Benchmarking is indispensable in HPC for optimizing performance, making informed resource decisions, and ensuring efficient resource utilization. It provides an understanding of system capabilities and helps in maintaining the efficiency and effectiveness of programs.

**Using Conda**

Benchmarking can be easily performed using the time -v command. However, its functionality is limited on the Hellgate Cluster. To overcome this limitation, we can create a Conda environment with the time package installed.

**Setting up Conda**

To enable Conda on your account, we utilize LMOD, a Lua based module system that handles configuration for us.

module load conda

conda init

We will need to refresh our session for Conda to apply it’s configuration, this can be done by either reloading our start script or logging out and logging back in.

To reload the start script:

source ~/.bashrc

Your prompt should now appear something like:

(base) [NetID@login1 ~]$

**Creating Conda Environment Time**

To install time, we will need to configure Conda to include the conda-forge channel which has the time installation:

conda config --add channels conda-forge

By default Conda will use an environment named ‘base’, since this is a shared environment we will not be able to change it, so we will need to create one that we can alter.

conda create -n benchmark

Now we should be able to install time into it by changing to the new environment then installing time:

conda activate benchmark

conda install time

We should now have access to use time -v when we are in the Conda environment. NOTE: We will need to specify which time, as there is an existing time on the Hellgate Cluster with limited functionality:

~/.conda/envs/<NAME>/bin/time -v sleep 30

**Batch Script**

Here is an example of an Sbatch script calling the Conda environment and benchmarking a command:

#!/bin/bash

#SBATCH --job-name=conda\_benchmark\_job

##SBATCH --mail-type=END,FAIL # Mail events (NONE, BEGIN, END, FAIL, ALL)

##SBATCH --mail-user= # Where to send mail

#SBATCH --partition='cpu(all)'

#SBATCH --nodes=1 #Number of nodes

#SBATCH --ntasks=10 # Number of simultaneous processes

#SBATCH --cpus-per-task=1 # Number of cores per process

#SBATCH --mem=32G # Memory (RAM)

#SBATCH --time=0-8 # Days-Hours

#SBATCH --output=output.txt

#SBATCH --error=err.txt

## Enable Conda environment for benchmarking

source /hellgate/home/$USER/.bashrc

conda activate benchmark

## Benchmark info will be in error file

## Command to be run:

~/.conda/envs/benchmark/bin/time -v sleep 60

**Time -v Information**

**User Time (seconds):** This measures the amount of CPU time spent in user mode. It's useful for understanding how much processing time your application requires.

**System Time (seconds):** This measures the amount of CPU time spent in kernel mode. It helps identify the overhead introduced by system calls and kernel operations.

**Percent of CPU this command got:** This indicates the CPU utilization percentage. High values suggest efficient CPU usage, while low values might indicate bottlenecks or inefficiencies.

**Elapsed (wall clock) time:** This is the total time taken from start to finish of the command. It's crucial for understanding the real-world time performance of your application.

**Maximum Resident Set Size (RSS)(kbytes):** This shows the peak memory usage of the process. It's important for ensuring that your application fits within the available memory resources.

**Major Page Faults:** These occur when the system has to read data from disk because it wasn't found in memory. High numbers can indicate memory pressure or inefficient memory usage.

**Minor Page Faults:** These occur when the system can resolve the fault without reading from disk, usually by reclaiming a frame. While less severe than major faults, high numbers can still indicate memory inefficiencies.

**Voluntary Context Switches:** These happen when a process willingly yields the CPU, often due to waiting for I/O operations. High numbers can indicate I/O-bound processes.

**Involuntary Context Switches:** These occur when the system forcibly switches the CPU from one process to another, often due to CPU contention. High numbers can indicate CPU-bound processes or contention issues.

**File System Inputs/Outputs:** These metrics show the amount of data read from and written to the file system. They are useful for identifying I/O bottlenecks.

**Exit Status:** This indicates whether the command completed successfully (0) or encountered an error (non-zero). It's essential for verifying the success of your benchmarking runs.

**Using SLURM Accounting**

SLURM has accounting functionality through the sacct command, this provides information about jobs with summaries. The command will only provide full information after the job has completed.

sacct -j <JOBID> --format=jobid,jobname,elapsed,totalcpu,avecpu,cputime,maxvmsize,maxrss

**Sacct Information**

There are many different fields which can also be displayed by sacct, in the command above we specify limited information:

**jobid:** The unique identifier for the job.

**jobname:** The name of the job.

**elapsed:** The total elapsed time for the job, from start to finish.

**totalcpu:** The total CPU time used by all tasks in the job.

**avecpu:** The average CPU time used per task.

**cputime:** The total CPU time used, which is the sum of the CPU time for all tasks.

**maxvmsize:** The maximum virtual memory size used by the job.

**maxrss:** The maximum resident set size (physical memory) used by the job.