Performance Analysis of the Sparse Grid Technique on Hybrid Systems
Alin Murarasu, Daniel Butnaru, Gerrit Buse, Dirk Pflüger, Josef Weidendorfer

1. Multi-cores, Accelerators and High-Dimensional Data
A recent HPC trend is reflected in the increasing use of heterogeneous systems containing general-purpose multi-core CPUs and accelerators, e.g. GPUs. CPUs provide a complex logic for branch prediction, out-of-order execution, and data prefetching. In contrast, GPUs are high-throughput processors and require regular parallelism, present for instance in vectorizable codes. On hybrid systems, the goal is two-fold: (1) keep the processors busy and (2) distribute the work between processors according to their characteristics. It is not trivial to reach this goal and, depending on the application, the simultaneous usage of CPUs and GPUs may not be possible.

In our application we want to explore high-dimensional simulation data. To be able to cope with the amount of output generated by high-dimensional settings (i.e. 4 - 10 dimensions and more), we employ the sparse grid technique to reduce the size of the data. The technique offers a way to compress grid data in a slightly lossy compression step. The key operations are “compression” and “decompression” (Fig. 1). These two operations serve us as benchmarks for the hybrid systems in our application.

3. The Sparse Grid Technique on Multi-core CPUs and GPUs
Developing programs for hybrid systems implies writing multiple implementations tuned for various processors, e.g. for CPUs and GPUs (Fig. 3). Our set of optimizations for multi-core CPUs include cache and SIMD optimizations. The cache optimization accelerates decompression up to 1.5x, whereas the SSE optimization provides a speedup of 2x. Note that even in its optimized form, the irregular memory access pattern of compression is highly incompatible with cache and SSE optimizations.

On GPUs, the inherent space and time optimizations of our algorithms are complemented by a set of GPU specific optimizations: use of shared memory (4x faster decompression), use of constant cache (4x compression speedup), high occupancy to hide pipeline stalls (up to 1.6x), bank conflict optimizations (up to 1.3x), and coalesced accesses to device’s global memory.

4. Hybrid Computing, Challenge and/or Opportunity?
Simultaneously using CPUs and GPUs in an application is not always feasible. Sparse grid compression is such an example. Due to its poor data locality determined by accessing dependencies, compression is not suited for a system with multiple address spaces as one composed of CPUs and GPUs. In this case, while one GPU compresses the data, the other processors are idle. In contrast, decompression is embarrassingly parallel, has a computational intensity of \(O(\text{dimensionality})\), has a packed and predictable memory access pattern, and, hence, is a perfect match for hybrid systems with CPUs and GPUs. For decompression, efficient load balancing can be achieved by assigning work to the CPUs and GPUs based on their FLOPS rates represented as functions of three parameters: dimensionality, refinement level, and number of evaluations. This approach provides an efficiency of 98% (Fig. 4).