I/O-Optimizing compilers for Sparse Matrix Vector Product

Riko Jacob (rjacob@inf.ethz.ch), Tobias Lieber (lieberto@inf.ethz.ch)

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

1) Motivation

Consider evaluating $x^T Ay := \sum_{1\le i,j\le n} x_i a_{ij} y_j$ for a given sparse matrix $A \in \{0,1\}^{s\times t}$ in the semiring-I/O-Model [1] with $B = 1$.

By [3]: $\#I/O[x^T Ay] = \Theta(\#I/O[Ay])$

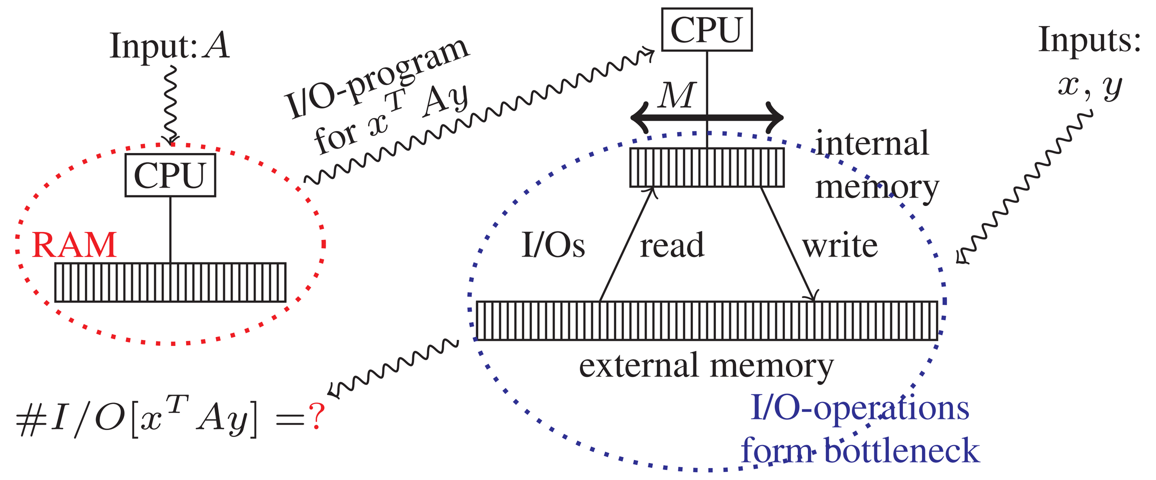
The complexity of h-SpMxV is known [1]:

$$\#I/O[Ay] = \Theta\left(\min\left\{\frac{hN}{B}\log_{\frac{M}{B}}\frac{N}{hM}, hN\right\}\right)$$

For star stencil computations A_s on a $k_1 \times k_2$-grid [4]:

$$\#I/O[A_s y] = 2k_1k_2 + 4\frac{k_1k_2}{M-4} + \mathcal{O}(k_2)$$

As SpMxV is notoriously memory bound and a key component of many numerical applications we consider the complexity of writing I/O-optimal programs for A:



Here we ignore I/Os due to matrix values (a_{ij}).

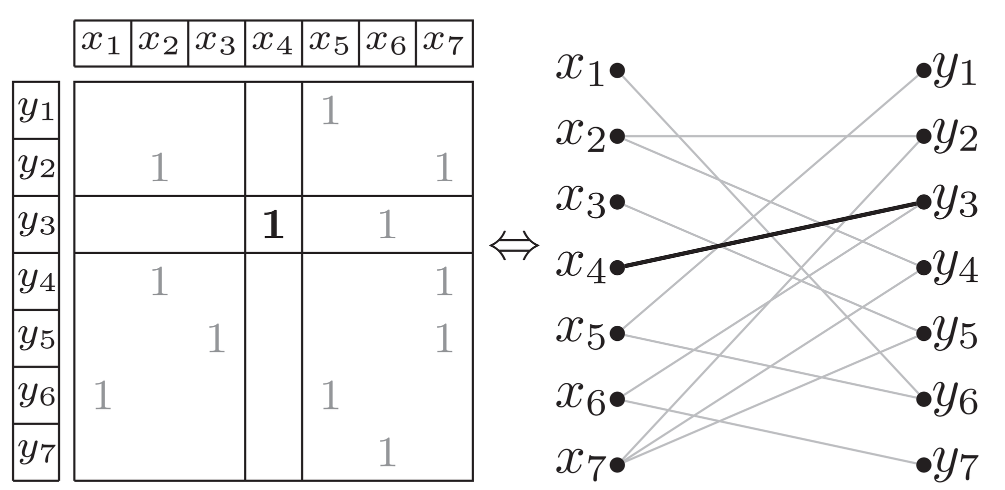
2) Problem Definition

We consider two problems:

P1. Is there a program that evaluates $x^T Ay$ with at most ℓ I/Os and memory M?

P2. Can $x^T Ay$ be evaluated with ℓ I/Os on an I/O machine with memory M?

To answer these questions we view the matrix A as the adjacency matrix of a bipartite graph $G(A)$.



We count the number k of non-compulsory I/Os. Meaning the I/Os needed besides scanning x and y.

3) Results

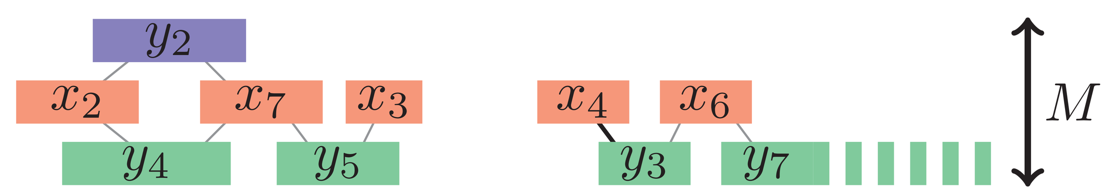
We show:

- The memory needed for computing $x^T Ay$ in the I/O-Model corresponds to the pathwidth [2] of $G(A)$ (Box 4). Therefore P1 is NP-Complete.
- Problem P2 is NP-Complete (Box 6).
- There is an easy 2-approximation algorithm for the special case, $M = 2$, of P2 (Box 8).

4) Connection to Pathwidth

Lemma 1 *The bilinearform $x^T Ay$ can be evaluated with memory M and without non-compulsory I/Os iff $G(A)$ has pathwidth $M - 1$.*

Proof For each vector-record, there is only one time-interval where it is in memory.



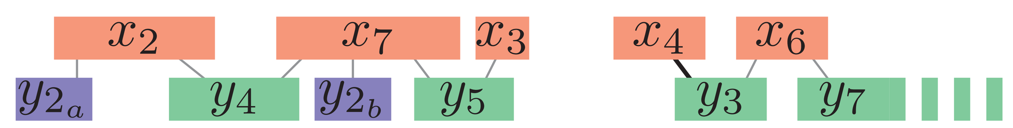
Title: Parallel Main-Memory Indexing for Moving-Object Query and Update Workloads

The interval thickness equals the pathwidth of $G(A)$. □

The result above can be generalized to arbitrary B and k.

There are FPT-algorithms to check if $x^T Ay$ can be evaluated for constant M and k. They exploit the following corollary:

Corollary 2 *Splitting a node and partitioning its edges corresponds to loading a vector-entry twice.*



References

[1] Michael A. Bender, Gerth Stølting Brodal, Rolf Fagerberg, Riko Jacob, and Elias Vicari. Optimal sparse matrix dense vector multiplication in the I/O-model. In *Proceedings of SPAA '07*, pages 61–70, 2007.

[2] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[3] Gero Greiner and Riko Jacob. Evaluating non-square sparse bilinear forms on multiple vector pairs in the I/O-model. In *Proceedings of MFCS '10*, pages 393–404, 2010.

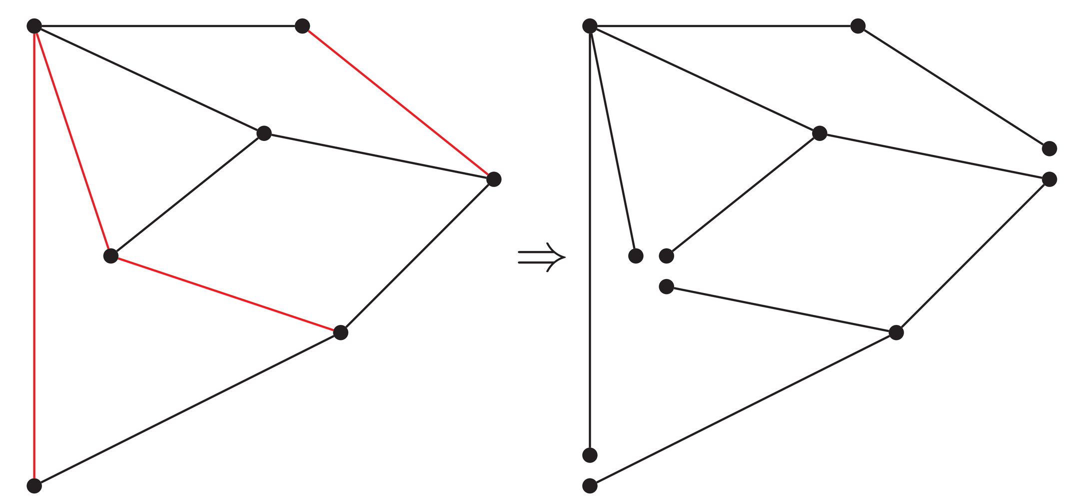
[4] Philipp Hupp and Riko Jacob. Tight bounds for low dimensional star stencils in the external memory model. *CoRR*, abs/1205.0606, 2012.

5) Splitting a Graph to a Tree

An equivalent definition of problem P2 is: Can $G(A)$ be transformed by k splits into a graph G' of pathwidth $M - 1$.

The variant, how many splits are needed to obtain from $G(A)$ a tree, yields an important structural insight.

The problem **Split to Tree** is exactly solvable by splitting all edges off, which are not in a spanning tree.



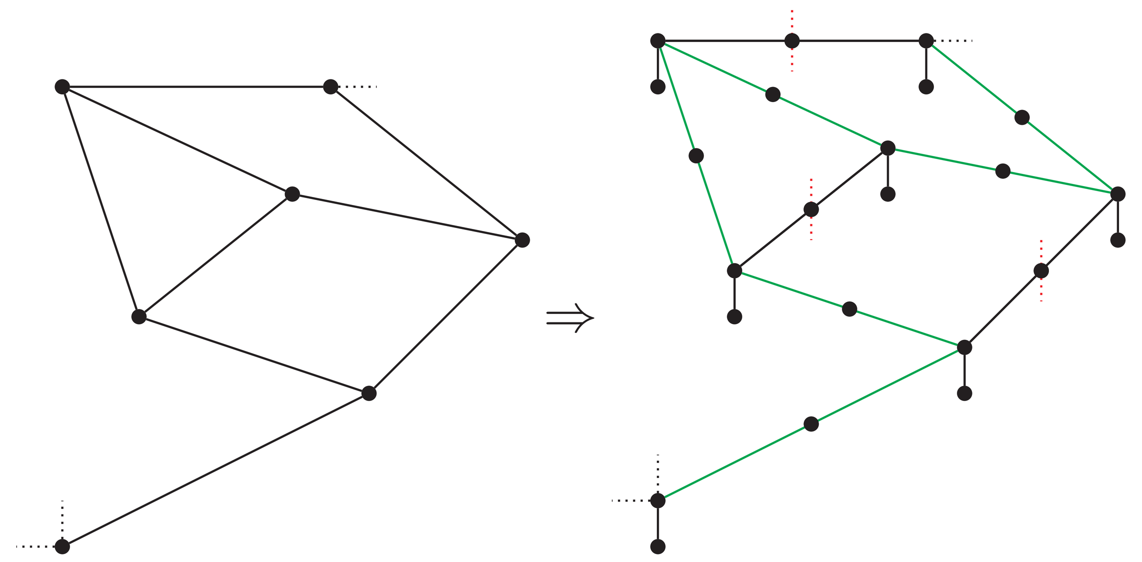
Lemma 3 *This split sequence is* ***optimal***.

Proof Each split increases the number of nodes by one, but does not change the size of the edge set. □

6) NP-Completeness

The problem Cubic Planar Hamiltonian Path, which is NP-complete, can be reduced to Split to Caterpillar, which is equal to Split to Pathwidth 1.

Transformation:



Lemma 4 *The transformed graph has pathwidth 1 after $m - n + 1$ node splits iff G has an Hamiltonian path.*

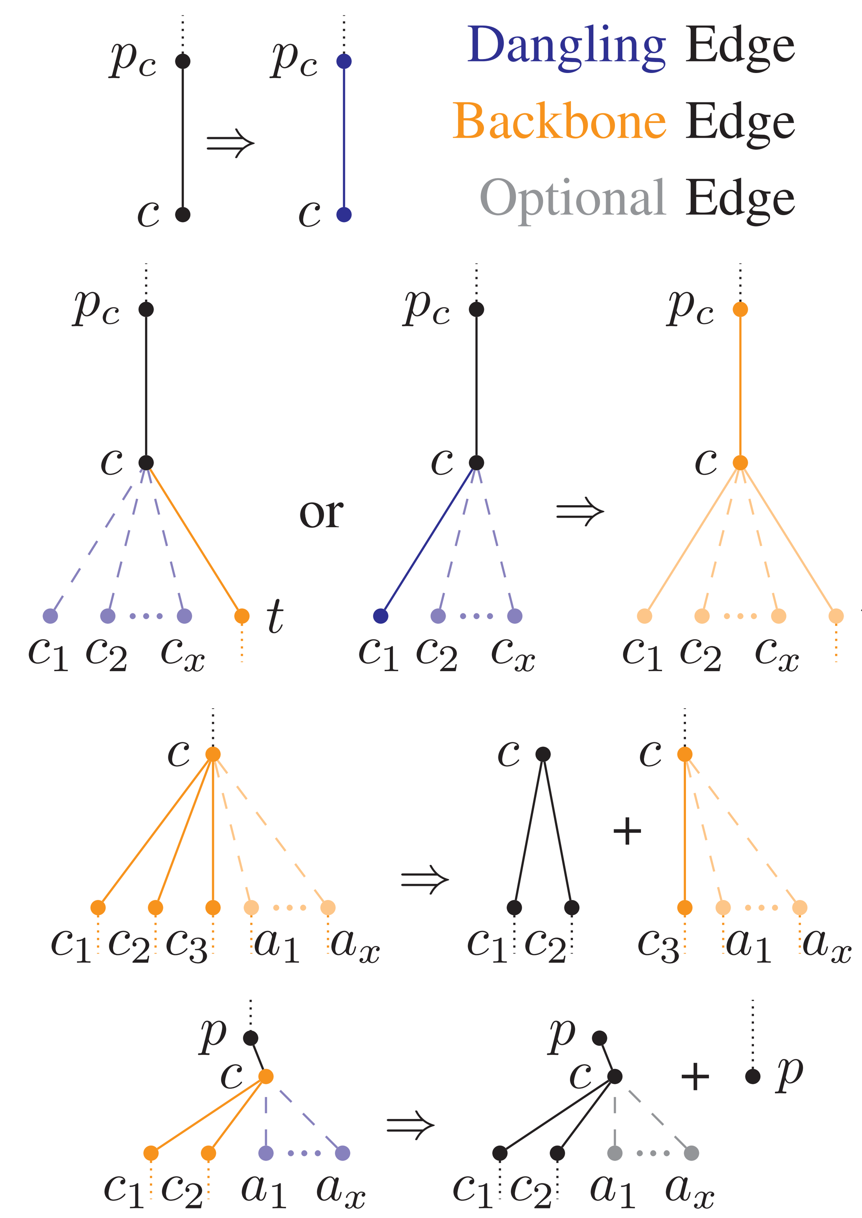
Proof ⇐ Split nodes as implied by transformation.
⇒ The dangling edges imply a linear order on the original vertex nodes, since their incident nodes have to be in a bag. □

This result can be generalized to arbitrary $M > 2$.

7) Optimal Splitting of Tree to CP

Lemma 5 *There is an efficient algorithm, which computes a minimal splitting sequence, which turns a tree T into a caterpillar.*

Root T arbitrarily to apply *bottom up* the following *greedy* coloring- and splitting-rules at each node c.



8) Approximation

Lemma 6 *After splitting a graph G into a tree (see Box 4), applying the optimal split-to-caterpillar sequence of G results in a collection of caterpillars.*

Theorem 7 *Splitting a connected graph G into a tree T with t splits and splitting T into a collection of caterpillars with c splits is a 2-approximation algorithm for Split-to-Caterpillar.*

Proof Lemma 3 yields: $t \le opt$. Since there is an *optimal* algorithm to split a tree T into a collection of caterpillars and applying an optimal split-sequence of G to T yields a collection of caterpillars, too: $c \le opt$. Thus, $t + c \le 2 \cdot opt$. □

9) Open Problems

- Approximation bounds for Split to Pathwidth M.
- Improved Bounds for Split to Caterpillar.
- Generalization to $B > 1$.