Structured programming consists of base constructs that represent how programs are written. When optimizing programs, compilers typically operate on the intermediate representation (IR) of a control flow graph (CFG), which is derived from program source code analysis and represents basic blocks of instructions (nodes) and control flow paths (edges) in the graph. Thus, the overall program structure is captured in the CFG and the IR abstracts machine-specific intrinsics that the compiler ultimately translates to machine code. In particular, compilers can benefit from prior knowledge of optimizations that may be effective for specific CFG structures.

Objectives

- Systematic process to construct control flow graphs for GPU kernels
- Techniques to perform subgraph matching on various kernel CFGs and GPUs
- Approaches to reveal thread divergence behavior based on CFG properties

Results

- Transition probability matrices calculated for each kernel subgraph
- Spline interpolation employed to scale transition matrix before performing pairwise comparisons
- Affinity scores for CFGs ($S_1$ and $S_2$ for $G_1$ and $G_2$) matched via similarity measures (IsoRank, Euclidean)
- Methodology evaluated on 3 GPUs (Kepler, Maxwell, Pascal)

Future Work

- Incorporate memory reuse distance statistics of a kernel to help optimize memory subsystem and expose compute/memory overlaps
- Characterize deep learning workloads to optimize placement of tasks on multi-node multi-GPU setups

Acknowledgements

- NVIDIA for providing access to PSG Clusters
- American Society for Engineering Education (ASEE)