Hypergraphs in Scientific Computing

Rob Bisseling

Department of Mathematics

Utrecht University

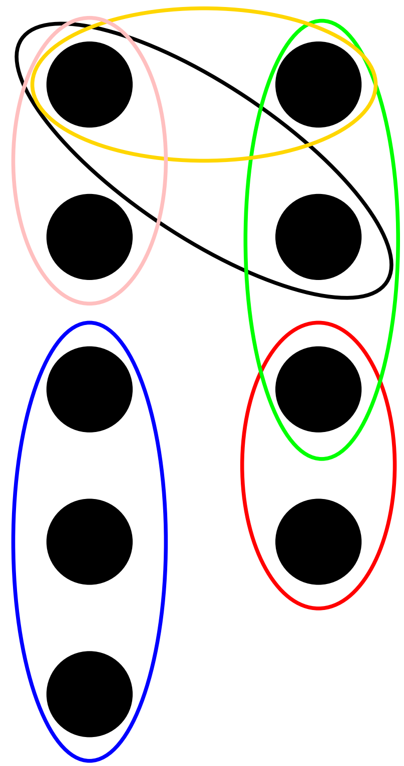
Support from BSIK-BRICKS/MSV and NCF

Universiteit Utrecht

Outline

1. Introduction
   - Hypergraphs
   - Motivation: parallel iterative solvers
2. Hypergraph partitioning
   - Sequential: Mondriaan
   - 1D and 2D sparse matrix partitioning
   - Parallel: Zoltan
3. Hypergraph applications
   - Parallel Google PageRank computation
   - Call-graph partitioning
4. Conclusions and future work

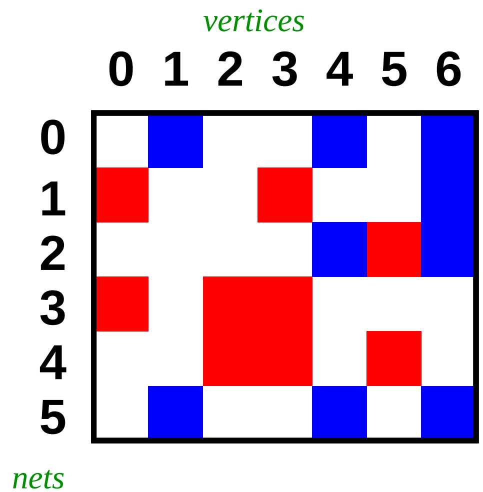
1. Introduction: Hypergraph



Hypergraph with 9 vertices and 6 hyperedges (nets)

Universiteit Utrecht

1D matrix partitioning using hypergraphs



Column bipartitioning of $m \times n$ matrix

- Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$
- Columns $\equiv$ Vertices: $0, 1, 2, 3, 4, 5, 6$.
  Rows $\equiv$ Hyperedges (nets, subsets of $\mathcal{V}$):

$$n_0 = \{1,4,6\}, \quad n_1 = \{0,3,6\}, \quad n_2 = \{4,5,6\},$$
$$n_3 = \{0,2,3\}, \quad n_4 = \{2,3,5\}, \quad n_5 = \{1,4,6\}.$$

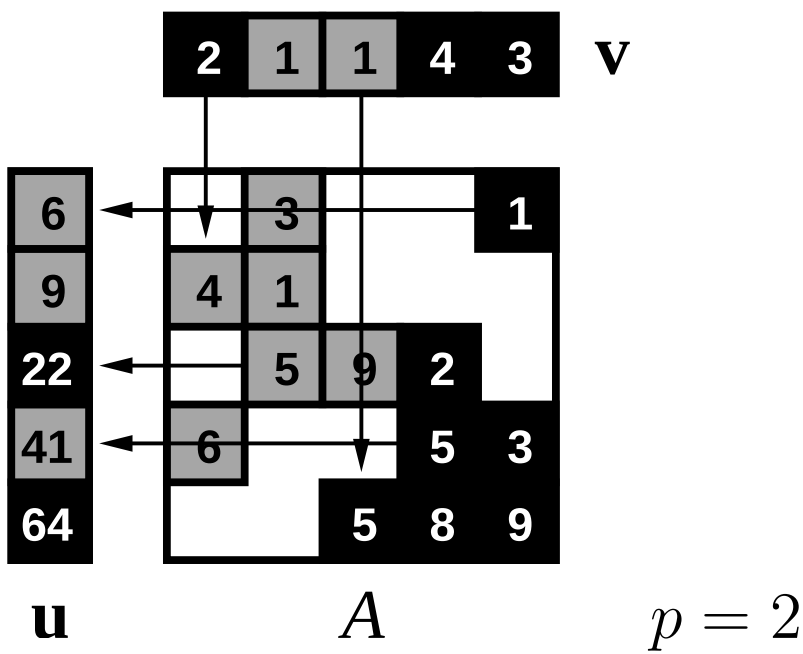
Motivation: parallel iterative solvers

- Iterative linear system solvers for $A\mathbf{x} = \mathbf{b}$.
- Iterative eigensystem solvers for $A\mathbf{x} = \lambda\mathbf{x}$.
- Basic building block: sparse matrix–vector multiplication.
- Parallel computation: often, the matrix is distributed by rows.

Parallel sparse matrix–vector multiplication $\mathbf{u} := A\mathbf{v}$

A sparse $m \times n$ matrix, $\mathbf{u}$ dense m-vector, $\mathbf{v}$ dense n-vector

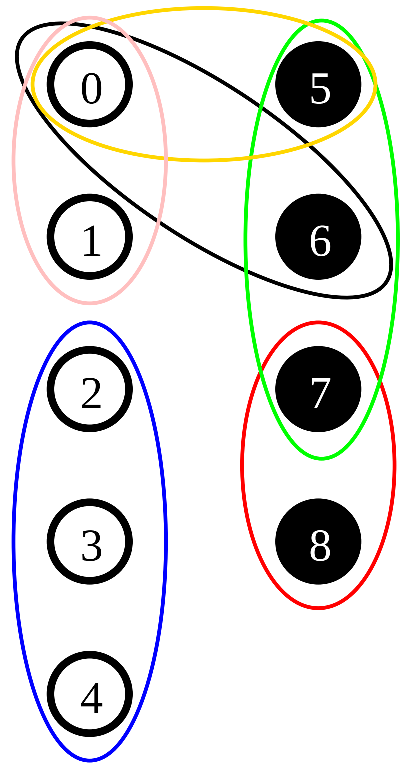
$$u_i := \sum_{j=0}^{n-1} a_{ij} v_j$$



4 phases: communicate, compute, communicate, compute

Universiteit Utrecht

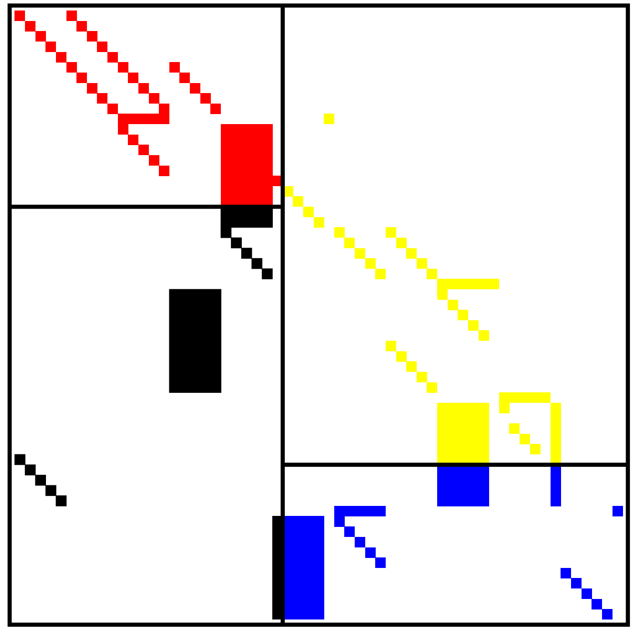
2. Hypergraph partitioning



Hypergraph with 9 vertices and 6 hyperedges (nets), partitioned in black and white vertices

Universiteit Utrecht

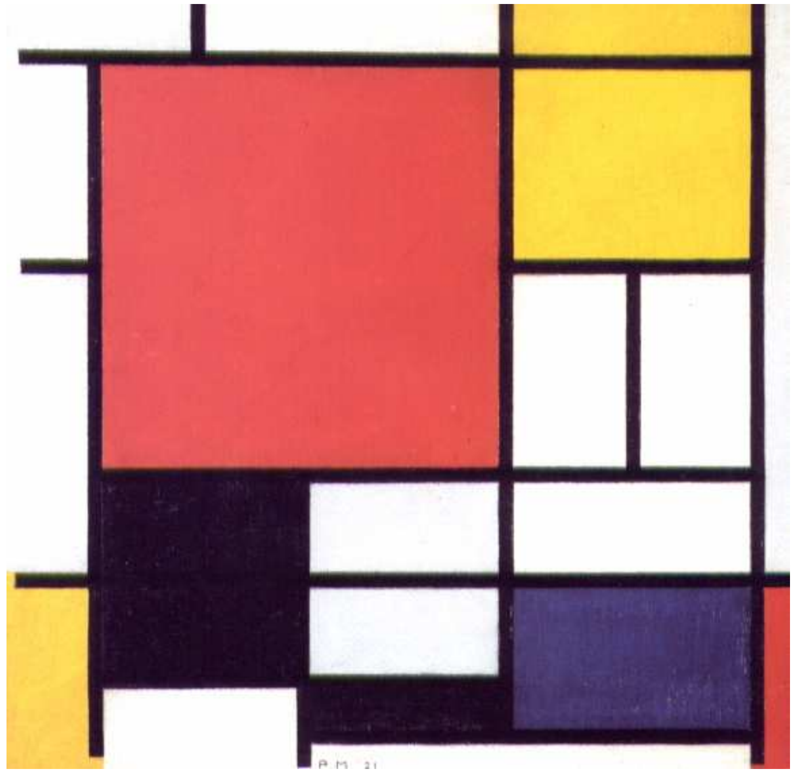
Mondriaan 2D matrix partitioning



- Block distribution (without row/column permutations) of 59×59 matrix `impcol_b` with 312 nonzeros, for $p = 4$
- Mondriaan package v1.0 (May 2002). Originally developed by Vastenhouw and Bisseling for partitioning term-by-document matrices for a parallel web search machine.

Universiteit Utrecht

Composition with Red, Yellow, Blue and Black



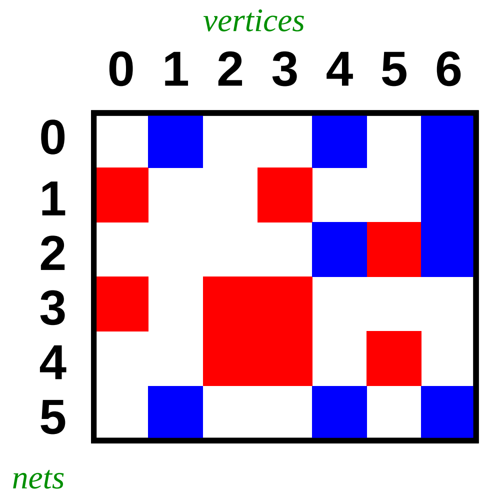
Piet Mondriaan 1921

Work imbalance criterion

$$nz(A_i) \leq \frac{nz(A)}{p}(1+\epsilon), \quad 0 \leq i < p.$$

The maximum amount of work should not exceed the average amount by more than a fraction ϵ.

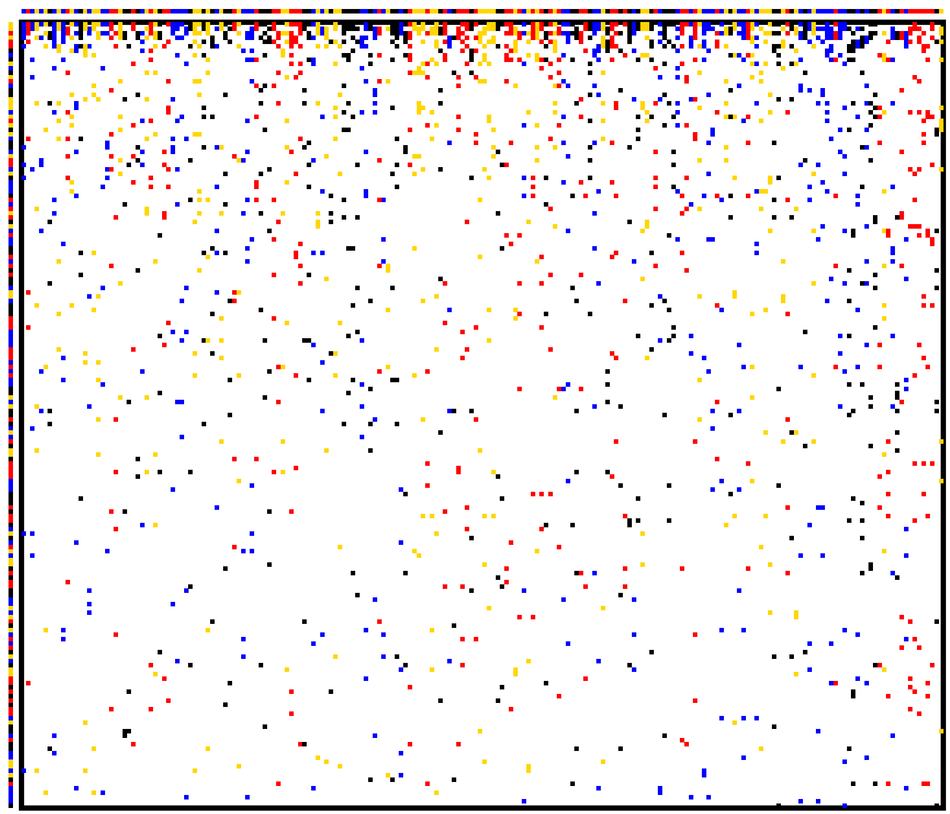
Minimising communication volume



- Cut nets: n_1, n_2 cause one horizontal communication
- Use Kernighan–Lin/Fiduccia–Mattheyses for hypergraph bipartitioning
- Multilevel scheme: merge similar columns first, refine bipartitioning afterwards
- Used in PaToH (Çatalyürek and Aykanat 1999) for 1D matrix partitioning.

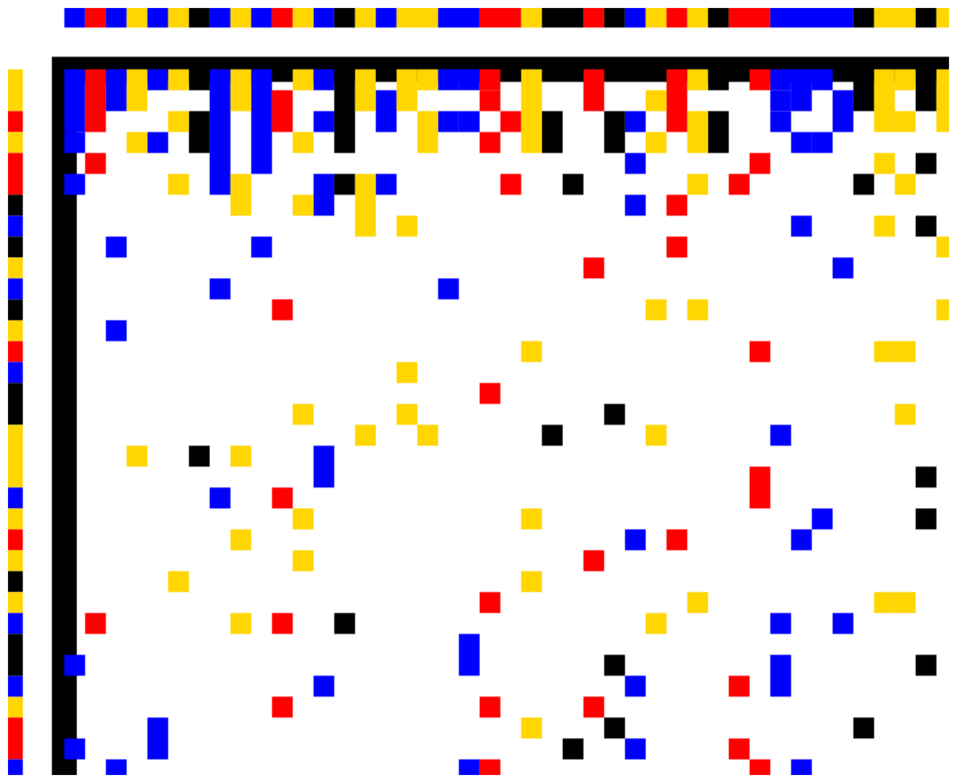
Quadratic sieving matrix MPQS30

Size 210×179, 1916 nonzeros, 30 decimal digits.
Partitioned for 4 processors (red, black, blue, orange) by the Mondriaan package



Matrix: courtesy of Richard Brent, 2001.

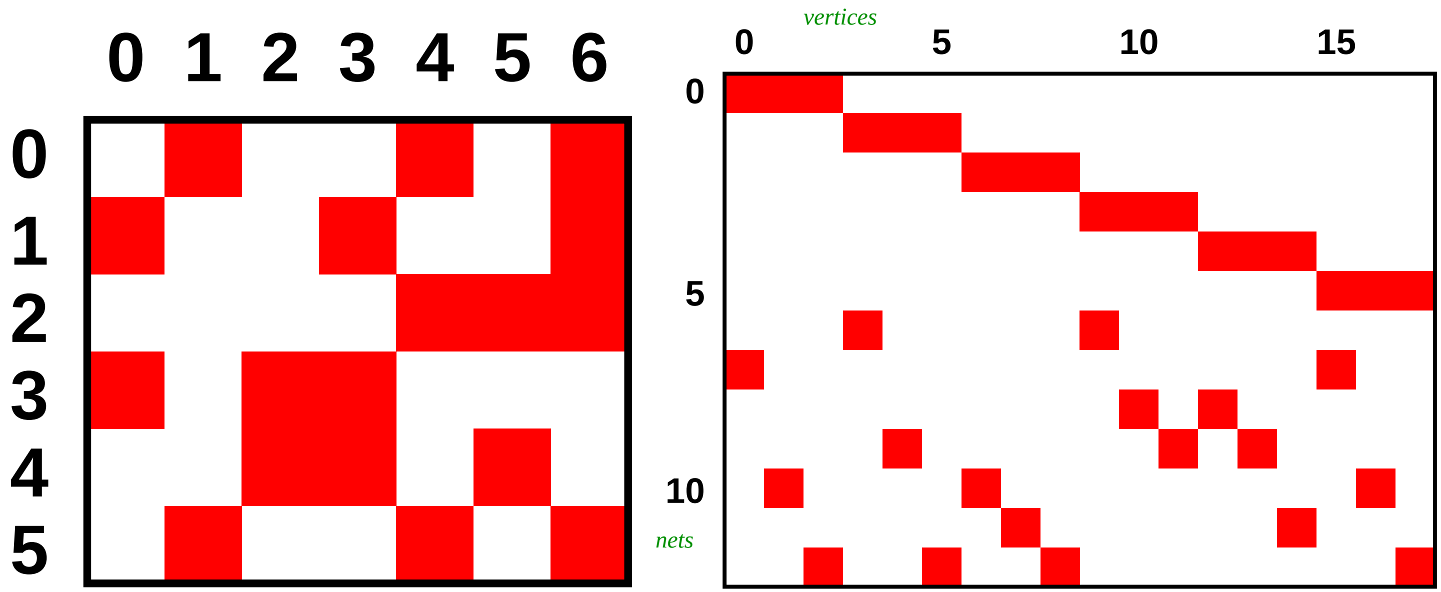
Left upper corner of `MPQS30`



Column distribution of matrix. Row spread over 4 processors causes 3 horizontal communications, hence: $\lambda - 1$ metric (or connectivity-1 metric, Lengauer 1990)

Universiteit Utrecht

A different hypergraph model: fine-grain

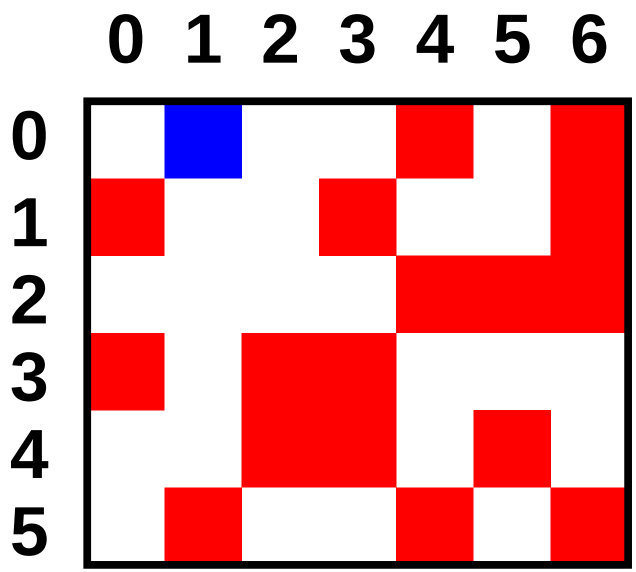


A $\qquad$ $F = F_A$

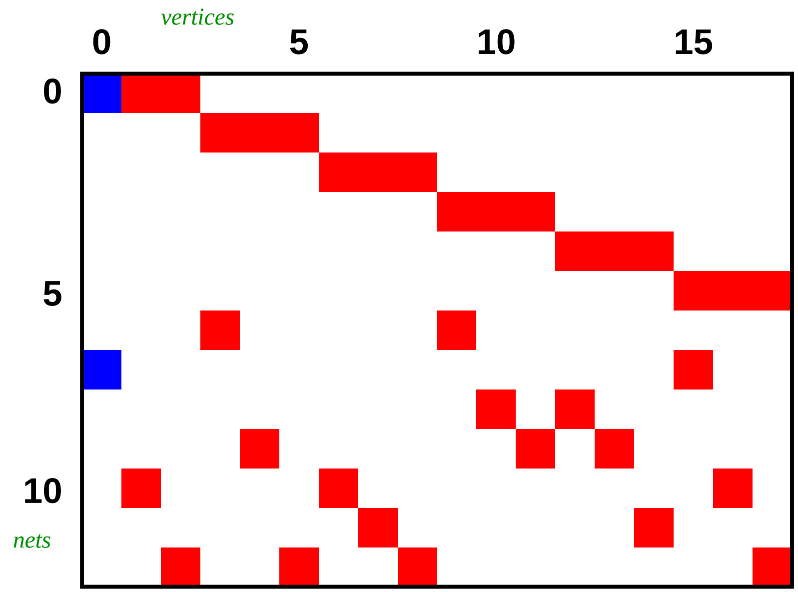
- Fine-grain model proposed by Çatalyürek and Aykanat, 2001
- $m \times n$ matrix A with $nz(A)$ nonzeros
- $(m+n) \times nz(A)$ matrix $F = F_A$ with $2 \cdot nz(A)$ nonzeros
- a_{ij} is kth nonzero of $A \Leftrightarrow f_{ik}$, $f_{m+j,k}$ are nonzero in F

Universiteit Utrecht

Communication for fine-grain model



A



$F = F_A$

- Cut net in first m nets of hypergraph of F: nonzeros from row a_{i*} are in different parts, hence horizontal communication in A.
- Cut net in last n nets of hypergraph of F: nonzeros from column a_{*j} are in different parts, vertical communication in A.

Recursive, adaptive bipartitioning algorithm

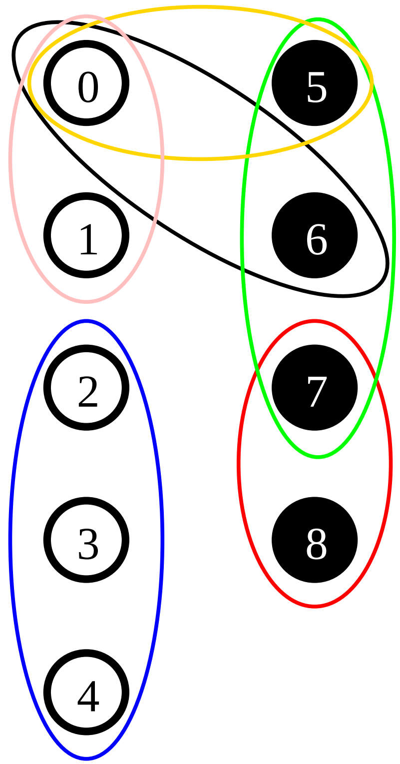
MatrixPartition(A, p, ϵ)
input: ϵ = allowed load imbalance, $\epsilon > 0$.
output: p-way partitioning of A with imbalance $\leq \epsilon$.
if $p > 1$ **then**
$q := \log_2 p$;
$(A_0^{\mathrm{r}}, A_1^{\mathrm{r}}) := h(A, \mathrm{row}, \epsilon/q)$; hypergraph splitting
$(A_0^{\mathrm{c}}, A_1^{\mathrm{c}}) := h(A, \mathrm{col}, \epsilon/q)$;
$(A_0^{\mathrm{f}}, A_1^{\mathrm{f}}) := h(A, \mathrm{fine}, \epsilon/q)$;
$(A_0, A_1) :=$ best of $(A_0^{\mathrm{r}}, A_1^{\mathrm{r}})$, $(A_0^{\mathrm{c}}, A_1^{\mathrm{c}})$, $(A_0^{\mathrm{f}}, A_1^{\mathrm{f}})$;

$maxnz := \frac{nz(A)}{p}(1+\epsilon)$;
$\epsilon_0 := \frac{maxnz}{nz(A_0)} \cdot \frac{p}{2} - 1$; MatrixPartition($A_0, p/2, \epsilon_0$);
$\epsilon_1 := \frac{maxnz}{nz(A_1)} \cdot \frac{p}{2} - 1$; MatrixPartition($A_1, p/2, \epsilon_1$);
else output A;

Parallel hypergraph partitioning: Zoltan

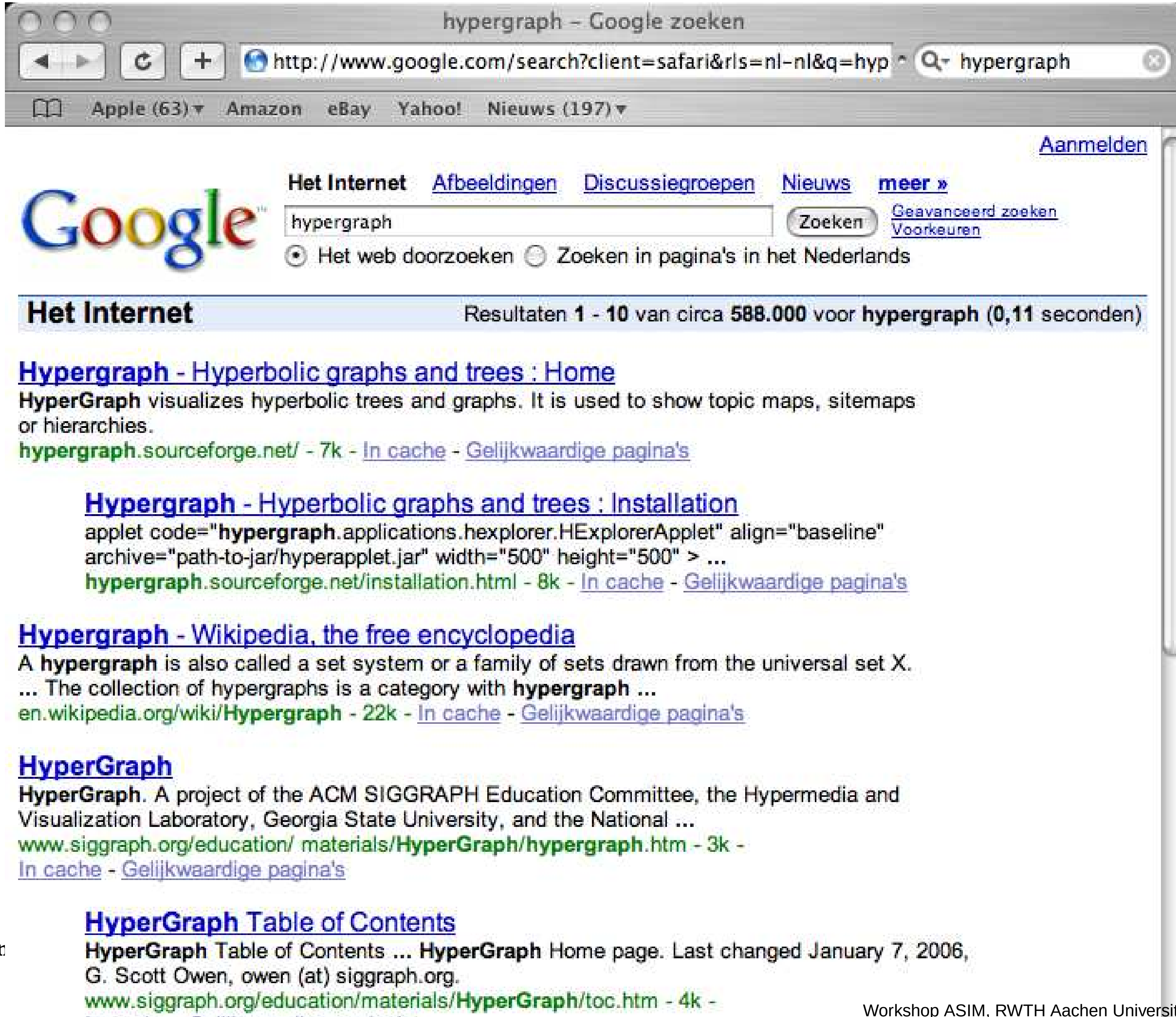
- Parallel hypergraph partitioner has been released in Zoltan version 2.0 in April 2006 by Sandia National Laboratories (Devine, Boman, Heaphy, Bisseling, Çatalyürek 2006).
- Internally: 2D Cartesian matrix distribution
- Performs parallel coarsening and refinement, and replicated initial partitioning on small hypergraph.
- Dynamic repartitioning recently added (2007), by adding fixed vertices representing partitions and adding migration nets.

3. Hypergraph applications



Universiteit Utrecht

Web searching: which page ranks first?



Un

The link matrix A

- Given n web pages with links between them.
  We can define the sparse $n \times n$ link matrix A by

$$a_{ij} = \begin{cases} 1 & \text{if there is a link from page } j \text{ to page } i \\ 0 & \text{otherwise.} \end{cases}$$

- Let $\mathbf{e} = (1, 1, \ldots, 1)^T$, representing an initial uniform importance (rank) of all web pages. Then

$$(A\mathbf{e})_i = \sum_j a_{ij} e_j = \sum_j a_{ij}$$

  is the total number of links pointing to page i.

- The vector $A\mathbf{e}$ represents the importance of the pages; $A^2\mathbf{e}$ takes the importance of the pointing pages into account as well; and so on.

The Google matrix

- A web surfer chooses each of the outgoing N_j links from page j with equal probability. Define the $n \times n$ diagonal matrix D with $d_{jj} = 1/N_j$.
- Let α be the probability that a surfer follows an outlink of the current page. Typically $\alpha = 0.85$. The surfer jumps to a random page with probability $1 - \alpha$.
- The Google matrix is defined by (Brin and Page 1998)

$$G = \alpha A D + (1 - \alpha)\mathbf{e}\mathbf{e}^T / n.$$

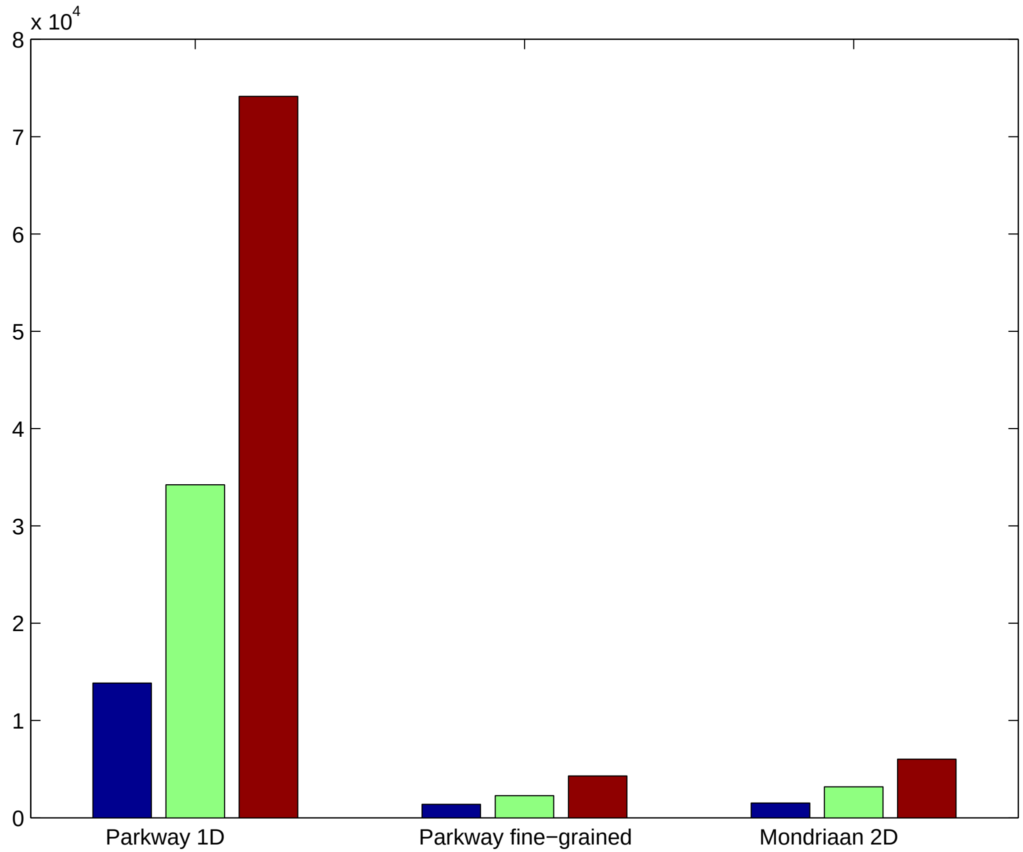
- The PageRank of a set of web pages is obtained by repeated multiplication by G, involving sparse matrix–vector multiplication by A, and some vector operations.

Comparing 1D, 2D fine-grain, and 2D Mondriaan

- The following 1D and 2D fine-grain communication volumes for PageRank matrices are published results from the parallel program Par$k$way v2.1 (Bradley, de Jager, Knottenbelt, Trifunović 2005).
- 2D fine-grain means: every nonzero becomes a vertex in the hypergraph.
- The 2D Mondriaan volumes are results for new features to be incorporated in v2.0.
- 2D Mondriaan means: in row-wise splits, every row becomes a vertex in the hypergraph. Similar for columns.
- All methods are hypergraph-based.

Universiteit Utrecht

Communication volume: PageRank matrix Stanford

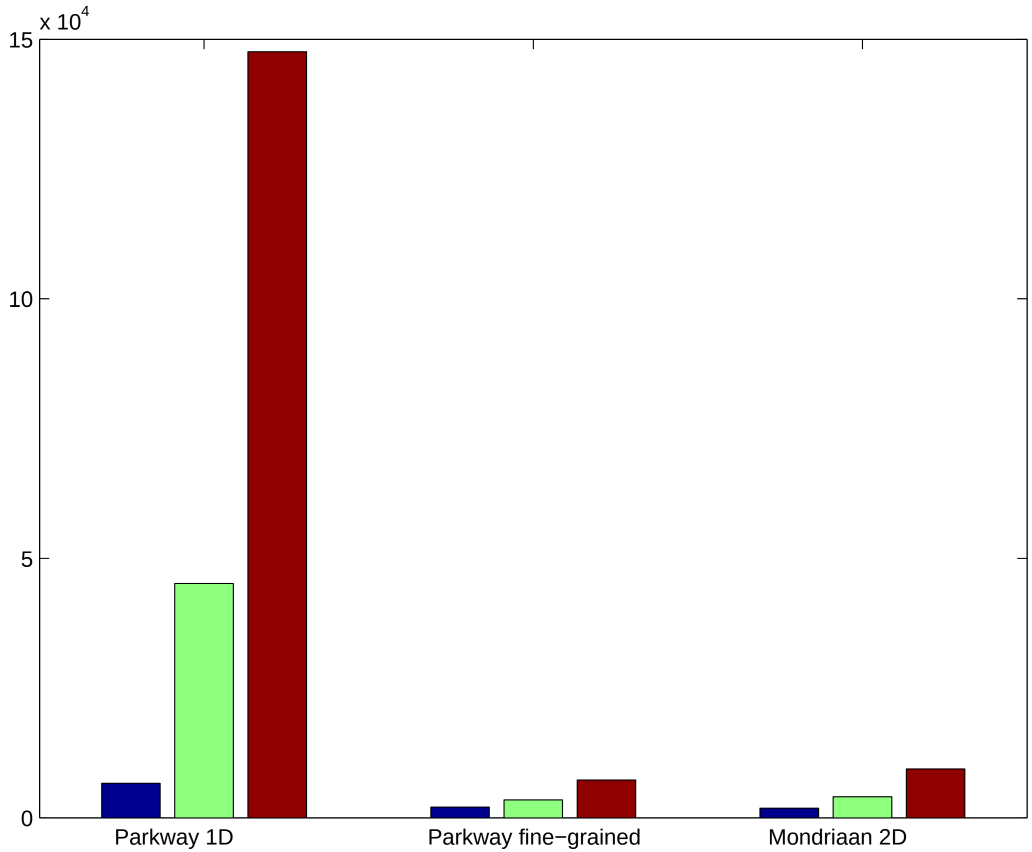


$p = 4, 8, 16$

- $n = 281,903$ (pages), $nz(A) = 2,594,228$ nonzeros (links).
- Represents the Stanford WWW subdomain, obtained by a web crawl in September 2002 by Sep Kamvar.

Universiteit Utrecht

Communication volume: `Stanford_Berkeley`



$p = 4, 8, 16$

- $n = 683,446$, $nz(A) = 8,262,087$ nonzeros.
- Represents the Stanford and Berkeley subdomains, obtained by a web crawl in Dec. 2002 by Sep Kamvar.

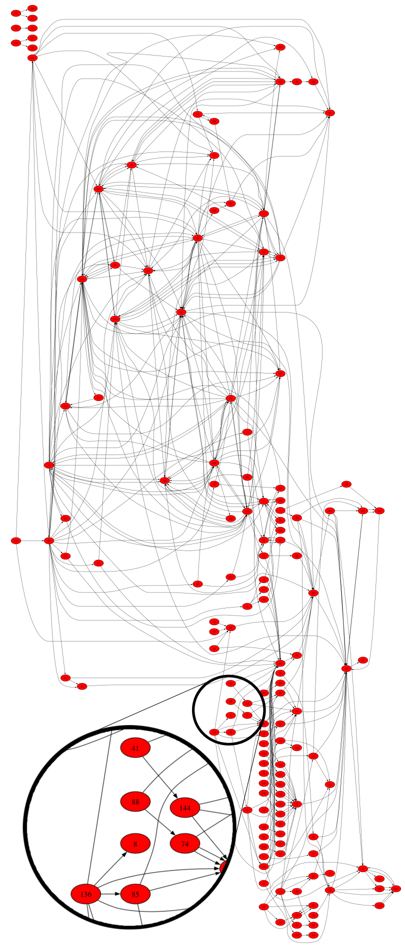
Universiteit Utrecht

Discussion of results

- 2D methods save an order of magnitude in communication volume compared to 1D.
- Parkway fine-grain is slightly better than Mondriaan, in terms of partitioning quality. This may be due to a better implementation, or due to the fine-grain method itself. Further investigation is needed.
- 2D Mondriaan is much faster than fine-grain, since the hypergraphs involved are much smaller: 7×10^5 vs. 8×10^6 vertices for `Stanford_Berkeley`.

Universiteit Utrecht

Call-graph partitioning



System with $N = 158$ vertices (programs, Java classes) provided by Software Improvement Group, Amsterdam.

Motivation: legacy code

- Huge Cobol systems with 1000s of programs calling each other.
- Today, more programs are written in Cobol than ever.
- Cobol programs are also written in Java . . .
- Software Improvement Group tries to split systems into manageable modules.
- Size of interfaces to other modules should be minimized.

$$\Downarrow$$

Graph formulation of the problem

- Program i is vertex in directed graph (V, A).
- Call from i to j (i uses j) is an arc,

$$(i, j) \in A \equiv i \rightarrow j.$$

- Partition the vertices of V into disjoint subsets, or modules, $V_1, \ldots, V_L$.
- A vertex $j \in V_s$ with an incoming edge $i \rightarrow j$, where $i \in V_t$ $(t \neq s)$, is an interface vertex. It represents a program that has to serve other modules.
- The problem: find a partitioning $V_1, \ldots, V_L$ with a minimum number $|I|$ of interface vertices and a reasonable workload for each module, i.e., $|V_l| \leq K$ for all l.

The sparse matrix connection

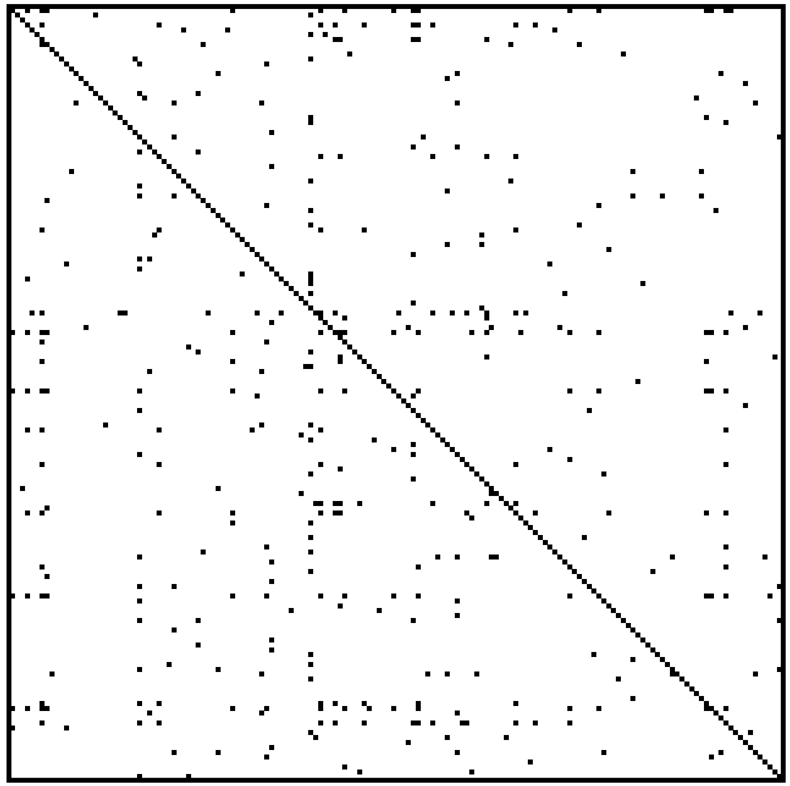
If you are a hammer, everything looks like a sparse matrix

- For a directed graph with N vertices, we define the $N \times N$ adjacency matrix A by

$$a_{ij} = \begin{cases} 1 & \text{if } i \rightarrow j, \\ 0 & \text{otherwise.} \end{cases}$$

- The matrix is square, unsymmetric, sparse.
- We also assume that each program calls itself, $a_{ii} = 1$.

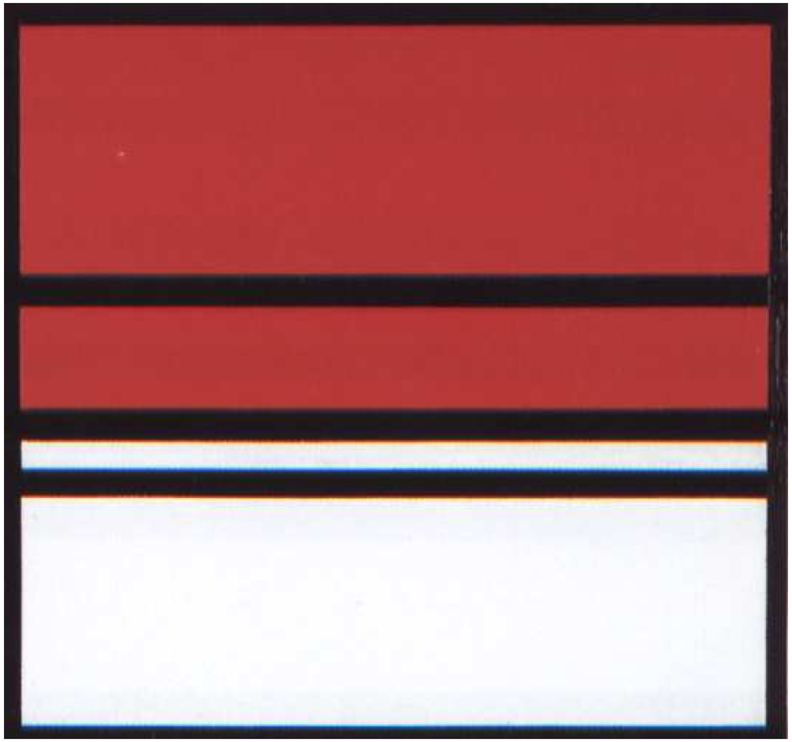
Adjacency matrix of Java1



158×158 adjacency matrix with 158 programs and 422 calls from programs to other programs. The matrix, including the unit diagonal, has 580 nonzeros.

Universiteit Utrecht

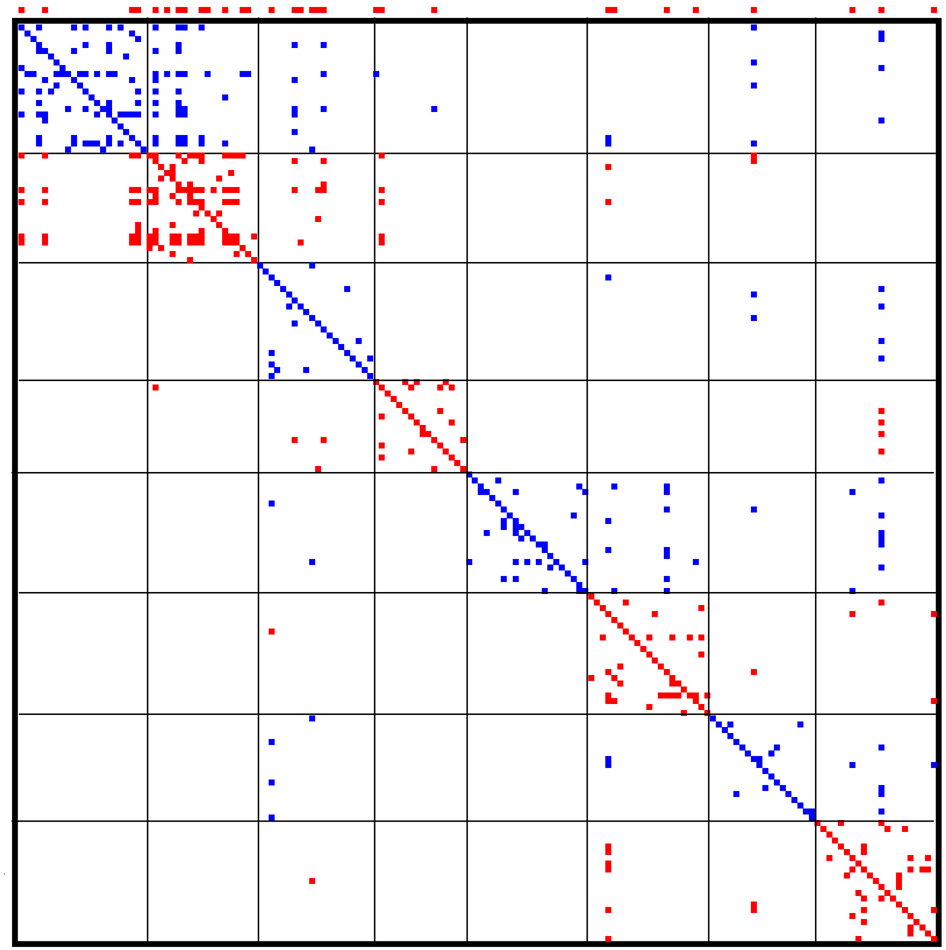
Mondriaan in 1D mode



Piet Mondriaan, 1943 (detail)

- We partition the rows of the adjacency matrix into blocks of equal size. We are allowed to permute the rows.

Partitioned adjacency matrix, after permutation



8 modules (row blocks)

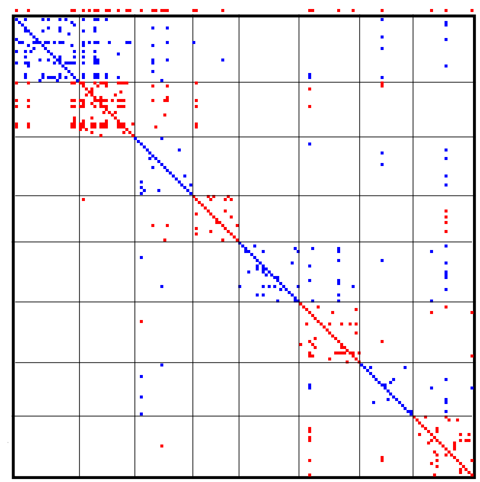
Universiteit Utrecht

Permutations

- Rows of the 158×158 adjacency matrix `Java1` are permuted to bring programs (matrix rows) of the same module in the partitioning together.
- Columns are permuted by the same permutation.
- Each column j with nonzeros in different blocks represents an interface program.
- The permutation corresponds to a solution with $|I| = 30$ interface programs produced by Mondriaan.

Universiteit Utrecht

Hypergraph formulation



- Each row is a vertex in the hypergraph.
- Each column is a hyperedge (net).
- Each column j with nonzeros in different blocks represents a cut net.
- The total number of interfaces is the number of cut nets. This is the cut-net metric. (The number of parts involved in a column does not matter.)

Universiteit Utrecht

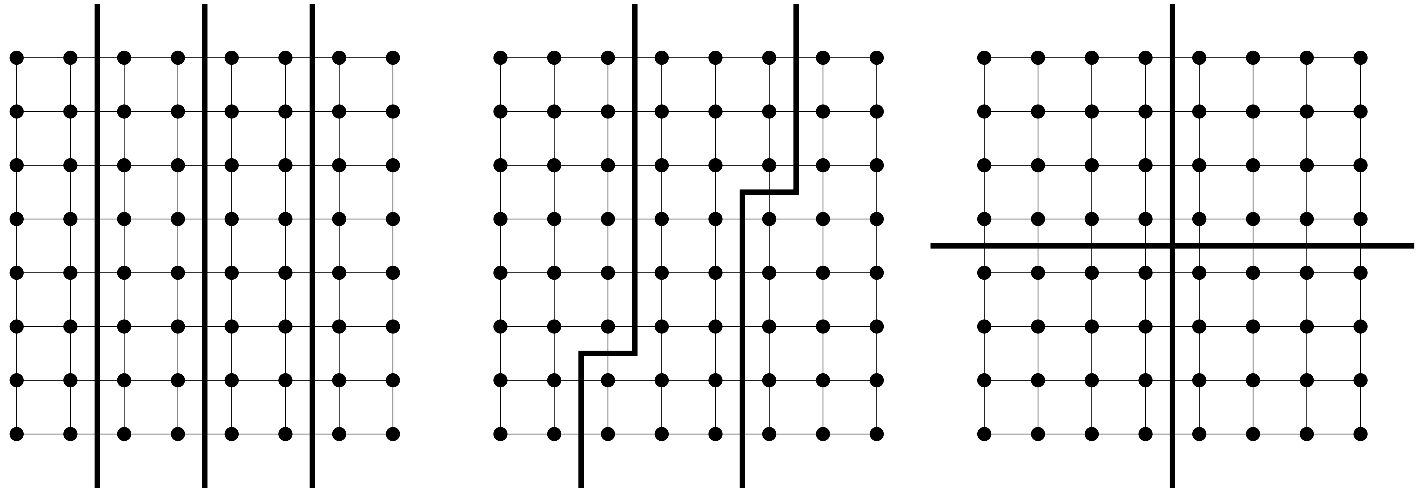
Results for hypergraph partitioning vs. optimal ILP

Problem	$\|V\|$	$\|A\|$	K	Interface size		Time (s)	
				ILP	HP	ILP	HP
`Java1`	144	422	23	26	30	103	0.06
`Java3`	837	5252	127	242	275	246456	0.54
`Java4`	15	39	2	11	11	0.22	0.001
`Cobol1`	947	1900	209	13	17	118	0.33
`Cobol2`	449	659	81	6	10	351	0.12
`Cobol3`	1145	2686	203	51	69	6452	0.34
`Cobol4`	1100	2951	167	32	52	742172	0.41

- Integer linear programming (ILP) using CPLEX v8.1, hypergraph partitioning (HP) using Mondriaan v1.01.
- 8 modules, 20% allowed imbalance.

Universiteit Utrecht

Application: static 8×8 *computational mesh*



(a) (b) (c)

- Computation cost: corner point 3 flops, border point 4, interior point 5.
- Communicating one data word costs, say, 10 flops.
- Limit communication by short borders.

WARNING: A mesh is not a matrix!

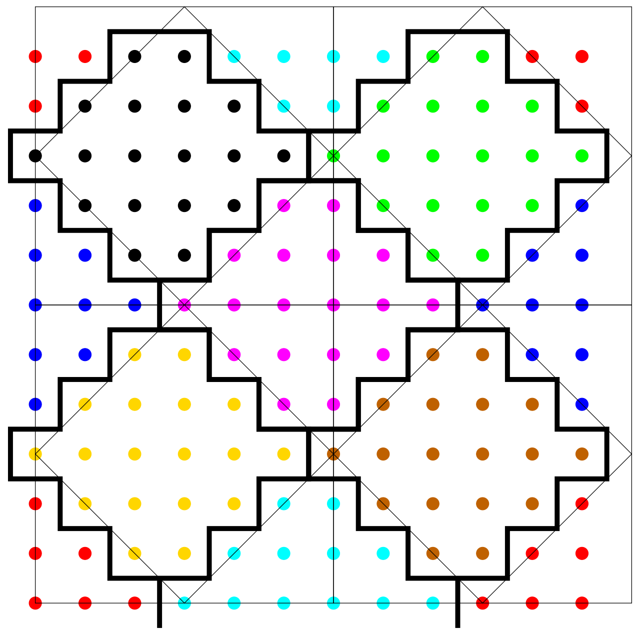
Universiteit Utrecht

Alhambra: diagonal cuts

August 2002

Universiteit Utrecht

12×12 *computational mesh: periodic partitioning*

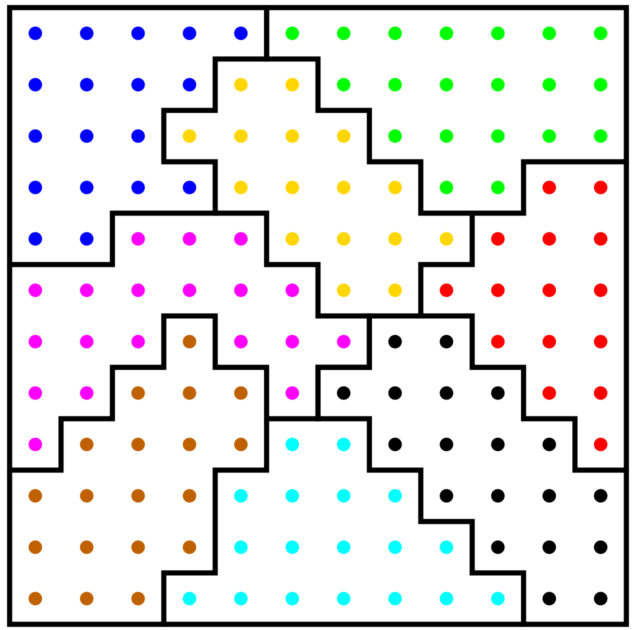


8 processors

- Total computation: 672 flops. Avg 84. Max 90.
- Communication: 104 values. Avg 13. Max 14.
- Total time: $90 + 10 \cdot 14 = 230$.
- Rectangular 6×3 blocks: time is $87 + 10 \cdot 15 = 237$.

Universiteit Utrecht

Hypergraph-based 1D Mondriaan partitioning



8 processors

- Total computation: 672 flops. Avg 84. Max 91.
- Communication: 85 values. Avg 10.525. Max 16.
- Total time: $91 + 10 \cdot 16 = 251$.
- Can be improved manually. Current best solution is 199 [Bas den Heijer, March 2006, using simulated annealing].

Universiteit Utrecht

4. Conclusions and . . .

- Hypergraphs are a powerful tool in scientific computing.
- Applications are everywhere:
  - Parallel iterative solvers ($\lambda - 1$ metric)
  - Parallel Google Pagerank computation ($\lambda - 1$ metric)
  - Call-graph partitioning (cut-net metric)
- Hypergraph partitioning algorithms for parallel computing are an example of Combinatorial Scientific Computing, the area of combinatorial algorithms enabling scientific computation.
- Parallel hypergraph partitioner has been released in Zoltan version 2.0 in April 2006 by Sandia National Laboratories.

. . . *future work*

- We keep on improving the serial Mondriaan hypergraph partitioner.
- We also work (with Ken Stanley) on visualisation and Matlab interface.
  Movie by Sarai Bisseling.