Structure-based Optimizations for Sparse Matrix-vector Multiply

We introduce two novel techniques to improve the performance of Sparse Matrix-vector Multiply (SMVM) kernels, which dominate the runtime of most iterative solvers. SMVM is historically known as a challenging kernel that achieves only small fractions of peak CPU speeds. Sparse representation formats are often to blame, since they increase memory bandwidth usage, impose irregular accesses to memory, and cause short loops that lead to inefficient pipelining.

We observe that most matrices share identical structures or sub-structures, therefore a large part of their representation data can be eliminated by identifying and exploiting recurring patterns, mainly to reduce memory bandwidth usage, the primary limiter on the performance. In addition, these methods take advantage of micro-level optimizations, such as vectorization, loop unrolling and software prefetching.

We consider two broad cases in which redundant matrix structure occurs. The first is when multiple matrices of problems running in a large ensemble share an identical structure. To improve performance, we solve these problems simultaneously, such that all problems share only one copy of the indexing information to reduce the memory bandwidth usage. Since we ‘stack’ several problems to reduce bandwidth usage, we refer to our framework as **Operation Stacking Framework (OSF)**. Our evaluation of OSF yielded performance improvements by up to 1.94x on an AMD Opteron compared the widely used CSR method. We show that OSF implementation costs are amortized after as few as five iterations. We validate performance results using hardware counters that demonstrates significant improvements in cache and pipeline utilization.

The second case is when a single matrix can be decomposed into blocks that include recurring nonzero patterns. We generate a custom code for each recurring block pattern; therefore no indexing data are read from memory for each of the individual nonzero locations, thereby significantly reducing the memory bandwidth usage (by up to 98%). We refer to this method as **Pattern Based Representation (PBR)**. Our custom code generator emits highly tuned codes by automatically implementing SSE vectorization, software prefetching and loop unrolling. We describe a performance predictor model to identify a blocksize that achieves an optimal or near-optimal performance. Our evaluation using two recent multicore AMD and Intel machines demonstrates performance improvements by up to 3x (sequential) and 5x (in parallel with 8-cores) compared to the CSR method. We developed a library that performs matrix conversions on the fly, allowing our method to be used as a drop-in replacement for existing methods. We compare PBR overheads relative to its benefits and show that PBR is beneficial for many applications, which repetitively call the SMVM kernel for the same matrix structure.