Data placement and movement in a heterogeneous memory environment

Heterogeneous memory architectures are increasingly becoming more prominent in upcoming HPC systems. High Bandwidth Memory (HBM) is available on most GPGPUs to enable fetching of large amount of data from DRAM based memory. Non-volatile, byte-addressable memory (NVM) has been introduced by Intel in the form of NVDIMMs named Intel® OptaneTM DC PMM. This memory module has the ability to persist the data stored in it without the need for power. These memory technologies expand the memory hierarchy into a hybrid/heterogeneous memory system due the differences in access latency and memory bandwidth from DRAM, which has been the predominant byte-addressable main memory technology. The Optane DC memory modules have up to 8x the capacity of DDR4 DRAM modules which can expand the byte-address space up to 6 TB per node. Many applications can now scale up their problem size given such a memory system. Although, the complexity of data allocation, placement and movement can make the use of heterogeneous memory difficult for HPC application programmers. Existing codes were written for homogeneous memory system and rewriting them would be a big challenge in terms of portability. Our aim is to move the onus from the application programmer to the compiler to modify and adapt the HPC applications for a heterogeneous memory system.

The compiler framework analyzes the code at the IR level. It narrows down on the dynamic allocations in the code and replaces the allocation function calls like malloc()/realloc() with equivalent function calls from the SICM library. SICM (Simple Interface Complex Memory) library is an interface to allocate memory on different memory devices available on a given compute node. It is a bare-metal library that utilizes NUMA and jemalloc libraries to create arenas where memory can be allocated and the arenas can be moved between the different memory devices. However, it required an initialization a finalizing procedure before allocating memory in a program which could be a daunting task. Also, the SICM library was not performance-aware, i.e. it was not able to classify the NUMA nodes based on their memory performance.

We added a small memory characterization script that ran micro-benchmarks to measure the memory performance of each NUMA node for every CPU group and also measured the memory transfer speed between NUMA nodes for every CPU group. This classification is read by the SICM library during runtime to be completely performance-aware in a heterogeneous memory system. We introduced additional wrapper APIs that would enable easy allocation based on the required performance from the system for a given data structure.

The compiler framework utilizes these wrapper APIs to transform the existing codes to use the SICM library and allocate memory in a performance aware manner on a heterogeneous system. It consists of a transformation pass that analyzes the code on a Module, Function and Instruction level to narrow down on the initialization, finalization and allocation/free points in the code. It then inserts the IR code equivalents of the SICM APIs to transform the code. The transformed IR or bitcode is then linked into an executable and then executed on the targeted system.

LA-UR-20-25790