Abstract

The effective use of GPUs for accelerating applications depends on a number of factors including effective asynchronous use of heterogeneous resources, reducing data transfer between CPU and GPU, increasing occupancy of GPU kernels, overlapping data transfers with computations, reducing GPU idling and kernel optimizations. Overcoming these challenges require considerable effort on the part of the application developers. Most optimization strategies are often proposed and tuned specifically for individual applications.

Message-driven executions with over-decomposition of tasks constitute an important model for parallel programming and provide multiple benefits including communication-computation overlap and reduced idling on resources. Charm++ is one such message-driven language which employs over decomposition of tasks, computation-communication overlap and a measurement-based load balancer to achieve high CPU utilization. This research has developed an adaptive runtime framework for efficient executions of Charm++ message-driven parallel applications on GPU systems.

In the first part of our research, we have developed a runtime framework, G-Charm with the focus primarily on optimizing regular applications. At runtime, G-Charm automatically combines multiple small GPU tasks into a single larger kernel which reduces the number of kernel invocations while improving CUDA occupancy. G-Charm also enables reuse of existing data in GPU global memory, performs GPU memory management and dynamic scheduling of tasks across CPU and GPU in order to reduce idle time. In order to combine the partial results obtained from the computations performed on CPU and GPU, G-Charm allows the user to specify an operator using which the partial results
are combined at runtime. We also perform compile time code generation to reduce programming overhead. For Cholesky factorization, a regular parallel application, G-Charm provides 14% improvement over a highly tuned implementation.

In the second part of our research, we extended our runtime to overcome the challenges presented by irregular applications such as aperiodic generation of tasks, irregular memory access patterns and varying workloads during application execution. We developed models for deciding the number of tasks that can be combined into a kernel based on the rate of task generation, and the GPU occupancy of the tasks. For irregular applications, data reuse results in uncoalesced GPU memory access. We evaluated the effect of altering the global memory access pattern in improving coalesced access. We’ve also developed adaptive methods for hybrid execution on CPU and GPU wherein we consider the varying workloads while scheduling tasks across the CPU and GPU. We demonstrate that our dynamic strategies result in 8-38% reduction in execution times for an N-body simulation application and a molecular dynamics application over the corresponding static strategies that are amenable for regular applications.