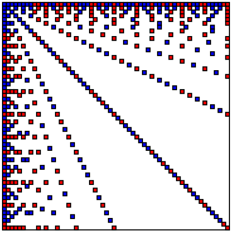
New developments in sparse matrix partitioning for parallel computations



Rob H. Bisseling
Mathematical Institute, Utrecht University
Daan M. Pelt
Centrum voor Wiskunde en Informatica, Amsterdam

Daan Pelt

Parallel Sparse Matrix-Vector Multiplication

Sparse matrices

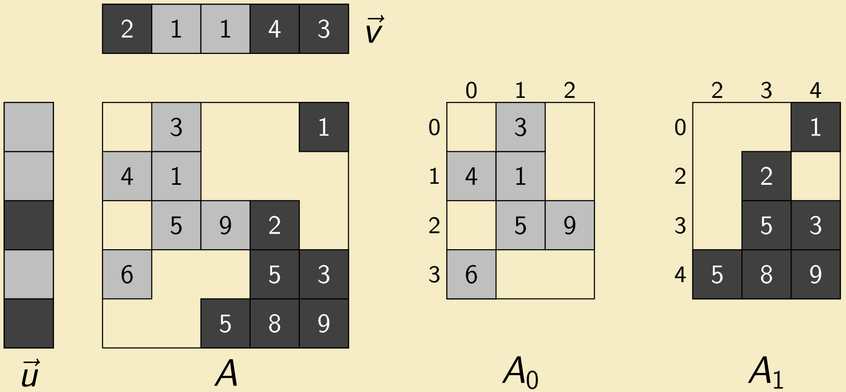
An $m \times n$ matrix with N nonzeroes is called sparse if a large number of its elements is equal to zero ($N \ll mn$).

Matrix-vector multiplication

- Dense matrix-vector multiplication: $O(mn)$
- Sparse matrix-vector multiplication: $O(N)$
- Parallel sparse matrix-vector multiplication: ideally, $O(\frac{N}{p})$

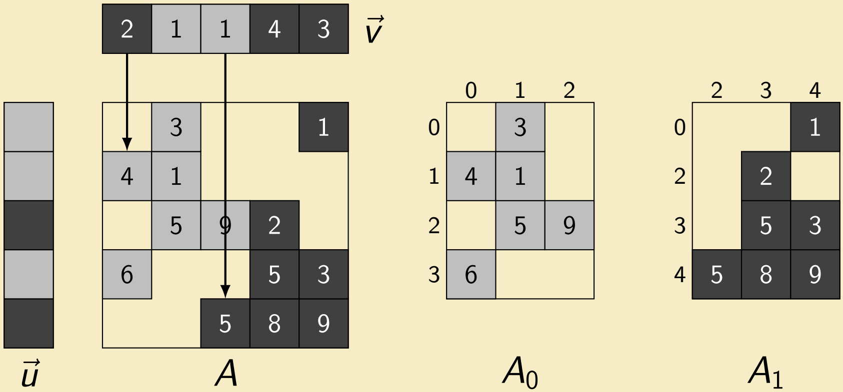
Parallel Sparse Matrix-Vector Multiplication

Parallel Algorithm: Matrix and vector distribution



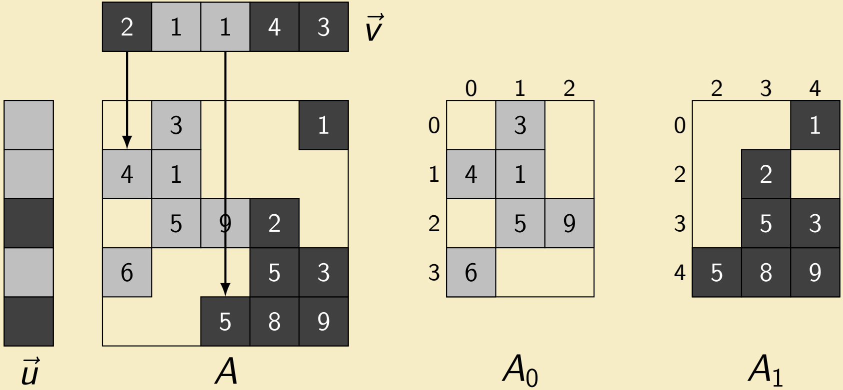
Parallel Sparse Matrix-Vector Multiplication

Parallel Algorithm: Fanout



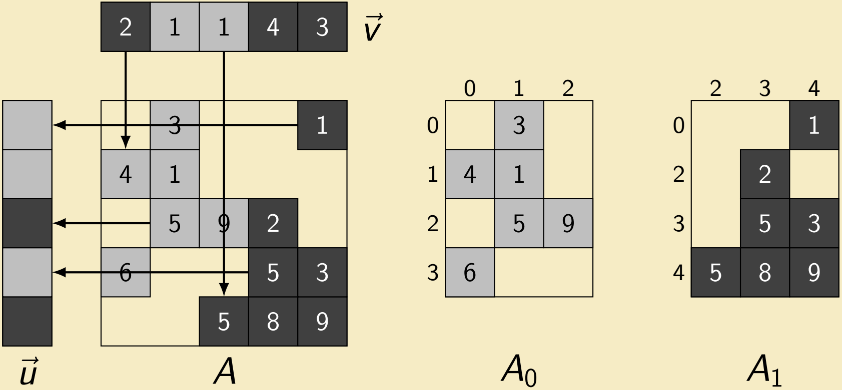
Parallel Sparse Matrix-Vector Multiplication

Parallel Algorithm: Local Multiplication



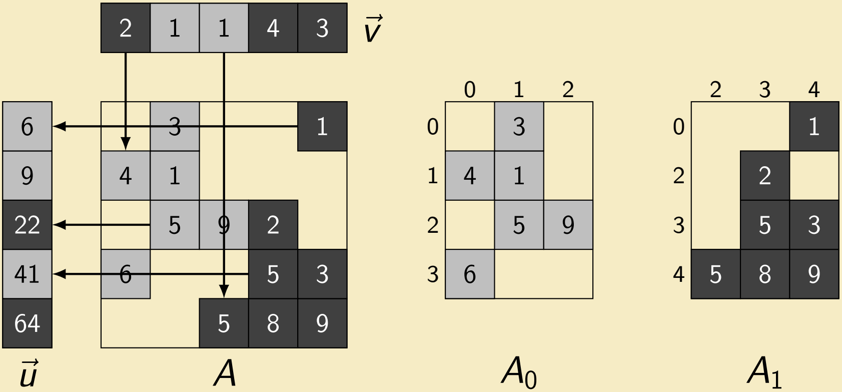
Parallel Sparse Matrix-Vector Multiplication

Parallel Algorithm: Fanin



Parallel Sparse Matrix-Vector Multiplication

Parallel Algorithm: Partial Sum Summation



Parallel Sparse Matrix-Vector Multiplication

Problem

- Balance work evenly between processors during local multiplication
- Avoid communication during fanout and fanin
- No need to avoid global synchonisation: only 4 supersteps

Parallel Sparse Matrix-Vector Multiplication

Computational load imbalance

- Define:
  - A_i: the subset of A distributed to processor i
  - $|A_i|$: the number of nonzeroes assigned to processor i
- Number of multiplications during local multiplication step is:

$$O(\max_i |A_i|)$$

- Allowed computational load imbalance ε:

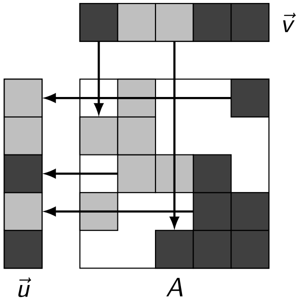
$$\max_i |A_i| \leq (1 + \varepsilon)\frac{N}{p}$$

Parallel Sparse Matrix-Vector Multiplication

Necessary communication

Note that communication during fanout and fanin is only necessary if:

- Nonzeroes of a single *column* are distributed to different processors.
- Nonzeroes of a single *row* are distributed to different processors.

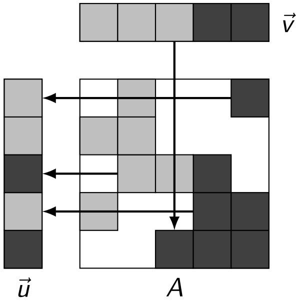


Parallel Sparse Matrix-Vector Multiplication

Necessary communication

Note that communication during fanout and fanin is only necessary if:

- Nonzeroes of a single *column* are distributed to different processors.
- Nonzeroes of a single *row* are distributed to different processors.



Matrix Partitioning

Communication Volume

- The communication cost of a single row or column i is given by:

$$C(i) = \lambda_i - 1$$

- λ_i is the number of different processor indices in row/column i
- The communication volume V of a matrix partitioning is given by:

$$V = \sum_{\text{rows,columns } i} C(i)$$

Matrix Partitioning

The matrix partitioning problem

- Given:
  - (sparse) matrix A, $m \times n$ and N nonzeroes
  - number of processors p
  - allowed load imbalance ε
- Find:
  - Partitioning of A into p subsets A_i, $0 \leq i < p$
  - $\max_i |A_i| \leq (1+\varepsilon)\frac{N}{p}$
  - V is the minimum out of all possible partitionings
- Problem is NP-Complete $\Rightarrow$ no polynomial-time algorithm
  - Naive optimal algorithm is $O(p^N)$

Matrix Partitioning

Earlier research

Mainly heuristic solutions: try to find 'good' solutions in polynomial time:

- `Mondriaan`, `PaToH`, `hMetis`
- Based on modelling the problem as a hypergraph partitioning problem
- Use recursive bisection instead of partitioning into p parts directly
- Multilevel scheme to enable solving large instances
- Partitioning based on Kernighan–Lin graph partitioning (1972), in Fiduccia–Matheyses version with vertex moves instead of swaps.

A new model for optimal partitioning

Naive algorithm

- For each nonzero, try every processor index $\in [0, p)$
- Return the partitioning with the lowest volume that obeys the load imbalance constraint

$$\implies O(p^N)$$

Restrictions

To reduce computation time for optimal solution:

- Only consider $p = 2$, i.e. bipartitioning
- Use strict load imbalance constraint:

$$m_1 = N - m_0 = \left\lfloor (1+\varepsilon)\frac{N}{p} \right\rfloor$$

where processor i has m_i nonzeroes $(i = 0, 1)$ out of N

A new model

The new model

- Define variables:
  - $v_c(i)$ for each column i
  - $v_r(i)$ for each row i
- If:
  - $v_c(i) = 0$: all nonzeroes in column i are assigned to processor 0
  - $v_c(i) = 1$: all nonzeroes in column i are assigned to processor 1
  - $v_c(i) = 2$: column i is *cut*
- Volume V is the number of rows and columns with a value of 2

A new model

Number of solutions

- Each row and column can have 3 different values $\Rightarrow 3^{m+n}$, for $m \times n$ matrix A
- Naive algorithm tries all solutions $\Rightarrow O(3^{m+n})$
- $O(2^N) > O(3^{m+n})$: new model is better if

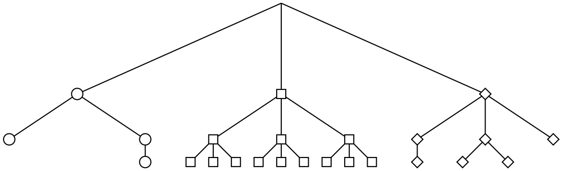
$$N > \log_2 3 \cdot (m+n) \approx 1.58 \cdot (m+n)$$

Infeasible solutions

Many solutions are infeasible:

- For a nonzero a_{ij} in column i and row j, if $v_c(i) = 0$ and $v_r(j) = 1$ or vice versa, the solution is infeasible!
- $\Rightarrow$ Large reduction of search space

Branch-and-Bound



Method

- Divide problem into increasingly smaller subproblems
- Arrange subproblems in branching tree
- Traverse tree (Depth-First Search)
- If lower bound of subproblem $\geq$ global upper bound, prune subtree
- A tree leaf is a feasible solution of the problem

Branch-and-Bound

Advantage

Branch-and-bound methods can reduce the number of subproblems that need to be checked by a large amount

Requirement

Good methods to provide lower bounds to subproblems and upper bounds to the main problem

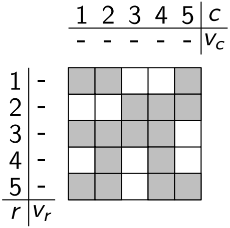
- Heuristics can be used to provide upper bounds

Branch-and-Bound matrix partitioning

Model

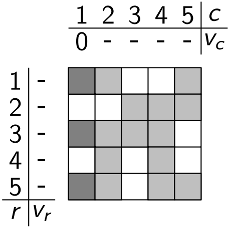
- Start with all $v_c(i)$ and $v_r(i)$ variables *unassigned*
- Alternately assign the next available unassigned $v_c(i)$ or $v_r(i)$ variable to 0, 1 or 2
- Prune subtree if partial assignment is infeasible, or a lower bound $\geq$ upper bound
- If all $v_c(i)$ and $v_r(i)$ variables are assigned, we have a feasible solution to the matrix partitioning problem

Branch-and-Bound matrix partitioning



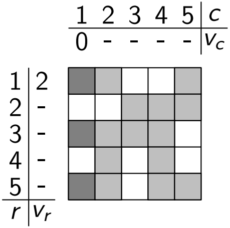
$5 \times 5,\ N = 16\ ,\ m_0 = m_1 = 8$

Branch-and-Bound matrix partitioning



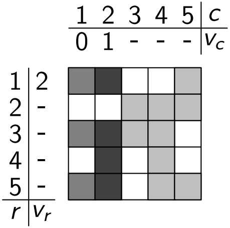
$$5 \times 5,\ N = 16\ ,\ m_0 = m_1 = 8$$

Branch-and-Bound matrix partitioning



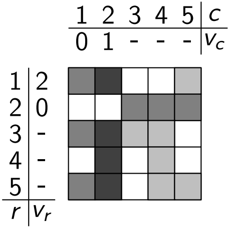
5×5, $N = 16$, $m_0 = m_1 = 8$

Branch-and-Bound matrix partitioning



$5 \times 5, \ N = 16 \ , \ m_0 = m_1 = 8$

Branch-and-Bound matrix partitioning



$$5 \times 5, \ N = 16 \ , \ m_0 = m_1 = 8$$

Branch-and-Bound matrix partitioning

Lower bound for communication

Given:

- A partial assignment of $v_c(i)$ and $v_r(i)$ variables

Find:

- A lower bound to the communication volume of all feasible partitionings that can be obtained by extending this partial assignment

Multiple bounds

We use three independent lower bounds, and the actual lower bound is the sum:

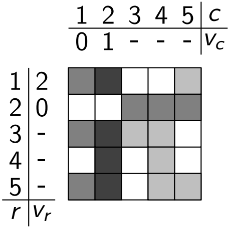
$$LB = b_1 + b_2 + b_3$$

Lower bounds

The first lower bound: b_1

Based on rows and columns that are defined to be cut:

- The number of rows and columns with a value of 2 is equal to b_1
- Here: $b_1 = 1$



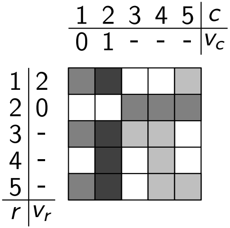
5×5, $N = 16$, $m_0 = m_1 = 8$

Lower bounds

The second lower bound: b_2

Based on rows and columns that are implicitly cut:

- Assigned columns j_1 and j_2 both have a nonzero in unassigned row i
- If $v_c(j_1) = 0$ and $v_c(j_2) = 1$ (or vice versa), row i *has* to be cut
- Here: $b_2 = 2$, because of rows 3 and 5



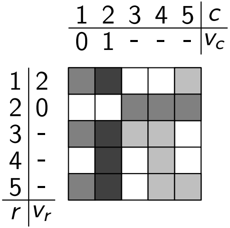
$$5 \times 5,\ N = 16\ ,\ m_0 = m_1 = 8$$

Lower bounds

The third lower bound: b_3

Based on partially assigned rows and columns:

- Some nonzeroes of column j are assigned to a certain processor
- To prevent cutting column j, *all* remaining nonzeroes have to be assigned to that processor
- The strict load imbalance equation might not allow this
- Here: column 4 must be cut, so $b_3 = 1$



5×5, $N = 16$, $m_0 = m_1 = 8$

Optimal results

Benchmark Results

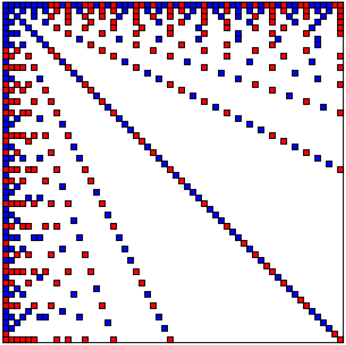
Solved matrices from University of Florida sparse matrix collection (max runtime 24 hours):

- Most nonzeroes: `bcsstk04`, 132×132, 3648 nonzeroes, $V_{opt} = 48$
- Largest dimensions: `bwm200`, 200×200, 796 nonzeroes, $V_{opt} = 4$
- Largest optimal volume: `bcsstk04`, 132×132, 3648 nonzeroes, $V_{opt} = 48$
- Largest fraction of rows and columns cut: `cage9` and `Stranke94`, both half of number of rows and columns cut

Optimal solutions inspire heuristic solutions

Characteristics

- Most optimal solutions are 2-dimensional
- Most columns and rows are completely assigned to a single processor
- If a row is completely assigned to processor *i*, the columns connected to this row are often also completely assigned to *i*



A partitioning model

Model

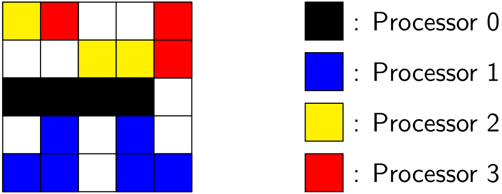
A model for matrix partitioning with general p, and normal load imbalance:

- Define a variable $v_c(i)$ for each column and $v_r(i)$ for each row
- $v_c(i)$ indicates to which processor column i is completely assigned
  $\Rightarrow$ if $v_c(5) = 3$, column 5 is assigned to processor 3
- Conflict if row and column of nonzero a_{ij} are assigned to different processors
  Resolution $\Rightarrow$ lowest processor index wins

Model example

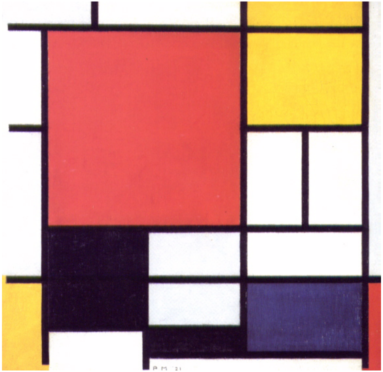
1	2	3	4	5	c
2	3	2	2	3	v_c

1	3
2	3
3	0
4	1
5	1
r	v_r



$$5 \times 5,\ N = 16\ ,\ m_0 = 4, m_1 = 6, m_2 = 3, m_3 = 3,\ V = 9$$

Composition with Red, Yellow, Blue and Black



Piet Mondriaan 1921

Matrix lns3937 (Navier–Stokes, fluid flow)

Splitting the sparse matrix lns3937 into 5 parts.

A greedy heuristic

Flipping rows/columns

- Define function $flip(q, i, j) \Rightarrow$ change $v_q(i)$ variable to value j ($q = r, c$)
- Flip cost is volume difference after and before the flip

Greedy algorithm

Partition $m \times n$ matrix A over p parts, with load imbalance constraint:

- Start with $v_c(i) = v_r(j) = p - 1$
- For $k = 0$ to $p - 2$:
  - Repeatedly flip the row or column from $p - 1$ to processor k with the lowest flip cost
  - Stop if no flips that obey load imbalance are available
- Return the partitioning obeying load imbalance with the lowest cost

KLFM

Kernighan–Lin Fiduccia–Matheyses

A popular local search method for partitioning is KLFM. We use a variant:

- Start with a feasible partitioning
- While there exists a feasible flip:
  - Apply feasible $\mathit{flip}(q, i, j)$ with lowest cost
  - Lock the flipped row/column to its new value
- Return partitioning with the lowest total cost encountered during run

Multiple runs

We can use the output of a single KLFM run as input to another KLFM run!

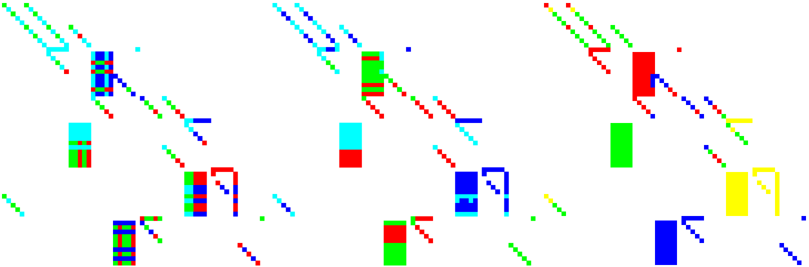
Mondriaan 2D matrix partitioning

- $p = 4$, $\epsilon = 0.2$, global non-permuted view

Fine-grain 2D matrix partitioning

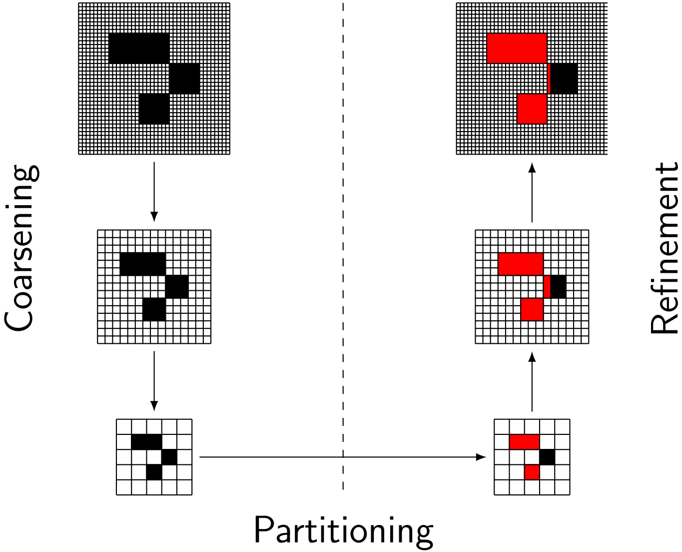
- Each individual nonzero is a vertex in the hypergraph, Çatalyürek and Aykanat, 2001.

A combined heuristic



`Mondriaan localbest` $V = 67$, `hybrid` $V = 33$ and combined heuristic $V = 30$

Multilevel method for matrices



Multilevel method

Merging rows and columns

- Rows and columns are merged, similar to the 2D coarsening method of Uçar, Çatalyürek, and Aykanat (2010).
- If more rows than columns, then split horizontally.

Variables maintained

- $nr_r(A_k, i)$: number of input matrix rows represented by row i of A_k
- $nr_c(A_k, i)$: number of input matrix columns represented by column i of A_k
- $nz(A_k, i, j)$: number of input matrix nonzeroes represented by nonzero a_{ij} of A_k

Results new heuristic

Matrix	N	m	n	Mond	New
gemat11	33185	4929	4929	1512	1122
lp_dfl001	35632	6071	12230	6131	6643
memplus	126150	17758	17758	13576	6667
cage10	150645	11397	11397	18952	15794
onetone2	227628	36057	36057	6277	6948
lp_cre_b	260785	9648	77137	9386	13031
finan512	596992	74752	74752	9289	8850
lhr34	764014	35152	35152	6858	6645
tbdmatlab	430171	19859	5979	52595	53622
bcsstk32	2014701	44609	44609	18763	17506
bcsstk30	2043492	28924	28924	22224	19937
tbdlinux	2157675	112757	20167	143863	177936

- Communication volume for Mondriaan 3.11 vs. new method
- $p = 64$

Conclusions and outlook

Conclusions

- Matrix partitioning is a different art than hypergraph partitioning.
- It is 2D!
- The new method was inspired by viewing optimal solutions, and by the benefits of keeping the nonzeros of rows or columns together in Mondriaan.
- We call the new method *overpainting*, since we paint rows and columns, and sometimes paint over old layers.

Outlook

ありがとうございます Thank you!

- Fast implementation is ready. It will be included in Mondriaan 4.0.
- MSc thesis Daan Pelt (August 2010, also for ILP results) is available via: http://igitur-archive.library.uu.nl/student-theses/2011-0404-200428/UUindex.html