Reducing Remote GPU Execution’s Overhead with mrCUDA

Pak Markthub, Akihiro Nomura and Satoshi Matsuoka

Tokyo Institute of Technology

Background

Our previous work [1] addressed the scattered idle-GPU problem in multi-GPU batchQueue systems, which cause the systems to have idle GPUs despite having jobs waiting. Our solution was to virtually consolidate unoccupied GPUs into some nodes running remote GPU execution (e.g. rCUDA [2]) to serve more jobs.

How mrCUDA Works

mrCUDA uses Replay Method [3] proposed by Nakada et al. with some modification to reproduce remote GPUs’ states and memory to local GPUs.

Fig 1: Virtually consolidating unoccupied GPUs to solve the scattered idle-GPU problem

Problem

Communication-intensive GPU applications usually experience detrimentally large remote GPU execution’s overhead. The equation below shows that rCUDA’s overhead depends on each application’s and network’s characteristics.

\[
time_{rCUDA} = \text{latency}_{rCUDA} \cdot \text{kernel}_{call} + \text{data}_{size} \cdot \text{coefficient}_{rCUDA}
\]

\[
\text{coefficient}_{rCUDA} = 1.03, \text{latency}_{rCUDA} = 50.62\mu s
\]

rCUDA’s overhead dominates the execution time

Fig 2: Comparison of LAMMPPS’s execution time when using a local GPU and a remote GPU with rCUDA

Proposal

Minimize remote GPU execution’s overhead

A local GPU may become available because a job sharing the same node finishes

mrCUDA: very low overhead middleware for migrating work from remote to local GPUs

Case Study: LAMMPS

mrCUDA can reduce LAMMPPS’s execution time as much as 60%, 40%, and 20% when it migrates right after LAMMPS finished 25%, 50%, and 75% of the total iterations respectively.

Fig 3: Comparison of LAMMPPS’s execution time when using mrCUDA to migrate remote CUDA execution to a local GPU at various migration points

Case Study: MRQ Scheduling Algorithm

- In our previous work [1], we proposed the RQ scheduling algorithm, an improved version of the first-come-first-serve (FCFS) scheduling algorithm that uses rCUDA to virtually consolidate unoccupied GPUs to serve more jobs. By using RQ, systems can reduce GPU jobs’ wait time as much as 25% while increasing jobs’ execution time less than 0.01% on average.
- However, for GPU-communication-intensive job sets and busy systems, rCUDA’s overhead may degrade the performance of RQ.
- We proposed an improved version of RQ called MRQ that uses mrCUDA instead of rCUDA, and migrates remote GPU execution as soon as a local GPU becomes available.
- Lifetime = Wait Time + Execution Time
- Lifetime Decrease on RQ/MRQ is a job’s lifetime when using RQ/MRQ compared with the same job’s lifetime when using FCFS.

Fig 4: Comparison of jobs’ lifetime decrease when using RQ and MRQ on various number of GPU invocations

Conclusion

- mrCUDA has low migration overhead (< 5% of rCUDA’s overhead), and a job suffers from migration overhead at most once per remote GPU.
- mrCUDA can greatly improve applications’ performance compared with keeping using remote GPUs (40% for LAMMPS given the migration point is 50% of LAMMPS’s total iterations), and can improve our solution to the scattered idle-GPU problem to handle job sets whose GPU communication intensity is 10-fold higher.

References

