On Optimization of Distributed Array Partitioning Schemes

Avah Banerjee, Guoli Ding

Louisiana State University

ibanerjee@lsu.edu

November 7, 2019

A Matrix Tiling Problem

- Consider a sequence of matrix operations involving a set of matrices.
- We assume these matrices are large and the operations are performed in a distributed fashion.
- Each matrix is partitioned into a collection of sub-matrices, called tiles.

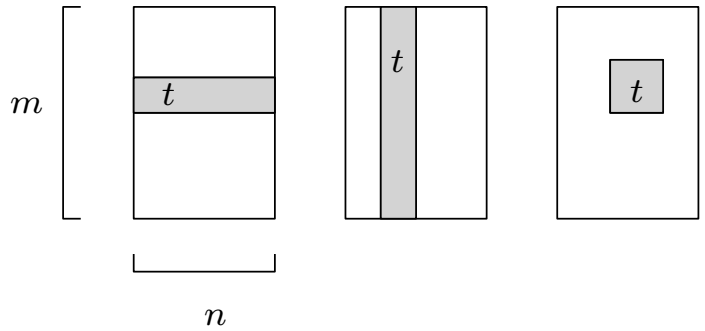

Figure: A tile is highlighted as the shaded rectangular region. Three types of tilings. From left to right order: row (r), column (c) and block (b).

A Matrix Tiling Problem

- Size of these tiles are uniform (dense case) and only depends on the size of the matrix, number of tiles and the tiling type.
- A fixed set of matrix operations are considered.
- $f(A, B; C, D)$ is an operation with inputs A, B and outputs C, D. Arity of the operation is 4.
- Let t_A be the tiling for A.
- Let f_A be a fixed set of algorithm implementing f.
- The tiling cost of an operation is a function of the tilings of the operands and the algorithm implementing f.
- Tiling cost $C_f(operands, algo)$ (should be determined experimentally)
- Each matrix is write-once.

A Matrix Tiling Problem

- These operations may use the same matrices or the output of one is could the input of another.
- Let P be a user program.
- Dependency of these operations create a DAG G_P (which may not be fully known before runtime).
- Annotations can help.
- Now consider a hypergraph H_P whose vertices are matrices and whose edges are the operations.
- Coloring a vertex is equivalent to choosing a tiling for it.
- Coloring an edge is equivalent to choosing an algorithm for it.

A Matrix Tiling Problem

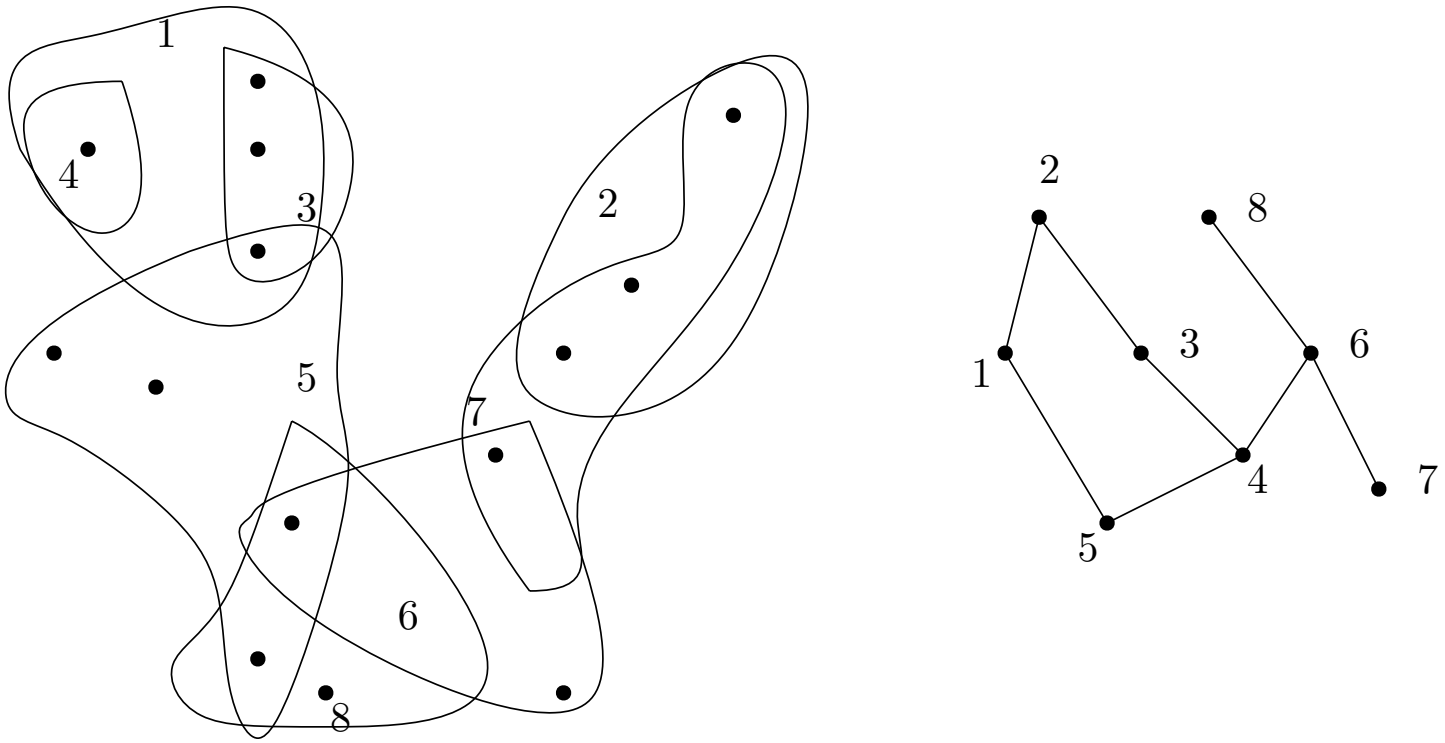

Figure: The hypergraph H_P corresponding to P (left). Partial order of the edges according to G_p (right).

A Matrix Tiling Problem

- If we do not put any constraint on memory then we may as well keep copies of different tiles.
- Let M be the total number of additional tiles we can keep per processor for all matrices.
- Keeping multiple copies of matrices is equivalent to assigning multiple colors to the corresponding vertex.
- Cost of a coloring is the sum of the cost of the edges (C_f)
- A coloring is *good* if (total colors used - $n < M$).
- A coloring is *great* if it also minimizes the cost.
- A weight on the edges may be used to indicate the complexity of the operation.

An Algorithm for MTP

- If $|H_P|$ is small then solve H_P using brute force.
- Else partition H_P to number of sub-hypergraphs $H_1, \ldots, H_k$ so that weight of the total cut edges are as small as possible.
- H_i is small then solve using brute force , else recurse.
- Merge the solutions.

A Vague Algorithm for MTP

- This is an approximation-hard problem in general.
- Our hypergraph may have special properties.
- For example the edges are partially ordered and there is a correlation between cut-edges between two partitions and this partial order.
- H_P is likely to be small.
- Edge cardinalities and vertex degrees are likely to be bounded as well.
- If memory is less of an issue many good colorings are also great.
- More detail can be found in the writeup (we use IP to describe the problem there).

Looking Forward

- Determination of H_P given a user program:
  1. If we have conditionals then not all parts of the H_P will be known.
  2. May be approximated by running the program few times.
- Experimentally solving MTP on H_P:
  1. There are experimental algorithms to solve different hypergraph partitioning problems.
  2. We should avoid overengineering.

What we been up to (Avah and Guoli)

- Online Mincut and other sub-modular functions: Advice complexity, Benevolent adversary , Average case complexity (A paper is forthcoming)
- The k-server problem on trees and circles.

Even simple tiling is hard

Binary Tiling Problem

Our universe of operations: $(=, +, ()^T, \times)$

- Input: 1) A set of n variables V (matrices)
  2) An ordered sequence of binary trees $(T_1, \ldots, T_m)$ with four types of expressions: $\{V, = +, = \times, =^T\}$
- Leaves and the root of each tree are always an elements from V.
- $V(T_i) \subset V$ are the variables involved in the computation of T_i.
- $r(T_i)$ is the root of T_i. For $i > j$, $r(T_j) \notin V(T_i)$. (P1)
- Two possible (uniform) tilings : row-wise, column-wise (r, c)

Tiling Constraints

- For each expression we create a constraint.
- t_a = tiling of the matrix A
- 1) $A = B + C, A = B : (t_a = t_b = t_c), (t_a = t_b)$ (resp.)
  2) $A = B^T : (t_a \neq t_b)$
  3) $A = BC :$
  $(t_a = r, t_b = r) \vee (t_a = c, t_c = c) \vee (t_b = c, t_c = r)$

Tiling Constraints

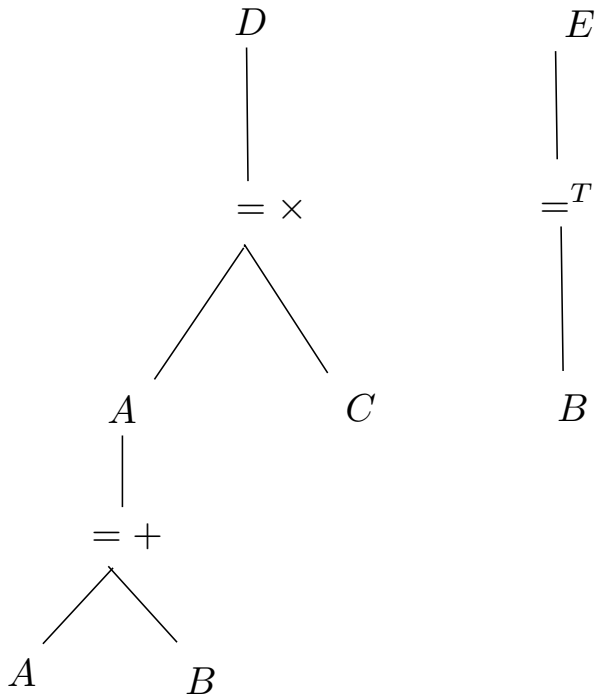

Figure: Constraints: $\Phi = (t_a = t_b) \wedge ((t_d = r, t_a = r) \vee (t_d = c, t_c = c) \vee (t_a = c, t_c = r)) \wedge (t_b \neq t_e)$

A Minimization Problem

- Suppose we are allowed keep both types of tiling for a matrix.
- It can be shown that we can avoid (extra) communication in leu of a constant factor slowdown in total computation time.
- Keeping two copies of a matrix is **equivalent** to deleting the corresponding variable from Φ.
- Problem (BTP1): Minimize the number of variables to delete such that Φ is satisfiable.
- Problem (BTP2): Minimize the number of clauses to delete such that Φ is satisfiable.
- Let this value be $\nu(BTP_i)$ $(i = 1, 2)$

Special Cases

- Is determining $\nu(BTP_i)(=, =^T, = +)$ is NP-hard. (SPARTAN paper)
- Determining $\nu(BTP_i)(=, =^T, = +)$ within a $\Omega(\log n)$ ($n =$ number of matrices) of the optimal is also hard (unless UGC is false). (Our observation)