GPUs have been widely used to parallelize and accelerate applications for its high throughput. Traditionally, a GPU function can only be launched from the CPU side. This results in the fact that GPUs are preferable for those application which express a flat data parallelism, a simple data parallelism that is known at compiling time and can be easily distributed to different GPU blocks and threads. However, for those applications that contain nested data parallelism, which is not known a priori and can only be discovered at running time, it is difficult to write a GPU function that achieve high performance on parallelization and acceleration. One can easily end up with either a too coarse-grained or too fine-grained GPU function. Since Kepler architecture, Nvidia introduced a new feature--Dynamic Parallelism (DP), which enables the initiation of GPU functions from inside a GPU function. This makes the nested parallelism easy to be explored on GPU since one can program in a way that a new GPU function can be launched whenever a local nested parallelism is met during the execution. What is more, DP makes implementing recursion on GPU without the intervention of CPUs possible.

Many computations exhibit a pattern of nested data parallelism and among those is parallel recursion. However, preliminary data shows that simple DP-based implementations of recursion result in poor performance. This work focus on how to efficiently exploit DP for parallel recursive applications on GPU. Specifically, the goal is to free the users from programming with the complexity of GPUs’ hardware and software and to automatically generate high performance GPU recursive functions implemented with DP given the inputs of simple parallel CPU recursive functions. To this end, first, I propose several DP-based parallel recursive templates that can be generated from a serial CPU recursive function. I compare the parallel recursive templates with non DP-based counterparts (flat kernels) to see if using DP in parallel recursive application can be beneficial or not. Second, to reduce the overhead of DP, I propose compiler techniques that improve the efficiency of simple DP-based parallel recursive functions by performing workload consolidation.

My evaluation shows that GPU kernels consolidated with the proposed code transformations achieve an average speedup in the order of 1500x over basic implementations using DP and an average speedup of 3.9x over optimized flat GPU kernels for both tree traversal and graph based applications.