200 Gbps	< 2us

Figure 1: An example topology of a commodity server. The highlighted links depict the complex intra-host network within a modern server. The performance numbers describe the capacity and basic latency of common devices in commodity data center servers, such as Intel Cascade Lake and AMD EPYC CPUs, and PCIe 4.0 buses.

https://dl.acm.org/doi/10.1145/3593856.3595890

Best Practice:
Use Local NUMA Domains



Most are already done on infrastructure / library level via

- Hardware Design
- Hardware Locality Detection (HWLOC)
- NUMA-local Binding for CPU, GPU, Memory and NIC

GPU Best Practice:
Think Parallel

As Jensen Huang presented this year

Tips: Program like “chain of matrix operations”, not imperative loops

Parallel File System

On-disk FS

XFS on a M.2 NVMe

- Metadata blocks
- Data blocks

Specifications

- IOPS : ~1M
- Seq. BW: ~5GB/s

Same fast local NVMe for metadata and data

Always micro-second latency

No concurrent users

Parallel FS

Lustre on DDN All-Flash Storage, Ceph, NFS

- MDS (Metadata Server)
- OSS (Object Storage Server)

Specifications

- IOPS: ~1M (depends on synchronization semantics)
- Seq. BW : ~40 GB/s

Many more OSS than MDS

Network latency matters when reading file one-by-one

Concurrent load from other users increases latency

Best Practice:
Aggregate Small Files into Large Data Blocks

The fast way to use Parallel File System

- Use block data formats, each block should be at least ~10MB
  - JSONL for json
  - WebDataset for image
  - HDF5 for Scientific data output
  - npz for numerical array input (e.g. Tokenized text)
- Keep a filename list of data blocks, do not `ls` or `glob`
- Use parallel file utilities, e.g. fpsync, mpiFileUtils
- Use containers

Containerization in HPC

Docker Containerization

- docker run …
- hub.docker.io
- PULL …/python:latest

- Unarchive layers
- layer sha256:…
- layer sha256:…
- …

- Mounts
- Network
- Volume
- etc

- Docker: with root privilege
- CMD …

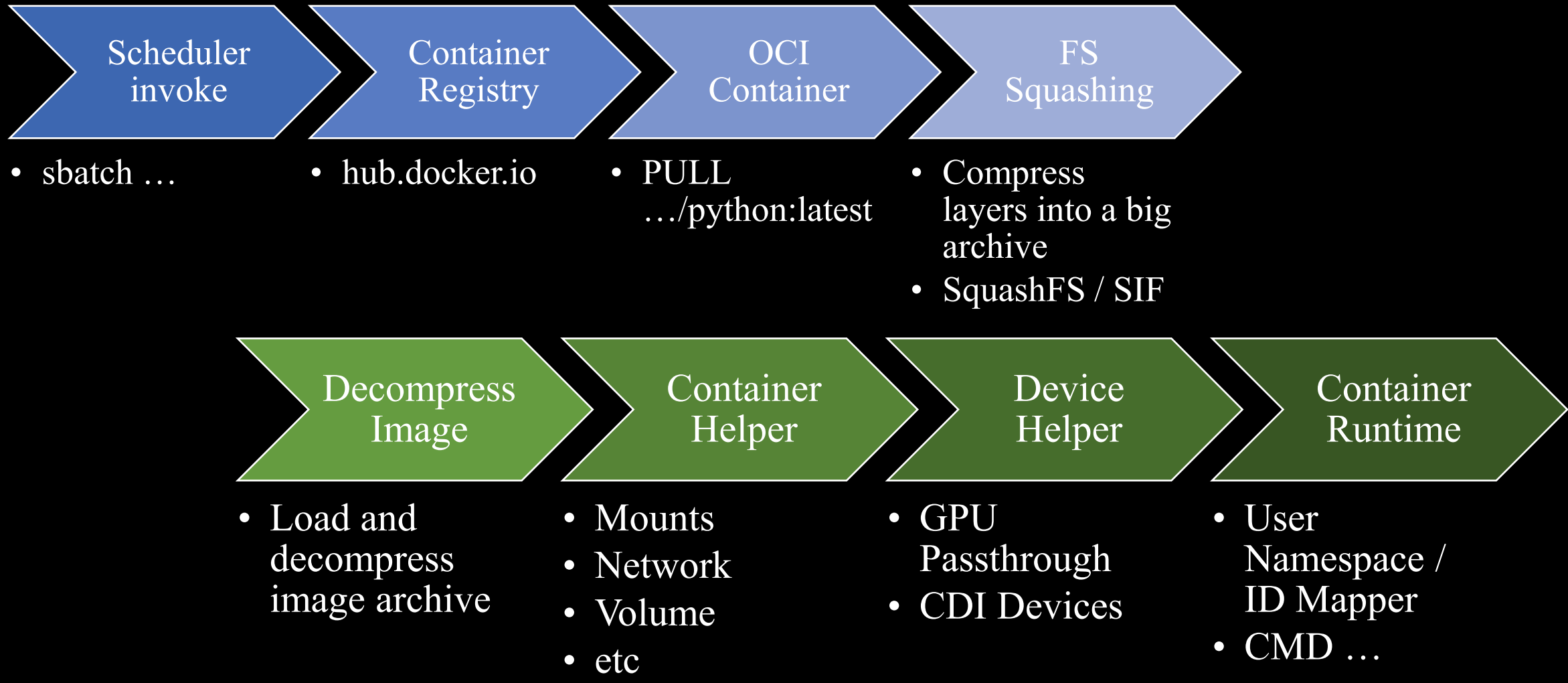
Why *NOT* Docker

Incompatible offerings

- Different design philosophy and trade-off

Docker Offerings	Multi-tenant Platform Offerings
Command-line / Local daemon based invoke	Scheduler
Uses OverlayFS	Poor performance for large number of small files
Fixed Resources / GPU shared among all containers	Scheduler-coordinated Resources / GPU allocation
GPU via plugin support	Scalable GPU / IB support
Root-privileged	Root not allowed on shared cluster

HPC Containerization



Best Practice:
Adapt to HPC Containers

Similar flexibility as in Docker

- `apt-get install`
- Choose your own base image
- Usual container build workflow

Be aware of the difference

- Different launch command / syntax
- Convert image before use
- ***No export back to Docker***
- ***No background daemon***
- Ultimately, it is still rootless: No network logging, kernel hooking etc

Debugging AI/HPC Workload

3 Pillars of Observabilities using user-space tools only

- Monitoring
- Logging
- Tracing

Scaling Job Monitoring

3 Main Indicators

1. GPU Utilization %
   - Is there enough work on GPU?
2. GPU Power Consumption
   - Calculation takes a lot of power
   - Memory / Network transfer do not
   - Typical DL training
     Calculation ≫ Transfers
3. PCI-e Receive (RX) / Send (TX) [to/from GPU]
   - Loading batched inputs: sustained RX
   - Checkpointing: A spike TX, slightly longer for large models
   - Logging: Small sustained TX
     e.g. logging loss every 50 steps

Simple Monitoring along SLURM job

```
test $SLURM_LOCALID -gt 0 ||
  nvidia-smi dmon --delay 15 --select pumt --options DT --filename
$PWD/out/slurm/slurm-$SLURM_JOB_ID.$(hostname).gutil &
```

Full Example

https://github.com/hkust-hpc-team/hkust-hpc/tree/main/examples/cross-node-dl-slurm-script

	Utilization	Power	PCI-e RX/TX
Column Label	**sm**	**pwr**	**rxpci txpci**
Interpretation	Higher is better	Higher is better	Below Threshold
Normal Range	95-100%	500-700W	100-10000MB/s
Threshold[1]	>= 90%	>= 350W	< 20000MB/s

[1] Close to or below this threshold, you should seriously review whether you can justify the underutilization or attempt to work on improving it.

Typical Output – nvidia-smi dmon

Please report if gtemp >=85C

#Date	Time	gpu	pwr	gtemp	mtemp	sm	mem	enc	dec	jpg	ofa	fb	bar1	ccpm	rxpci	txpci
#YYYYMMDD	HH:MM:SS	Idx	W	C	C	%	%	%	%	%	%	MB	MB	MB	MB/s	MB/s
20240101	19:49:31	0	552	55	60	100	31	0	0	0	0	74333	993	0	8432	1276
20240101	19:49:46	0	564	55	60	100	61	0	0	0	0	74333	993	0	6776	5024
20240101	19:50:02	0	583	55	62	100	59	0	0	0	0	74333	993	0	6156	1848
20240101	19:50:17	0	571	55	62	100	66	0	0	0	0	74333	993	0	5183	2441
20240101	19:50:33	0	617	50	59	100	66	0	0	0	0	74333	993	0	1304	146
20240101	19:50:48	0	566	50	58	100	51	0	0	0	0	74333	993	0	5881	599
20240101	19:51:04	0	115	40	50	100	0	0	0	0	0	74333	993	0	0	0
20240101	19:51:19	0	569	54	60	100	37	0	0	0	0	74333	993	0	5842	1798
20240101	19:51:35	0	583	54	61	100	31	0	0	0	0	74333	993	0	6053	1818
20240101	19:51:50	0	582	54	62	100	63	0	0	0	0	74333	993	0	6628	1496
20240101	19:52:06	0	565	54	60	100	61	0	0	0	0	74333	993	0	6098	1565
20240101	19:52:21	0	564	55	63	100	59	0	0	0	0	74333	993	0	5879	1561
20240101	19:52:37	0	558	55	62	100	66	0	0	0	0	74333	993	0	3839	139
20240101	19:52:52	0	613	49	58	100	66	0	0	0	0	74333	993	0	1347	386
20240101	19:53:08	0	577	50	59	100	50	0	0	0	0	74333	993	0	5747	607
20240101	19:53:23	0	571	53	60	100	58	0	0	0	0	74333	993	0	5882	612
20240101	19:53:39	0	562	53	61	100	31	0	0	0	0	74333	993	0	8127	1351
20240101	19:53:54	0	571	54	62	100	33	0	0	0	0	74333	993	0	6555	1519
20240101	19:54:10	0	574	56	61	100	62	0	0	0	0	74333	993	0	6484	1549
20240101	19:54:25	0	561	54	62	100	60	0	0	0	0	74333	993	0	5769	1270

Saving checkpoint

Can stay ~550W even for small model on multi-node[1]

If lower than ~70GB increase batch size by 1 until out of GPU mem

Typical RX~10GB/s and TX~1GB/s

[1] Llama LLM down-sized from 7B to ~0.5B fp-16 4096 context windows, DDP on 16 nodes, minibatch=16/GPU, no grad-accu, i.e. global batch size=16*8*16=2048

Correlating Distributed Logging

As simple as

- “Have a timestamp”
- “Have a hostname”

Solution (minimal):

- Print a timestamp
- Print separate log files

Not that simple

- Not out-of-the-box feature
- *Order of logs may appear different* due to buffers

See https://github.com/hkust-hpc-team/hkust-hpc/tree/main/examples/cross-node-dl-slurm-script

Tracing within Frameworks

Common scenario

1. Stuck at init

```
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
```

2. Splitted

```
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
Initializing distributed: GLOBAL_RANK: 0, MEMBER: 1/16
```

Look for environment variables that enables tracing

This helps >50% of networking / framework issues

Tracing (Framework dependent)

- NCCL
  - https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html#nccl-debug-subsys

```
export PYTHONFAULTHANDLER=1
export CUDA_LAUNCH_BLOCKING=1
export NCCL_DEBUG=INFO
# export NCCL_DEBUG=TRACE
export NCCL_DEBUG_SUBSYS=INIT
```

- UCX
  - `UCX_LOG_LEVEL`
- Intel MPI
  - `I_MPI_DEBUG` https://www.intel.com/content/www/us/en/docs/mpi-library/developer-guide-linux/2021-6/displaying-mpi-debug-information.html

Debugging C/C++/Fortran code using GDB

“Segmentation Fault”

General Workflow

1. Compile with `-g -O1` or below
2. Enable coredump
3. Reproduce the fault
4. Retrieve the dump from node
5. Analyse the dump using GDB

See https://hkust-hpc-docs.readthedocs.io/latest/kb/dev/dev-how-to-debug-segmentation-fault---d8CWH.html

- Get the backtrace to see the function call stack at the crash point

```
(gdb) bt
#0  initialize_array (array=0x153635fff010, rank=3) at src/mpi_impl.c:14
#1  process_array_and_calculate_sum (rank=3) at src/mpi_impl.c:27
#2  0x000055c955b2cda1 in main (argc=1, argv=0x7ffc29621628) at src/main.c:21
```

- Examine the source code around the crash location

```
(gdb) list
9     const unsigned long base = (unsigned long)rank * PER_RANK_ARRAY_SIZE;
10    for (int i = 0; i < PER_RANK_ARRAY_SIZE; i++)
11    {
12      // BUG: should be array[i] = base + i;
13      // This causes segmentation fault when rank > 1
14      array[i * (rank + 1)] = base + i;
15    }
16  }
```

- Inspect variable values at the time of crash

```
(gdb) print i
$1 = <optimized out>
(gdb) print rank
$2 = 3
(gdb) print array
$3 = (unsigned long *) 0x153635fff010
```

Additional Resources

Advanced Tooling for Profiling / Debugging

For HPC

Debugging

- MPIGDB

Profiling

- gprof + FlameGraph https://github.com/brendangregg/FlameGraph
- mpiProf
- Intel Vtune Analyser
- AMD-uprof

For GPU/Python

Debugging

- Nvidia Nsight
- Python Debugger

Profiling

- Nvidia Nsight
- Python profile

Additional Resources for AI / HPC

Parallel File System

- Lustre / Ceph
- OpenZFS impl: https://www.youtube.com/watch?v=ptY6-K78McY

Networking Topology / RDMA

- Infiniband / RoCEv2
- rdma-core: https://github.com/linux-rdma/rdma-core
- Nvidia DOCA: https://docs.nvidia.com/doca/sdk/index.html
- Multi-path Networking

GPU Accelerated Computing

- cuPy (GPU algorithm): https://cupy.dev/
- OpenACC (for HPC GPU-offload): https://www.openacc.org/
- CUDA (Performant GPU kernel): https://developer.nvidia.com/cuda-toolkit

NUMA Domains

- hwloc / numactl
- Intra-host Networking

Additional Resources to understand HPC Containers

HPC Container Projects

- Enroot
- Apptainer

Custom Device Passthrough

- Container Device Interface (CDI)

Research HPC

- K8s-based HPC

Container Spec/Impl

- OCI
- crun
- Linux Network Namespace
- User Namespace / ID Mapper
- CGroup

Q & A

Thank you for your participation.

This presentation is available at: https://github.com/hkust-hpc-team/hkust-hpc/tree/main/workshops/20251002-ai-hpc-systems