Experimental results on Cartesius

Section 5.10 of Parallel Scientific Computation, 2nd edition

Rob H. Bisseling

Utrecht University

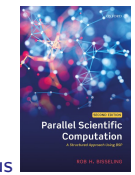
Test set of graphs from SuiteSparse Matrix Collection

Name	n	m	d	Δ	Origin
`tx2010`	914 231	2 228 136	4.9	121	redistricting Texas
`mouse_gene`	45 101	14 461 095	641.3	8 031	gene regulatory network
`cage15`	5 154 859	47 022 346	18.2	46	DNA electrophoresis
`kmer_P1a`	139 353 211	148 914 992	2.1	40	protein k-mer

- Parameters: n = number of vertices, m = number of edges, d = average degree, Δ = maximum degree.
- `tx2010`: V = Texas land areas from the 2010 US Census; E = connections to neighbouring areas; ω = length of the shared border.
- `mouse_gene`: V = probes from a DNA microarray; E = regulatory interactions; ω = mutual information value.
- `cage15`: V = states of a polymer of length 15; E = possible state transitions; ω = probability.
- `kmer_P1a`: V = segments of length k of amino acids; E = overlapping segment pairs; $\omega = 1$.

Partitioning time vs. matching time

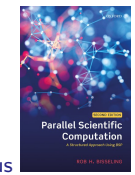
- The vertices of the test graphs were partitioned by a run of the Mondriaan partitioner in 1D row mode for the purpose of a parallel SpMV with $p = 1, 2, 4, \ldots, 1024$.
- Here, the input graph was translated to a matrix by creating the sparse symmetric adjacency matrix A and adding a diagonal I.
- Partitioning a test graph takes much longer than running a matching algorithm on a partitioned test graph. Still, this resembles a likely use case, where the graph is available in a sensible distributed form as part of a larger application.
- In contrast, randomly distributing the vertices would be cheap, but would cut most edges and make the algorithm communication-bound.



Measured execution time (in ms) for graph matching

p	tx2010	mouse_gene	cage15	kmer_P1a
1	208.5	671	2 358	59 057
2	107.0	356	1 237	32 473
4	52.4	322	646	17 618
8	28.1	305	424	9 054
16	15.1	251	194	4 454
32	9.1	317	178	2 756
64	7.5	538	88	911
128	7.9	836	76	561
256	8.3	2 005	77	299
512	15.1	2 526	129	204
1 024	29.3	6 295	218	302

- Experiments performed on p processor cores of the Broadwell subsystem of Cartesius running BSPonMPI.
- The largest speedup achieved, compared to the parallel program with $p = 1$, is $S_{512} = 289$ for kmer_P1a. The smallest speedup is $S_{16} = 2.7$ for mouse_gene.

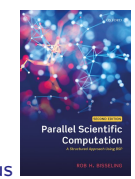


Upper bound on the number of matches

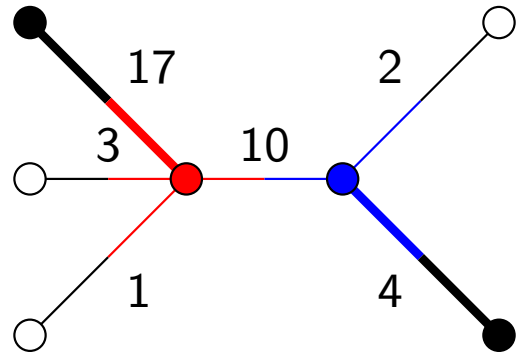
- A trivial upper bound on the number of matches is

$$|\mathcal{M}| \leq \left\lfloor \frac{|\mathcal{V}|}{2} \right\rfloor,$$

because every match involves 2 vertices.



Upper bound on the total matching weight

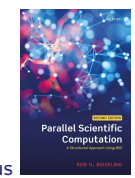


$$\omega(\mathcal{M}) = 17 + 4 = 21 \leq \tfrac{1}{2}(17 + 3 + 1 + 17 + 10 + 2 + 4) = 27$$

- An upper bound on the total matching weight is

$$\omega(\mathcal{M}) \leq \frac{1}{2} \sum_{v \in \mathcal{V}} \max \ \{\omega(u, v) \ : \ (u, v) \in \mathcal{E}\},$$

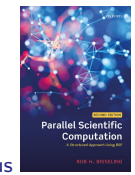
because every vertex v contributes at most the weight of one half-edge to the total matching weight, and this weight is at most half the weight of its heaviest edge.



Number of matches and total matching weight

	tx2010	mouse_gene	cage15	kmer_P1a
Matches	375 342	18 273	2 575 446	59 735 594
Matches upper bound	457 115	22 550	2 577 429	69 676 605
Weight	28 933 021 703	1 287.998	76 890.186	59 735 594
Weight upper bound	39 547 303 682	1 553.424	77 709.076	69 676 605

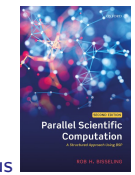
- Given are the total number of matches $|\mathcal{M}|$ and the total matching weight $\omega(\mathcal{M})$, together with their upper bounds, for $p = 1$.
- The preference for a local match in tie-breaking causes a variation of $\leq 0.1\%$ for varying p.
- The matching weight is between 73.2% of the upper bound (tx2010) and 98.9% (cage15), so that the 50% guarantee of the 1/2-approximation is more than satisfied.
- Compared to the (unknown) maximum matching weight, the percentages will even be better.



Number of operations performed

	p	tx2010	mouse_gene	cage15	kmer_P1a
Operations lower bound		8.9	58	188	596
Operations	1	9.5	59	193	873
	32	9.5	148	206	872
	1 024	9.8	168	225	872

- The number of operations (in millions) performed was obtained by summing the sizes of the ranges encountered.
- The lower bound given is $4m$, the cost of partially sorting only the upper parts of adjacency lists.
- The operation counts for $p = 1$ fit well with their lower bound, meaning that in practice the partial sort leads to a linear-time sequential algorithm.
- The number of operations grows a bit with p, because operations are performed on the basis of increasingly incomplete information, e.g., about a new suitor for a halo vertex.

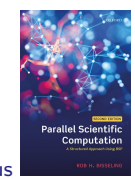


Number of supersteps

	p	tx2010	mouse_gene	cage15	kmer_P1a
Supersteps	2	8	957	55	26
	32	10	1 613	69	67
	1 024	13	2 099	82	72
Parallel depth		146	309	141	145

- The number of supersteps needed for parallel matching grows with p, again because of increasingly incomplete information.
- The parallel depth of an algorithm is the length of its critical path.
- An theorem by Ferdous *et al.* states that the parallel depth for matching on a graph with uniformly random edge weights is $\mathcal{O}((\log_2 m) \log_2 \Delta)$. Note: our test graphs are not random.
- The parallel depth given is $(\log_2 m) \log_2 \Delta$, which we take as an asymptotic lower bound on the number of supersteps for $p = \infty$ in the random case.

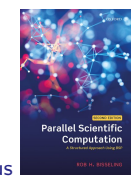
S. M. Ferdous, A. Khan, and A. Pothen, In: Proceedings IPDPS 2018, IEEE, pp. 22–33.



Load balancing for cage15 and $p = 8$

Max operations ($\times 10^3$)	Time (in ms)	Supersteps
4	845	6 618
8	657	3 340
16	555	1 694
32	500	875
64	465	465
128	445	261
256	429	159
512	417	108
1 024	400	83
2 048	377	71
4 096	352	65
8 192	346	62
16 384	341	60
∞	341	60

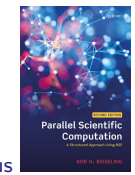
- We can try to balance the computational work load by imposing a maximum number of operations carried out by a processor in a superstep.



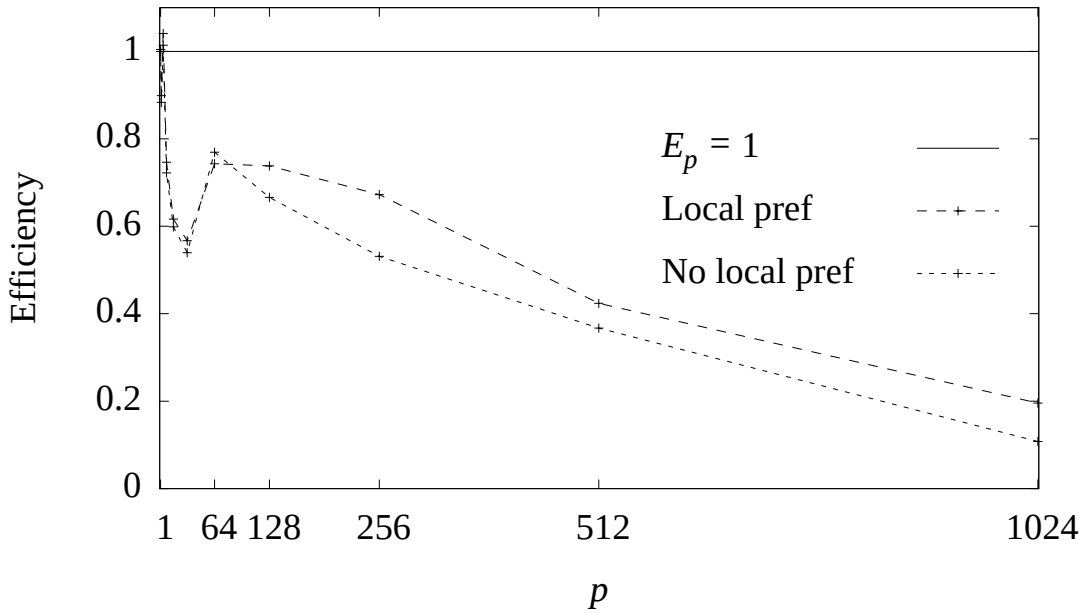
Load balancing for cage15 and $p = 8$

Max operations ($\times 10^3$)	Time (in ms)	Supersteps
4	845	6 618
...	...	...
1 024	400	83
2 048	377	71
4 096	352	65
8 192	346	62
16 384	341	60
∞	341	60

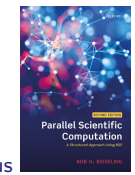
- The total synchronization time for 60 supersteps based on the benchmark value of l is only 2.2 ms, so that the cost of the synchronizations themselves is insignificant.
- Still, we do not observe any gain from the load balancing procedure, which indicates that our work counters are not accurate enough: communication operations were not taken into account, and, how sobering a thought, perhaps not all $\mathcal{O}(1)$-operations are created equal.



Local preferences are beneficial



- Shown is the eficiency (relative to $p = 1$) for the graph `kmer_P1a`, with and without tie-breaking by preferring local matches.
- This graph has unit weights, so that all weight comparisons are ties.
- For $p = 1\,024$, the efficiency is 19.6% with local preferences, and 10.7% without.



Summary

- Partitioning a test graph takes much longer than running a matching algorithm. Still, parallel matching is useful as part of a larger application.
- An upper bound on the total matching weight is

$$\omega(\mathcal{M}) \leq \frac{1}{2} \sum_{v \in \mathcal{V}} \max \ \{\omega(u,v) \ : \ (u,v) \in \mathcal{E}\}.$$

- In practice, the matching weight achieved by the parallel 1/2-approximation algorithm is much higher than the guarantee of 50% and the total operation count is linear in the number of edges.
- The load balancing procedure based on imposing a maximum number of operations should be improved.
- There is always further work to do. Fortunately!