Data Distribution and Performance Optimization Models for Parallel Data Mining

Eray Özkural

Frequent Itemset Mining (FIM)

- Frequent Itemset Mining
  - A transaction is a set $X \subseteq I$ of items (an itemset)
  - A transaction DB is a list of transactions
  - Itemset $Y \subseteq X \rightarrow Y$ occurs in X
  - Frequent itemsets Y:
    - occur in DB F(Y) times
    - $F(Y) \geqslant \epsilon$ (support threshold)
    - if itemset Y is frequent, then any subset of it is, also (downward closure)
    - Mining: iterate frequent itemset length
- Example:
  - I = {a,b,c,d,e,f,g,h,i}
  - Transaction DB on the right
  - Frequent itemsets with $\epsilon=3$ are:
    - {}, {b}, {e}, {b}, {d ,{6}, {a}, {h , {f}, {d}, {a}, {e,a}, {g,a}, {b,c}, {b,e}, {b,e}, {b,f}, {c,f}, {c,g}, {e,d}, {d,h}, {e,g}, {b,c,f}

a	b	c	d	e	f	g	h	i
	x	x			x	x		
		x				x		
	x						x	x
	x			x				
x			x			x	x	
			x	x				
	x	x		x	x			
x	x	x			x			
	x	x	x				x	x
	x	x		x		x		
x			x	x		x		
			x				x	
x			x	x			x	
	x			x				
x				x		x		

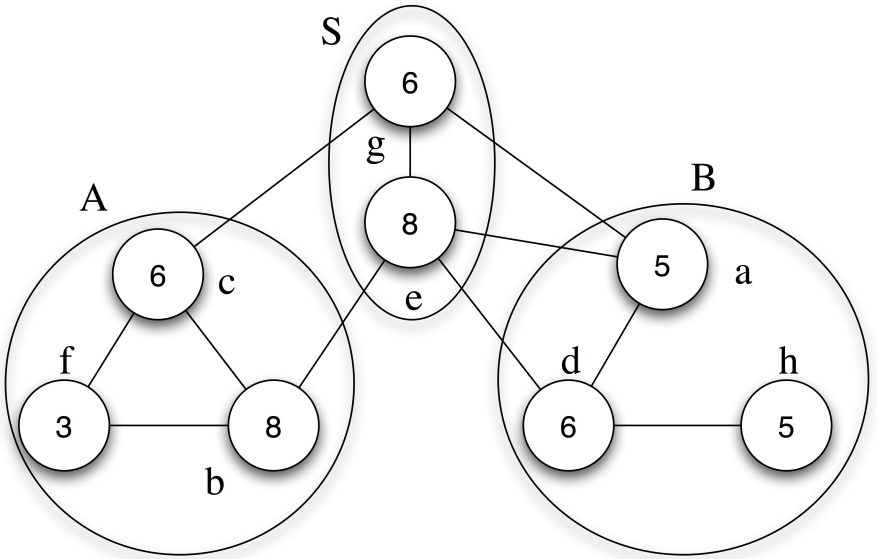
Parallel FIM Algorithms

- Some existing efficient parallelizations
  - Count-Distribution (Agrawal and Shafer, 1996)
    - Partition data, candidate counts
    - Replicate candidate set
  - Data-Distribution (Agrawal and Shafer, 1996)
    - Partition data, candidate set
    - Too much communication
  - IDD and HD (Han et. Al, 2000)
    - Partition set of candidates and data
    - Intelligent Data Dist.: 1-D, partition hash-tree
    - Hybrid Data Dist.: 2-D, level 1: count-distribution, level 2: IDD
    - Tested on Cray T3D
  - ParDCI (Orlando et. al, 2002)
    - Replicate entire database
    - Distribute candidates by partitioning prefix-tree

NoClique2

- Parallel algorithm:
  - Partition data item-wise
  - Finds sets of items that can be independently mined
  - Mine with ParDCI like algorithm until level k
  - Graph G_{F_k} of frequent itemsets at level k:
    - Items → Vertices
    - Edges → frequent itemsets of length 2
  - Graph Partitioning by Vertex Separator (GPVS) of G_{F_k} model:
    - Number of processors → n
    - Find a separator S set of items that are replicated and n disjoint parts V_i
    - Mine S in parallel using algorithm similar to ParDCI
    - Other parts V_i can be mined independently concurrently
    - Frequent itemsets in S and V_i have to be merged, independently
    - Induces a task distribution

Example GPVS



b	c	e	f	g		a	d	e	g	h
x	x		x	x					x	
	x			x					x	
x										x
x		x						x		
				x		x	x		x	x
		x					x	x		
x	x	x	x					x		
x	x		x			x				
x	x						x			x
x	x	x		x				x	x	
		x		x		x	x	x	x	
							x			x
		x				x	x	x		x
x		x					x			
		x		x		x	x			

NoClique2 Experiments

- Compared three implementations:
  - NoClique2 (NC2): our algorithm
  - Repl-Bitdrill (RBD): our adaptation of ParDCI
  - ParDCI (PDCI): Orlano et. al's algorithm
- One synthetic, and three real-world DB's
- 500K – 2M transactions
- 6000 – 5.5M items
- 325K – 2.1M frequent itemsets output
- Cluster: 32 processors, distributed memory

Speedup Results

DB	NC2	RBD	PDCI	NC2	RBD	PDCI
	4 procs			8 procs		
T60.I10.2000K	3.4	2.3	1.2	4.3	4.0	2.3
user-likesmovies	2.9	2.9	2.8	5.0	5.0	7.4
trec	2.6	2.7	–	4.2	4.5	–
trec.lp.200000	2.8	2.8	0.7	4.9	4.9	1.5

DB	NC2	RBD	PDCI	NC2	RBD	PDCI
	16 procs			32 procs		
T60.I10.2000K	7.7	6.2	4.4	11.1	2.6	8.7
user-likesmovies	7.5	7.9	11.4	10.7	10.6	8.2
trec	6.3	5.1	–	8.7	7.1	–
trec.lp.200000	7.9	1.6	2.8	13.0	1.7	5.2

NoClique2 Summary

- New FIM parallelization
- Applies to synthetic and real-world data
- Especially good for
  - Sparse datasets
  - DB's with large # of items
- Compares favorably to Repl-Bitdrill & ParDCI:
  - Faster than both for 32 processors
  - On user-likesmovies, Repl-Bitdrill almost as fast

Intelligent Candidate Distribution With Selective Item Replication

- Alternative FIM parallelization
  - Addresses high load imbalance of NoClique2
  - Eschews independent mining condition
- Hypergraph models of task-data dependency
  - Vertices → Candidates (Tasks)
  - Hyperedges → Items (Data)
- Hypergraph partitioning model:
  - Objective → minimize communication volume (weight = item freq.)
  - Constraint → balance load (vertex weight = min. freq of items)
- Parallel overhead
  - Spurious candidates for I>0
  - Multiple hypergraph partitionings
  - DB re-distributions

Hypergraph Partitioning Model for Frequent Itemset Mining

Intelligent Candidate and Item Distribution (ICID) Algorithm

- Similar to NoClique2 algorithm
- Phases of parallel FIM computation:
  - Mine in parallel up to level k
  - Generate candidates up to level k+l
  - Partition hypergraph model
  - Re-distribute transaction database
  - Mine independently on each processor:
    - Distribute candidates assigned to each processor
    - Each processor tests freq of assigned candidates

Re-partitioning Model

Benefits of Re-partitioning Model

- Candidates of a few levels can be generated
- Multiple iterations for full mining
- Re-partitioning model:
  - Fixed processor vertices (0 weight)
  - Previous item distribution/replication via hyperedges
- Min. re-distribution given previous dist.
- Incremental ICID
  - HG Partitioning & re-distribution phases are modified
  - Minimizes parallel overhead of DB re-distribution

All Pairs Similarity Problem

- V = set of n sparse input vectors in R^m
- t = similarity threshold
- sim(x,y) = x . y
- Find pairs x,y where sim(x,y)≥t
- In practice $O(n^2)$ time, more precisely
- Time = $O(\sum_{d=1}^{m} \binom{|I_d|}{2}) = O(\sum_{d=1}^{m} |I_d|^2)$
- We use the notation in Bayardo et al

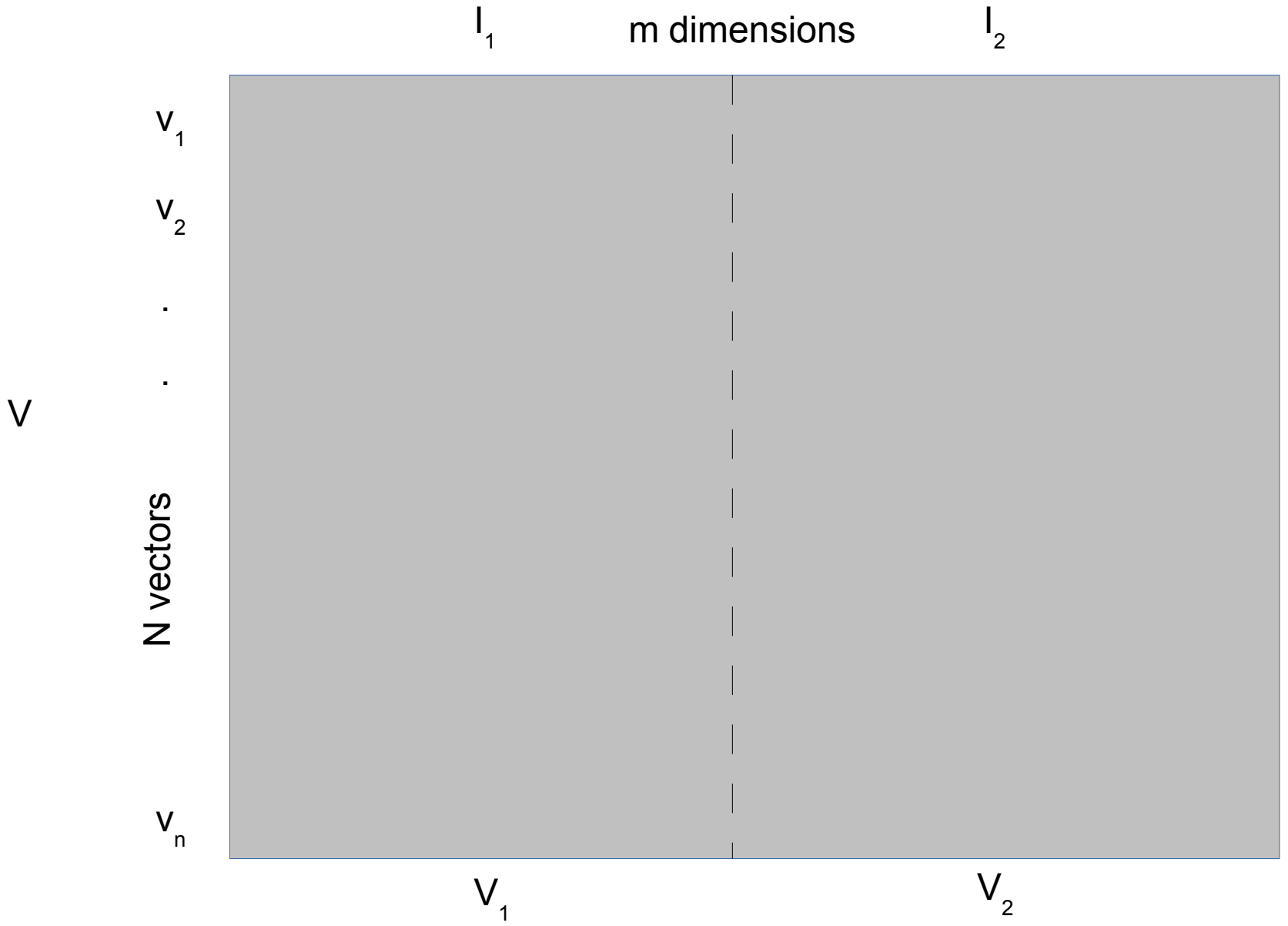
Fast All-Pairs Algorithm

- All-Pairs-0(V,t):
  - M ← Empty-Set
  - I ← Make-Sparse-Matrix(m,n)
  - For v_i in V do
    - M ← M U Find-Matches-0(v_i,I,t)
    - For all vi[h] where $v_i[j]>0$ do
      - I_{ji} ← $v_i[j]$
  - Return M
- Find-Matches-0(x,I,t)
  - A ← Make-Array[m]
  - For (I,x[i]) in x where x[i]≠0 do
    - For (y,y[i]) in I do
      - A[y] ← A[y] + x[i].y[i]
  - Return {(y,A[y]) | A(y)≥t}

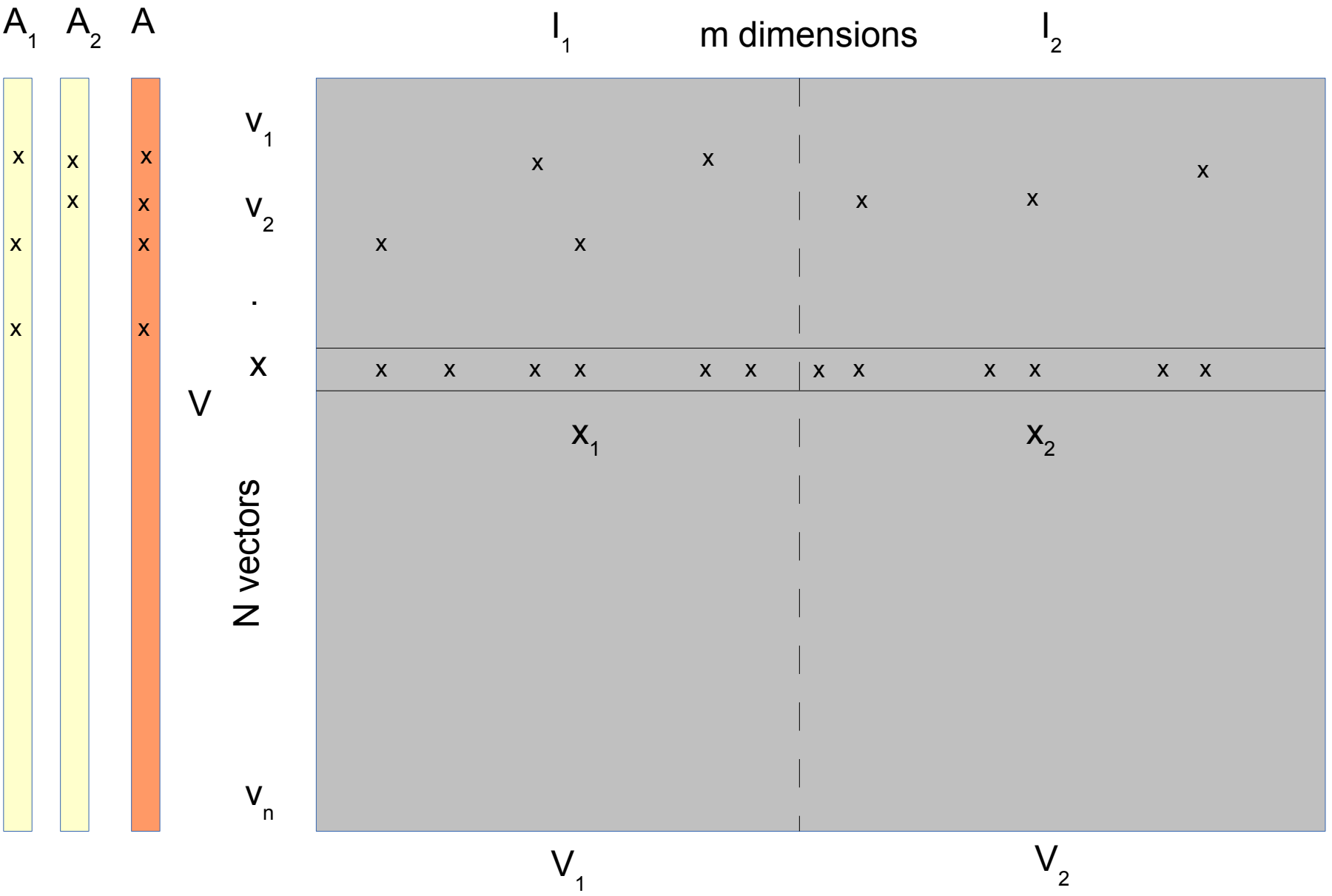
1-D and 2-D Parallel Algorithms for All-Pairs Similarity Problem

- We use a fast sequential algorithm
  - All-pairs-0 from Bayardo et al with array maps
- We distribute data along two dimensions:
  - Vertical (dimension-wise): distribute dimensions
  - Horizontal (vector-wise): distribute vectors
- We also distribute data along both dimensions

Dimension-Wise (Vertical) Parallel All-Pairs Similarity Algorithm



Vertical Partitioning



Vertical Algorithm

- We partition dimensions
  - Root proc: bin dimensions according to load estimate
    - Quadratic in number of non-zeroes in a dimension
    - Simple first-fit greedy binning algorithm
    - $d = \{1, 2, \ldots m\}, \Pi_d = \{d_1, d_2, \ldots, d_p\}$
  - Distribute dimensions to processors with OAPC
  - Timing stars afterwards
- We partition data, inverted index
  - V is partitioned into projections $V_1, V_2, \ldots, V_p$
  - Inverted index I is partitioned, $I_1, I_2, \ldots, I_p$aligned with V
  - Finding matches of x: x is partitioned, score map A replicated

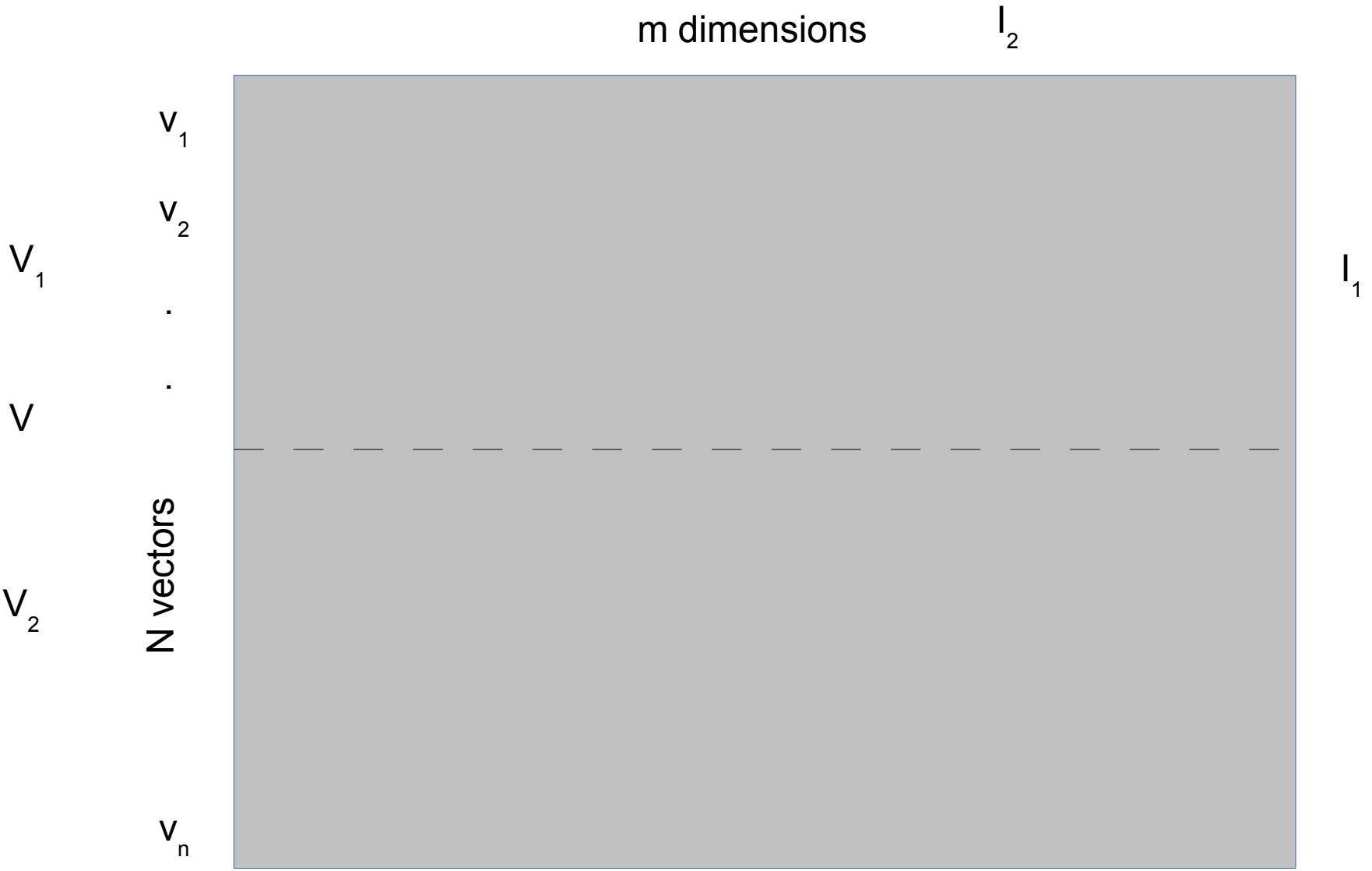
Vertical algorithm (cont.)

- Local pruning:
  - $t_{local} = t/p$
  - For vec x, if $sim(x, y) \geq t$, for any y
    - There is a proc. i, where $sim(\pi_{d_i}(x), \pi_{d_i}(y)) \geq t_{local}$
- Score accumulation with local pruning:
  - Global candidates aggregate local matches:
    - $C = \bigcup_{1 \leq i \leq p} \{y \mid sim(\pi_{d_i}(x), \pi_{d_i}(y)) \geq t_{local}\}$
  - Each processor prunes local scores in score maps
    - $\forall y \in C, A'_i[y] = A_i[y] = w > t_{local}$
    - $A_g = \sum_{i=1}^{p} A'_i$, calculated with collective reduction op.

Vertical Algorithm (cont.)

- Recursive local pruning:
  - Local pruning works best with few processors
  - Local pruning can be recursively applied
    - Can improve pruning, with some overhead
- Block processing:
  - There is fine-grained imbalance in vertical algo.
  - If we process vectors in blocks of 64:
    - Synchronization becomes cheaper
    - Less latency is incurred

Vector-Wise (Horizontal) Parallel All-Pairs Similarity Algorithm



Horizontal Algorithm (cont.)

- Outer loop parallelization
  - Each vector has a home processor
  - At each iteration:
    - Local query vec x is broadcast to other processors
    - The queries (Find-Matches) are executed locally
      - In the same order on all processors
- We also parallelize other variants of all-pairs
  - It turns out that all-pairs-0 is the best in experiments

2-D Parallel Algorithm for All-Pairs

- Simple algorithmic approach
  - Elegant combination of two 1-D parallelizations
  - Use outer loop parallelization:
    - broadcast query vectors
  - Use inner loop parallelization:
    - Call find-matches subroutine of vertical parallelization
  - Use load balancing of vertical algo.

2-D Parallel Algorithm (cont.)

References

R.Agrawal and J.C.Shafer, “Parallel mining of association rules”, *IEEE Trans. On Knowledge And Data Engineering*, vol. 8, pp. 962–969, 1996.

E.-H. Han, G. Karypis, and V. Kumar, “Scalable parallel data mining for association rules”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 337–352, May 2000.

S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, “Adaptive and resource-aware mining of frequent sets”, in *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9–12 December 2002, Maebashi City, Japan*. IEEE Computer Society, 2002, pp. 338–345.