Cache-oblivious matrix reordering

based on data partitioning for parallelism

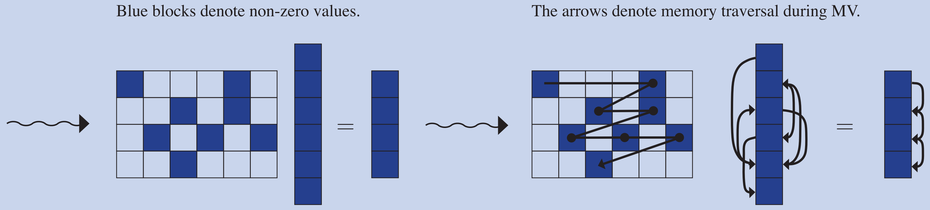
Albert-Jan Yzelman & Rob Bisseling,
Dept. of Mathematics, Utrecht University

The Problem:

When performing sparse matrix-vector multiplications, a large amount of CPU time is wasted on moving data around in memory.

Let us consider sparse matrix-vector (MV) multiplications on matrices stored in Compressed Row Storage (CRS) format.
On the right, we display an example MV run on a small matrix.

$Ax = z$

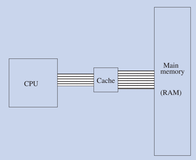


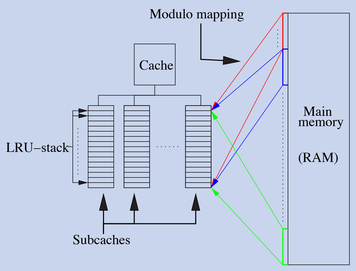
Jumping around in memory takes CPU time. Note that in any efficient MV algorithm, non-zeros from A are accessed precisely once.

Also note that memory access in z is optimal (no jumps), while it is horrible for x.

Preliminaries: the Cache Model

We assume a simple single-level cache structure.
Our cache-oblivious aim will ensure efficiency on multi-level cache architectures.





Assumed is a k-way set-associative cache. This means the cache is subdivided into k subcaches of equal size.
Data from RAM is modulo-mapped in a cache row, while the column is selected by a Least Recently Used (LRU) policy.

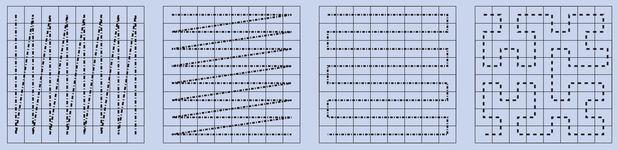
Method of attack: Matrix orderings

Various ways of storing sparse matrices result in different cache hit or miss behaviour.

We investigated the Compressed Row Storage (CRS) format, Incremental CRS (ICRS), and Hilbert-ordering.

Investigation entailed both wall-clock measured experiments, as well as running an MV through a custom run-time cache simulator.

We used the simulator to obtain cache statistics for MV runs on L1 and L2 Intel Core 2 caches.



(I)CCS (I)CRS Zig-zag Hilbert

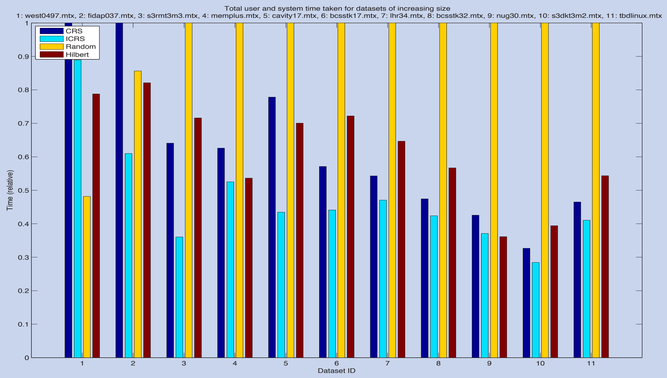
Where we are now: experimental results

Matrix Market input

CRS, ICRS, Randomised order data structures

Matrix Market to Hilbert ordered binary file conversion

Wall-clock timings & cache simulation



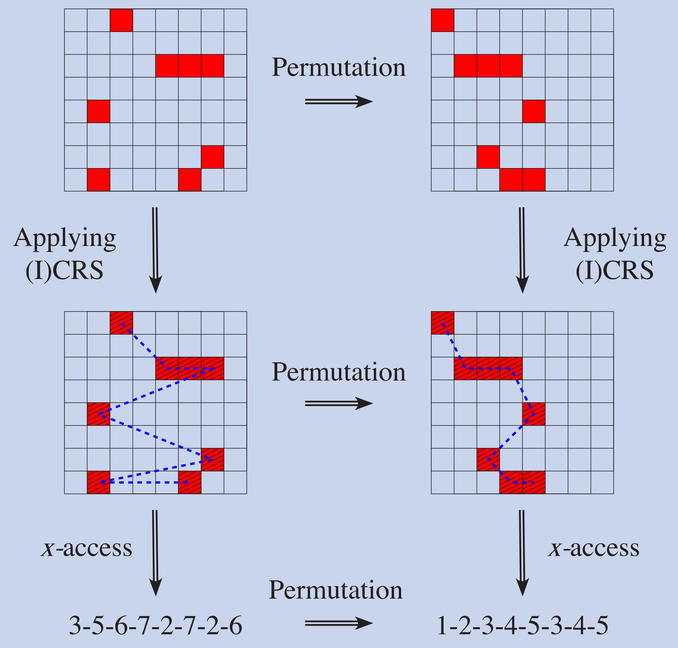
The Triplet scheme requires *3nnz* storage for any n times m sparse matrix A consisting of *nnz* non-zeros. (I)CRS however, requires only *2nnz+m*, worst case.
Combining this with the observation that elements from A need only be accessed once, the Triplet scheme thus defaults to an obligatory number of *nnz-m* extra cache misses when compared to (I)CRS.

Any Triplet-based ordering scheme will hence have a hard time competing with a CRS-based scheme. Note that both the Random and Hilbert orderings are stored in Triplet format.

Where we are going: Matrix permutations

What if we combine a cache-friendly matrix ordering together with a matrix column / row permutation algorithm to transform the matrix structure such that cache performance is improved even more?

What are the similarities to matrix partitioning regarding parallel load balancing? Instead of reducing communication, we here want to increase data locality.



Universiteit Utrecht