

When Cache Blocking Sparse Matrix Vector Multiply Works and Why

Rajesh Nishtala Richard Vuduc James W. Demmel Katherine A. Yelick

Computer Science Division
University of California, Berkeley
{rajeshn,richie,demmel,yelick}@cs.berkeley.edu

January 29, 2004

We consider the problem of building high-performance implementations of sparse matrix-vector multiply (SpM×V), or $y = y + A \cdot x$, which is an important and ubiquitous computational kernel. Prior work indicates that cache blocking of SpM×V is extremely important for some matrix and machine combinations, with speedups as high as 3x. In this paper we present a new, more compact data structure for cache blocking for SpM×V and look at the general question of when and why performance improves. Cache blocking appears to be most effective for large dimensional matrices, such that the vector x does not fit in cache, and for matrices with little or no structure in the pattern of non-zeros. Prior work on performance modeling assumed that the matrices were small enough so that x and y fit in the cache. However when this is not the case, there maybe significant drops in performance implying that the original models are insufficient as a basis for a heuristic for picking the optimum block size for large matrices, thus we also present updated performance models.

Making SpM×V fast is complicated both by modern hardware architectures and by the overhead of manipulating sparse data structures. Indeed, it is not unusual to see SpM×V run at under 10% of the peak floating point performance of a single processor. Designing locality-aware data structures and algorithms can be a daunting and time-consuming task, because the best implementation will vary from processor to processor, from compiler to compiler, and from matrix to matrix. This need to have a different data structure for each sparse matrix does not exist in the problem of tuning dense matrix kernels (dense BLAS).

Cache blocking improves locality of accesses to the vectors x and y by dynamically inspecting the matrix data structure and changing it into a sequence of sparse submatrices so that the portions of the vectors x and y for each submatrix fit in the cache. The fundamental tradeoff we need to make is whether the benefits of the added locality outweigh the costs associated with the added accesses to the data structures.

We first present two new techniques for cache blocking. The first technique eliminates the need to iterate over rows that have no non-zero elements. Performance results indicate that this technique creates a finer granularity of blocking in the row dimension. The second technique is a software engineering aid that allows easier incorporation of cache blocking with our prior SpM×V routines as well as enabling recursion for multiple levels of cache blocking.

Next, we develop upper and lower bounds on the execution rate of SpM×V, based on the nonzero pattern in the matrix and the cost of basic memory operations, such as cache hits and misses. The two bounds differ only in their assumption about whether conflict misses occur: in the upper bound any value that has been used before is modeled as a cache hit (no conflict misses), whereas the lower bound assumes that all data must be reloaded. We then use detailed hardware counter data collected on 3 different computing platforms over a test set of 14 sparse matrices to validate our models. We then analyze a set of randomly generated matrices, to get more intuition into why and when cache blocking helps. Finally we analyze the parameters of the matrices that make them the most amenable to cache blocking in the context of these models.

Our findings indicate that matrices that have large column dimensions (over 100,000 columns) and relatively small row dimensions (less than 10,000 rows) benefit the most from cache blocking. The large column dimensions imply that the size of the x vector is too large to fit in the largest level of cache necessitating the need for cache blocking. The small row dimensions imply that the overhead for blocking the matrix is not high. We also find that cache blocking does not help with band matrices since the matrix structure already lends itself to the optimal access pattern. The optimum block sizes predicted by the performance models generally match the measured optimum block sizes and therefore the models can be used as a basis for a heuristic to pick the block size. We conclude with architectural suggestions that would make processor and memory systems more amenable to SpM×V.