Week 3 Lab Session

Nalin Bhatt, Ethan Kozlowski, Max Zhu
Adapted from Adam Wu, Wonje Yun (Spring 2024)

April 11, 2025

University of Chicago

Contents

Review of Access

- Github setup
- Midway code editing
- Set up the necessary libraries at the base environment

Assignment Submission Example

- `git clone` assignment repository to Midway3
- Create folders so that assignments are clear
- Create separate python files if necessary (do not have to attempt to answer all in one file)
- Write down your code, run the code and get results
- Write down the `README` as described in `Canvas < Assignment 1` and link

Review of Numba/MPI/sbatch

- In `pyCC`, function signs are crucial. Need to figure out your input and output type.
- What happens when `mpirun` a job and between `comm.Scatter` and `comm.Gather`?
  - Unless `rank` is specified, every node receives the input and returns output
  - To have only one output, put the `print` or `return` under

```
if rank == 0:
    # YOUR CODE HERE
```

- `.sbatch` file contains the configuration of HPC hardware the user is requesting, and the command that it is requested to run.

For more SLURM commands see also: Slurm documentation

MPI for Python - 1

- MPI.COMM_WORLD: set up communication for mpi
- comm.Get_size(): return the number of processes
- comm.Get_rank(): return the rank of this process
- comm.Bcast(): broadcast a message from one process to all other processes in a group
- comm.Scatter(): scatter data from one process to all other processes in a group
- comm.Gather(): gather together values from a group of processes

For more explanation: mpi4py.MPI documentation and bcast/scatter/gather example

MPI for Python - 2

- comm.Bcast(buf, root=0): broadcast the data(`buf`) at rank 0 to all workers
- comm.Scatter(sendbuf, recvbuf, root=0): scatter data(`sendbuf`) at rank 0 to `recvbuf` at each workers.
- comm.Gather(sendbuf, recvbuf, root=0): gather the data(`sendbuf`) in each workers to the `recvbuf` at rank 0.
- If you want to do some global-like operation before scattering or after gathering, the operation should go under "`if rank == 0:`" statement.

Using GPU with OpenCL

PyOpenCL - 1

- `pyopencl` gives a pythonic access to `OpenCL` API.
- As in `mpi4py`, we need codes within the python function to make it run as an `opencl` job.
- `cl.create_some_context()`: create a opencl `context` 'somehow' (a device is chosen in an implementation-defined manner).
  - The `Context` is a construct that manages the set of devices, memory and command-queues.
- `cl.CommandQueue()`: create a command queue using the specified context

```
import pyopencl as cl
ctx = cl.create_some_context()
queue = cl.CommandQueue(ctx)
```

PyOpenCL - 2

- `pyopencl.array.Array`: The `numpy.ndarray` work-alike, so that it functions like an numpy array but operates on OpenCL. pyopencl.array.Array
- `pyopencl.elementwise.ElementwiseKernel`: Performs element-wise computation on the `pyopencl.array.Array` object.
  - context: the OpenCL context
  - arguments: the variable argument of the operation (need to assign data types, for arrays add * before the variable)
  - operation: the element-wise operation to perform
  - name: the function name of the compiled kernel
  - Ref: ElementwiseKernel

ElementwiseKernel Code Example

```python
import pyopencl.array as cl_array
from pyopencl.elementwise import ElementwiseKernel

ctx = cl.create_some_context()
queue = cl.CommandQueue(ctx)

a_g = cl.array.to_device(queue, a_np)
b_g = cl.array.to_device(queue, b_np)

lin_comb = ElementwiseKernel(ctx,
    "float k1, float *a_g, float k2, float *b_g, float *res_g",
    "res_g[i] = k1 * a_g[i] + k2 * b_g[i]",
    "lin_comb")

res_g = cl.array.empty_like(a_g)
lin_comb(2, a_g, 3, b_g, res_g)
```

PyOpenCL - 3

- `pyopencl.reduction.ReductionKernel`: Performs reduction operation (eg. count, sum, etc.) on the `pyopencl.array.Array` object.
    - context: the OpenCL context
    - dtypeout: the dtype of the output
    - neutral: initial value
    - map_expr: the entry operation of the vector
    - reduce_expr: the operation to perform on the result of map_expr
    - arguments: the input (needs at least one argument to be vector)
    - Ref: ReductionKernel

ReductionKernel Code Example

```python
import pyopencl.array as cl_array
from pyopencl.reduction import ReductionKernel

ctx = cl.create_some_context()
queue = cl.CommandQueue(ctx)

a = cl_array.arange(queue, 400, dtype=numpy.float32)
b = cl_array.arange(queue, 400, dtype=numpy.float32)

krnl = ReductionKernel(ctx, numpy.float32, neutral="0",
        reduce_expr="a+b", map_expr="x[i]*y[i]",
        arguments="__global float *x, __global float *y")

my_dot_prod = krnl(a, b).get()
```

GPU sbatch Commands

For using GPUs, different sbatch commands are needed.

- **–time**: set a limit on the total run time of the job allocation
- **–nodes**: number of GPU nodes to use
- **–partition**: set `gpu` for Midway3
- **–gres**: the number of gpus to request (`gpu:1` for 1 GPU usage)
- **–ntasks-per-node**: number of cores to drive each gpu
- **–mem-per-cpu**: minimum memory required per usable allocated CPU

For more, see also: RCC user guide for SLURM and Slurm documentation

GPU sbatch

```
#!/bin/bash

#SBATCH --job-name=gpu_mpi
#SBATCH --output=gpu_mpi.out
#SBATCH --error=gpu_mpi.err
#SBATCH --nodes=1 # using 1 GPU node
#SBATCH --ntasks-per-node=1 # 1 CPU node to drive the GPU
#SBATCH --partition=gpu # using the GPU on Midway3
#SBATCH --gres=gpu:1 # requesting only one GPU
#SBATCH --account=macs30123

module load cuda python

mpirun python3 ./<YOUR_FILE_NAME>.py
```

Question 1,2 Review

- Question 1 (a) explicitly asks you to use `pyCC`, not `jit`.
- To make the code running, it would be helpful to have the below code.

```python
if __name__ == "__main__":
    # YOUR CODE HERE
```

- Question 1 (b) asks you to compare the time speed comparison of up to using 20 cores, not more and not less.
- You should call the maximum number of nodes you are going to use in the sbatch.

Question 3 Objective

- **Objective:** compare the computation time and result between CPU and GPU
- Question (a): compare the computation time and result between CPU and GPU.
- Question (c): compare the computation time when the image is increased 10 times and 20 times.

Question 3 Cautions

- The main framework code and expected image of Q3 is given in the assignment description. You only need to make it into a code that can be run in OpenCL.
- The folder for accessing the input data are located in `/project2/macs30123/landsat8/`.
- **Only read the files in this directory, do not write on it**.
- **Restrict your CPU memory usage by 30G**: use `#SBATCH --meme-per-cpu=30G`
- **They are all in the assignment pdf, so please read carefully for the instructions before you run the code**

References i

- Lisandro Dalcin. "MPI for Python". Accessed April 5, 2024. https://mpi4py.readthedocs.io/en/stable/index.html
- Khronos OpenCL Working Group. "The OpenCL Specification." Accessed April 5, 2024. https://registry.khronos.org/OpenCL/specs/3.0-unified/html/OpenCL_API.html
- PyOpenCL. "pyopencl 2024.1 documentation." Accessed April 5, 2024. https://documen.tician.de/pyopencl/
- Research Computing Center. "RCC User Guide." Accessed March 20, 2024. https://rcc-uchicago.github.io/user-guide/
- SCHEDMD. "Slurm Workload Manager - sbatch." Accessed April 5, 2024. https://slurm.schedmd.com/sbatch.html/