Self-improving and exact methods for sparse matrix partitioning

Rob H. Bisseling

Mathematical Institute, Utrecht University

Joint work with Jan-Willem Buurlage (UU), Daniël M. Pelt (CWI Amsterdam), Timon Knigge (UU)

HPC Days in Lyon, April 8, 2016

Universiteit Utrecht

Motivation: solving systems of linear equations

- Given a matrix $A \in \mathbb{R}^{m \times n}$ solve $A\vec{x} = \vec{b}$ for $\vec{b} \in \mathbb{R}^m$.
- Iterative solvers approximate $\vec{x} \in \mathbb{R}^n$ efficiently, by looking only at appropriate subspaces.

Krylov subspace methods

- Let $r_0 = b - A\vec{x}_0$, where $\vec{x}_0$ is an initial guess.
- Iteratively construct the family of Krylov subspaces

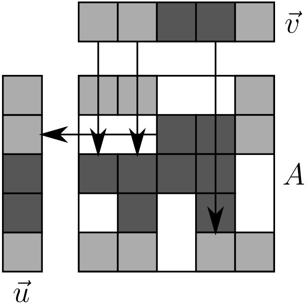
$$\mathcal{K}_k = \text{span}\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}.$$

- From the space $\mathcal{K}_k$, take $\vec{x}_k$ as the kth guess minimising the residual:

$$\vec{x}_k = \text{argmin}_{z \in \mathcal{K}_k} ||\vec{b} - A\vec{z}||.$$

- Terminate when $||\vec{b} - A\vec{x}_k|| < \rho$, where ρ is some tolerance level.

Parallel sparse matrix-vector multiplication



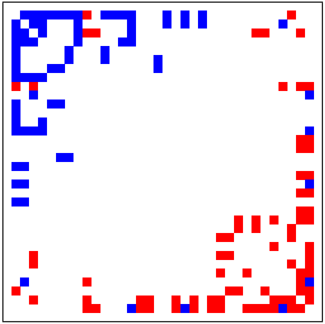
- Parallel multiplication of a 5×5 sparse matrix A and a dense input vector $\vec{v}$,

$$\vec{u} = A\vec{v}$$

- 2D matrix distribution over 2 processors
- $V = 4$ data words of communication
- Perfect load balance: 8 nonzeros per processor

Universiteit Utrecht

Sparse matrix partitioning



34 × 34 matrix `karate`, $nz(A) = 156$ (Zachary's karate club, 1977), $V = 8$

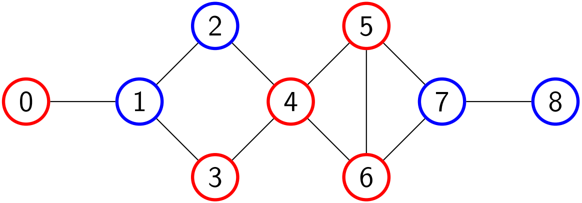
Chicken-and-egg problem

- Getting a good partitioning can be very expensive. Thus, you need to find it in parallel. Therefore, you need a good partitioning.
- We propose the scheme:
  1. Begin with an initial partitioning of reasonable quality.
  2. While running the iterative solver, try to guess the number of iterations still required (based on the convergence behaviour).
  3. If this number is large, spend some time refining the partitioning.
- For our scheme, we require an iterative partitioner. The one we develop is based on label propagation for graphs.

Label propagation on graphs

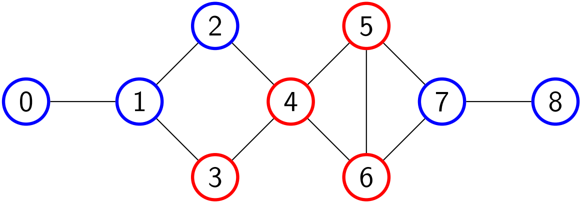
- Goal: Given a graph $G = (V, E)$, obtain a p-way partitioning that minimises the edge-cut (i.e., the number of edges between different parts).
- Here, we describe a simplified version of the PuLP algorithm (Partitioning using Label Propagation) [Slota, Madduri, and Rajamanickam 2014]:
  1. Assign to each $v \in V$ a random label $L(v) \in \{0, \ldots, p-1\}$.
  2. Consider each vertex v in turn, and update to the majority label amongst its neighbours. Ties are broken randomly.

Label propagation example, $p = 2$

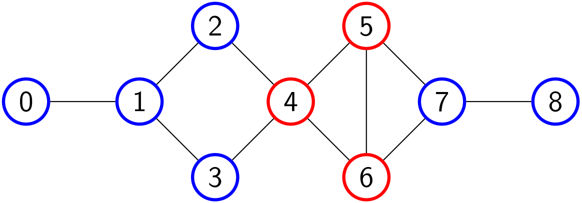


Universiteit Utrecht

Label propagation example, $p = 2$

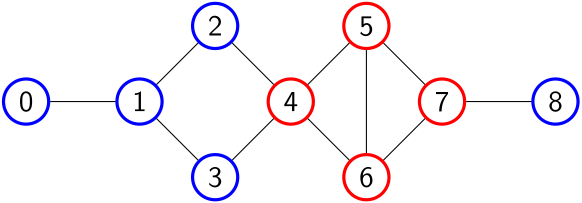


Label propagation example, $p = 2$

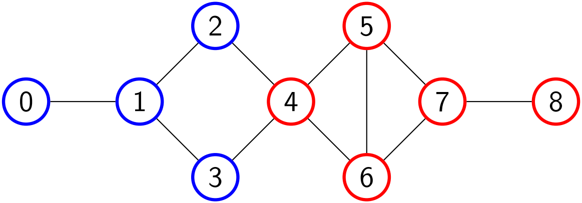


Universiteit Utrecht

Label propagation example, $p = 2$



Label propagation example, $p = 2$



Hypergraph model

- We want to find a p-way partitioning of the matrix A while minimising the communication volume V.
- A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a collection of vertices $\mathcal{V}$, along with a set of nets (or hyperedges) $\mathcal{N}$ such that every $n \in \mathcal{N}$ is a subset of $\mathcal{V}$.
- Consider a hypergraph $\mathcal{H}$ associated to the sparsity pattern of the matrix A, where each vertex represents a matrix column, and each net represents the nonzeros in a matrix row (row-net model).

Label propagation on graphs

- To update the label (part in the partitioning) of a vertex v, we count the labels of its neighbours:

$$C_s(v) = \sum_{(v,u) \in E} \delta(L(u), s), \text{ for } s = 0, \ldots, p-1.$$

Here, $\delta(i,j) = 1$ if $i = j$ and $\delta(i,j) = 0$ otherwise.

- We can give more weight to neighbours with high degree, hoping that vertices of low degree end up at the boundary of a part:

$$C_s(v) = \sum_{(v,u) \in E} \delta(L(u), s) \cdot \deg(u).$$

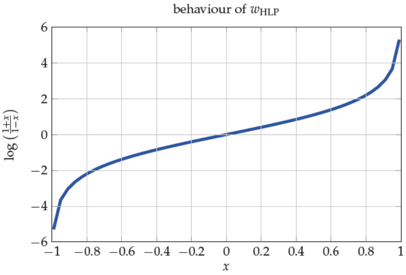
Label propagation on hypergraphs

- The sum will now be over nets instead of edges. Let $\mathcal{N}_v$ be the collection of nets containing v. Then

$$C_s(v) = \sum_{n \in \mathcal{N}_v} w(n, s).$$

- The weight function w should encode two key ideas:
  - We do not want to introduce any new labels to a net, and we should try to eliminate labels with few vertices in the net.
  - When a net is almost pure (single-label) a differently labeled vertex in this net should strongly prefer taking over the majority label.

Quality function



- We will let the weight function depend only on the relative size of part s in net n,

$$x(n, s) = \frac{2|\{v \in n \ : \ L(v) = s\}|}{|n|} - 1.$$

- A function that represents the key ideas is

$$w = \log\left(\frac{1+x}{1-x}\right), \text{ for } x \in (-1, 1).$$

Initial partitioning

- The PuLP algorithm initially constructs parts around vertices with high degree, because this is expected to lead to good partitionings.
- For hypergraphs, we observe that relatively small nets are most easily kept pure. We could therefore ignore larger nets at first.
- We construct a chain of growing hypergraphs:

$$\mathcal{H}_0 \subset \mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset H_M = \mathcal{H}.$$

Here, $\mathcal{H}_i = (\mathcal{V}, \mathcal{N}^i)$, and $\mathcal{N}^i$ holds the smallest 2^i nets.

Partitioner iteration

- Begin with some initial partitioning, e.g. distribute the vertices cyclically, choosing the label $s = v \bmod p$ for vertex $v \in \mathcal{V}$.
- For iteration i with $0 \leq i \leq M$, consider each vertex $v \in \mathcal{V}$ in turn. Choose the label s that maximises $C_s(v)$ in the hypergraph $\mathcal{H}_i$.
- For $i > M$, we have $\mathcal{H}_i = \mathcal{H}$, and we perform this label propagation on the entire hypergraph $\mathcal{H}$.

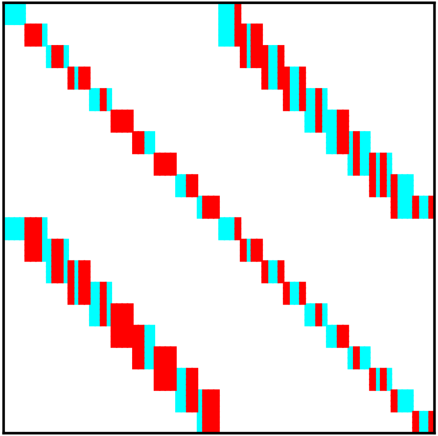
Balancing criterion

- Let N_{it} be the projected number of iterations left, fitted to the norm of the residual as a function of solver iterations.
- Let ΔV be the projected volume decrease, taken as the decrease of the communication volume in the previous partitioner iteration.
- Perform a partitioner iteration if

$$T_{\mathrm{part}} + N_{\mathrm{it}}\, T_{\mathrm{sol}}(V - \Delta V) < N_{\mathrm{it}}\, T_{\mathrm{sol}}(V),$$

where T_{part} is the time of a partitioner iteration and $T_{\mathrm{sol}}(V)$ the time of a solver iteration.

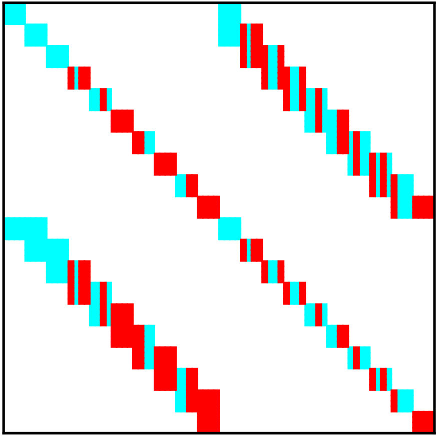
HyperPULP in action for 80 × 80 matrix steam3



$V = 80$

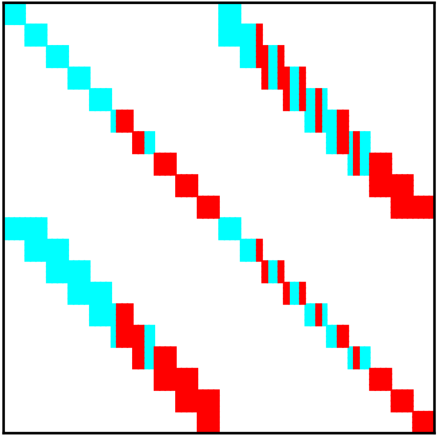
Universiteit Utrecht

HyperPULP in action for 80 × 80 matrix steam3



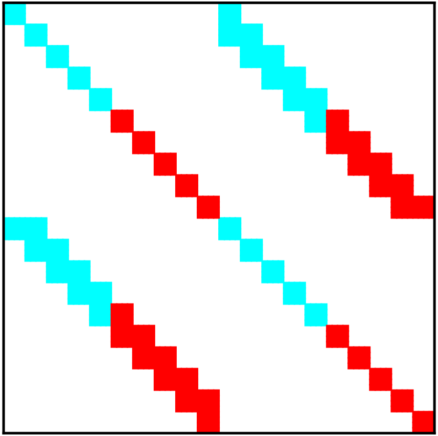
$V = 68$

HyperPULP in action for 80×80 matrix steam3



$V = 52$

HyperPULP in action for 80×80 matrix steam3



$V = 8$

Universiteit Utrecht

Partitioning volume and time vs. SpMV time

Matrix	m	n	nz	V_{HP}	V_C	T_{HP} (in ms)	T_C (in ms)	T_{part} (in ms)
cage7	340	340	3084	223	339	3.72	4.30	5.84
cage8	1015	1015	11003	519	1009	5.15	6.30	30.44
cage9	3534	3534	41594	1806	3512	11.07	14.93	113.96
cage10	11397	11397	150645	5420	11356	30.30	43.41	647.76
cage11	39082	39082	559722	20988	38957	102.82	133.43	2656.46
bcspwr06	1454	1454	5300	597	1242	4.98	6.81	12.89
cdde1	961	961	4681	935	961	5.75	5.92	8.31
bp_800	822	822	4534	467	582	4.84	5.10	10.43
well1033	1033	320	4732	206	274	3.96	4.05	8.44
ex24	2283	2283	48737	979	2283	9.56	12.33	112.93

- $p = 2$ using BSPonMPI on Bull supercomputer
- V_{HP}, V_C = communication volume of HyperPULP, 1D cyclic partitioning
- T_{HP}, T_C = time of 100 sparse matrix–vector multiplications
- T_{part} = partitioning time until local optimum

Universiteit Utrecht

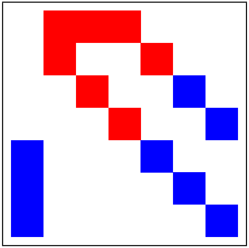
Software: mixing partitioners and linear solvers

- Mixing partitioning and solver iterations requires significant changes to existing software workflows.
- The new framework that was developed, Zee, is an attempt to provide a unified library that uses familiar syntax for common operations.

- Completely open-source and free to use, written by Jan-Willem Buurlage.

Universiteit Utrecht

Optimal bipartitioning



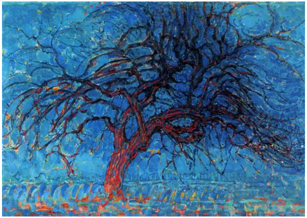
7×7 matrix `b1_ss`, $nz(A) = 15$

- Benchmark $p = 2$ because heuristic partitioners are often based on recursive bipartitioning.
- Problem $p = 2$ is easier to solve than $p > 2$.
- Load balance criterion is

$$nz(A_i) \leq (1+\varepsilon)\left\lceil \frac{nz(A)}{2} \right\rceil, \quad \text{for } i = 0, 1.$$

- Rounding enables a feasible solution even for $\varepsilon = 0$ and odd $nz(A)$.

Branch-and-bound method

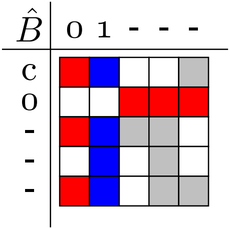


Piet Mondriaan 1908
Evening - the red tree

- Construct a ternary tree representing all possible solutions
- Every node in the tree has 3 branches, representing a choice for a matrix row or column:
  - completely assigned to processor $P(0)$
  - completely assigned to processor $P(1)$
  - cut
- The tree is pruned by using lower bounds on the communication volume or number of nonzeros

Universiteit Utrecht

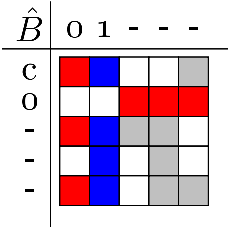
Lower bounds L_1, L_2 on communication volume



- Partial solution: value 0, 1, or c has been assigned to 2 rows and 2 columns
- Row 0 has been cut: lower bound on volume $L_1 = 1$
- Rows 2 and 4 have been implicitly cut: $L_2 = 2$

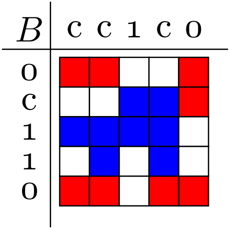
Universiteit Utrecht

Lower bound L_3 on communication volume



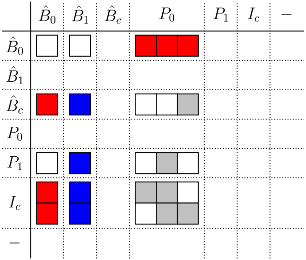
- Columns 2, 3, 4 have been partially assigned to $P(0)$
- They can only be completely assigned to $P(0)$ or cut.
- For perfect load balance ($\varepsilon = 0$), we can assign at most 2 more red nonzeros
- Thus we have to cut column 3, and one more: $L_3 = 2$

Optimal solution



- Optimal solution: volume $= 4$.
- Total lower bound is $LB = L_1 + L_2 + L_3 = 5$.
- Prune partial solution since $LB > UB$.

Lower bound L_4 by conflicting partial assignments



- Permute matrix to create blocks:
  - $\hat{B}_0$: completely assigned to processor $P(0)$
  - P_0: partially assigned to processor $P(0)$
  - $\hat{B}_c$: cut
  - $\hat{I}_c$: implicitly cut
- Conflict for nonzero in row block $P_1 \cap$ column block P_0: $L_4 = 1$

Maximum bipartite graph matching

- Assume row block $P_0 \cap$ column block P_1 contains several nonzeros.
- Define bipartite graph $G = (V_0 \cup V_1, E)$:
  - vertex set V_0 contains the rows of P_0,
  - vertex set V_1 contains the columns of P_1,
  - edge set E containing edges (i, j) for $a_{ij} \neq 0$.
- Compute a maximum matching $M \subseteq E$. Then $L_4 = |M|$, since every nonzero (edge) in the matching causes at least one cut row or column.
- Two nonzeros from the matching cannot be in the same matrix row or column.

Alternative view: minimum vertex cover

- König's theorem (1931): maximum matching in bipartite graph is equivalent to minimum vertex cover (minimum number of vertices needed to cover at least one end point for all edges).
- This gives the minimum number of cut rows or columns.

Dynamic maximum matching

- The conflict graph is small, because we solve small sparse matrix problems and solutions with many conflicts get pruned early.
- Therefore, we maintain a maximum graph matching as the conflict graph changes.
- We prove, as a direct consequence of Berge's theorem (1957):
  - when adding a vertex i with all its edges: sufficient to search for an augmenting path starting at i;
  - when deleting a matched vertex i with all its edges: sufficient to search for an augmenting path starting at the match of i.

Results for 10 largest matrices solved

Matrix	m	n	nz	V_{LB}	V_{MG}	V_{FG}	V_{Opt}	Time (s)
stoch_air	3754	7517	20267	14	14	13	6	0.39
rosen1	520	1544	23794	8	8	24	8	0.03
add32	4960	4960	23884	40	13	13	4	381.29
mhd4800b	4800	4800	27520	3	2	2	2	161.83
Chebyshev3	4101	4101	36879	4	22	15	4	0.07
rosen2	1032	3080	47536	8	8	33	8	0.05
lp_fit2p	3000	13525	50284	25	25	70	21	0.79
rosen10	2056	6152	64192	8	8	26	8	0.10
c-30	5321	5321	65693	1583	43	790	30	6.07
lp_fit2d	25	10524	129042	25	25	27	21	0.76

LB = localbest = best of 1D row, 1D column (v1-v3)

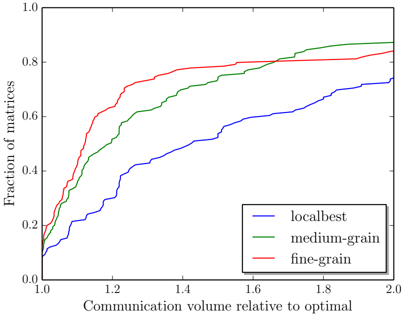
MG = medium-grain method (v4.0)

FG = fine-grain model (Çatalyürek and Aykanat 2001)

Opt = optimal using MondriaanOpt (Mondriaan v4.1, soon)

Universiteit Utrecht

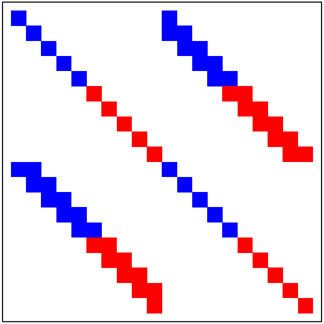
Benchmarking 3 methods vs. optimal partitioning



- 217 matrices from U. Florida collection with $nz \leq 1000$
- 85% were solved to optimality for $\varepsilon = 0.03$
- $V_{Opt} = 0$ excluded from test suite
- Medium-grain method solves 87% of test suite within factor 2 of optimal volume.

Universiteit Utrecht

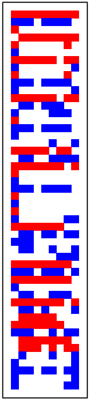
Matrix steam3



- 80×80 matrix steam3, $nz(A) = 928$
- 1D steam model of oil reservoir (Roger Grimes 1983)
- 20 points, 4 degrees of freedom
- $V = 8$, perfect balance

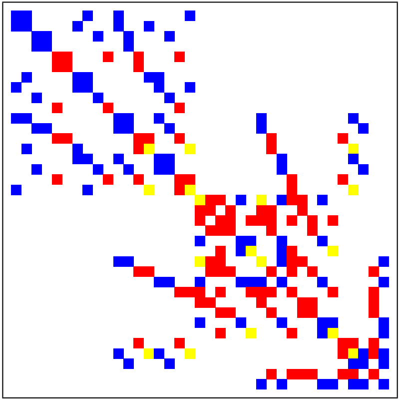
Universiteit Utrecht

Matrix divorce



- 50×9 matrix `divorce`, $nz(A) = 225$
- Divorce laws in the 50 US states
- Row 0 is Alabama, ..., Row 49 is Wyoming
- Column 0 is Incompatibility, Column 1 is Cruelty, ...
- $V = 8$, imbalance $= nz_{\max} - nz_{\min} = 1$

Matrix cage5



- 37×37 matrix cage5, $nz(A) = 233$
- Markov model of DNA electrophoresis, 5 monomers in polymer (Alexander van Heukelum 2003)
- $nz_0 = 106$; $nz_1 = 110$; $nz_{free} = 17$
- $V = 14$, imbalance $= nz_{max} - nz_{min} = 1$

Conclusion

- We have introduced a relatively cheap hypergraph partitioning method that is capable of improving itself over time.
- We minimise the total running time of partitioners and linear solvers by mixing the two operations.
- We also presented an exact branch-and-bound algorithm for computing optimal bipartitionings of small sparse matrices.
- Currently, optimal partitionings have been determined for over 260 matrices.
- Lessons learned from optimal partitioning: the heuristic medium-grain method, the use of volume 0 luck, and the benefit of free nonzeros.

Future work

- Parallelise every component of the self-improving method (linear algebra operations / partitioner / solver / . . . , Jan-Willem Buurlage).
- Expand the MondriaanOpt database: one matrix a day, at `http://www.staff.science.uu.nl/~bisse101/Mondriaan/Opt`
- Further improve the lower bounds (Timon Knigge)
- Spring 2016: release Mondriaan v4.1 software package, including MondriaanOpt v1.0. β version available upon request.

"An exact algorithm for sparse matrix bipartitioning", by Daniël M. Pelt and Rob H. Bisseling, *Journal of Parallel and Distributed Computing* **85** 2015, pp. 79–90.

Thank you!