Spring 2021: ME759 Final Project Proposal

Project Title (state the title): (tentative) Cautiously Aggressive GPU Sharing

Link to git repo for project (this is the link to your ME759 GitLab repo):
https://euler.wacc.wisc.edu/ruipeterpan/me759-ruipeterpan

Problem statement (explain in clear terms what you plan to accomplish)

We want to improve the GPU utilization of a shared cluster by aggressively packing multiple workloads that utilize different types of cores on the same GPU concurrently.

In shared GPU clusters, users submit jobs to a scheduling framework. The scheduler then puts the jobs submitted in a queue and allocates resources to the jobs when it deems fit. One optimization current GPU cluster scheduling frameworks are not considering is that they do not push the sharing of GPU resources to the limit. We aim to break the current allocation model, where each job gets exclusive access to one or more whole GPUs during its whole training time, and use aggressive time-sharing/space-sharing of GPUs to pack multiple jobs on the same GPU concurrently. We hope to improve the overall cluster efficiency & utilization, and the makespan & average Job Completion Time (avg JCT) of all jobs in a workload trace. To summarize, the pieces that contribute to this project include:

(1) A better understanding of GPU utilization than the current, coarse estimation provided by State-Of-The-Art (STOA) GPU monitoring tools

(2) A fine-grained profiling of how different GPU cores get utilized in common Deep Learning (DL) / High Performance Computing (HPC) workloads

(3) The speedup provided by the NVIDIA Multi-Process Service (MPS), which improves utilization when multiple processes share the same GPU by allowing multiple CUDA processes to share a single GPU context, particularly for Volta architectures

Motivation/Rationale (explain why you chose to work on this project)

Almost all State-Of-The-Art (STOA) GPU utilization monitoring tools used are wrapped around the NVIDIA System Management Interface (nvidia-smi), which is developed on top of the NVIDIA Management Library (NVML). However, the GPU utilization provided by NVML is a very coarse, upper-bound estimation of the utilization: the GPU utilization reported by NVML is defined as the "percent of time over the past sample period during which one or more kernels was executing on the GPU". As a result, at a certain point in time, as long as 1 kernel is being executed on the GPU, NVML considers the GPU to be fully utilized. However, modern GPU architectures exhibit high versatility by including a myriad combination of cores. For example, a full NVIDIA GV100 GPU (84 SMs) has a total of 5376 FP32 cores, 5376 INT32 cores, 2688 FP64 cores, 672 Tensor Cores, and 336 texture units. For some GPU workloads (e.g., full FP32 precision), a considerable amount of these cores (e.g., FP16/64) are idle and not utilized properly. In current large-scale GPU computing clusters, the GPUs are allocated exclusively in a one-off fashion in the sense that they are not shared by multiple processes concurrently. However, with the bullet points listed above, the cluster schedulers can enact policies that pack multiple workloads on the same GPU concurrently to fully utilize the whole set of resources provided by a GPU. In conclusion, current GPU monitoring tools do not provide fine-grained monitoring of GPU core utilization, and there is possible potential for packing multiple jobs on a GPU concurrently so as to fully utilize its resources.

Explain how you contemplate going about it (indicate if you'll use GPU/OpenMP/MPI parallel computing, what libraries, etc. Indicate what algorithms/approaches you are considering.)

- Note that I might include more content in the final report if things go well, but I hope to cover at least the following

- Background
    - First, I will provide a brief introduction to the popular workloads and benchmarks used in DL and HPC that utilize GPUs
    - Then, I will briefly introduce the existing GPU profiling/monitoring tools, and their underlying mechanisms & pros/cons
    - At last, I will also introduce the existing model of how STOA GPU cluster managers allocate GPUs for incoming jobs
- Experiments
    - I will first locate a trace of DL/HPC jobs that is representative of jobs in a cluster
    - Then, I will use GPU profiling and monitoring tools to extract core utilization characteristics on the trace of workloads
    - At last, I will present microbenchmarks that confirms our hypothesis about the benefits of aggressive GPU sharing
- Implementation
    - I hope to implement a profiler that schedulers can use to profile a submitted job and cautiously determine its potential for packing by evaluating the utilization of different cores
    - If the profiler is nicely done, I hope I can implement a simulator for running a trace of workloads to evaluate the end-to-end benefits and prove our hypothesis

ME759 aspects the proposed work draws on:

- GPU Computing
- GPU Profiling
- Deep learning workloads/libraries that utilize CUDA and MPI
- HPC workloads that utilize CUDA/OpenMP/MPI

Team member[s]:

- Name: Rui Pan
- Email: rpan33@wisc.edu
- Home department: Computer Science
- Advisor: Shivaram Venkataraman (I'm an undergraduate student so I don't have an actual advisor but I'm doing research with Shivaram)
- Student's role in the project: Solo project :|

Deliverables (what you expect to deliver on 05/07/2021, 10:05 AM: code, input files, tech report, etc.)

- A technical report in the style of a conference research paper
- All source code for replicating the results in the report
- Microbenchmark results, either attached in the appendix of the technical report or delivered in a separate document

How you will demonstrate what you accomplished: this is particularly important if what you do is a small piece of a bigger project that you will continue to pursue after wrapping up ME759.
I will demonstrate the speedups by presenting microbenchmark experiments results and hopefully come up with a simulator of a scheduler. If the results of the microbenchmarks/simulation prove to be good, the next step will be to work on a physical implementation either from the ground up or as integration into existing scheduling frameworks to demonstrate the actual effectiveness in a real-world, end-to-end setup.