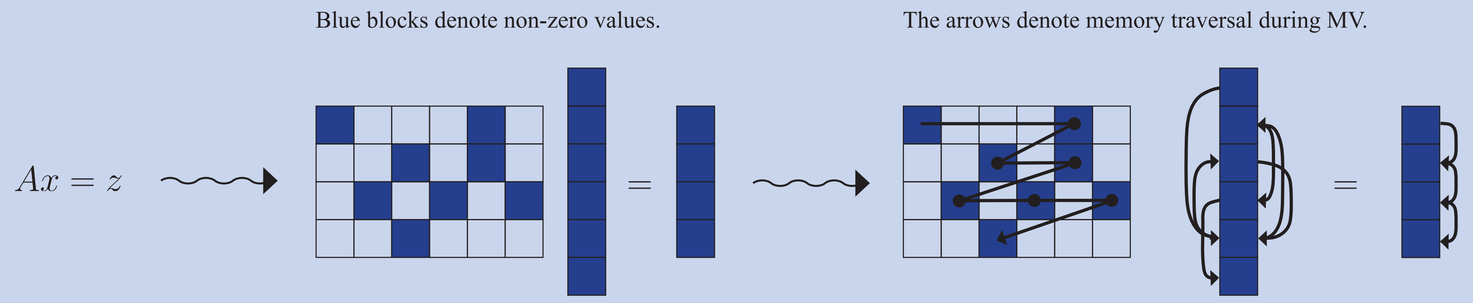
Cache-oblivious sparse matrix-vector multiplication

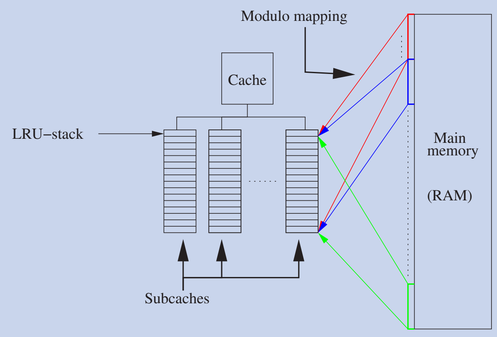
by using sparse matrix partitioning methods

The Problem:

When performing sparse matrix-vector multiplications, irregular data access induces many cache misses, wasting much CPU time. On the right, we illustrate the above with respect to a small sample matrix.

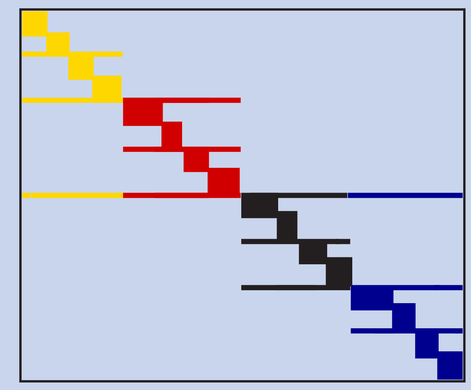


We assume a k-way set-associative, single-level cache structure; see right. The cache can be represented as a matrix containing cache lines. Data with an address from main memory is modulo-mapped in a cache row, while the column is selected by the Least Recently Used (LRU) policy.

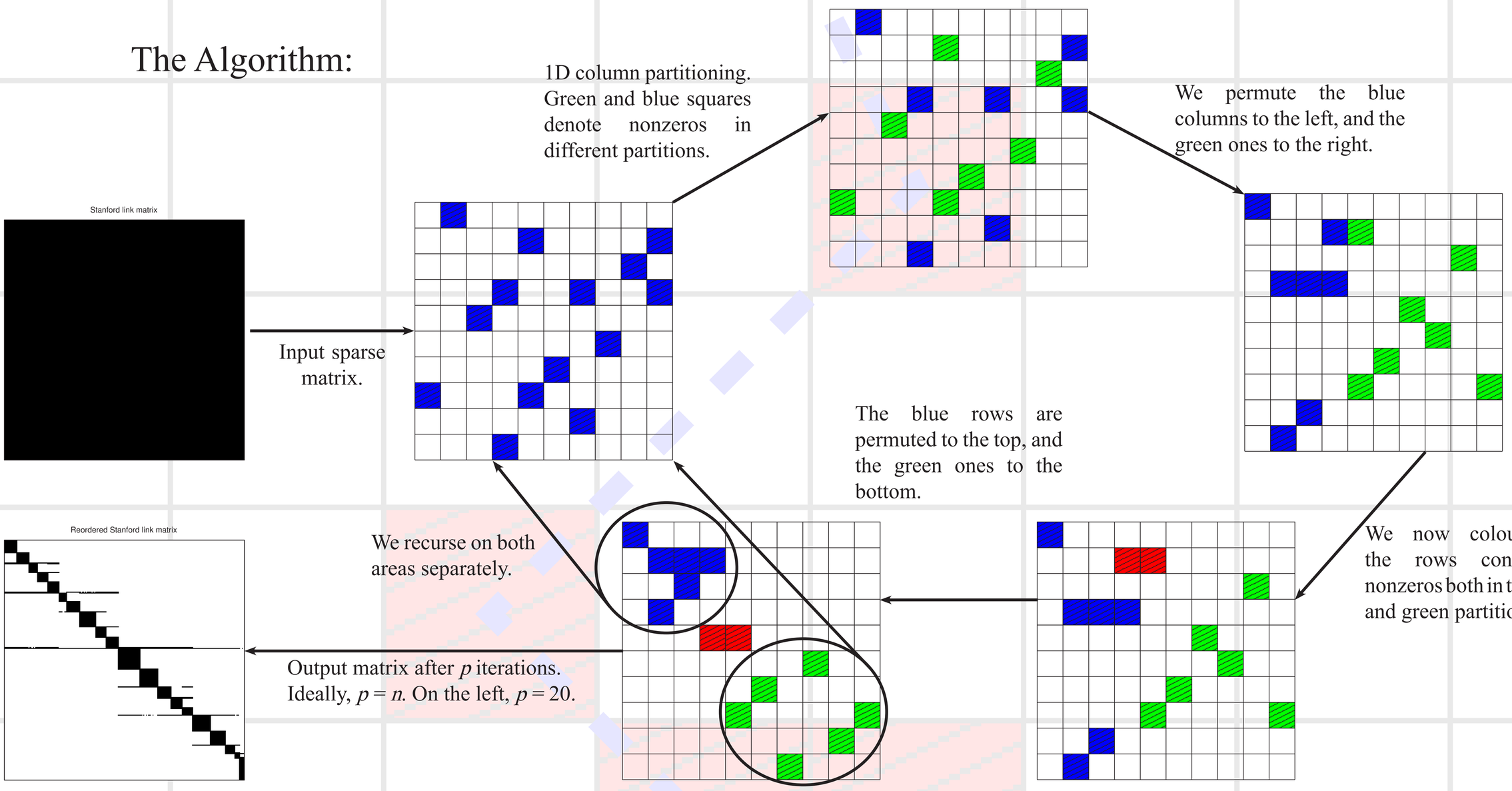


A matrix structure efficient in terms of cache use is the Separated Block Diagonal (SBD) form. This structure ensures some locality in the traversal of the x-vector, when both rows and columns are processed in straight passes, as is the case with Compressed Row Storage (CRS).

So what happens if we combine a common sparse matrix ordering like CRS together with a method permuting the input matrix to SBD form so that cache performance is improved when multiplying the resulting matrix?



The Algorithm:



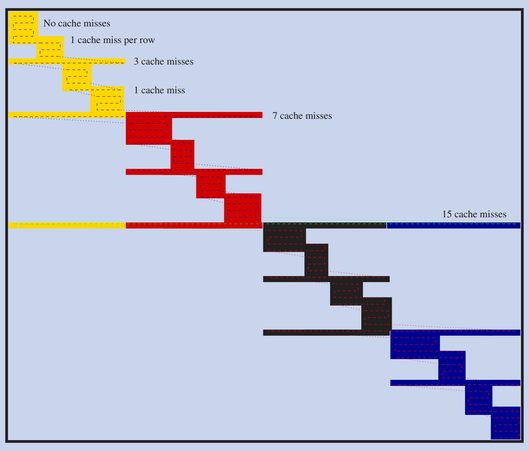
Article:
A.N. Yzelman & R.H. Bisseling, *Cache-oblivious sparse matrix-vector multiplication by using sparse matrix partitioning methods*, pre-print August 2008.

Hypergraph partitioning:

Suppose we traverse matrices in SBD-form by a zig-zag ordering. Denote the cache size by S, the cache line size by L_S, the number of cache lines by $L = \frac{S}{L_S}$, and the number of data words per cache line by w. If we partition the n columns of A into p sets, with

$$p = \frac{n}{wL},$$

and assuming $k \to \infty$, the number of cache misses on each row is as shown for $p = 16$.



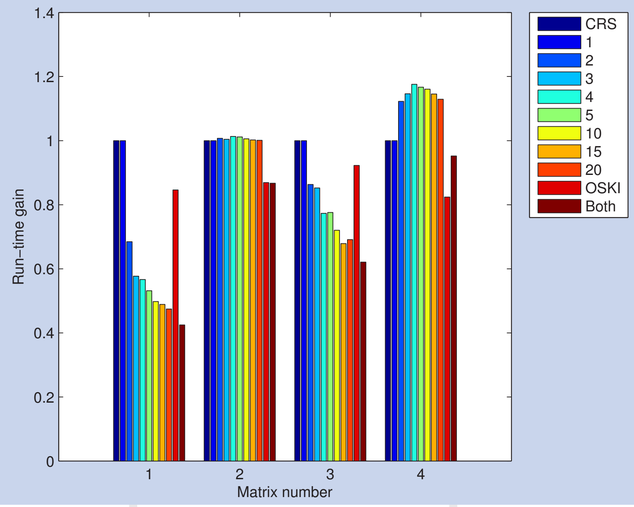
We use a hypergraph-based partitioner. We denote columns using vertices $v_j \in \mathcal{V}$, and rows using *nets* $n_i \in \mathcal{N}$. This is the row-net model. The number of vertex sets over which the net n_i is distributed is denoted by λ_i. Assuming all coloured matrix areas are sufficiently dense, we see that the number of cache misses equals

$$\sum_{i:\, n_i \in \mathcal{N}} \lambda_i - 1,$$

which is exactly the $\lambda - 1$ metric used in sparse matrix partitioning for parallelism.

Experiments:

Experiments were performed using the Stanford, Standford_Berkeley and wikipedia-20051105 link matrices, as well as the cage14 matrix. Below are the run-time gains from executing our method for various p, as opposed to using plain CRS or the Optimised Sparse Kernel Interface (OSKI) developed at Berkeley.



Universiteit Utrecht

Albert-Jan Yzelman & Rob Bisseling,
Dept. of Mathematics, Utrecht University