Deep Learning Best Practices on Ibex

GLENDON HOLST, HPC ML & VISUALIZATION
DAVID PUGH, DATA SCIENTIST

Deep Learning on Ibex

- Quick Intro
  - Prerequisites: Ibex, Slurm, Conda, ...
- Deep Learning Best Practices
  - Get More Done with Available Resources
  - Get Faster Access to More Resources
- Hands-On Clinic — By Appointment

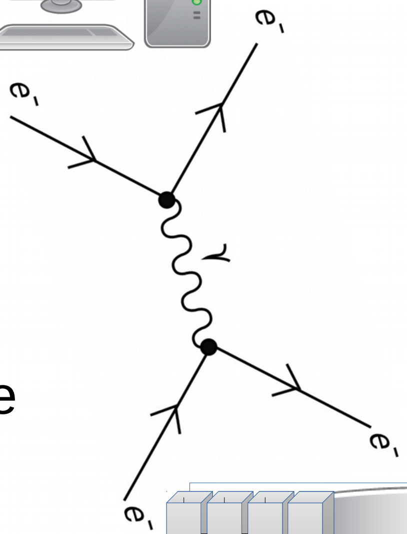
Ibex GPU Composition

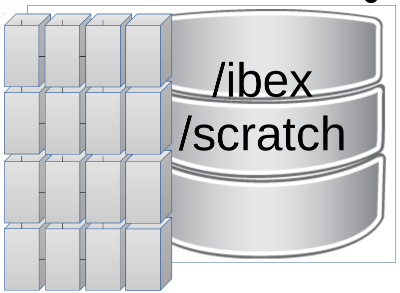
As of January 2020

GPU Model	#	Mem	CPU	Cores	Cores per GPU	Memory	Mem per GPU	Storage	Nodes	Total
GTX1080Ti	4	11.5GB	Haswell	36	**9**	256 GB	**64 GB**	200GB	8	32
GTX1080Ti	8	11.5GB	Skylake	32	**4**	384 GB	**48 GB**	900GB	4	32
Tesla P100	4	16 GB	Haswell	36	**9**	256 GB	**64 GB**	200GB	6	24
Quadro P6000	2	22 GB	Haswell	36	**9**	256 GB	**64 GB**	200GB	2	4
RTX2080Ti	8	11 GB	Skylake	32	**4**	384 GB	**48 GB**	900GB	4	32
Tesla V100	4	32 GB	Skylake	32	**8**	384 GB	**48 GB**	800GB	8	32
Tesla V100	8	32 GB	Cascade Lake	48	**6**	768 GB	**96 GB**	32TB	30	240
										396

Getting Started: Workstation → Ibex

- Workstation
  - Personal – all the resources!
  - Interactive
  - Local Storage
  - Packaged Software
    - Conda Envs





- Ibex
  - Shared – specify resources; CPU, Mem...
  - Batch – scripted
  - Remote Filesystem
  - App Stack – Modules
    - Conda Envs

Getting Started: Workstation → Ibex

- `ssh vlogin.ibex.kaust.edu.sa`
- First login auto-generates keys & ssh config
  - `.ssh/config`
    - ```
      Host vlogin #GPU login nodes
          Hostname vlogin.ibex.kaust.edu.sa
          User $USER
          IdentityFile ~/.ssh/ksl-internal
          StrictHostKeyChecking no
          ForwardX11 yes
          ForwardX11Trusted yes
      ```

https://www.hpc.kaust.edu.sa/ibex/new_user
https://www.hpc.kaust.edu.sa/ibex/faq
      ```

Batch Training: Workstation → Ibex

- WS: Parameterize `train.py`
  - ```
    import argparse
    parser = argparse.ArgumentParser(description="training")
    parser.add_argument("--batch-size", type=int, default=32,
                        help="training batch size")
    args = parser.parse_args()
    ```
  - `tf.data.Dataset. … .batch(args.batch_size). …`
- WS: Write shell script to perform training
  - ```
    #!/bin/bash
    conda activate ./env
    python train.py "$@"
    ```
  - `./run_train.sh --batch-size=64`

Batch Training: Workstation → Ibex

- Ibex: Add #SBATCH resource specs
  - ```
    #!/bin/bash
    #SBATCH --nodes=1
    #SBATCH --ntasks=1
    #SBATCH --mem=45G
    #SBATCH --cpus-per-task=4
    #SBATCH --gres=gpu:1
    #SBATCH --constraint="[p100|gtx1080ti]&intel"
    #SBATCH --time=24:00:00
    #SBATCH --partition=batch
    #SBATCH --output=log-%x-slurm-%j.out
    #SBATCH --error=log-%x-slurm-%j.err

    ...
    ```
    ```

Batch Training: Workstation → Ibex

- Ibex: Run training job via Slurm
  - `sbatch run_train.sh --batch-size=64`
  - `sbatch --time=15:00 --constraint=[p6000] --partition=debug \`
    `run_train.sh --batch-size=16`
- Ibex: Slurm job management
  - `squeue -u $USER`
  - `scancel` <*jobid*>
  - `ginfo --all`
  - `sinfo -o "%20N %10c %10m %25G %f" | grep -E '(FEATURES|gpu:)'`
  - `scontrol show job` <*jobid*>

- Get More Done with Available Resources
  - Making your jobs run faster helps everyone
- Get Faster Access to More Resources
  - Good things come to those who share...

Best Practices: Allocate Sufficient...

- Allocate Sufficient CPUs
  - `CPU_PER_GPU`
    - Depends on node configuration (Ibex hardware chart)
    - Specify 4, or proportional fair share (from chart)
  - `--cpus-per-task=$(( NUM_GPUS * CPU_PER_GPU ))`
- Allocate Sufficient Memory
  - `MEM_PER_GPU in GB`
    - Depends on node configuration (Ibex hardware chart)
    - Specify 45, or a little less than proportional fair share (from chart)
  - `--mem=$(( NUM_GPUS * MEM_PER_GPU ))G`

Best Practices: Local Data...

- Load Training Data from Local Storage
  - `/tmp`
  - AI GPU nodes have very large and fast local storage:
    - Less IO contention with other users; fast SSD
    - Only visible on the compute node it is attached to (per node + job)
    - Local storage is temporary — only lasts as long as the job

Best Practices: Local Data...

- Use Reference Datasets
  - Eliminates initial dataset copy time
  - Faster local storage
  - Use sinfo command to find nodes with reference data:
    - ```
      sinfo -o "%20N %10c %10m %25G %f" | \
             grep -E '(AVAIL_FEATURES|ref_)'
      ```
  - Ensure GPU, CPU, Memory, and GRES match job spec; then add constraint:
    - `--constraint=[ref_32T]`
  - Provided reference data is available on node local storage:
    - `/local/reference/`<*reference-data-set*>
  - Request to provide dataset: <ibex@hpc.kaust.edu.sa>
    - `/ibex/reference/CV/{COCO,GQA,ILSVR}`
      ```

Best Practices: Local Data...

- Copy Training Data to Local Storage
  - Ensure requested nodes have sufficient local storage (Ibex hardware chart)
  - For single node allocation:
    - `cp -r /ibex/scratch/${USER}/data/set /tmp/data`
    - ```
      cp /ibex/scratch/${USER}/data/set.tgz /tmp/data
      cd /tmp/data ; tar xvpf set.tgz
      ```
  - For single or multi-node allocations:
    - ```
      sbcast /ibex/scratch/${USER}/data/set.tgz /tmp/data
      cd /tmp/data ; srun -N $N -n $N tar xvpf set.tgz
      ```
    - Broadcast copy operation to all nodes

Best Practices: Local Data...

- Filesystem Guidance
  - For improved copy performance, prefer source data with fewer, larger files
    - Avoid using large directories with many (e.g., thousands of) small files, especially on `/ibex/scratch/`
    - Archive source data in `*`.tar or `*`.zip files (and un-compress on the compute node after the copy)
    - `*` Or, place individual files inside containers, like an HDF5 file, and work with these files via the container API
  - Avoid using `/home` for data files (read or write)

Best Practices: Loading Data...

- Maximize data loading performance
  - Use framework provided data loading and processing tools:
    - E.g., `tf.io.*`, `tf.image.*`, `tf.data.Dataset.*`, `tf.data.*.AUTOTUNE`
  - Load and process data in parallel
    - E.g., `.map(preprocess, num_parallel_calls=AUTOTUNE)`
  - Ensure that data buffers are sufficiently large (for hiding IO latency and caching)
    - ```
      .shuffle(args.shuffle_buffer_size,
               reshuffle_each_iteration=True)
      .prefetch(args.prefetch_buffer_size)
      ```
  - See example:
    - https://github.com/kaust-vislab/tensorflow-gpu-data-science-project
      - src/horovod-keras-example/train.py
      ```

Best Practices: GPU Utilization...

- Maximize utilization of large-mem GPUs
  - E.g., Ibex V100 has 32GB (2x standard 16GB V100) — use it
  - Double batch size, double learning rate, half number of GPUs per job
    - Job performance stays the same (similar)
    - Double number of simultaneous jobs
  - Deep Learning at Scale, Large Mini-batch SGD
    - https://towardsdatascience.com/deep-learning-at-scale-accurate-large-mini-batch-sgd-8207d54bfe02
    - Linear Scaling Rule: **lr'** = **lr** * **k** (for k time batch size increase)
    - Warmup Strategies: **lr** → **lr'**, gradual constant increase from small **lr** over 5 epochs
    - Batch Normalization: Variety of techniques… subtle pitfalls*
  - * Ask <ibex@hpc.kaust.edu.sa> for assistance with scaling on Ibex

Best Practices: GPU Utilization...

- Monitor GPU utilization
  - Even basic GPU and CPU monitoring can diagnose common performance issues
- Add monitoring to job script
  - ```
    # run monitoring tool in background (&), pipe to *.log file
    # get process id (pid) to stop monitoring after training
    nvidia-smi dmon --delay 60 --options DT >> \
                                 nvidia-smi_${SLURM_JOBID}.log &
    NVIDIA_SMI_PID=$!

    # run training
    <launch training + output processing here>

    # terminate previous monitoring when training completes
    kill $NVIDIA_SMI_PID
    ```
    ```

Best Practices: GPU Utilization...

- Interactively monitor running job
  - ```
    # show running jobs and find <jobid>
    squeue -u $USER
    ```
  - ```
    # examples of interactive session GPU / system monitoring
    > srun --jobid=<jobid> -u --pty bash -i
    > srun --jobid=<jobid> -u --pty nvidia-smi dmon
    > srun --jobid=<jobid> -u --pty top -H -u $USER
    ```

Best Practices: GPU Utilization...

- GPU compute utilization can exhibit:
  - Consistent (e.g., > 90%) compute utilization — this is good
  - Consistent middling (e.g., < 70%) utilization — let’s improve
    - Small batch size? Low profile sampling rates (hidden oscillation)?
  - Multi-GPU request; some GPUs have 0% utilization — let’s improve
    - GPU device not bound process in `nvidia-smi pmon`?
      - Initialization issue
    - GPU devices are bound to process, but show 0% utilization?
      - Framework need explicit multi-gpu support added to code
      - See *Utilize all allocated GPUs*

Best Practices: GPU Utilization...

- GPU compute utilization can exhibit:
  - Oscillation between high (e.g., 100%) and low (e.g., 10%) utilization — let’s improve
    - Insufficient CPUs to load / process batch data?
      - Ensure sufficient CPUs per GPU, and framework data loader uses them
    - Data load is not in parallel with training operation?
      - Use framework data loader utilities with sufficient CPUs
    - GPU isn’t busy for long enough?
      - Increase batch size and buffer sizes to help hide latency
    - Slow filesystem IO performance?
      - Load data from local storage
    - Checkpointing pauses?
      - Save checkpoint to local storage; copy to /ibex/scratch concurrently

Best Practices: Multi-GPU Utilization...

- Utilize all allocated GPUs
  - **Multi-GPU support requires support in / changes to training code…**
  - TensorFlow: Multi-GPU aware; use `tf.distribute.*Strategy`
    - ```
      strategy = tf.distribute.MirroredStrategy()

      BATCH_SIZE = 64
      BATCH_SIZE = BATCH_SIZE * strategy.num_replicas_in_sync

      with strategy.scope():
        model = create_model()
      ```
  - PyTorch: Tensors are assigned to GPU devices manually
  - Horovod: Distributed TensorFlow / PyTorch training over MPI
    - Ask <ibex@hpc.kaust.edu.sa> for assistance with scaling on Ibex
      ```

Best Practices: Writing Models...

- Checkpoint to local storage
  - Local storage (`/tmp`) is fast...
  - Local storage is ***temporary***!
  - Write (`rsync`) to persistent storage (`/ibex/scratch`) in background

Best Practices: Writing Models...

- Copy in background...
  - ```
    # configuration
    RSYNC_DELAY_SECONDS=3600
    PID_CHECK_DELAY_SECONDS=60
    RSYNC_DELAY_LOOP=$((RSYNC_DELAY_SECONDS / PID_CHECK_DELAY_SECONDS))
    ```
  - ```
    # launch training command in background, get process ID on next line
    python train.py "${LOCAL_CKPNT_DIR}" ${TRAINING_ARGS} &
    RUN_PID=$!

    # asynchronous rsync of training logs to persistent storage
    RUN_DONE=false
    while [ "${RUN_DONE}" != true ] ; do
      for i in $(seq 1 ${RSYNC_DELAY_LOOP}) ; do
        sleep ${PID_CHECK_DELAY_SECONDS}
        if [[ "$(ps -h --pid $RUN_PID -o state | head -n 1)" = "" ]] ; then
          RUN_DONE=true ; break
        fi
      done
      rsync -a "${LOCAL_CKPNT_DIR}/" "${PERSISTENT_CKPNT_DIR}"
    done
    ```

Deep Learning: Best Practices on Ibex

- Get More Done with Available Resources
  - Making your jobs run faster helps everyone

- Get Faster Access to More Resources
  - Good things come to those who don’t wait...

AWS
Spot Instances

Bonus Resources: gpu_wide24

- New `gpu_wide24` partition
  - For wide (>= 4 GPUs) and short (< 24 hrs) jobs
  - 4 AI-GPU nodes with 8xV100 GPUs connected via NVLink
  - Benefits:
    - Faster turn-around for early results
    - Interleaved jobs make regular progress
    - Access extra resources
  - Satisfying jobs are automatically eligible to run in the partition
    - E.g., `--time=24:00:00 --gres=gpu:v100:4`

Best Practices: Checkpoint Support...

- Add checkpoint / restore support
  - Checkpoints are written / read by single task – usually root rank – restored to all
    - For single-task multi-GPU, <*checkpoint*> and <*restore*> are sufficient
    - E.g., For Horovod
      - `if hvd.rank() == 0:`
        <*checkpoint*>
      - `if hvd.rank() == 0:`
        <*restore*>
        <*broadcast*>
  - Checkpoint to local storage (`/tmp`); copy to persistent storage in parallel
  - Restore from persistent storage
  - Checkpoint at reasonable intervals — regular, but not too frequently
    - checkpoint delay should be small fraction of epoch training time

Best Practices: Checkpoint Support...

- Write checkpoint to local storage
  - E.g., TensorFlow
  - ```
    ckpnt_dir = pathlib.Path(args.write_checkpoints_to)
    if not os.path.isdir(ckpnt_dir):
      os.mkdir(ckpnt_dir)

    _checkpoints = (keras.callbacks.ModelCheckpoint( \
                      f"{ckpnt_dir}/ckpnt-epoch-{{epoch:02d}}.h5", \
                      save_best_only=False, \
                      save_freq="epoch"))

    callbacks.extend([_checkpoints])
    ```
  - **Note**: If `save_freq` = `int`, it is the number of samples (not batches) to process before saving
    ```

Best Practices: Checkpoint Support...

- Restore checkpoint from /ibex/scratch
  - E.g., TensorFlow
  - ```
    ckpnt_dir = pathlib.Path(args.read_ckpnt_from)
    ckpnt_filepath = None
    initial_epoch = 0

    for _epoch in range(args.epochs, 0, -1):
        _ckpnt_filepath = f"{ckpnt_dir}/ckpnt-epoch-{_epoch:02d}.h5"
        if os.path.exists(_ckpnt_filepath):
            ckpnt_filepath = _ckpnt_filepath
            initial_epoch = _epoch
            break

    if ckpnt_filepath is not None:
      model_fn.load_weights(ckpnt_filepath)
    ```
    ```

Best Practices: Sequence Jobs...

- Split training over multiple jobs
  - Shorter job runtime means more jobs to run; each job does a fraction of the work
  - Modify training code
    - Restore from latest checkpoint file
    - Process a fixed number of epochs / fixed time limit…
    - Exit cleanly (before job terminates).
    - Use reference data on local storage
  - Launch multiple jobs at once
    - Jobs must run in sequence
      - i.e., Next job depends upon previous job successfully completing
    - Automate (script) launches via Slurm sbatch
    - Future: Use workflow management tools (e.g., decimate) — support in development…
      - Improved: error retry, early exit, management tools

Best Practices: Launch Jobs...

- Create `launch_jobs.sh` script
  - ```
    #!/bin/bash

    # configs
    TOTAL_EPOCHS=${1:-100}              # 1st argument (default 100)
    EPOCHS_PER_JOB=${2:-10}             # 2nd argument (default 10)
    ## Note: TOTAL_JOBS = ceil(TOTAL_EPOCHS / EPOCHS_PER_JOB)
    TOTAL_JOBS=$(((TOTAL_EPOCHS + (EPOCHS_PER_JOB - 1)) / EPOCHS_PER_JOB))

    PARAMETERS="--lr=0.001 --bsz=32 --exp=3"
    PARAMETER_SPACE="${PARAMETERS//[-=\ .]/}"
    PROJECT_VERSION=$(git rev-parse --short HEAD 2> /dev/null || \
                      echo 'unver')
    JOB_NAME=train_job-${PROJECT_VERSION}-${PARAMETER_SPACE}
    ```
    ```

Best Practices: Launch Jobs...

- Create `launch_jobs.sh` script
  - Launch all jobs at once; each job depends upon the previous job completing successfully
  - ```
    # launch
    _DEPENDENCY_ARG=""
    for i in $(seq 1 ${TOTAL_JOBS}) ; do
      _JOB_ID=$(sbatch --job-name="${JOB_NAME}" ${_DEPENDENCY_ARG} \
                       train_job.sbat ${PARAMETERS} \
                                      --total_epochs=${TOTAL_EPOCHS} \
                                      --num_epochs=${EPOCHS_PER_JOB})
      _DEPENDENCY_ARG="--dependency=afterok:${_JOB_ID##* }"
    done
    ```
- Manage jobs manually
  - `squeue -u $USER`
  - `scancel <jobid>`
    ```

Best Practices: Clean Exit...

- Exit cleanly prior to job termination
  - Do not wait until job gets killed
    - Termination can cause checkpoint corruption (if it interrupts the copy operation)
  - Ensure ample time provided to job to complete work given (e.g., number of epochs)
  - Notify yourself in cases where jobs fail, timeout, or come close to running out of time:
    - `#SBATCH --mailtype=FAIL,TIMEOUT,TIMEOUT_90`
      `#SBATCH --mailuser=`<*username@kaust.edu.sa*>
- Use reference datasets
  - Eliminate dataset copy times — important for shorter jobs

Best Practices: Wrapping Up...

- Engage Us
  - `gpu_wide24` & `ref_32T` early adopters — provide feedback, ask questions
    - Queue wait times? Job overhead? Experience? Scaling?
  - Contact us:
    - Email: <ibex@hpc.kaust.edu.sa>
    - Slack: https://kaust-ibex.slack.com — #general, #gpu, #conda
- Example Projects
  - https://github.com/kaust-vislab/tensorflow-gpu-data-science-project
    - src/horovod-keras-example/train.py
  - https://github.com/kaust-vislab/pytorch-gpu-data-science-project
    - src/horovod-example/train.py