Workload-aware scheduling techniques for general purpose applications on Graphics Processing Units

In the last decade, there has been a wide scale adoption of Graphics Processing Units (GPUs) as a co-processor for accelerating data-parallel general purpose applications. A primary driver of this adoption is that GPUs offer orders of magnitude higher floating point arithmetic throughput and memory bandwidth compared to their CPU counterparts. As GPU architectures are designed as throughput processors, they adopt a manycore architecture with 10 to 100s of cores, each with multiple vector processing pipelines. A significant amount of the die area is dedicated to floating point units, at the expense of not having hardware units used for memory latency hiding in conventional CPU architectures. The quintessential technique used for memory latency tolerance is exploiting data-level parallelism in the workload, and interleaving execution of multiple SIMD threads, to overlap the latency of threads waiting on data from memory with computation from other threads.

With each architecture generation, GPU architectures are providing an increasing amount of floating point throughput and memory bandwidth. Alongside, the architectures support an increasing number of simultaneously active threads. The work proposed in this research envisions, to continue making advancements in GPU computing, workload-aware scheduling techniques are required. Precisely, in the GPU computing work flow, scheduling is performed at three levels - the system or chip level, the core level and the thread level. This research proposes novel scheduling techniques at each of the three levels. We show that GPU computing workloads have significantly varying characteristics, and design techniques that monitor the hardware state to aide at each of the three levels of scheduling. Each technique is implemented in a cycle level GPU architecture simulator, and their effect on performance is analyzed against state of the art scheduling techniques used in current GPU architectures.