Mondriaan partitioning for faster parallel integer factorisation

Rob Bisseling and Ildikó Flesch

Mathematical Institute

Utrecht University, the Netherlands

Universiteit Utrecht

Outline

1. Attacking cryptosystems:
   - integer factorisation attack on RSA
   - sparse binary matrix
   - block Lanczos algorithm
2. Mondriaan partitioning
   - sparse matrix–vector multiplication
   - matrix partitioning (joint with Brendan Vastenhouw)
   - vector partitioning (joint with Wouter Meesen)
3. Experimental results
4. Another application: PageRank
   - Ranking web pages (joint with Tristan van Leeuwen, Ümit Çatalyürek)
5. Conclusions and future work

Cracking RSA

- RSA cryptosystem is based on difficulty of integer factorisation.
- Aim: given large n, find primes p, q such that $pq = n$.
- Recent record: May 9, 2005. RSA-200 with 200 decimal digits by Bahr, Böhm, Franke, Kleinjung.
- 55 CPU years of sieving (on 2.2 GHz Opterons) gives many pairs (a, b) with $a \equiv b \pmod{n}$. Each a and b is composed of small primes.
- Example for $n = 33$:
  $a_1 = 2^2 \cdot 7$, $b_1 = -1 \cdot 5$
  $a_2 = 7^3$, $b_2 = -1 \cdot 2^2 \cdot 5$.
- Note: $a_1 \cdot a_2 = (2 \cdot 7^2)^2$ and $b_1 \cdot b_2 = (-1 \cdot 2 \cdot 5)^2$.

Universiteit Utrecht

Solving sparse linear systems in GF(2)

- In general, desired subset S of pairs (a_j, b_j) such that $\prod_{j\in S} a_j$ and $\prod_{j\in S} b_j$ are both square.
- Translate into linear algebra. Bitmatrix A:
  a_{ij} = exponent of prime p_i in a_j (mod 2), where p_i is the ith prime, i.e., $p_1 = 2, p_2 = 3, p_3 = 5$, etc. and $p_0 = -1$.
- A is sparse, since not all primes are represented in an a_j.
- $A\mathbf{x}$ is linear combination of columns in A.
  Solving $A\mathbf{x} = 0$ in GF(2) gives S = $\{j : x_j = 1\}$.

Universiteit Utrecht

Example: matrix A

a	25	32	1	28	40	35	2560	128	125	343
$p = 2$	0	5	0	2	3	0	9	7	0	0
$p = 5$	2	0	0	0	1	1	1	0	3	0
$p = 7$	0	0	0	1	0	1	0	0	0	3

Take the entries modulo 2:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Example: matrix C

- Also generate B. Solve $A\mathbf{x} = 0$ and $B\mathbf{x} = 0$ together. Let $C\mathbf{x} = 0$ be the larger simultaneous system:

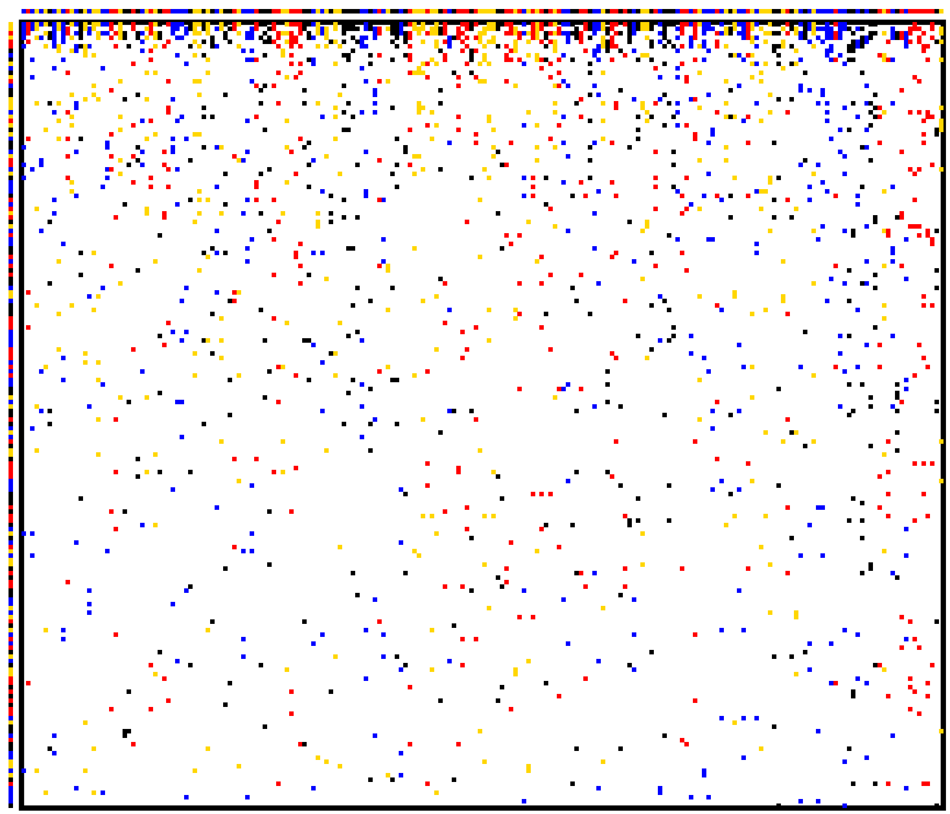
$$C = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

- For RSA-200, solution took 3 months on a cluster of 80 processors. Sparse matrix C has 64 million rows and columns and 11×10^9 nonzeros.

Universiteit Utrecht

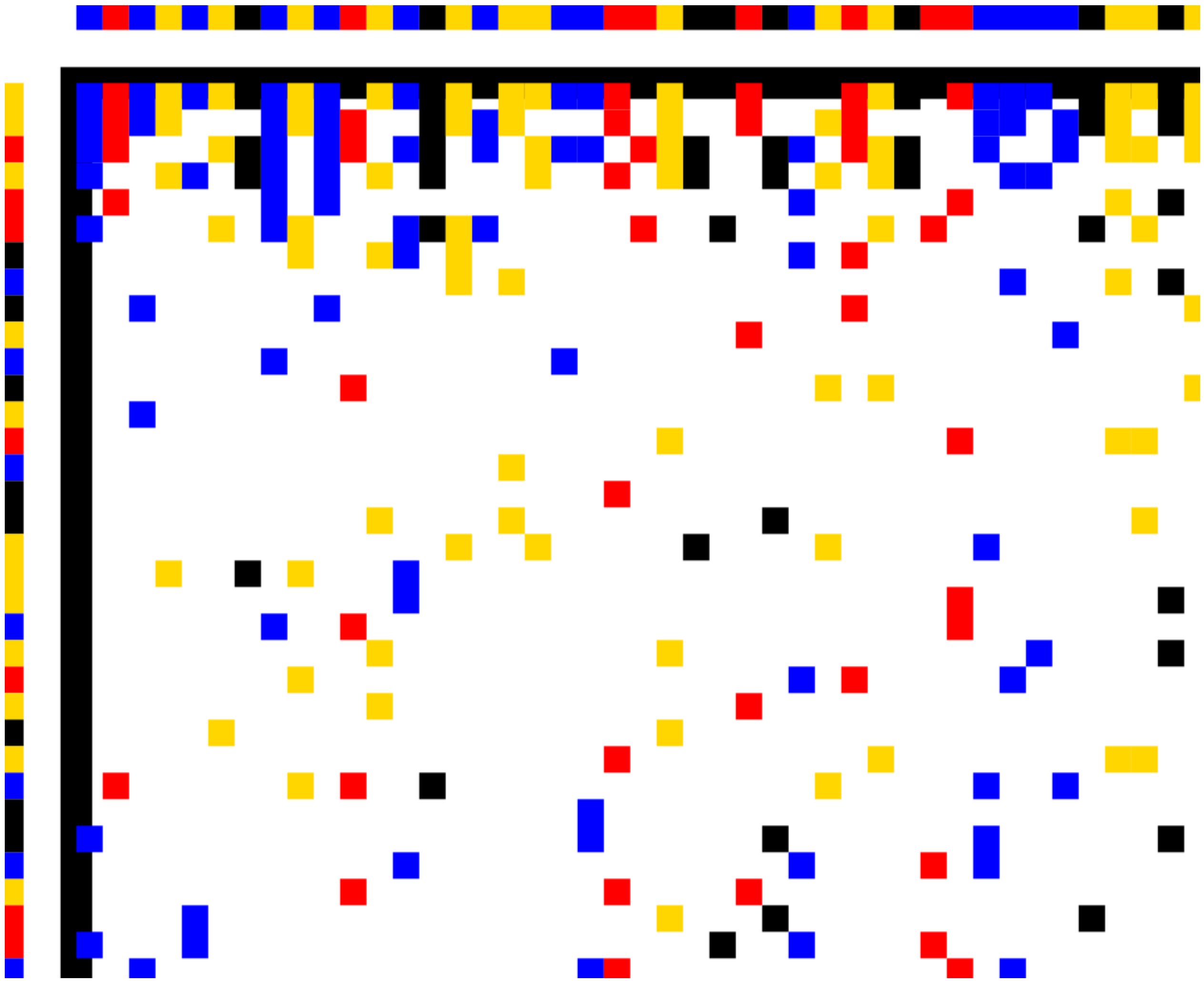
Quadratic sieving matrix `MPQS30`

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*



Universiteit Utrecht

Block Lanczos algorithm by Montgomery (1995)

- Use Lanczos for symmetric systems: solve $C^T C\mathbf{x} = 0$.
- Find 32 different solutions by the block Lanczos algorithm, solving a system $C^T C X = 0$.
  X has 32 columns (word size of the computer) and can be viewed as an integer vector.
- C and C^T are not explicitly multiplied.
  Only C is stored: rectangular sparse bitmatrix.

Main loop of block Lanczos algorithm

input: C = sparse $n_1 \times n_2$ bitmatrix,
Y = dense random $n_2 \times 32$ bitmatrix.
output: X = dense $n_2 \times 32$ bitmatrix such that $C^T C X = C^T C Y$.
while $Cond_i \neq 0$ **do**

$[W_i^{\text{inv}}, SS_i^T] = \ldots;$ $\{32 \times 32\}$
$X = X + V_i * (W_i^{\text{inv}} * (V_i^T * V_0));$
$C^T CV_i = C^T \circledast CV_i;$ {matvec}
$K_i = (V^T C_i^T * (C \circledast (C^T CV_i))) * SS_i^T + Cond_i;$

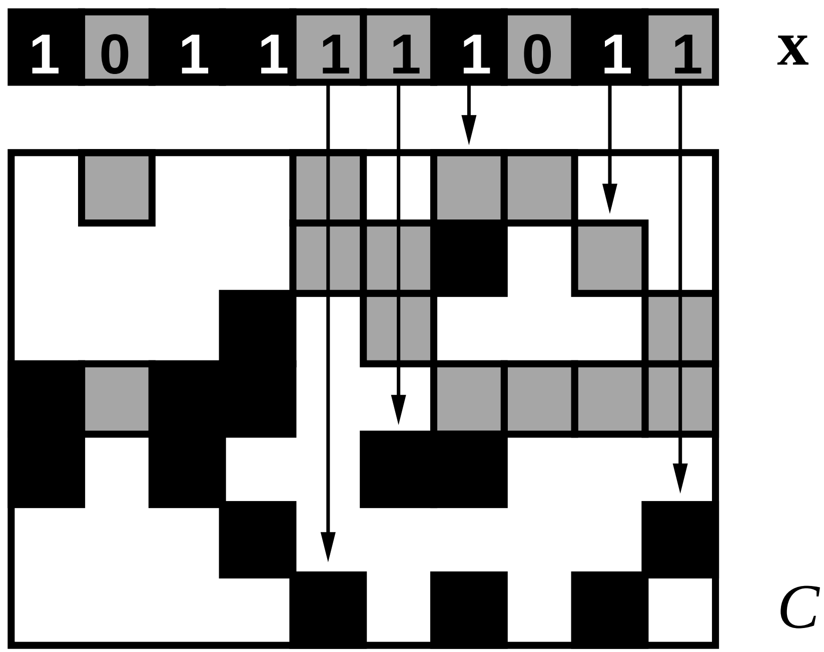
$D_{i+1}, E_{i+1}, F_{i+1} = \ldots;$ $\{32 \times 32\}$
$V_{i+1} = C^T CV_i * SS_i^T + V_i * D_{i+1} + V_{i-1} * E_{i+1} + V_{i-2} * F_{i+1}$
$V^T C_{i+1}^T = V_{i+1}^T \circledast C^T;$
$CV_{i+1} = C \circledast V_{i+1};$
$Cond_{i+1} = V^T C_{i+1}^T * CV_{i+1};$
$i = i + 1$

Universiteit Utrecht

Parallel sparse matrix–vector multiplication $\mathbf{y} := C\mathbf{x}$

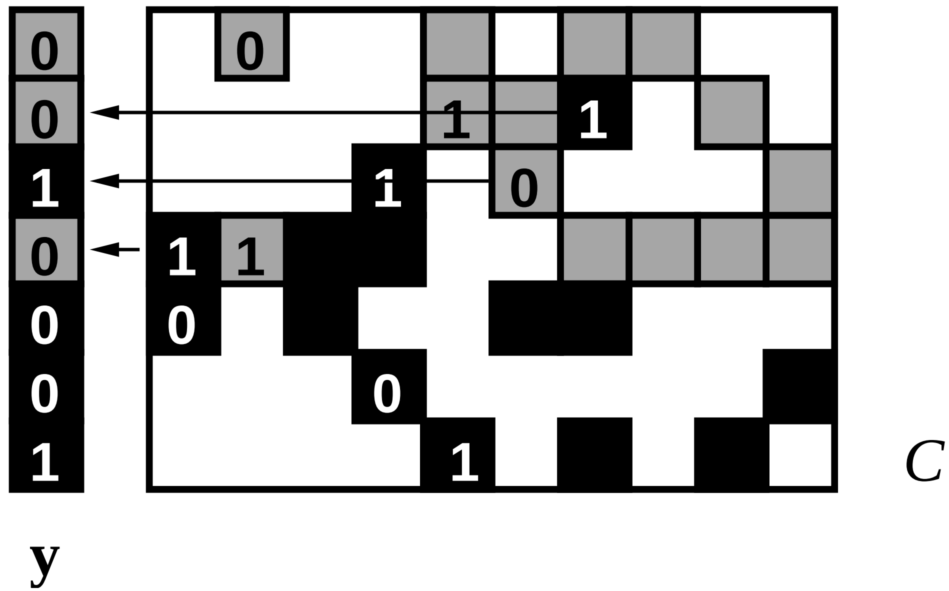
C sparse $n_1 \times n_2$ matrix, $\mathbf{y}$ dense n_1-vector, $\mathbf{x}$ dense n_2-vector

$$y_i := \sum_{j=0}^{n_2-1} a_{ij} x_j$$



Vertical communication. $p = 2$

Parallel sparse matrix–vector multiplication (cont'd)

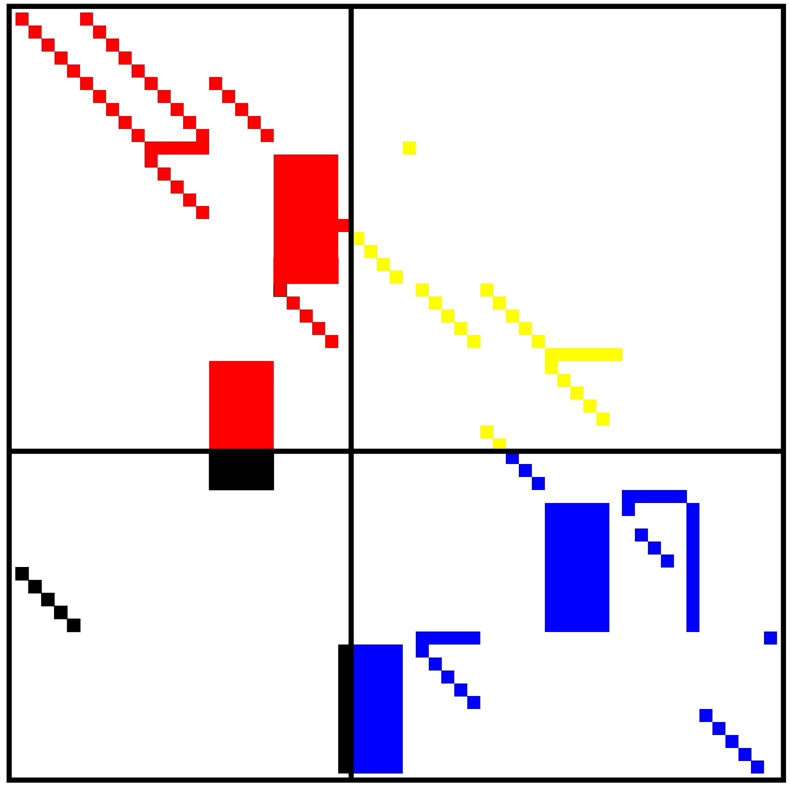


Horizontal communication. $p = 2$

- Algorithm has 4 supersteps: communicate, compute, communicate, compute

Universiteit Utrecht

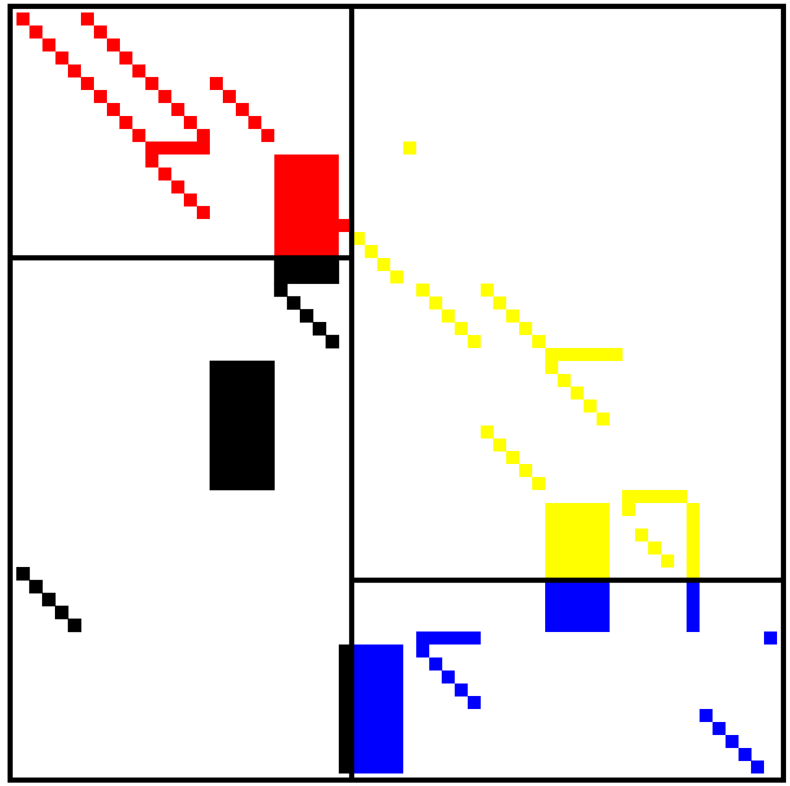
Cartesian matrix partitioning



- Block distribution of 59×59 matrix `impcol_b` with 312 nonzeros, for $p = 4$
- #nonzeros per processor: 126, 28, 128, 30

Universiteit Utrecht

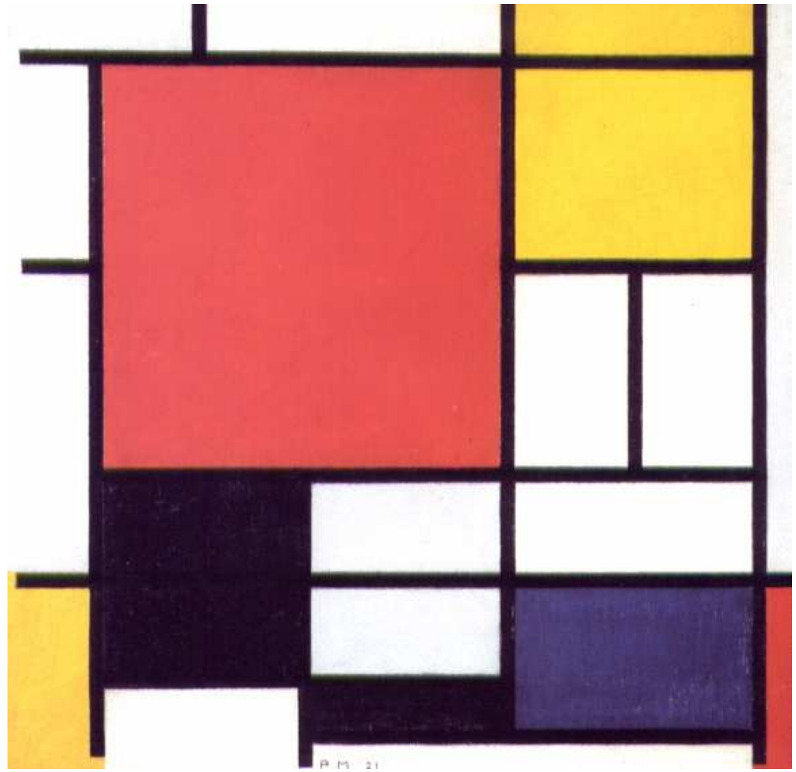
Non-Cartesian matrix partitioning



- Block distribution of 59×59 matrix `impcol_b` with 312 nonzeros, for $p = 4$
- #nonzeros per processor: 76, 76, 80, 80

Universiteit Utrecht

Composition with Red, Yellow, Blue and Black



Piet Mondriaan 1921

Universiteit Utrecht

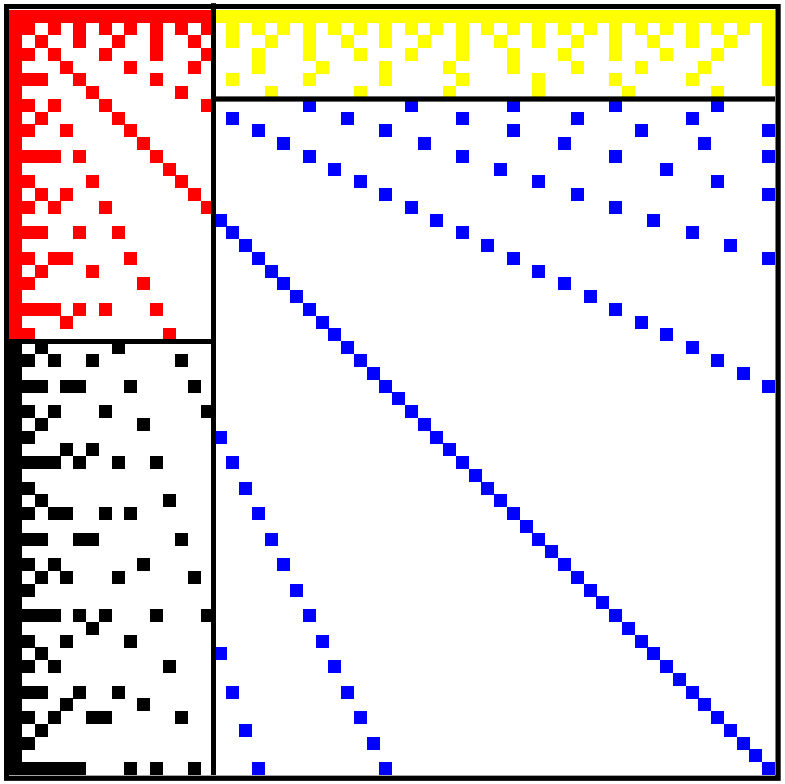
Mondriaan painted here

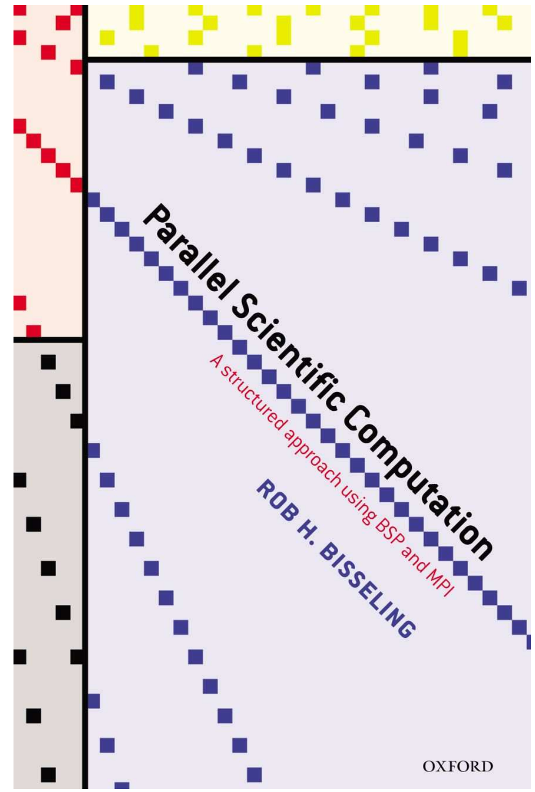
Richard, Erin, Rona, Sarai (Abcoude, NL, 2001)

Mill in Sunlight

Piet Mondriaan 1908

Matrix `prime60`

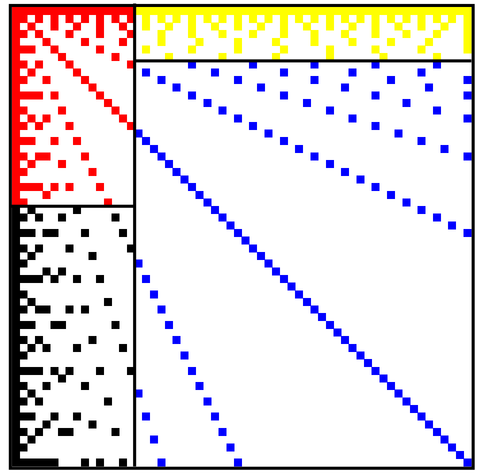




- Block distribution of 60×60 matrix `prime60` with 462 nonzeros, for $p = 4$
- $a_{ij} \neq 0 \Longleftrightarrow i|j \text{ or } j|i \qquad (1 \leq i, j \leq 60)$

Universiteit Utrecht

Communication volume for partitioned matrix



$$V(A_0, A_1, A_2, A_3) = V(A_0, A_1, A_2 \cup A_3) + V(A_2, A_3)$$

Here, $V(A_0, A_1, A_2, A_3)$ is the global matrix–vector communication volume corresponding to the partitioning A_0, A_1, A_2, A_3

Universiteit Utrecht

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**

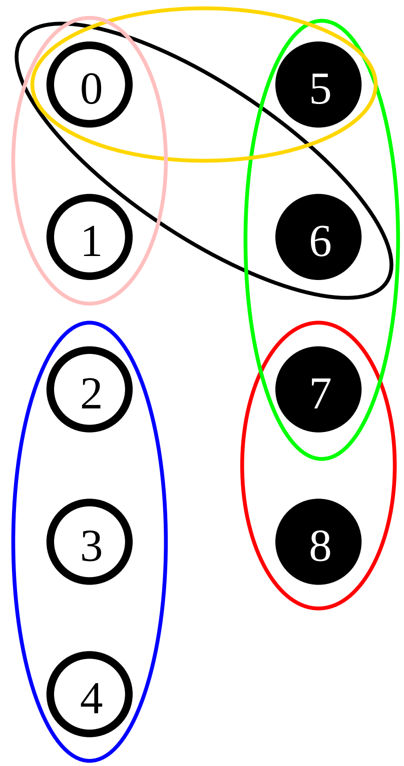
$q := \log_2 p$;
$(A_0^{\mathrm{r}}, A_1^{\mathrm{r}}) := h(A, \mathrm{row}, \epsilon/q)$; hypergraph splitting
$(A_0^{\mathrm{c}}, A_1^{\mathrm{c}}) := h(A, \mathrm{col}, \epsilon/q)$;
if $V(A_0^{\mathrm{r}}, A_1^{\mathrm{r}}) \leq V(A_0^{\mathrm{c}}, A_1^{\mathrm{c}})$ **then**
$(A_0, A_1) := (A_0^{\mathrm{r}}, A_1^{\mathrm{r}})$
else $(A_0, A_1) := (A_0^{\mathrm{c}}, A_1^{\mathrm{c}})$

$maxnz := \frac{nz(A)}{p}(1+\epsilon)$;
$\epsilon_0 := \frac{maxnz}{nz(A_0)} \cdot \frac{p}{2} - 1$; 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;

Universiteit Utrecht

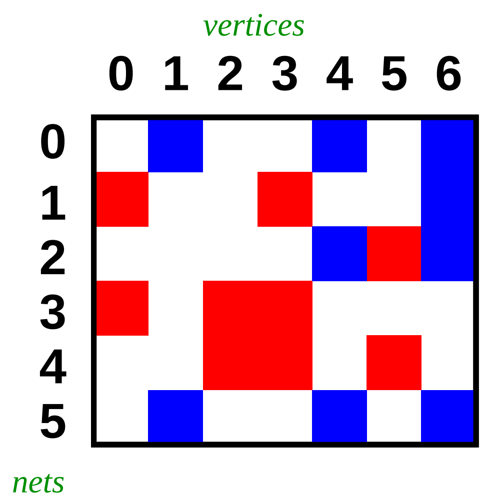
Hypergraph



Hypergraph with 9 vertices and 6 hyperedges (nets), partitioned over 2 processors

Universiteit Utrecht

The h-function

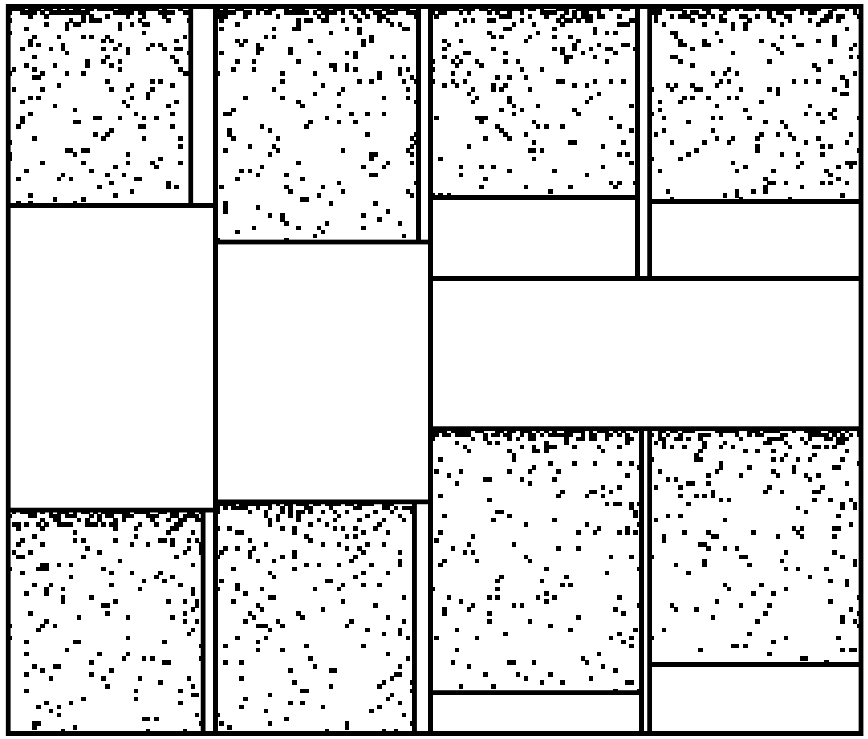


Column bipartitioning of $m \times n$ matrix

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

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

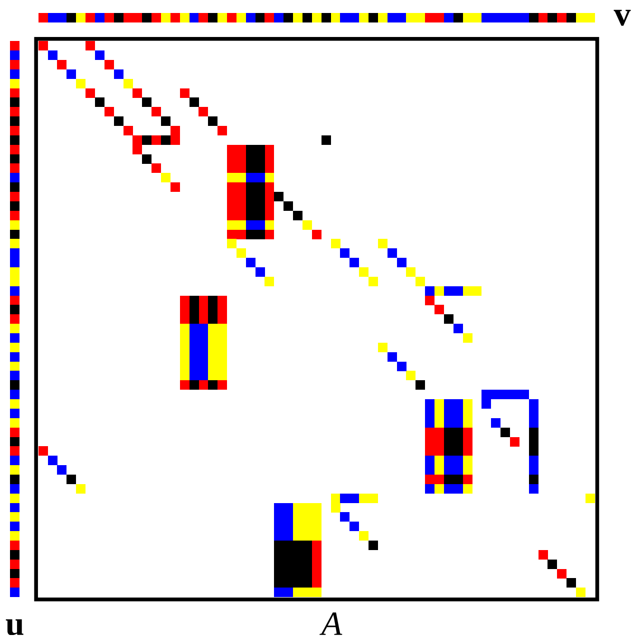
Local view of Mondriaan distribution for 8 processors

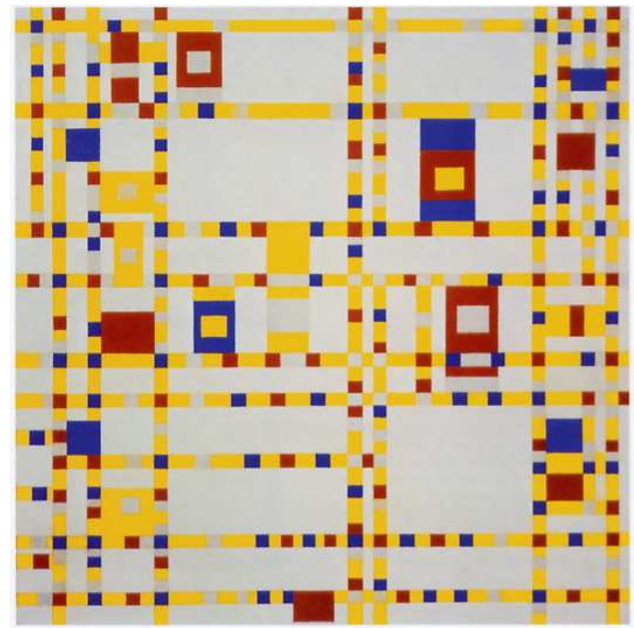


- Imbalance $\epsilon = 3\%$
- First split is vertical
- Empty blocks collect empty row/column parts

Universiteit Utrecht

Vector partitioning





Broadway Boogie Woogie, Piet Mondriaan 1943

- No extra communication if:
  $v_j \mapsto$ one of the owners of a nonzero in matrix column j
  $u_i \mapsto$ owner in matrix row i
- This creates a separate vector partitioning problem.

Universiteit Utrecht

Balance the communication!

Reduce the cost by the bulk synchronous parallel (BSP) model

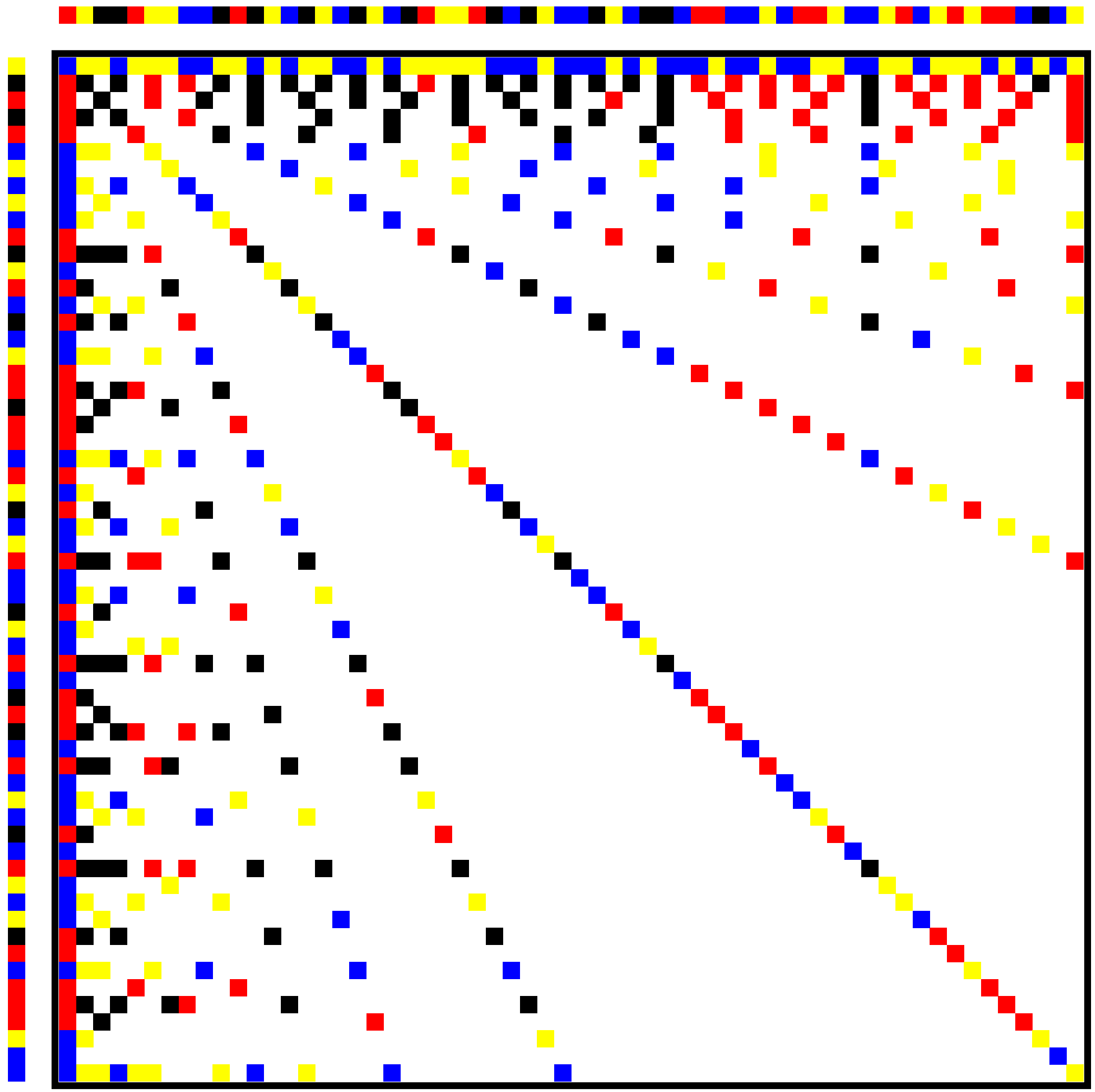
$$\max_{0 \le s < p} h(s),$$

where

$$h(s) = \max(h_{\mathrm{send}}(s), h_{\mathrm{recv}}(s))$$

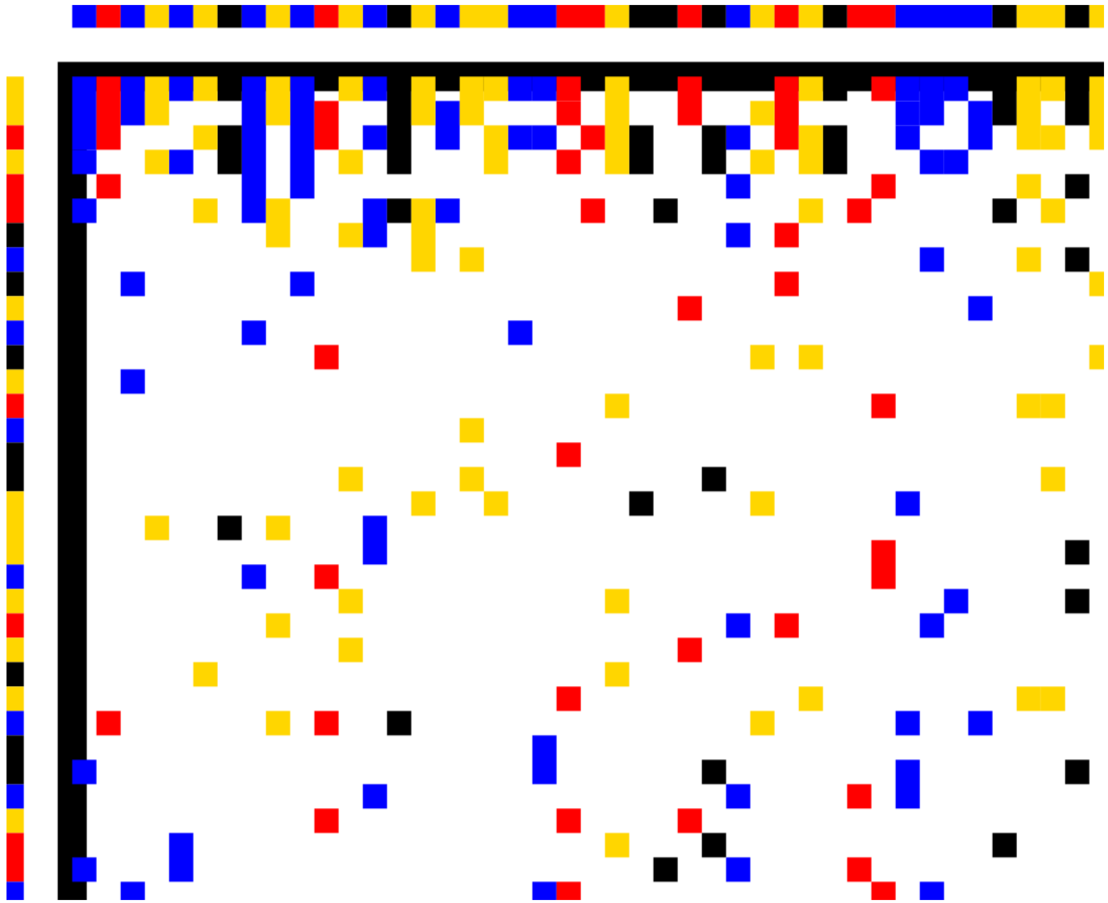
for processor s

Vector partitioning for `prime60`



Universiteit Utrecht

Vector partitioning for `MPQS30`



One-dimensional column partitioning of matrix fixes input vector partitioning. Much freedom for output vector.

Vector partitioning for parallel block Lanczos

- Matrix C, vector X, and vector $Y = CX$ are distributed by Mondriaan.
- C^T multiplication is reverse of C: swap input/output vectors, sends/receives.
- $C^T * C * X$: Output of C = Input of C^T. Hence: independent vector distributions, full freedom for communication balancing.

Vector inner products

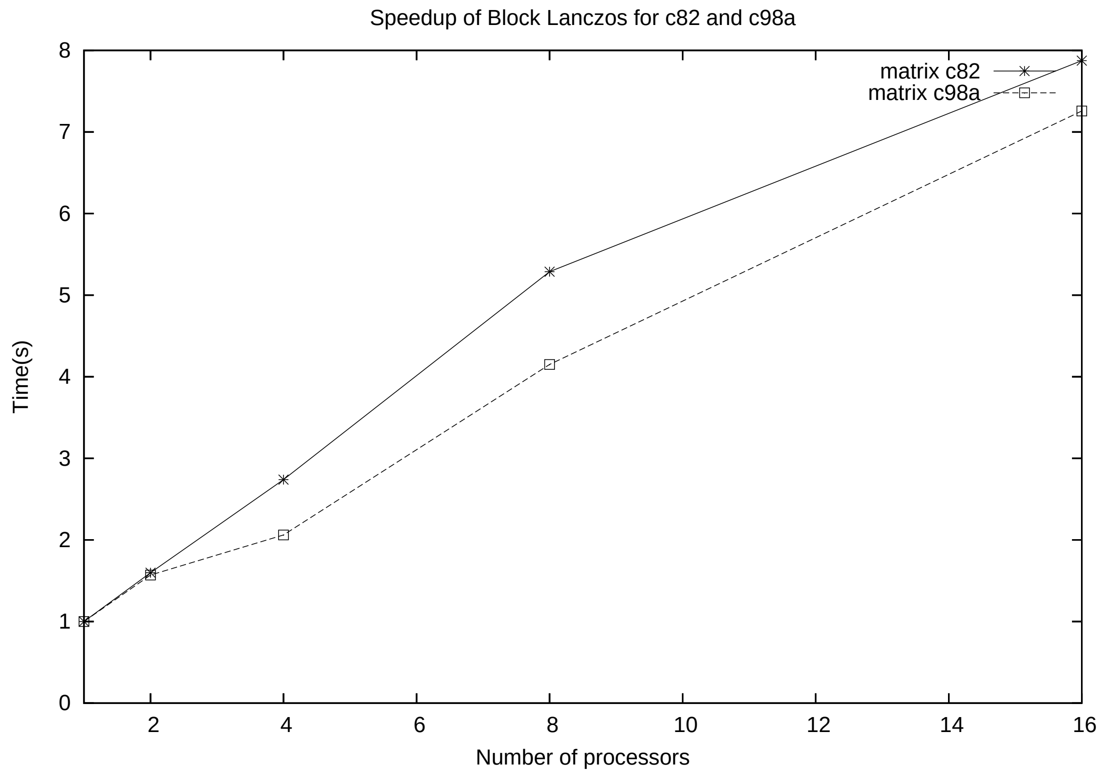
- $V^T * V$ with V an $n_2 \times 32$ bitmatrix, i.e., an integer vector of length n_2.
- Easy if all vectors of the same length are partitioned in the same way.

Global-local indexing mechanism

- Processor owning matrix nonzero a_{ij} knows that it needs vector component x_j, but does not know where it is.
- Processor owning vector component x_j does not know where to send it.
- Solution: use a notice board (or data directory).
- x_j has global index j. Its address (its owner and local index) is first stored at a place that everyone can inspect, in processor $j \bmod p$ at location $j \operatorname{\mathbf{div}} p$.
- Before getting x_j, the owner of a_{ij} obtains its address in a preprocessing step.

Universiteit Utrecht

Experimental results on SGI Origin 3800



Name	n_1	n_2	$nz(C)$
c82	16307	16338	507716
c98a	56243	56274	2075889

Source: Richard Brent

Timings of main algorithm parts for matrix c82

p	Input	Lanczos	PP	Total
1	1.15	78.27	0.47	79.90
2	1.12	48.98	0.25	50.36
4	1.13	28.57	0.15	29.85
8	1.15	14.80	0.08	16.02
16	1.30	9.94	0.07	11.31

- Time (in s) of input phase, block Lanczos algorithm, postprocessing (PP), and total run time.
- Average over three runs.

Timings of main algorithm parts for matrix c98a

p	Input	Lanczos	PP	Total
1	4.1	1186.4	4.0	1194.5
2	4.0	755.8	1.9	761.7
4	3.9	575.5	0.6	580.0
8	4.0	285.8	0.5	290.3
16	4.1	163.5	0.2	167.8

BSP cost for Mondriaan partitioning of c82

p	Comp	Comm	Sync	V/p
1	1015432			
2	522926	$6277g$	l	6277
4	261462	$8078g$	$2l$	7154
8	130730	$7911g$	$2l$	5778
16	65366	$8298g$	$2l$	4296

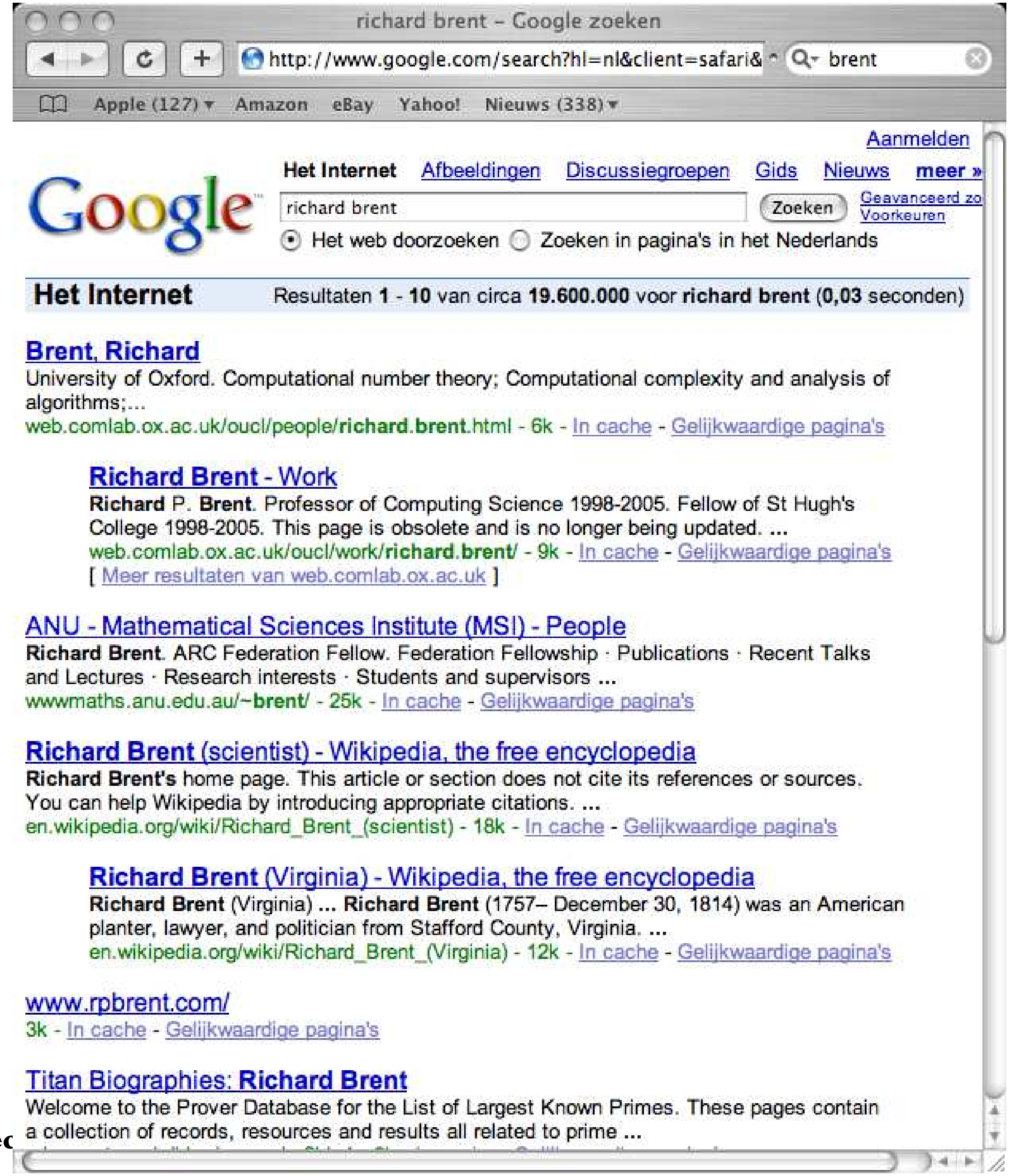
Compare with cost for 2D square partitioning

$$
\begin{aligned}
T_{\mathrm{Matvec}} &\approx \frac{2nz(C)}{p} + \frac{n_1 + n_2}{\sqrt{p}} g + 2l \\
&= 63464 + 8161g + 2l \quad \text{for } \texttt{c82},\ p = 16.
\end{aligned}
$$

Here, g = time for communicating one data word;
l = global synchronisation time

Universiteit Utrecht

Web searching: which page ranks first?



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.

Universiteit Utrecht

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.

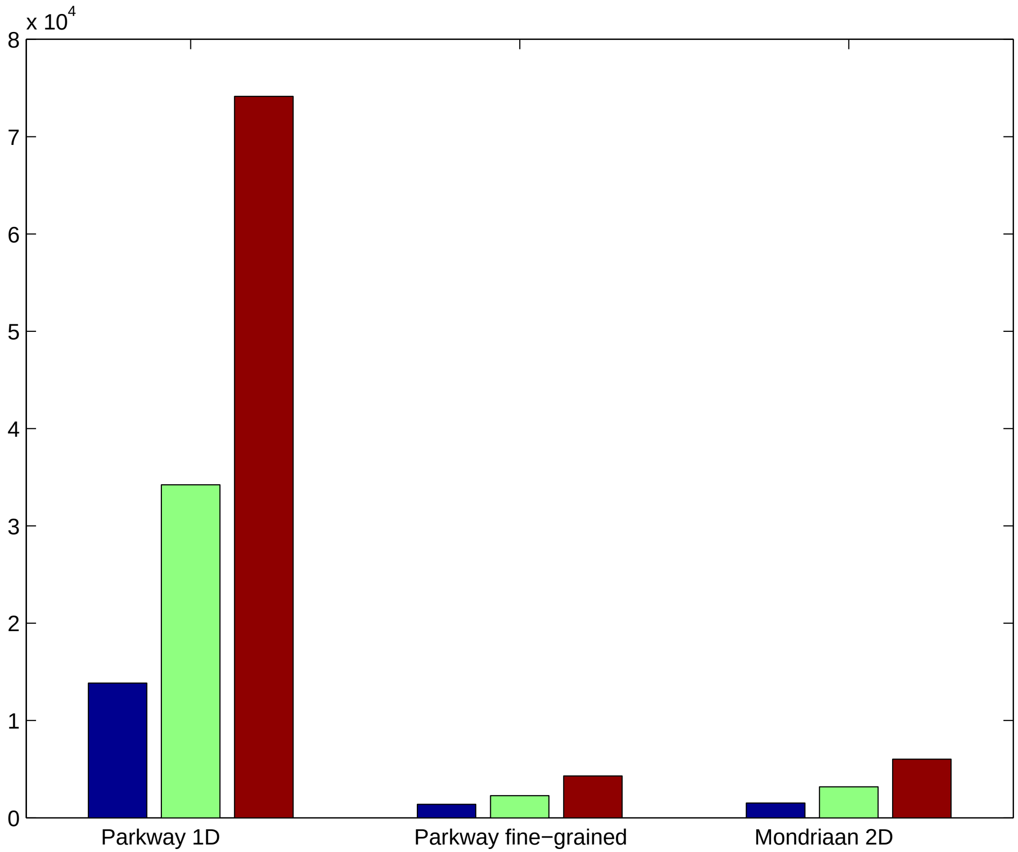
Universiteit Utrecht

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).
- The fine-grain method has been proposed by Çatalyürek and Aykanat in 2001.
- The 2D Mondriaan volumes are results with our recent improvements (to be incorporated in version 2.0), using only row/column partitioning, not the fine-grain option.

Universiteit Utrecht

Communication volume: PageRank matrix Stanford

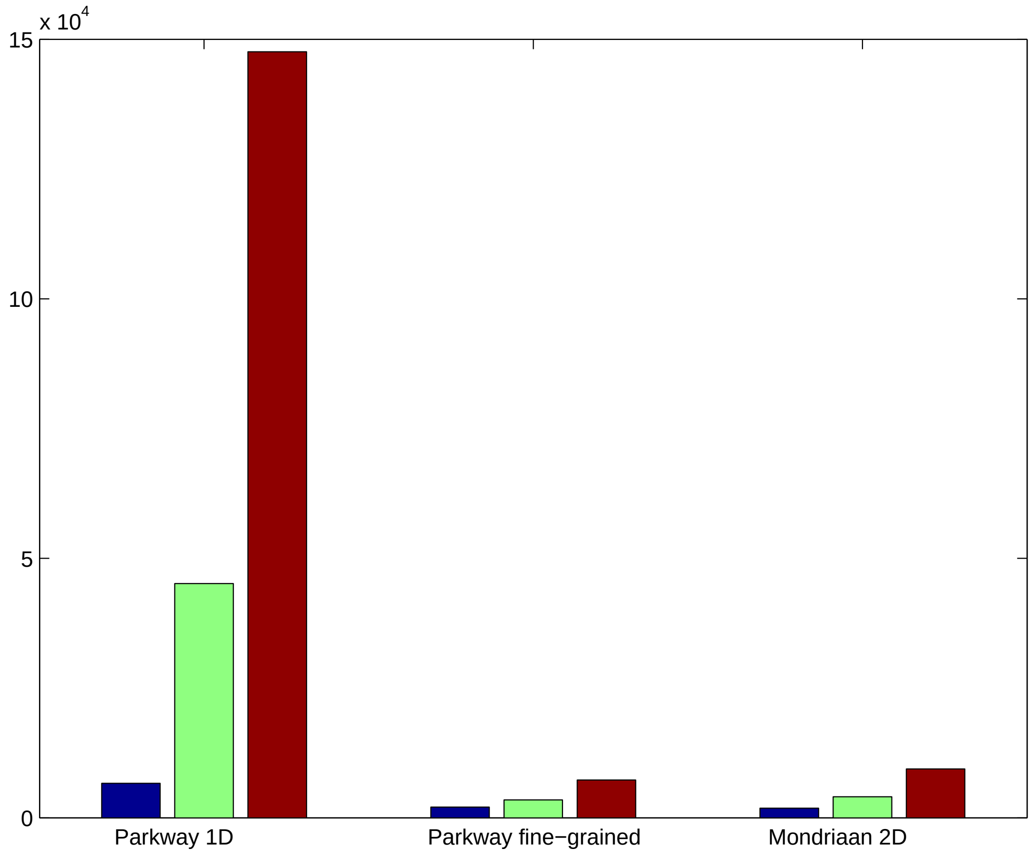


$p = 4, 8, 16$

- $n = 281,903$ (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

Meaning of PageRank results

- Both 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`.

Conclusion

- We have identified 3 main building blocks for parallel integer factorisation:
  - sparse matrix–vector multiplication: most intensive computation
  - sparse matrix partitioning: reduces communication volume
  - vector partitioning: balances communication load
- Integer factorisation matrices remain a challenge for partitioners.
- Partitioning must be two-dimensional, both for integer factorisation and PageRank matrices.

Universiteit Utrecht