Parallel Algorithm Design via Approximation

Alex Pothen and Arif Khan

Computer Science, Purdue University

Thanks: NSF, DOE, Intel

CERFACS, Sep 7, 2017

Thanks, Iain!

Structure of Talk

- Paradigms for designing parallel algorithms
  - Approximation Algorithms
- Edge covers and Matchings
  - The trouble with exact algorithms
  - 3/2-approximate *b*-Edge cover
    Greedy; Locally Subdominant edge algorithms
  - 2-approximate *b*-Edge cover
    Locally Subdominant edge with no weight update algorithm
    Complement of a *b*-Matching
  - Parallel depth for matching algorithms

Paradigms for Designing Parallel Graph Algorithms

Task and data partitioning
Coloring a task graph to identify independent tasks
Pipelining
Balanced trees

Approximation Algorithms

Paradigms for Designing Parallel Graph Algorithms

Task and data partitioning
Coloring a task graph to identify independent tasks
Pipelining
Balanced trees

Approximation Algorithms

Example: Maximum Vertex-Weighted Matching

Graph	Vertices	Edges	Exact time	Exact weight	1/2-Approx. time	1/2-Approx. % opt. wt./
IG5-16	37K	588K	10 s	1.4 e4	1.6e-2 s	98.7 %
Image-interp	360K	712K	1.2 s	1.5 e8	3.5e-2 s	96.5 %
LargeRegFile	2.9M	4.9M	6.9 s	9.7 e8	0.2 s	98.9 %
Rucci1	2.1M	7.8M	4 h 36 m	1.6 e8	1.3 s	99.7 %
GL7d16	1.5M	14.5M	9 h 50 m	5.8 e8	1.3 s	94.5 %
GL7d20	3.3M	29.9M	$>$ 100 h	NA	4.8 s	NA
GL7d18	3.5M	35.6M	$>$ 100 h	NA	5.5 s	NA
GL7d19	3.9M	37.3M	$>$ 100 h	NA	6.3 s	NA

*Ahmed Al-Herz (CS, Purdue)

Approximation Algorithms for Parallelism

Exact algorithms are often too slow for large graphs. Little concurrency.

Approximation algorithms have near-linear time complexity; orders of magnitude faster than exact algorithms on serial computers. Solutions nearly optimal in practice ($>$ 90% of the optimal weights)

However, many approximation algorithms have little concurrency. They need to be modified or new algorithms needed for concurrency.

Approximation Algorithms for Parallelism

Exact algorithms are often too slow for large graphs. Little concurrency.

Approximation algorithms have near-linear time complexity; orders of magnitude faster than exact algorithms on serial computers. Solutions nearly optimal in practice ($>$ 90% of the optimal weights)

However, many approximation algorithms have little concurrency. They need to be modified or new algorithms needed for concurrency.

Approximation Algorithms for Parallelism

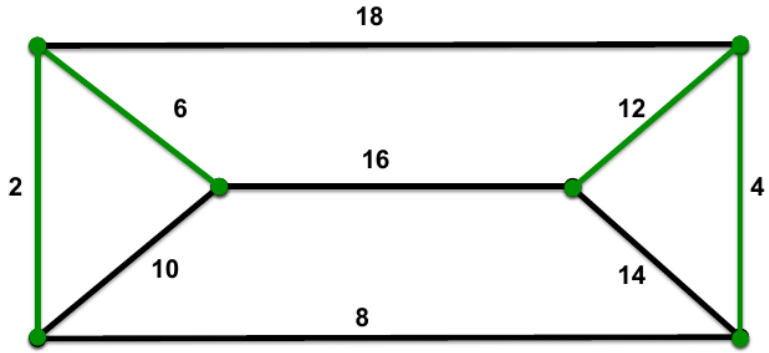
Exact algorithms are often too slow for large graphs. Little concurrency.

Approximation algorithms have near-linear time complexity; orders of magnitude faster than exact algorithms on serial computers. Solutions nearly optimal in practice ($>$ 90% of the optimal weights)

However, many approximation algorithms have little concurrency. They need to be modified or new algorithms needed for concurrency.

b-Edge Cover

Given a function $b(v)$, a set of edges with at least $b(v)$ edges incident on each vertex v.



Here $b(v) = 1$.
Minimum cardinality, edge-weight, etc.
Exact algorithm in polynomial time, but slow, and little concurrency.

Approximation Algorithms

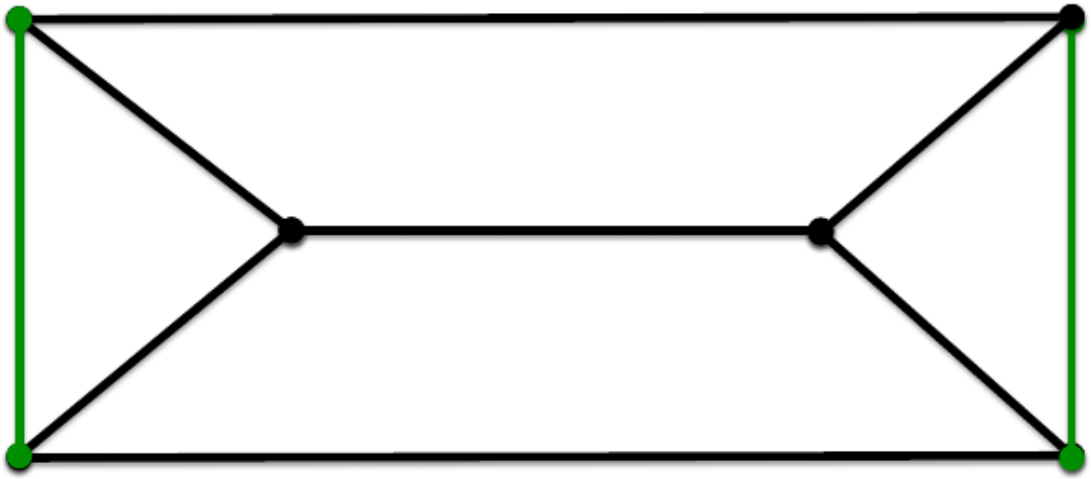
- Greedy algorithm
  - Effective weight of an edge: its weight divided by the number of its endpoints for which $b(v)$ edges have not been added to the edge cover yet.
  - add edge (u, v) of least effective weight to cover
  - update $b(u)$, $b(v)$ values
  - update the effective weights of neighboring edges if needed
  - Repeat until all $b(w)$ values are satisfied.

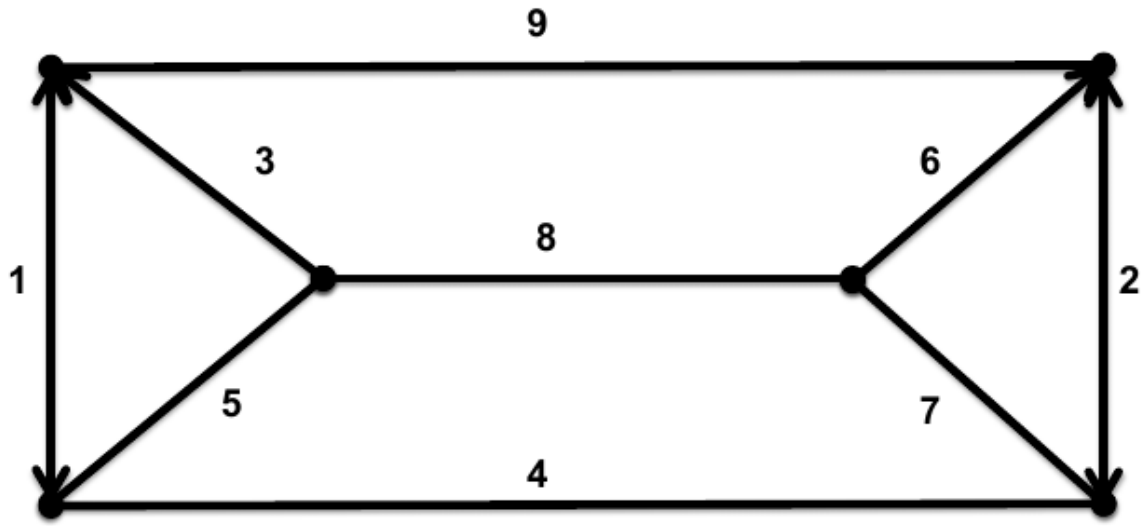
Approximation Algorithm

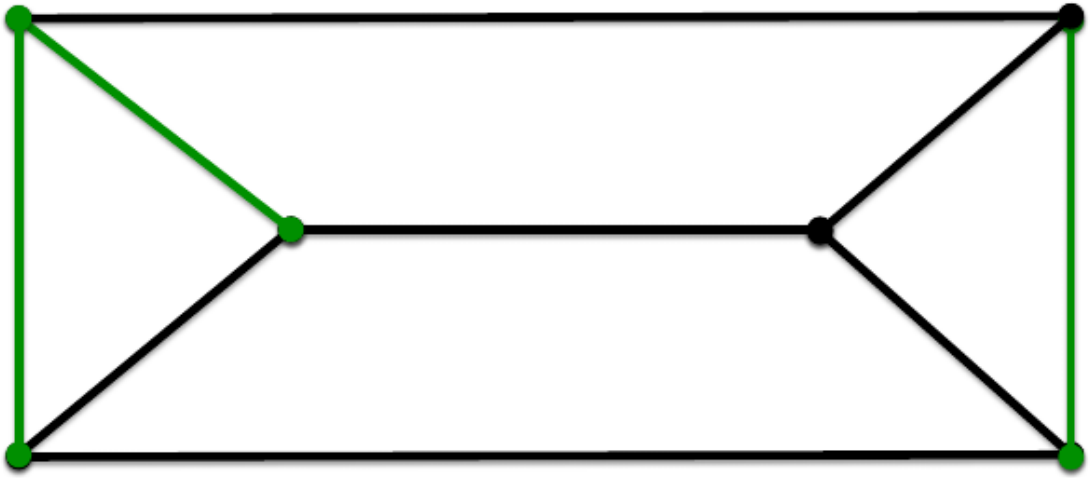
- Greedy algorithm is $3/2$-approximate.
- Effective weights change during the algorithm (unlike matching).
- Other approximation algorithms (e.g., Hall and Hochbaum, $O(\Delta)$ approximation ratio, max degree).
- Not much concurrency.

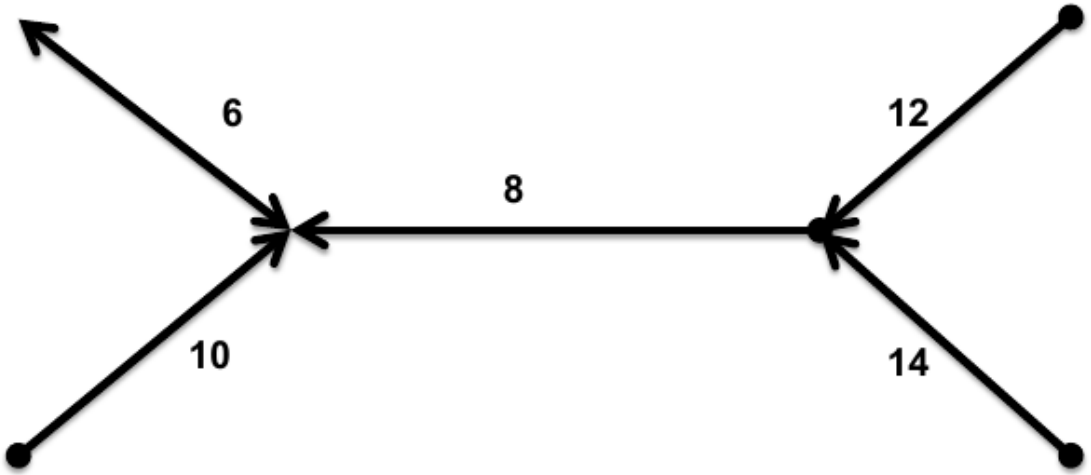
Locally Subdominant Edge Algorithm (LSE)

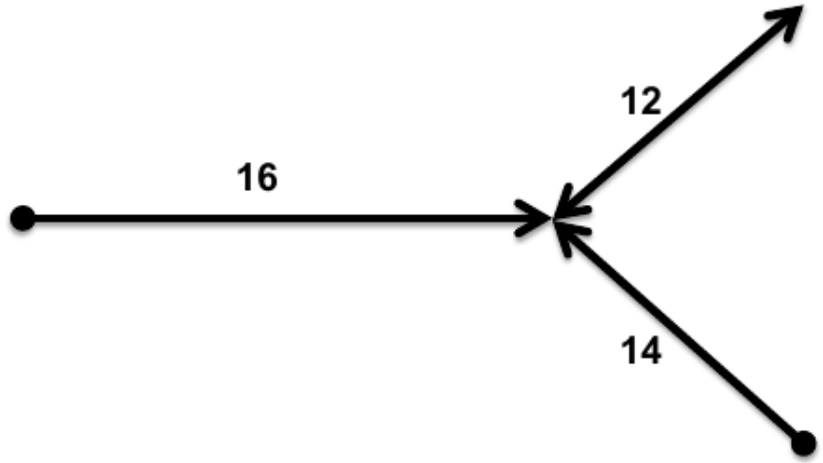
- Locally Subdominant edge: an edge with minimum effective weight among neighboring edges.
  Add LS edges to the edge cover, update effective weights, and repeat.
- Also 3/2-approximation, but with more concurrency than the Greedy algorithm. It computes the same edge cover as Greedy.




9
3
6
8
1
2
5
7
4

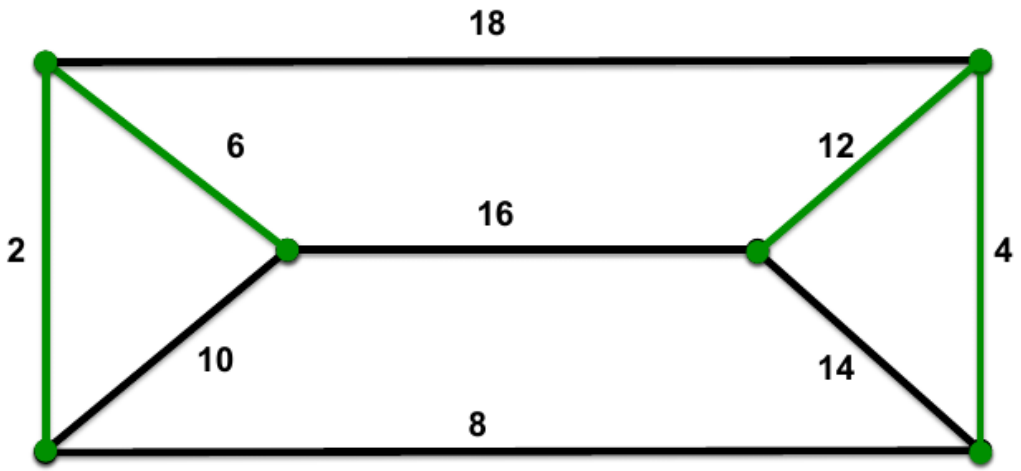



6
12
8
10
14


12
16
14

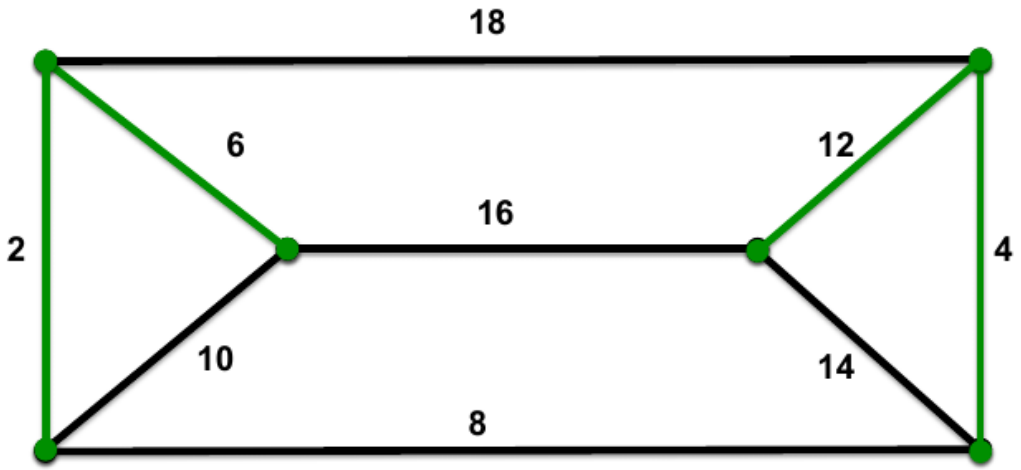
Edge cover from a Matching

Given a b-Edge cover problem, define $b'(v) = \deg(v) - b(v)$ for each vertex v. Compute a maximum weight $b'(v)$-Matching; its complement is a minimum weight $b(v)$-Edge cover.



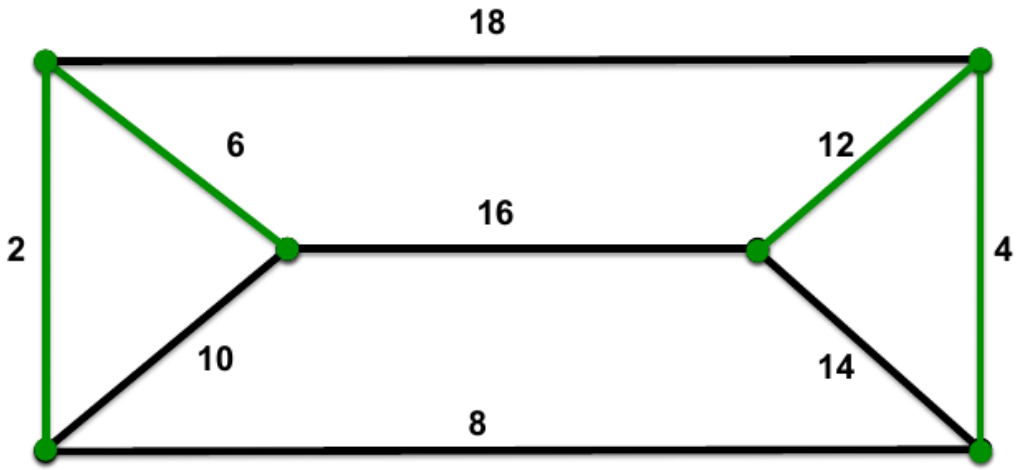
Weighted b-Matching

Given a graph $G = (V, E, W)$, and a function $b(v)$ for each vertex v, a maximum edge-weighted b-Matching is a subset of edges M such that at most $b(v)$ edges in M are incident on a vertex v, and the weight of the edges in M is maximized.



Weighted b-Matching

Given a graph $G = (V, E, W)$, and a function $b(v)$ for each vertex v, a maximum edge-weighted b-Matching is a subset of edges M such that at most $b(v)$ edges in M are incident on a vertex v, and the weight of the edges in M is maximized.



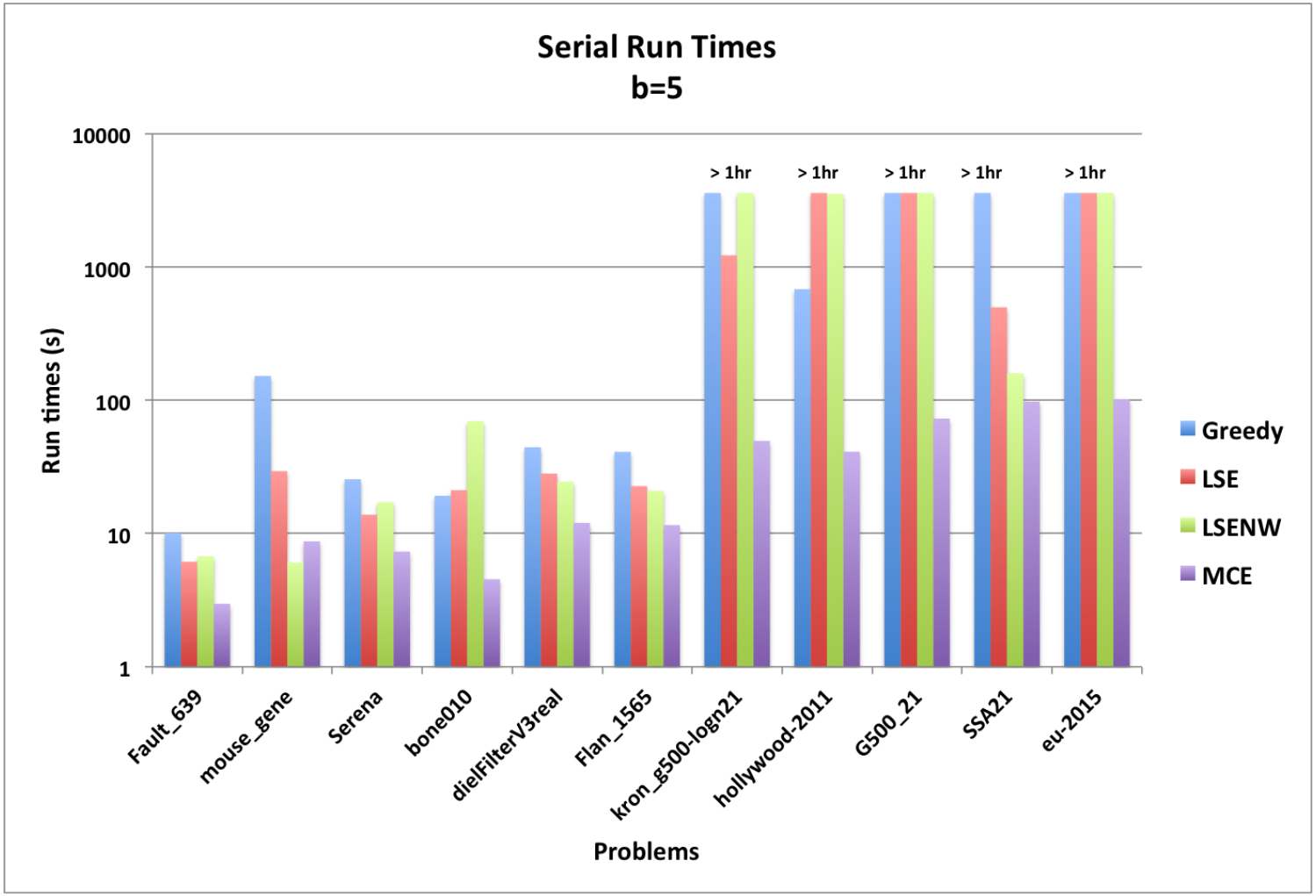
Edge Cover from Matching: Approximations

- How about approximation algorithms? Compute a $1/2$-approximate $b'(v)$-Matching by Greedy or Locally Dominant edge (or b-Suitor) algorithm; its complement is a 2-approximate $b(v)$-Edge cover.
- This MCE (Matching Complement Edge Cover) algorithm has more concurrency than other approximation algorithms for b-Edge cover.

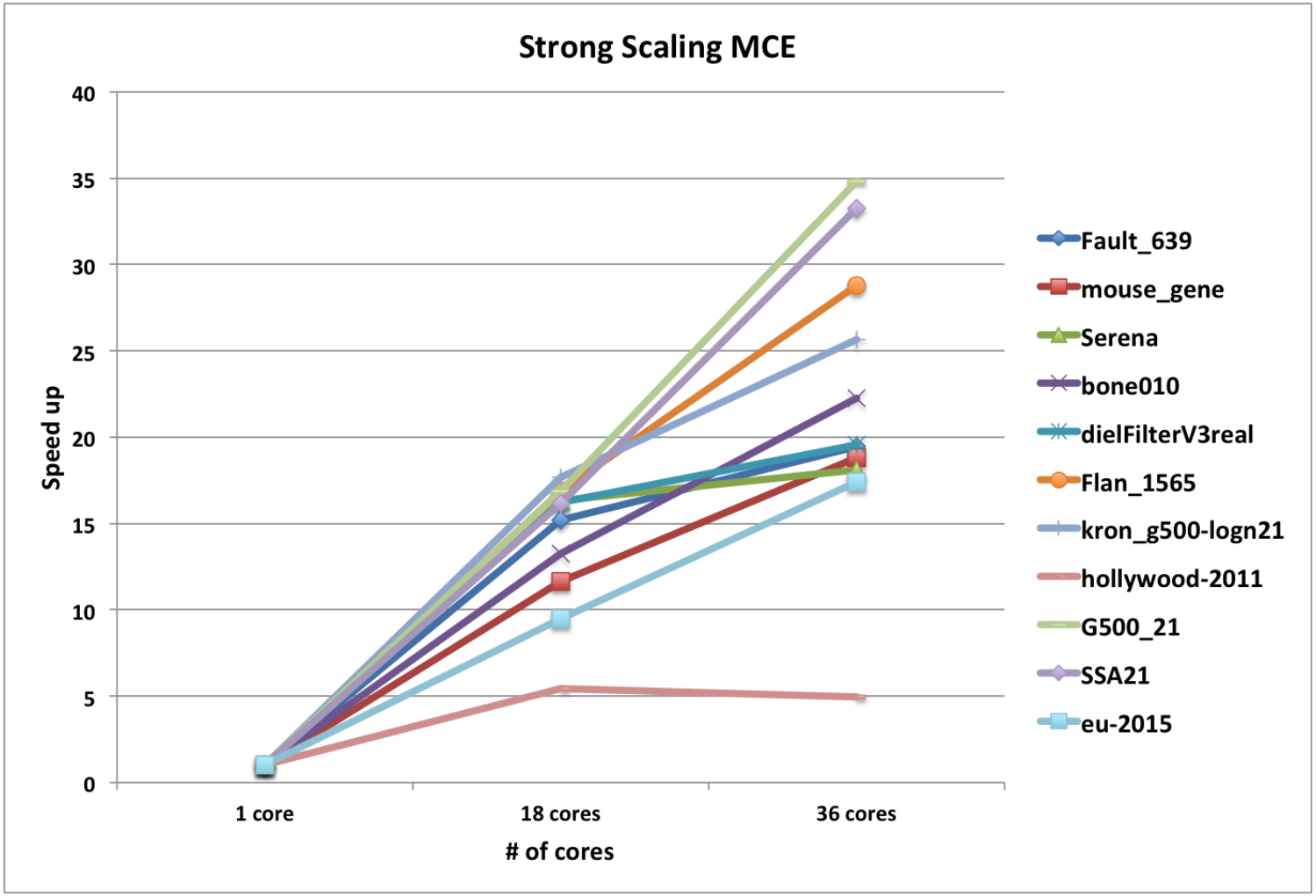
Difference in Weights: 3/2 to 2-Approximation

Problems	**b=1**	**b=5**
Fault_639	3.56%	1.13%
mouse_gene	12.12%	6.55%
Serena	4.65%	1.51%
bone010	2.00%	0.96%
dielFilterV3real	1.88%	0.11%
Flan_1565	9.33%	4.41%
kron_g500-logn21	***16.42%***	***13.53%***
hollywood-2011	5.52%	1.74%
G500_21	8.88%	3.26%
SSA21	12.30%	4.89%
eu-2015	6.78%	2.33%
Geo. Mean	**6.15%**	**2.14%**

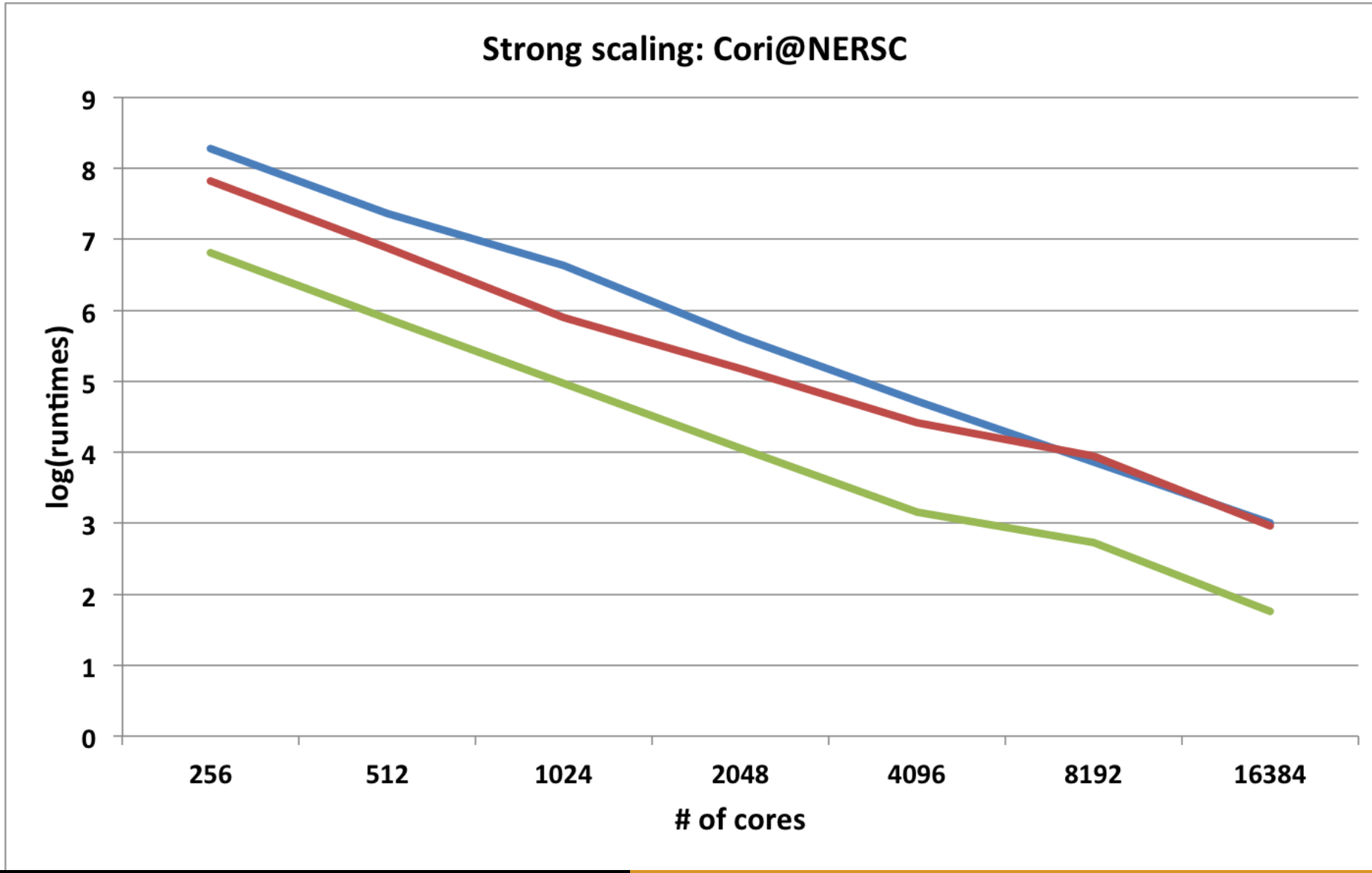
Serial Performance of MCE algorithm



Parallel Performance of MCE algorithm



Distributed-Memory Parallel *b*-Matching



Adaptive Anonymity

Users	*Features*					
	F_1	F_2	F_3	F_4	F_5	F_6
$U1$	1	0	1	0	1	0
$U2$	1	1	1	1	1	0
$U3$	0	1	0	1	0	1
$U4$	0	0	0	0	0	1
$U5$	1	1	0	0	0	0
$U6$	1	1	0	0	0	1

Adaptive Anonymity

Users	*Features*					
	F_1	F_2	F_3	F_4	F_5	F_6
$U1$	1	0	1	0	1	0
$U2$	1	1	1	1	1	0
$U3$	0	1	0	1	0	1
$U4$	0	0	0	0	0	1
$U5$	1	1	0	0	0	0
$U6$	1	1	0	0	0	1
$U1$	1	$*$	1	$*$	1	0
$U2$	1	$*$	1	$*$	1	0
$U3$	0	$*$	0	$*$	0	1
$U4$	0	$*$	0	$*$	0	1
$U5$	1	1	0	0	0	$*$
$U6$	1	1	0	0	0	$*$

Adaptive Anonymity Algorithm

$X \in \mathbb{R}^{n \times f}$ is the instance-feature matrix;
Initialize weight matrix $W \in \mathbb{R}^{n \times f}$ with all one's;

for $i = 1$ to niter
Calculate a weighted graph G from W and X
Compute a min weight b-edge cover C in G
Recalculate weight matrix W using C
if convergence criterion is met **break**; **endif**
endfor

The edge cover groups each instance v with $\geq b(v)$ others

Adaptive Anonymity with Approximation Algorithms

Prob.	*Inst.*	*Feat.*	b−*Matching*		b−*EdgeCover*	
			Time	*Util.*	*Time*	*Util.*
Caltech	768	101	0.9	94.5	6.6	93.7
Reed	7962	139	1.5	95.6	229	95.0
Haverford	1,446	145	3.0	97.2	34	96.6
*Simmons*81	1,518	140	3.2	96.7	62	96.0
UCIAdult	32.6*K*	101	26	97		
*Census*1990	158*K*	68	9*m*	90		
PokerHands	500*K*	95	2*h*17*m*	84		
CMS	1*M*	512	10*h*33*m*	81		

Two orders faster than exact *b*-Matching.

Depth of Parallel Algorithms

Depth of a parallel algorithm

maximum number of time steps in the algorithm.

We conjecture that when the edge weights are chosen uniformly at random, the depth of the Locally Dominant edge algorithm for $1/2$-approximate weighted 1-matching is $O(\log^2 m)$ with high probability, where m is the number of edges.

Work in Parallel Algorithms

Work in a parallel algorithm

the total number of operations in the algorithm, which is the product of the number of processors and the depth.

We conjecture that when the edge weights are chosen uniformly at random, the work in Locally Dominant edge algorithm for $1/2$-approximate weighted 1-matching is $O(m)$ with high probability, where m is the number of edges.
This requires choosing subsets of edges to work with at each step.

Open Problems

Can we post-process a 2-approximate algorithm to obtain $3/2$-approximation, while maintaining good concurrency?

What can be extended to set multicover problems?

One application of edge covers is to anonymity problems. Others to graph construction, sparsification, visualization, etc.

Recent Work: Matching

- Arif Khan, Scalable Approximation Algorithms for *b*-Matching and *b*-Edge cover, PhD thesis, Purdue, July 2017.
- Khan, P: A new 3/2-approximation algorithm for *b*-Edge cover, Proc. SIAM CSC 2016.
- Khan, P, Patwary, Manne, Halappanavar, et al: Approximation algorithms for weighted b-Matching, SIAM J. Scientific Computing, 2016.
- Khan, Pothen, Patwary et al, Scalable *b*-Matching Algorithms on Distributed Memory Processors, Supercomputing, 2016.

Recent Work: Matching

- Halappanavar, P, Azad, Langguth, Manne, Khan: Codesign of matching algorithms and multicore machines, IEEE Computer, August 2015.
- Dobrian, Halappanavar, P, Al-Herz: 2/3-Approximation Algorithms for Vertex-weighted Matchings in Bipartite Graphs, Submitted to SISC, 2017.
- Azad, Buluc, P: Parallel Tree-Grafting Algorithm for Maximum Cardinality Matching, IPDPS 2015; IEEE TPDS, 2016.

Collaborators

b-Matching, *b*-Edge cover	Arif Khan
b-Suitor Algorithm	Fredrik Manne
	Mahantesh Halappanavar
Adaptive Anonymity	Krzysztof Choromanski (Google)
b-matching	Intel PCL
	(Mostofa Patwary, Nadathur Satish,
	Narayanan Sunderam, Pradeep Dubey)
Multicore, XMT, GPU	MH, John Feo, Antonino Tumeo,
	Oreste Villa
max cardinality	Ariful Azad, Aydin Buluc
Push Relabel	Johannes Langguth
Network Alignment	David Gleich
Comp. Immunology	Ariful Azad, Bartek Rajwa
Vertex Wted Matching	Ahmed Al-Herz, Florin Dobrian, MH

Eighth SIAM Workshop on Combinatorial Scientific Computing

June 6-8, 2018
University of Bergen, Norway

www.csc-research.org/CSC18/
Organizer: Fredrik Manne (Bergen)
PC Chairs: Sivan Toledo (Tel Aviv) and Peter Sanders (Karlsruhe)
Please submit papers Dec 1, 2018.
Proceedings published by SIAM.