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 batch-queue systems, which cause the systems to have idle GPUs despite having jobs waiting. Our solution was to virtually consolidate unoccupied GPUs into some nodes using remote GPU execution (e.g. **rCUDA** [2]) to serve more jobs.

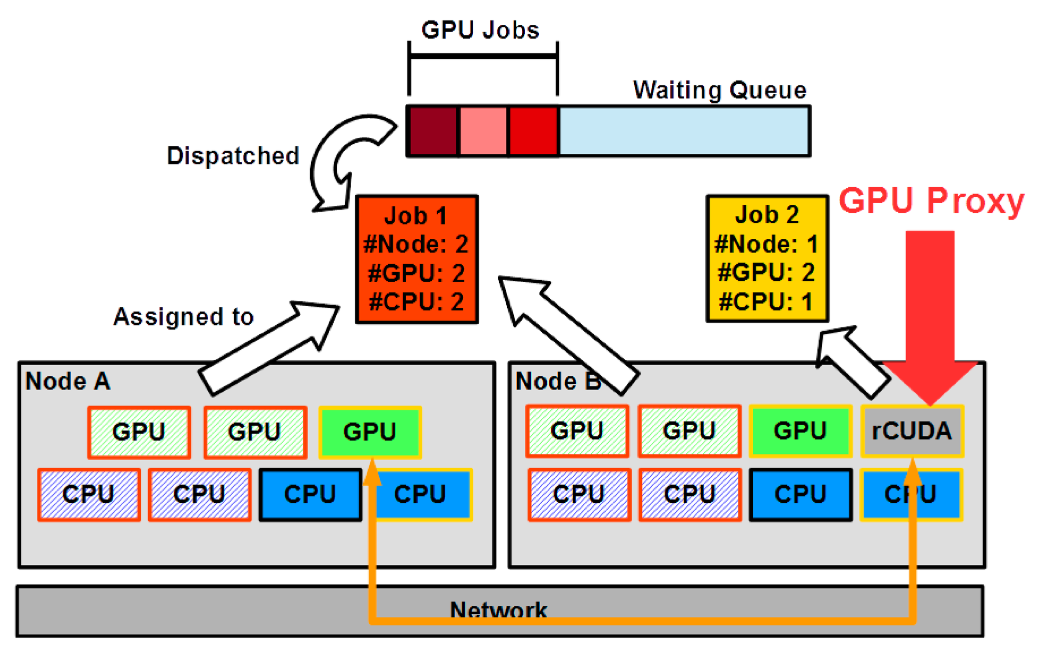


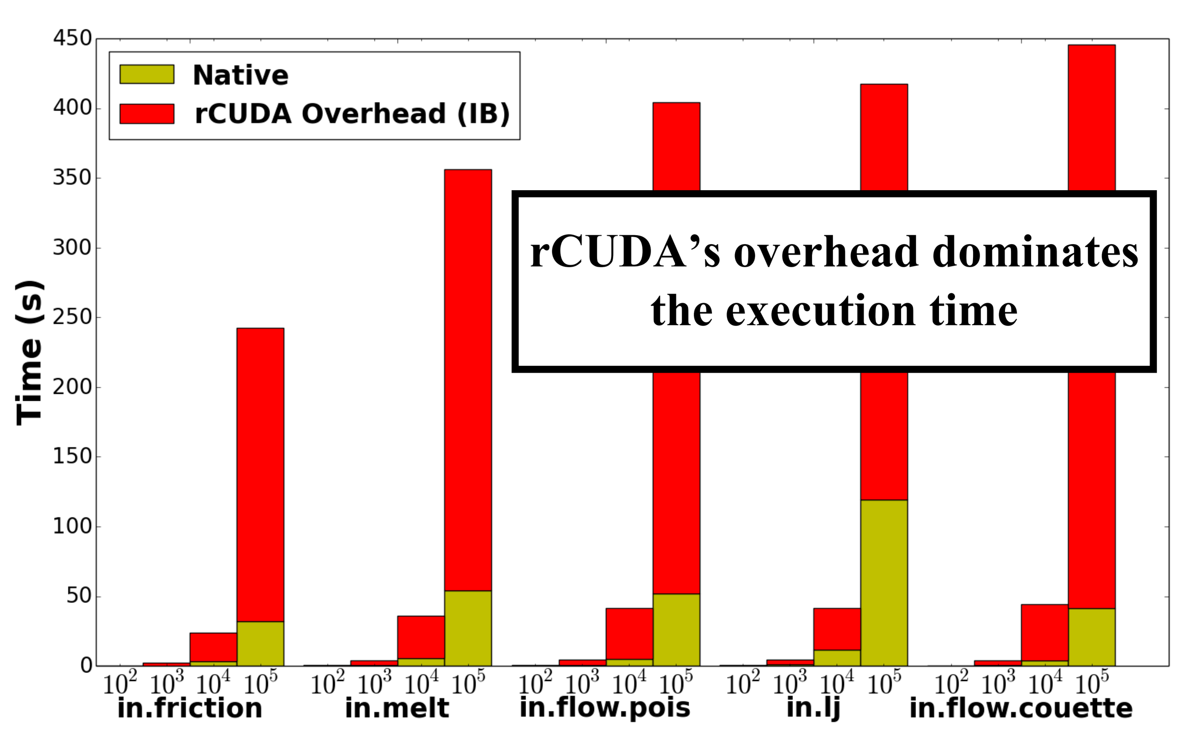
Fig 1: Virtually consolidating unoccupied GPUs to solve the scattered idle-GPU problem

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.

$$\text{time}_{rcuda} = (\text{latency}_{rcuda} + \text{latency}_{net})\text{gpu\_call\_count} + \frac{\text{datasize}}{\text{bw}_{eff}}\text{coef}_{rcuda}$$

$$\text{coef}_{rcuda} = 1.03, \text{latency}_{rcuda} = 50.62\ \mu\text{s}$$ ← **More than 5 times larger than CUDA's latency**



* **Major x-axis:** input file; **Minor x-axis:** total iterations

* **Network:** InfiniBand 4xFDR; **GPU:** NVIDIA Tesla K20c

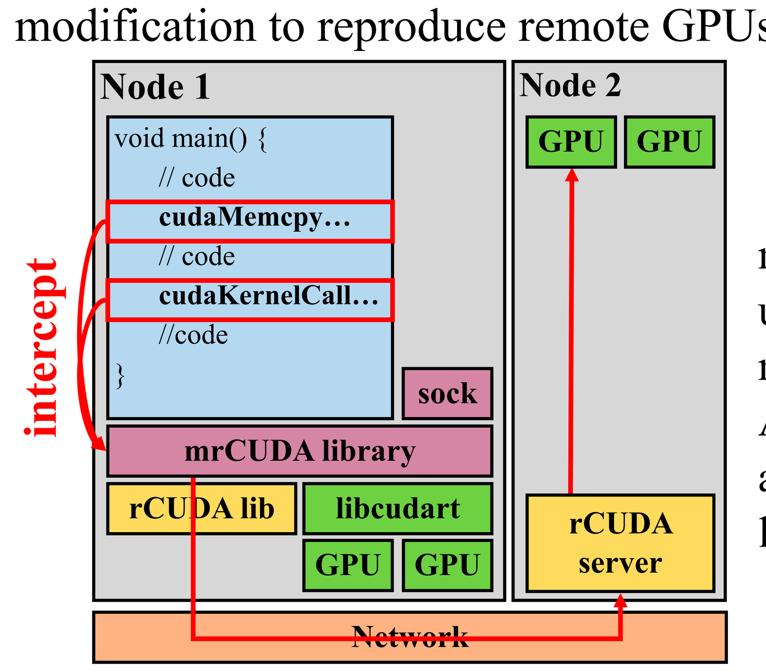
Fig 2: Comparison of LAMMPS's execution time when using a local GPU and a remote GPU with rCUDA

Proposal

Objective	Minimize remote GPU execution's overhead
Motivation	A local GPU may become available because a job sharing the same node finishes
Proposed Solution	**mrCUDA:** very low overhead middleware for migrating work from remote to local GPUs

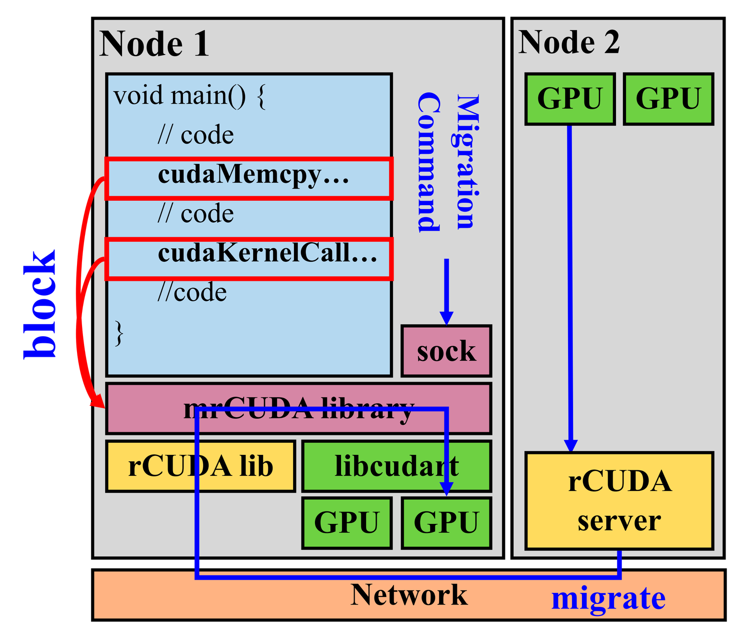
How mrCUDA Works

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



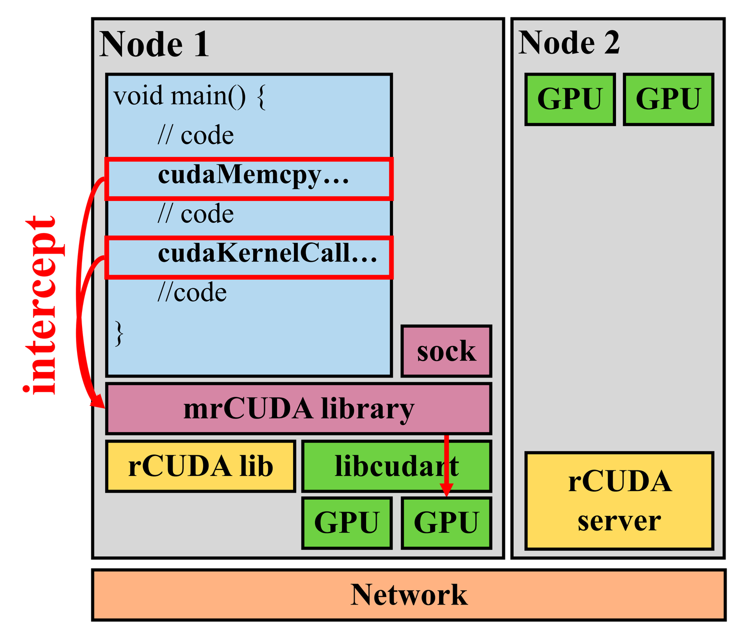
First: rCUDA Pass-through Mode

mrCUDA allows CUDA applications to use remote GPUs while preparing for migration. It intercepts CUDA Runtime API calls, **records** a subset of those calls, and passes all of them to the rCUDA library.



Then: Migration Mode

mrCUDA seamlessly migrates remote CUDA execution to the selected local GPU. It temporary blocks all further CUDA Runtime API calls, waits for the running calls to finish, **replays** the recorded calls to make the local GPU's states and active memory regions the same as the remote GPU's, and copies **(mem-sync)** GPU data to the local GPU.

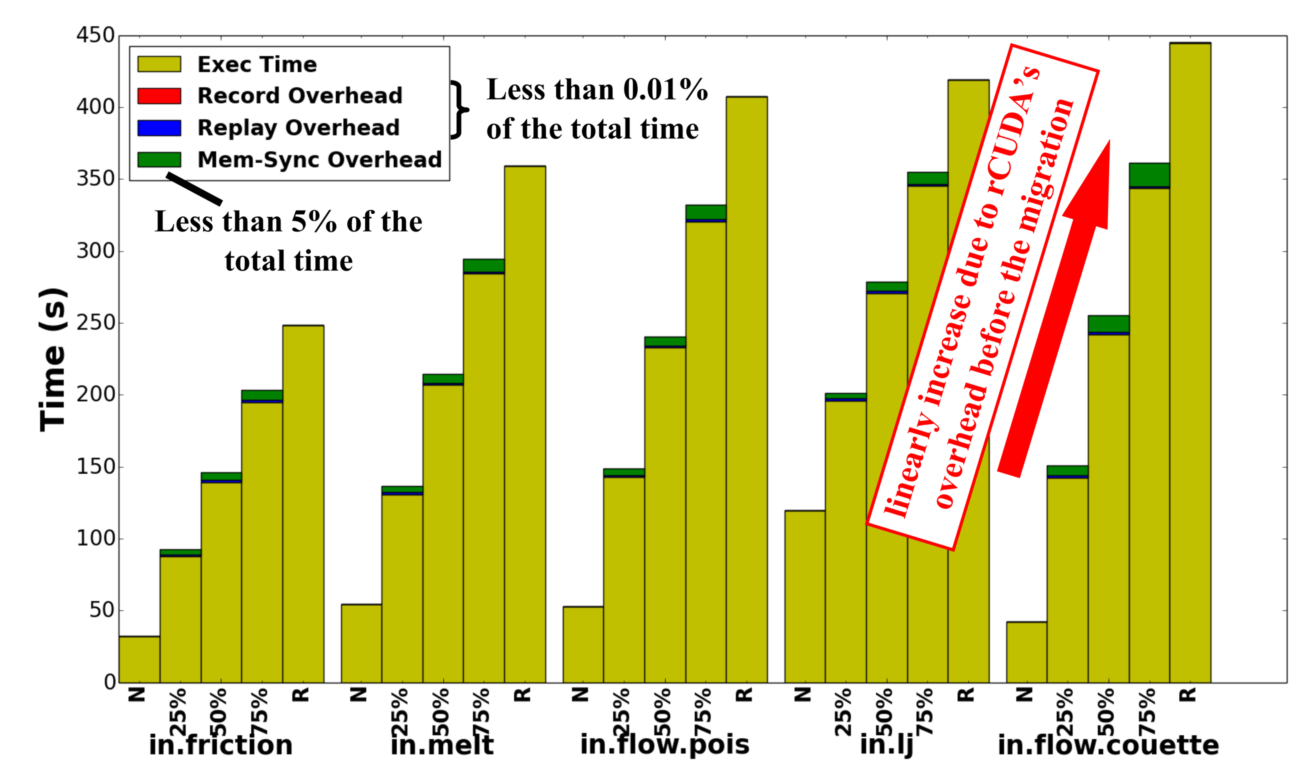


Last: CUDA Pass-through Mode

mrCUDA uses the local GPU to execute further intercepted CUDA Runtime API calls. Since mrCUDA completely cuts off remote GPU execution, it greatly reduces further CUDA Runtime API calls' overhead as well as network congestion, which also benefits other jobs on the system.

Case Study: LAMMPS

mrCUDA can reduce LAMMPS'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.



* **N:** used native CUDA; **R:** mrCUDA w/o migration; **x%:** migrated when LAMMPS finished x% of its total iterations (10^5)

Fig 3: Comparison of LAMMPS'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.

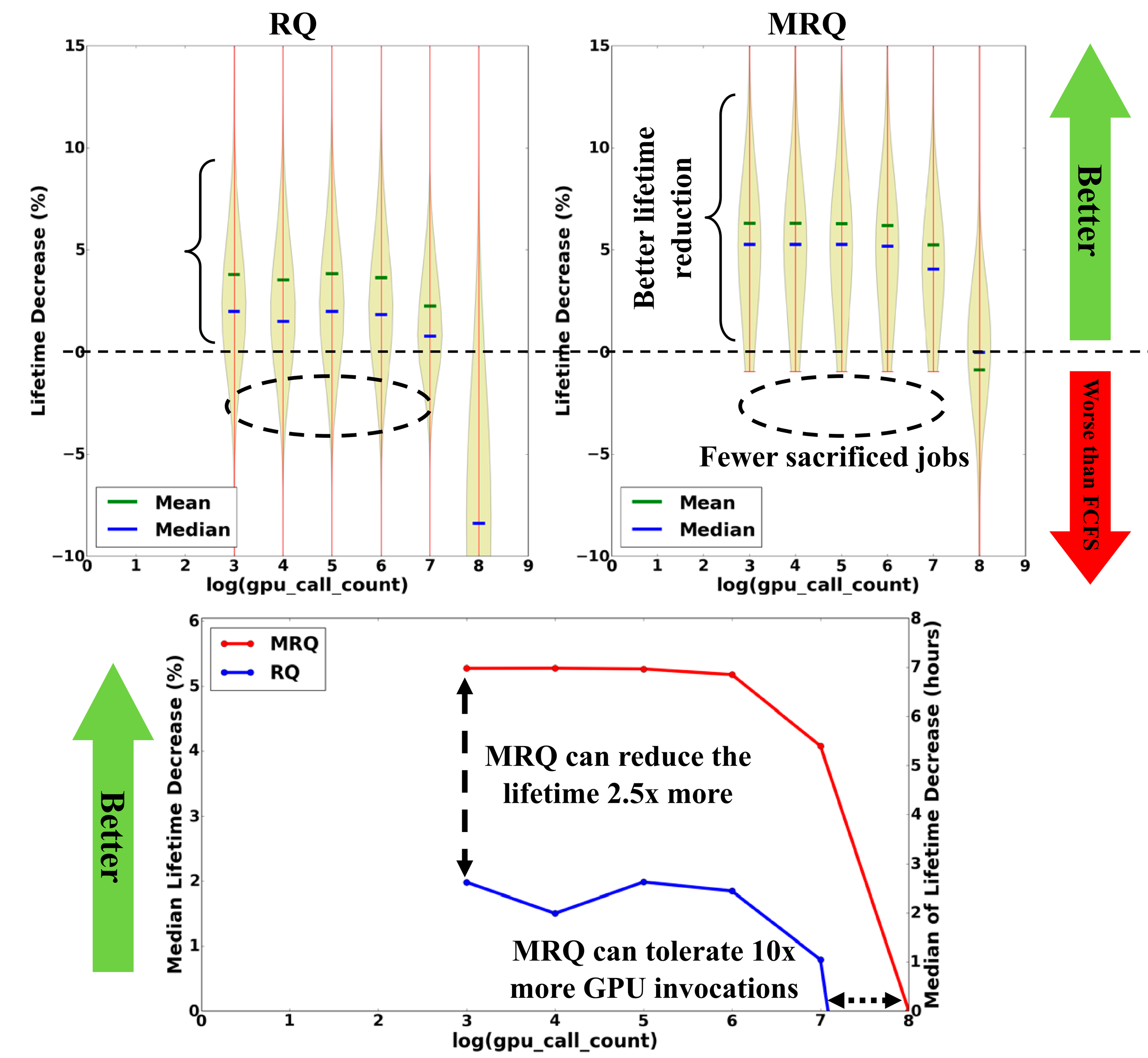


Fig 4: Comparison of jobs' lifetime decrease when using RQ and MRQ on various number of GPU invocations

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

[1] Pak Markthub, Akihiro Nomura, and Satoshi Matsuoka. Using rCUDA to reduce GPU resource-assignment fragmentation caused by job scheduler. In Parallel and Distributed Computing, Applications and Technologies (PDCAT2014), 2014 15th International Conference on, pages 105–112. IEEE, 2014.

[2] J Duato, FD Igual, and R Mayo. An efficient implementation of GPU virtualization in high performance clusters. Euro-Par 2009 Parallel Processing Workshops, pages 385–394, 2010.

[3] Akira Nukada, Hiroyuki Takizawa, and Satoshi Matsuoka. NVCR: A transparent checkpoint-restart library for NVIDIA CUDA. In Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), 2011 IEEE International Symposium on, pages 104–113. IEEE, 2011.