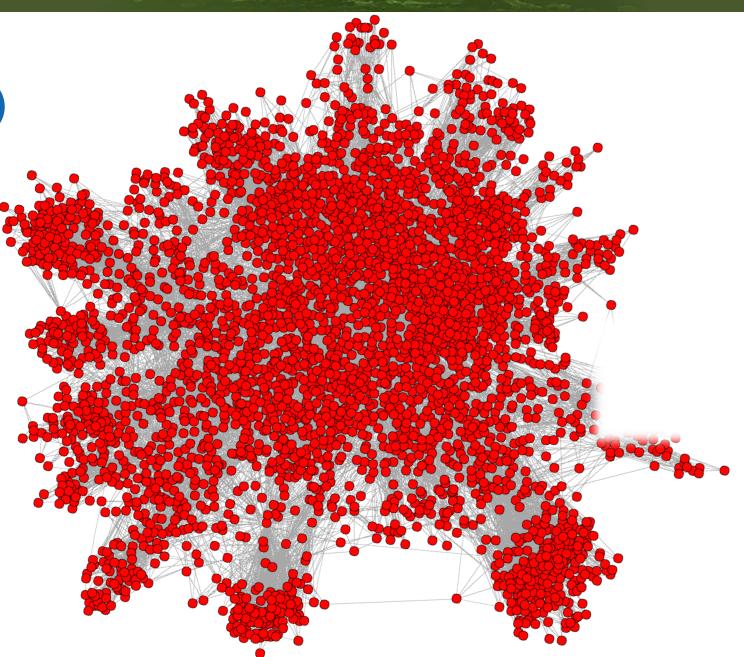








Why do we care?

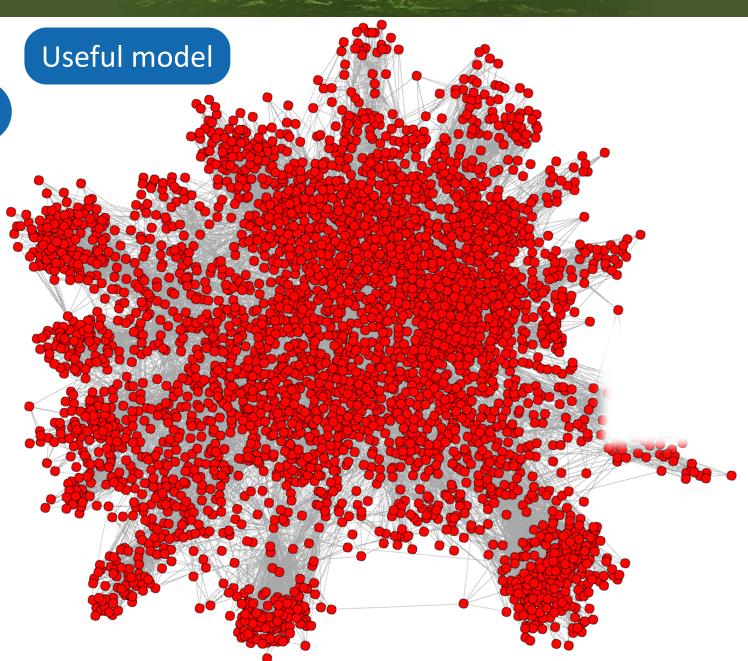








Why do we care?



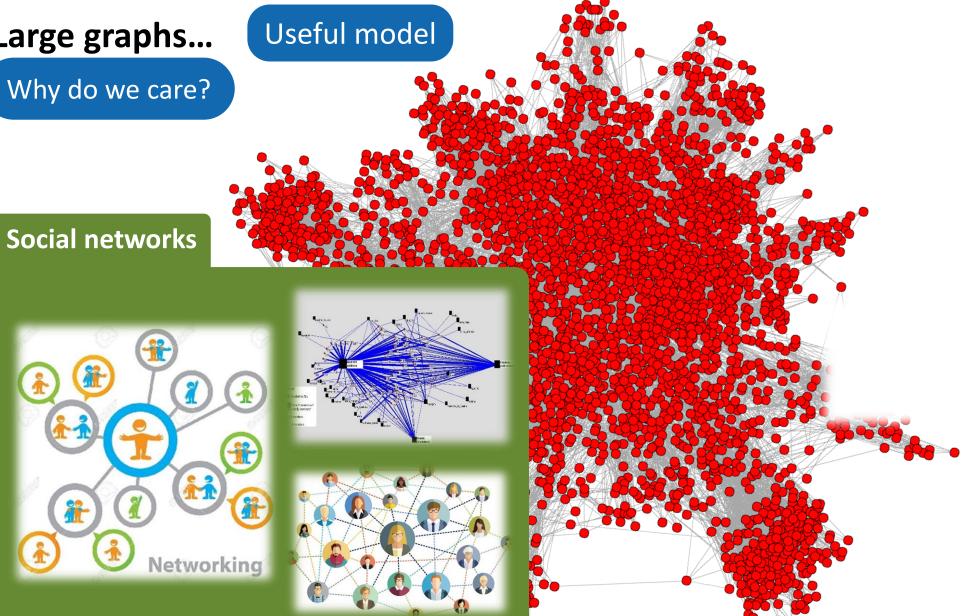








Why do we care?







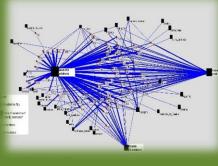


Why do we care?

Useful model





















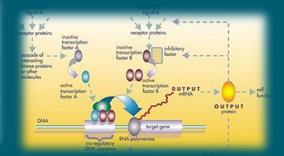
Why do we care?

Social networks

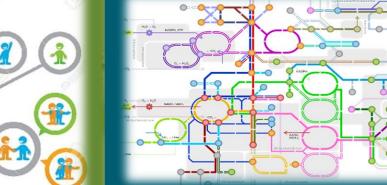
Useful model

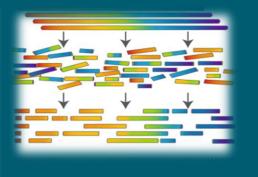
Biological networks

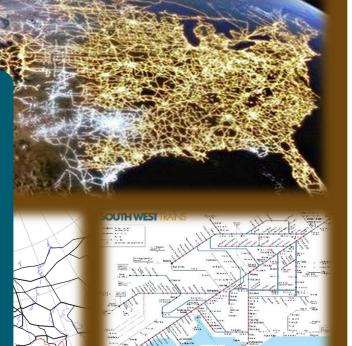




Engineering networks

















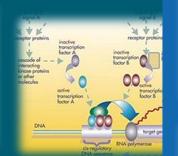
Social networks

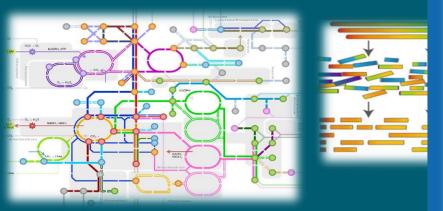
Why do we care?

Useful model

Biological networks



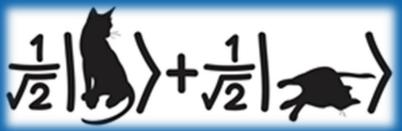


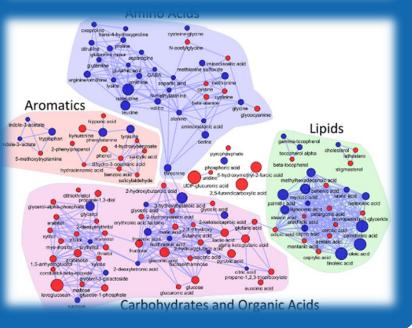




Physics, chemistry

Engineering networks











Why do we care?

Useful model

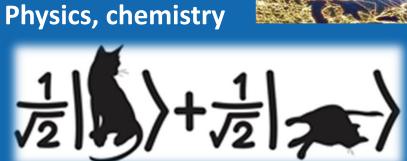
Biological networks

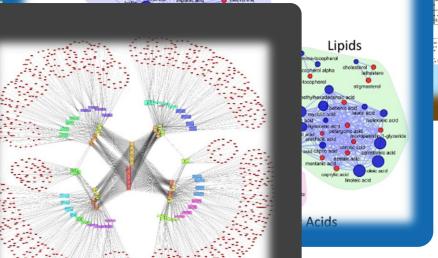


Communication networks



Engineering networks



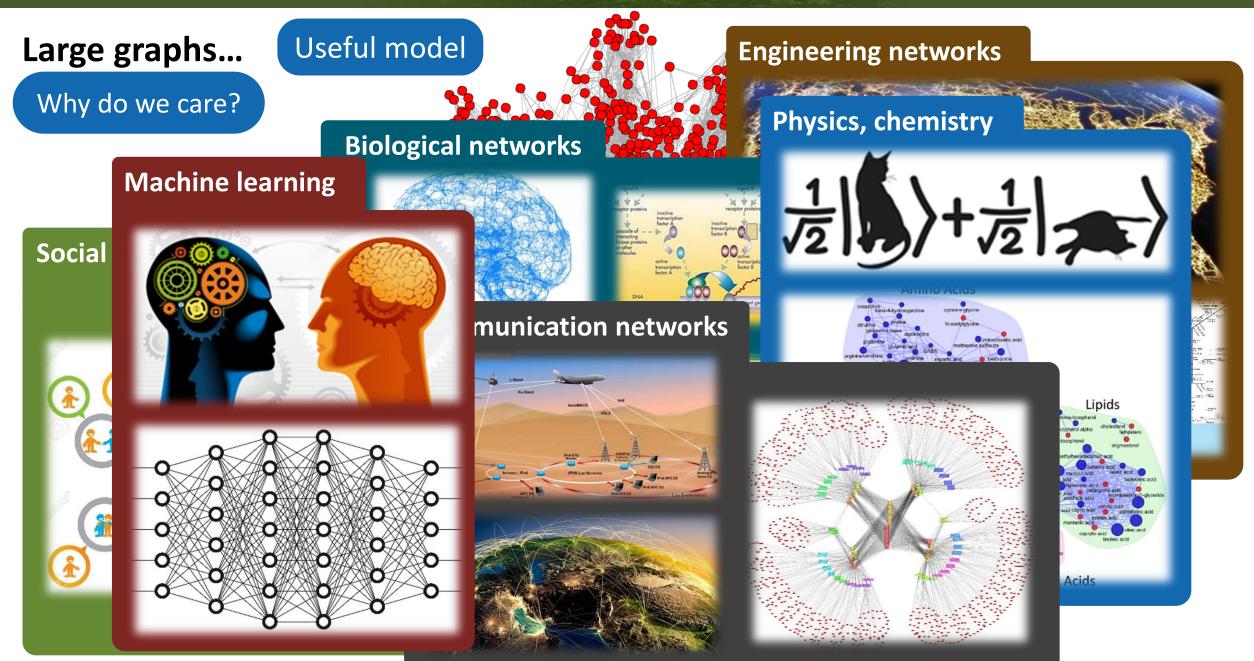


Social networks















Useful model

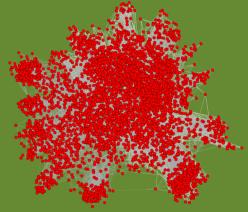
Why do we care?

...even philosophy ©

Engineering networks

hysics, chemistry





FOSDEM 2016 / Schedule / Events / Developer rooms / Graph Processing / Modeling a Philosophical Inquiry: from MySQL to a graph database

Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

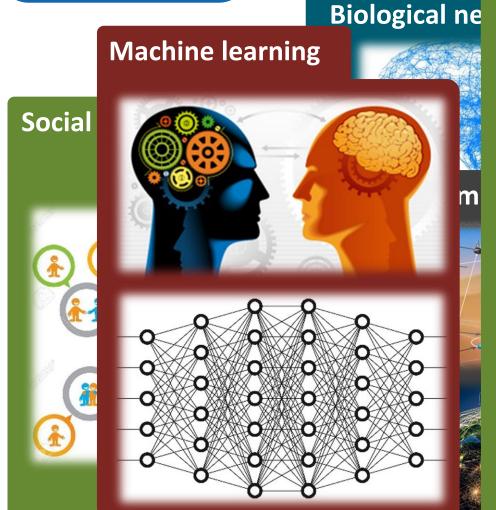
A Track: Graph Processing devroom ♠ Room: AW1.126

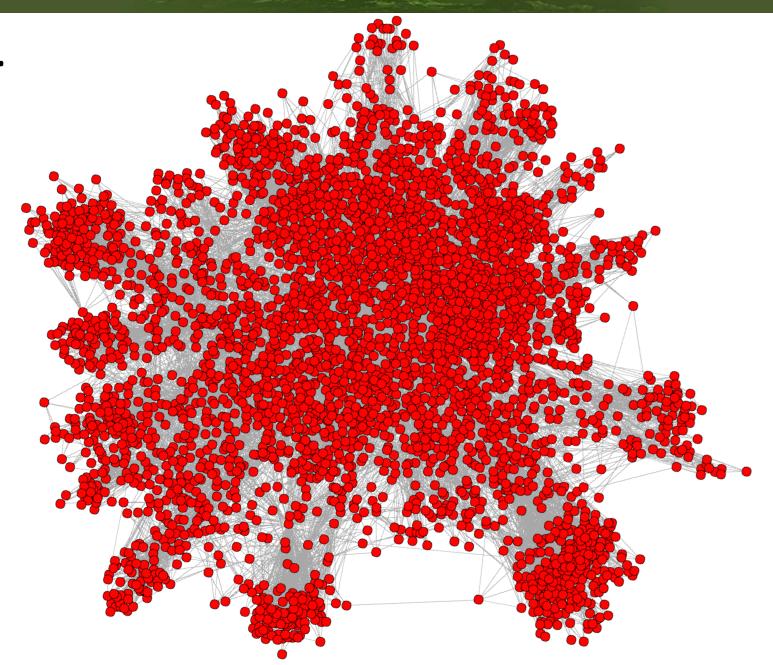
Day: Saturday ▶ Start: 12:45

■ End: 13:35



Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version



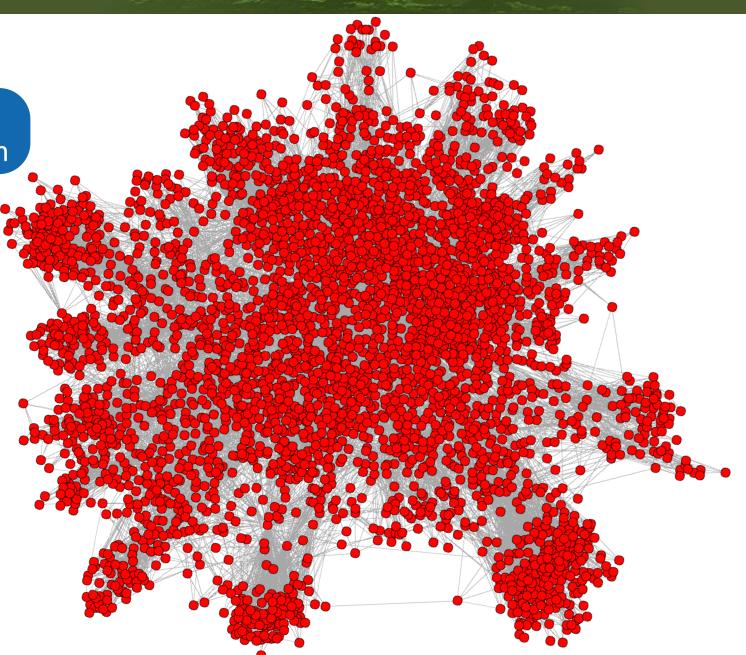








One particularly important problem

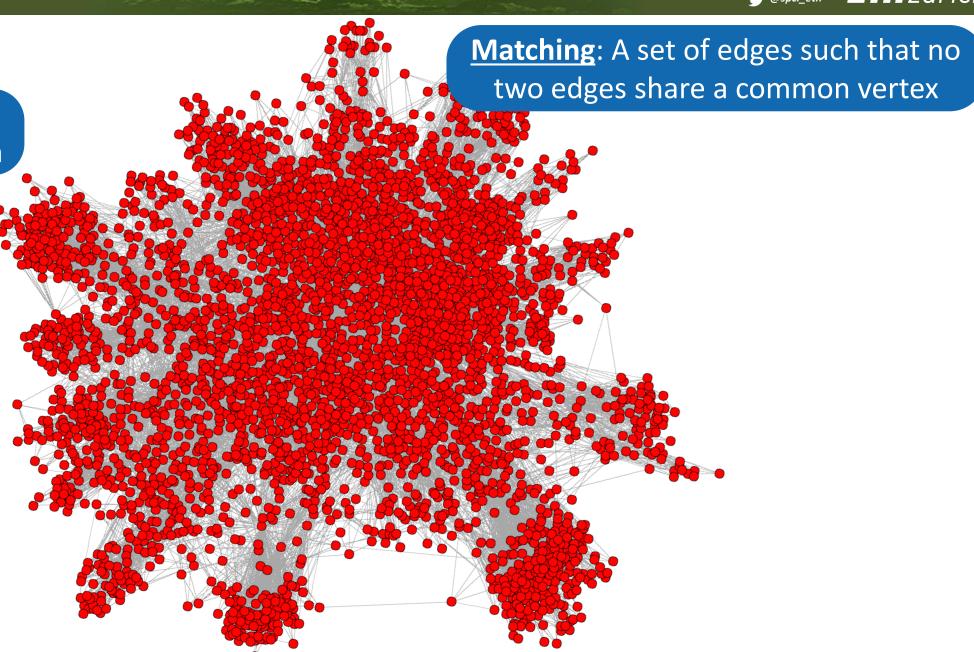








One particularly important problem

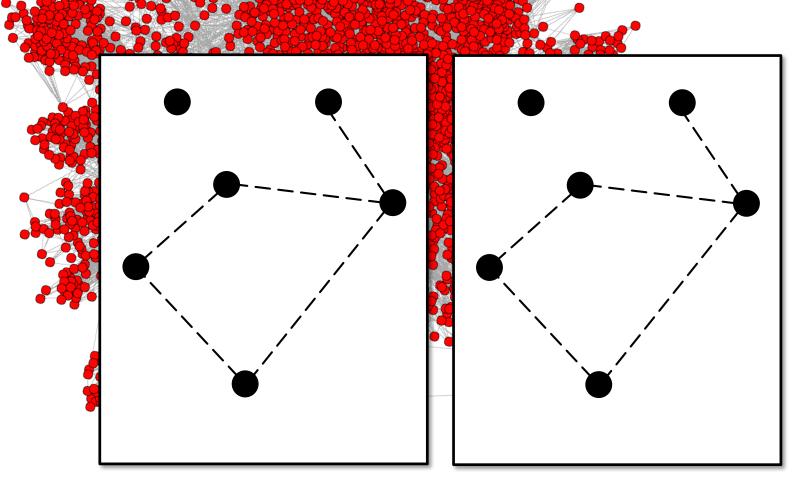








One particularly important problem

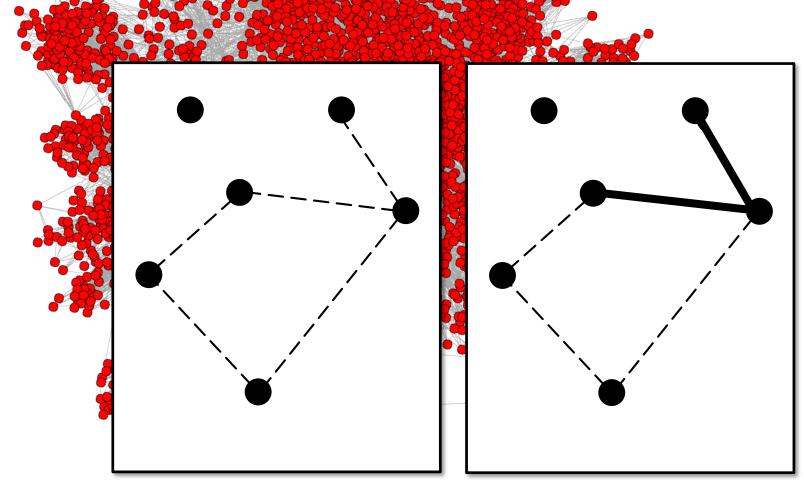








One particularly important problem

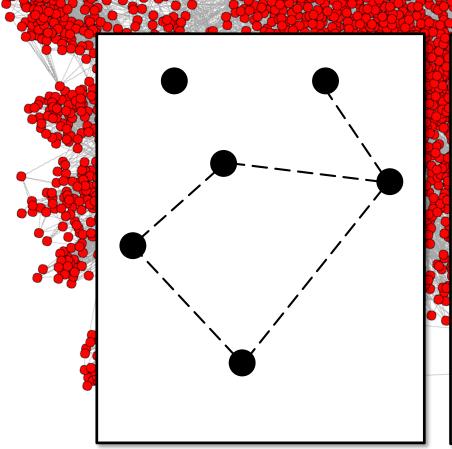


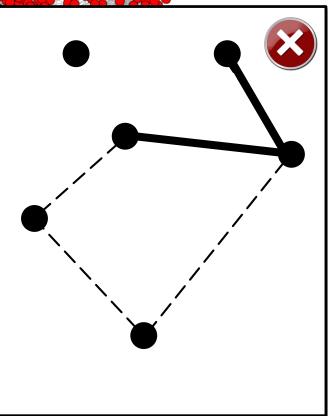






One particularly important problem



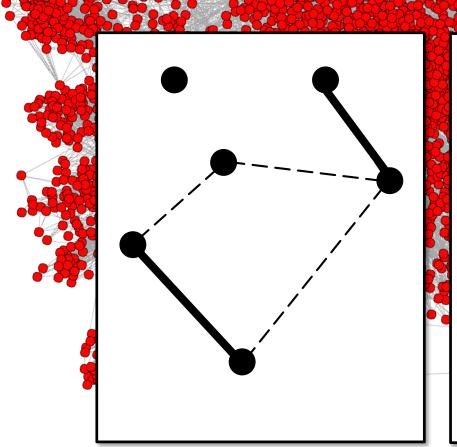


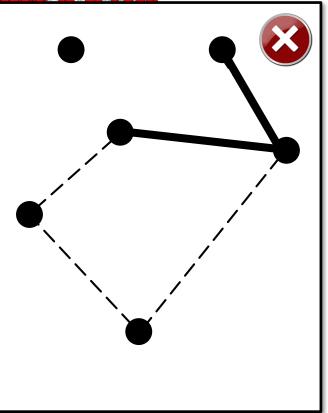






One particularly important problem



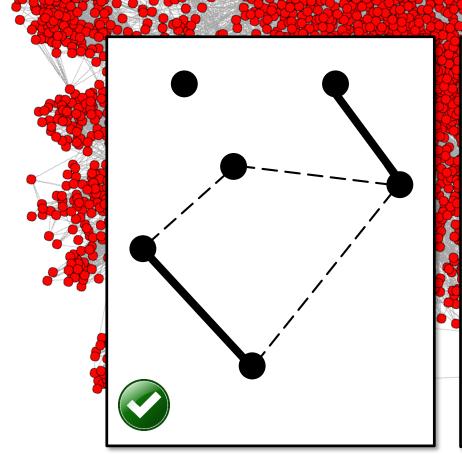


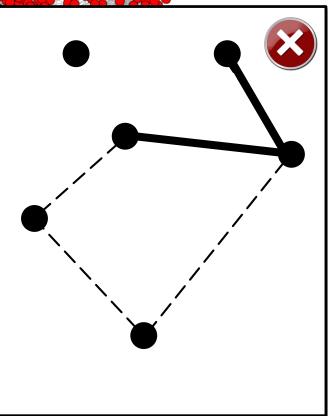






One particularly important problem





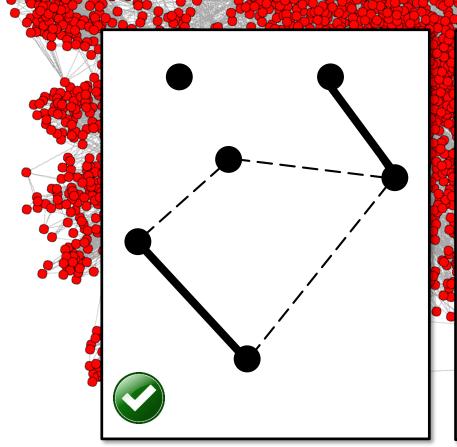


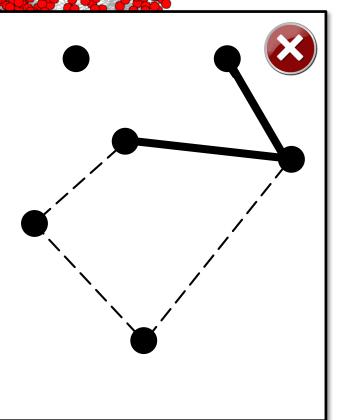




One particularly important problem

Matching: A set of edges such that no two edges share a common vertex





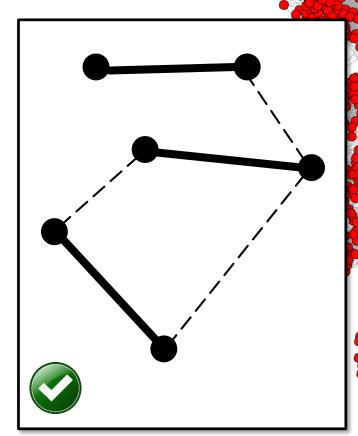


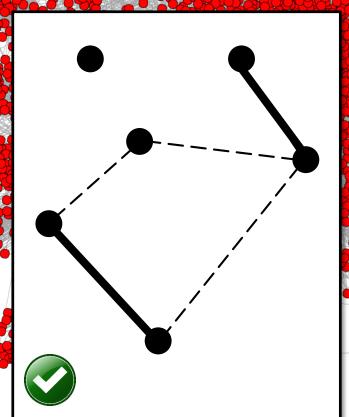


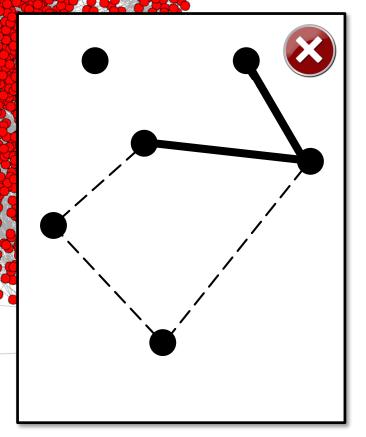


One particularly important problem

Matching: A set of edges such that no two edges share a common vertex





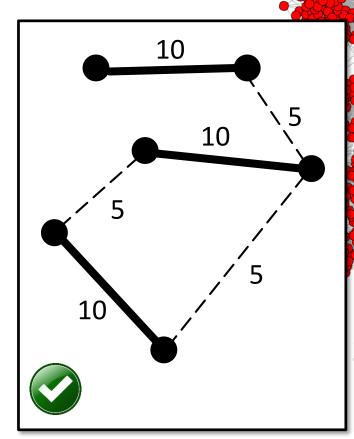


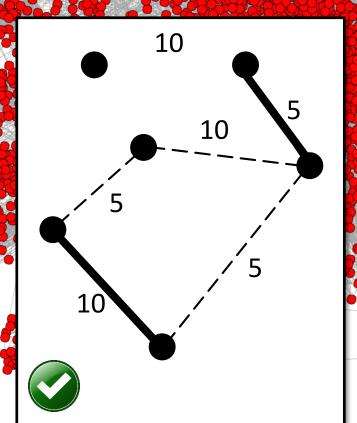


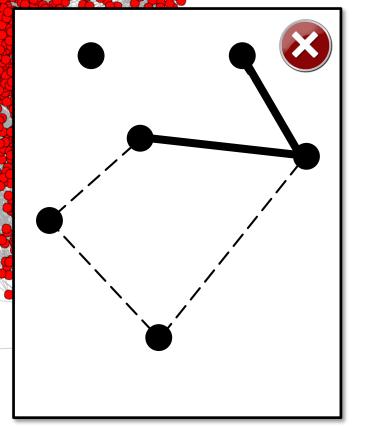


One particularly important problem

Matching: A set of edges such that no two edges share a common vertex



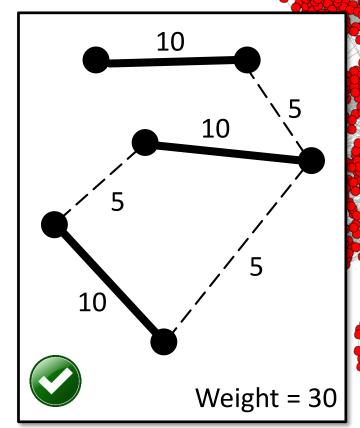


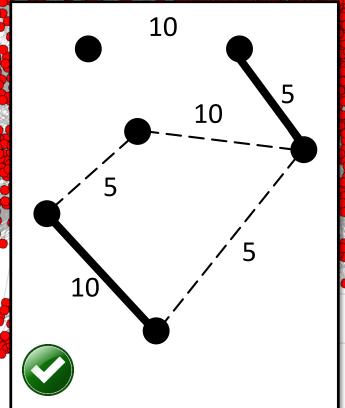


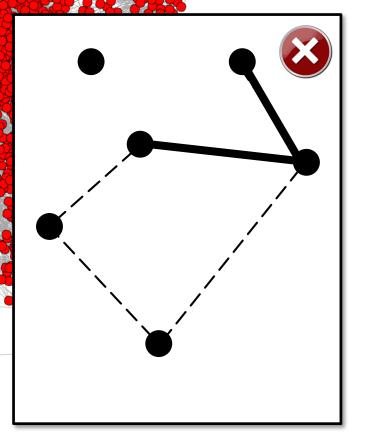


One particularly important problem

Matching: A set of edges such that no two edges share a common vertex



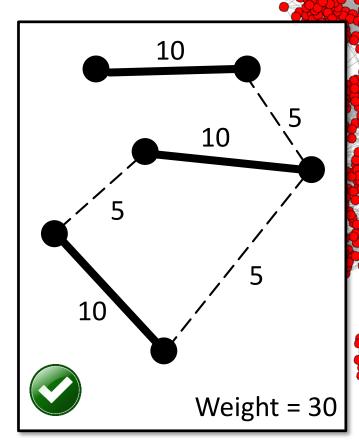


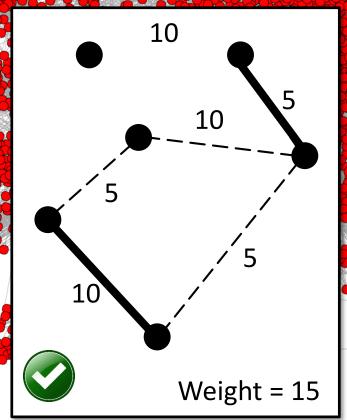


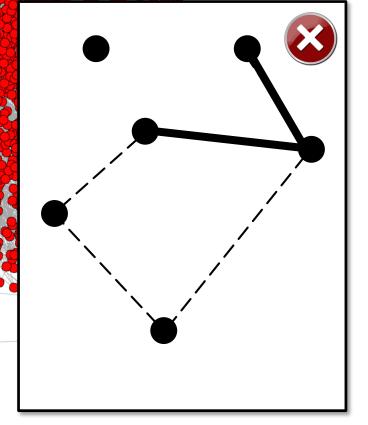


One particularly important problem

Matching: A set of edges such that no two edges share a common vertex



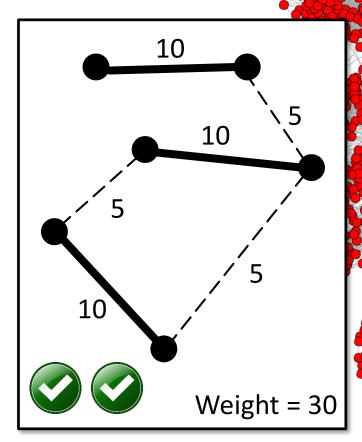


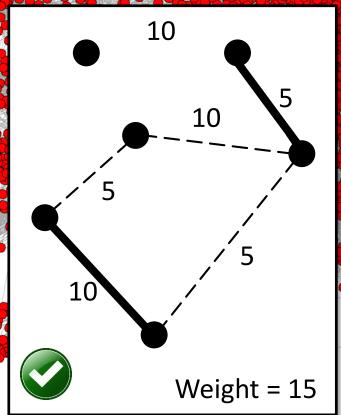


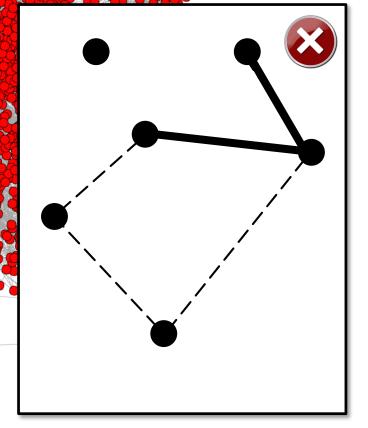


One particularly important problem

Matching: A set of edges such that no two edges share a common vertex



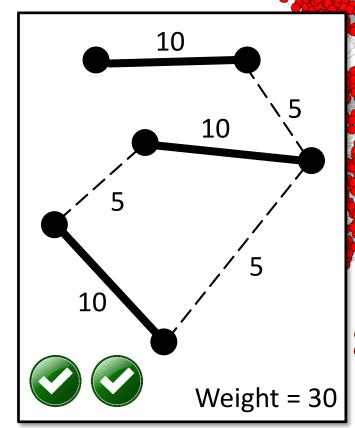


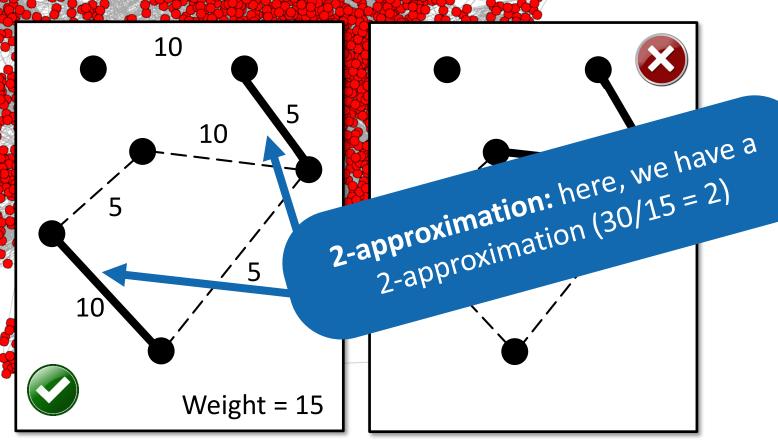




One particularly important problem

Matching: A set of edges such that no two edges share a common vertex







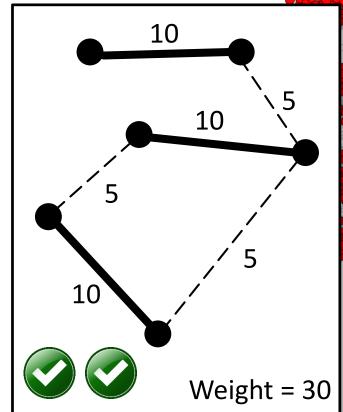


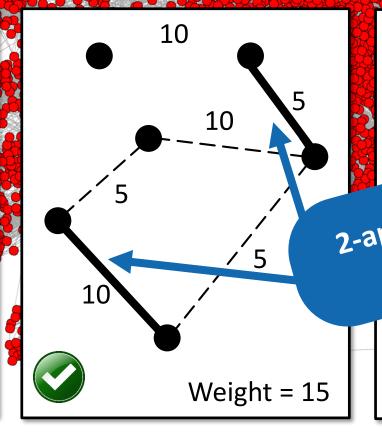


One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized





2-approximation: here, we have a 2-approximation (30/15 = 2)

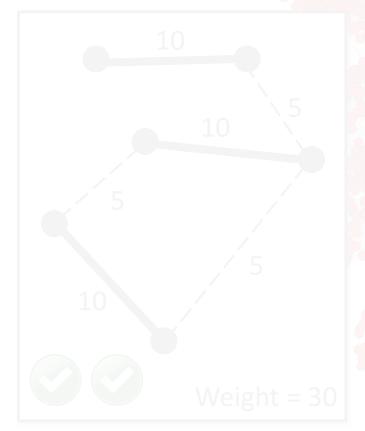
Approximate MWMs are crucial to many problems

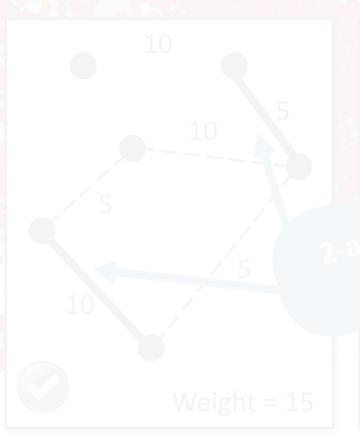


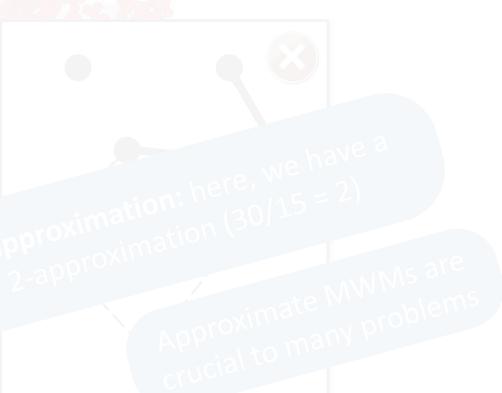
One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (<u>MWM</u>): A matching such that the sum of the edge weights is maximized







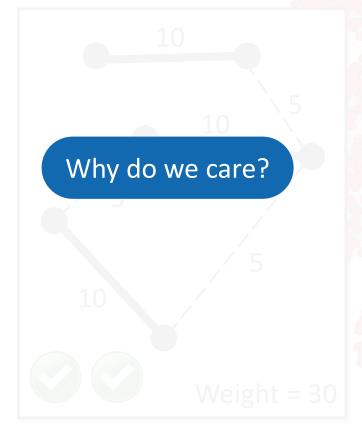


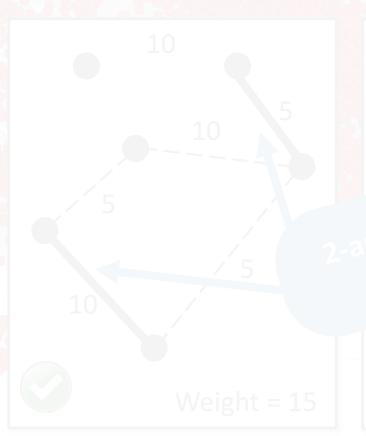


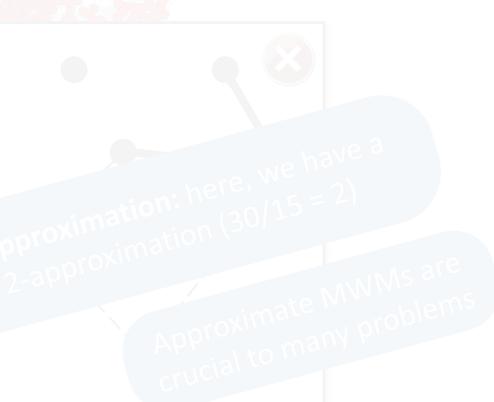
One particularly important problem

Matching: A set of edges such that no two edges share a common vertex

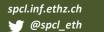
Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized















<u>Matching</u>: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching uch that the sum of the edge weights is maximized

5 Scheduling

Why do we care?

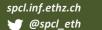
Weight = 30

Weight = 15

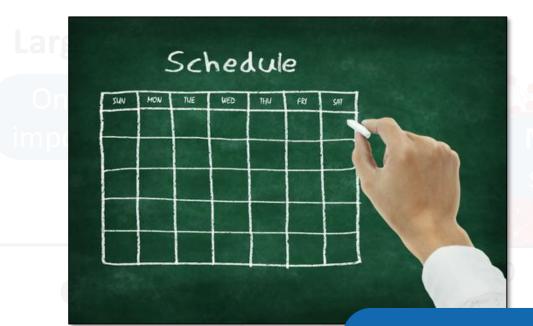
Approximate MWMs

Crucial to many problems









Matching: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

Scheduling

Why do we care?

Quantum Error-Correction

[Quantum] error correcting codes

Weight = 15









<u>Matching</u>: A set of edges such that no two edges share a common vertex

Maximum Weighted Matching (MWM): A matching such that the sum of the edge weights is maximized

Scheduling

Why do we care?

Quantum Error-Correction

[Quantum] error correcting codes





Transplant matching









SOUTH WEST TRANS

Traveling

Salesman

edges such that no a common vertex

<u>IWM</u>): A matching Ights is maximized

Scheduling

Why do we care?

Quantum Error-Correction [Quantum] error correcting codes

Problem



Transplant matching

Weight = 15









SOUTH WEST TRANS

SOUTH WEST TRANS

SOUTH WEST TRANS

STATE OF THE STA

edges such that no a common vertex

WM): A matching

Many, many others...

Scheduling

Why do we care?

Quantum Error-Correction [Quantum] error correcting codes



Problem



Transplant matching

Weight = 15







In all cases, approximations ("reasonably" accurate) are useful **Traveling**

SOUTH WEST

Problem Scheduling

"We live in a system of approximations" — Ralph Waldo Emerson

> Many, many others...

Why do we care?

Quantum Error-Correction [Quantum] error correcting codes

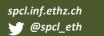


Salesman



Transplant matching













Which programming paradigm to use for (approximate) MWM (and other graph problems)?







How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Which programming paradigm to use for (approximate) MWM (and other graph problems)?







How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Which programming paradigm to use for (approximate) MWM (and other graph problems)?

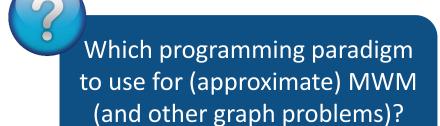
What is the HW FPGA design that ensures high performance?







How to design a highperformance MWM algorithm (as dictated by the used paradigm)?



What is the HW FPGA design that ensures high performance?



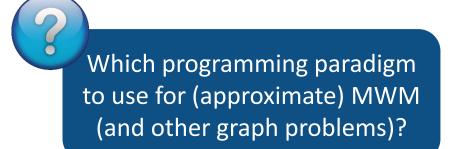
What is the ultimate performance, power consumption, and the related tradeoffs?







How to design a highperformance MWM algorithm (as dictated by the used paradigm)?



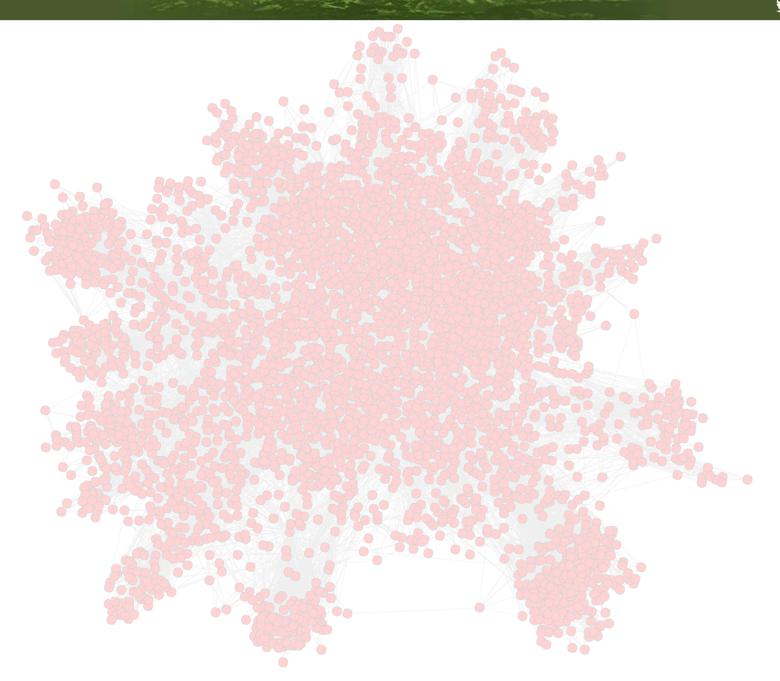
What is the HW FPGA design that ensures high performance?



What is the ultimate performance, power consumption, and the related tradeoffs?

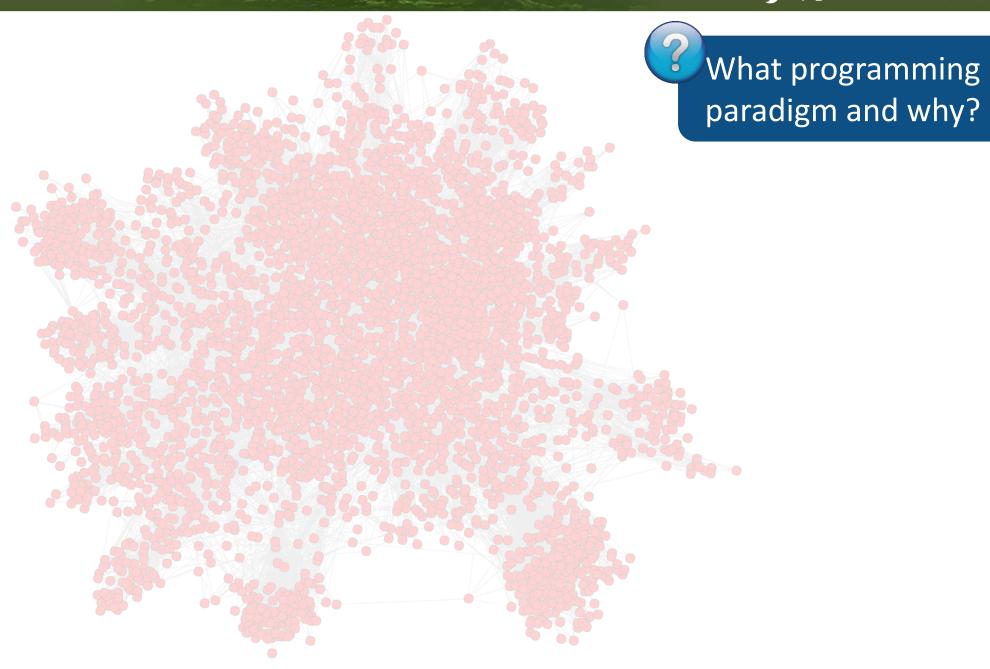






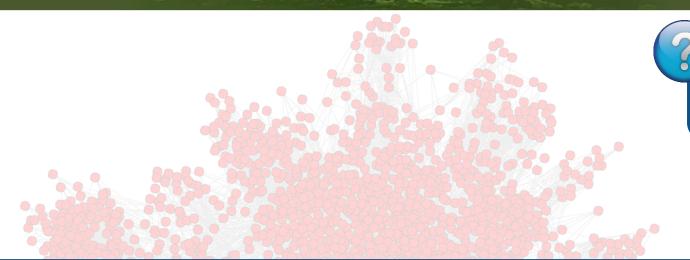










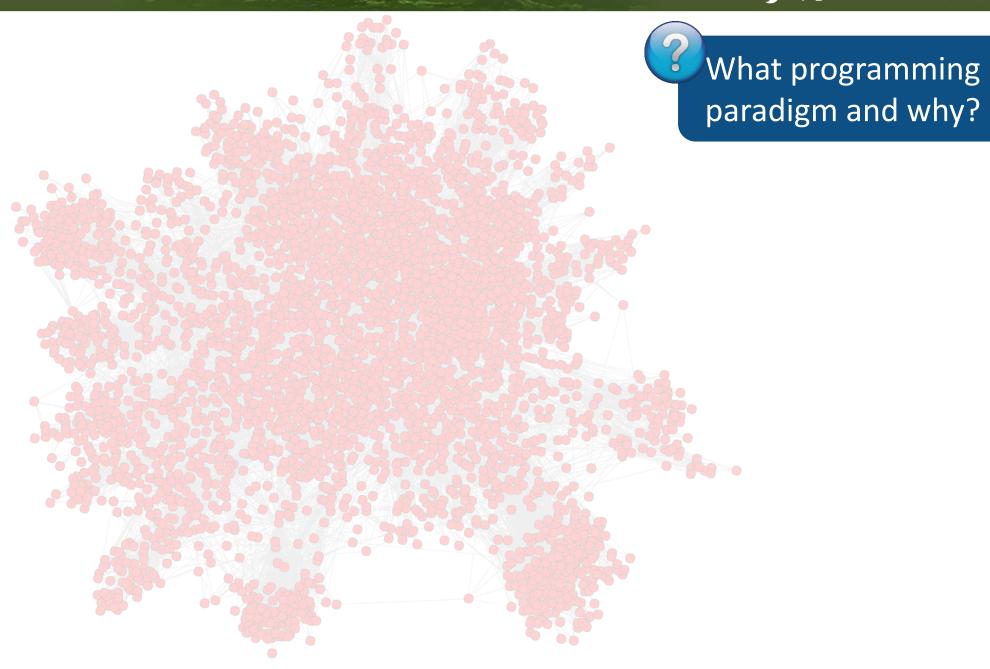


What programming paradigm and why?

Part 1: Seeking "the best paradigm", we conducted a detailed analysis of graph processing on FPGAs



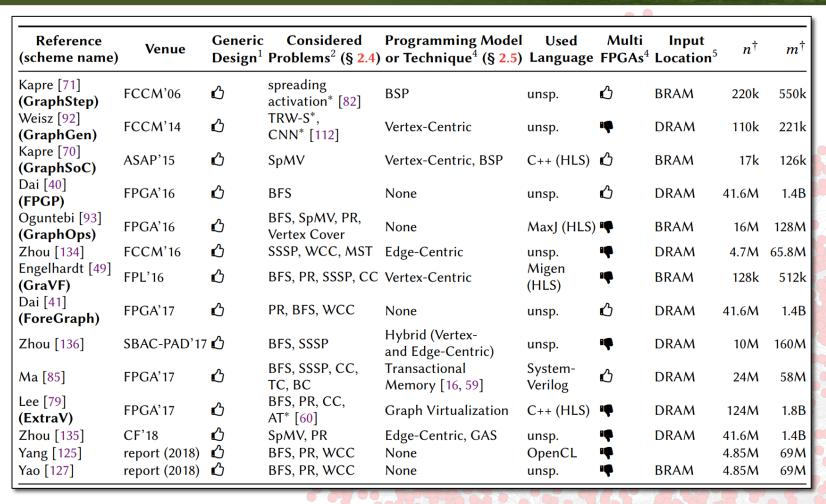














~15 FPGA graph processing frameworks







Reference (scheme name)	Ven	ue Generic Design ¹		Programming .4) or Technique ⁴	Model Used M (§ 2.5) Language FP	lulti Input GAs ⁴ Location ⁵	n^{\dagger} m^{\dagger}		
Kapre [71] (GraphStep)	FCCM'	06 戊	spreading activation* [82]	BSP	unsp. 🖒	BRAM	220k 550k		
Weisz [92] (GraphGen)	FCCM'	14 🖒	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	DRAM	110k 221k		
Kapre [70] (GraphSoC) Dai [40]	ASAP'1	Babb [4] Dandalis [43]	report (1996) • report (1999) •	SSSP SSSP	None None	Verilog 🖒 unsp.	Hardwired Hardwired		2051 32k
(FPGP)	FPGA'1	Tommiska [116]	report (2001)	SSSP	None	VHDL •	BRAM	64	4096
Oguntebi [93] (GraphOps)	FPGA'1	Mencer [87]	FPL'02	Reachability, SSSP	None	PAM- -Bloks II	Hardwired (3-state buffers)	88	7744
Zhou [134] Engelhardt [49] (GraVF)	FCCM' FPL'16	Bondhugula [27] Sridharan[110]	TENCON'09	APSP SSSP	Dynamic Program. None	unsp. •• VHDL ••	DRAM BRAM	unsp.	88
Dai [41] (ForeGraph)	FPGA'1	Wang [121] Betkaoui [21] Jagadeesh [65]	ICFTP'10	BFS GC SSSP	None Vertex-Centric None	SystemC Verilog 🖒 VHDL 📭	DRAM DRAM Hardwired	65.5k 300k I 128	1M 3M 466
Zhou [136]	SBAC-F	Betkaoui [22] Betkaoui[23]	FPL'12 ASAP'12	APSP BFS	Vertex-Centric Vertex-Centric	Verilog 🖒 Verilog 🖒	≈ DRAM DRAM	38k 16.8M	72M 1.1B
Ma [85]	FPGA'1	Attia [2] (CyGraph)	IPDPS'14	BFS	Vertex-Centric	VHDL 🖒	DRAM	8.4M	536M
Lee [79] (ExtraV)	FPGA'1	Ni [91]	report (2014)	BFS	None	Verilog 📭	DRAM, SRAM	16M	512M
Zhou [135] Yang [125] Yao [127]	CF'18 report (report (Zhou [132] Zhou [133] Umuroglu [117] Lei [80]	ReConFig'15 FPL'15 report (2016)	SSSP PR BFS SSSP	None Edge-Centric None None	unsp. unsp. Chisel unsp.	DRAM DRAM ≈ DRAM DRAM	1M 2.4M 2.1M 23.9M	unsp. 5M 65M 58.2M
		Zhang [129] Zhang [130] Kohram [76]	FPGA'17 FPGA'18 FPGA'18	BFS BFS BFS	MapReduce None None	unsp. unsp. unsp.	HMC HMC HMC	33.6M	536.9M
		Besta [13]	FPGA'19	MM	Substream-Centric	Verilog 📭	DRAM	4.8M	117M



~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms







Reference (scheme name)	Ven	ue Generic Design ¹			Programming or Technique ⁴	Model Used (§ 2.5) Language		ulti Inp GAs ⁴ Locat		n^{\dagger}	m^{\dagger}		
Kapre [71] (GraphStep)	FCCM'	06 🖒	spreading activation* [8	2]	BSP	unsp.	Ů	BRAN	1	220k 5	550k		
Weisz [92] (GraphGen)	FCCM'	14 🖒	TRW-S*, CNN* [112]		Vertex-Centric	unsp.	•	DRAA	1	110k 2	221k	ļ.,	
Kapre [70] (GraphSoC) Dai [40]	ASAP'1	Babb [4] Dandalis [43]	1 ,	* F	SSSP SSSP	None None		Verilog unsp.	<u>ර</u>	Hardv Hardv		512 2048	2051 32k
(FPGP)	FPGA'1		. , ,	•	SSSP	None		VHDL	•	$BRA\mathcal{N}$	١	64	4096
Oguntebi [93] (GraphOps)	FPGA'1	Mencer [87]	FPL'02	•	Reachability, SSSP	None		PAM- -Bloks II	•	Hardv (3-stat buffer	te	88	7744
Zhou [134] Engelhardt [49] (GraVF)	FCCM' FPL'16	Bondhugula [27] Sridharan[110]	TENCON'09	* F	APSP SSSP	Dynamic Progra None	am.	unsp. VHDL	# #	DRAN BRAN	1	unsp. 64	88
Dai [41] (ForeGraph)	FPGA'1	Wang [121] Betkaoui [21] Jagadeesh [65]	FTP'11	" ? " ? " ?	BFS GC SSSP	None Vertex-Centric None		SystemC Verilog VHDL	∆ •	DRAM DRAM Hardv	1	65.5k 300k 128	1M 3M 466
Zhou [136]	SBAC-F	Betkaoui [22] Betkaoui[23]		•	APSP BFS	Vertex-Centric Vertex-Centric		Verilog Verilog	ů ů	≈ DRA DRAM		38k 16.8 <i>M</i>	72M 1.1B
Ma [85]	FPGA'1	Attia [2] (CyGraph)	IPDPS'14	•	BFS	Vertex-Centric		VHDL	Ů	DRAM	1	8.4M	536M
Lee [79] (ExtraV)	FPGA'1		, (a.,)	•	BFS	None		Verilog	•	DRAN SRAM		16M	512M
Zhou [135] Yang [125] Yao [127]	CF'18 report (report (Zhou [132] Zhou [133] Umuroglu [117] Lei [80]	ReConFig'15 FPL'15	** ** **	SSSP PR BFS SSSP	None Edge-Centric None None		unsp. unsp. Chisel unsp.	**	DRAA DRAA ≈ DRA DRAA	1 AM	1M 2.4M 2.1M 23.9M	unsp. 5M 65M 58.2M
		Zhang [129] Zhang [130] Kohram [76]	FPGA'17 FPGA'18 FPGA'18	: + : + : +	BFS BFS BFS	MapReduce None None		unsp. unsp. unsp.	•	HMC HMC HMC		33.6M	536.9M
		Besta [13]	FPGA'19	•	MM	Substream-Cent	tric	Verilog	•	DRAM	1	4.8M	117M

What programming paradigm and why?

Key techniques, paradigms, challenges, features, ...

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms







Reference (scheme name)	Ven	ue Generic Design ¹			Programming A or Technique ⁴ (Model Used § 2.5) Language F	Multi Inp PGAs ⁴ Locat		n^{\dagger} n	<u>,</u> †	
Kapre [71] (GraphStep)	FCCM'	06 戊	spreading activation* [8	2]	BSP	unsp.	b BRAN	1	220k 550	0k	
Weisz [92] (GraphGen)	FCCM'	14 🖒	TRW-S*,		Vertex-Centric	unsp.	DRAM	1	110k 22	1k	
Kapre [70] (GraphSoC) Dai [40]	ASAP'1 FPGA'1	Bat Se	elected	q b	arts are	in the F	PGA	,	Hardwir Hardwir		
(FPGP) Oguntebi [93] (GraphOps)	FPGA'1	Ton	pap	oer	, the re	st is in			BRAM Hardwir (3-state	64 ed 88	
Zhou [134] Engelhardt [49]	FCCM' FPL'16	Bondhugula [27] Sridharan[110]		• •	APSP SSSP	Dynamic Program. None	unsp. VHDL	# #	buffers) DRAM BRAM	unsp 64	
(GraVF) Dai [41]	FPGA'1	Wang [121] Betkaoui [21]	ICFTP'10 FTP'11	** **	BFS GC	None Vertex-Centric	SystemC Verilog	ů	DRAM DRAM	65.5k 300k	c 1M
(ForeGraph) Zhou [136]	SBAC-F	Jagadeesh [65] Betkaoui [22] Betkaoui[23]	FPL'12	* F * F	SSSP APSP BFS	None Vertex-Centric Vertex-Centric	VHDL Verilog Verilog	♥ ②	Hardwir ≈ DRAM DRAM		c 72M
Ma [85]	FPGA'1	Attia [2] (CyGraph)		•	BFS	Vertex-Centric	VHDL	Ů	DRAM	8.4 <i>N</i>	
Lee [79] (ExtraV)	FPGA'1		1	•	BFS SSSP	None None	Verilog	•	DRAM, SRAM DRAM	16 <i>N</i> 1 <i>N</i>	
Zhou [135] Yang [125]	CF'18 report (Zhou [132] Zhou [133] Umuroglu [117]	ReConFig'15	· F	PR BFS	Edge-Centric None	unsp. unsp. Chisel	**	DRAM ≈ DRAM	2.4 <i>N</i>	1 5M
Yao [127]	report (Lei [80] Zhang [129] Zhang [130]	FPGA'17	* F	SSSP BFS BFS	None MapReduce None	unsp. unsp. unsp.	4	DRAM HMC HMC	23.9 <i>N</i> 33.6 <i>N</i>	1 58.2M 1 536.9M
		Kohram [76] Besta [13]		•	BFS MM	None Substream-Centric	unsp. Verilog	•	HMC DRAM	4.8 <i>N</i>	1 117M

What programming paradigm and why?

Key techniques, paradigms, challenges, features, ...

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms





512

unsp.

65.5k 300k

128

38k

1M

2.4M

2.1M23.9M 58.2M

16.8M 8.4M 2051 32k 4096

7744

88 1*M*

3M

466

72M 1.1B

536M

unsp. 5M

65M

16M 512M

33.6M 536.9M

4.8M 117M



Reference (scheme name)	Venue	Generi Design	c Considered ¹ Problems ² (§ 2.4	Programming <i>N</i>) or Technique ⁴ (§		Mu e FPG		n^{\dagger}	m
Kapre [71] (GraphStep)	FCCM'06	Ů	spreading activation* [82]	BSP	unsp.	மீ	BRAM	220k	550
Weisz [92] (GraphGen)	FCCM'14	Ů	TRW-S*,	Vertex-Centric	unsp.	•	DRAM	110k	221
Kapre [70] (GraphSoC) Dai [40] (FPGP) Oguntebi [93] (GraphOps)	ASAP'1 Bat Bat Ton FPGA'1 Me	S	elected p pape	parts are r, the res			GA	Har BRA Har (3-s	dwire tate
Zhou [134] Engelhardt [49] (GraVF) Dai [41]	FPL'16 Sridl	dhugula [27 haran[110] ng [121]	7] IPDPS'06 TENCON'09 ICFTP'10	APSP SSSP BFS	Dynamic Prog None None	ram.	unsp. VHDL SystemC	buff DR/ BR/ DR/	AM AM
(ForeGraph) Zhou [136] Ma [85] Lee [79] (ExtraV) Zhou [135] Yang [125] Yao [127]		•	ocessing on F		•	•	hallenges		
	MAC	CIEJ BEST	standing of Modern A*, DIMITRI STAN E FINE LICHT, TA	OJEVIC*, Departm	ent of Computer S	cience, E			

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or

social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field Programmable Gate Arrays (FPGAs) can be an energy-efficient solution to deliver specialized hardware for What programming paradigm and why?

Key techniques, paradigms, challenges, features, ...

> ~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms



FPGA'1

FPGA'1 Me

Dai [40]

(FPGP)

Oguntebi [93]

(GraphOps)

Engelhardt [49]

Zhou [134]

(GraVF)

Dai [41] (ForeGraph)

Zhou [136]

Ma [85]

Lee [79] (ExtraV)

Zhou [135] Yang [125]

Yao [127]





Reference (scheme name	venue	Gene Desig	eric Considered gn ¹ Problems ² (§ 2.4	Programming Mo 1) or Technique (§		Mult e FPGA:		n^{\dagger}	m^{\dagger}
Kapre [71] (GraphStep)	FCCM'06	Ů	spreading activation* [82]	BSP	unsp.	Ů	BRAM	220k	550k
Weisz [92] (GraphGen)	FCCM'14	Ů	TRW-S*,	Vertex-Centric	unsp.	u p	DRAM	110k	221k
Kapre [70] (GraphSoC)	ASAP'1 Bat		Selected i	narts are	in the	FPG	Δ	Har	dwired

paper, the rest is in...

http://spcl.inf.ethz.ch/Publications/.pdf/ graphs-fpgas-survey.pdf (submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field Programmable Gate Arrays (FPGAs) can be an energy-efficient solution to deliver specialized hardware for

f 512	2051	ı
2048	32k	ı
64	4096	ı
ł		ı
88	7744	ľ
unsp.		ı
64	88	ı
65.5k	1M	ı
300k	3M	ı
128	466	ı
38k	72M	ı
16.8M	1.1B	L
8.4M	536M	
16M	512M	
1M	unsp.	
2.4M	5M	ľ
2.1M	65M	L
23.9M	58.2M	
33.6M	536.9M	ı
		ı
4.8M	117M	
	1 2048 64 1 88 unsp. 64 65.5k 300k 1 128 38k 16.8M 8.4M 16M 2.4M 2.1M 23.9M 33.6M	1 2048 32k 64 4096 1 88 7744 unsp. 64 88 65.5k 1M 300k 3M 1 128 466 38k 72M 16.8M 1.1B 8.4M 536M 16M 512M 1M unsp. 2.4M 5M 2.1M 65M 23.9M 58.2M 33.6M 536.9M

0.0

What programming paradigm and why?

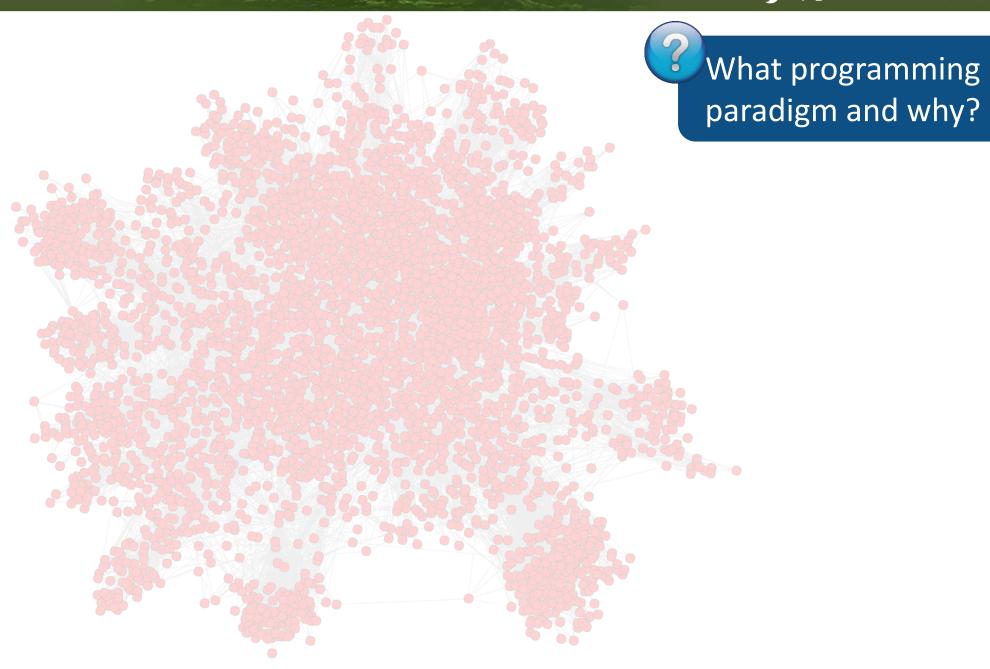
Key techniques, paradigms, challenges, features, ...

> ~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms











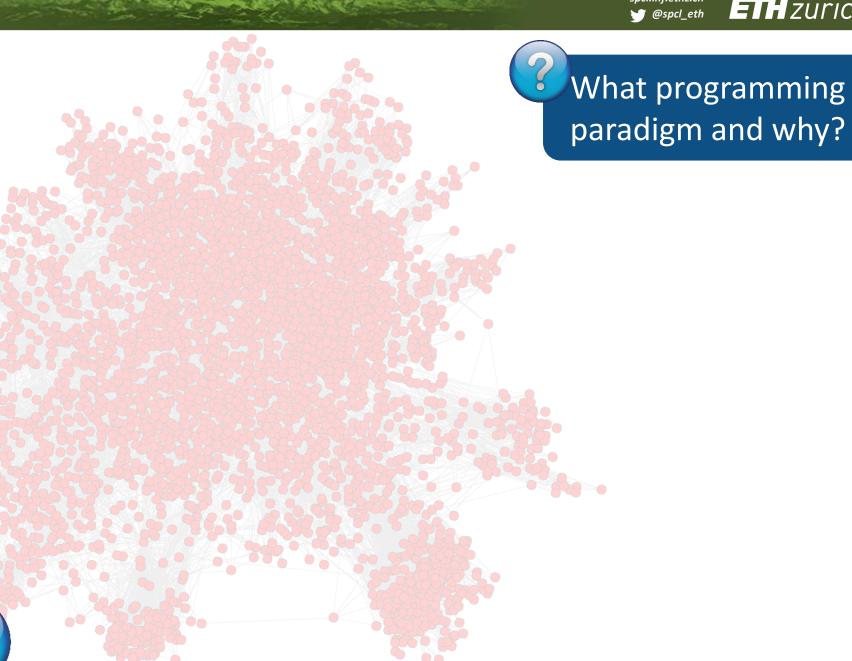


"(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]

+ other issues

[1] S. Salihoglu and J. Widom, "Optimizing graph algorithms on Pregel-like systems". VLDB. 2014.

Vertex-centric, Gather-Apply-Scatter, ...?









"(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]

+ other issues

[1] S. Salihoglu and J. Widom, "Optimizing graph algorithms on Pregel-like systems". VLDB. 2014.

To be able to <u>utilize pipelining</u>
<u>well</u>, we really want to use
<u>streaming</u> (aka edge-centric)

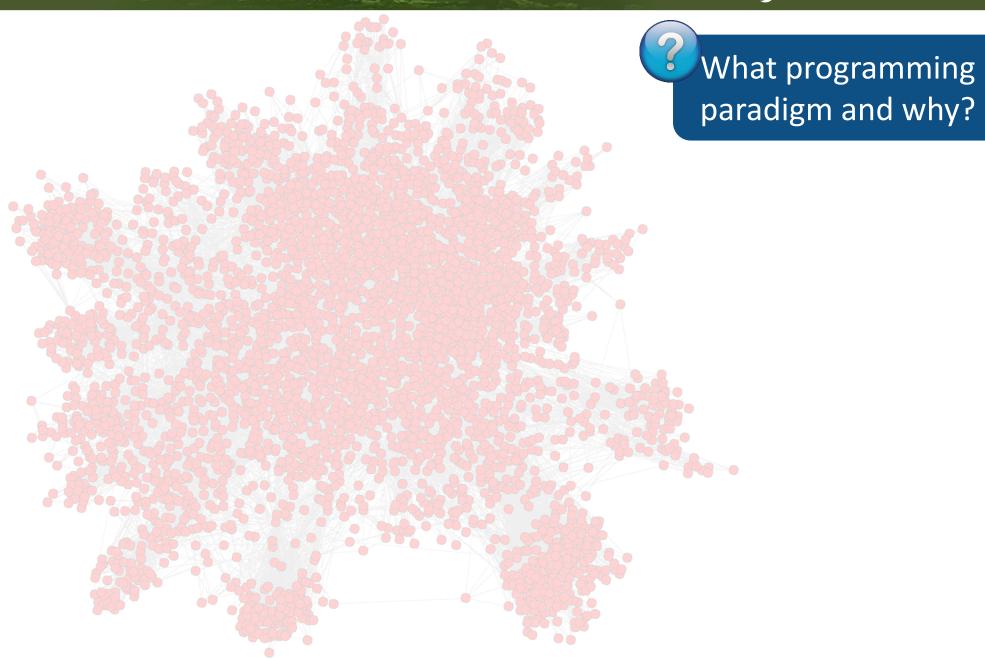
Vertex-centric, Gather-Apply-Scatter, ... ?

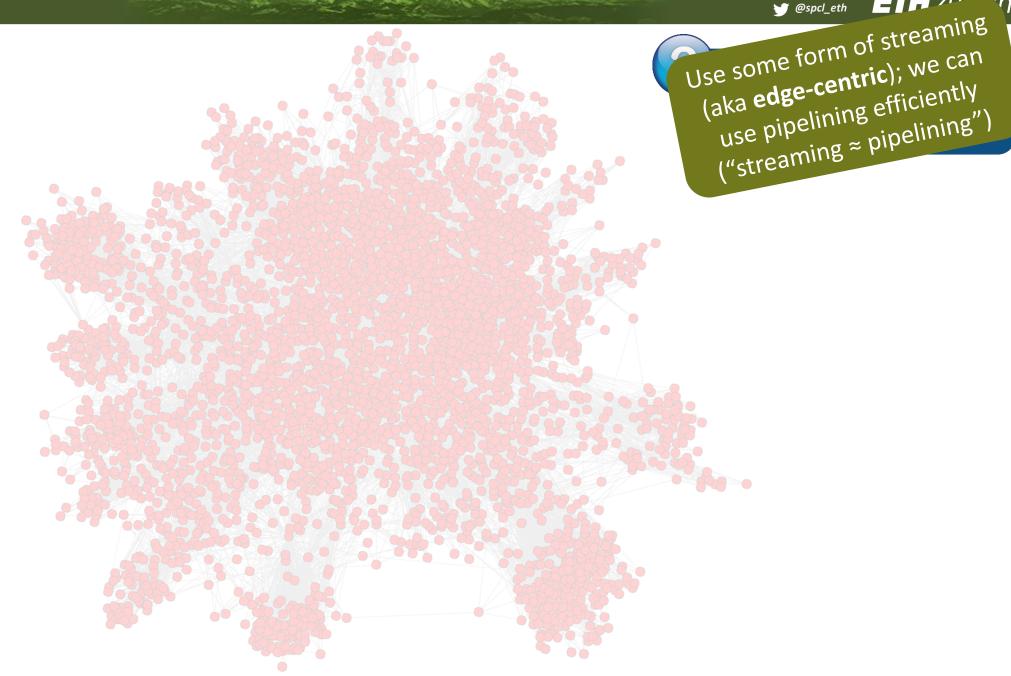


What programming paradigm and why?





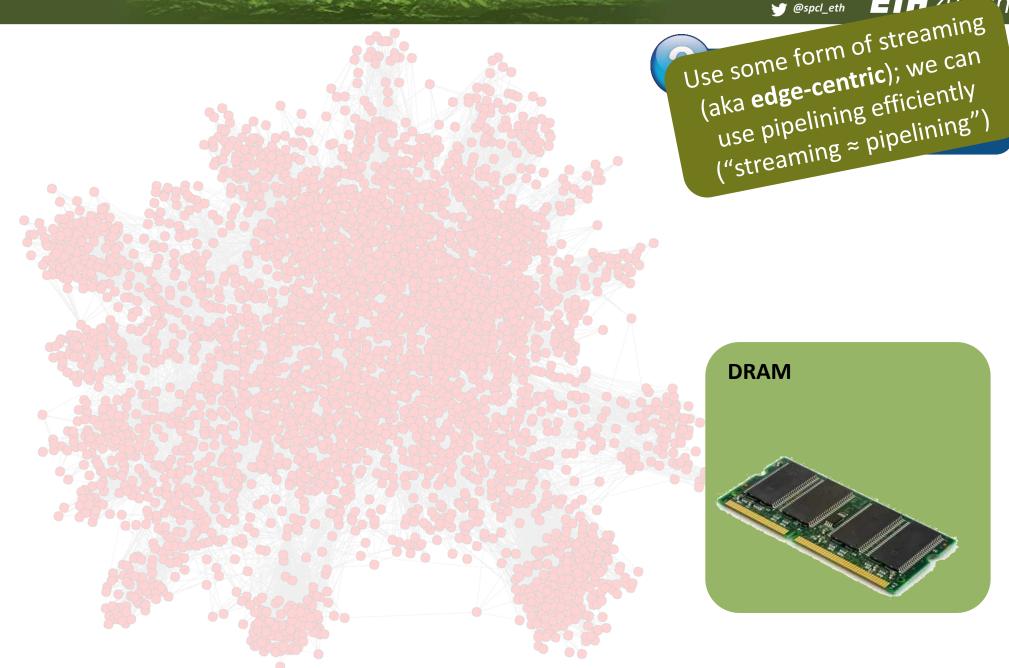








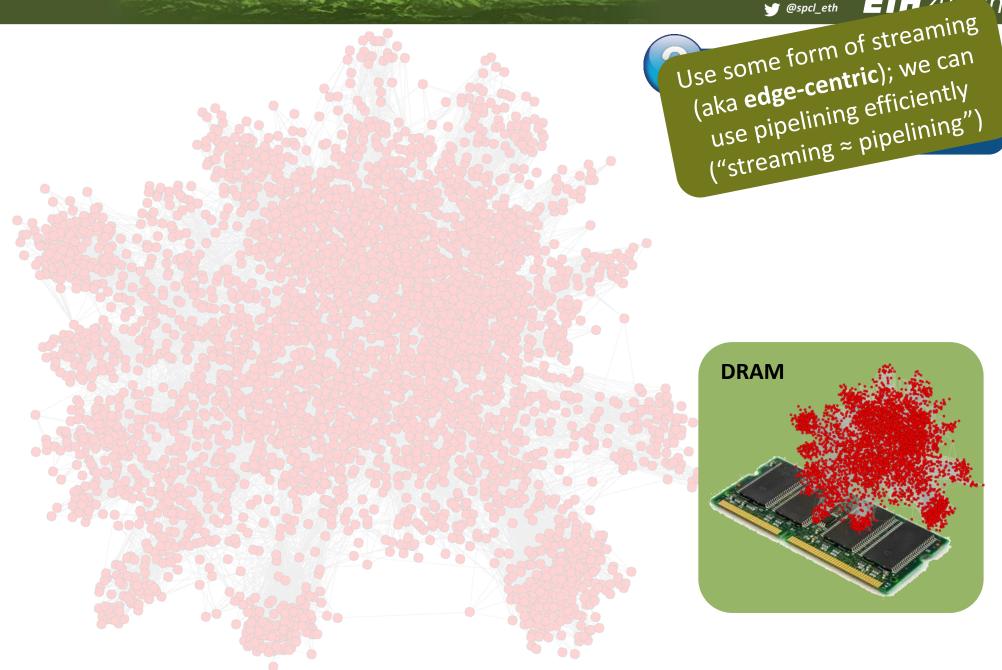














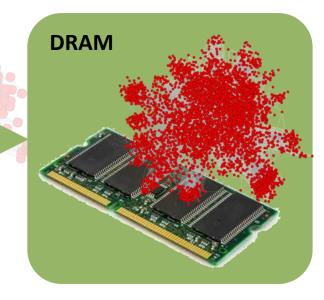














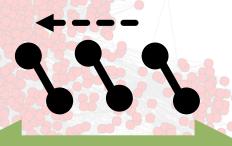


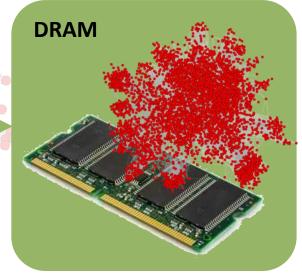
















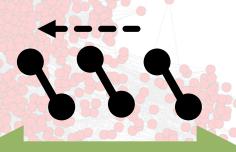


















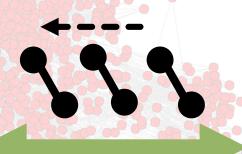


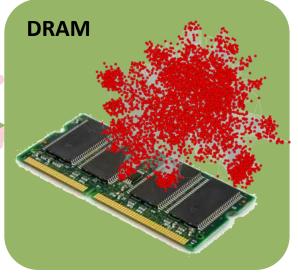
















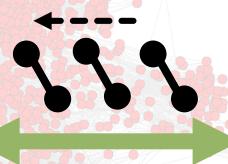




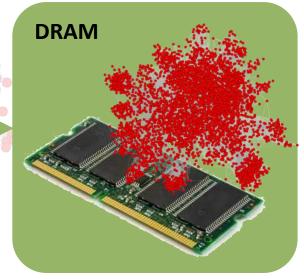


















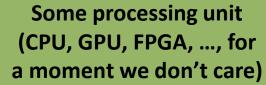








Streaming all edges in and out is one "pass". Repeat it a certain (algorithm-dependent) number of times











Issues...



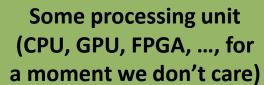




DRAM



Streaming all edges in and out is one "pass". Repeat it a certain (algorithm-dependent) number of times









DRAM



...How to minimize the number of "passes" over edges? This can get really bad in the "traditional" edge-centric approach (e.g., BFS needs D passes; D = diameter [1]).

Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges is sequential – how to incorporate parallelism?





Issues...

Streaming all edges in and out is one "pass". Repeat it a certain (algorithm-dependent) number of times

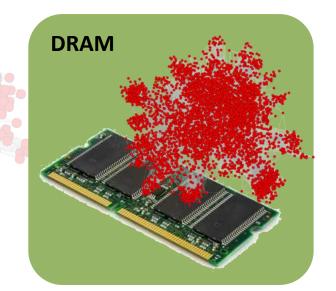






(aka edge-con use pipelining efficiently ("streaming ~ pipelining")

...Processing edges is sequential – how to incorporate parallelism?









is seq to i

Part 2: Substream-Centric: A new paradigm for processing graphs





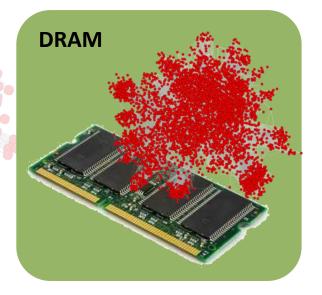


Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

...Processing edges is sequential – how to incorporate parallelism?

Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ≈ pipelining")







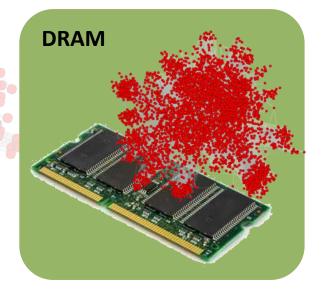


Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

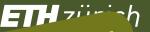
It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges is sequential – how to incorporate parallelism?







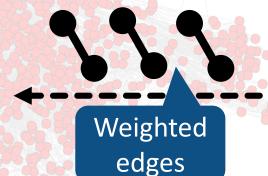


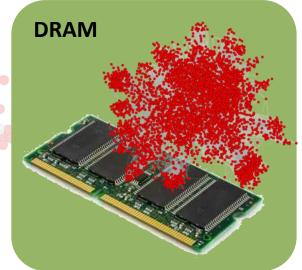
Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges
is sequential – how
to incorporate
parallelism?







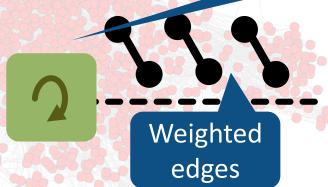




Part 2: A new paradigm for processing graphs

It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges is sequential – how to incorporate parallelism?







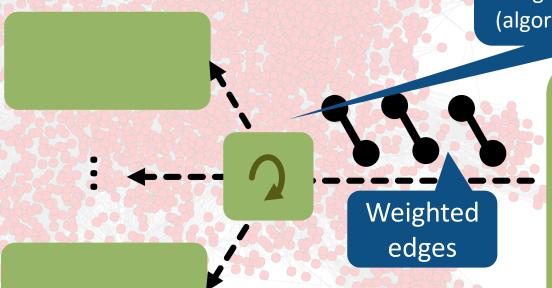


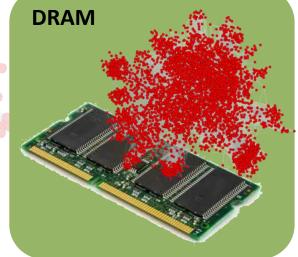


Part 2: A new paradigm for processing graphs

It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges
is sequential – how
 to incorporate
 parallelism?











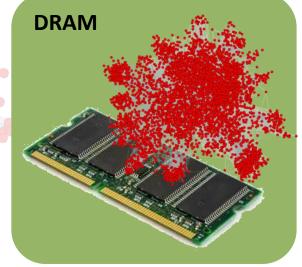
Part 2: A new paradigm for processing graphs

It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges is sequential – how to incorporate parallelism?

Process "substreams" independently







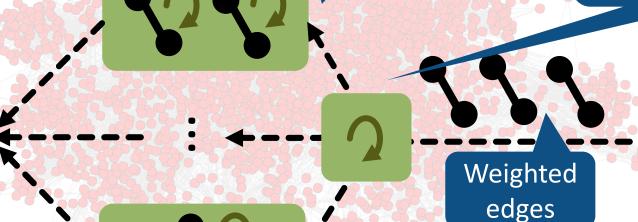


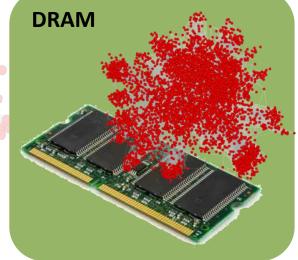
Part 2: A new paradigm for processing graphs

It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges
is sequential – how
 to incorporate
 parallelism?

Process "substreams" independently







Divide the input stream of

edges according to some

(algorithm-specific) pattern



Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges is sequential – how to incorporate parallelism?

Process "substreams" independently

3333

Weighted

Weighted edges

DRAM

Merge substreams





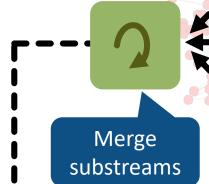
Part 2: A new paradigm for processing graphs

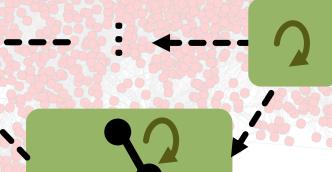
It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

...Processing edges is sequential – how to incorporate parallelism?

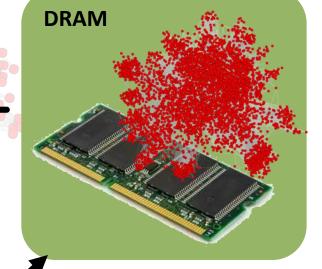
Process "substreams" independently

Divide the input stream of edges according to some (algorithm-specific) pattern





Weighted edges









Part 2: A new paradigm for processing graphs

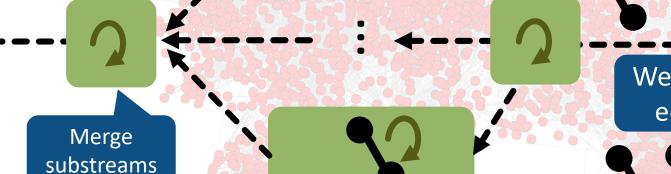
It enhances edgecentric streaming approaches Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

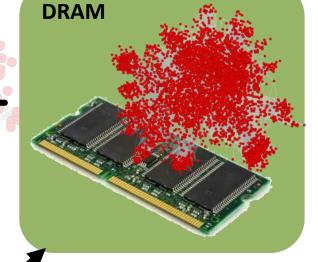
Process "substreams" independently

3232

Divide the input stream of edges according to some (algorithm-specific) pattern



Weighted edges





(aka edge-centric); we can

use pipelining efficiently

("streaming ≈ pipelining")



Substream-Centric Graph Processing

Part 2: A new paradigm for processing graphs

It enhances edgecentric streaming aches

Also, it enables (tunable) approximation and a (tunable) number of passes independently

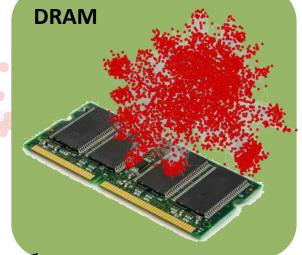
Divide the input stream of edges according to some (algorithm-specific) pattern

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

Merge substreams

Weighted edges











Part 2: A new paradigm for processing graphs

It enhances edgecentric streaming

Also, it enables (tunable)
approximation and a (tunable)
number of passes
independently

Use some form of streaming (aka edge-centric); we can use pipelining efficiently ("streaming ~ pipelining")

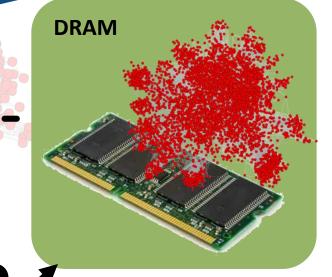
Divide the input stream of edges according to some (algorithm-specific) pattern

Substreams (pipelines) are processed in parallel, in a simple way (independently, except for merging)

3232

How to express MWM in this paradigm?

thted ges



Merge substreams







Research Questions

How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Which programming paradigm to use for (approximate) MWM (and many other problems)?

What is the HW FPGA design that ensures high performance?



What is the ultimate performance, power consumption, and the related tradeoffs?





Research Questions

How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Use <u>substream-centric</u>

<u>processing</u> (exposes parallelism,
enables easy pipelining,
supports approximation)

What is the HW FPGA design that ensures high performance?



What is the ultimate performance, power consumption, and the related tradeoffs?







Research Questions

How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Use <u>substream-centric</u>

<u>processing</u> (exposes parallelism,
enables easy pipelining,
supports approximation)

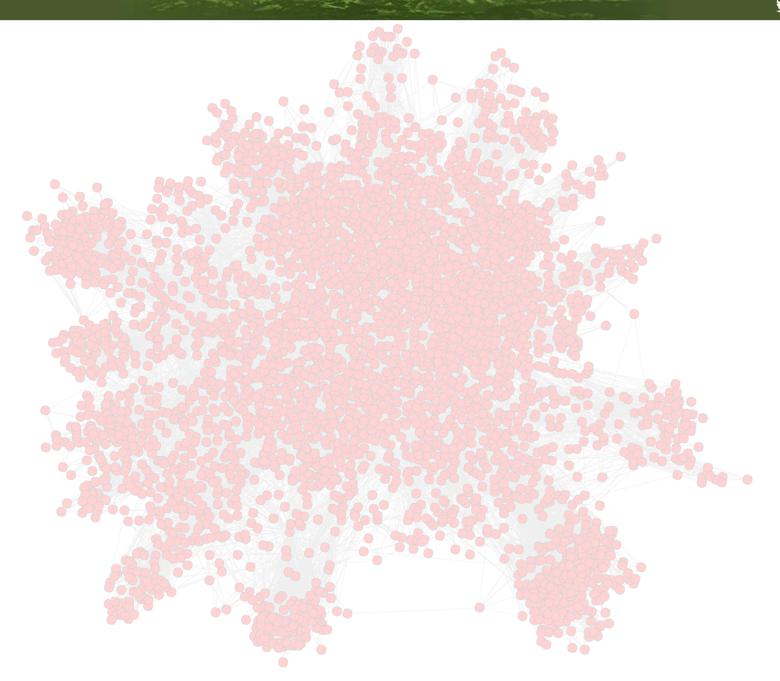
What is the HW FPGA design that ensures high performance?

?

What is the ultimate performance, power consumption, and the related tradeoffs?











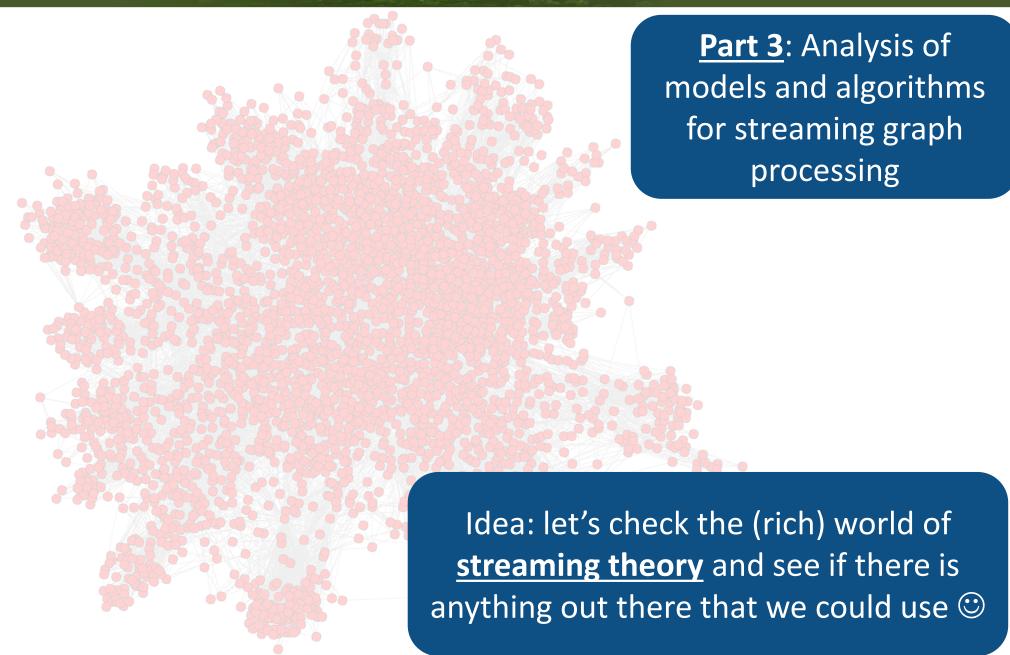






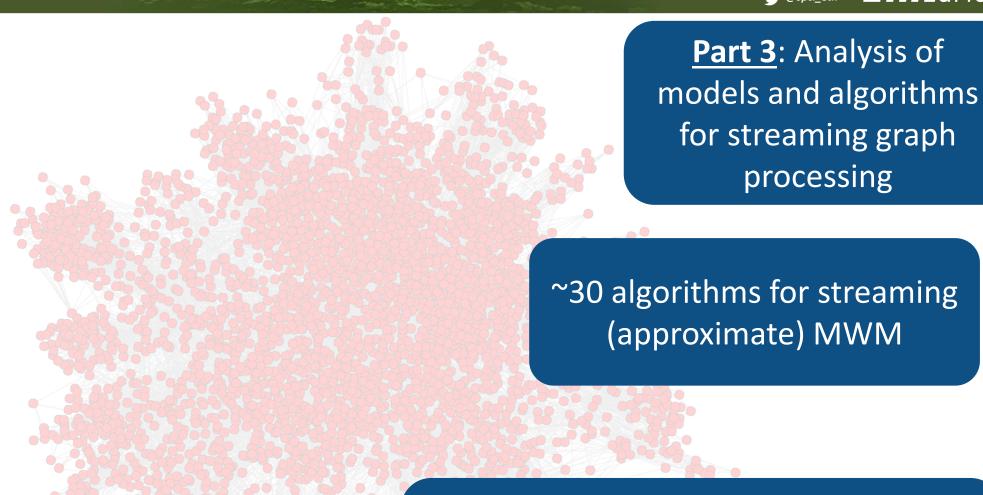












Reference	Approx.	Space	#Passes	\mathbf{Wgh}^1	Gen ²	Par ³
[26]	1/2	O(n)	1	•	Ô	:
[41, Theorem 6]	1/2 + 0.0071	$O(n \operatorname{polylog}(n))$	2	•	Ô	•
[41, Theorem 2]	1/2 + 0.003	O(n polylog(n))	1	•	<u>C</u>	i de
[36, Theorem 1.1] [26, Theorem 1]	$O(\operatorname{polylog}(n))$ 2/3 – ε	$O(\operatorname{polylog}(n))$ $O(n \log n)$	$ \begin{array}{c} 1 \\ O\left(\log\left(1/\varepsilon\right)/\varepsilon\right) \end{array} $	* F		<u>^</u>
[6, Theorem 19]	$1-\varepsilon$	$O\left(n \operatorname{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	•	•	:•
[41, Theorem 5]	1/2 + 0.019	O(n polylog(n))	2	•		
[41, Theorem 1]		$O(n \log n)$	1	•	•	
[41, Theorem 4]	_	O(n polylog(n))	2	•	•	:
[39]	1 - 1/e	$O(n \operatorname{polylog}(n))$	1	•	•	ı 🛊
[28, Theorem 20]	1 - 1/e	O(n)	1	*	*	1
[35, Theorem 2]	$1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	O(n)	k	•	•	:
[14]	1	$\tilde{O}\left(k^2\right)$	1	•	Ô	Ô
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	•	Ô	Ô
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	•	•	Ô
[26, Theorem 2]	6	$O(n \log n)$	1	Ô	Ô	0
[44, Theorem 3]	$2+\varepsilon$	$O(n \operatorname{polylog}(n))$	O(1)	©	Ģ	0
[44, Theorem 3]	5.82	O(n polylog(n))	1			0
[63] [25]	5.58 $4.911 + \varepsilon$	O(n polylog(n)) O(n polylog(n))	1	2		?
[29]	$3.5 + \varepsilon$	$O(n \operatorname{polylog}(n))$	1	<u> </u>	000000	0
[53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1	Ô	Ò	8
[27]	$2 + \varepsilon$	$O(n \log n)$	1	Ô	Ô	•
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	Ô	Ô	8
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$	Ů	Ô	:
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O(n\log n)$ $O\left(n\log(n)/\varepsilon^4\right)$ $O\left(n\left(\frac{\varepsilon\log n - \log \varepsilon}{\varepsilon^2}\right)\right)$		Ô	Ô	:•
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ů	•	:•
Crouch and Stubbs	[1] &	$O(n \operatorname{polylog}(n))$	1	Ô	Ô	Ô

~30 algorithms for streaming (approximate) MWM







Reference	Approx.	Space	#Passes	\mathbf{Wgh}^1	Gen ²	Par ³
[26]	1/2	O(n)	1	19	Ď	100
[41, Theorem 6]	1/2 + 0.0071	O(n polylog(n))	2	100		E (1)
[41, Theorem 2] [36, Theorem 1.1]	1/2 + 0.003	O(n polylog(n))	1	10		3
[26, Theorem 1]		$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	100		
[6, Theorem 19]	$1-\varepsilon$	$O(n \text{ polylog}(n)/\varepsilon^2)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	H.	RIP.	16
[41, Theorem 5]	1/2 + 0.019	O(n polylog(n))	2	100	100	
[41, Theorem 1]	$1/2 + 0.005^*$	$O(n \log n)$	1	H.	RIP.	100
[41, Theorem 4]	$1/2 + 0.0071^*$	O(n polylog(n))	2		R. C.	100
[39] [28, Theorem 20]	1 - 1/e 1 - 1/e	O(n polylog(n)) O(n)	1	16	s de	
	$1-1/e$ $e^{-k}k^{k-1}$		1.			
[35, Theorem 2]	$1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	O(n)	K		s.	B
[14]	1	$\tilde{O}\left(k^2\right)$	1	1	O	
[14]	$1/\epsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	R. C.	0	O
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	R. C.	R. P.	6
[26, Theorem 2]	6	$O(n\log n)$	1	3	3	
[44, Theorem 3]	$2 + \varepsilon$	O(n polylog(n))	O(1)		0	
[44, Theorem 3]	5.82	O(n polylog(n))	1			
[63] [25]	5.58 $4.911 + \varepsilon$	O(n polylog(n)) O(n polylog(n))	1			
[29]	$3.5 + \varepsilon$	O(n polylog(n))	1	3	3	
[53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1	3	3	
[27]	$2+\varepsilon$		1	3	3	
[26, Section 3.2]	$_{2+\varepsilon}$ IVO V	worries, no	need to	3	6	
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$ ana	lyze it her	e all the	ß	0	B .
[6, Theorem 22]	1	$O\left(n\left(\frac{\varepsilon \log n - \log \varepsilon}{1}\right)\right)$	$O(\varepsilon^{-2}\log(\varepsilon^{-1}))$	ß	O	B .
[6, Theorem 22]	detai	ls are in th	e paper 😊	Ô	r ip	:6
Crouch and Stubbs	[1] ε	O(n polylog(n))	1	ß	ß	ß

~30 algorithms for streaming (approximate) MWM







Reference	Approx.	Space	#Passes	\mathbf{Wgh}^1	Gen ²	Par ³
	.0071	O(n)	1 2	ile ile		: ()
Most	.0071	O(n polylog(n)) O(n polylog(n))	1	16	6	
importa	(()	$O(\operatorname{polylog}(n))$	1	R PR	3	3
•		$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	1 Park	R. C.	10
goals	:	$O\left(n \operatorname{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	N. C.		8
	J.019	O(n polylog(n))	2			84
[41, Theorem 1]	1/2 + 0.005	$O(n \log n)$	1			
[41, Theorem 4] [39]	1/2 + 0.0071 1 - 1/e	O(n polylog(n)) O(n polylog(n))	1	100	nde nde	
[28, Theorem 20]	1 - 1/e	O(n)	i	R P	1	
[35, Theorem 2]	$1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	O(n)	k	rip.	N. P.	10
[14]	1	$\tilde{O}\left(k^2\right)$	1	i q	ß	3
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	i p	ß	ß
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	nde.	nda.	ß
[26, Theorem 2]	6	$O(n \log n)$	1		3	
[44, Theorem 3]	$2+\varepsilon$	O(n polylog(n))	O(1)			
[44, Theorem 3] [63]	5.82 5.58	O(n polylog(n)) O(n polylog(n))	1	3		
[25]	$4.911 + \varepsilon$	O(n polylog(n)) O(n polylog(n))	1		3	
[29]	$3.5 + \varepsilon$	O(n polylog(n))	1	3	3	
[53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1		3	
[27]	$2+\varepsilon$	$O(n\log n)$	1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	6	3	
[26, Section 3.2]	$_{2+\varepsilon}$ IVO	worries, no	o need to	3	6	
[6, Theorem 28]	$\frac{1}{1-\epsilon}$ ana	alyze it her	e all the	3	6	100
[6, Theorem 22]		$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{2}\right)\right)$	$O(\varepsilon^{-2}\log(\varepsilon^{-1}))$	ß	Ů	n dr
[6, Theorem 22]	3 detai	Is are in th	e paper 😂	ß	n q	16
Crouch and Stubbs	[1] 8	$O(n \operatorname{polylog}(n))$	1	ß	ß	3

~30 algorithms for streaming (approximate) MWM







				-		
Reference	Approx.	Space	#Passes	\mathbf{Wgh}^1	Gen ²	Par ³
	1 (0	O(n)	1	16		100
Most	.0 (1	O(n polylog(n)) O(n polylog(n))	1	16	6	
importa	ant ^{og}	$O(\operatorname{polylog}(n))$	1		3	O
		$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	10	16	
goals	1.01	$O(n \text{ polylog}(n)/\varepsilon^2)$ O(n polylog(n))	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	10	10	
[41, Theorem	1 /0 + 0.00	$(n \operatorname{polylog}(n))$	1	R PA	N. C.	16
[41, Theorem	Maximi:	$ze \frac{ y \log(n)}{n}$	2			10
[39] [28, Theorem	accurac	SV	1	ide ide	16	
[35, Theorem 2]	$1 - \frac{k}{(k-1)!}$	O(n)	k	19	N.	10
[14]	1	$\tilde{O}(k^2)$	1	19	ß	6
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	19	ß	6
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	n il	rip.	ß
[26, Theorem 2]	6	$O(n \log n)$	1	6	6	
[44, Theorem 3]	$2 + \varepsilon$ 5.82	O(n polylog(n)) O(n polylog(n))	O(1)			
[44, Theorem 3] [63]	5.58	O(n polylog(n)) O(n polylog(n))	1		3	
[25]	$4.911 + \varepsilon$	O(n polylog(n))	1	6		
[29] [53]	$3.5 + \varepsilon$ $2 + \varepsilon$	O(n polylog(n)) $O\left(n \log^2 n\right)$	1	6	<u>-</u>	
[27]			1	<u></u>	3	
[26, Section 3.2]	$_{2+\varepsilon}^{-1}$ No	worries, no	need to	3	6	
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$ and	alyze it her	e. all the	Ď	ß	1
[6, Theorem 22]		$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	3	O	n d r
[6, Theorem 22]	a deta	ils are in th	e paper 🕲	ß	n il	16
Crouch and Stubbs	[1] ε	O(n polylog(n))	1	ß	ß	Ô

~30 algorithms for streaming (approximate) MWM







		•	#P	1		2 - 3
Reference	Approx.	Space	#Passes	Wgh	Gen	² Par ³
	1./0	O(n)	1	100		10
Most	.0 7	O(n polylog(n))		1		10
	.0	O(n poly(g(n))) O(polylog(n))) 1	100	3	
importa	ant 📴	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	100		
goals		O(n polylog)	$/\varepsilon^2$) $O(\log\log(1/\varepsilon)/\varepsilon$	2)	RIP.	100
guais	1.01	(n polylog)	2	100	RIM.	100
[41, Theorem 1]	1/0 + 0.00	(n polying(n polying	1	R.	R PA	
[41, Theorem	Maxim	$ize \frac{1}{ y \log(n)}$	2	19	R. C.	
[39]		$\operatorname{lylog}(n)$	1	1		10
[28, Theorem	accura	Су	Minimize	100	1 de	100
[35, Theorem 2]	$1 - \frac{k}{(k-1)!}$	- $O(n)$	local space	1	a de	
[14]	1	$\tilde{O}\left(k^2\right)$	local space	nde.	O	O
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	H.	ß	ß
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	$-\varepsilon$) 1	H.	R. C.	ß
[26, Theorem 2]	6	$O(n \log n)$	1	3	3	
[44, Theorem 3]	$2 + \varepsilon$	O(n polylog(n))	O(1)			
[44, Theorem 3]	5.82	O(n polylog(n))		Ď		
[63] [25]	5.58 $4.911 + \varepsilon$	O(n polylog(n)) O(n polylog(n))		6	3	
[29]	$3.5 + \varepsilon$	O(n polylog(n))		6	3	
[53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1		3	
[27]	$2+\varepsilon$		1	3	3	
[26, Section 3.2]	$_{2+\varepsilon}$ IVO	worries,	no need to	O	ß	
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$ ar	alvze it h	ere, all the	O	ß	:4
[6, Theorem 22]	1	$O(n(\frac{\epsilon \log n - 10}{n}))$	$\log \varepsilon$)) $O(\varepsilon^{-2} \log (\varepsilon^{-1})$. 0	ß	10
[6, Theorem 22]	adeta	alis are in	the paper ©		s de	:6
Crouch and Stubbs	[1] ε	O(n polylog(n))) 1	ß	ß	6

~30 algorithms for streaming (approximate) MWM







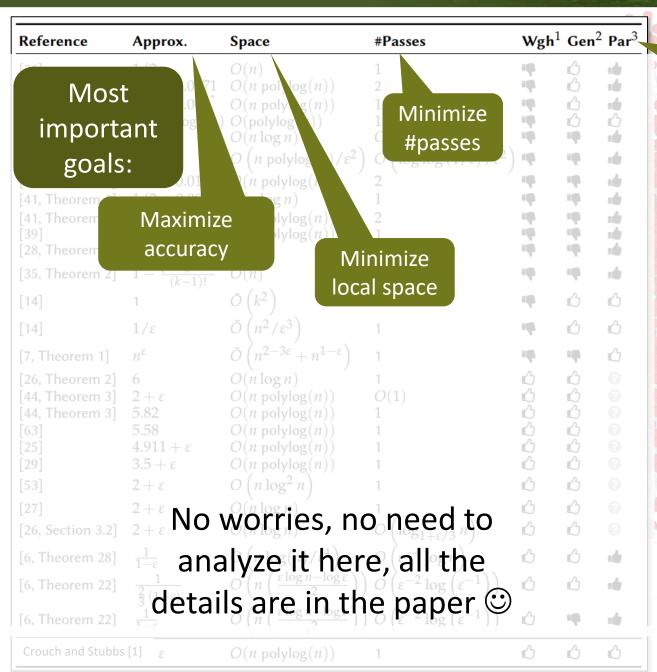
Reference	Approx.	Space	#Passes	Wgh	1 Gen 2	² Par ³
	1 (0	O(n) 1 $O(n polylog(n))$	1	14		: dr
Most	0.0	O(n poly) g(n		R. C.	6	10
importa	ant 🥫	$O(\operatorname{polylog} O(n \log n))$	1	100		
goals		O(n polylog)	(2) #passes		N/A	100
	0.01	(n polylog)	2	nda.	H.	B. (1)
[41, Theorem [41, Theorem	Maxim	g(n) $g(n)$ $g(n)$	1 2	10	140	
[39]	Maxim	$\operatorname{lylog}(\eta)$		RIP.	R I	100
[28, Theorem	accura	СУ	Minimize	1	-	16
[35, Theorem 2]	$1 - \frac{k}{(k-1)!}$	- O(n)	local space	ndv.		
[14]	1	$O(k^2)$		n die		0
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	ada.		Ď
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n\right)$	$^{1-\varepsilon}$) 1	nde.	E I	
[26, Theorem 2] [44, Theorem 3]	6 2 + ε	$O(n \log n)$ $O(n \operatorname{polylog}(n)$	1 (1) O(1)			
[44, Theorem 3]	5.82	O(n polylog(n))		6	6	
[63] [25]	5.58 $4.911 + \varepsilon$	O(n polylog(n O(n polylog(n O(n polylog(n O(n O(n O(n O(n O(n O(n O(n O(n O(n O				
[29]	$3.5+\varepsilon$	O(n polylog(n))		6	6	
[53]	$2 + \varepsilon$	$O\left(n\log^2 n\right)$	1			
[27]	$^{2+\varepsilon}$ No	worries	s, no need to	Ď		
[26, Section 3.2]	_ , .	(110811)	(1081+8/31)			
[6, Theorem 28]	1 an	alyze it	here, all the			
[6, Theorem 22]	1	$O(n(\frac{\epsilon \log n}{n}))$	$\frac{-\log \varepsilon}{2}$) O ($\varepsilon^{-2}\log(\varepsilon^{-1})$)	O		
[6, Theorem 22]	deta	ilis are il	n the paper ${\mathfrak S}$) ()	H.	16
Crouch and Stubbs	[1] ε	O(n polylog(n))	1)) 1	ß	ß	ß

~30 algorithms for streaming (approximate) MWM









Expose parallelism (match substream-centric)

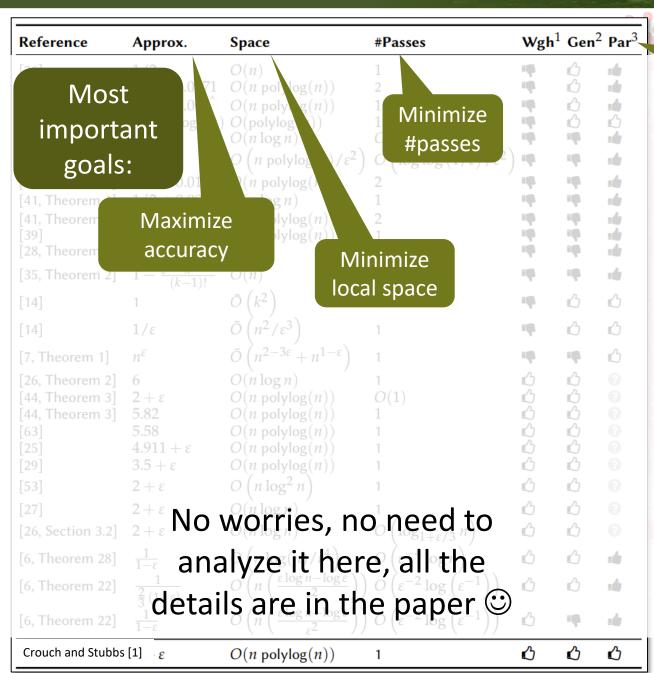
Part 3: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM









Expose parallelism (match substream-centric)

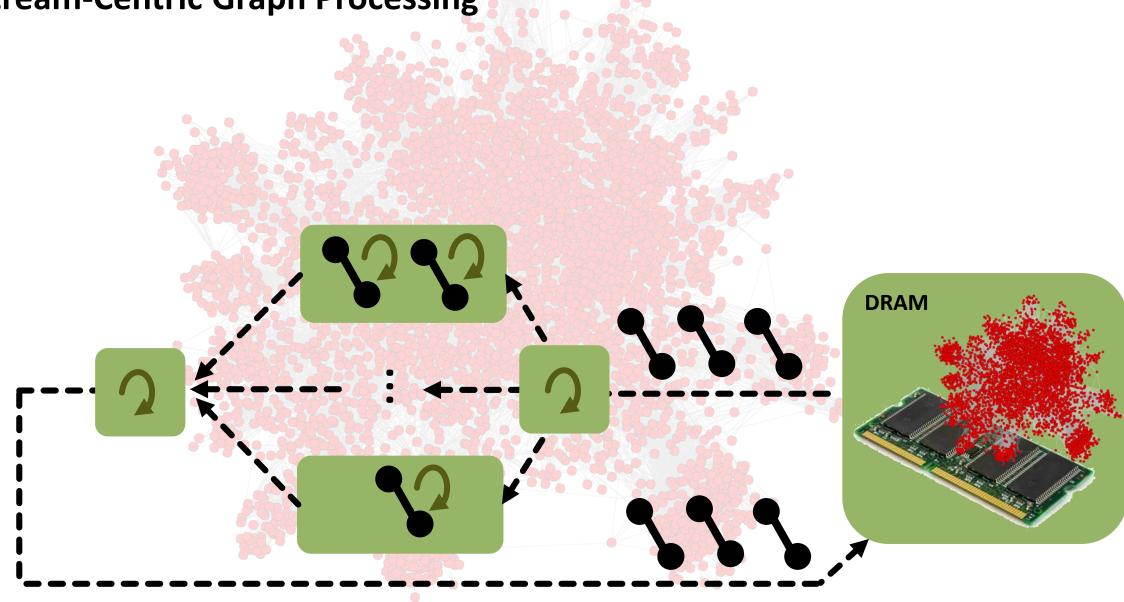
Part 3: Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM





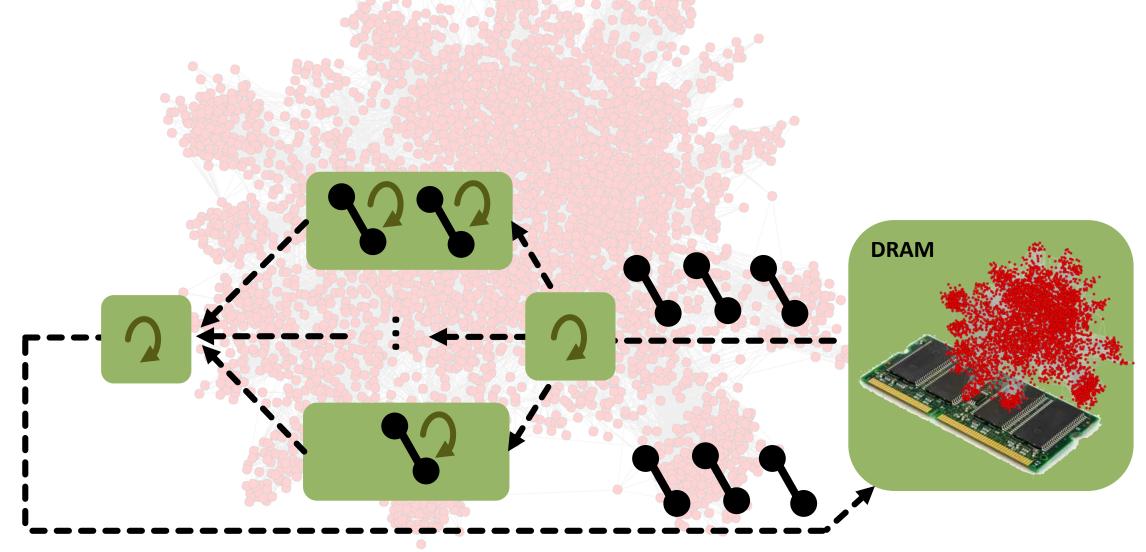








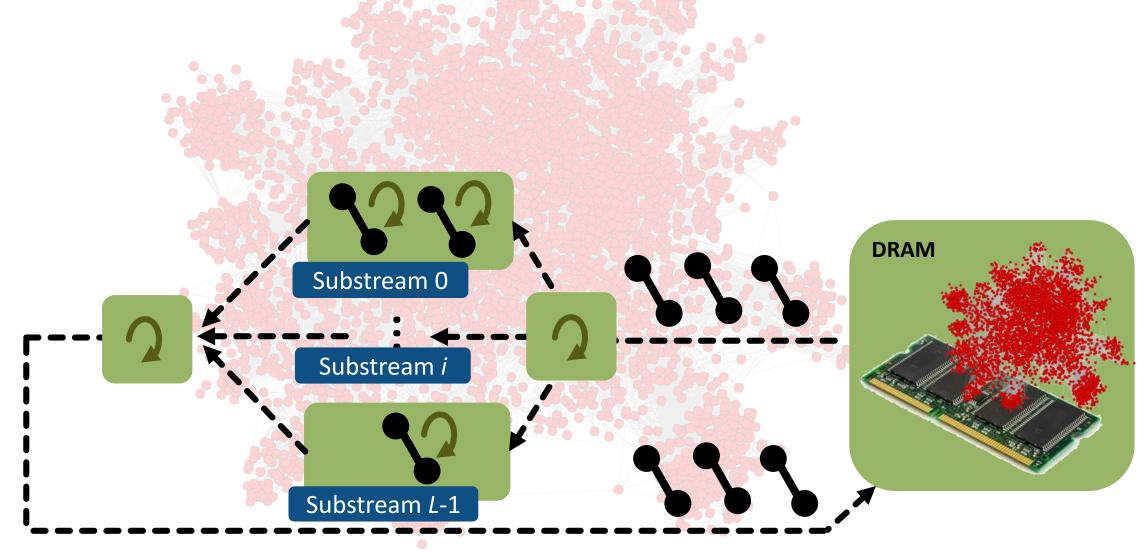








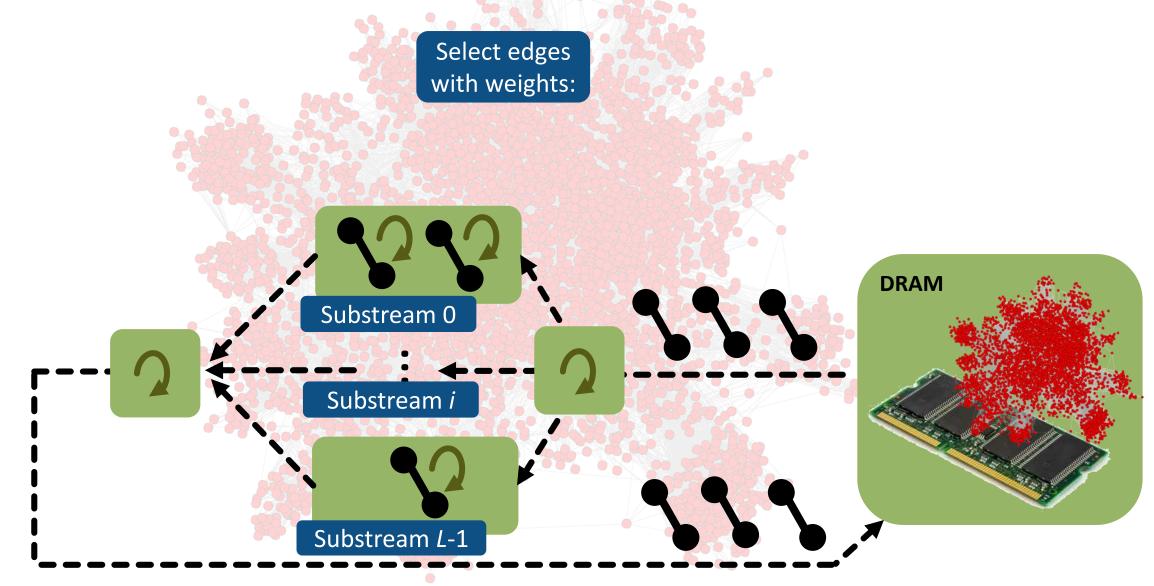








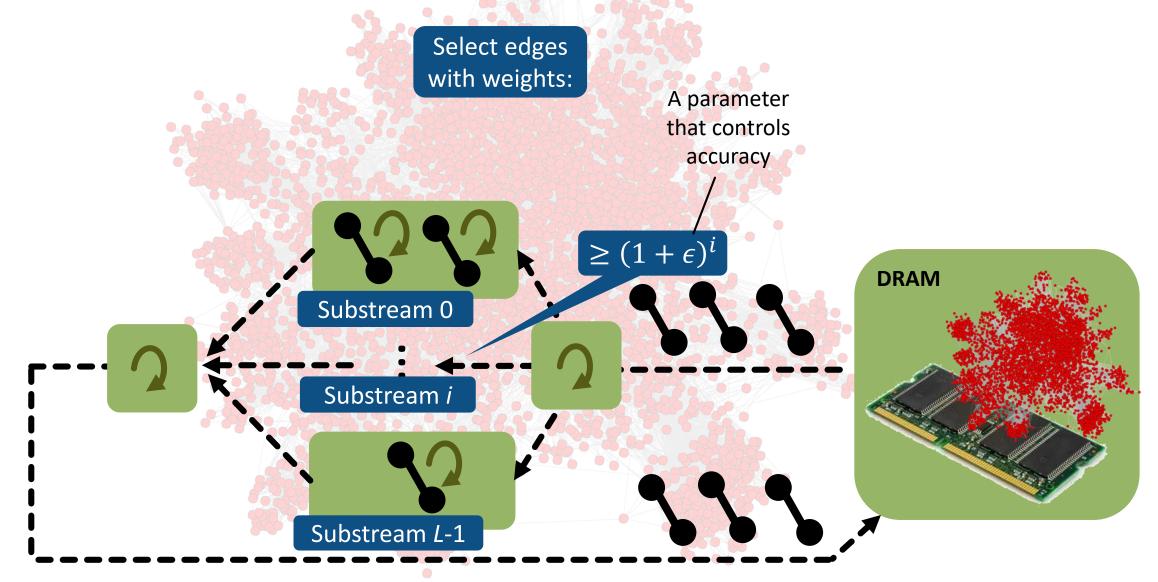






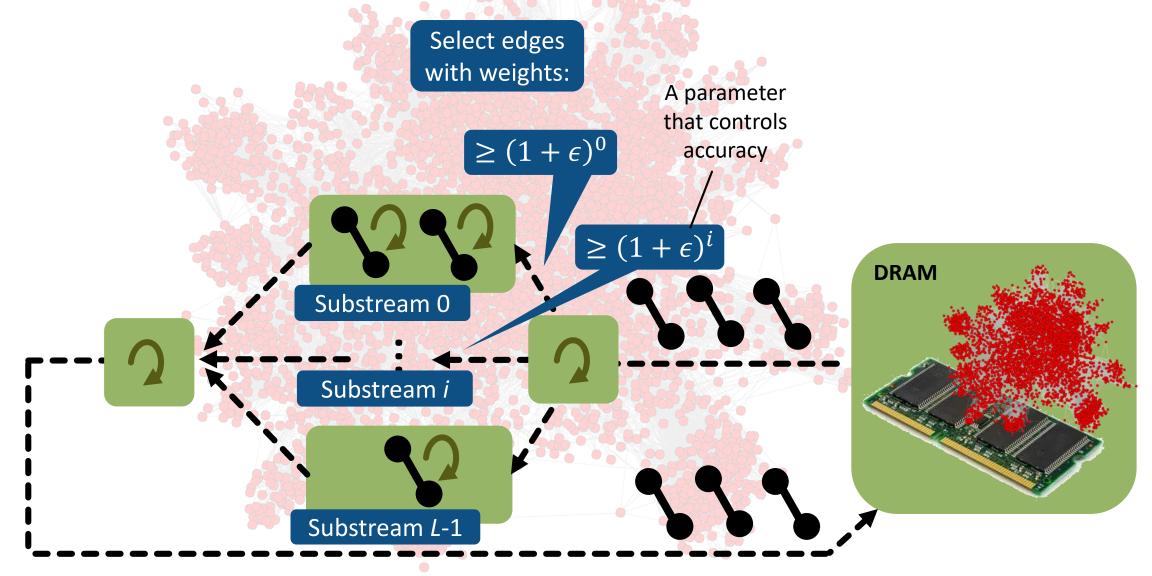






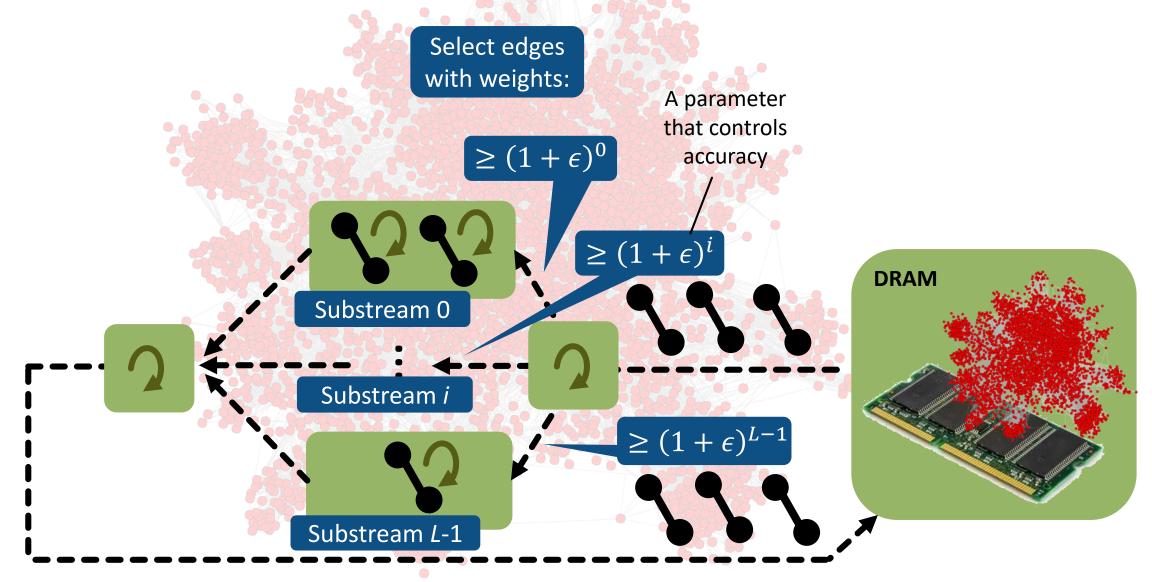






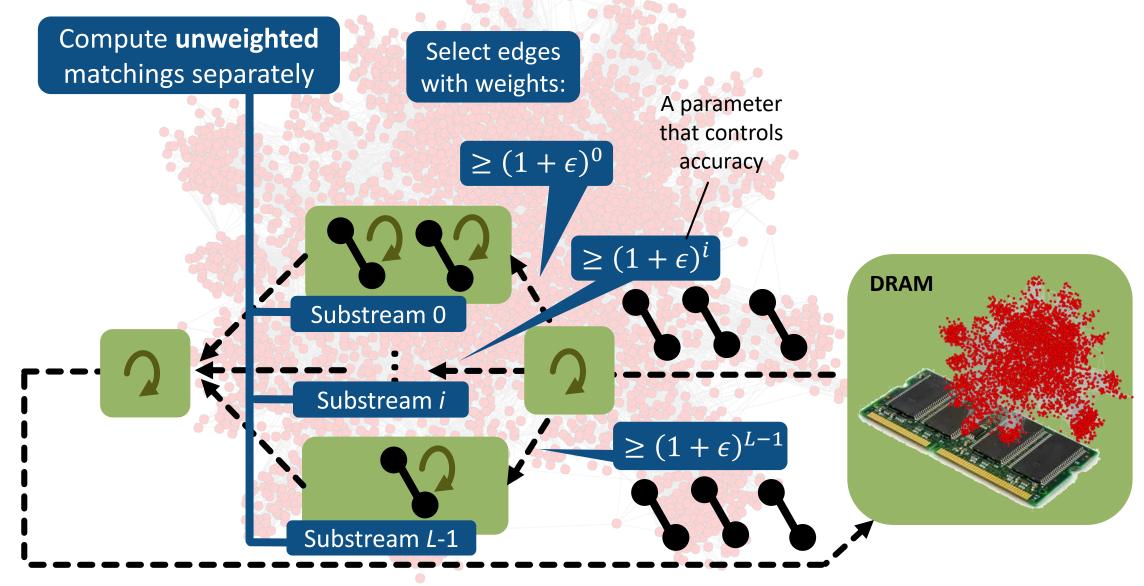






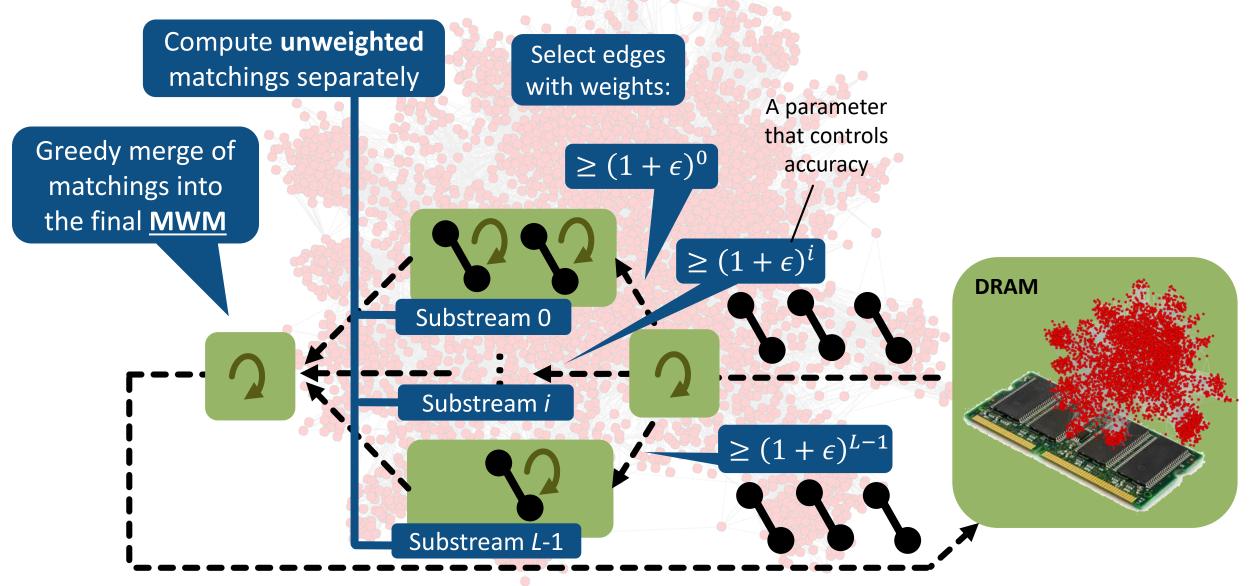








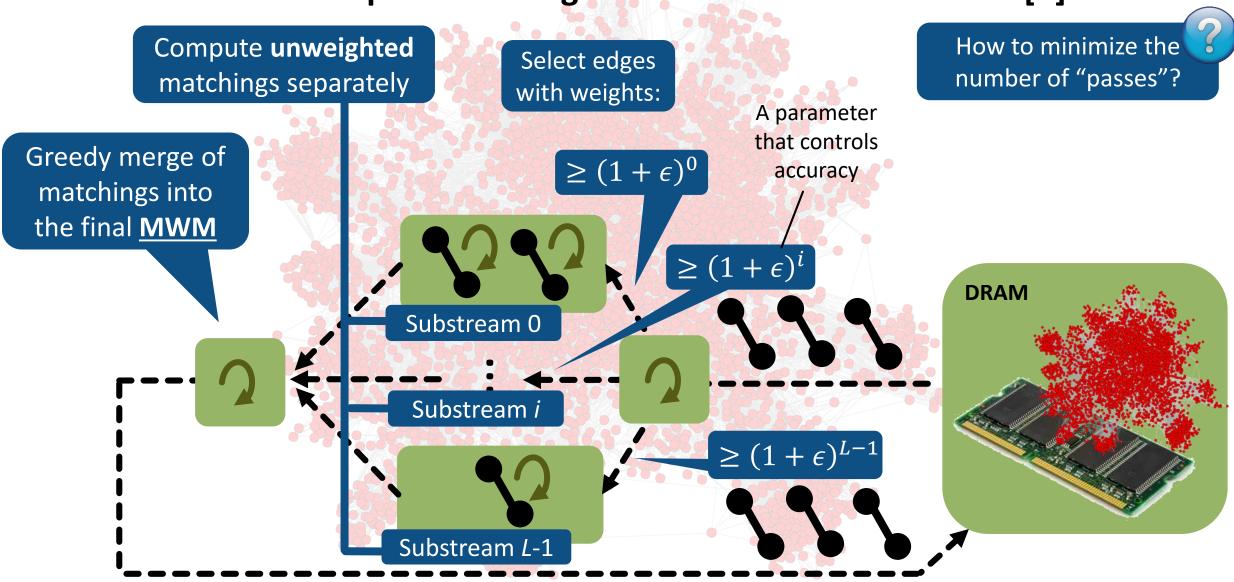








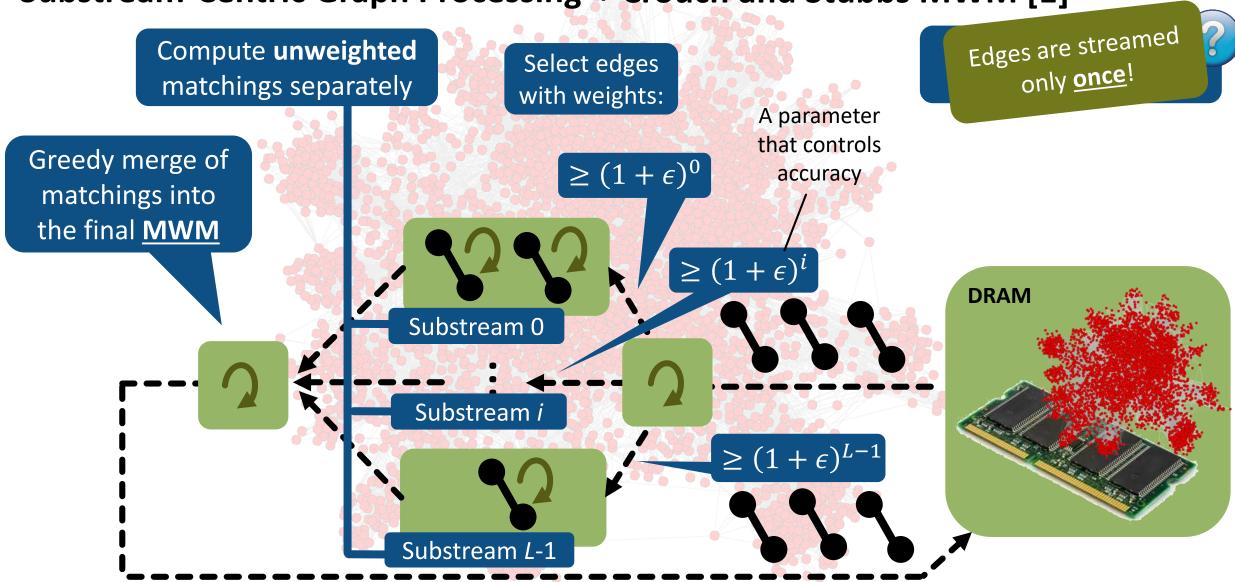








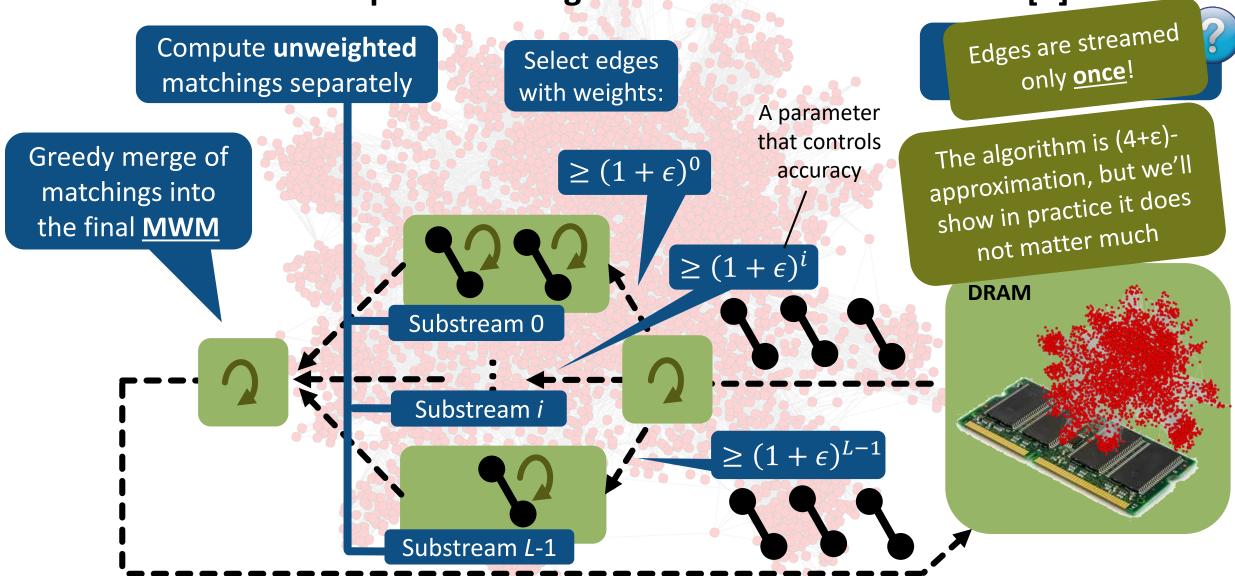








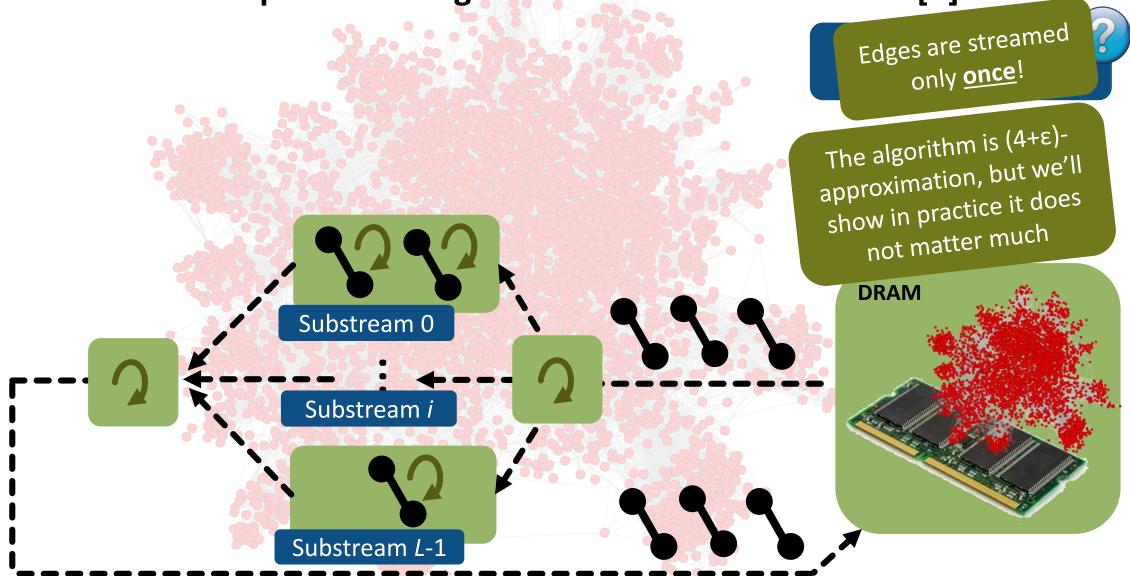














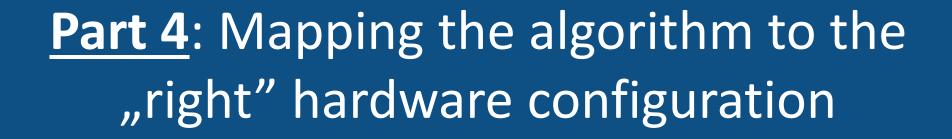




oes

Substream-Centric Graph Processing + Crouch and Stubbs MWM [1]

Edges are streamed only <u>once</u>!

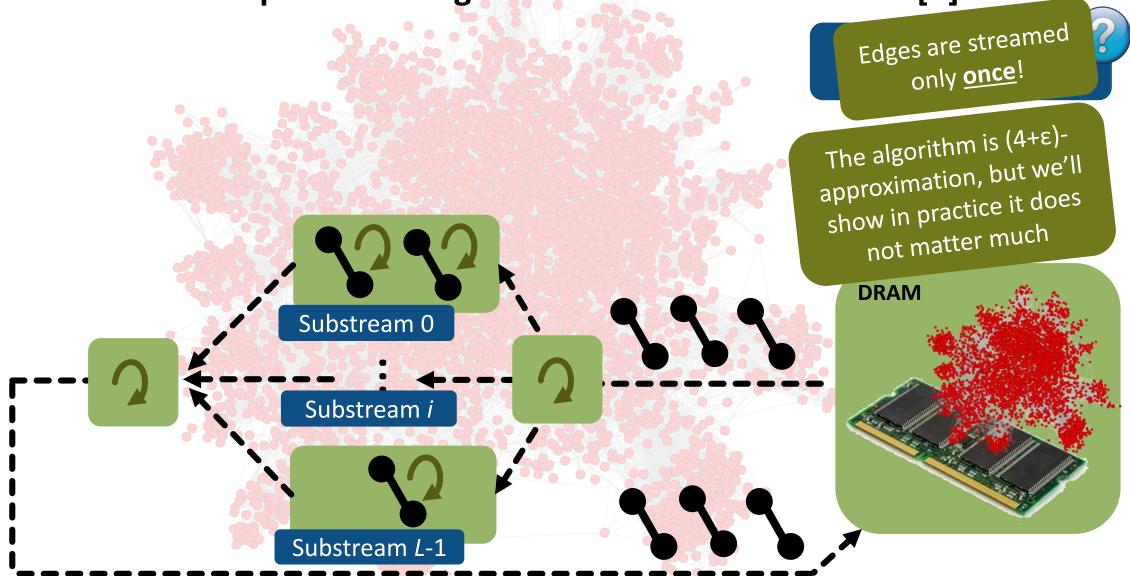








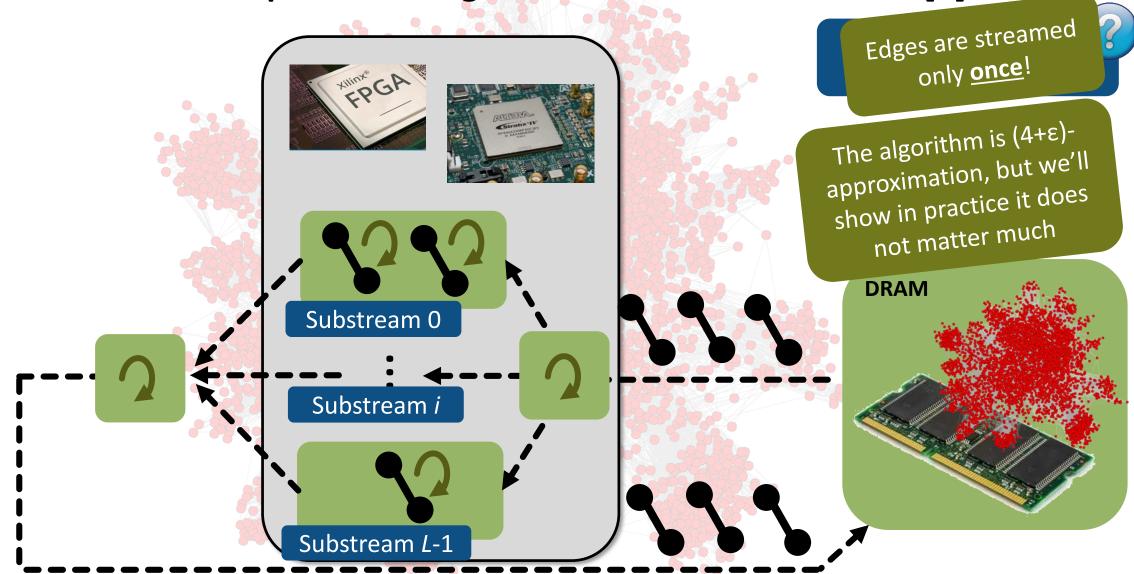








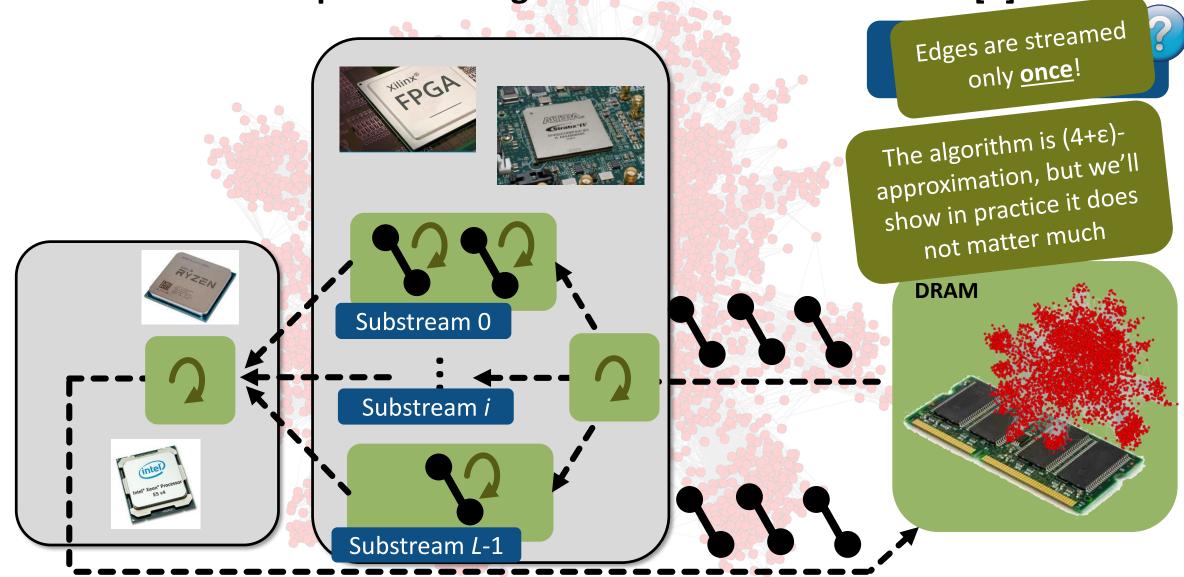








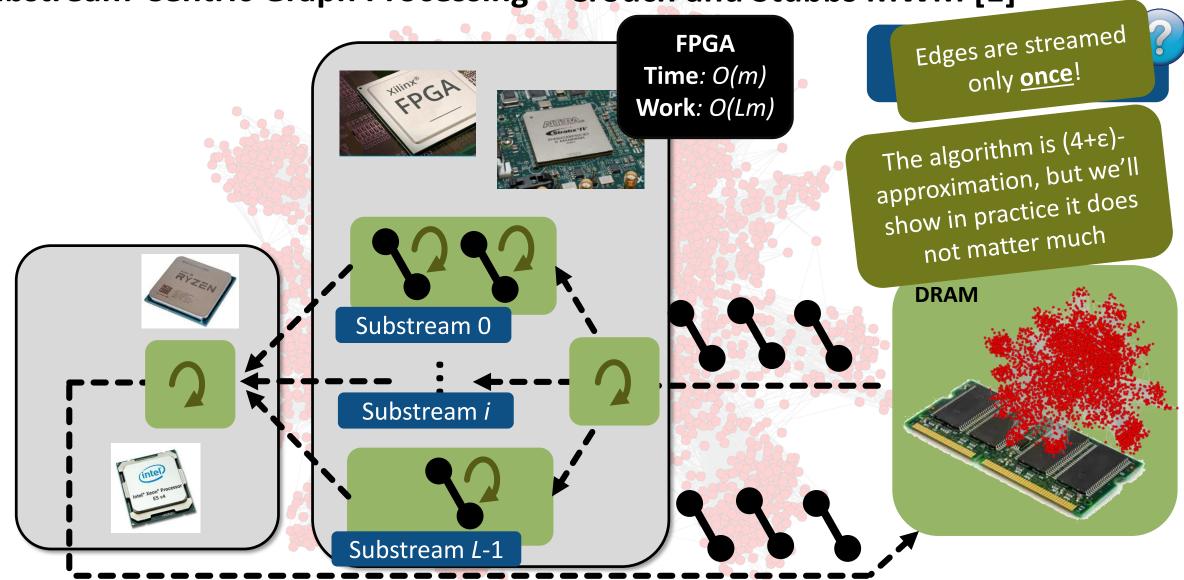








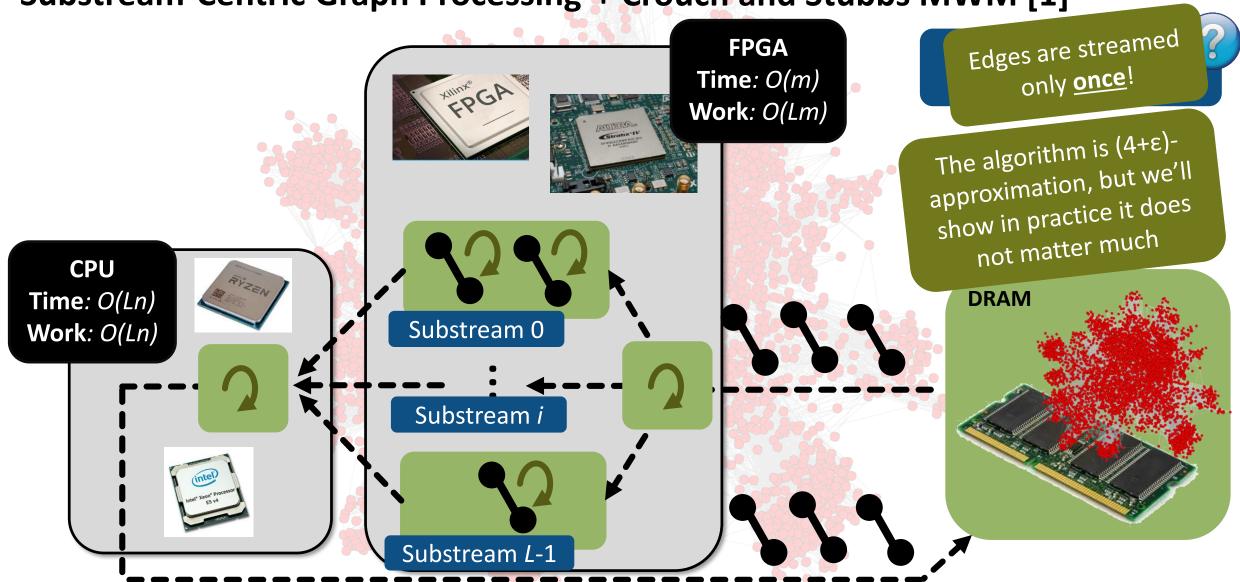








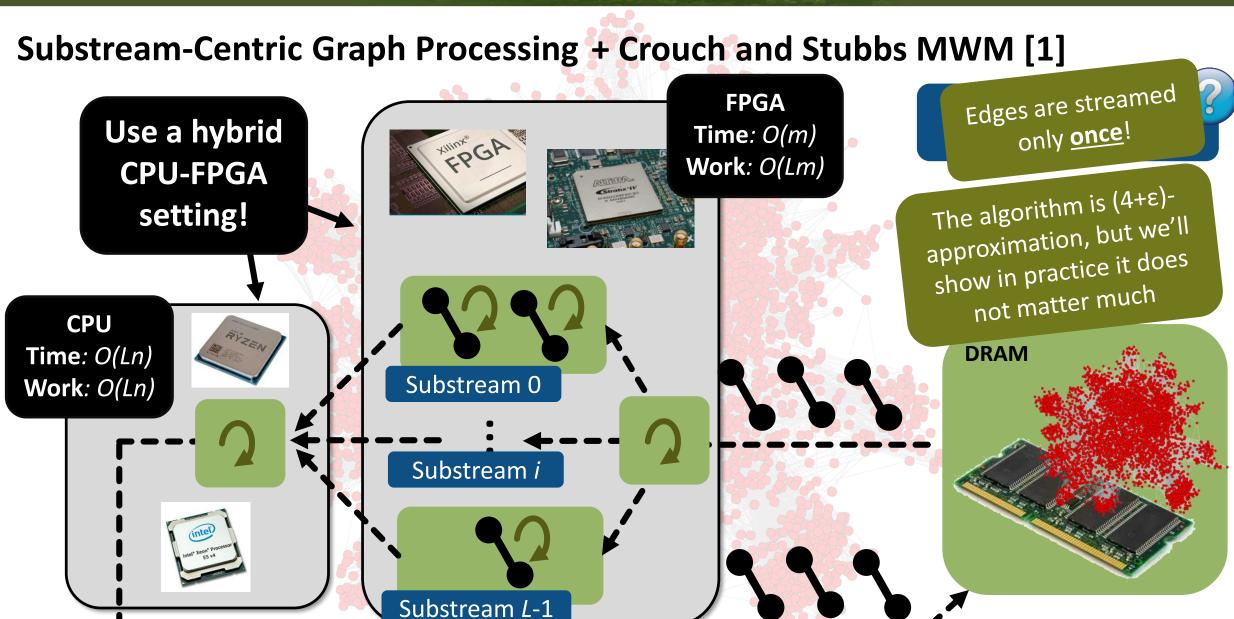








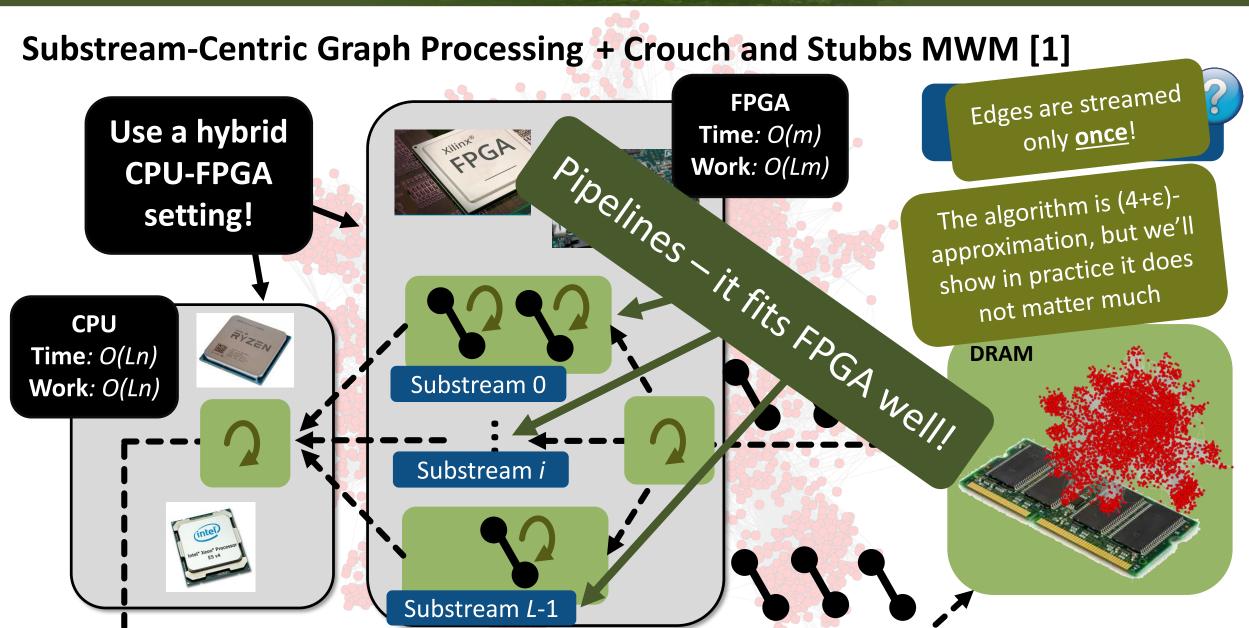








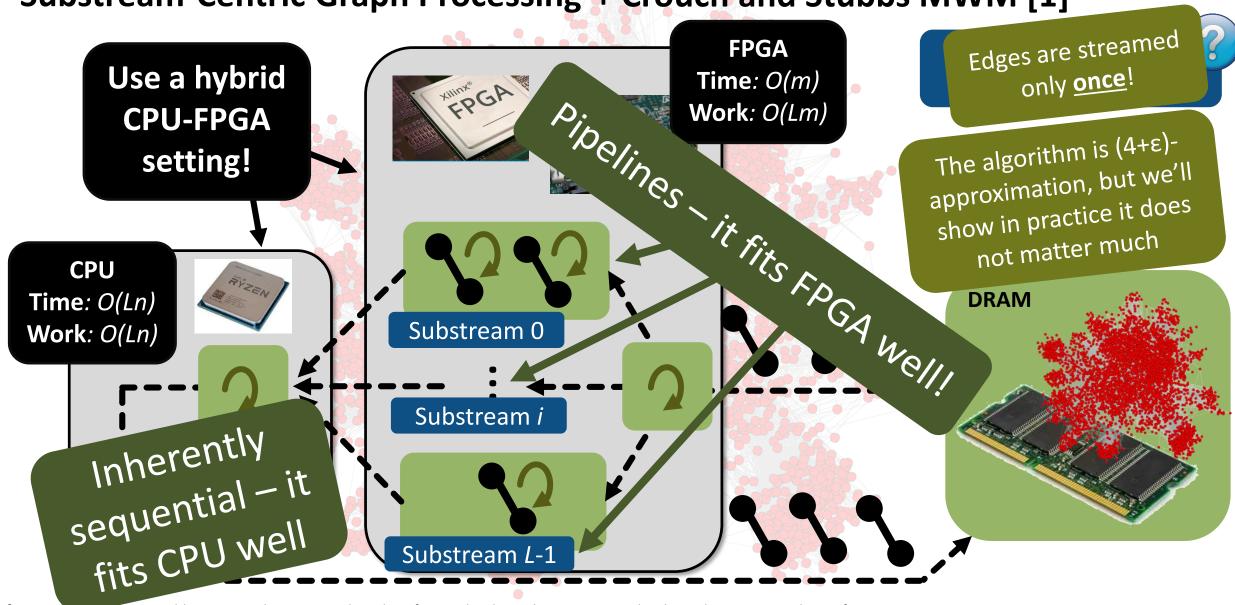






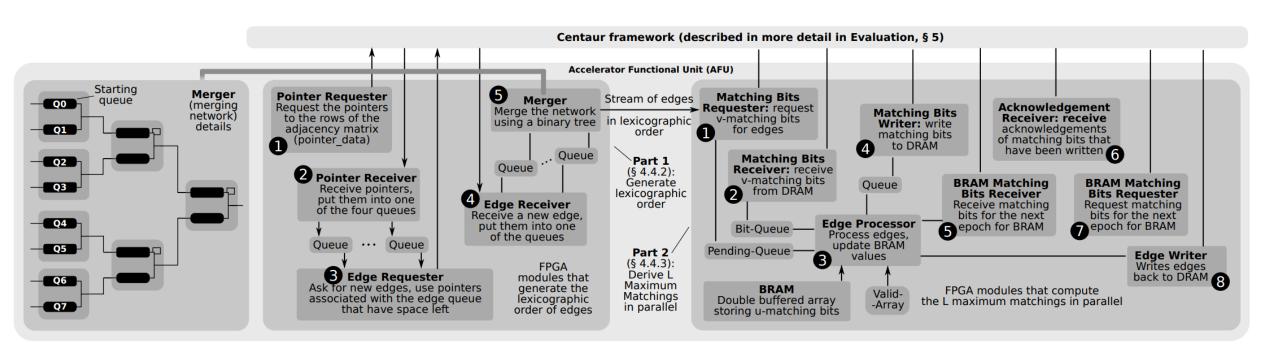








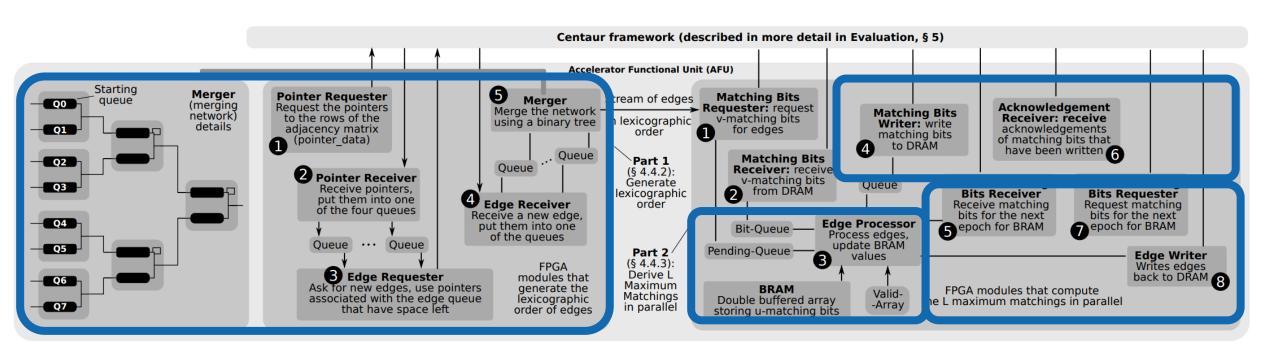








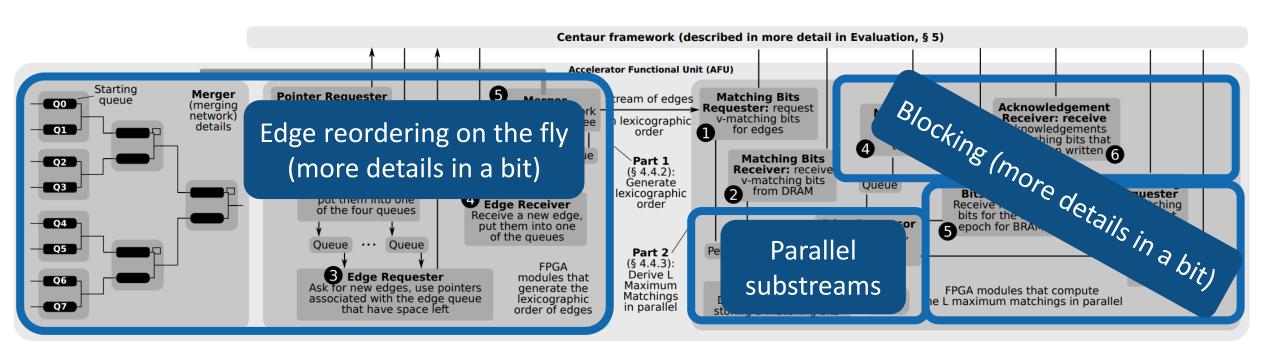








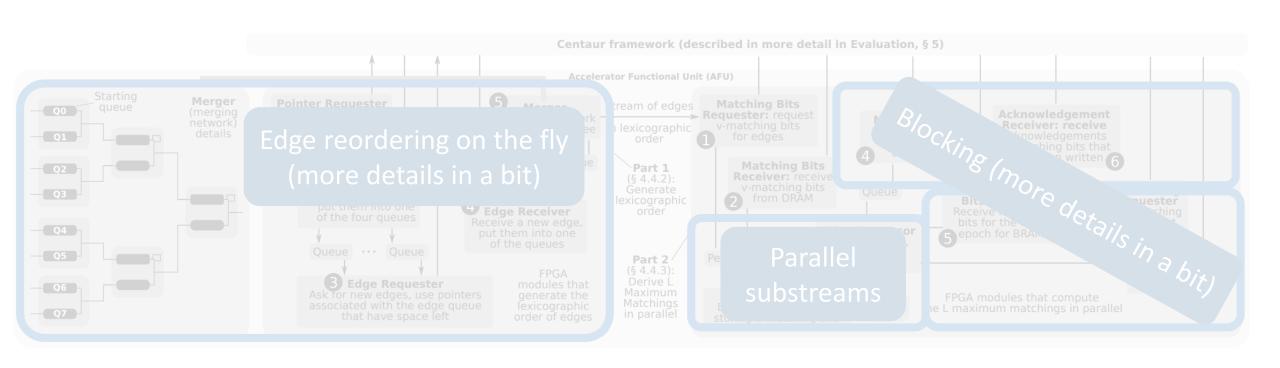


























Blocking



Prefetching

Pipelining







Blocking



Prefetching

They are often used in graph processing schemes on FPGAs; we apply them as well.

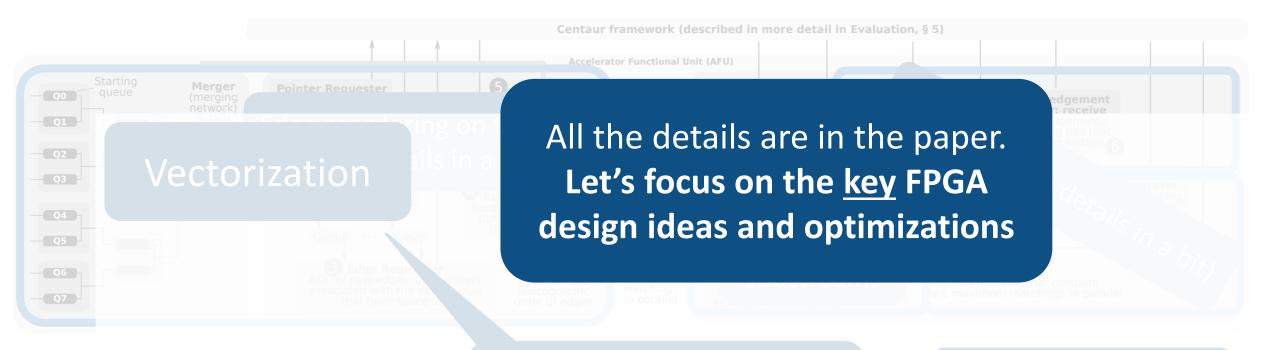
Pipelining







Blocking



Prefetching

They are often used in graph processing schemes on FPGAs; we apply them as well.

Pipelining



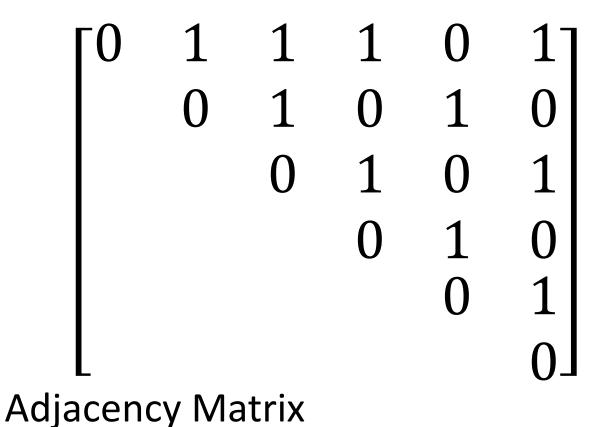








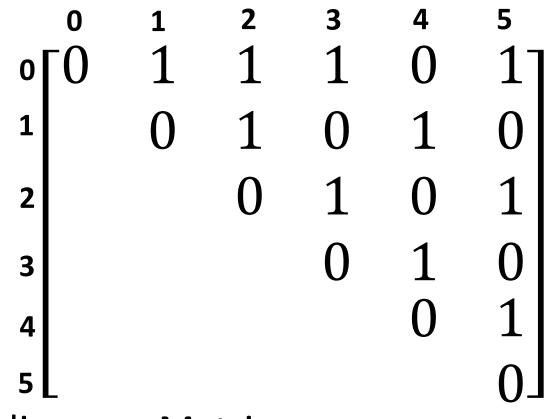








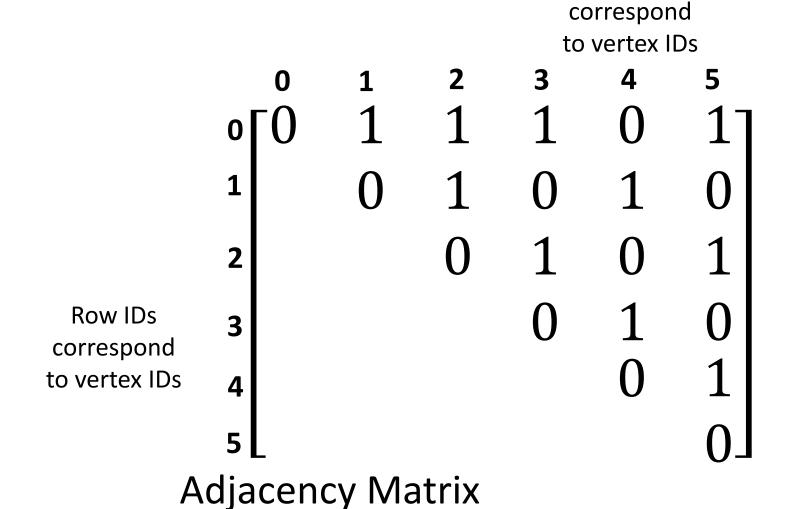








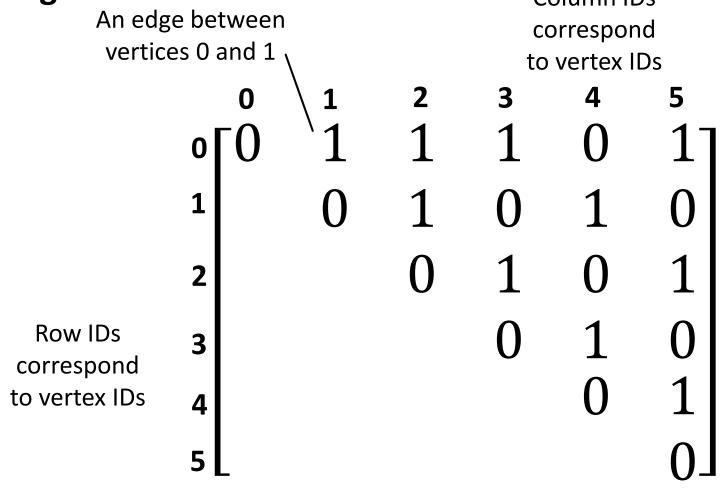








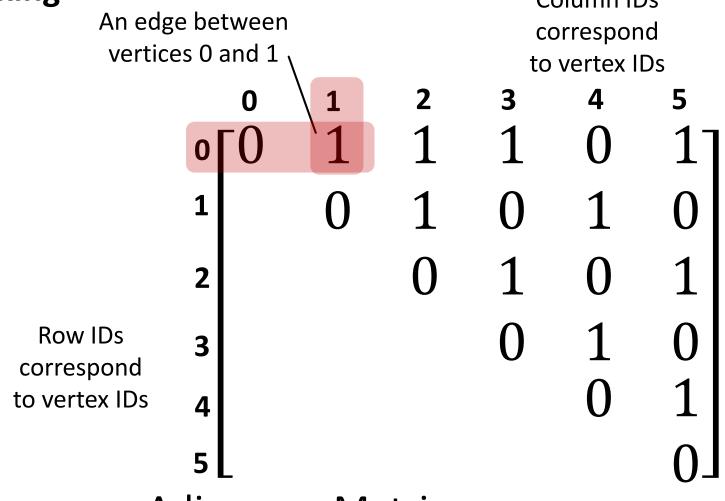








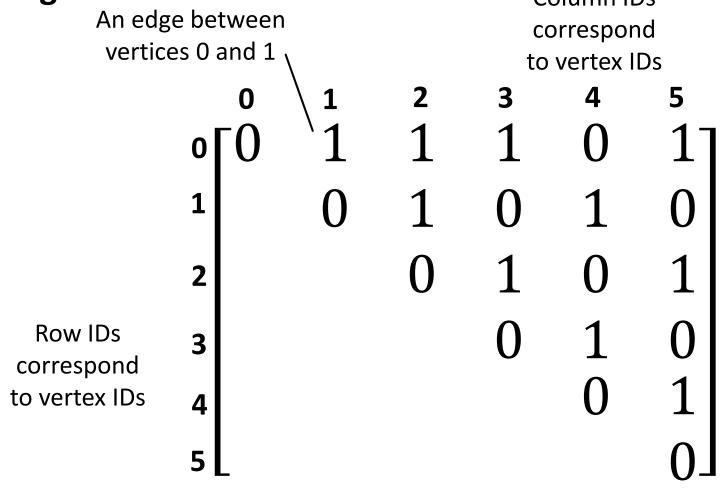








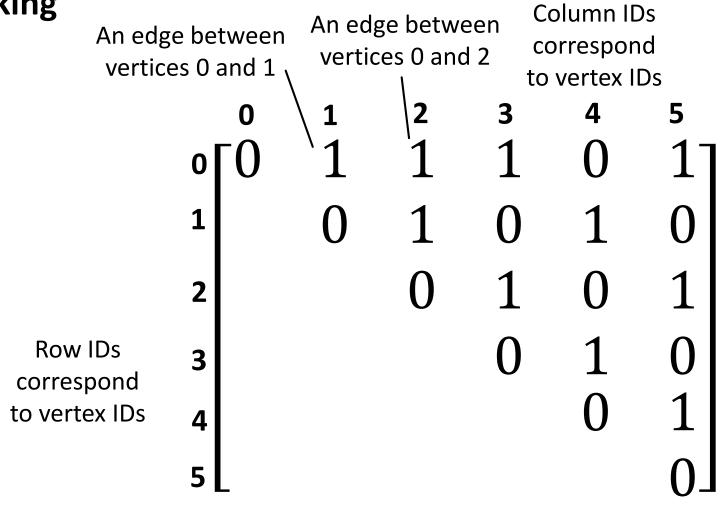








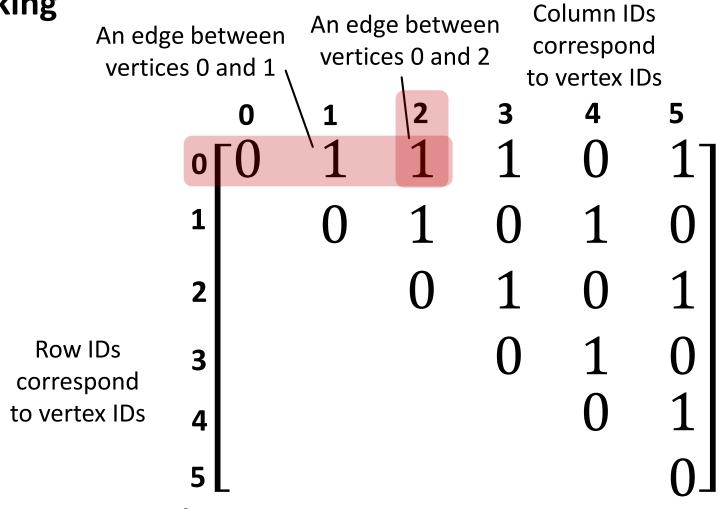








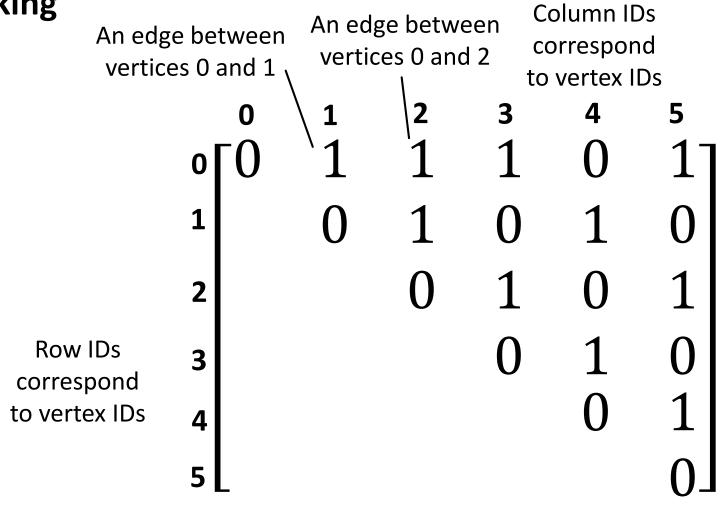








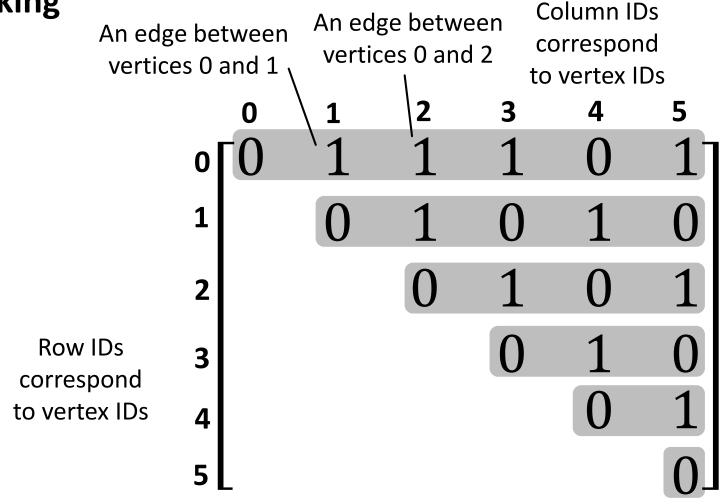








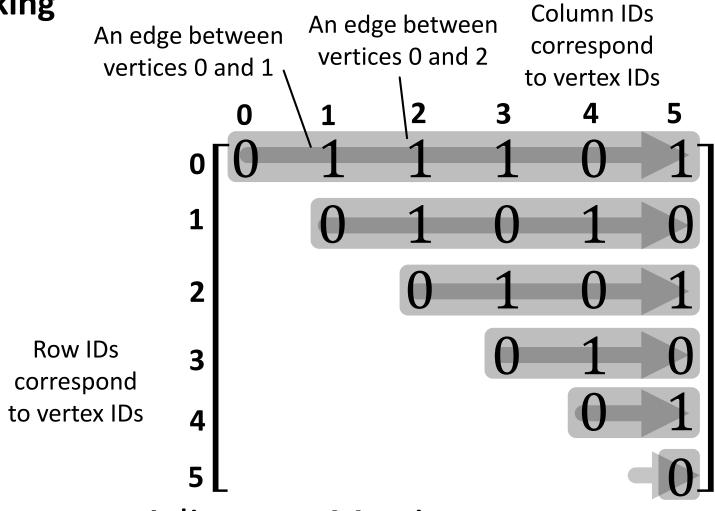








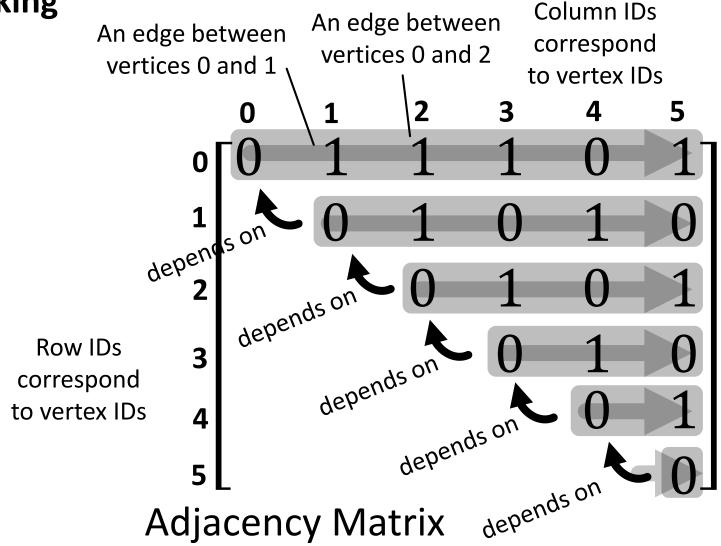








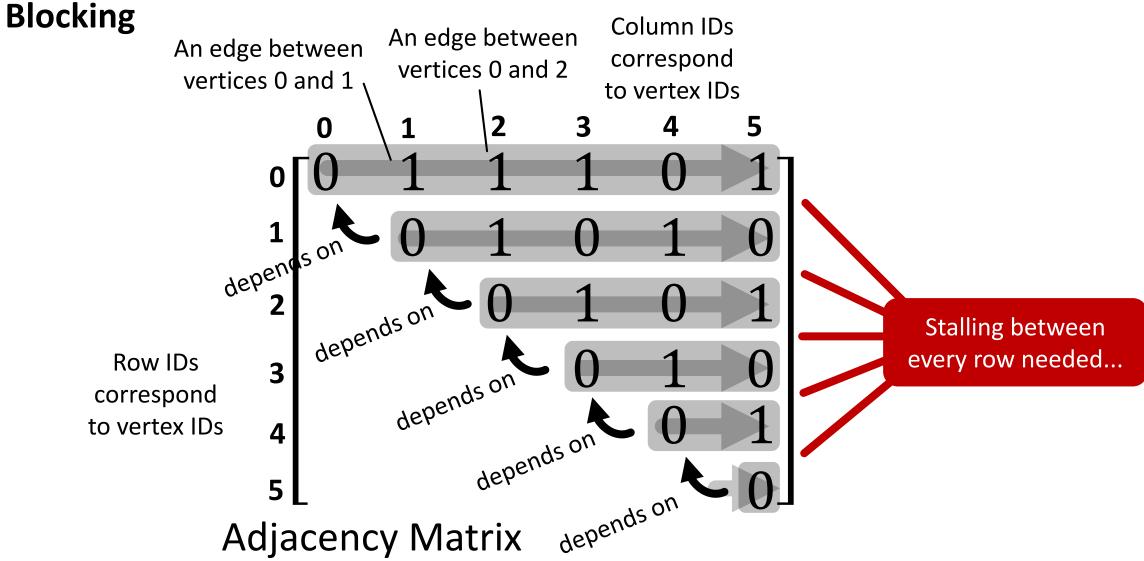








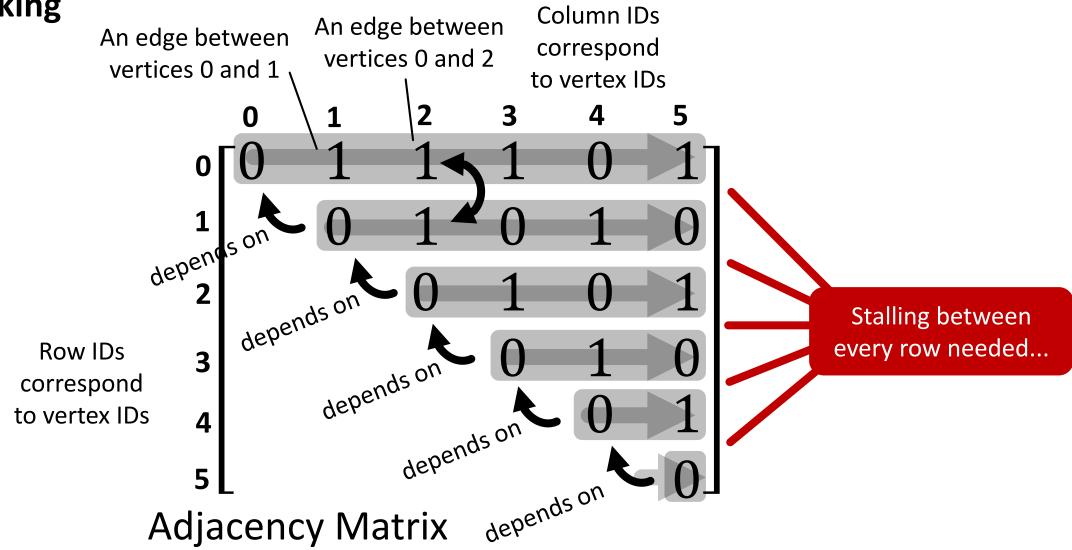








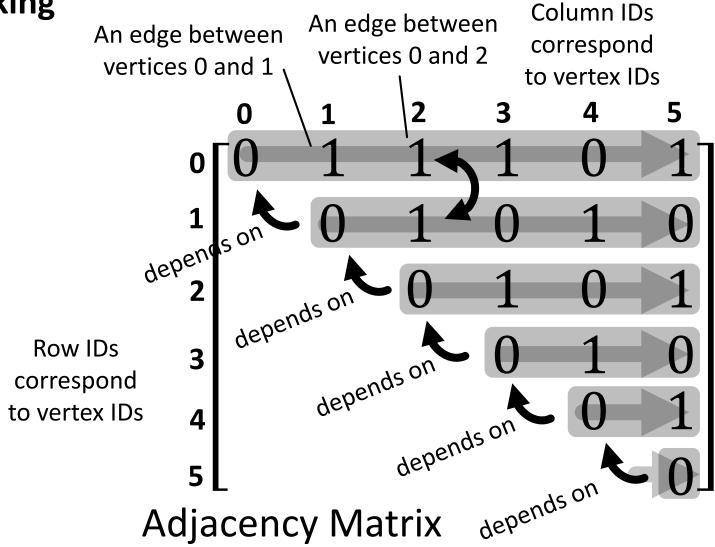








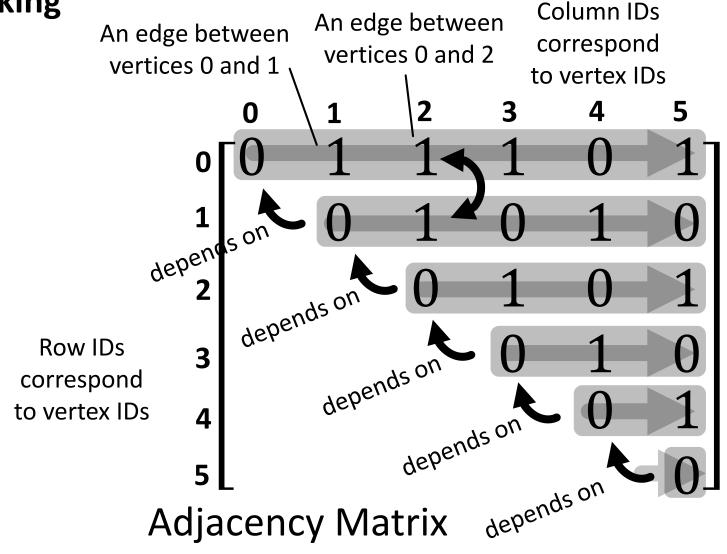










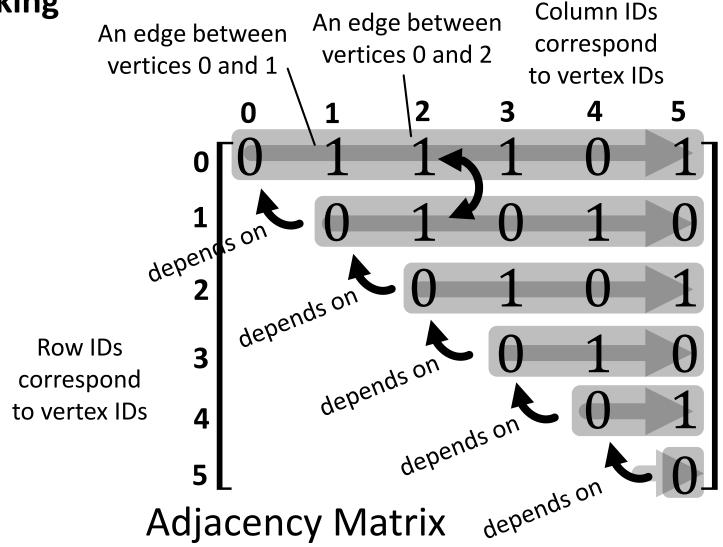


Introduce a (tunable) "blocking parameter" K









Introduce a (tunable) "blocking parameter" K

K determines how many stalls are allowed







Substream-Centric MWM: FPGA optimizations Blocking Column ID

Column IDs An edge between An edge between correspond vertices 0 and 2 vertices 0 and 1 K = 3to vertex IDs 2 0 depends on depends on **Row IDs** 3 depends on correspond to vertex IDs depends on depends on **Adjacency Matrix**

Introduce a (tunable) "blocking parameter" K

K determines how many stalls are allowed

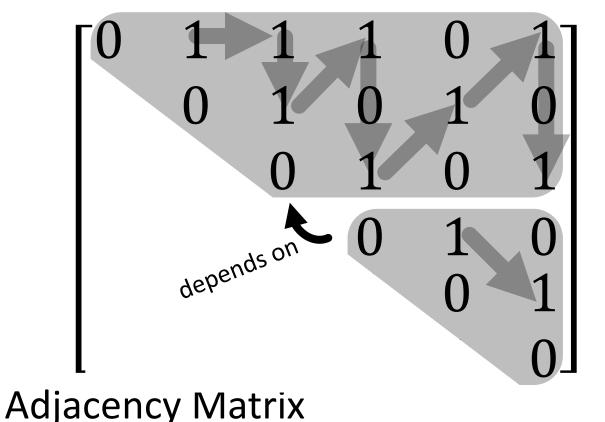






Substream-Centric MWM: FPGA optimizations Blocking

K = 3



Introduce a (tunable) "blocking parameter" K

K determines how many stalls are allowed

Portions of rows are ordered "lexicographically" (i.e., no strict ordering that enforces a stall is required)

Algorithm still (provably) correct







Substream-Centric MWM: FPGA optimizations Blocking

K = 3

K is **tunable**: it controls the tradeoff between the amount of the used FPGA depends on resources and the performance **Adjacency Matrix**

Introduce a (tunable) "blocking parameter" K

K determines how many stalls are allowed

Portions of rows are ordered "lexicographically" (i.e., no strict ordering that enforces a stall is required)

Algorithm still (provably) correct







Research Questions

How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Use <u>substream-centric</u>

<u>processing</u> (exposes parallelism,
enables easy pipelining,
supports approximation)

What is the HW FPGA design that ensures high performance?

?

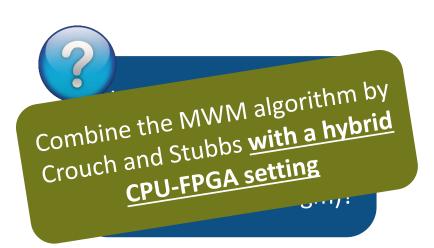
What is the ultimate performance, power consumption, and the related tradeoffs?







Research Questions



Use <u>substream-centric</u>

<u>processing</u> (exposes parallelism,
enables easy pipelining,
supports approximation)

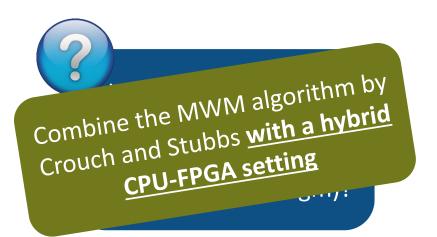
The proper use of blocking, vectorization, pipelining, prefetching

What is the ultimate performance, power consumption, and the related tradeoffs?





Research Questions



Use <u>substream-centric</u>

processing (exposes parallelism, enables easy pipelining, supports approximation)

The proper use of blocking, vectorization, pipelining, prefetching

What is the ultimate performance, power consumption, and the related tradeoffs?



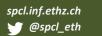




TYPES OF MACHINES

Part 5: Evaluation







TYPES OF MACHINES







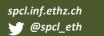
TYPES OF MACHINES

CPU: Intel Broadwell Xeon E5-2680 v4 @3.3 GHz 14 Cores (28 Threads)











TYPES OF GRAPHS







TYPES OF GRAPHS

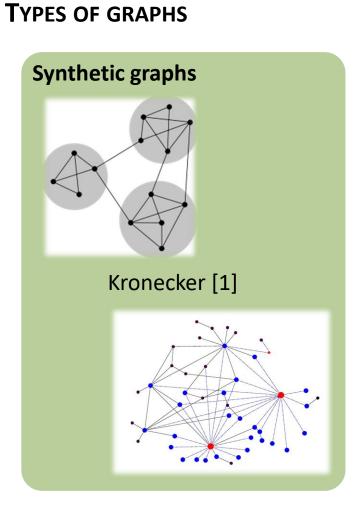
Synthetic graphs







Performance Analysis

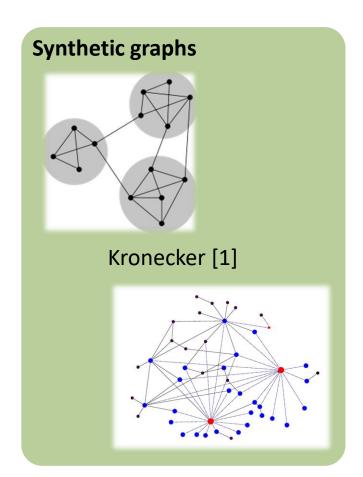








PERFORMANCE ANALYSIS Types of graphs



Real-world graphs (SNAP [2], KONECT [3], DIMACS [4])

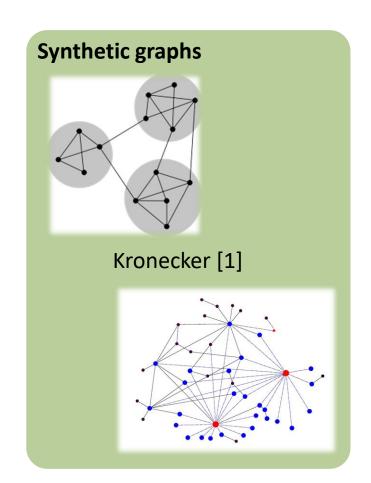
[2] SNAP. https://snap.stanford.edu

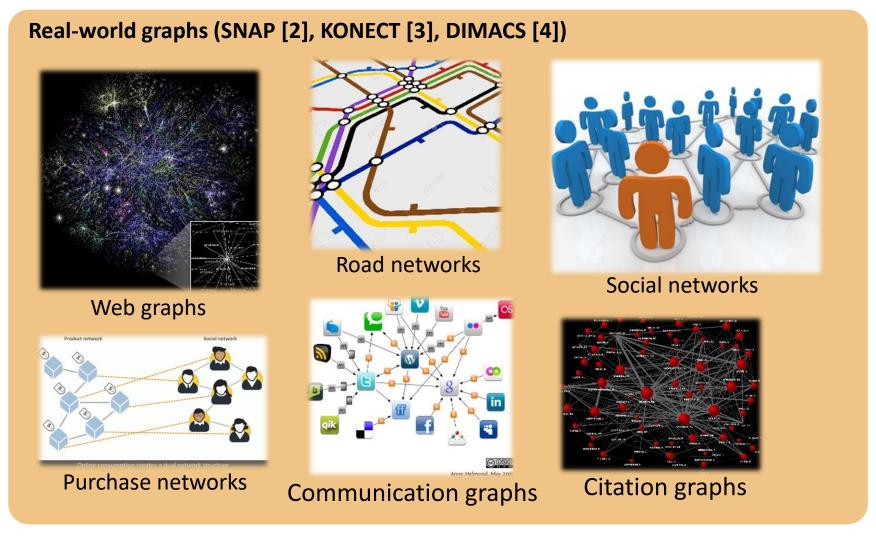






Performance Analysis **TYPES OF GRAPHS**





- [2] SNAP. https://snap.stanford.edu[3] KONECT. https://konect.cc
- [4] DIMACS Challenge







Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid







Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Our FPGA design, (4+ε)-approximation

- [1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.
- [2] M. Ghaffari. Space-optimal semi-streaming for(2+ε)-approximatematching. arXiv:1701.03730, 2017.







CPU implementations of the original Crouch scheme, (4+ε)-approximation

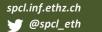
Algorithm Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Chaffari [2] Sequential (G-SEQ) CPU Substream-Centric (SC-OPT) Hybrid

Our FPGA design, (4+ε)-approximation

[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

[2] M. Ghaffari. Space-optimal semi-streaming for(2+ε)-approximatematching. arXiv:1701.03730, 2017.







CPU implementations of the original Crouch scheme, (4+ε)-approximation

State-of-the-art MWM algorithm, space-optimal, time-optimal (O(m)), (2+\varepsilon)-approximation

Algorithm

Crouch et al. [1] Sequential (CS-SEQ)

Crouch et al. [1] Parallel (CS-PAR)

Ghaffari [2] Sequential (G-SEQ)

Substream-Centric (SC-OPT)

P' orm

CPU CPU CPU

Hybrid

Our FPGA design, (4+ε)-approximation

[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

[2] M. Ghaffari. Space-optimal semi-streaming for (2+ε)-approximatematching. arXiv:1701.03730, 2017.







CPU implementations of the original Crouch scheme, (4+ε)-approximation

State-of-the-art MWM algorithm, space-optimal, time-optimal (O(m)), (2+ε)-approximation

Algorithm

Crouch et al. [1] Sequential (CS-SEQ)

Crouch et al. [1] Parallel (CS-PAR)

Ghaffari [2] Sequential (G-SEQ)

Substream-Centric (SC-OPT)

P' orm

CPU CPU CPU Hybrid

Our FPGA design, (4+ε)-approximation

We test both CPU and hybrid (FPGA+CPU) platforms

[1] M. Crouch and D. M. Stubbs. Improved streaming Algorithms for weighted Matching, via unweighted Matching. LIPIcs-Leibniz Informatics. 2014.

[2] M. Ghaffari. Space-optimal semi-streaming for (2+ε)-approximatematching. arXiv:1701.03730, 2017.



PERFORMANCE ANALYSIS VARIOUS GRAPHS

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

Blocking size (K) = 32, #Substreams (L) = 64 #Threads = 4, ε = 0.1





Graph	Type	m	п
Orkut Stanford Berkeley	Synthetic power-law Social network Social network Social network Social network Hyperlink graph Hyperlink graph Citation graph	\approx 48 <i>n</i> 950,327 33,140,017 68,993,773 117,184,899 2,312,497 7,600,595 352,807	2^k ; $k = 16,, 21$ 196,591 2,302,925 4,847,571 3,072,441 281,903 685,230 27,770



Parameters:

Blocking size (K) = 32, #Substreams (L) = 64 #Threads = 4, ε = 0.1

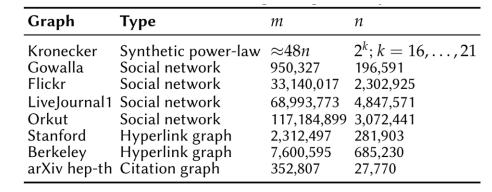
•

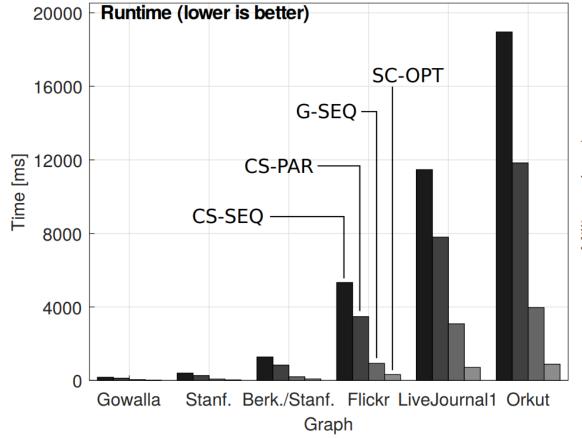


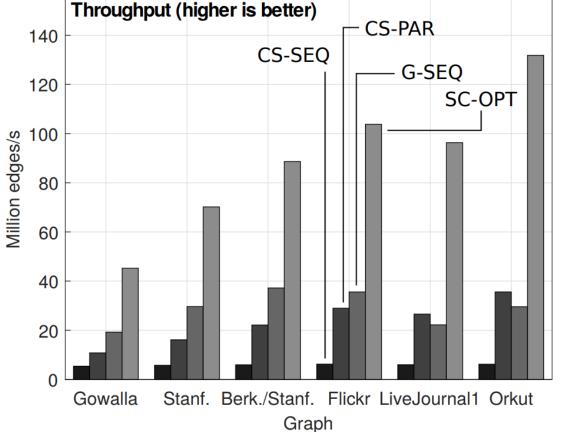


PERFORMANCE ANALYSIS VARIOUS GRAPHS

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid









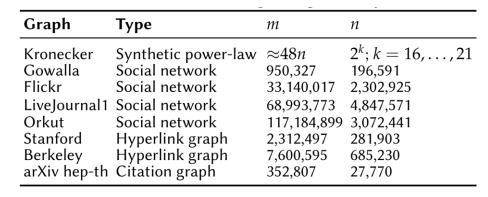
Parameters:

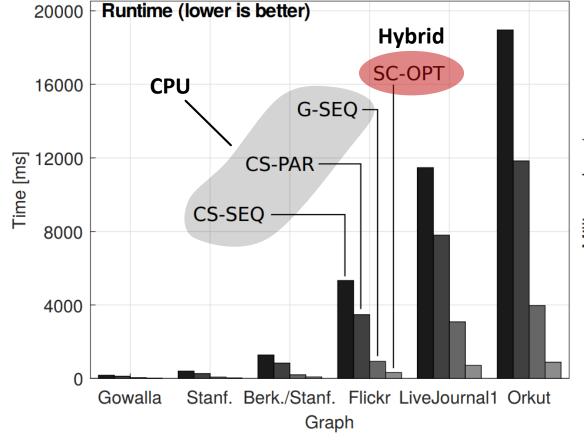
Blocking size (K) = 32, #Substreams (L) = 64 #Threads = 4, ε = 0.1

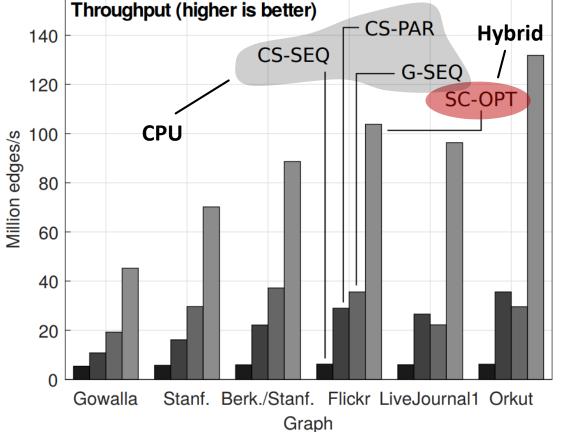
pcl.inf.ethz.ch	- ::	niah
<pre> @spcl_eth </pre>	ıZU.	rich

Performance A	ANALYSIS
Various Graphs	

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid









spcl.inf.ethz.ch @spcl_eth

ETH zürich

PERFORMANCE ANALYSIS VARIOUS GRAPHS

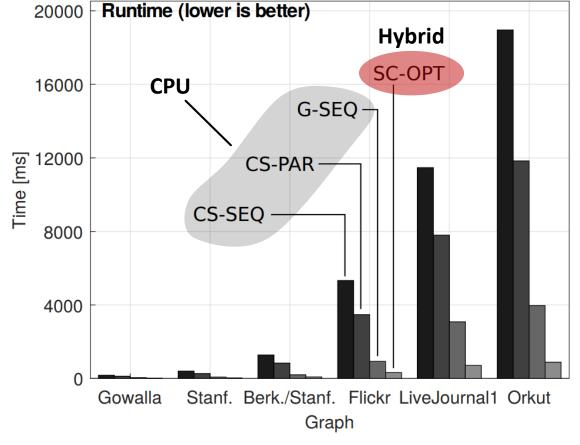
Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

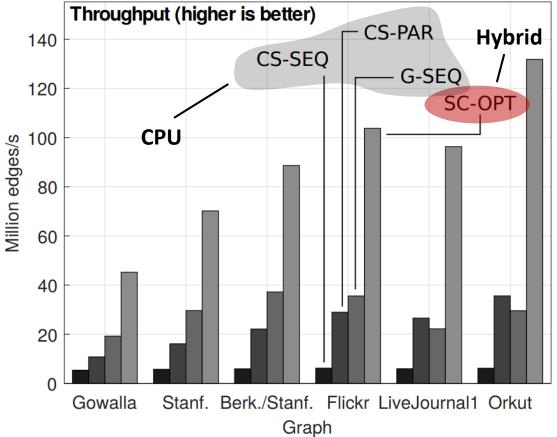
Parameters:

Blocking size (K) = 32, #Substreams (L) = 64 #Threads = 4, ε = 0.1

SC-OPT secures highest performance

Graph	Туре	m	n
Orkut Stanford Berkeley	Synthetic power-law Social network Social network Social network Social network Hyperlink graph Hyperlink graph Citation graph	\approx 48 <i>n</i> 950,327 33,140,017 68,993,773 117,184,899 2,312,497 7,600,595 352,807	2^k ; $k = 16,, 21$ 196,591 2,302,925 4,847,571 3,072,441 281,903 685,230 27,770







spcl.inf.ethz.ch @spcl_eth



PERFORMANCE ANALYSIS VARIOUS GRAPHS

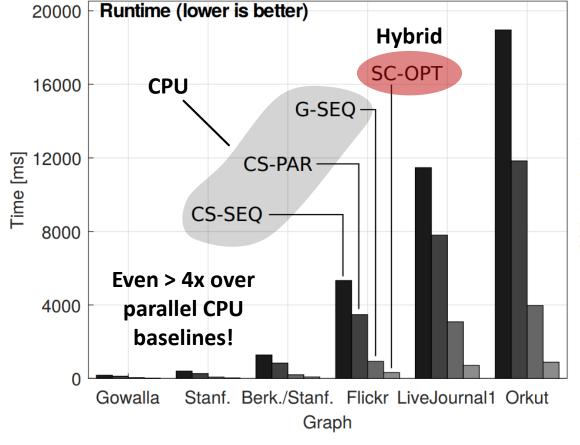
Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

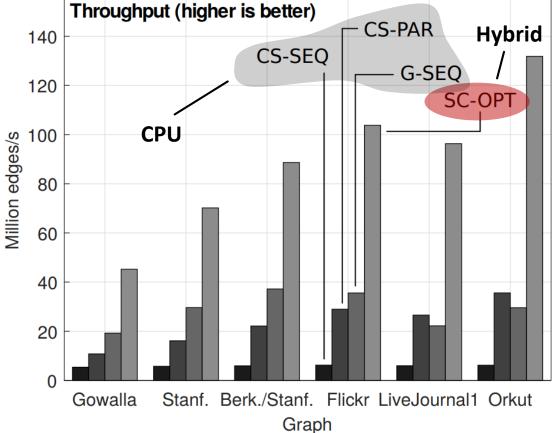
Parameters:

Blocking size (K) = 32, #Substreams (L) = 64 #Threads = 4, ε = 0.1

SC-OPT secures highest performance

Graph	Type	m	п
Orkut Stanford Berkeley	Synthetic power-law Social network Social network Social network Social network Hyperlink graph Hyperlink graph Citation graph	\approx 48 <i>n</i> 950,327 33,140,017 68,993,773 117,184,899 2,312,497 7,600,595 352,807	2^k ; $k = 16,, 21$ 196,591 2,302,925 4,847,571 3,072,441 281,903 685,230 27,770









Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric (SC-OPT)	CPU CPU CPU Hybrid

Parameters:

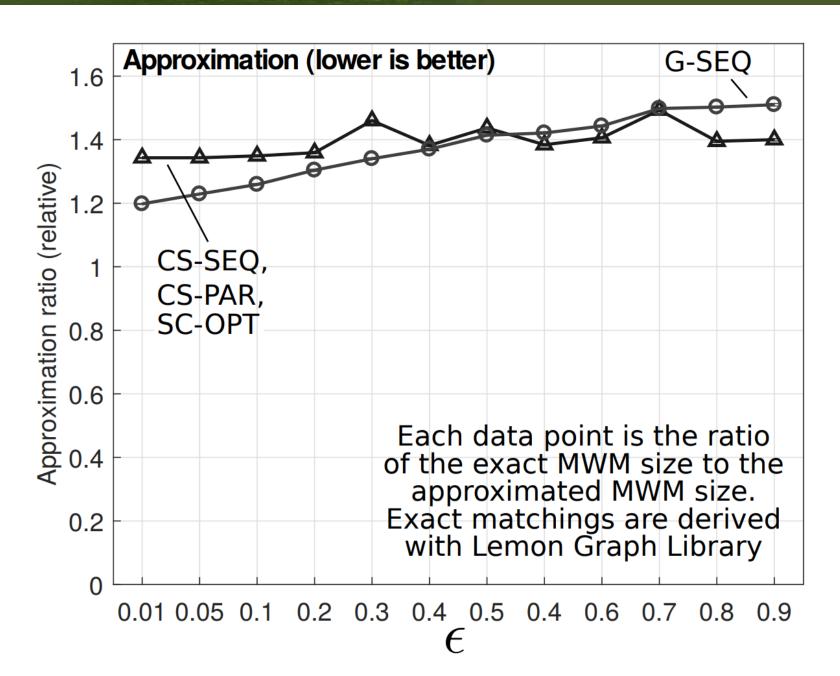
#Substreams (L) = 128, Blocking size (K) = 32, #threads = 4, #edges = 8M (Kronecker)



Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

#Substreams (L) = 128, Blocking size (K) = 32, #threads = 4, #edges = 8M (Kronecker)

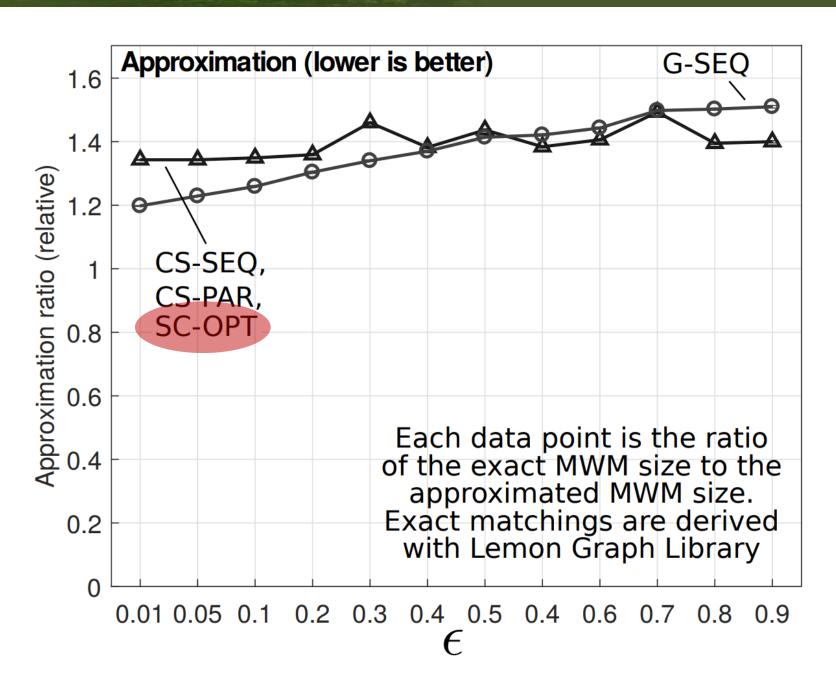




Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

#Substreams (L) = 128, Blocking size (K) = 32, #threads = 4, #edges = 8M (Kronecker)





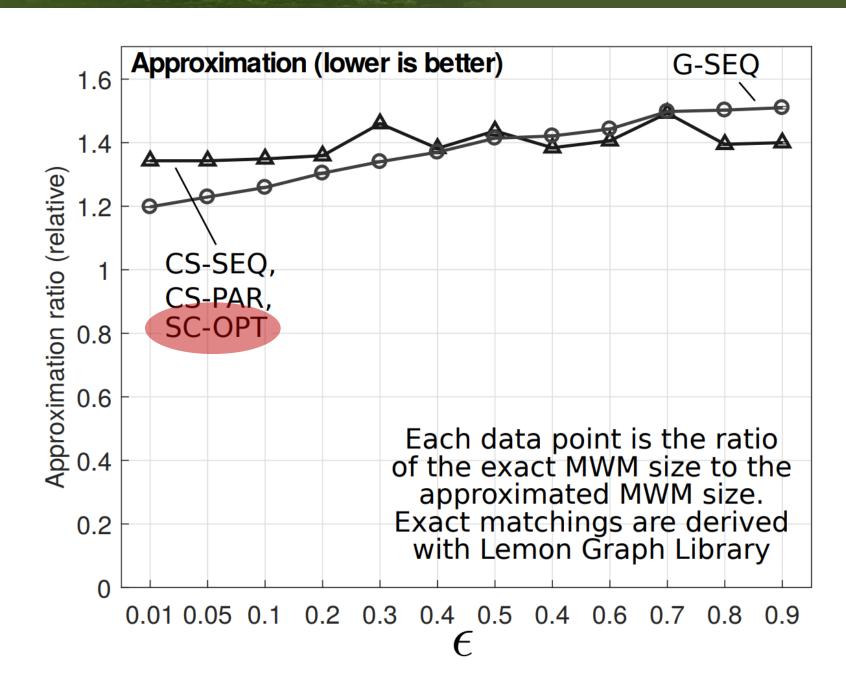


Algorithm		Platform
Crouch et al. [1] Sequer Crouch et al. [1] Paralle Ghaffari [2] Sequential Substream-Centric (SC-C	l (CS-PAR) (G-SEQ)	CPU CPU CPU Hybrid

Parameters:

#Substreams (L) = 128, Blocking size (K) = 32, #threads = 4, #edges = 8M (Kronecker)

SC-OPT is comparable to the $(2+\epsilon)$ -approximation by Ghaffari et al.







L: #Substreams (pipelines),

K: Blocking size,

T: #CPU threads





Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid





L: #Substreams (pipelines),

K: Blocking size,

T: #CPU threads





Platform
CPU
CPU
CPU
Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	K = 32, L = 512	14.789
SC-OPT	K = 256, L = 128	14.789
SC-OPT	K = 32, L = 64	14.657
CS-PAR	T = 64	120



Parameters:

L: #Substreams (pipelines),

K: Blocking size,

T: #CPU threads





Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	K = 32, L = 512	14.789
SC-OPT	K = 256, L = 128	14.789
SC-OPT	K = 32, L = 64	14.657
CS-PAR	T = 64	120







Parameters:

L: #Substreams (pipelines),

K: Blocking size,

T: #CPU threads

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	K = 32, L = 512	14.789
SC-OPT	K = 256, L = 128	14.789
SC-OPT	K = 32, L = 64	14.657
CS-PAR	T=64	120

SC-OPT (Hybrid) is ~8x more power-efficient than the CPU implementation







ENERGY CONSUMPTION, RESOURCE UTILIZATION

Parameters:

L: #Substreams (pipelines),

K: Blocking size,

T: #CPU threads

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	K = 32, L = 512	14.789
SC-OPT	K = 256, L = 128	14.789
SC-OPT	K = 32, L = 64	14.657
CS-PAR	T=64	120

SC-OPT (Hybrid) is ~8x more power-efficient than the CPU implementation

FPGA Algorithm	Parameters	Used BRAM	Used ALMs
SC-OPT	K = 32, L = 512	11.5 MBit (21%)	151,998 (32%)
SC-OPT	K = 256, L = 128	24.8 MBit (45%)	350,556 (82%)







Performance Analysis

ENERGY CONSUMPTION, RESOURCE UTILIZATION

Parameters:

L: #Substreams (pipelines),

K: Blocking size,

T: #CPU threads

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Algorithm	Parameters	Energy Consumption [W]
SC-OPT	K = 32, L = 512	14.789
SC-OPT	K = 256, L = 128	14.789
SC-OPT	K = 32, L = 64	14.657
CS-PAR	T=64	120

SC-OPT (Hybrid) is ~8x more power-efficient than the CPU implementation

FPGA Algorithm	Parameters	Used BRAM	Used ALMs
SC-OPT	K = 32, L = 512	11.5 MBit (21%)	151,998 (32%)
SC-OPT	K = 256, L = 128	24.8 MBit (45%)	350,556 (82%)

Blocking needs more resources (but is **tunable**!)







PERFORMANCE ANALYSIS DESIGN SPACE EXPLORATION

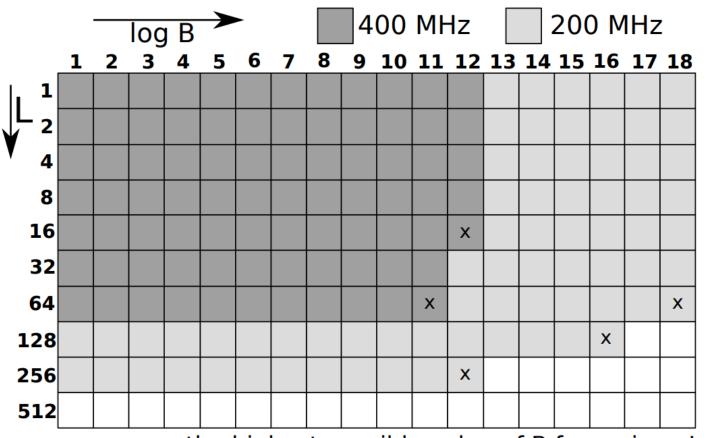




PERFORMANCE ANALYSIS DESIGN SPACE EXPLORATION

B – BRAM size allocated for matching data structures,

L – number of substreams (pipelines)



x - the highest possible value of B for a given L



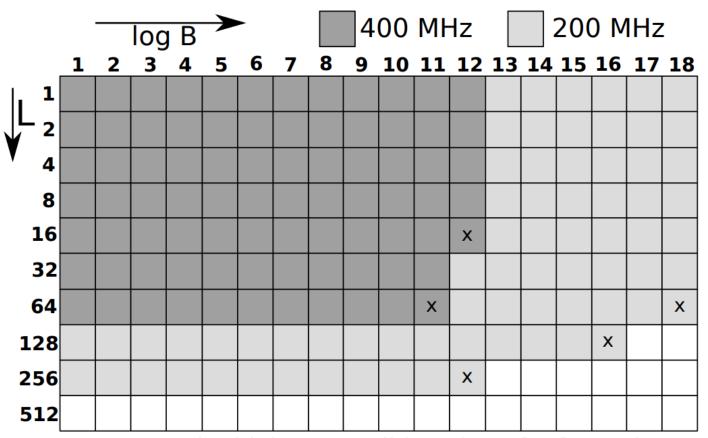


PERFORMANCE ANALYSIS DESIGN SPACE EXPLORATION

B – BRAM size allocated for matching data structures,

L – number of substreams (pipelines)

Addition complexity grows linearly with *L*



x - the highest possible value of B for a given L





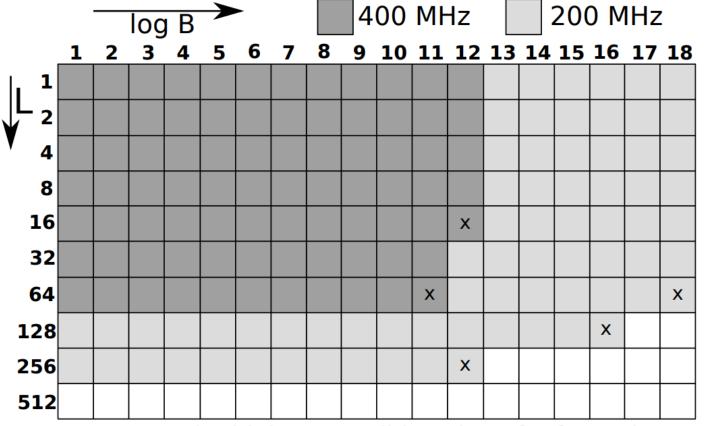
PERFORMANCE ANALYSIS DESIGN SPACE EXPLORATION

Addition complexity grows linearly with *L*

BRAM signal propagation limits the frequency

B – BRAM size allocated for matching data structures,

L – number of substreams (pipelines)



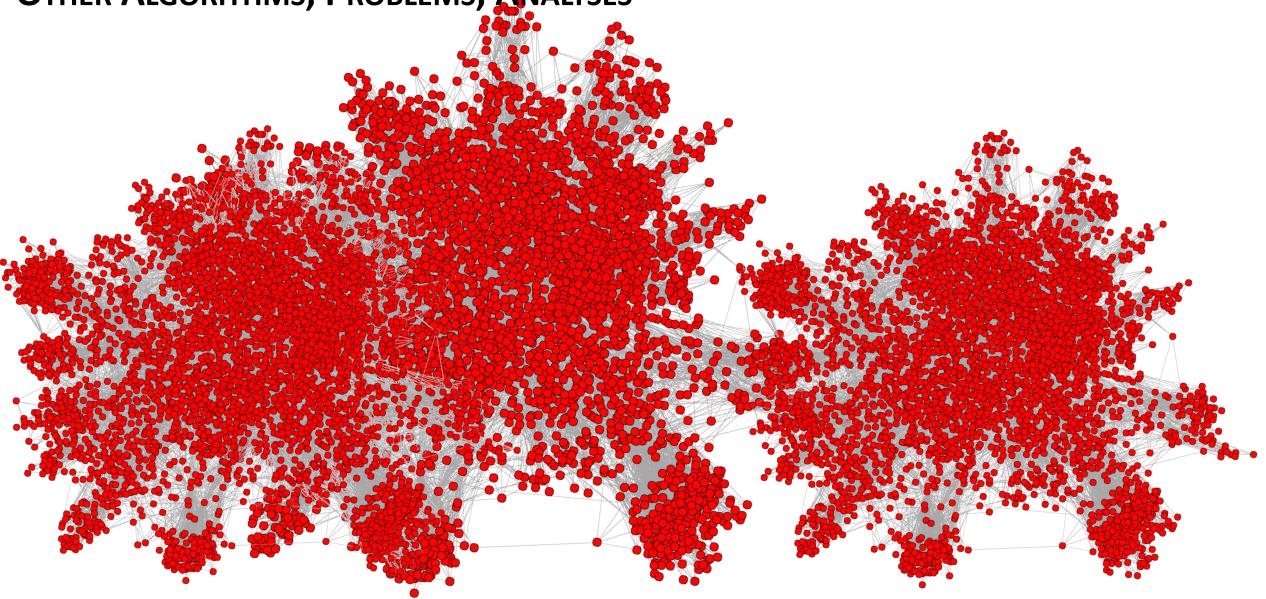
x - the highest possible value of B for a given L







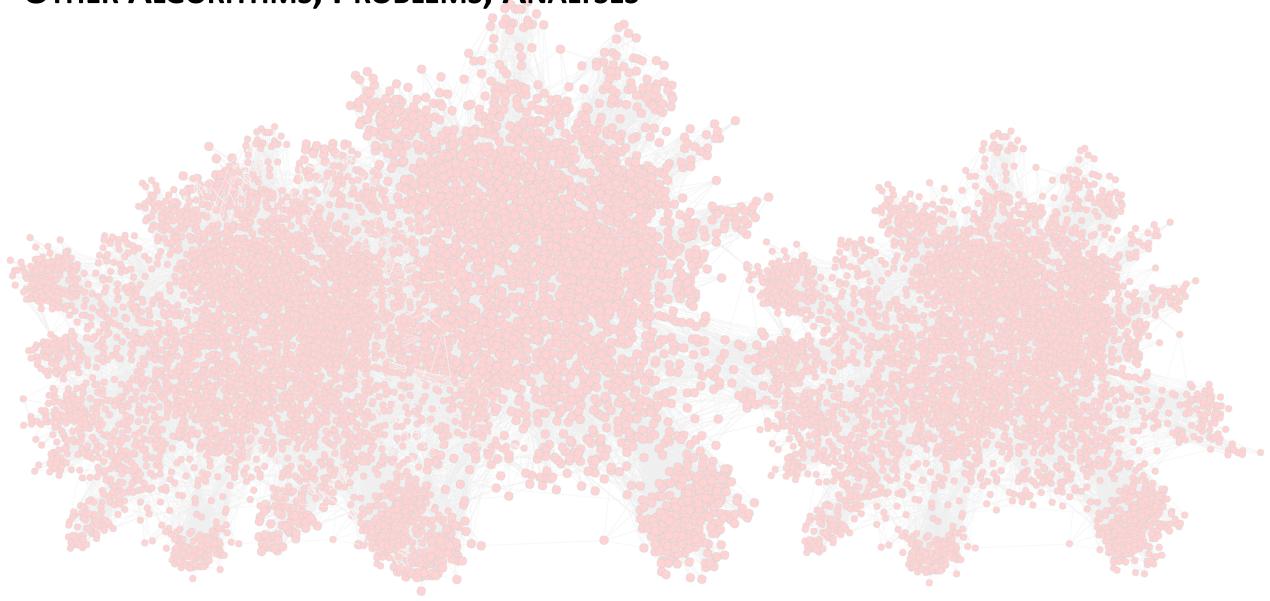








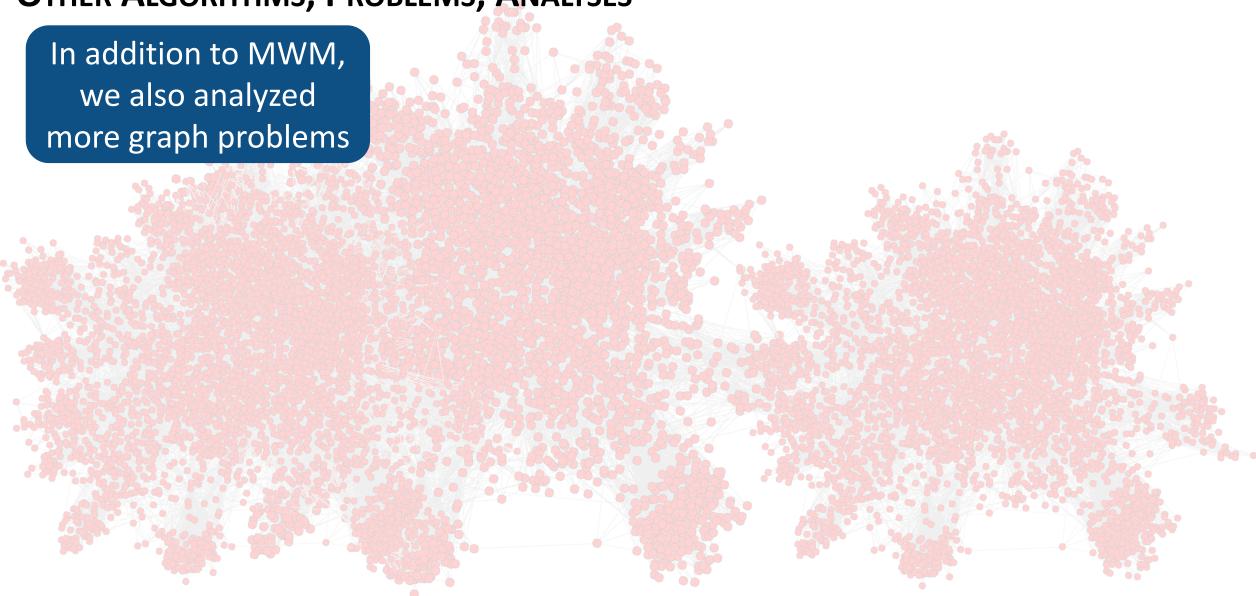






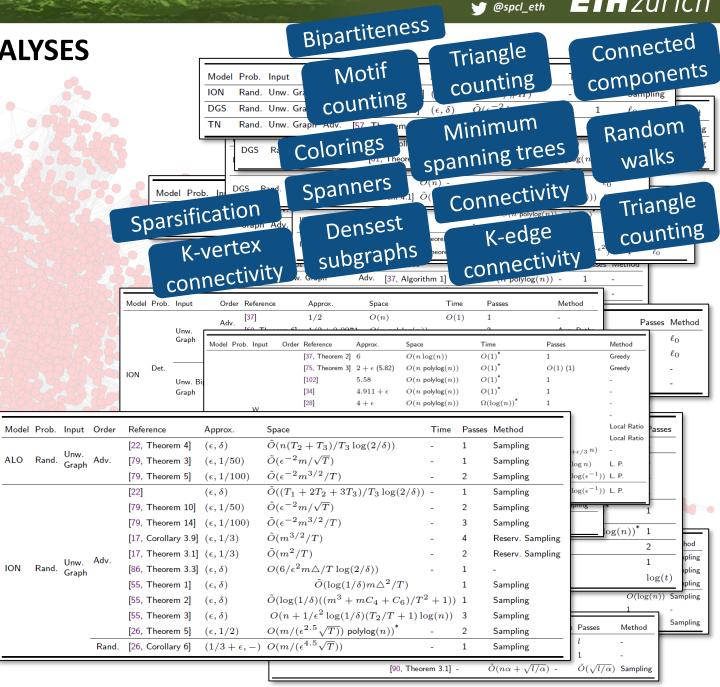






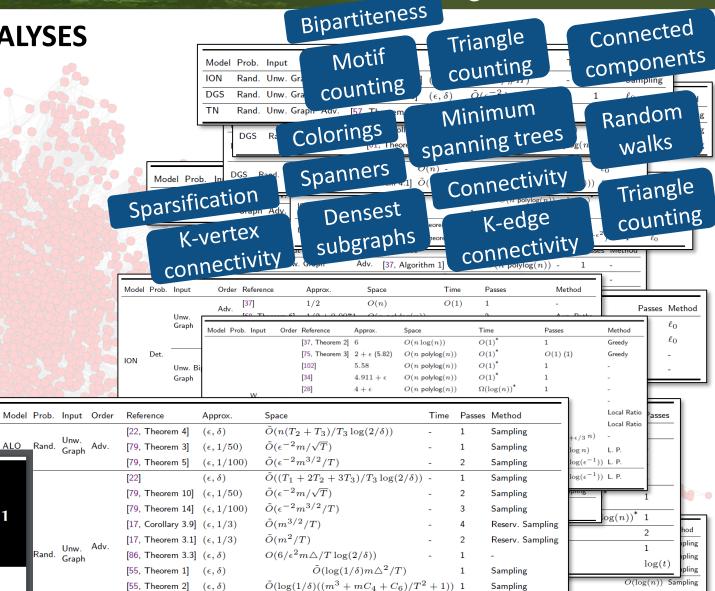


In addition to MWM, we also analyzed more graph problems





In addition to MWM, we also analyzed more graph problems



 $O(n + 1/\epsilon^2 \log(1/\delta)(T_2/T + 1)\log(n))$

 $O(m/(\epsilon^{2.5}\sqrt{T})) \operatorname{polylog}(n))^*$

Sampling

Sampling

Sampling

 $O(n\alpha + \sqrt{l/\alpha})$

Method

 $O(\sqrt{l/\alpha})$ Sampling

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

ALO_Rand.

[55, Theorem 3]

[26, Theorem 5]

 $(\epsilon, 1/2)$

Rand. [26, Corollary 6] $(1/3 + \epsilon, -)$ $O(m/(\epsilon^{4.5}\sqrt{T}))$

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich



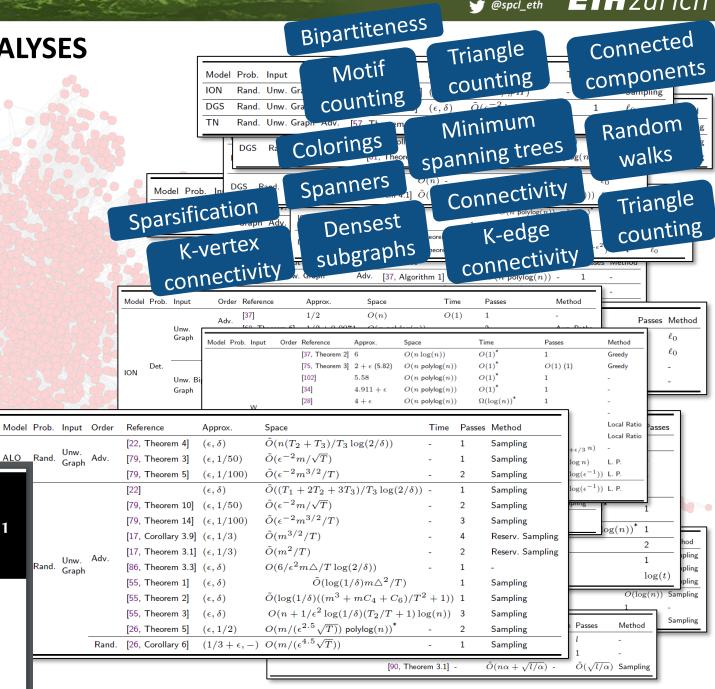
In addition to MWM, we also analyzed more graph problems

http://spcl.inf.ethz.ch/Publications/.pdf/ graphs-fpgas-survey.pdf (submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich





In addition to MWM, we also analyzed more graph problems

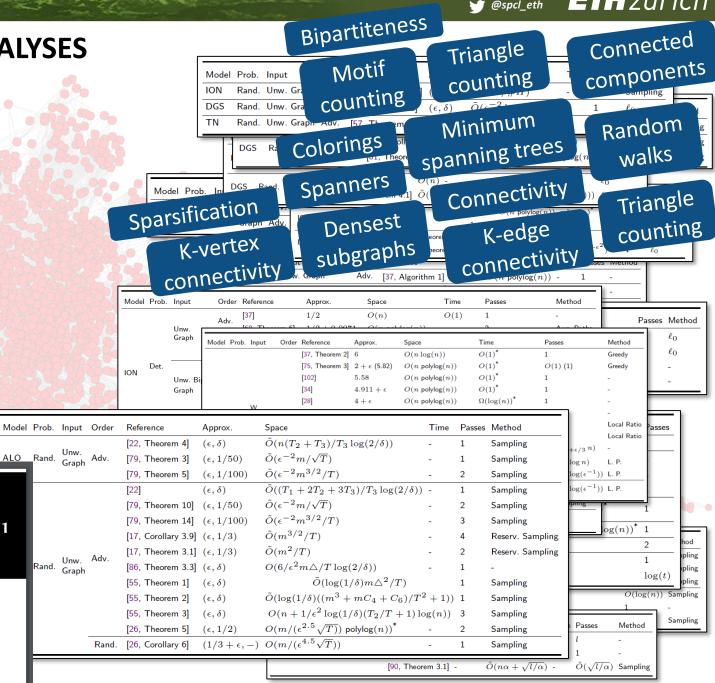
To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

http://spcl.inf.ethz.ch/Publications/.pdf/ graphs-fpgas-survey.pdf (submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich





spcl.inf.ethz.ch



OTHER ALGORITHMS, PROBLEMS, ANALYSES

In addition to MWM, we also analyzed more graph problems

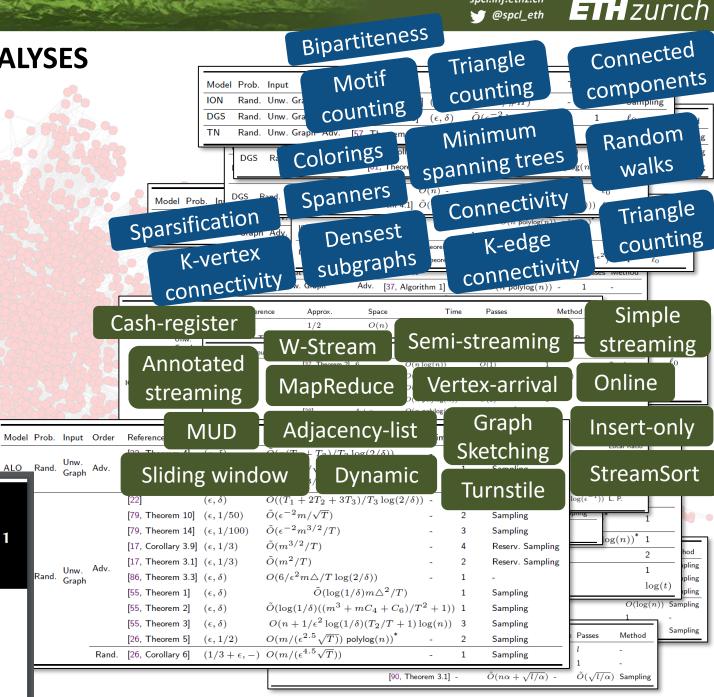
> To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

http://spcl.inf.ethz.ch/Publications/.pdf/ graphs-fpgas-survey.pdf (submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich





In addition to MWM, we also analyzed more graph problems

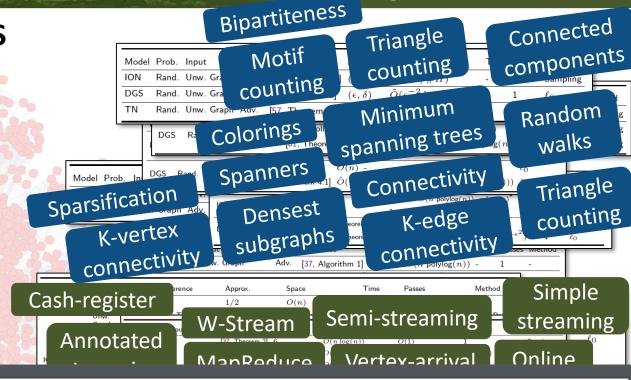
To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

http://spcl.inf.ethz.ch/Publications/.pdf/ graphs-fpgas-survey.pdf (submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich



spcl.inf.ethz.ch

@spcl_eth

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

1

ETH zürich



In addition to MWM, we also analyzed more graph problems

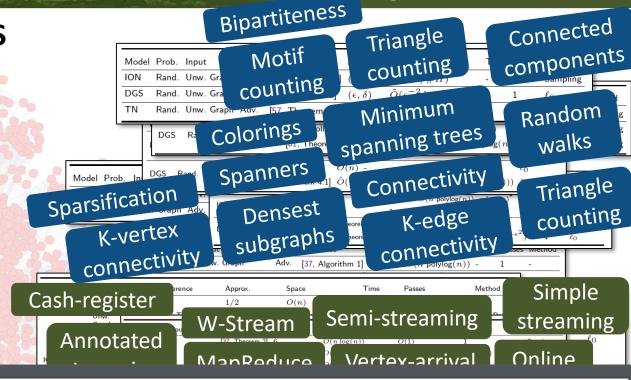
To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

http://spcl.inf.ethz.ch/Publications/.pdf/ graphs-fpgas-survey.pdf (submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich



spcl.inf.ethz.ch

@spcl_eth

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

1

ETH zürich



OTHER ALGORITHMS, PROBLEM

In addition to MWM, we also analyzed more graph problems Work in progress on the distributed setting ©

Bipartiteness Motif counting

Triangle counting

spcl.inf.ethz.ch

@spcl_eth

Connected components

ETH zürich

Minimum Random spanning trees walks

Spanners Connectivity

Densest subgraphs

K-edge connectivity

Cash-register

Annotated

ification

K-vertex

connectivity

W-Stream

Colorings

Semi-streaming

Simple streaming

Triangle

counting

Vertey-arriva

To enable rigorous reasoning, we analyzed ~15 models for streaming graph processing (and selected the best for FPGAs)

http://spcl.inf.ethz.ch/Publications/.pdf/ graphs-fpgas-survey.pdf (submitted to arXiv, will appear tonight)

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

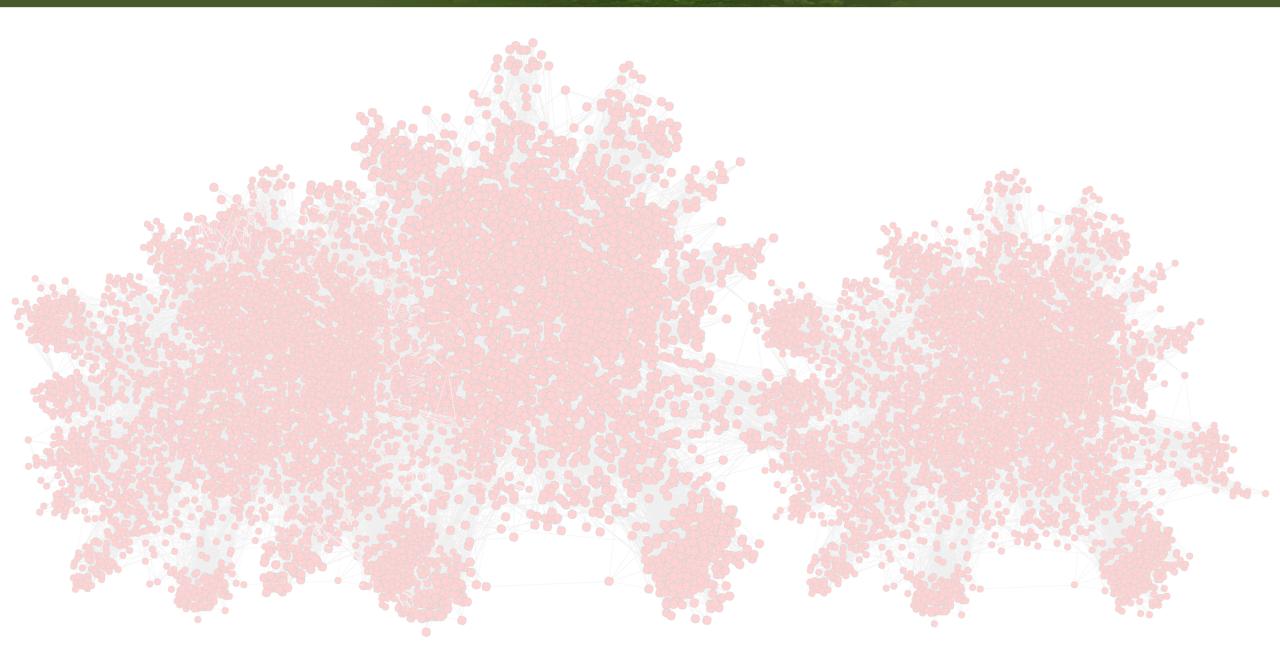
Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

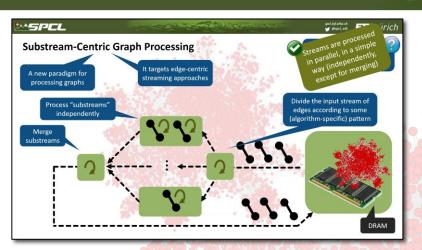
Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich





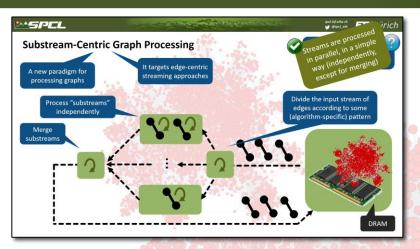






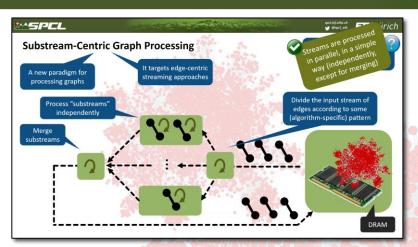






DETAILED DOMAIN ANALYSIS,
IDENTIFICATION OF SEMI-STREAMING
MODEL AS FPGA BEST-FIT, 2 SURVEYS





DETAILED DOMAIN ANALYSIS,
IDENTIFICATION OF SEMI-STREAMING
MODEL AS FPGA BEST-FIT, 2 SURVEYS

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TABEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including n ing, social network analysis, computational sciences, and others. Two key challenges that hinder graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing updates, with millions of edges added or removed per second. Graph streaming algorithms are

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

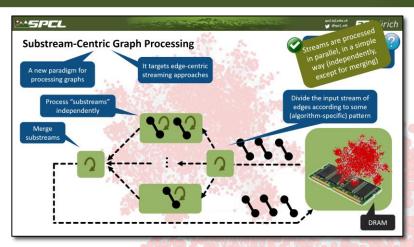
MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field









DETAILED DOMAIN ANALYSIS,
IDENTIFICATION OF SEMI-STREAMING
MODEL AS FPGA BEST-FIT, 2 SURVEYS

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including n ing, social network analysis, computational sciences, and others. Two key challenges that hinder graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing updates, with millions of edges added or removed per second. Graph streaming algorithms are the control of the control

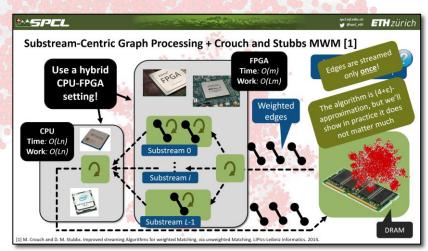
Graph Processing on FPGAs: Taxonomy, Survey, Challenges

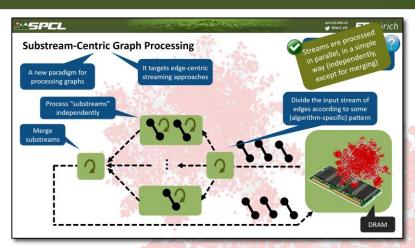
Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field

THEORY-INSPIRED MWM APPROXIMATE ALGORITHM ON A HYBRID CPU-FPGA SETTING





DETAILED DOMAIN ANALYSIS,
IDENTIFICATION OF SEMI-STREAMING
MODEL AS FPGA BEST-FIT, 2 SURVEYS

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including n ing, social network analysis, computational sciences, and others. Two key challenges that hinde graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing undates, with millions of edges added or removed per second. Graph streaming algorithms ar

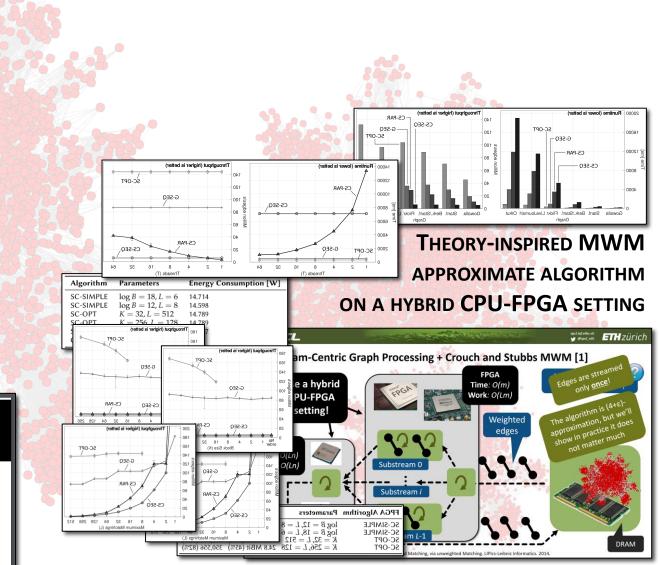
s. The input graph is pessed as a streem of undetes allowing to

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

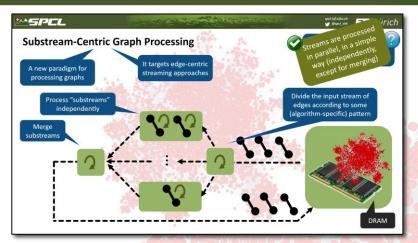
Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER. Department of Computer Science. ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field







DETAILED DOMAIN ANALYSIS,
IDENTIFICATION OF SEMI-STREAMING
MODEL AS FPGA BEST-FIT, 2 SURVEYS

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TALBEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

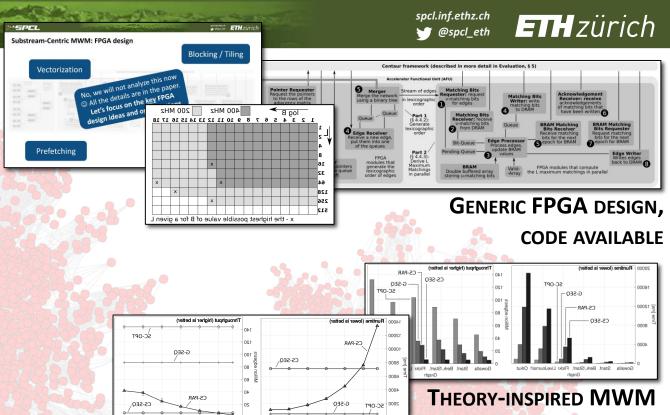
Graph processing has become an important part of various areas of computer science, including n ing, social network analysis, computational sciences, and others. Two key challenges that hinde graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing updates, with millions of edges added or removed per second. Graph streaming algorithms are the data discusses these increases. The insurance in creation is not designed to the control of the con

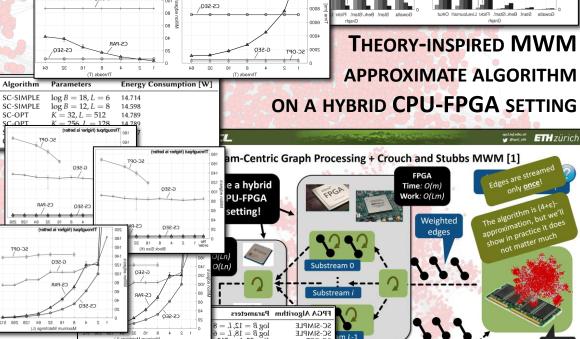
Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

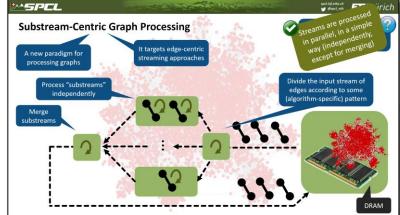
MACIEJ BESTA*, DIMITRI STANOJEVIC*. Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field









DETAILED DOMAIN ANALYSIS, IDENTIFICATION OF SEMI-STREAMING MODEL AS FPGA BEST-FIT, 2 SURVEYS

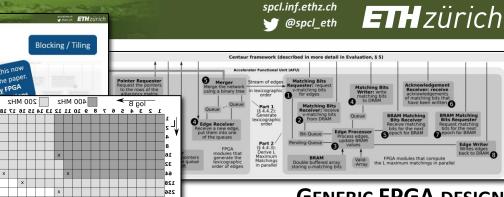
OTHER ALGORITHMS, PROBLEMS, ANALYSES more graph problem

Substream-Centric MWM: FPGA design

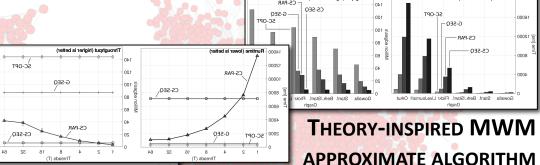
Blocking / Tiling

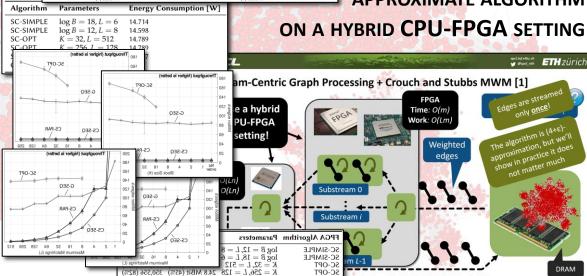
400 MHz

GENERALIZABILITY TO OTHER GRAPH PROBLEMS AND SETTINGS



GENERIC FPGA DESIGN, **CODE AVAILABLE**





Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including ing, social network analysis, computational sciences, and others. Two key challenges that hind graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing updates, with millions of edges added or removed per second. Graph streaming algorithms a s. The input group is passed as a streem of undates allow

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

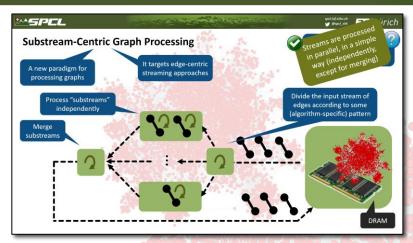
Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field

Website & code: http://spcl.inf.ethz.ch/Research/Parallel_Programming/Substream_Centric





SUBSTREAM-CENTRIC GRAPH PROCESSING PARADIGM, EXPOSES PARALLELISM, ENABLES EASY PIPELINING, SUPPORTS APPROXIMATION

DETAILED DOMAIN ANALYSIS,
IDENTIFICATION OF SEMI-STREAMING
MODEL AS FPGA BEST-FIT, 2 SURVEYS

OTHER ALGORITHMS, PROBLEMS, ANALYSES

Well, not enough time to present ©

In addition to MWM, we also analyzed more graph problems

Ocertification

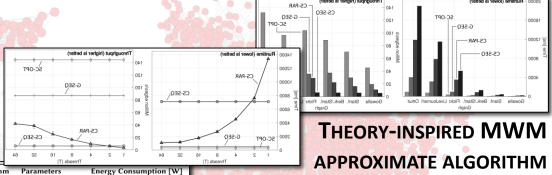
Overetx

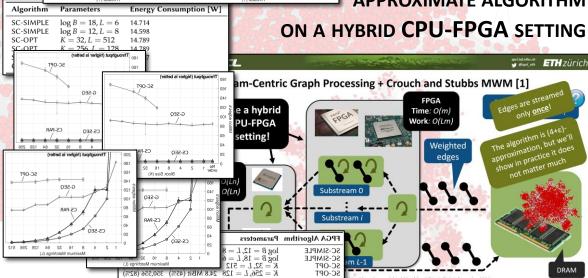
Orner of the control of the c

GENERALIZABILITY TO OTHER
GRAPH PROBLEMS AND SETTINGS

Acknowledgement Requester for the paper.

| Pointer Requester | Po





Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including n ing, social network analysis, computational sciences, and others. Two key challenges that hinde graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing undates. with millions of edges added or removed eer second. Graph streaming algorithms are

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

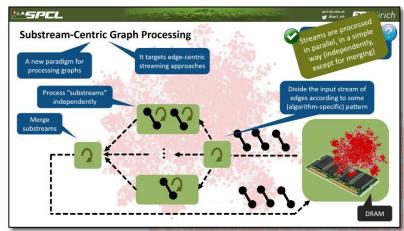
Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing pages unique challenges for the runtime and the consumed nower Field







DETAILED DOMAIN ANALYSIS,
IDENTIFICATION OF SEMI-STREAMING
MODEL AS FPGA BEST-FIT, 2 SURVEYS

THE ALGORITHMS, PROBLEMS, ANALYSES

Well, not enough time to present ©

In addition to MWM, we also analyzed more graph problems

Well and the complete of the

GENERALIZABILITY TO OTHER
GRAPH PROBLEMS AND SETTINGS

Thank you for your attention



400 MHz

ON A HYBRID CPU-FPGA SETTING

THEORY-INSPIRED MWM

SC-SIMPLE log B = 18, L = 6 14.714
SC-SIMPLE log B = 12, L = 8 14.598
SC-OPT K = 32, L = 512 14.789
SC-OPT K = 2756 L = 178 14.789
SC-OPT K = 2756 L = 178 14.789
SC-OPT K = 32, L = 512 1

C-SIMPLE

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including n ing. social network analysis, computational sciences, and others. Two key challenges that hinde graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing updates, with millions of edges added or removed per second. Graph streaming algorithms ar

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

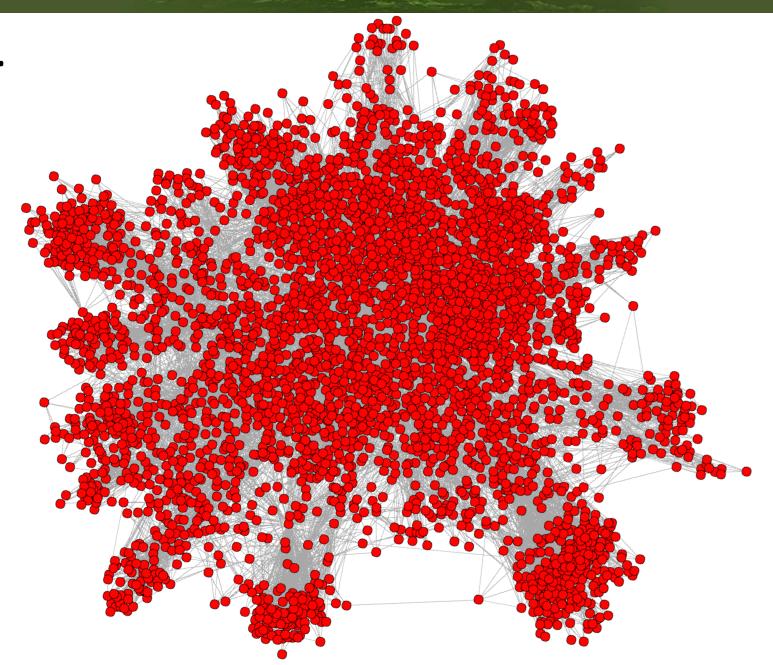
MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas, such as machine learning, computational sciences, medical applications, social network analysis, and many others. Various graphs such as web or social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing poses unione challenges for the runtime and the consumed power field







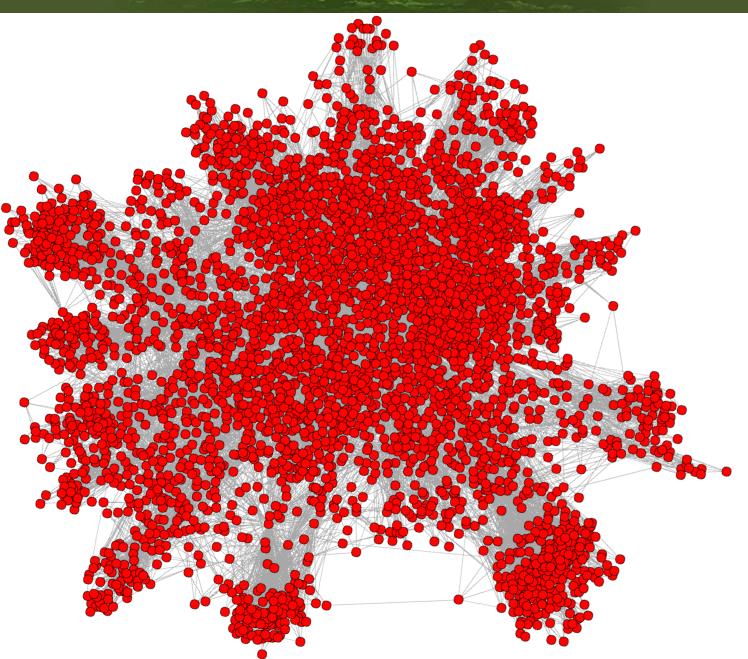










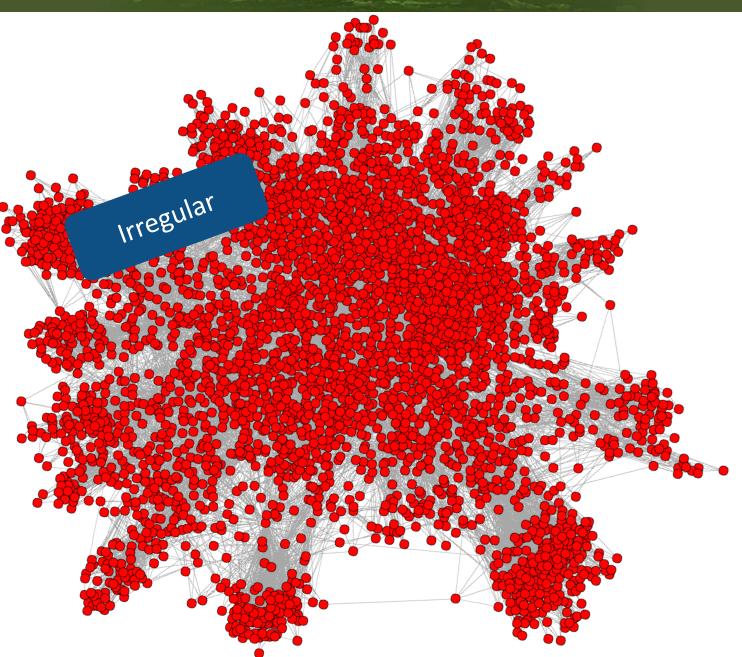




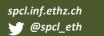






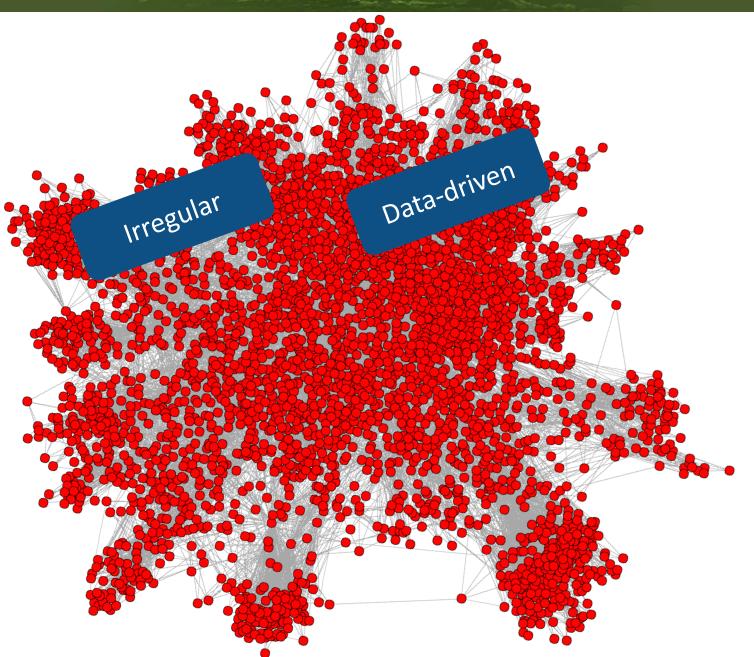










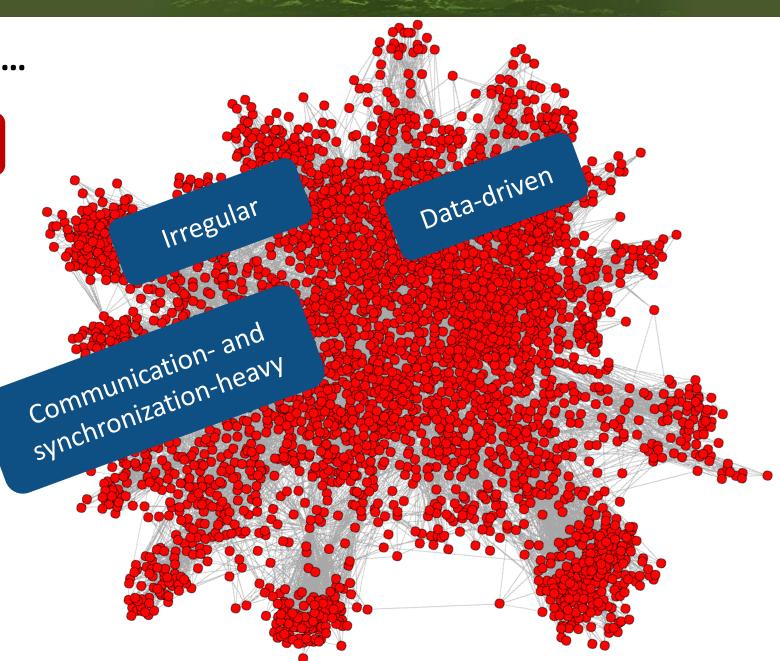










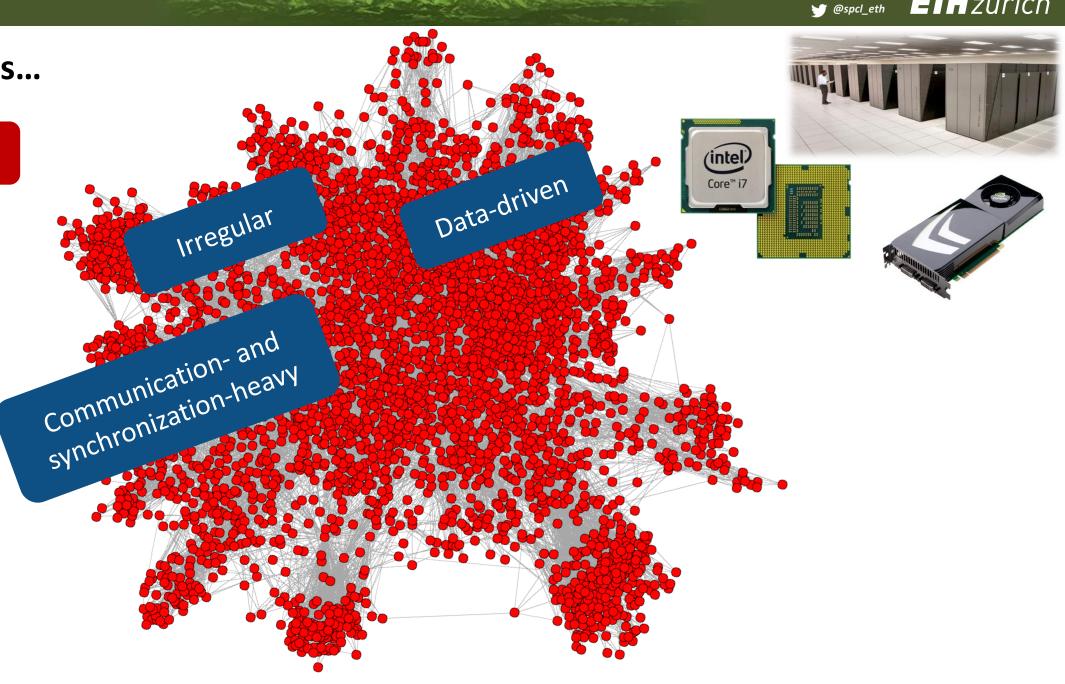


















250

Watts

Large graphs...



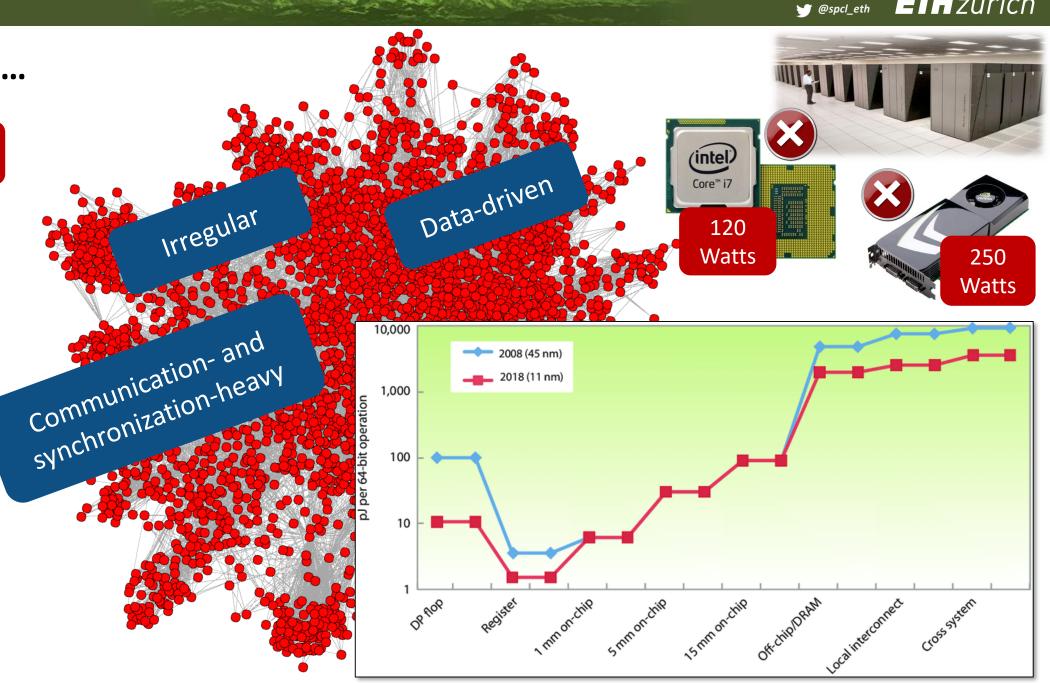




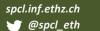




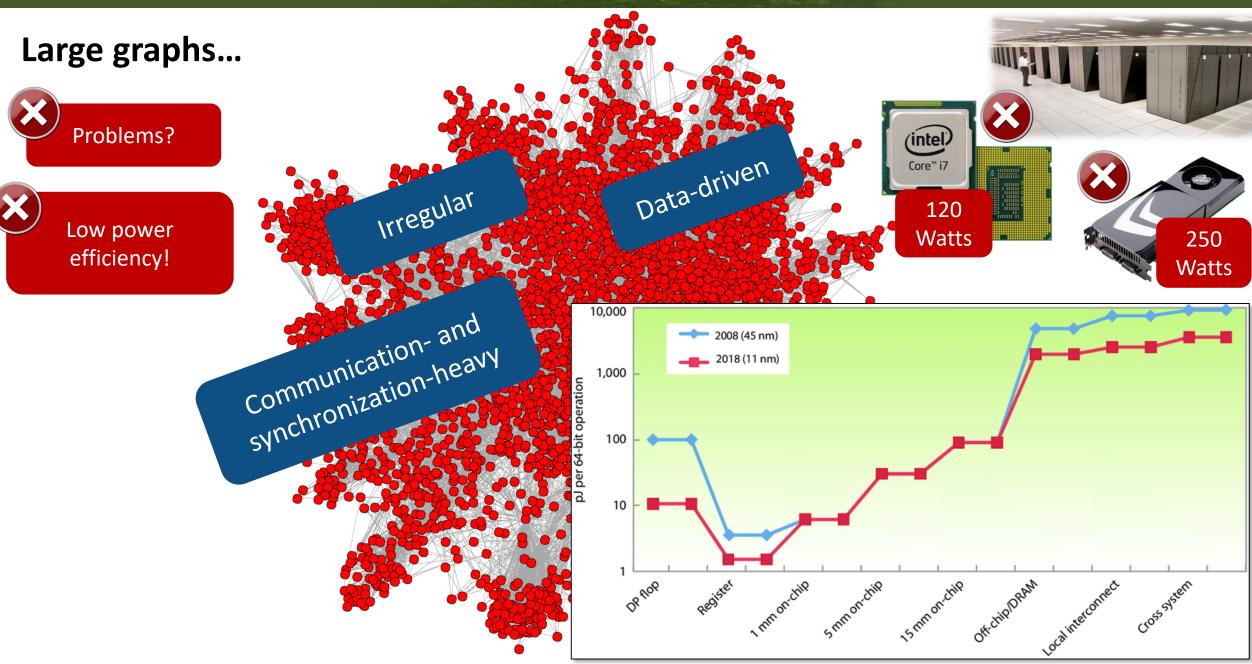




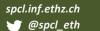














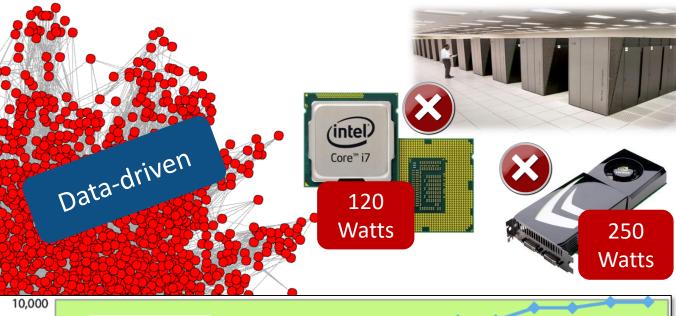


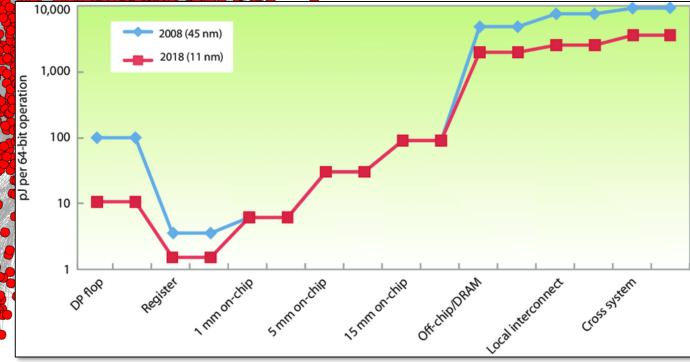
Problems?



Low power efficiency!















Problems?



Low power efficiency!



2008 (45 nm)

Graph Synchro CPU (MTEPS/Watt) [1] FPGA (MTEPS/Watt) [2] Problem **SSSP** 1.9 30.2 CC 0.5 48.1 **MST** 0.6 44.3 150 Offic

[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. ACM Symposium on Operating Syst. 2013.

[2] S. Zhou et al. High-throughput and Energy-efficient Graph Processing on FPGA. FCCM. 2016.









Problems?



Low power efficiency!



2008 (45 nm)

Graph Synchro CPU (MTEPS/Watt) [1] FPGA (MTEPS/Watt) [2] Problem **SSSP** 1.9 30.2 CC 0.5 48.1 **MST** 0.6 44.3 Offic 150

[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. ACM Symposium on Operating Syst. 2013.
[2] S. Zhou et al. High-throughput and Energy-

efficient Graph Processing on FPGA. FCCM. 2016.









Problems?



Low power efficiency!



ILLERNISI



20

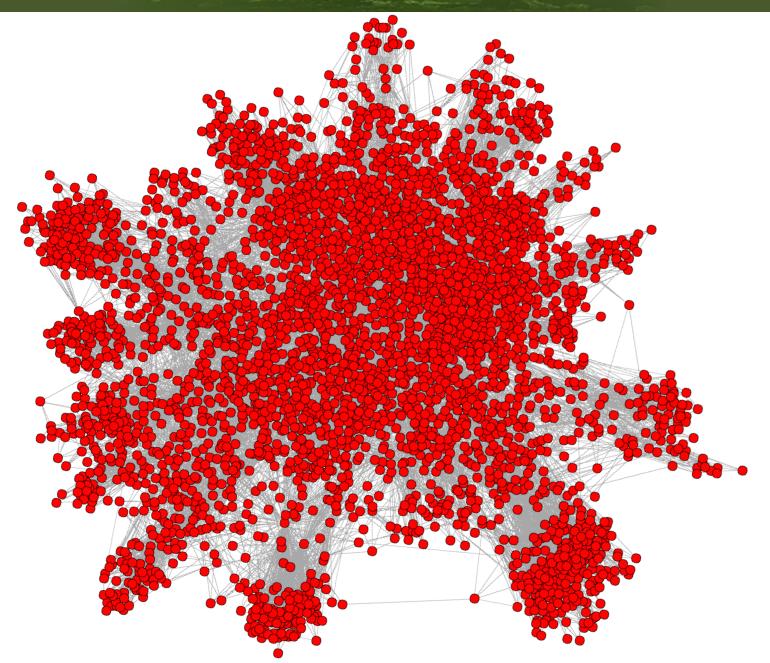
ynchr Omm'

Graph Problem	CPU (MTEPS/Watt) [1]	FPGA (MTEPS/Watt) [2]
SSSP	1.9	30.2
СС	0.5	48.1
MST	0.6	44.3
	10. 20.	72 Chr. 111.

[1] A. Roy et al. X-stream: Edge-Centric Graph Processing using Streaming Partitions. ACM Symposium on Operating Syst. 2013.
[2] S. Zhou et al. High-throughput and Energy-efficient Graph Processing on FPGA. FCCM. 2016.

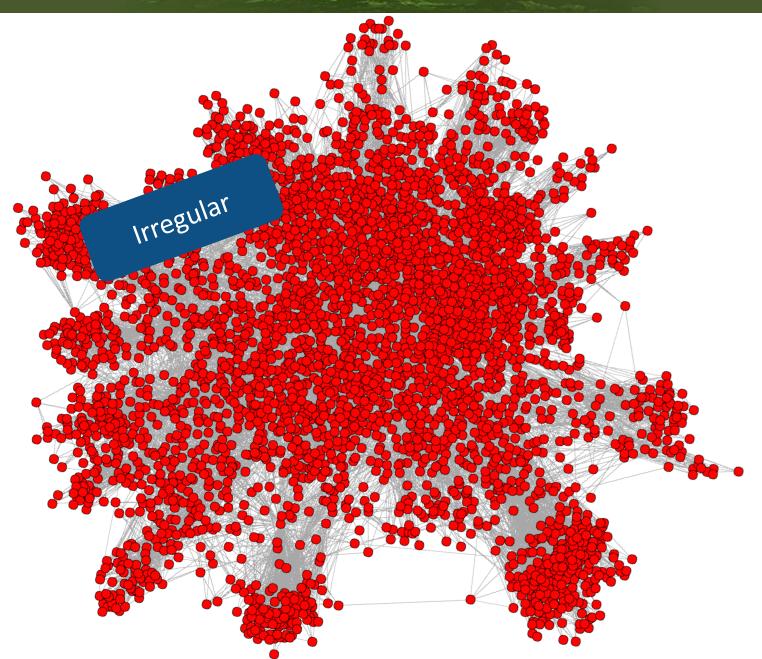






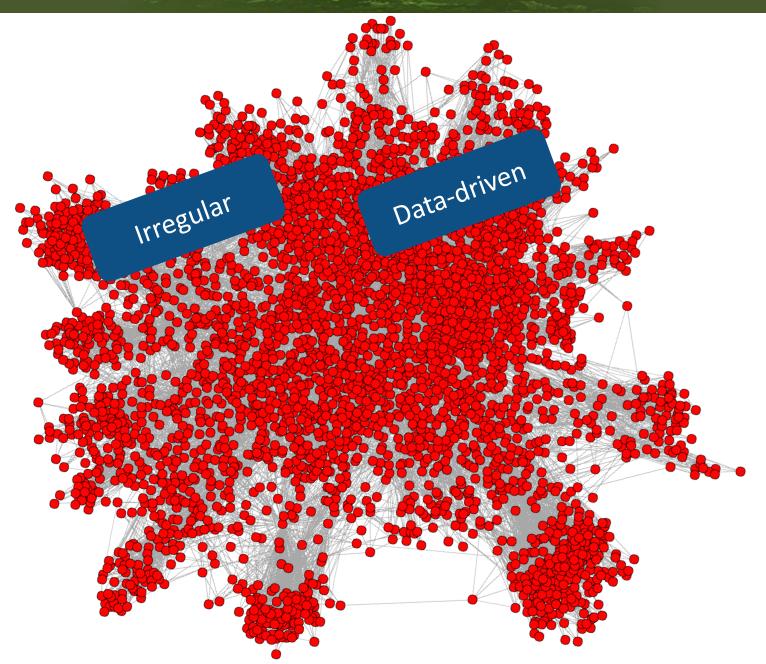






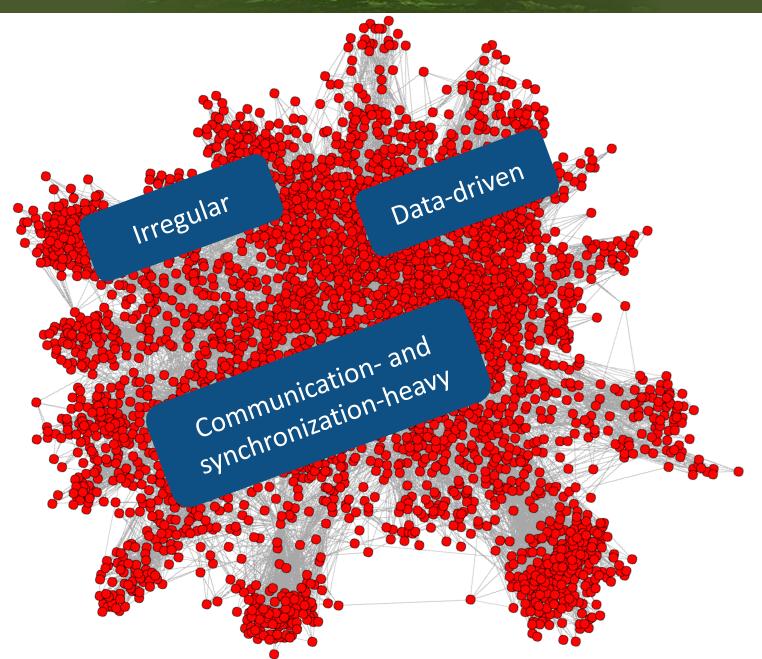






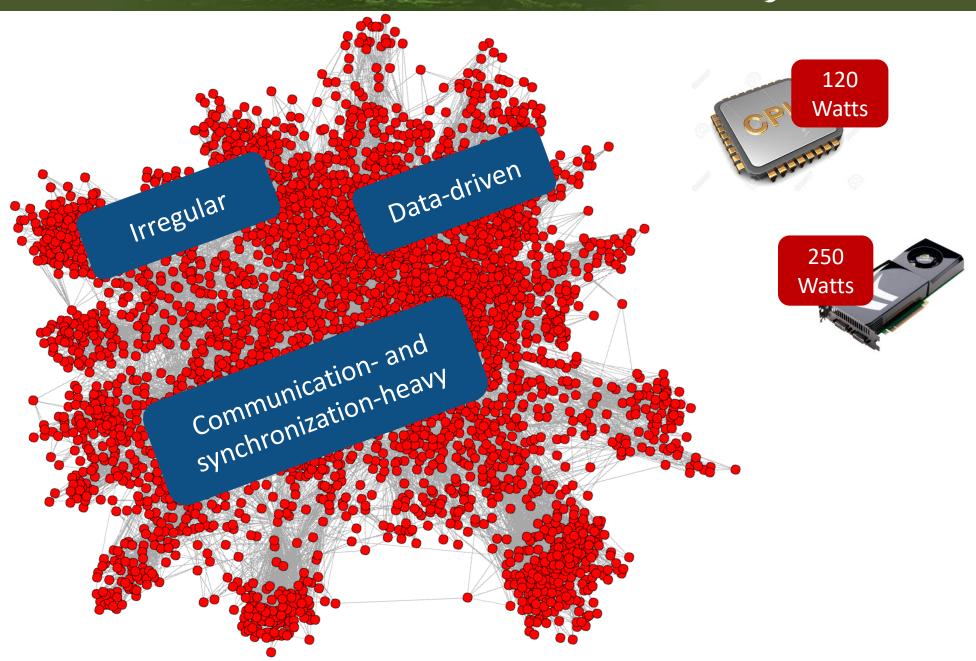












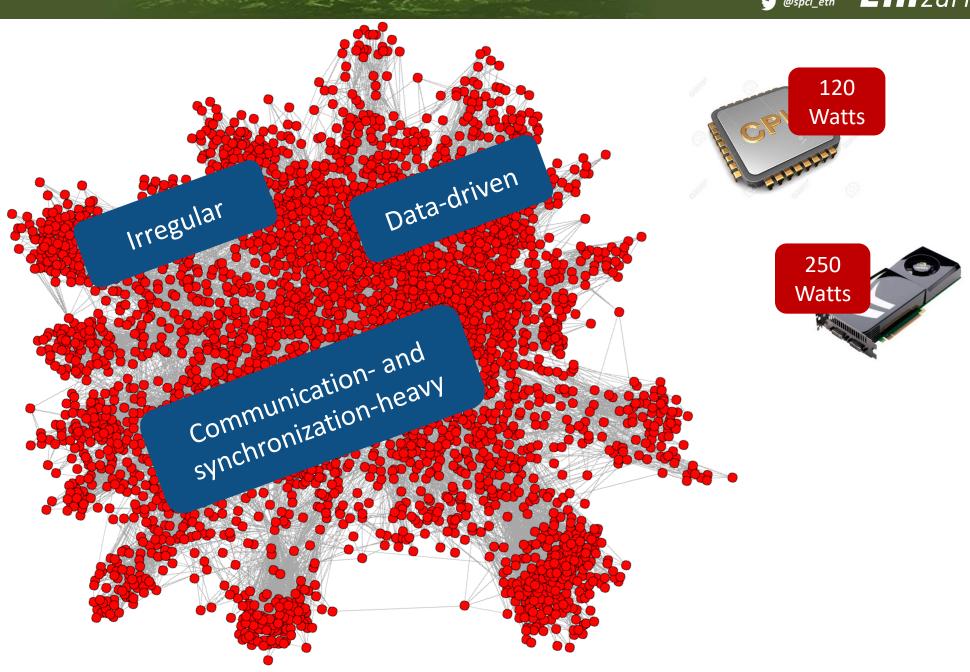












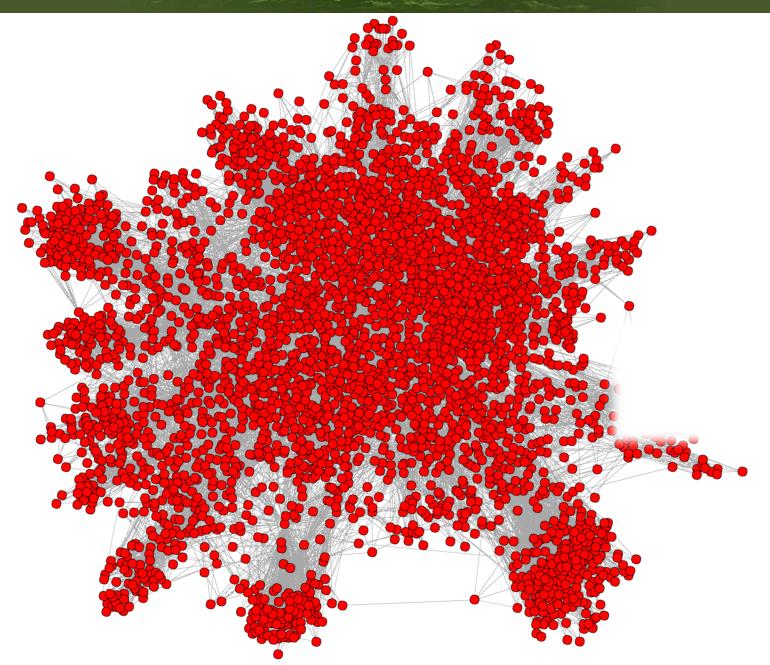










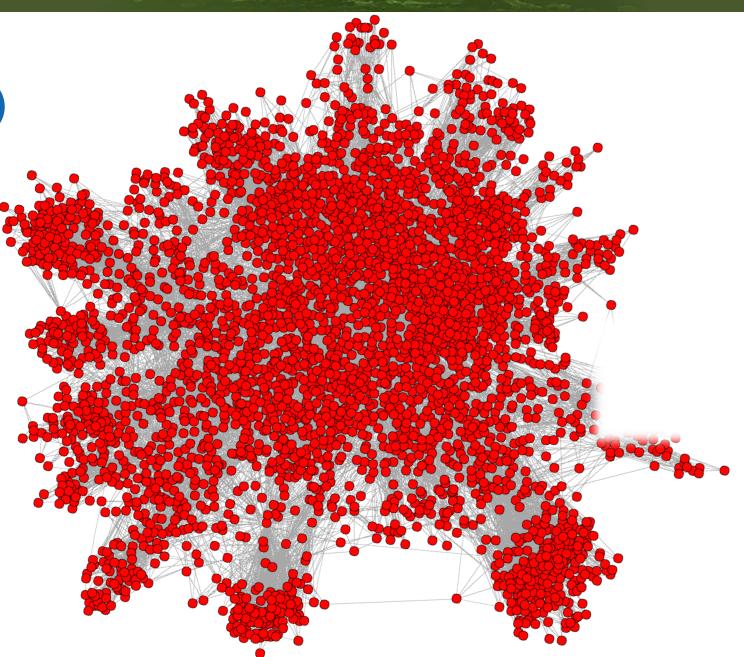








Why do we care?

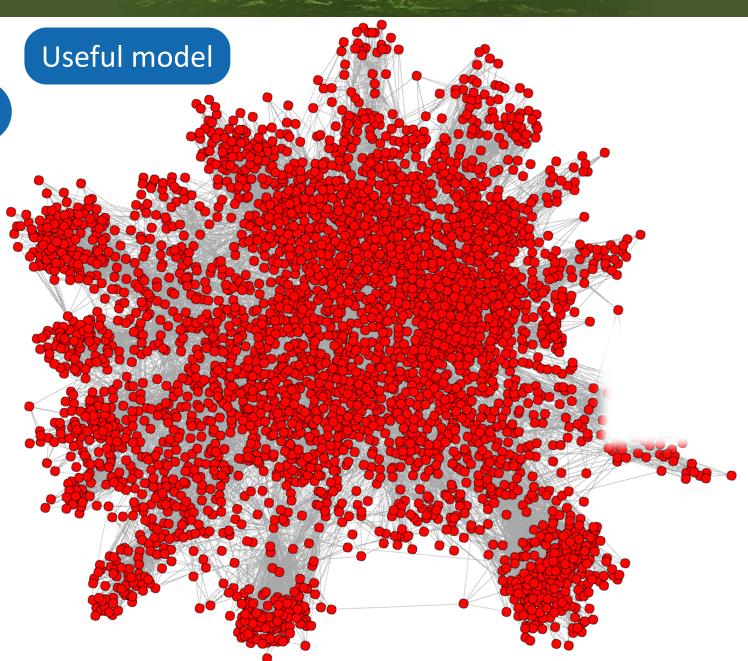








Why do we care?



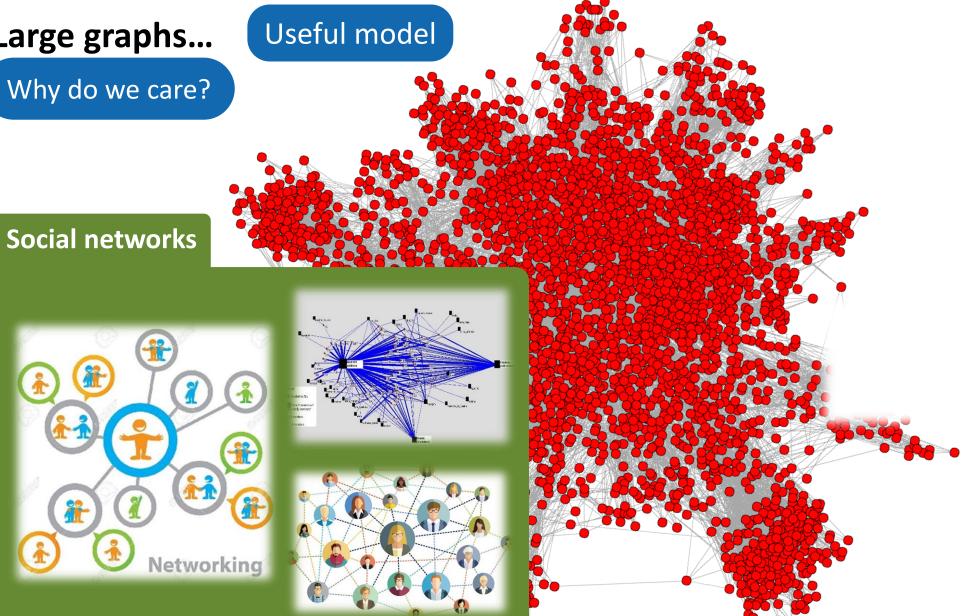








Why do we care?







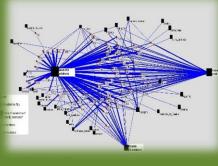


Why do we care?

Useful model



















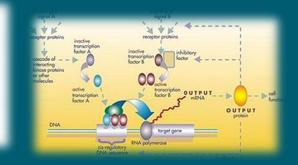


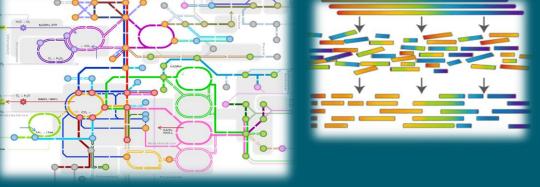
Why do we care?

Social networks



Useful model Engineering networks Biological networks











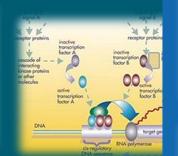
Social networks

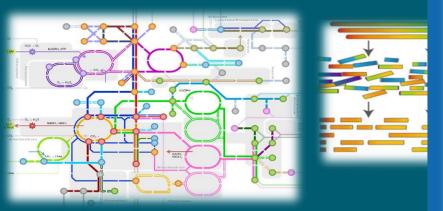
Why do we care?

Useful model

Biological networks



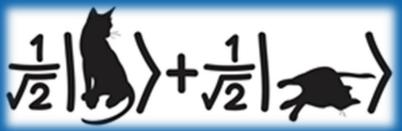


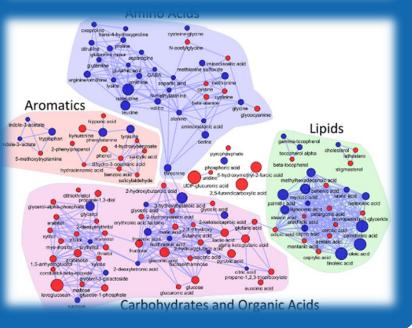




Physics, chemistry

Engineering networks











Why do we care?

Useful model

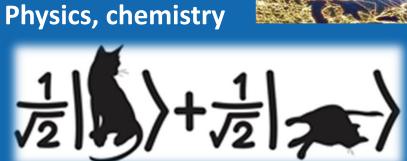
Biological networks

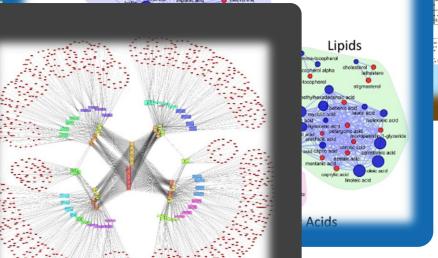


Communication networks



Engineering networks



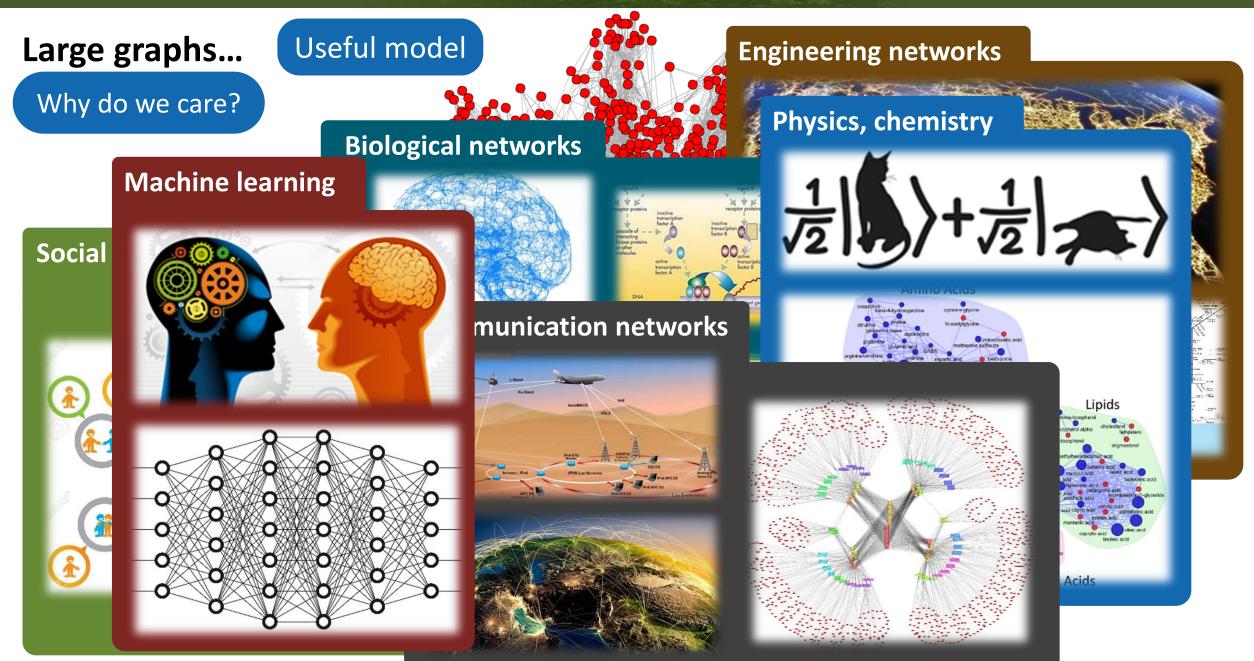


Social networks















Useful model

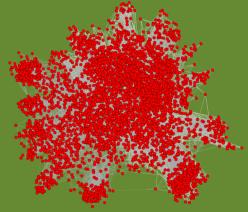
Why do we care?

...even philosophy ©

Engineering networks

hysics, chemistry





FOSDEM 2016 / Schedule / Events / Developer rooms / Graph Processing / Modeling a Philosophical Inquiry: from MySQL to a graph database

Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

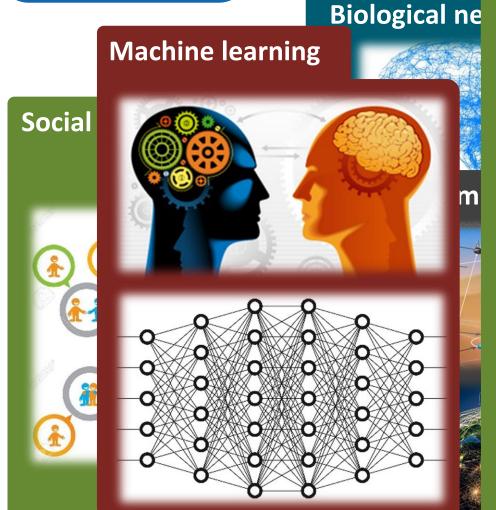
A Track: Graph Processing devroom ♠ Room: AW1.126

Day: Saturday ▶ Start: 12:45

■ End: 13:35



Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version

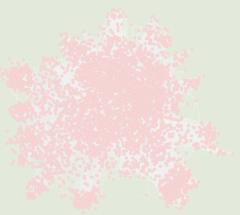












Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

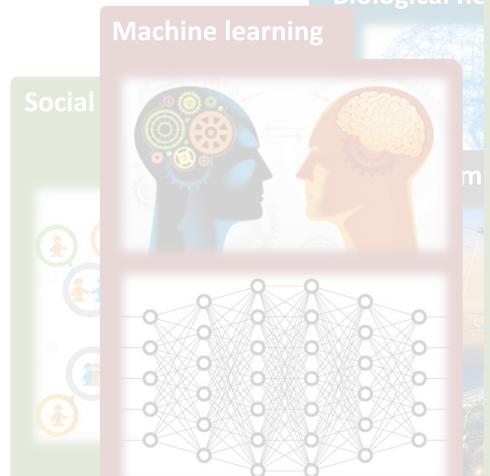
A Track: Graph Processing devroom

↑ Room: AW1.126

▶ Start: 12:45

■ End: 13:35







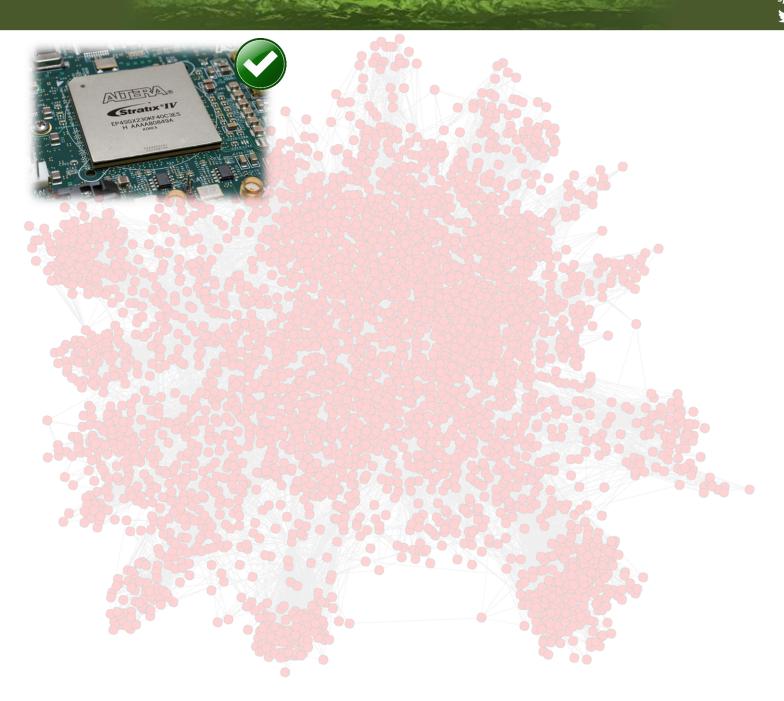


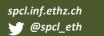




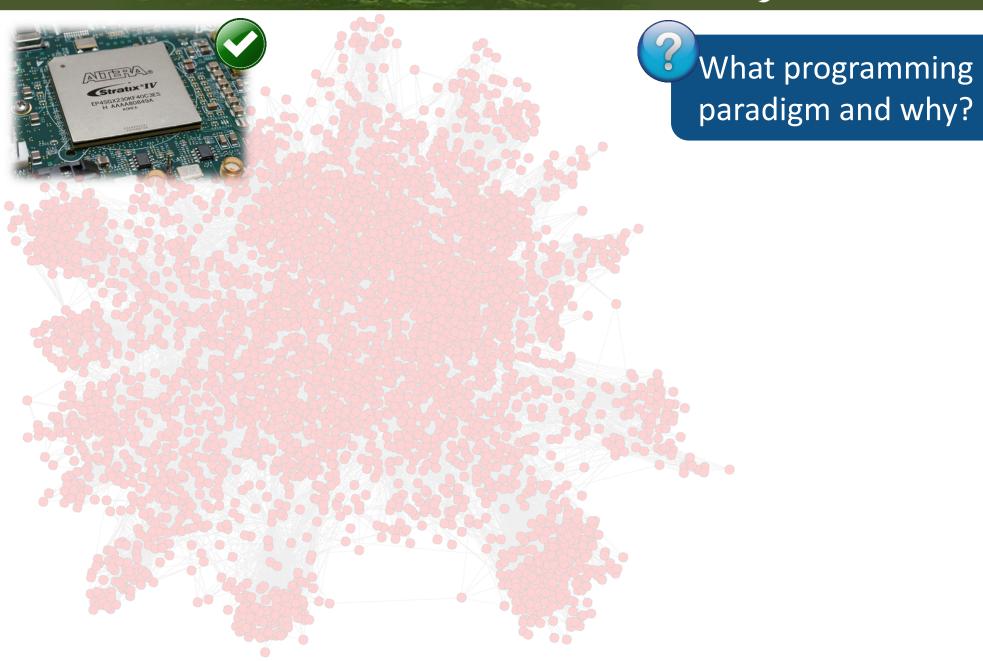








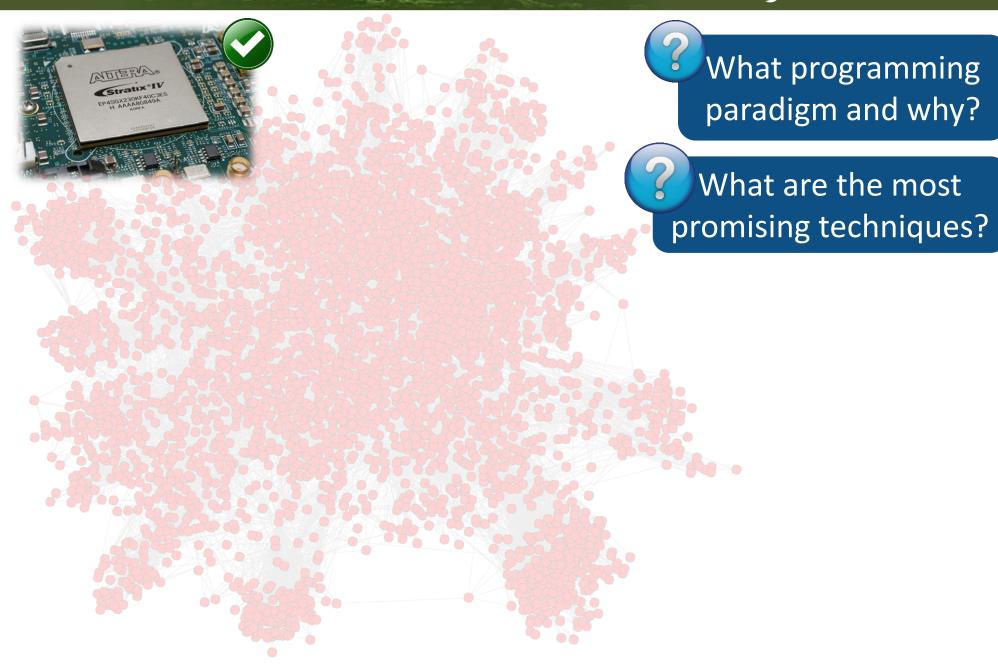




















- What programming paradigm and why?
- What are the most promising techniques?

Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs







<u>Part 1</u>: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

What programming paradigm and why?

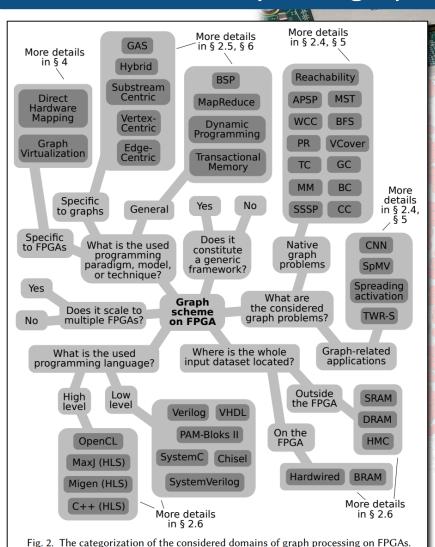
What are the most promising techniques?







Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs



What programming paradigm and why?

What are the most promising techniques?

7 paradigms







<u>Part 1</u>: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

More details in § 2.6

More details

Fig. 2. The categorization of the considered domains of graph processing on FPGAs.

More deta in § 4	Reference (scheme name)	Venue	Generic Design ¹		Programming Model or Technique ⁴ (§ 2.5)		Multi FPGAs ⁴	Input Location ⁵	n^{\dagger}	m^{\dagger}
Direct Hardware	Kapre [71] (GraphStep)	FCCM'06	Ô	spreading activation* [82]	BSP	unsp.	Ů	BRAM	220k	550k
Mapping	Weisz [92] (GraphGen)	FCCM'14	Ô	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	•	DRAM	110k	221k
Graph Virtualizati	Kapre [70] (GraphSoC)	ASAP'15	Ô	SpMV	Vertex-Centric, BSP	C++ (HLS)	Ů	BRAM	17k	126k
Sp	Dai [40] (FPGP)	FPGA'16	Ô	BFS	None	unsp.	Ô	DRAM	41.6M	1.4B
to (Oguntebi [93] (GraphOps)	FPGA'16	Ô	BFS, SpMV, PR, Vertex Cover	None	MaxJ (HLS)	•	BRAM	16M	128M
Specific to FPGAs	Zhou [134]	FCCM'16	C	SSSP, WCC, MST	Edge-Centric	unsp.	•	DRAM	4.7M	65.8M
Yes	Engelhardt [49] (GraVF)	FPL'16	Ô	BFS, PR, SSSP, CC	Vertex-Centric	Migen (HLS)	•	BRAM	128k	512k
No m	Dai [41] (ForeGraph)	FPGA'17	Ô	PR, BFS, WCC	None	unsp.	Ů	DRAM	41.6M	1.4B
Wha	Zhou [136]	SBAC-PAD'17	O	BFS, SSSP	Hybrid (Vertex- and Edge-Centric)	unsp.	•	DRAM	10M	160M
progran	Ma [85]	FPGA'17	Ô	BFS, SSSP, CC, TC, BC	Transactional Memory [16, 59]	System- Verilog	Ô	DRAM	24M	58M
Hi	Lee [79] (ExtraV)	FPGA'17	Ô	BFS, PR, CC, AT* [60]	Graph Virtualization	C++ (HLS)	•	DRAM	124M	1.8B
	Zhou [135]	CF'18	\(\cdot\)	SpMV, PR	Edge-Centric, GAS	unsp.	•	DRAM	41.6M	1.4B
	Yang [125]	report (2018)	Ô	BFS, PR, WCC	None	OpenCL			4.85M	69M
	Yao [127]	report (2018)	Ů	BFS, PR, WCC	None	unsp.	•	BRAM	4.85M	69M
I.	- J									

What programming paradigm and why?

What are the most promising techniques?

7 paradigms

~15 FPGA graph processing frameworks



Fig. 2. The categorization of the considered domains of graph processing on FPGAs.





<u>Part 1</u>: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

More deta in § 4		erence ne name)	Venue	Generio Design ¹		Programming Mod) or Technique ⁴ (§ 2					n^{\dagger}	m^{\dagger}
Direct Hardward	Kapre (Grap l		FCCM'06	Ô	spreading activation* [82]	BSP	unsp.	Ů	BRAM	2	220k	550k
Mapping	Weisz		FCCM'14	Ů	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	•	DRAM	1	110k	221k
Graph Virtualizati	Kapre (Grap Dai [4	Babb [4] Dandalis [43	report (1 3] report (1 116] report (2	999) 📭	SSSP SSSP SSSP	None None None	Verilog unsp. VHDL	∆ ∆ ••	Hardwired Hardwired BRAM	512 2048 64	2051 32k 4096	126k 1.4B
Specific Specific	(FPGI Ogunt (Grap	Mencer [87]		•	Reachability, SSSP	None	PAM- -Bloks II	•	Hardwired (3-state buffers)	88	7744	
Specific to FPGAs	7hou	Sridharan[1		l'09 ■	APSP SSSP	Dynamic Program. None	unsp. VHDL	1 4	DRAM BRAM	unsp.	88	5.8M
Yes D	(GraV	Wang [121] Betkaoui [2] Jagadeesh [4] Betkaoui [2]	65] report (2	#	BFS GC SSSP APSP	None Vertex-Centric None Vertex-Centric	SystemC Verilog VHDL Verilog	∆ ••	DRAM DRAM Hardwired ≈ DRAM	65.5k 300k 128 38k	1M 3M 466 72M	512k 1.4B
Wha		Betkaoui[23 Attia [2]	ASAP'12		BFS BFS	Vertex-Centric Vertex-Centric	Verilog VHDL	Õ O	DRAM DRAM	16.8M 8.4M	1.1B 536M	60 <i>N</i>
Hi	Ma [8 Lee [7	(CyGraph) Ni [91]	report (2	•	BFS	None	Verilog	•	DRAM, SRAM	16M	512M	58 <i>N</i>
le	(Extra	Zhou [132] Zhou [133] Umuroglu [IPDPS'15 ReConFi 117] FPL'15	•	SSSP PR BFS	None Edge-Centric None	unsp. unsp. Chisel	•	DRAM DRAM ≈ DRAM	1M 2.4M 2.1M	unsp. 5M 65M	1.8B
	Yao [1	Lei [80] Zhang [129] Zhang [130]	report (2 FPGA'17 FPGA'18	016) •	SSSP BFS BFS	None MapReduce None	unsp. unsp. unsp.	•	DRAM HMC HMC	23.9M 33.6M	58.2M 536.9M	69 <i>M</i>
	C++ (HL	Kohram [76] Besta [13]	FPGA'18 FPGA'19	*	BFS MM	None Substream-Centric	unsp. Verilog	**	HMC DRAM	4.8M	117M	

What programming paradigm and why?

What are the most promising techniques?

7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms



Fig. 2. The categorization of the considered domains of graph processing on FPGAs.





<u>Part 1</u>: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

e deta n § 4		erence ne name)	Venue	Generic Design		Programming Mod) or Technique ⁴ (§ 2			ulti Inpu SAs ⁴ Locati		n^{\dagger}	n
irect	•	hStep)	CCM'06	Ô	spreading activation* [82]	BSP	unsp.		BRAM	2	220k	55
pping	Weisz (Grap l	[92] hGen) F	CCM'14	Ô	TRW-S*, CNN* [112]	Vertex-Centric	unsp.	•	DRAM		110k	22
raph alizati	Kapre	Babb [4]	report (1	996) 📭	SSSP	None	Verilog	Ů	Hardwired	512	2051	1
	(Grap	Dandalis [43] report (1	999) 📭	SSSP	None	unsp.	L	Hardwired	2048	32k	ď
	Dai [4	Tommiska [1	16] report (2	.001) 📭	SSSP	None	VHDL	•	BRAM	64	4096	1
to	(FPGI Ogunt (Grap	Mencer [87]	FPL'02	•	Reachability, SSSP	None	PAM- -Bloks II	•	Hardwired (3-state buffers)	88	7744	н
	Zhou	Bondhugula	[27] IPDPS'0	6	APSP	Dynamic Program.	unsp.	•	DRAM	unsp.		5
		Sridharan[11	0] TENCO	V'09 📭	SSSP	None	VHDL		BRAM	64	88	ij
	Engell	Wang [121]	ICFTP'10	0	BFS	None	SystemC		DRAM	65.5k	1M	
	(GraV	Betkaoui [21] FTP'11		GC	Vertex-Centric	Verilog	L	DRAM	300k	3M	d
	Dai [4	Jagadeesh [6	5] report (2	011) 📭	SSSP	None	VHDL	•	Hardwired	128	466	5
m	(Fore	Betkaoui [22] FPL'12		APSP	Vertex-Centric	Verilog	L	$\approx DRAM$	38k	72M	
	Zhou	Betkaoui[23]	ASAP'12	•	BFS	Vertex-Centric	Verilog	Ů	DRAM	16.8M	1.1B	
Wha gran	Ma [8]	Attia [2] (CyGraph)	IPDPS'1	4 •	BFS	Vertex-Centric	VHDL	Ů	DRAM	8.4M	536M	-1
	Lee [7	Ni [91]	report (2	014)	BFS	None	Verilog	•	DRAM, SRAM	16M	512M	- 1
ie	(Extra	Zhou [132]	IPDPS'1	•	SSSP	None	unsp.	•	DRAM	1M	unsp.	
	Zhou	Zhou [133]	ReConFi	_	PR	Edge-Centric	unsp.	•	DRAM	2.4M	5 M	
	- 1	Umuroglu [1			BFS	None	Chisel	_	$\approx DRAM$	2.1M	65M	
	Yang [Lei [80]	report (2		SSSP	None	unsp.		DRAM	23.9M	58.2M	
	Yao [1	Zhang [129]	FPGA'17	•	BFS	MapReduce	unsp.		HMC	33.6M	536.9M	
	J	Zhang [130]	FPGA'18	•	BFS	None	unsp.	_	HMC			f
C-	++ (HL	Kohram [76]		•	BFS	None	unsp.	•	HMC			
		Besta [13]	FPGA'19	•	MM	Substream-Centric	Verilog	•	DRAM	4.8M	117M	. [

What programming paradigm and why?

What are the most promising techniques?

Key techniques, challenges, features, ...

7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms







<u>Part 1</u>: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

Reference Generic Venue More deta Design¹ (scheme name) Selected MWM-related parts are Kapre [71] FCCM'06 (GraphStep) in the FPGA paper, the rest is in... Hardwa Weisz [92] FCCM'14 (GraphGen) Graph Virtualizat Kapre Babb [4] report (1996) 555P None Hardwired 126k (Grap Dandalis [43] Ô report (1999) **SSSP** 32k None Hardwired 2048 unsp. Dai [4 Tommiska [116] 4096 report (2001) **SSSP** None **VHDL** BRAM 64 1.4B (FPGI Hardwired Reachability, PAM-Ogunt Mencer [87] FPL'02 (3-state 88 7744 None **SSSP** -Bloks II 28M buffers) (Grap Specific Bondhugula [27] IPDPS'06 **APSP** Dynamic Program. DRAM unsp. unsp. Zhou .8M SSSP Sridharan[110] **VHDL BRAM** TENCON'09 None 64 88 Engell 1M 512k Wang [121] **BFS DRAM** 65.5k ICFTP'10 None **SystemC** (Gra Betkaoui [21] GC FTP'11 Vertex-Centric **DRAM** 300k 3M Verilog Dai [Jagadeesh [65] SSSP 466 **VHDL** Hardwired 128 report (2011) None 1.4B (Fore Betkaoui [22] FPL'12 **APSP** Vertex-Centric Verilog ≈ DRAM 38k 72M 1.1B 60M Betkaoui[23] ASAP'12 BFS Vertex-Centric Verilog DRAM 16.8M Zhou Attia [2] Wh IPDPS'14 **BFS** Vertex-Centric **VHDL** DRAM 8.4M 536M progran (CyGraph) Ma [8 58M DRAM, **BFS** 16M 512M report (2014) None Verilog **SRAM** Lee 1.8B IPDPS'15 **SSSP DRAM** 1M Zhou [132] None unsp. unsp. (Extra Zhou [133] ReConFig'15 PR **Edge-Centric DRAM** 2.4M 5M unsp. Umuroglu [117] FPL'15 **BFS** Chisel ≈ DRAM 2.1M 65M None Yang Lei [80] **SSSP DRAM** 23.9M 58.2M report (2016) None unsp. Zhang [129] FPGA'17 **BFS HMC** 33.6M 536.9M MapReduce unsp. Zhang [130] FPGA'18 **BFS HMC** None unsp. FPGA'18 **BFS HMC** Kohram [76] None unsp. C++ (H FPGA'19 MM Substream-Centric **DRAM** Besta [13] Verilog 4.8M 117M

What programming paradigm and why?

What are the most promising techniques?

Key techniques, challenges, features, ...

7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

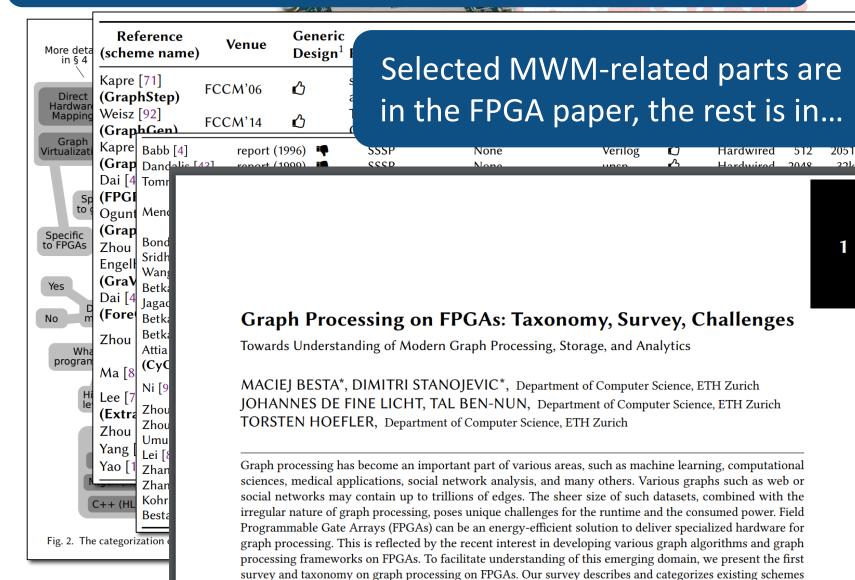
Fig. 2. The categorization of the considered domains of graph processing on FPGAs.







<u>Part 1</u>: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs



What programming paradigm and why?

What are the most promising techniques?

Key techniques, challenges, features, ...

7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms







Part 1: To understand the domain well, we conducted a detailed analysis of graph processing on FPGAs

Reference Generic Venue More deta Design¹ (scheme name) Selected MWM-related parts are Kapre [71] FCCM'06 (GraphStep) in the FPGA paper, the rest is in... Weisz [92] FCCM'14 (GraphGen) Kapre Babb [4] report (1996) 555P None (Grap Dandalia [42] CCCD Dai [4] Tomi (FPGI Ogunt Men (Grap https://arxiv.org/abs/... Zhou Engell (GraV Betk Dai [4 Jagad (Fore **Graph Processing on FPGAs: Taxonomy, Survey, Challenges** Zhou Towards Understanding of Modern Graph Processing, Storage, and Analytics Attia Ma [8 MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich Lee JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich (Extra TORSTEN HOEFLER, Department of Computer Science, ETH Zurich Zhou Yang Graph processing has become an important part of various areas, such as machine learning, computational Yao Zhan sciences, medical applications, social network analysis, and many others. Various graphs such as web or Zhar social networks may contain up to trillions of edges. The sheer size of such datasets, combined with the irregular nature of graph processing, poses unique challenges for the runtime and the consumed power. Field Programmable Gate Arrays (FPGAs) can be an energy-efficient solution to deliver specialized hardware for Fig. 2. The categorization graph processing. This is reflected by the recent interest in developing various graph algorithms and graph

processing frameworks on FPGAs. To facilitate understanding of this emerging domain, we present the first survey and taxonomy on graph processing on FPGAs. Our survey describes and categorizes existing schemes

What programming paradigm and why?

What are the most promising techniques?

Key techniques, challenges, features, ...

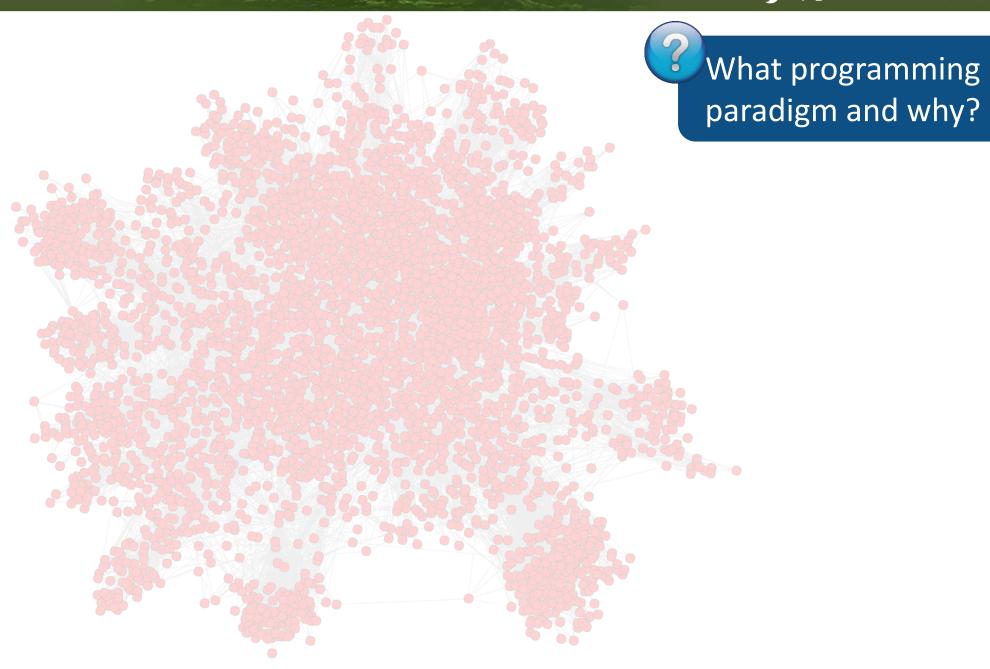
7 paradigms

~15 FPGA graph processing frameworks

~25 FPGA accelerators for specific algorithms

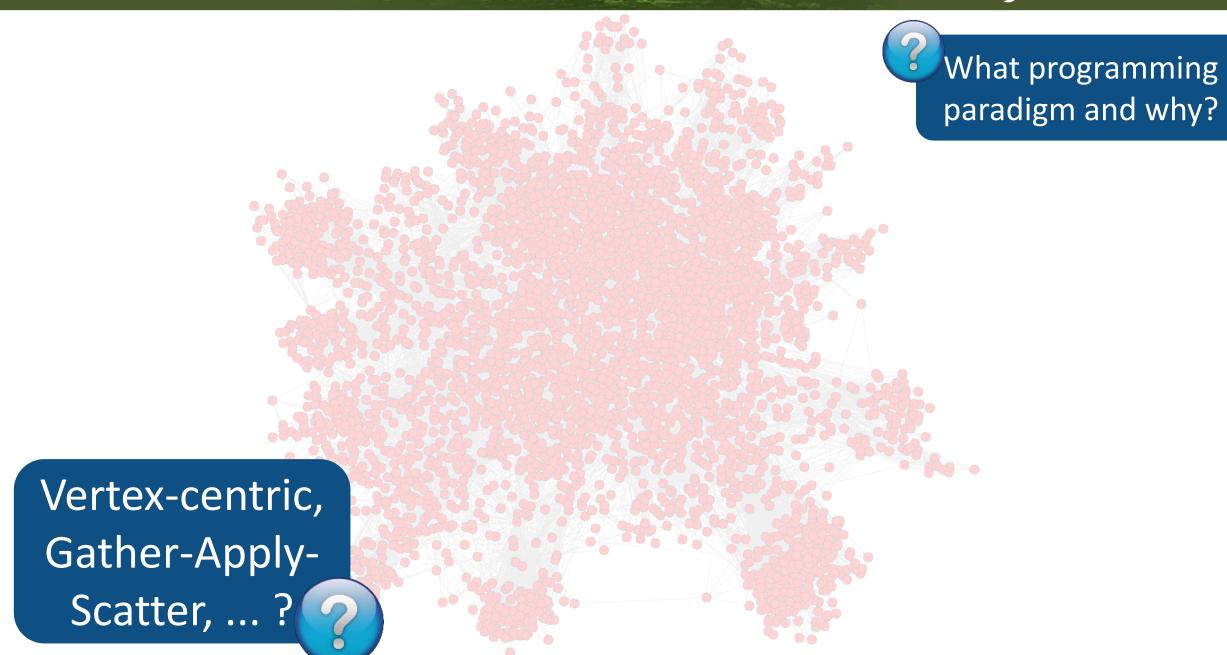




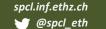




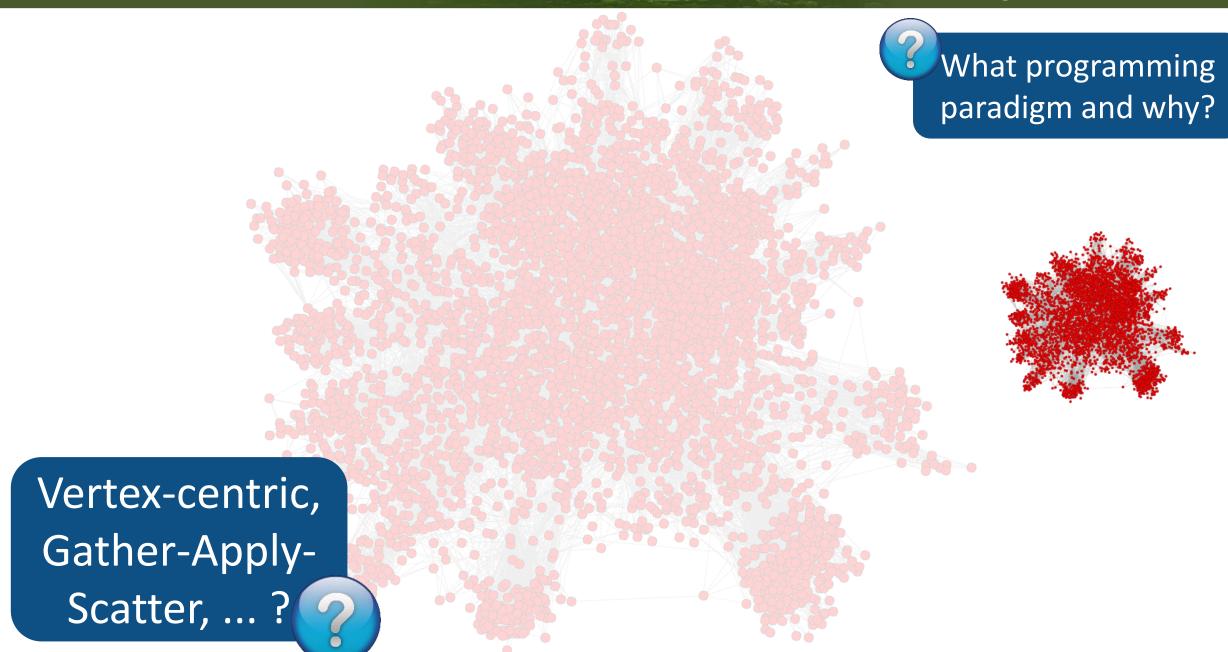








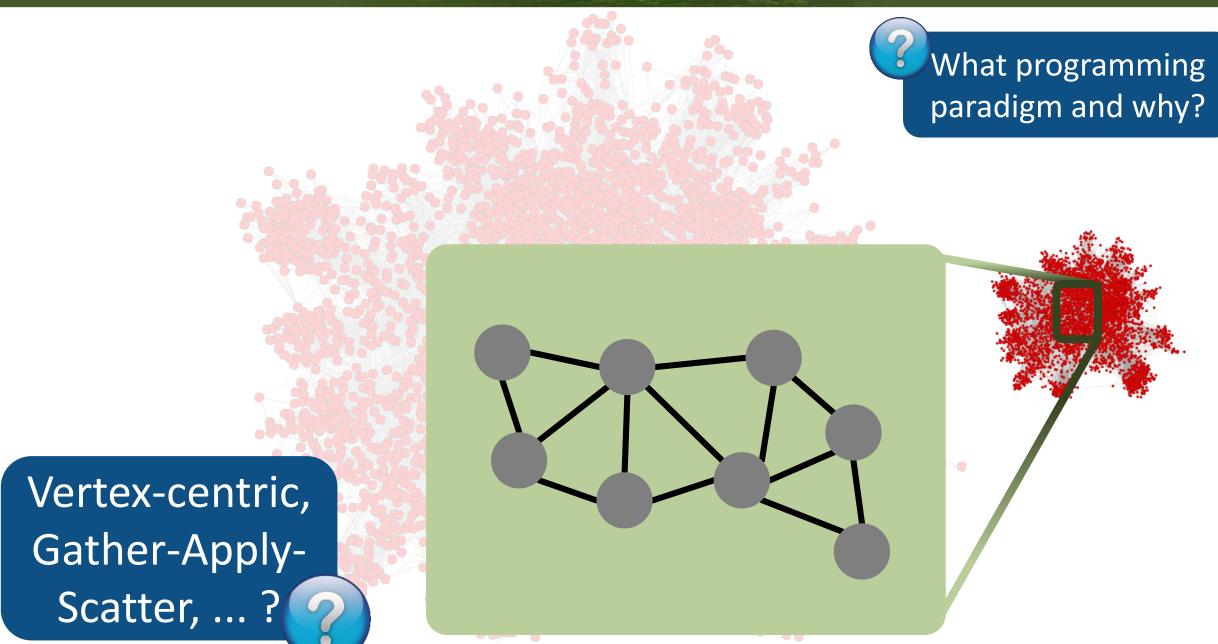










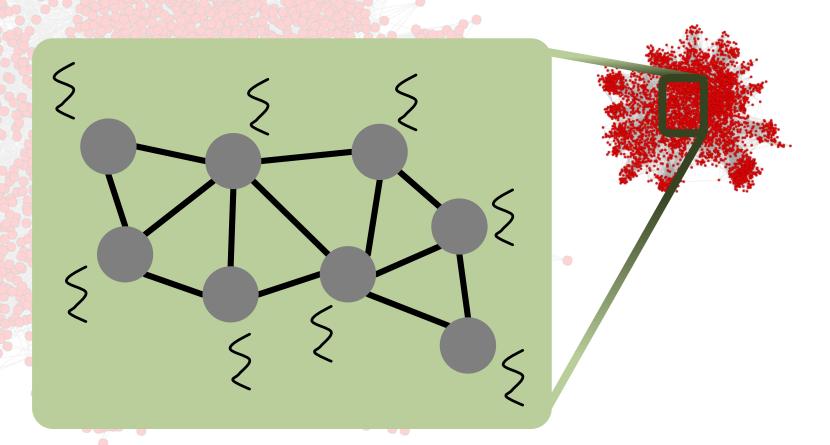










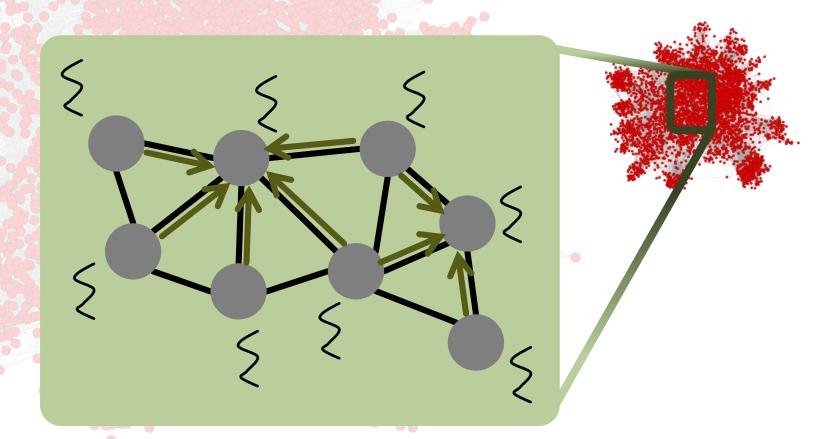










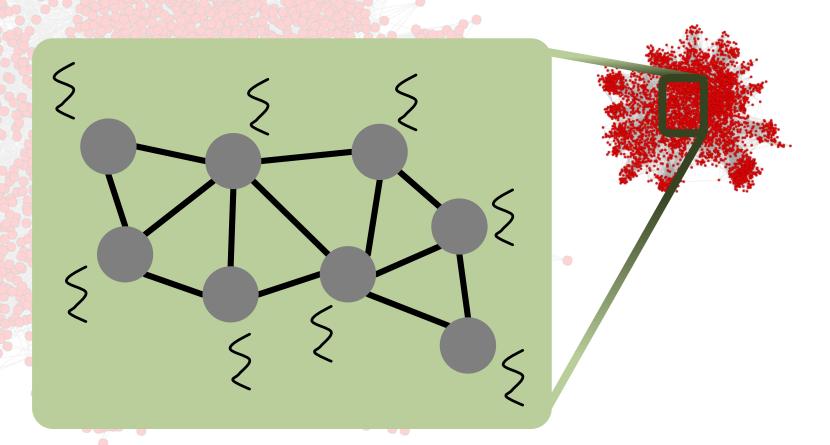










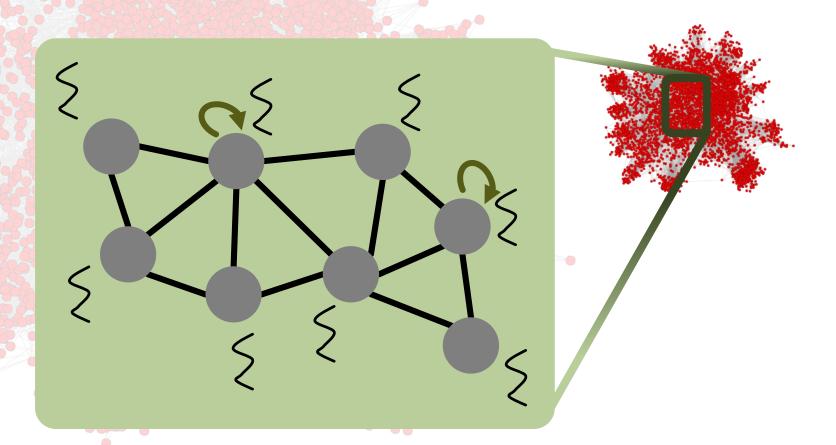










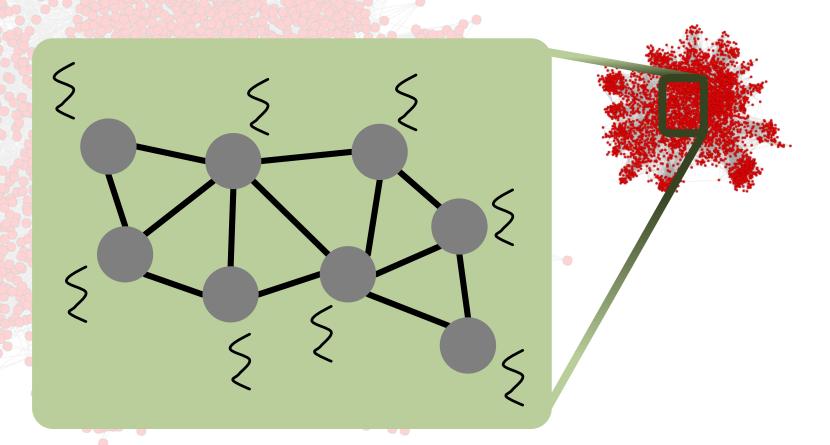










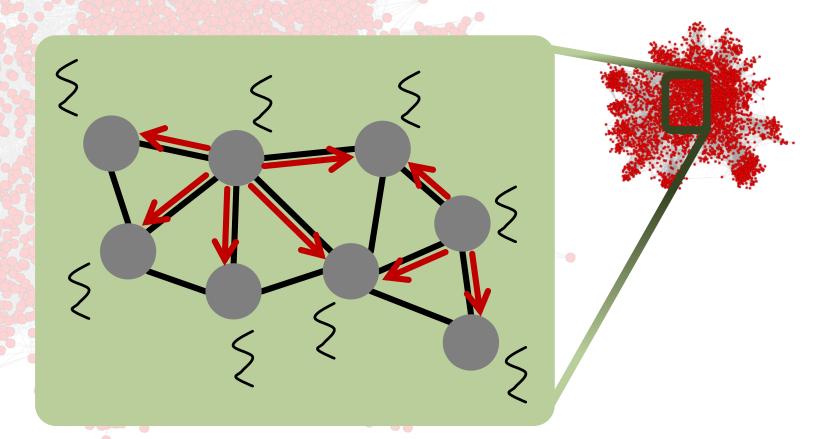










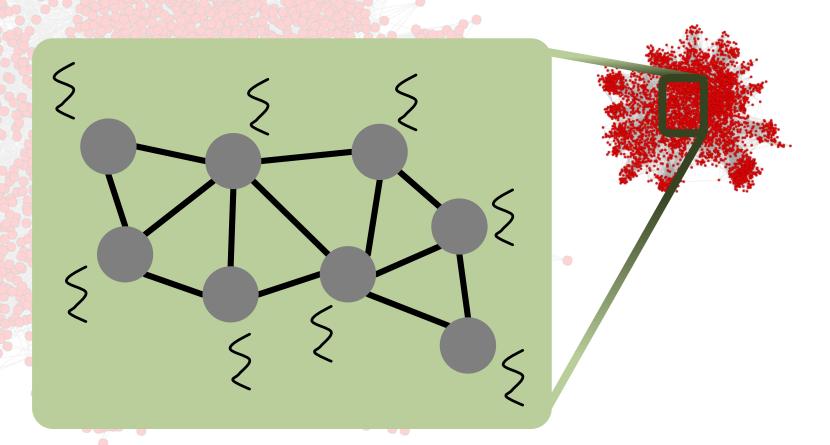












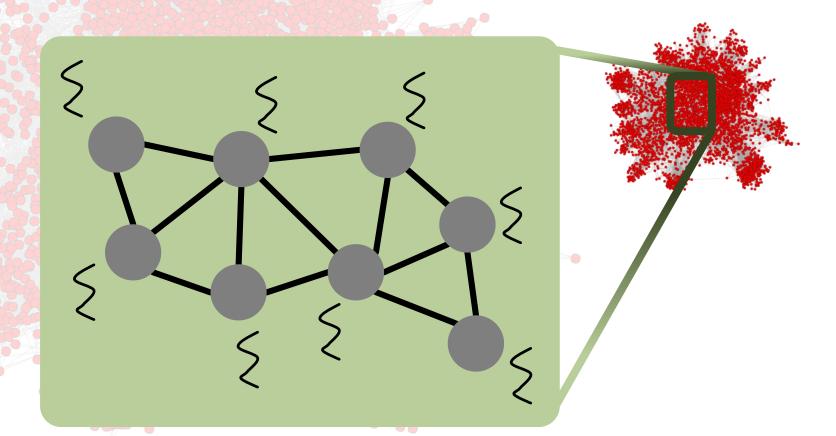












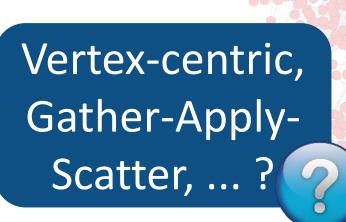


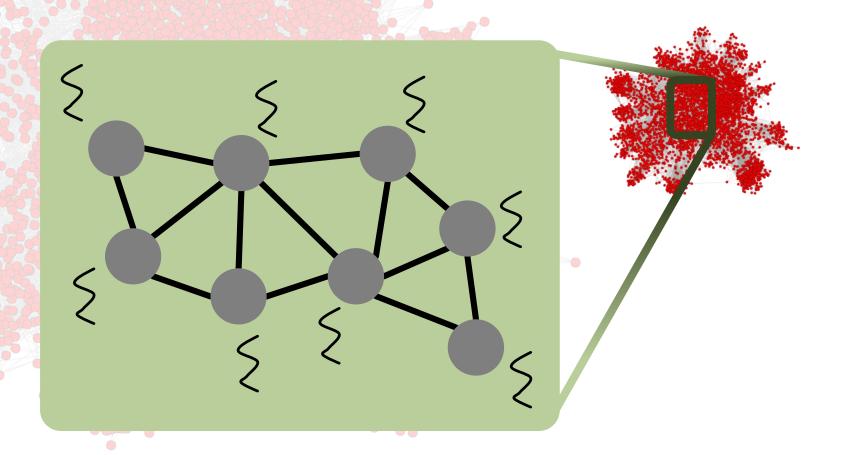




Well...

What programming paradigm and why?







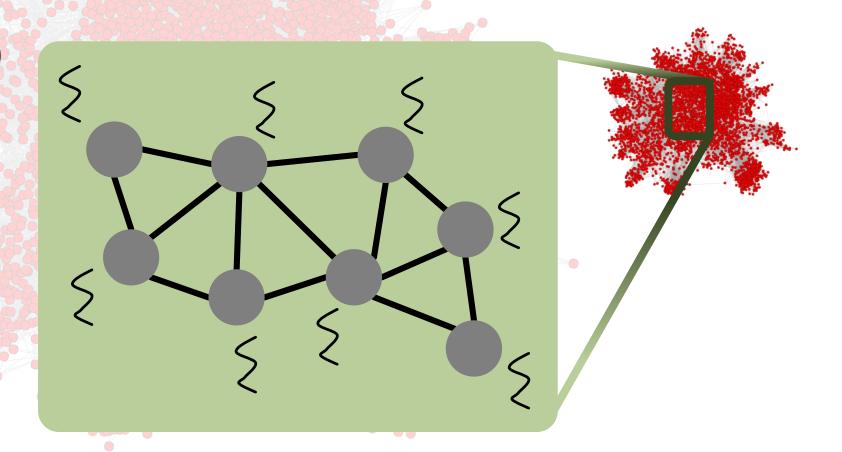




Well...

What programming paradigm and why?

Assumes the whole input graph is accessible...







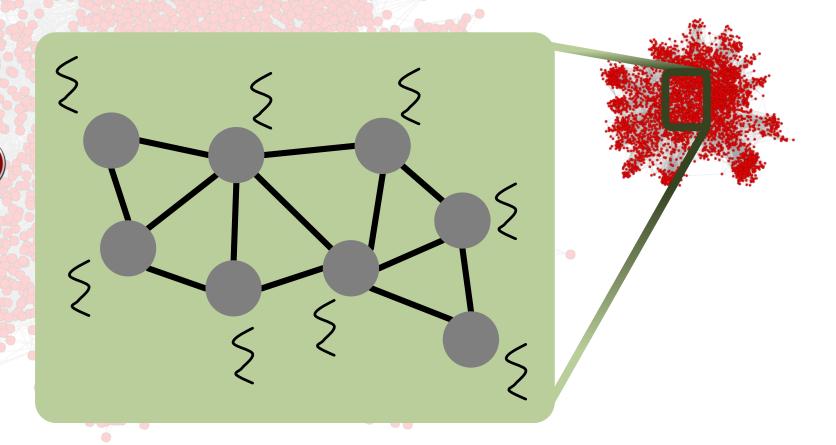


Assumes the whole input graph is accessible...

...when in BRAM, size is severely limited

Vertex-centric, Gather-Apply-Scatter, ... ? Well...

What programming paradigm and why?







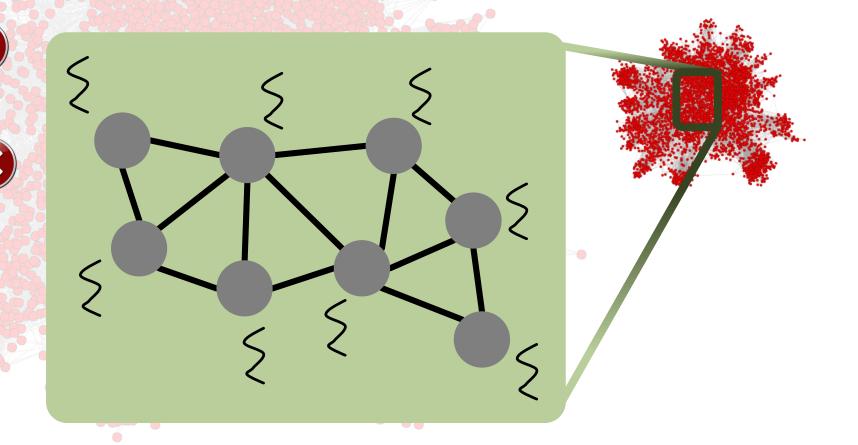


...when in DRAM, accessing & pipelining become complex

What programming paradigm and why?

Assumes the whole input graph is accessible...

...when in BRAM, size is severely limited









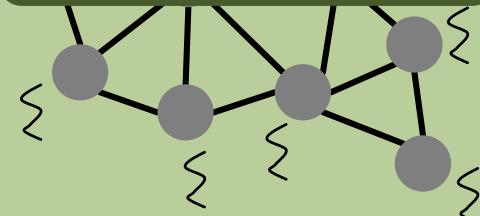
...when in DRAM, accessing & pipelining become complex

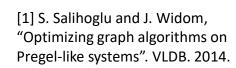
What programming paradigm and why?

Assumes the whole input graph is accessible...

...when in BRAM, size is severely limited

Vertex-centric, Gather-Apply-Scatter, ... ? "(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]











...when in DRAM, accessing & pipelining become complex

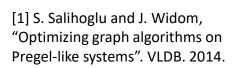
What programming paradigm and why?

Assumes the whole input graph is accessible...

...when in BRAM, size is severely limited

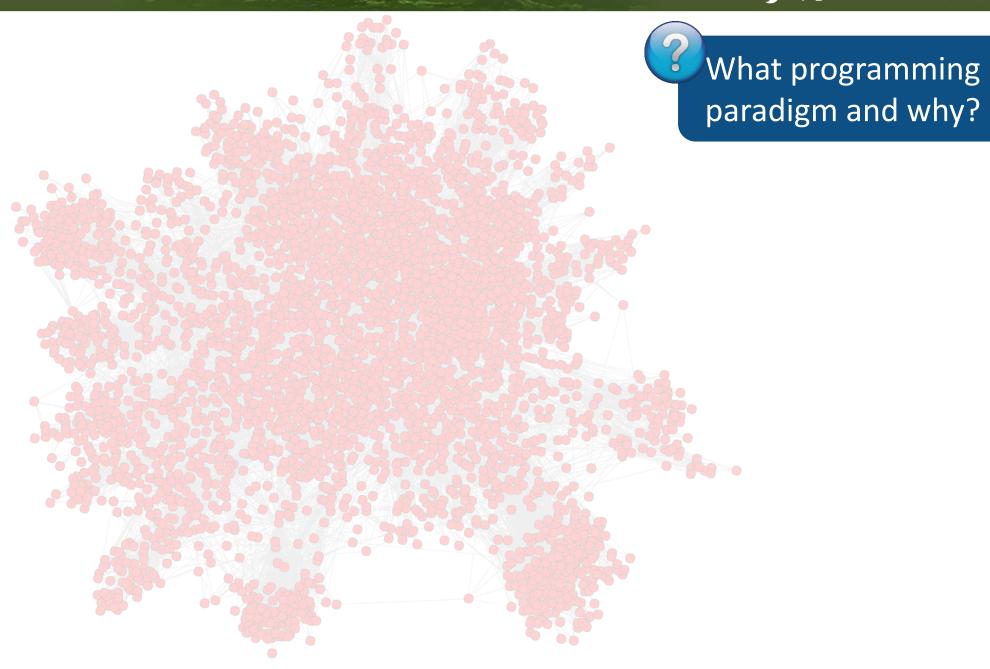
"(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]

Vertex-centric, Gather-Apply-Scatter, ...? Vertex-Centric (aka Pregellike) approach is complex for problems such as matchings, spanning trees, etc.



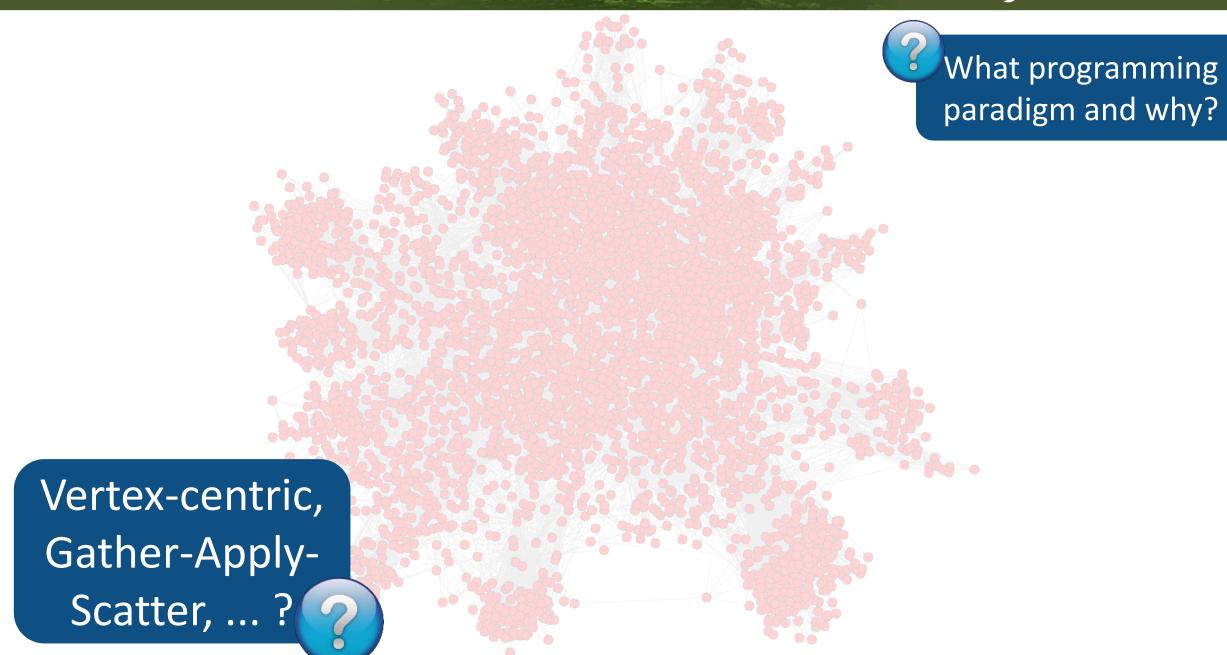




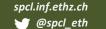




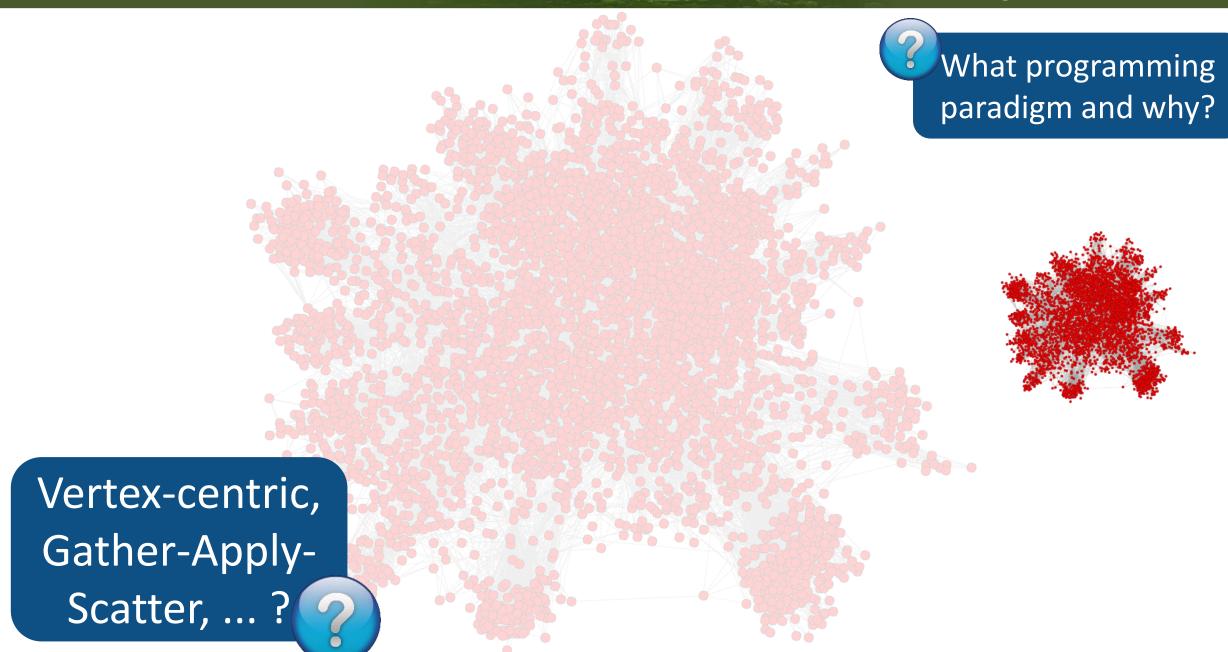








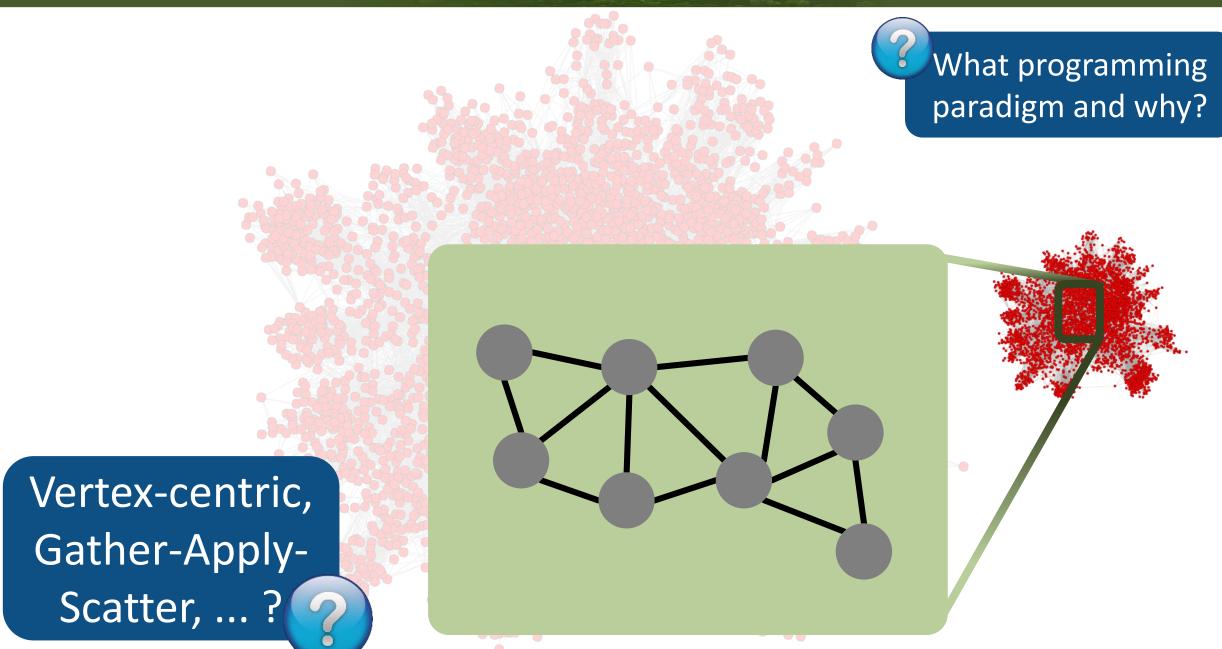










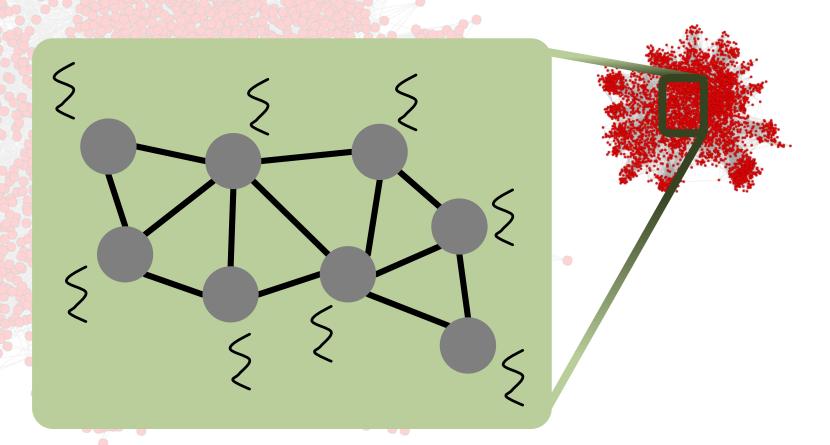










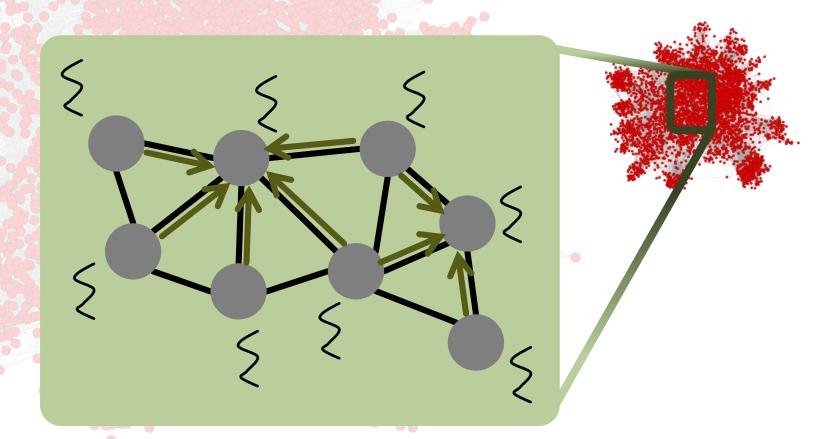










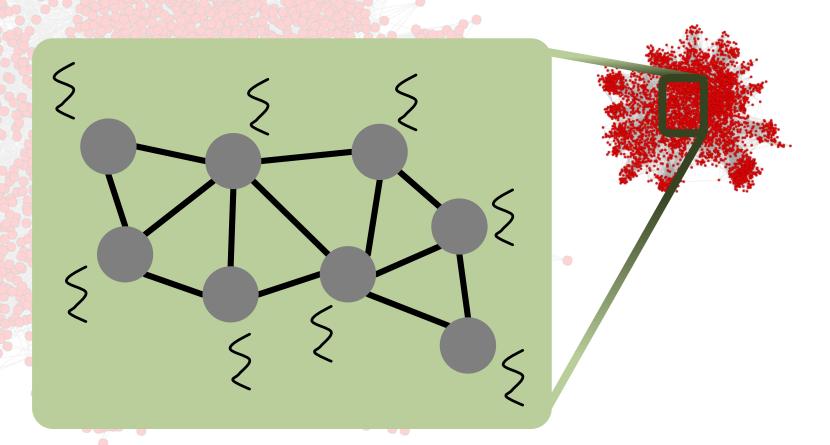










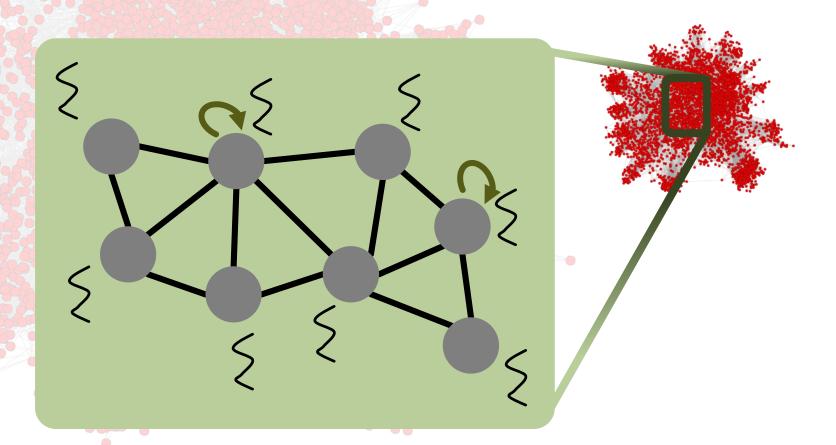










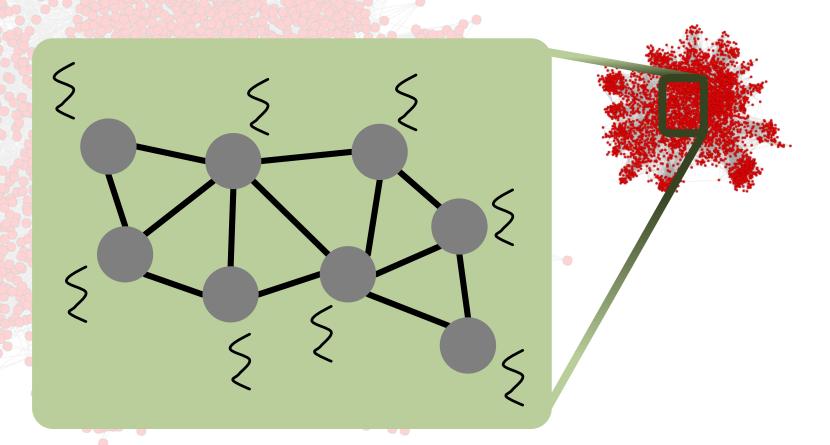










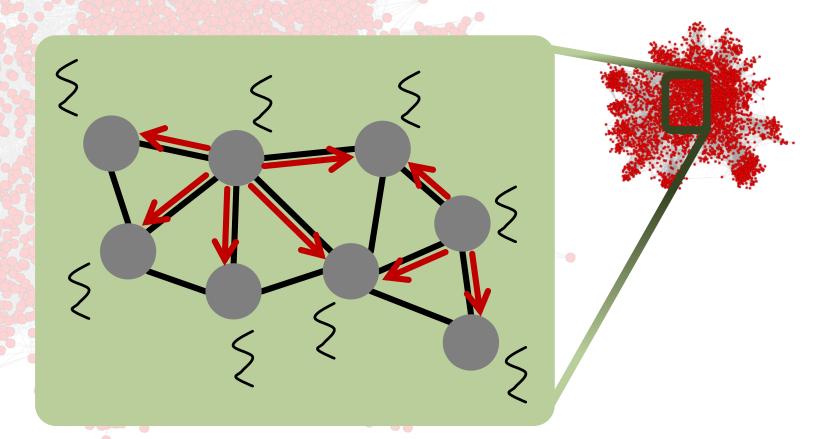










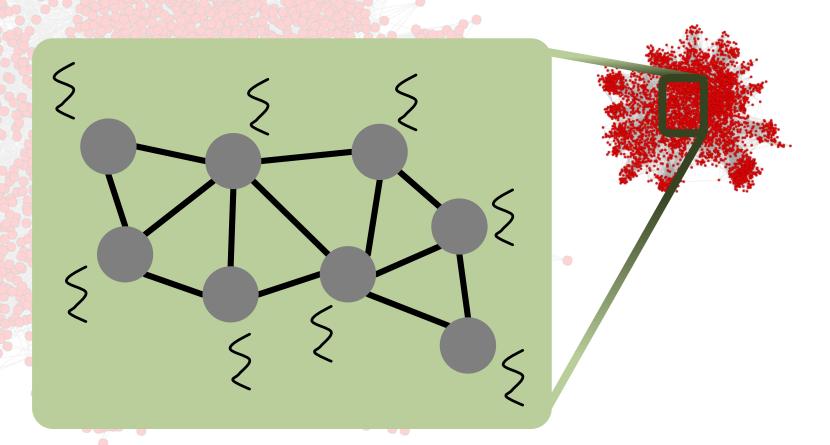












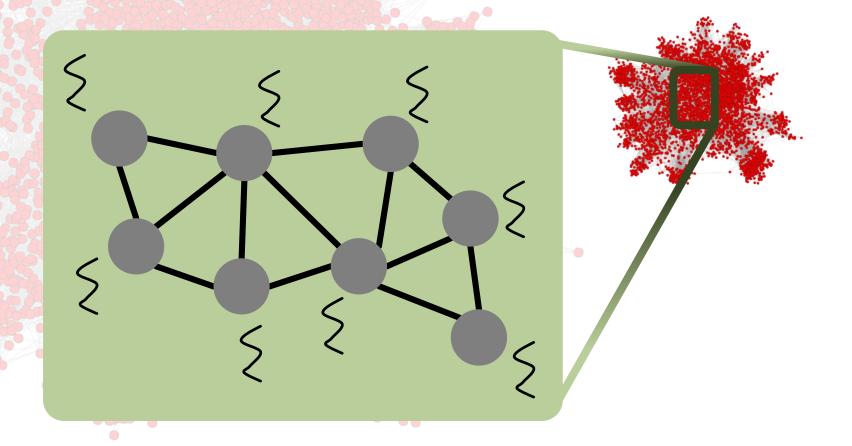






It assumes the whole input is accesible. When in DRAM, accessing & pipelining becomes complex.











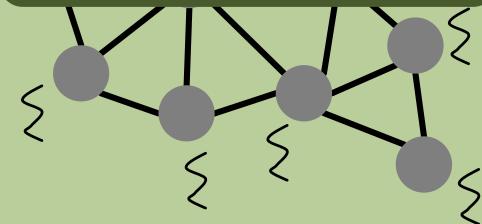


- It assumes the whole input is accesible. When in DRAM, accessing & pipelining becomes complex.

Vertex-centric, Gather-Apply-Scatter, ... ?



"(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]



[1] S. Salihoglu and J. Widom, "Optimizing graph algorithms on Pregel-like systems". VLDB. 2014.









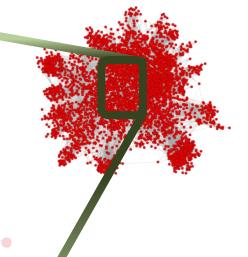
 It assumes the whole input is accesible. When in DRAM, accessing & pipelining becomes complex.

Vertex-centric, Gather-Apply-Scatter, ...?



"(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]

Vertex-Centric (aka Pregellike) approach is complex for problems such as matchings, spanning trees, etc.



[1] S. Salihoglu and J. Widom, "Optimizing graph algorithms on Pregel-like systems". VLDB. 2014.







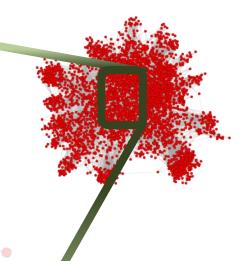
- It was designed with the "batch" analytics in mind.
- It assumes the whole input is accesible. When in DRAM, accessing & pipelining becomes complex.

To be able to <u>utilize pipelining</u>
<u>well</u>, we really want to use
<u>streaming</u> (aka the edge-centric
paradigm)

What programming paradigm and why?

"(...) implementing graph algorithms efficiently on Pregel-like systems (...) can be surprisingly difficult and require careful optimizations." [1]

Vertex-Centric (aka Pregellike) approach is complex for problems such as matchings, spanning trees, etc.



[1] S. Salihoglu and J. Widom, "Optimizing graph algorithms on Pregel-like systems". VLDB. 2014.







Can be used but it was designed with the "batch" analytics in mind

...when in DRAM, accessing & parallelization become complex

What programming paradigm and why?

Assumes the whole input graph is accessible...

...when in BRAM, size is severely limited

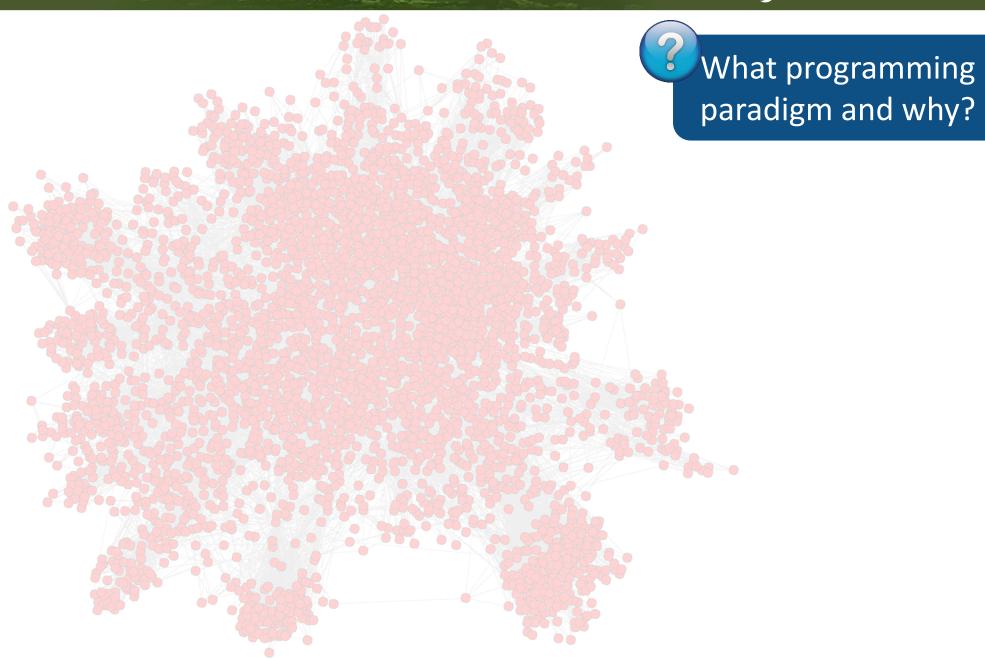
KONECT graph datasets



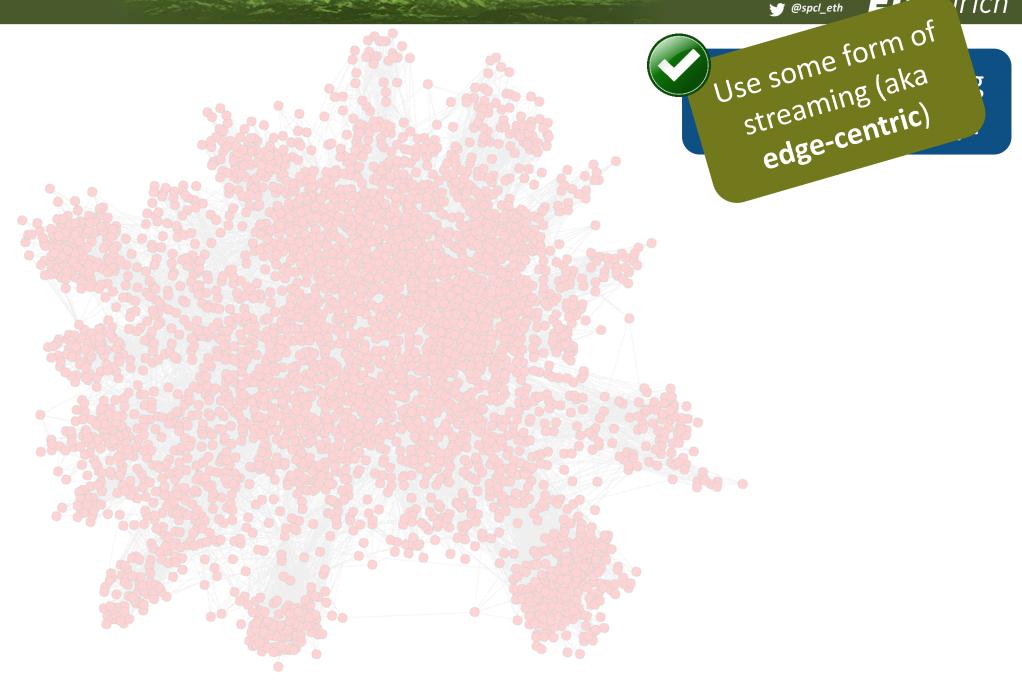
Graph \$	Crawl date \$	Nodes \$	Arcs \$
<u>uk-2014</u>	2014	787801471	47614527250
<u>eu-2015</u>	2015	1070557254	91 792 261 600
gsh-2015	2015	988490691	33877399152
uk-2014-host	2014	4769354	50829923
<u>eu-2015-host</u>	2015	11 264 052	386915963
gsh-2015-host	2015	68 660 142	1 802 747 600
<u>uk-2014-tpd</u>	2014	1766010	18244650
<u>eu-2015-tpd</u>	2015	6650532	170145510
<u>gsh-2015-tpd</u>	2015	30809122	602119716
clueweb12	2012	978408098	42 574 107 469
uk-2002	2002	18520486	298113762



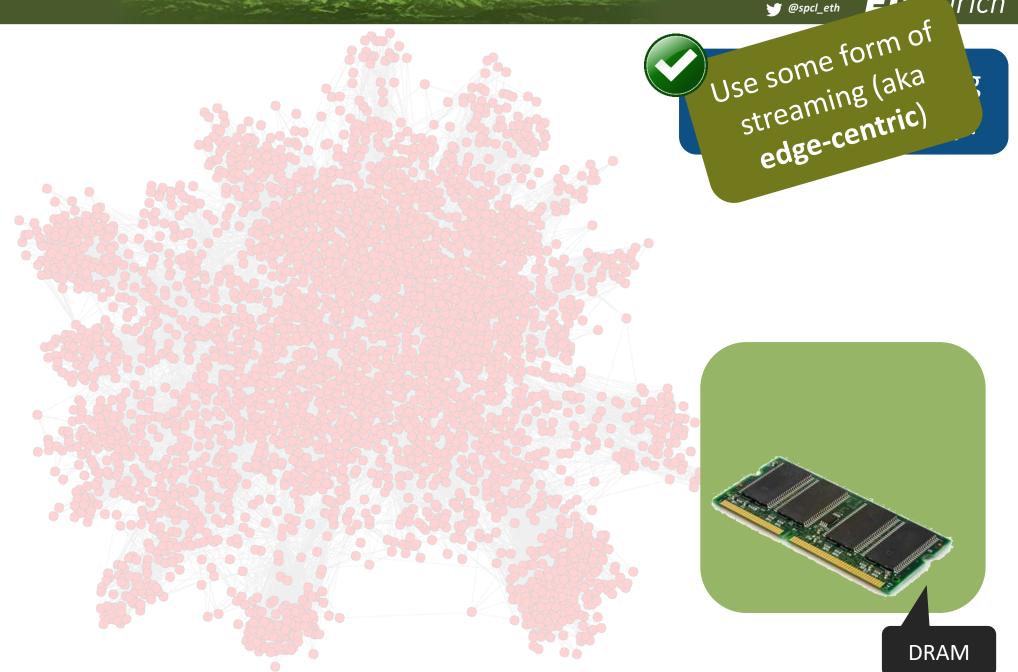








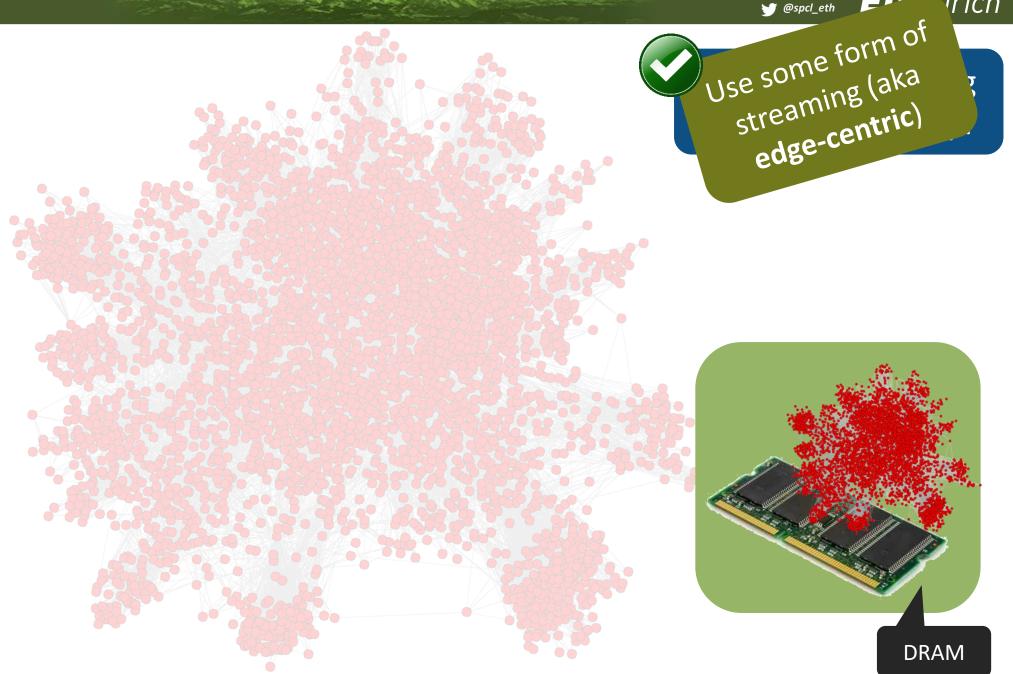
















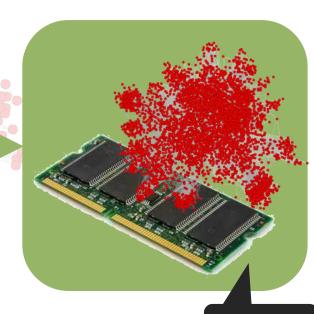




Some processing unit (CPU, GPU, FPGA, ...)







DRAM





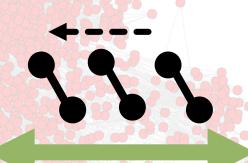


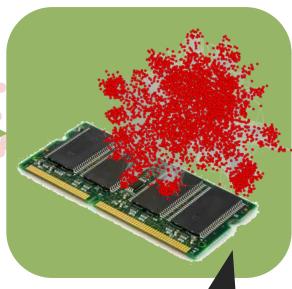


Some processing unit (CPU, GPU, FPGA, ...)









DRAM





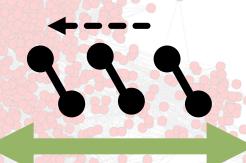


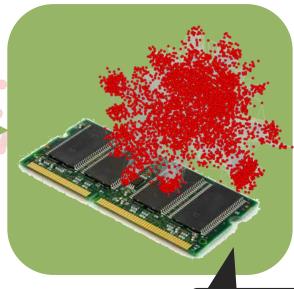


Some processing unit (CPU, GPU, FPGA, ...)

















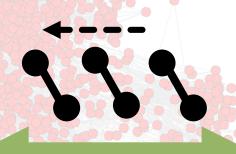


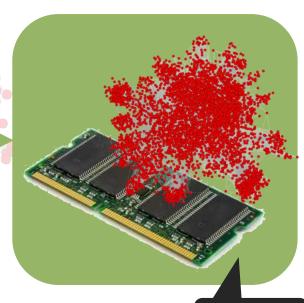
Some processing unit (CPU, GPU, FPGA, ...)

















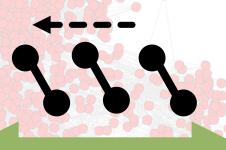


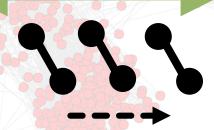
Some processing unit (CPU, GPU, FPGA, ...)

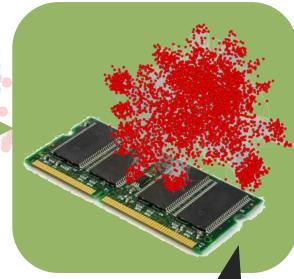




















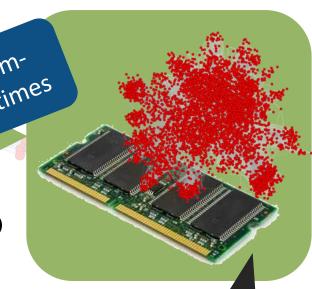
Some processing unit (CPU, GPU, FPGA, ...)







Repeat certain (algorithm-dependent) number of times













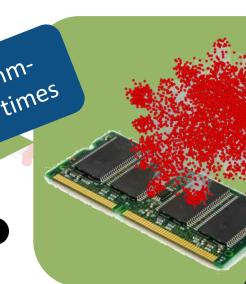
Some processing unit (CPU, GPU, FPGA, ...)







Repeat certain (algorithm-dependent) number of times







irich Use some form of streaming (aka edge-centric)

How to implement efficiently on an FPGA?

Processing edges is sequential – how to incorporate parallelism?

> Some processing unit (CPU, GPU, FPGA, ...)







Repeat certain (algorithm-dependent) number of times







spcl.inf.ethz.ch



How to implement efficiently on an FPGA?

Processing edges is sequential – how to incorporate parallelism? How to minimize the number of "passes" over edges? (This can get **really** bad in the "traditional" edge-centric approach, e.g., BFS normally needs O(m+n) work, while in the edge-centric approach it takes O(D m) work (D passes [1]),

Use some form of streaming (aka edge-centric)

> **m**: #edges in a graph n: #vertices in a graph **D**: graph's diameter (usually ~5-15)

Some processing unit (CPU, GPU, FPGA, ...)





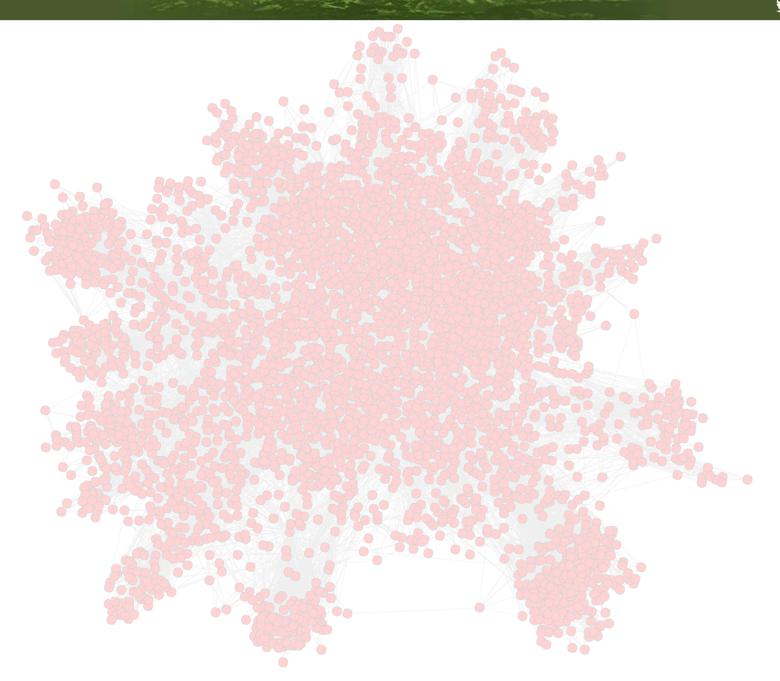


Repeat certain (algorithm-dependent) number of times







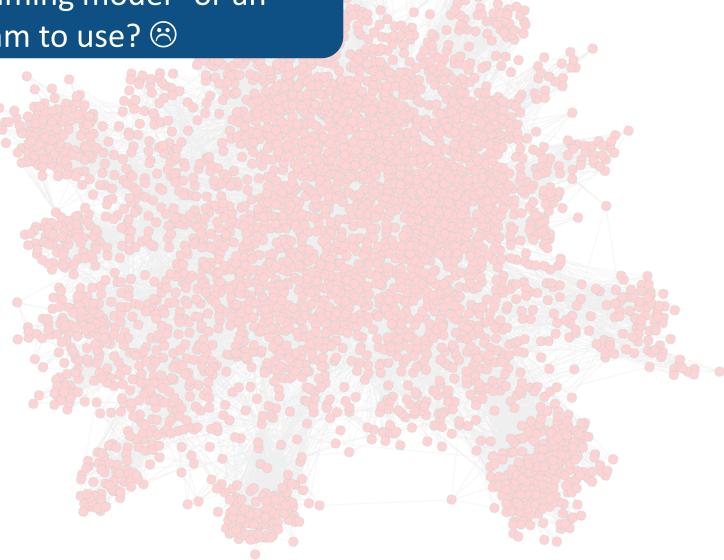








Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use?

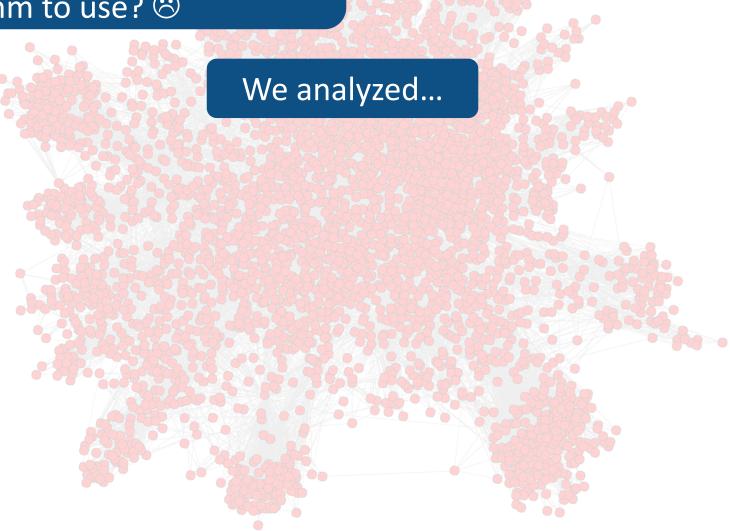








Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? 😊









Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use?

We analyzed...

~15 models for streaming graph processing







Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? 🕾

We analyzed...

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM







Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? 🙁

We analyzed...

Which one to select?

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM







Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use?

We analyzed...

Which one to select?

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Any interesting idea to use in the context of FPGAs and substream-centric processing?







Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use?

We investigated the vast majority of cases, and... guess what happened ©

We analyzed...

Which one to select?

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Any interesting idea to use in the context of FPGAs and substream-centric processing?







Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? 🗵

We investigated the vast majority of cases, and... guess what happened ©

We analyzed...

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Graph processing has become an important part of various areas of computer science, including machine learning, social network analysis, computational sciences, and others. Two key challenges that hinder accelerating graph processing are (1) sizes of input datasets, reaching trillions of edges, and (2) the growing rate of graph updates, with millions of edges added or removed per second. Graph streaming algorithms are specifically crafted to eliminate these issues: The input graph is passed as a stream of updates, allowing to add and remove edges in a simple way. Recent years have seen the development of many such algorithms. However, they differ in the time needed to add or remove an edge the required random access memory space, the number of passes

Which one to select?

~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Any interesting idea to use in the context of FPGAs and substream-centric processing?







Hundreds of papers and schemes, how to select a "streaming model" or an algorithm to use? 😊

We investigated the vast majority of cases, and... guess what happened ©

We analyzed...

Which one to select?

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processin

It is almost ready (may take ~1 month more) – if you want to check it out earlier, let us know! MARC FISCHER, Depart MACIEJ BESTA, TAL BEN-NUN, TORSTEN HOE

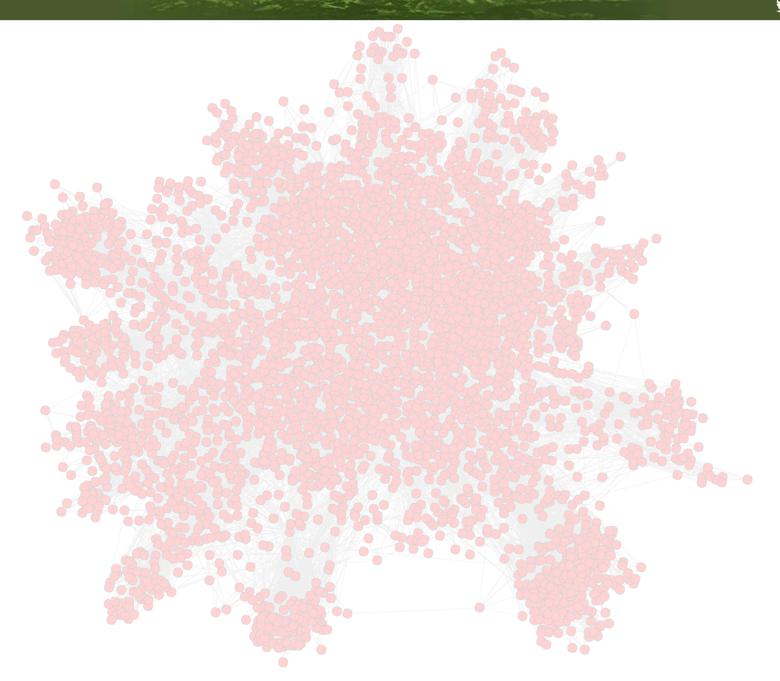
Graph processing has l ing, social network ana graph processing are (1) .a. Graph streaming algorithms are specifically updates, with millions of crafted to eliminate these passed as a stream of updates, allowing to add and remove ve seen the development of many such algorithms. However, they differ edges in a simple way. Rece in the time needed to add or remove an edge the required random access memory space the number of passes ~15 models for streaming graph processing

~30 algorithms for streaming (approximate) MWM

Any interesting idea to use in the context of FPGAs and substream-centric processing?





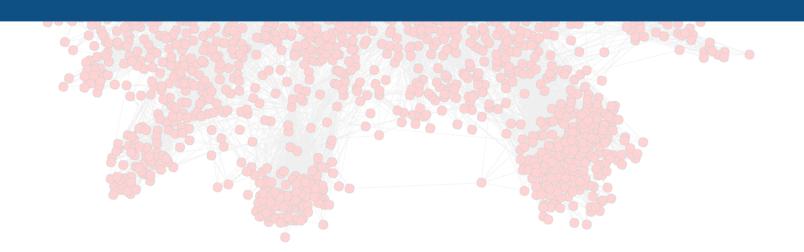








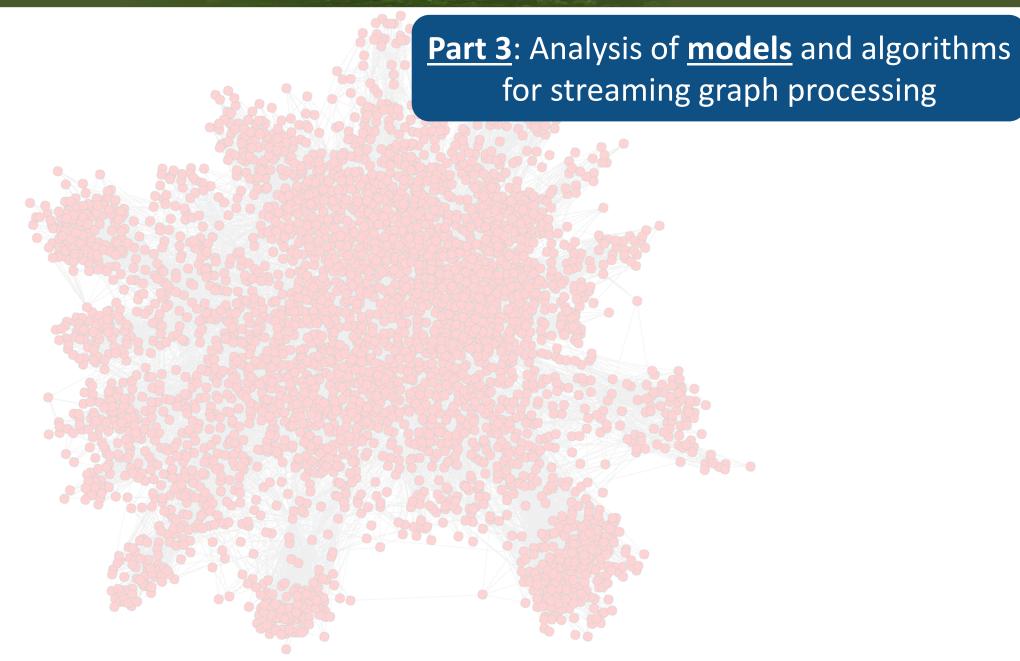








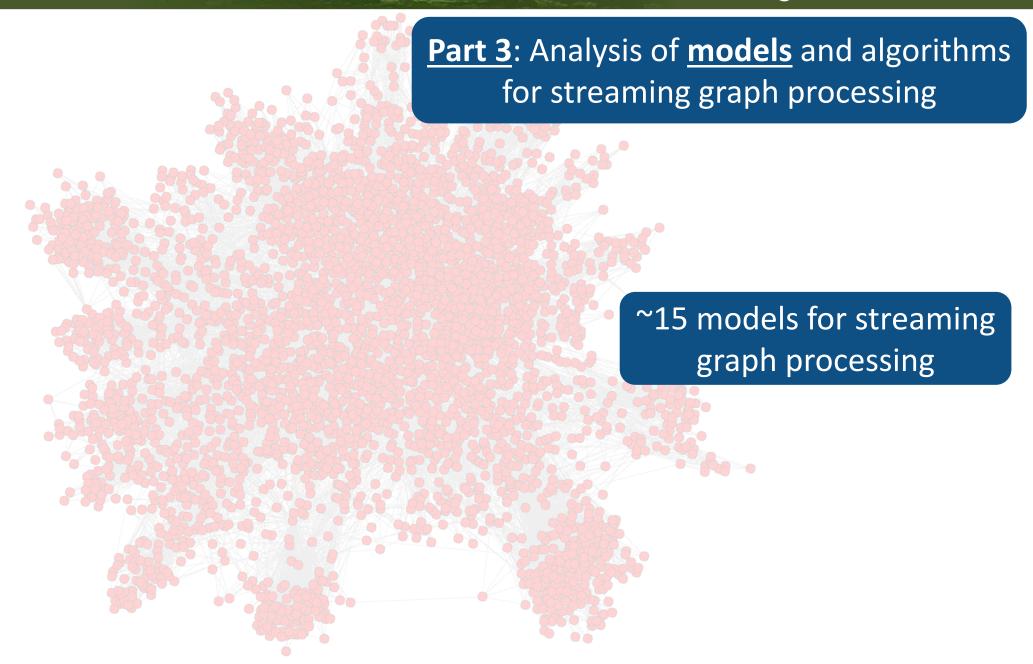








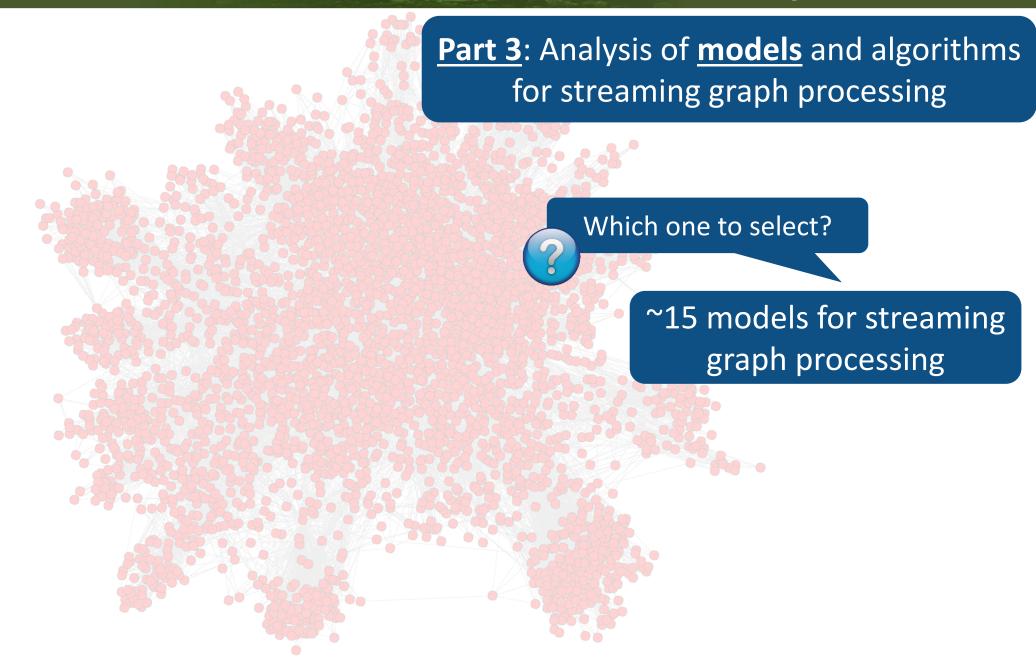








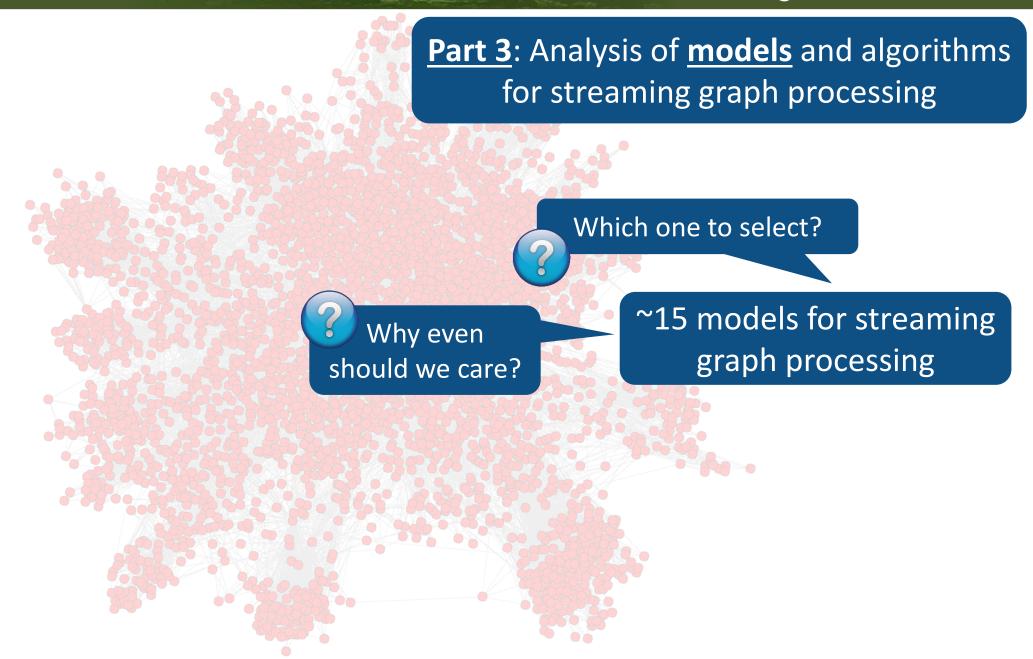


















Which one to select?

Using a model enables rigorous analysis of the algorithm behavior

Why even should we care?

~15 models for streaming graph processing







Which one to select?

Using a model enables rigorous analysis of the algorithm behavior

Why even should we care?

~15 models for streaming

graph processing

Selecting a right model enables more realistic predictions about the algorithm behavior on given hardware







Which one to select?

Using a model enables rigorous analysis of the algorithm behavior

Why even should we care?

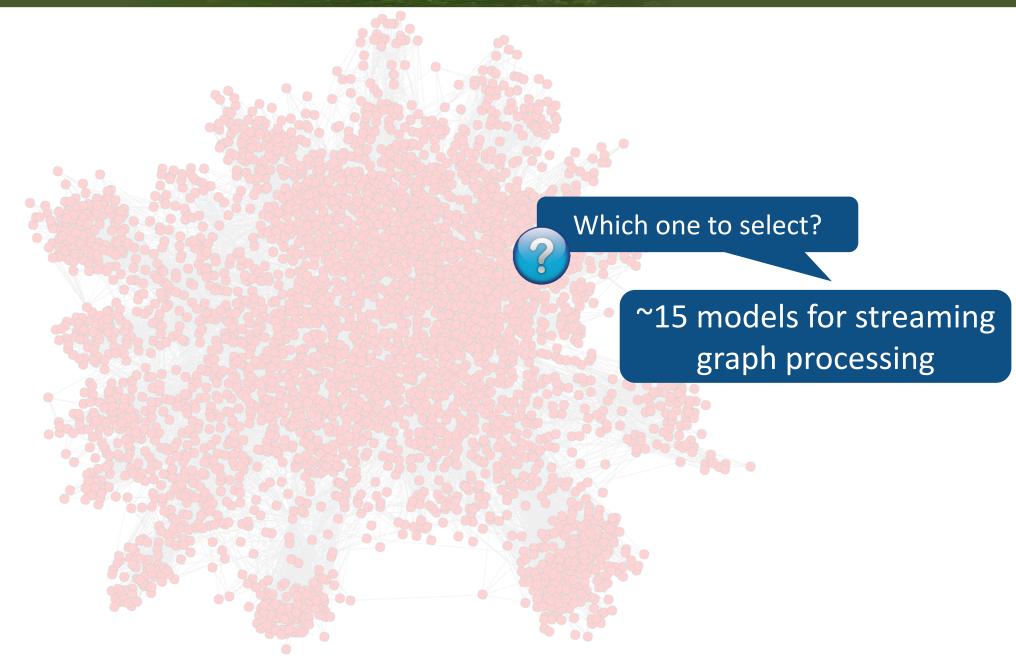
~15 models for streaming graph processing

Limiting oneself to a particular model helps to select the best algorithm or technique (within that model)

Selecting a right model enables more realistic predictions about the algorithm behavior on given hardware



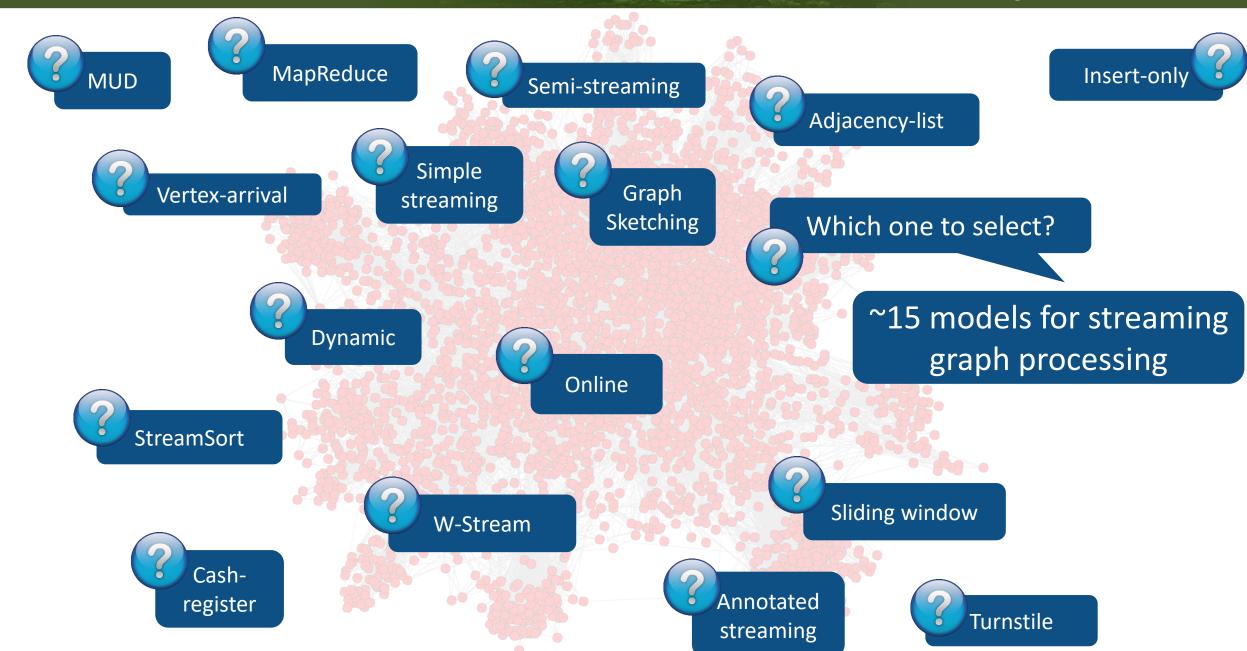








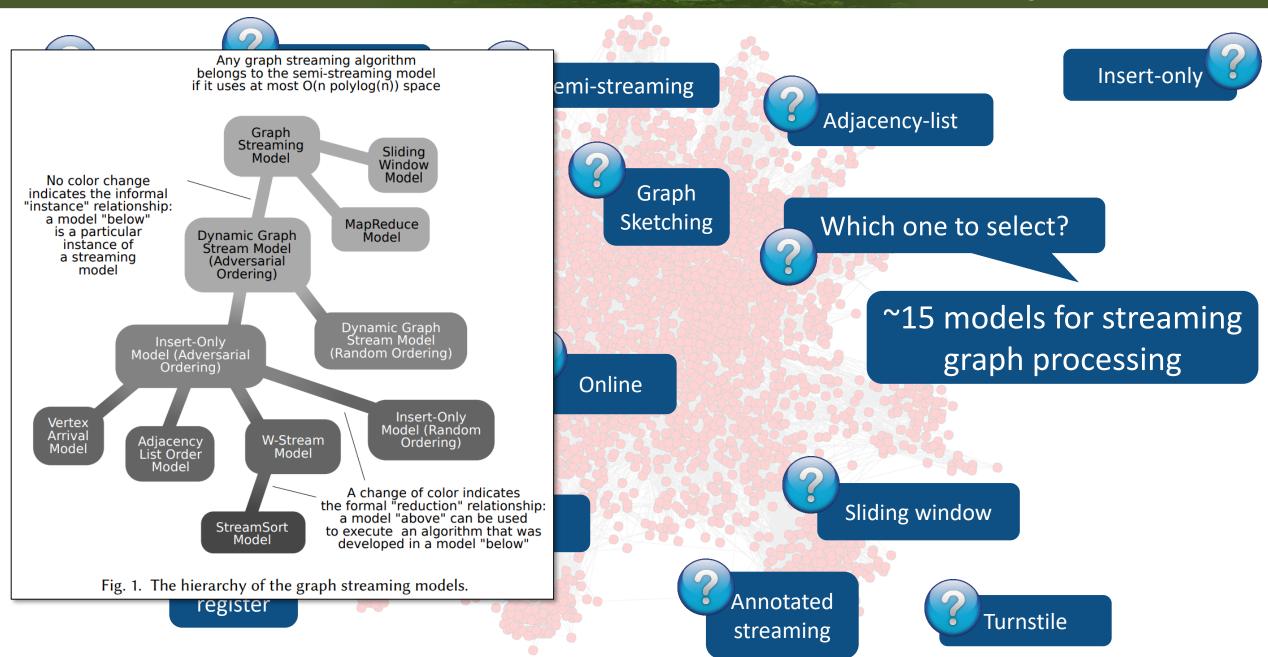


















Insert-only



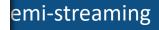
Any graph streaming algorithm belongs to the semi-streaming model if it uses at most O(n polylog(n)) space

To understand the models (and related caveats) well, we related caveats) well, we developed a formal taxonomy of the analyzed models with the aim of guiding future graph aim of guiding future graph (check the streaming designs (check the report for details ©)

StreamSort Model A change of color indicates the formal "reduction" relationship: a model "above" can be used to execute an algorithm that was developed in a model "below"

Fig. 1. The hierarchy of the graph streaming models.

register



Graph

Sketching



Adjacency-list



Which one to select?

Online

~15 models for streaming graph processing



Sliding window

Annotated streaming

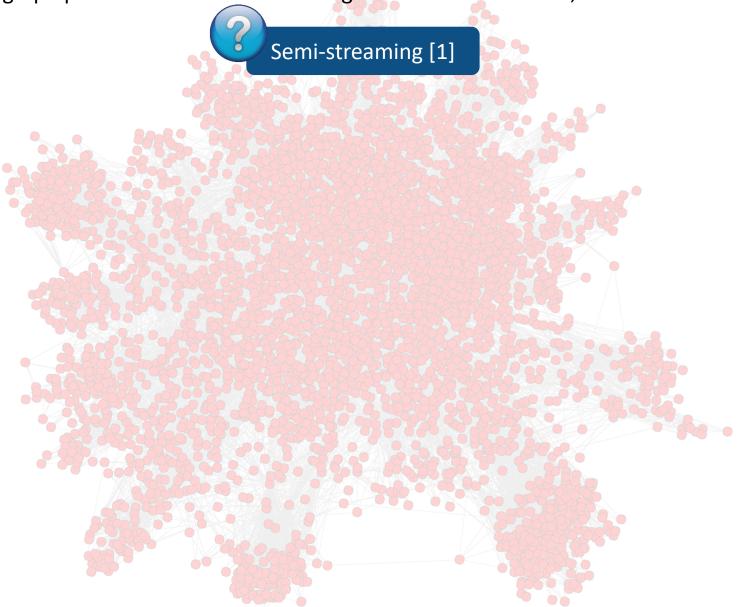








[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

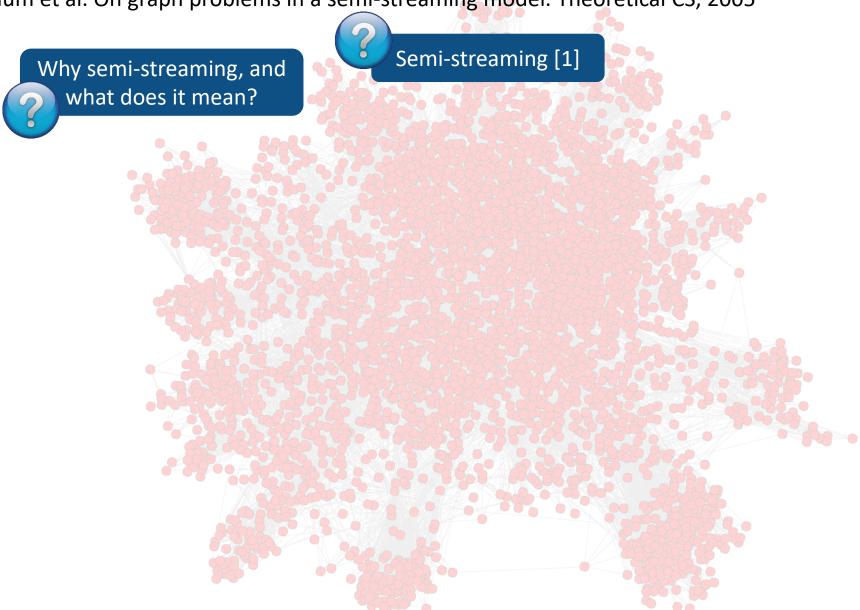








[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005









[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

Covers a general streaming

Semi-streaming [1]

Semi-streaming [1] Why semi-streaming, and what does it mean?

vers a general streamir setting (= works for substream-centric)

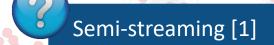




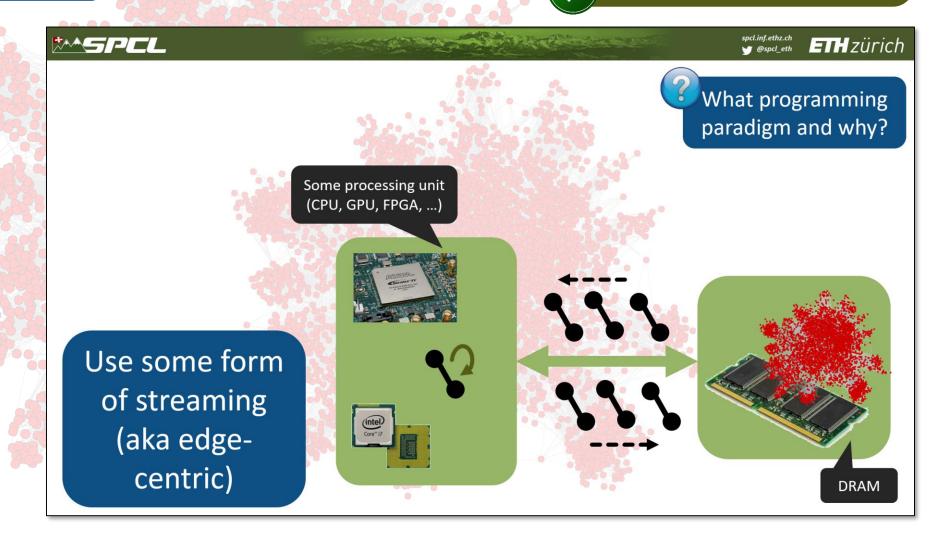


[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

Why semi-streaming, and what does it mean?



Covers a general streaming setting (= works for substream-centric)

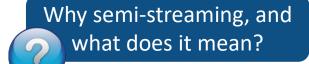






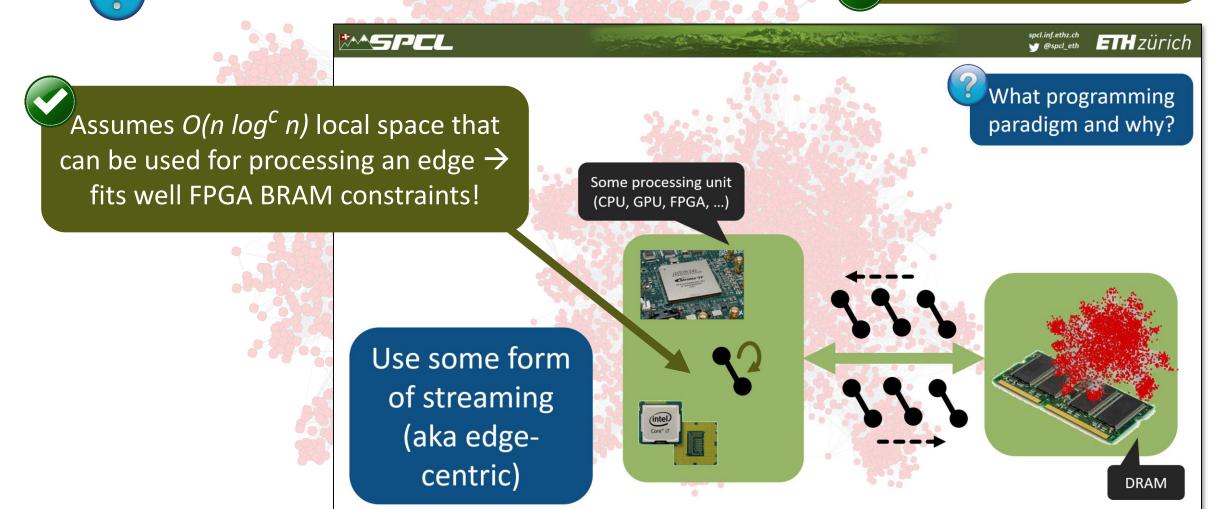


[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005



Semi-streaming [1]

Covers a general streaming setting (= works for substream-centric)









ETH zürich

[1] J. Feigenbaum et al. On graph problems in a semi-streaming model. Theoretical CS, 2005

Why semi-streaming, and what does it mean?



Covers a general streaming setting (= works for substream-centric)

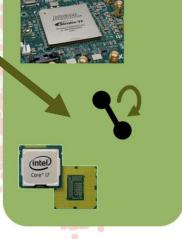


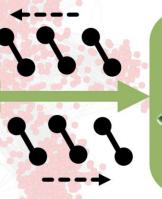
Assumes $O(n \log^c n)$ local space that can be used for processing an edge \rightarrow fits well FPGA BRAM constraints!

Some processing unit (CPU, GPU, FPGA, ...)

Offers (potentially powerful) MWM algorithms

Use some form of streaming (aka edge-centric)







What programming

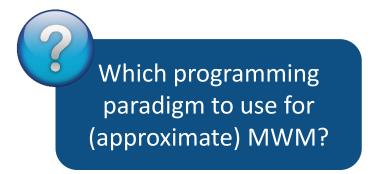
paradigm and why?







How to design a highperformance MWM algorithm (as dictated by the used paradigm)?



What is the HW FPGA design that ensures high performance?









How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Use substream-centric

processing (exposes

processing (exposes

parallelism)

parallelism use for

(approximate) MWM?

What is the HW FPGA design that ensures high performance?









How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Use substream-centric

processing (exposes

parallelism)

parallelism

theoretical analysis

and rigor in the context of

and rigor in the context of

the semi-streaming model

the semi-streaming model

What is the HW FPGA design that ensures high performance?









How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Use <u>substream-centric</u>

processing (exposes

parallelism)

parallelism)

theoretical analysis

and rigor in the context of

and rigor in the context of

the semi-streaming model

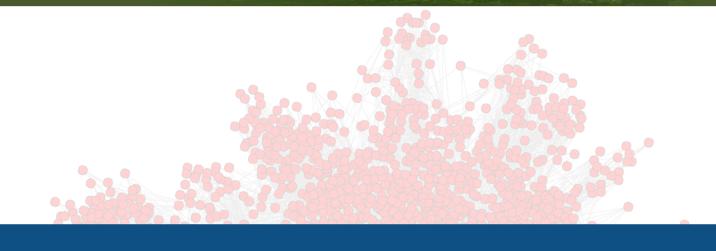
the semi-streaming model

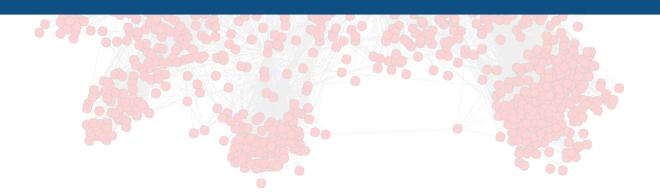
What is the HW FPGA design that ensures high performance?







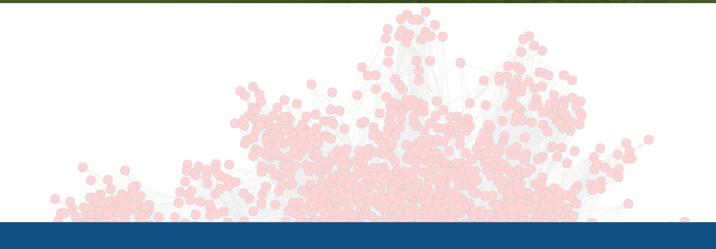


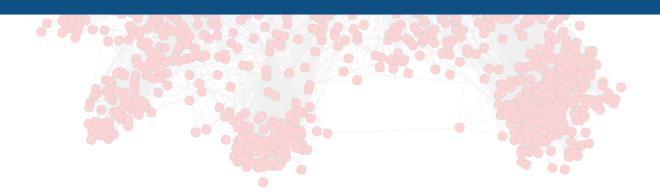








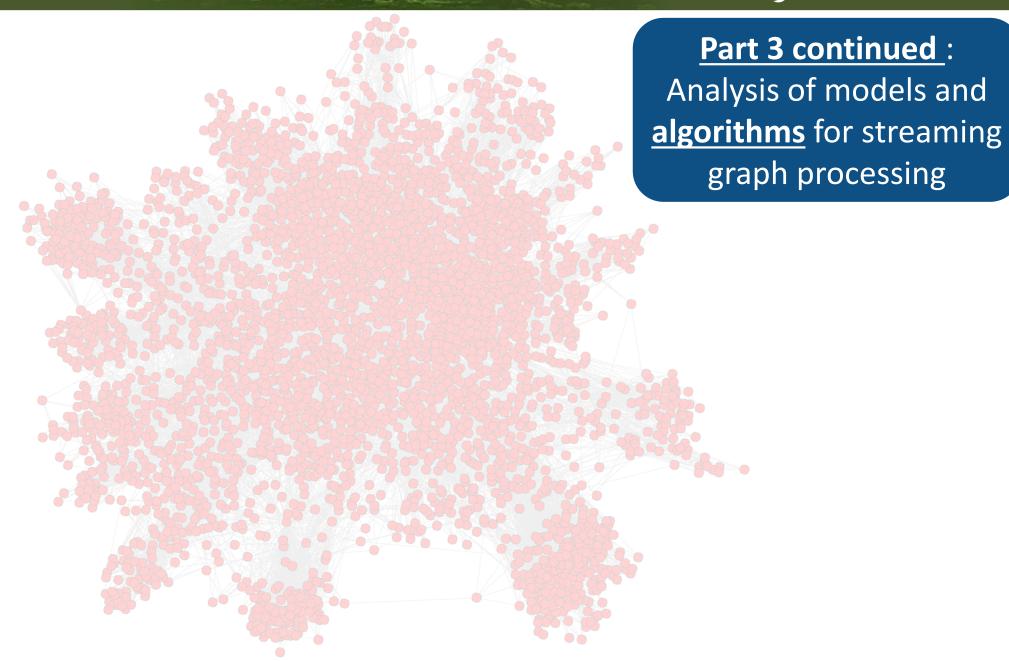








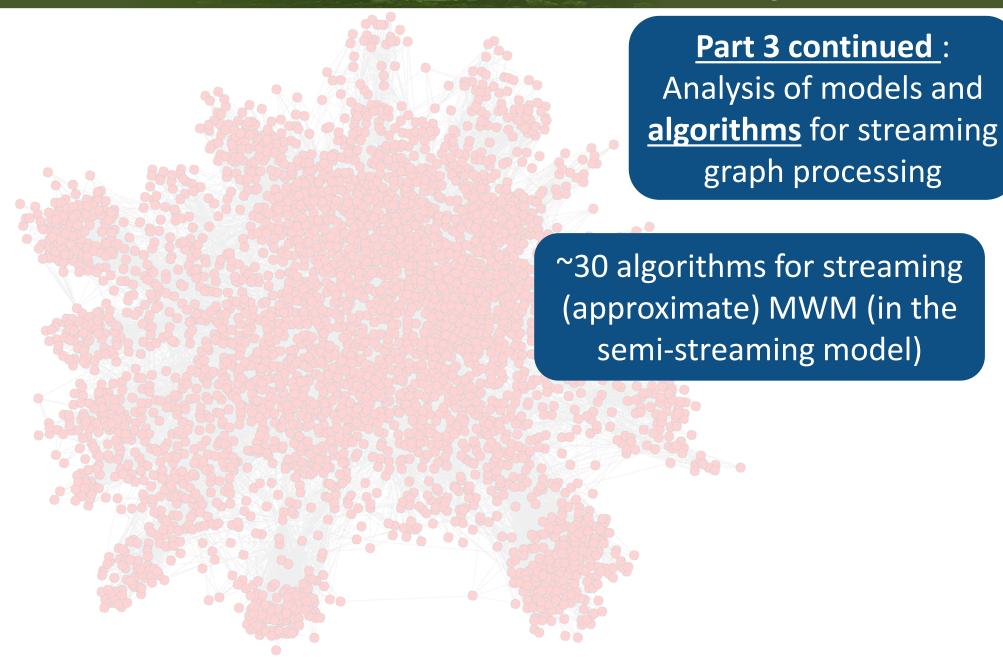








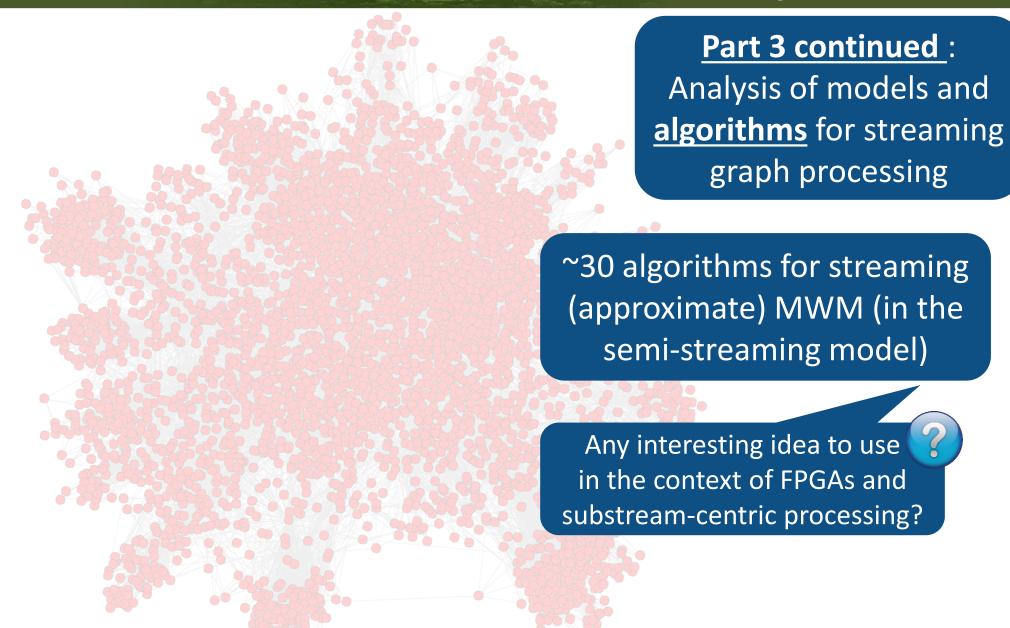




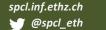




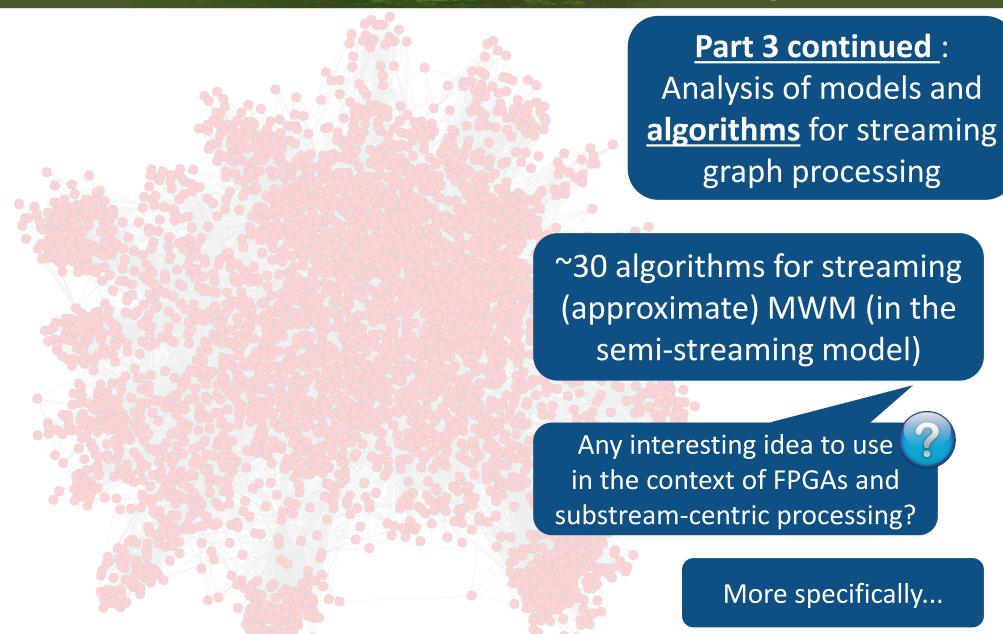












Reference	Approx.	Space	#Passes	Wgh^1	Gen ²	Par ³
[26] [41, Theorem 6] [41, Theorem 2]	$1/2$ $1/2 + 0.0071$ $1/2 + 0.003^*$	O(n) $O(n polylog(n))$ $O(n polylog(n))$	1 2 1	₩ ₩ ₩	<u></u>	: & : & : &
[36, Theorem 1.1] [26, Theorem 1]	O(polylog(n))	$O(\operatorname{polylog}(n))$ $O(n\log n)$	$ \begin{array}{c} 1 \\ O\left(\log\left(1/\varepsilon\right)/\varepsilon\right) \end{array} $	•	₫ III	∆ .•
[6, Theorem 19]	$1-\varepsilon$	$O\left(n \operatorname{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	•	•	•
[41, Theorem 5] [41, Theorem 1]	$1/2 + 0.005^*$	$O(n \operatorname{polylog}(n))$ $O(n \log n)$	1	•	* F	: (
[41, Theorem 4] [39] [28, Theorem 20]	$1/2 + 0.0071^*$ 1 - 1/e	O(n polylog(n)) O(n polylog(n)) O(n)	2 1 1	•	•	: 6
[35, Theorem 2]	$1 - \frac{1}{e}$ $1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	O(n)	k	•	•	:- :•
[14]	(<i>k</i> -1)!	$\tilde{O}(k^2)$	1	•	Ô	Ô
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	•	Ô	Ô
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	•	•	Ů
[26, Theorem 2] [44, Theorem 3] [44, Theorem 3]	$6 \\ 2 + \varepsilon \\ 5.82$	$O(n \log n)$ $O(n \operatorname{polylog}(n))$ $O(n \operatorname{polylog}(n))$	1 O(1) 1	0	<u> </u>	?
[63] [25] [29]	5.58 $4.911 + \varepsilon$ $3.5 + \varepsilon$	O(n polylog(n)) O(n polylog(n)) O(n polylog(n))	1 1 1	0 0 0	ථ ඨ ඨ	?
[53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1	Ô	Ô	0
[27]	$2+\varepsilon$	$O(n \log n)$	1	Ô	Ô	8
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	Ô	Ô	8
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$	Ô	Ů	:
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ů	Ů	:•
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ů	•	•
[17]	$4+\varepsilon$	$O(n \operatorname{polylog}(n))$	1	Ů	Ô	Ů

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?



Reference	Approx.	Space	#Passes	\mathbf{Wgh}^1	Gen ²	Par ³
[26]	1/2	O(n)	1	1		
[41, Theorem 6]	1/2 + 0.0071 $1/2 + 0.003^*$	O(n polylog(n)) O(n polylog(n))	2	16		
[41, Theorem 2] [36, Theorem 1.1]			1	100	3	3
[26, Theorem 1]		$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	H.	R. C.	100
[6, Theorem 19]	$1-\varepsilon$	$O\left(n \operatorname{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	R. C.	R. C.	1
[41, Theorem 5]	1/2 + 0.019	O(n polylog(n))	2	R. C.	R. C.	1
[41, Theorem 1]	$1/2 + 0.005^*$	$O(n \log n)$	1	N. C.	R. C.	100
[41, Theorem 4]	1/2 + 0.0071	O(n polylog(n))	2	100	16	
[39] [28, Theorem 20]	1 - 1/e 1 - 1/e	O(n polylog(n)) O(n)	1	16	100	
[35, Theorem 2]	$1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	O(n)	k	H.	N. P.	10
[14]	1	$\tilde{O}\left(k^2\right)$	1	nde.	O	ß
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	n q	ß	ß
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	191	rip.	0
[26, Theorem 2]	6	$O(n \log n)$	1	6	0	
[44, Theorem 3]	$2+\varepsilon$	O(n polylog(n))	O(1)			
[44, Theorem 3] [63]	5.82 5.58	O(n polylog(n)) O(n polylog(n))	1			
[25]	$4.911 + \varepsilon$	O(n polylog(n))	1	3	3	
[29]	$3.5 + \varepsilon$	O(n polylog(n))	1	6	6	
[53]	$2 + \varepsilon$	$O\left(n\log^2 n\right)$	1	6	6	
[27]	$2 + \varepsilon$	$O(n \log n)$	1	B	6	
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	ß	ß	
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$			n dr
[6, Theorem 22]	$\frac{1}{\frac{2}{2}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ď		ı
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ů	n il	16
[17]	$4 + \varepsilon$	O(n polylog(n))	1	ß	ß	ß

<u>Part 3 continued</u>: Analysis of models and

algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?





Reference	Approx.	Space	#Passes	\mathbf{Wgh}^1	Gen ²	Par
Para	1/2	O(n)	1	1	0	B.
Our	1/2 + 0.0071 $1/2 + 0.003^*$	O(n polylog(n)) O(n polylog(n))	2	16		: ()
goals: 1	O(polylog(n))		1	16	3	3
guais.	$2/3-\varepsilon$	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	R. C.	R. C.	1
6, Theorem 19]	$1-\varepsilon$	$O\left(n \operatorname{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	N/A	REP.	
41, Theorem 5]	1/2 + 0.019	$O(n \operatorname{polylog}(n))$	2	RIP.		
41, Theorem 1]	$1/2 + 0.005^*$	$O(n \log n)$	1	N. C.	Rep.	8
41, Theorem 4]	1/2 + 0.0071	O(n polylog(n))	2	100	ide ide	
[39] [28, Theorem 20]	1 - 1/e 1 - 1/e	O(n polylog(n)) O(n)	1	10	100	
35, Theorem 2]	$1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	O(n)	k	19	1 P	1
14]	1	$\tilde{O}\left(k^2\right)$	1	nde.	ß	0
14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	nde.	ß	ß
7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	19	rip.	ß
[26, Theorem 2]	6	$O(n \log n)$	1	3		
44, Theorem 3]	$2+\varepsilon$	O(n polylog(n))	O(1)			
[44, Theorem 3] [63]	5.82 5.58	O(n polylog(n))	1			
25]	$4.911 + \varepsilon$	O(n polylog(n)) O(n polylog(n))	1	3	3	
29]	$3.5 + \varepsilon$	O(n polylog(n))	1	3	3	
53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1	6	3	
[27]	$2+\varepsilon$	$O(n\log n)$	1	3	6	
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	6	0	
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$	O	Ď	
6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	ß	Ů	1
6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	ß	rift.	16
17]	$4+\varepsilon$	$O(n \operatorname{polylog}(n))$	1	ß	ß	ß

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

Reference	Approx.	Space	#Passes	\mathbf{Wgh}^1	Gen ²	Par
	1 (2	O(n)	1	1		100
Our	$12 + 0.0071 + 0.003^*$	O(n polylog(n)) O(n polylog(n))	2	ode ode		
goals:			1	16	3	3
godis.	ε	$O(n \log n)$	$O(\log(1/\varepsilon)/\varepsilon)$	H.	Rep.	
6. Theorem 19]		$O\left(n \operatorname{polylog}(n)/\varepsilon^2\right)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	H.	H.	
4 Maxir	nize	O(n polylog(n))	2	R. P.	R. C.	
4	5	$O(n \log n)$	1	1		
accur	acy	O(n polylog(n)) O(n polylog(n))	1	10		
[28, Theorem 20]		O(n)	1	H.	RIP.	
[35, Theorem 2]	$1 - \frac{e^{-k}k^{k-1}}{(k-1)!}$	O(n)	k	19	rife.	
[14]	1	$\tilde{O}\left(k^2\right)$	1	i q	ß	ß
[14]	$1/\varepsilon$	$\tilde{O}\left(n^2/\varepsilon^3\right)$	1	i p	ß	O
7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	r q	r ip	O
[26, Theorem 2]	6	$O(n \log n)$	1	0	0	
44, Theorem 3]	$2+\varepsilon$	O(n polylog(n))	O(1)			
[44, Theorem 3] [63]	5.82 5.58	O(n polylog(n)) O(n polylog(n))	1		6	
[25]	$4.911 + \varepsilon$	O(n polylog(n))	1	3	3	
[29]	$3.5 + \varepsilon$	O(n polylog(n))	1		6	
[53]	$2 + \varepsilon$	$O\left(n\log^2 n\right)$	1		0	
[27]	$2 + \varepsilon$	$O(n \log n)$	1		O	
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$		ß	
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$	Ô	O	
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	O	Ď	
6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	ß	rife.	16
[17]	$4 + \varepsilon$	O(n polylog(n))	1	Ď	ß	3

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?







Reference	Approx.	Space	#Passes	\mathbf{Wgh}^1	Gen ²	Par ³
Faci	1 (2	O(n)	1	nde.	Ď	1
Our	$12 + 0.0071 + 0.003^*$	(2	ide ide		:4
goals:	$\begin{array}{ccc} & + 0.003 \\ & & \text{lylog}(n) \end{array}$	- / / 0 (- / /	1	16	3	3
	ε .	$C = l \log n$	$O\left(\log\left(1/\varepsilon\right)/\varepsilon\right)$	H.	H.	1
[6, Theorem 19]		$(n \text{ polylog}(n)/\varepsilon^2)$	$O\left(\log\log\left(1/\varepsilon\right)/\varepsilon^2\right)$	S. Carrier	B. C.	
[4 Maxir	mize	$n \operatorname{polylog}(n)$	2	H.	B. C.	100
4 accui	racy	$n \log n$	1	1		
	acy	n polylog(n) n polylog(n)	1	100	100	
[28, Theorem 20]	1 - 1/e	n)	1	RAP.	R CO	
[35, Theorem 2]	Minim	iize	k	1 P	rife.	10
[14]	local sp	oace)	1	nder.		ß
[14]	$1/\varepsilon$	$O\left(n^2/\varepsilon^3\right)$	1	H.	0	O
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	1	N. C.	rip.	ß
[26, Theorem 2]	6	$O(n \log n)$	1		0	
[44, Theorem 3]	$2+\varepsilon$	O(n polylog(n))	O(1)			
[44, Theorem 3]	5.82 5.58	O(n polylog(n)) O(n polylog(n))	1		6	
[25]	$4.911 + \varepsilon$	O(n polylog(n))	1			
[29]	$3.5 + \varepsilon$	O(n polylog(n))	1		ß	
[53]	$2+\varepsilon$	$O\left(n\log^2 n\right)$	1		ß	
[27]	$2 + \varepsilon$	$O(n \log n)$	1	ß	ß	
[26, Section 3.2]	$2 + \varepsilon$	$O(n\log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$		6	
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$	Ô		
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	3	ß	100
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n - \log\varepsilon}{\varepsilon^2}\right)\right)$	$O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ô	n de	1
[17]	$4 + \varepsilon$	O(n polylog(n))	1	3	ß	ß

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?







Reference	Approx.	Space	#Passes	Wgh^1	Gen ²	² Par ³
[ac]	1 (2	O(n)		in.	ß	100
Our	12 + 0.0071		nimize	H.		84
goals: 1	+0.003 lylog(n)	O pol olyle #pa	asses			
goals:	\mathcal{E}	$C : \log n$	$O(\log(1/\varepsilon)/\varepsilon)$	n qu		100
[6, Theorem 19]		$n \operatorname{polylog}(n)$	$1/\varepsilon^2$ O $(\log \log (1/\varepsilon)/\varepsilon^2)$		R. P.	10
Maxir	mize	$n \operatorname{polylog}(n)$) 2	100	N. C.	8
[4 accui	racy 71	$n \log n$	1	10	10	: (a
[3>]	ucy	n polylog(n) n polylog(n)		100	100	
[28, Theorem 20]		11)	1	nde.	R. P.	84
[35, Theorem 2]	Minim	ize	k	s de	s.	10
[14]	local sp	ace	1	nde.		ß
[14]	$1/\varepsilon$	$O\left(n^2/\varepsilon^3\right)$	1	RIP.		Ô
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{1-\varepsilon}\right)$	$-\varepsilon$) 1	H.	RIP.	6
[26, Theorem 2]	6	$O(n \log n)$	1	ß	6	
[44, Theorem 3]	$2+\varepsilon$	O(n polylog(n))				
[44, Theorem 3]	5.82 5.58	O(n polylog(n)) O(n polylog(n))			6	
[25]	$4.911 + \varepsilon$	O(n polylog(n)) 1		3	
[29]	$3.5 + \varepsilon$	O(n polylog(n))) 1	Ď		
[53]	$2 + \varepsilon$	$O\left(n\log^2 n\right)$	1	Ô		
[27]	$2 + \varepsilon$	$O(n \log n)$	1			
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$			
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)/\varepsilon^4\right)$	$O\left(\varepsilon^{-4}\log n\right)$			100
[6, Theorem 22]	$\frac{1}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n - \log n}{\varepsilon^2}\right)\right)$	$\left(\frac{\log \varepsilon}{\varepsilon}\right)$ $O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	ß	O	16
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n - \log n}{\varepsilon^2}\right)\right)$	$\left(\frac{\log \varepsilon}{\varepsilon}\right) O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ô	1.	16
[17]	$4 + \varepsilon$	O(n polylog(n)) 1	ß	ß	ß

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

Reference	Approx.	Space	#Passes	Wgh	¹ Gen ²	Par ³
Paci	1 (2	O(n)		14		10
Our	12 + 0.007	* ~ /	inimize	100		100
goals:	1 + 0.003 $ (vlylog(n))$	44	passes	i de	3	3
guais.	ε	$C = l \log n$	$\frac{U(1r-\epsilon/\epsilon)/\epsilon}{\epsilon}$	1 miles	R. C.	
[6, Theorem 19]		(n polylor	$\log(1/\varepsilon)$ /8	2	R. C.	100
Maxing	mize	n polylog	Accept	1 P	R. C.	10
[4 acciii	racy	$n\log n$	weighted	1		
[4 accui	lacy	1 n polylog n polylog	graphs	B (A)	ide ide	
[28, Theorem 20]	1 - 1/e	11)	Brapins	H.	RIP	
[35, Theorem 2]	Minin	nize	k	H.	H.	1
[14]	local s	oace	1	H.	Ď	O
[14]	$1/\varepsilon$	$O\left(n^2/\varepsilon^3\right)$	1	H.	Ď	ß
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{2}\right)$	$1^{1-\varepsilon}$) 1	nde.	R.	3
[26, Theorem 2]	6	$O(n\log n)$	1	B	6	
[44, Theorem 3]	$2+\varepsilon$	O(n polylog(n))				
[44, Theorem 3] [63]	5.82 5.58	O(n polylog(n))			6	
[25]	$4.911 + \varepsilon$	O(n polylog(n))				
[29]	$3.5 + \varepsilon$	O(n polylog(n))	n)) 1			
[53]	$2 + \varepsilon$	$O\left(n\log^2 n\right)$	1			
[27]	$2 + \varepsilon$	$O(n \log n)$	1			
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$			
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)\right)$	$O\left(\varepsilon^{-4}\log n\right)$		0	100
[6, Theorem 22]	$\frac{1}{\frac{2}{2}(1-\varepsilon)}$	$O\left(n\left(\frac{\varepsilon\log n}{\varepsilon^2}\right)\right)$	$\left(\frac{-\log \varepsilon}{2}\right) O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ô	Ů	ndr
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	$O\left(n\left(\frac{\varepsilon\log n}{\varepsilon^2}\right)\right)$	$\left(-\log \varepsilon\right)$ $O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$	Ô	R.	10
[17]	$4 + \varepsilon$	O(n polylog(n	n)) 1	ß	ß	ß

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?





D. C.			#P		1.0	2 p. 3
Reference	Approx. 1 2 1 2+0.007 + 0.003	O(pol	#Passes Minimize #passes	Wgl	1 Gen	Par ³
goals:] ($\frac{1}{2}$ $\frac{1}{2}$ $\frac{1}{2}$	(1)) O folylog $\frac{1}{2}$ O	$\frac{1}{2}$	Acc	cept	
[4 Maxir [4 accur		$n \text{ polylog}$ $n \text{ polylog}$ $n \log n$ $n \text{ polylog}$ $n \text{ polylog}$	Accept weighted graphs	just bi		
[28, Theorem 20]		11)	1	10	1	100
[35, Theorem 2]	Minin		k	n (f	1	
[14]	local s	oace)	1	1		
[14]	$1/\varepsilon$	$O\left(n^2/\varepsilon^3\right)$	1	100		
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+\right)$	$n^{1-\varepsilon}$) 1	14	R.	6
[26, Theorem 2] [44, Theorem 3] [44, Theorem 3] [63] [25] [29] [53]	6 $2 + \varepsilon$ 5.82 5.58 $4.911 + \varepsilon$ $3.5 + \varepsilon$ $2 + \varepsilon$	$O(n \log n)$ $O(n \text{ polylog})$ $O(n \log^2 n)$ $O(n \log n)$	(n) $O(1)$ (n) 1 (n) 1 (n) 1 (n) 1 (n) 1			
[26, Section 3.2]	$2 + \varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3}n\right)$			
[6, Theorem 28]	$\frac{1}{1-\varepsilon}$	$O\left(n\log(n)\right)$		O		ndr
[6, Theorem 22]	$\frac{\frac{1}{\frac{2}{3}(1-\varepsilon)}}{1}$	$O\left(n\left(\frac{\varepsilon\log t}{\varepsilon}\right)\right)$	$\left(\frac{n-\log\varepsilon}{\varepsilon^2}\right) O\left(\varepsilon^{-2}\log(\varepsilon^{-2}\log(\varepsilon^{-2}\log(\varepsilon)}\right)\right)\right)\right)\right)\right)\right)\right)\right)\right)\right)\right)\right)\right)\right)\right)}\right)\right)}\right)}$		ß	100
[6, Theorem 22]	$\frac{1-\varepsilon}{1-\varepsilon}$	O(n)	e ² // \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	-1)) <u>o</u>	*	<u></u>

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?

Reference	Approx.	Space	#Passes	Wgl	1^1 Gen	² Par ³
Our	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	* O(pol	inimize	4		10
goals:] (Nylog(π - ε	C olylo H C olylo C C olylo C C olylo C C C C C C C C C	passes (E)/	Acc	cept	
[4 Maxir	5	n polylog n polylog n log n) n polylog n polylog n polylog	Accept weighted graphs	gener just bi _l gra		
[28, Theorem 20]		11)	graphs	R. C.	14	:4
[35, Theorem 2]	Minin	nize	k	·	E I	8.00
[14]	local s	oace)	1	nde.		
[14]	$1/\varepsilon$	$O\left(n^2/\varepsilon^3\right)$	1	nde.		
[7, Theorem 1]	n^{ε}	$\tilde{O}\left(n^{2-3\varepsilon}+n^{2}\right)$	$(1-\varepsilon)$ 1	nde.	R.	6
[26, Theorem 2] [44, Theorem 3] [44, Theorem 3] [63] [25] [29] [53]	6 $2 + \varepsilon$ 5.82 5.58 $4.911 + \varepsilon$ $3.5 + \varepsilon$ $2 + \varepsilon$ $2 + \varepsilon$	$O(n \log n)$ $O(n \operatorname{polylog}(n \log n))$ $O(n \operatorname{polylog}(n \log n))$ $O(n \operatorname{polylog}(n \log n))$ $O(n \operatorname{polylog}(n \log^2 n))$ $O(n \log n)$	$\begin{pmatrix} n \\ n \end{pmatrix} \begin{pmatrix} 1 \\ n \\ n \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	0 0 0 0 0 0 0 0 0 0 0 0 0		
[26, Section 3.2]	$2+\varepsilon$	$O(n \log n)$	$O\left(\log_{1+\varepsilon/3} n\right)$	O	B	
[6, Theorem 28] [6, Theorem 22]	$\frac{\frac{1}{1-\varepsilon}}{\frac{2}{3}(1-\varepsilon)}$	$O\left(n\log(n)\right)$ $O\left(n\left(\frac{\varepsilon\log n}{\varepsilon^2}\right)\right)$	e^4 $O\left(\varepsilon^{-4}\log n\right)$		6	: ú
[6, Theorem 22]	$\frac{1}{1-\varepsilon}$	-3	$\left(\frac{-\log \varepsilon}{2}\right) O\left(\varepsilon^{-2}\log\left(\varepsilon^{-1}\right)\right)$		·	:6
[17]	$4 + \varepsilon$	O(n polylog(n	n)) 1	ß	ß	ß

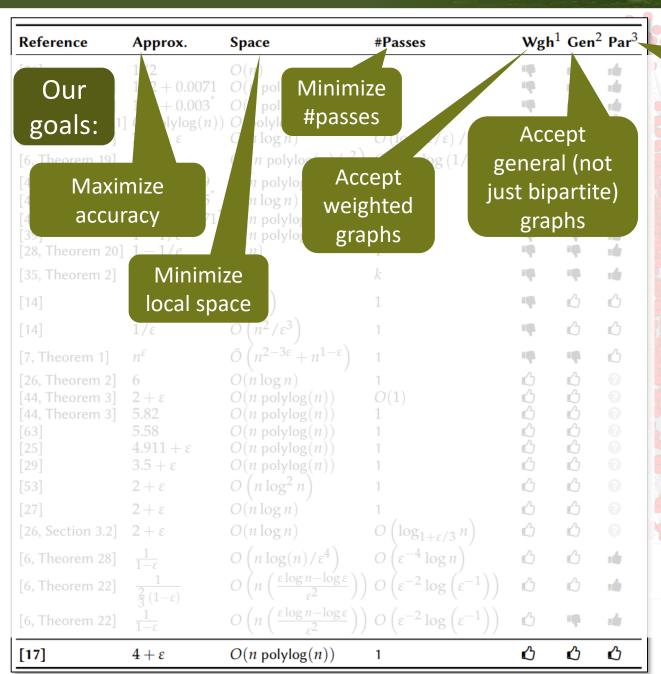
Expose parallelism (match substream-centric)

Part 3 continued:
Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?





Expose parallelism (match substream-centric)

Part 3 continued:
Analysis of models and algorithms for streaming graph processing

~30 algorithms for streaming (approximate) MWM (in the semi-streaming model)

Any interesting idea to use in the context of FPGAs and substream-centric processing?







How to design a highperformance MWM algorithm (as dictated by the used paradigm)? Use <u>substream-centric</u>

processing (exposes

parallelism)

parallelism)

parallelism theoretical analysis

and rigor in the context of

and rigor in the context of

the semi-streaming model

the semi-streaming model

What is the HW FPGA design that ensures high performance?







Use the MWM algorithm by

Crouch and Stubbs in the

Crouch and Stubbs

Use <u>substream-centric</u>

processing (exposes

parallelism)

parallelism)

parallelism theoretical analysis

and rigor in the context of

and rigor in the context of

the <u>semi-streaming model</u>

the <u>semi-streaming model</u>

What is the HW FPGA design that ensures high performance?







Use the MWM algorithm by

Crouch and Stubbs in the

Crouch and Stubbs

Use <u>substream-centric</u>

processing (exposes

parallelism)

parallelism)

parallelism theoretical analysis

and rigor in the context of

and rigor in the context of

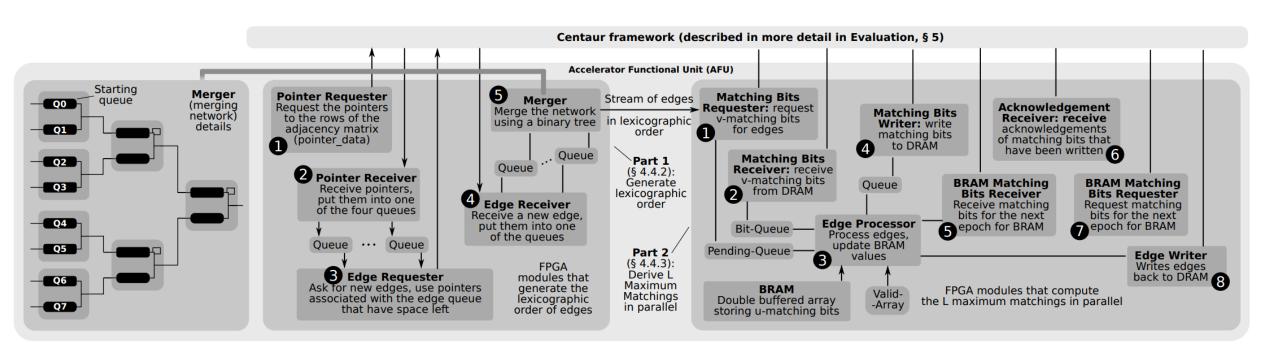
the <u>semi-streaming model</u>

the <u>semi-streaming model</u>

What is the HW FPGA design that ensures high performance?

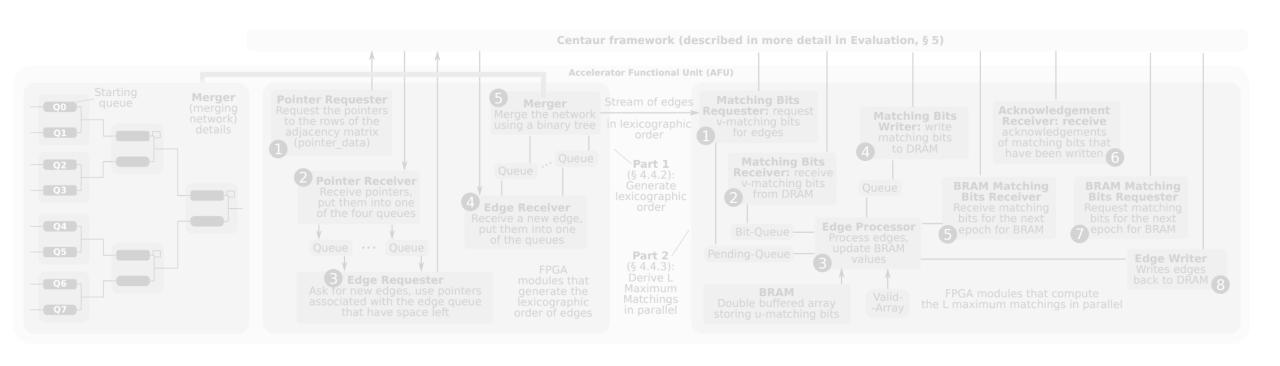


























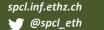
Blocking / Tiling



Prefetching

Pipelining







Blocking / Tiling

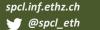


Prefetching

They are often used in graph processing schemes on FPGAs; we apply them as well.

Pipelining







Blocking / Tiling

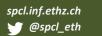


Prefetching

They are often used in graph processing schemes on FPGAs; we apply them as well.

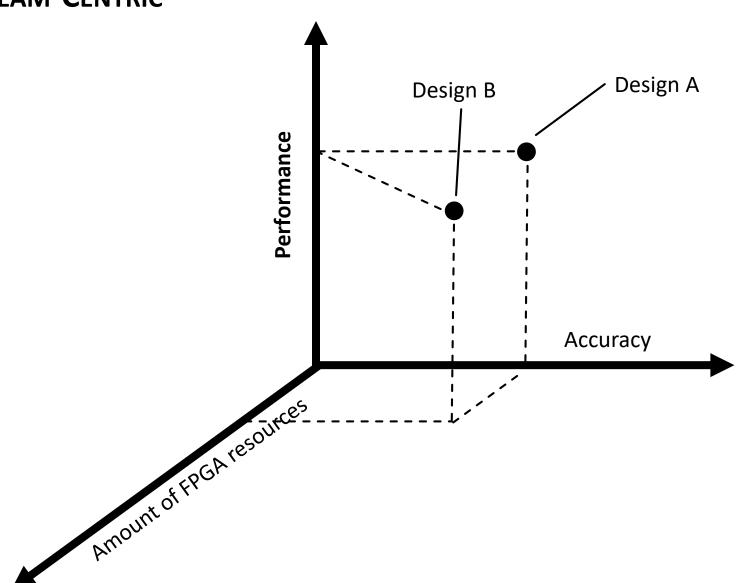
Pipelining







THE SPACE OF SUBSTREAM-CENTRIC









Use the MWM algorithm by

Crouch and Stubbs in the

Crouch and Crouch and Stubbs in the

Crouch and Crouch and

Use <u>substream-centric</u>

processing (exposes

parallelism)

parallelism

parallelism

theoretical analysis

and rigor in the context of

and rigor in the context of

the <u>semi-streaming model</u>

the <u>semi-streaming model</u>

What is the HW FPGA design that ensures high performance?







Use the MWM algorithm by

Crouch and Stubbs in the

Crouch and Crouch and Stubbs in the

Crouch and Stubbs in the

Crouch and Crouch

Use <u>substream-centric</u>

processing (exposes

parallelism)

parallelism

theoretical analysis

and rigor in the context of

and rigor in the context of

and rigor in the semi-streaming model

the <u>semi-streaming model</u>

The proper use of blocking, vectorization, and others (pipelining) prefetching)

What is the ult performance, p







Use the MWM algorithm by

Crouch and Stubbs in the

Crouch and Crouch and Stubbs in the

Crouch and Stubbs in the

Crouch and Crouch

Use <u>substream-centric</u>

processing (exposes

parallelism)

parallelism

theoretical analysis

and rigor in the context of

and rigor in the context of

the <u>semi-streaming model</u>

the <u>semi-streaming model</u>

The proper use of blocking, vectorization, and others (pipelining) prefetching)

?





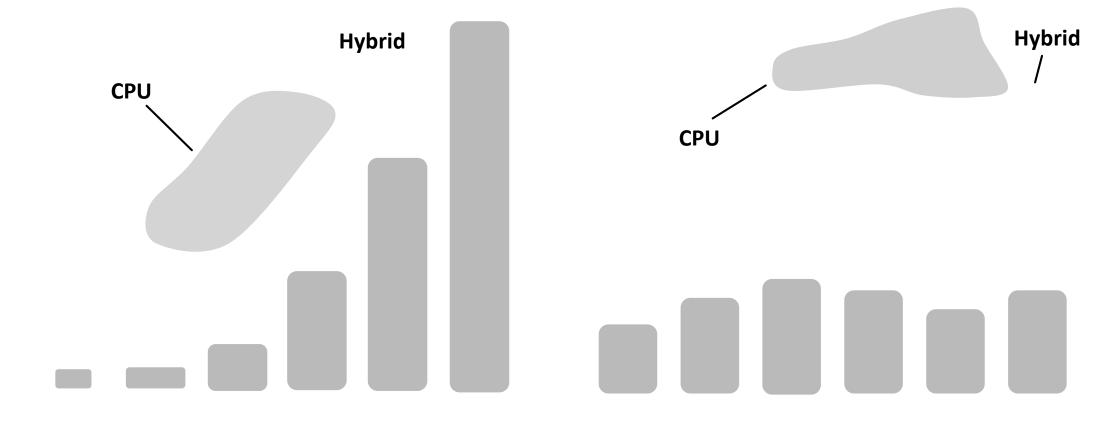


PERFORMANCE ANALYSIS VARIOUS GRAPHS

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ)	CPU CPU CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

Blocking size = 32, #Substreams = 64 #Threads = 4, ε = 0.1





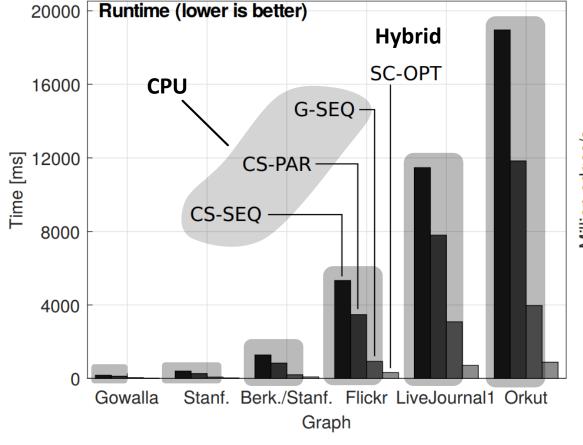
PERFORMANCE ANALYSIS VARIOUS GRAPHS

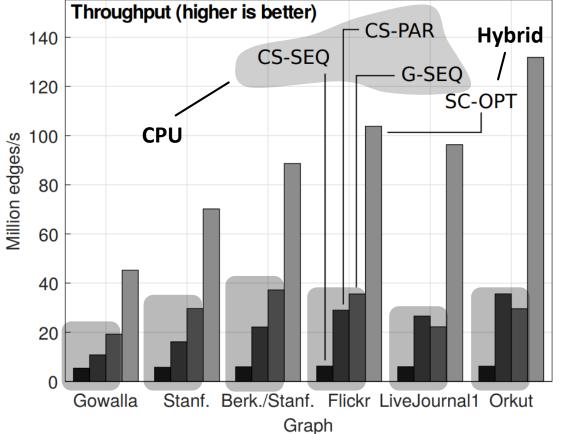
Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

Blocking size = 32, #Substreams = 64 #Threads = 4, ε = 0.1

Graph	Туре	т	п
Orkut Stanford Berkeley	Synthetic power-law Social network Social network Social network Social network Hyperlink graph Hyperlink graph Citation graph	\approx 48 <i>n</i> 950,327 33,140,017 68,993,773 117,184,899 2,312,497 7,600,595 352,807	2^k ; $k = 16,, 21$ 196,591 2,302,925 4,847,571 3,072,441 281,903 685,230 27,770







PERFORMANCE ANALYSIS VARIOUS GRAPHS

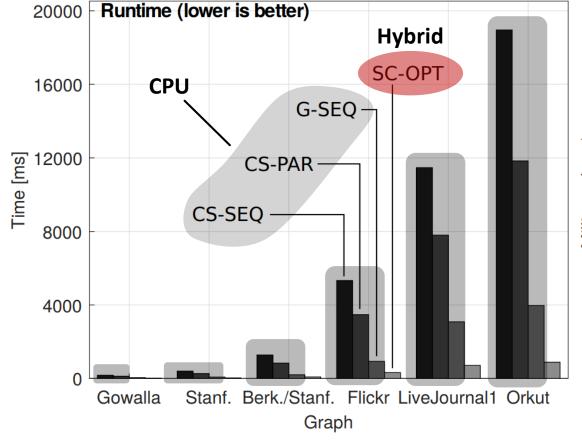
Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

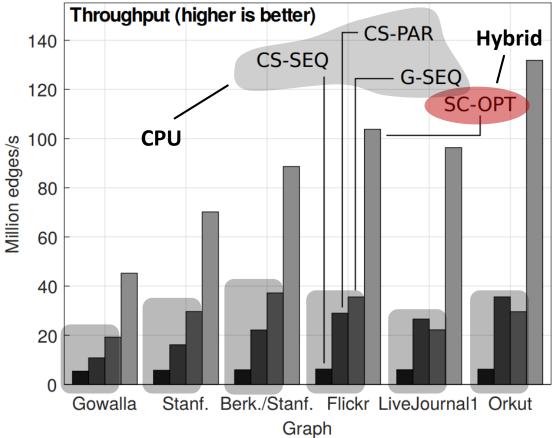
Parameters:

Blocking size = 32, #Substreams = 64 #Threads = 4, ε = 0.1

SC-OPT secures highest performance

Graph	Type	m	п
Orkut Stanford Berkeley	Synthetic power-law Social network Social network Social network Social network Hyperlink graph Hyperlink graph Citation graph	\approx 48 <i>n</i> 950,327 33,140,017 68,993,773 117,184,899 2,312,497 7,600,595 352,807	2^k ; $k = 16,, 21$ 196,591 2,302,925 4,847,571 3,072,441 281,903 685,230 27,770











PERFORMANCE ANALYSIS VARIOUS THREAD (CPU) COUNTS

Algorithm

Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric, no blocking (SC-SIMPLE) Substream-Centric, with blocking (SC-OPT)

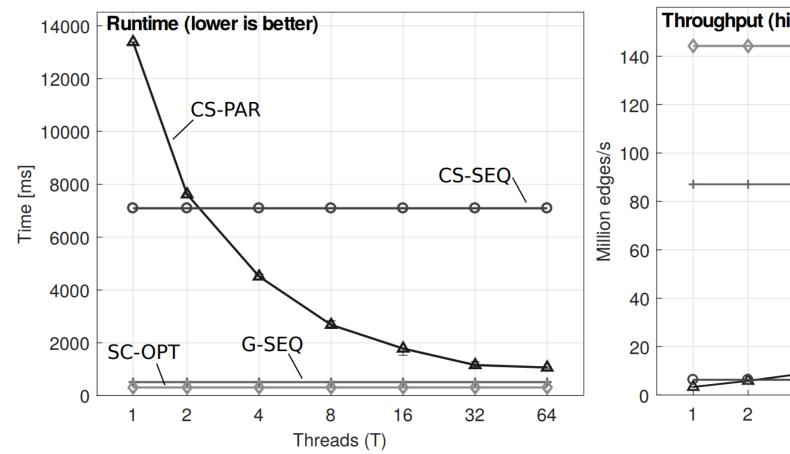


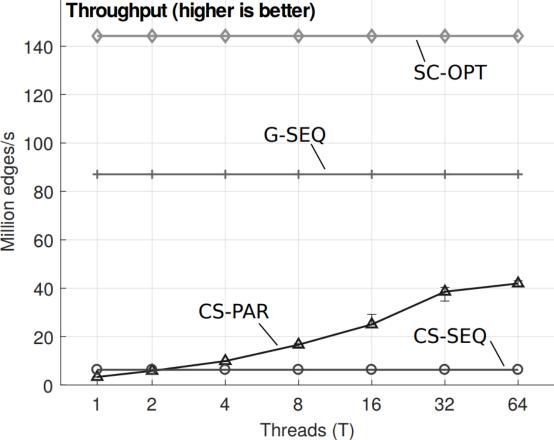


PERFORMANCE ANALYSIS VARIOUS THREAD (CPU) COUNTS

Algorithm

Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric, no blocking (SC-SIMPLE) Substream-Centric, with blocking (SC-OPT) Blocking size = 32, #Substreams = 64 #edges = 16M (Kronecker), ε = 0.1







PERFORMANCE ANALYSIS VARIOUS THREAD (CPU) COUNTS

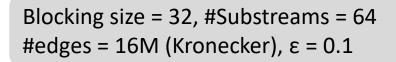
Algorithm

Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR)

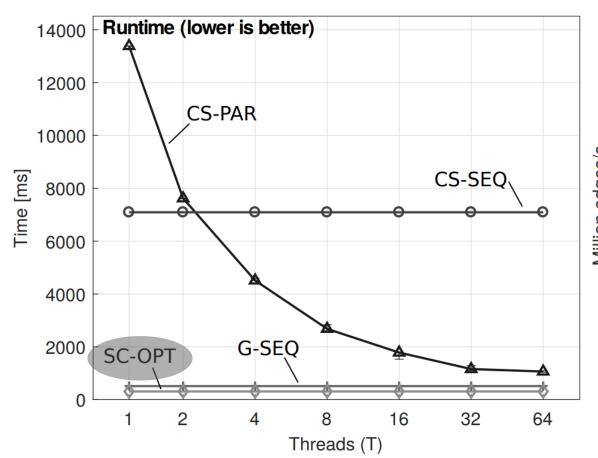
Ghaffari [2] Sequential (G-SEQ)

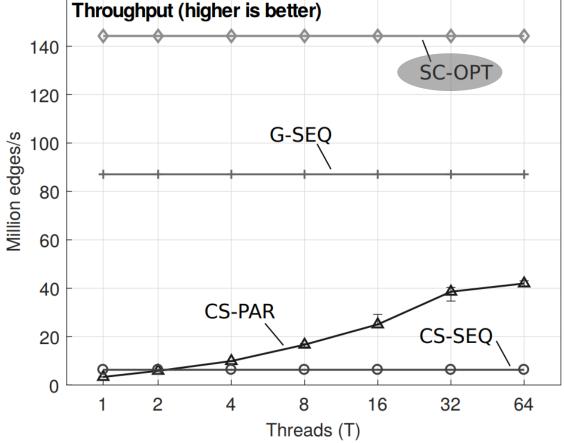
Substream-Centric, no blocking (SC-SIMPLE)

Substream-Centric, with blocking (SC-OPT)



SC-OPT secures highest performance for all considered numbers of threads











PERFORMANCE ANALYSIS VARIOUS BLOCKING SIZE (K) AND #SUBSTREAMS (L)

Algorithm

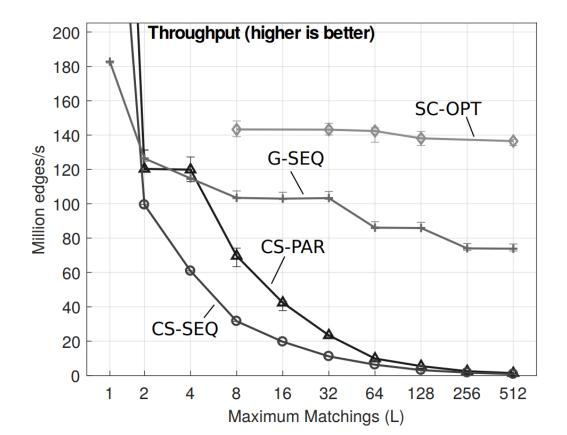
Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric, no blocking (SC-SIMPLE) Substream-Centric, with blocking (SC-OPT)

PERFORMANCE ANALYSIS VARIOUS BLOCKING SIZE (K) AND #SUBSTREAMS (L)

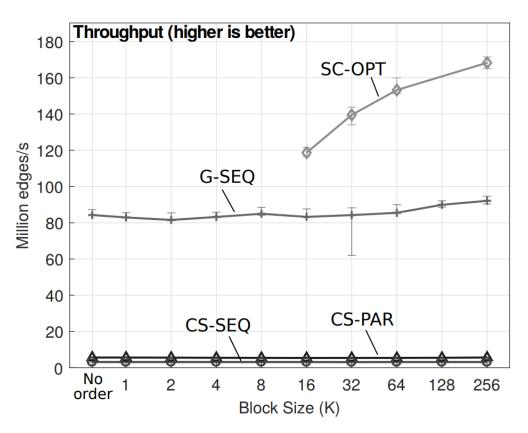
Algorithm

Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric, no blocking (SC-SIMPLE) Substream-Centric, with blocking (SC-OPT)

Blocking size (K) = 32, #threads = 4, #edges = 16M (Kronecker), ε = 0.1



#Substreams (L) = 128, #threads = 4, #edges = 16M (Kronecker), ε = 0.1

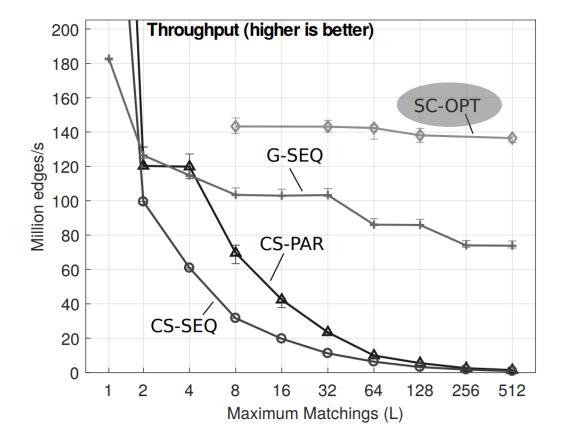


Performance Analysis VARIOUS BLOCKING SIZE (K) AND #SUBSTREAMS (L)

Algorithm

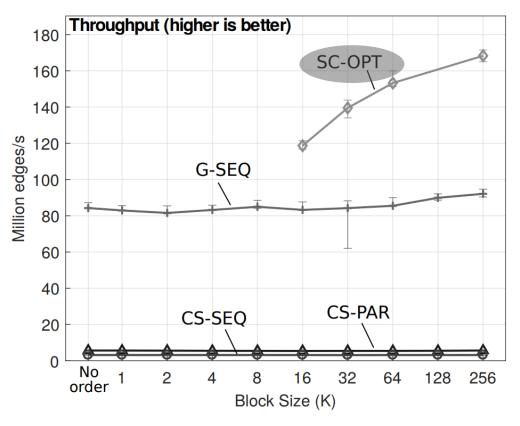
Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric, no blocking (SC-SIMPLE) Substream-Centric, with blocking (SC-OPT)

Blocking size (K) = 32, #threads = 4, #edges = 16M (Kronecker), ε = 0.1

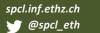


SC-OPT secures highest performance for all considered values of parameters

#Substreams(L) = 128, #threads = 4,#edges = 16M (Kronecker), ε = 0.1









PERFORMANCE ANALYSIS APPROXIMATION

Algorithm

Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric, no blocking (SC-SIMPLE) Substream-Centric, with blocking (SC-OPT)



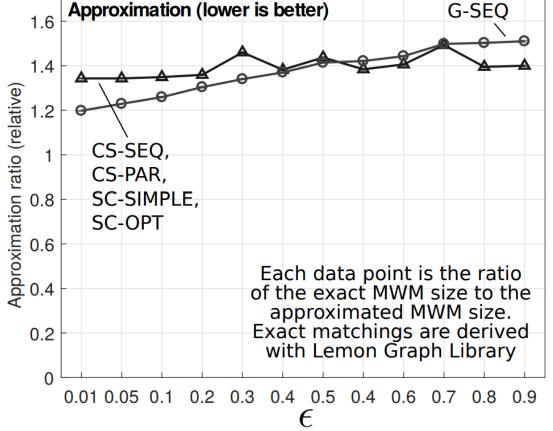


PERFORMANCE ANALYSIS APPROXIMATION

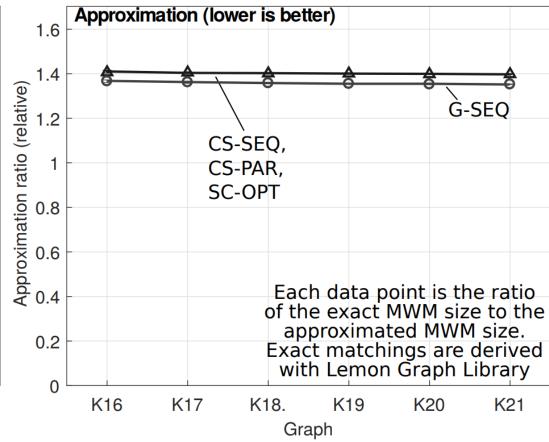
Algorithm

Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric, no blocking (SC-SIMPLE) Substream-Centric, with blocking (SC-OPT)

#Substreams (L) = 128, Blocking size (K) = 32, #threads = 4, #edges = 8M (Kronecker)



Blocking size (K) = 32, #threads = 4, #Substreams (L) = 128, ε = 0.1





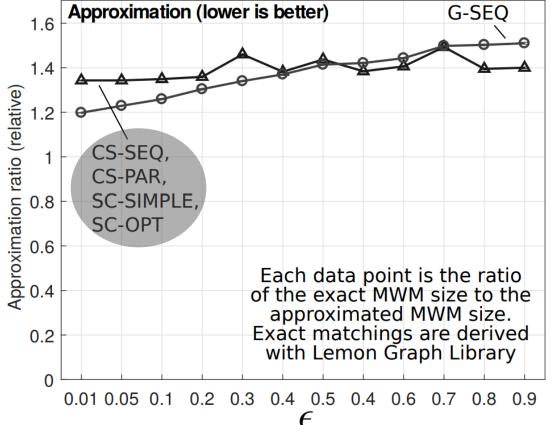


PERFORMANCE ANALYSIS APPROXIMATION

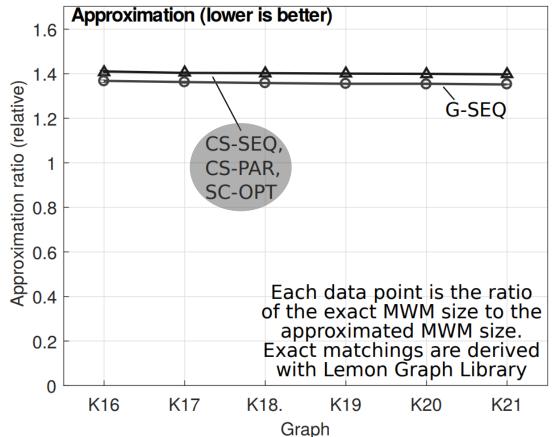
Algorithm

Crouch et al. [1] Sequential (CS-SEQ) Crouch et al. [1] Parallel (CS-PAR) Ghaffari [2] Sequential (G-SEQ) Substream-Centric, no blocking (SC-SIMPLE) Substream-Centric, with blocking (SC-OPT) SC-OPT is comparable to the $(2+\epsilon)$ -approximation by Ghaffari et al.

#Substreams (L) = 128, Blocking size (K) = 32, #threads = 4, #edges = 8M (Kronecker)



Blocking size (K) = 32, #threads = 4, #Substreams (L) = 128, ε = 0.1

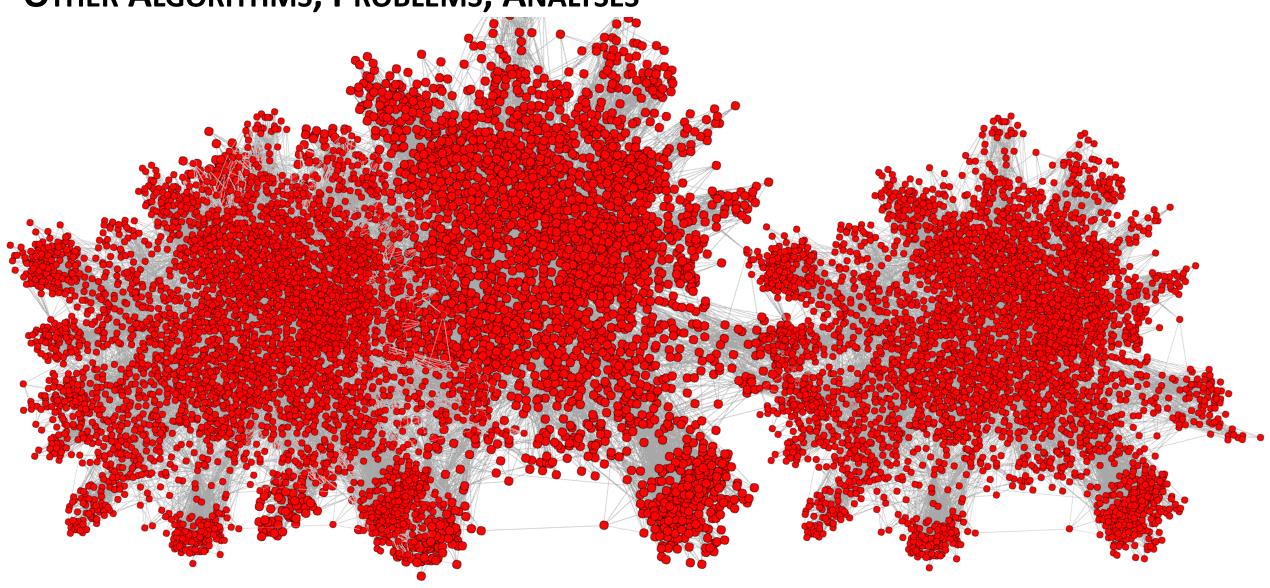




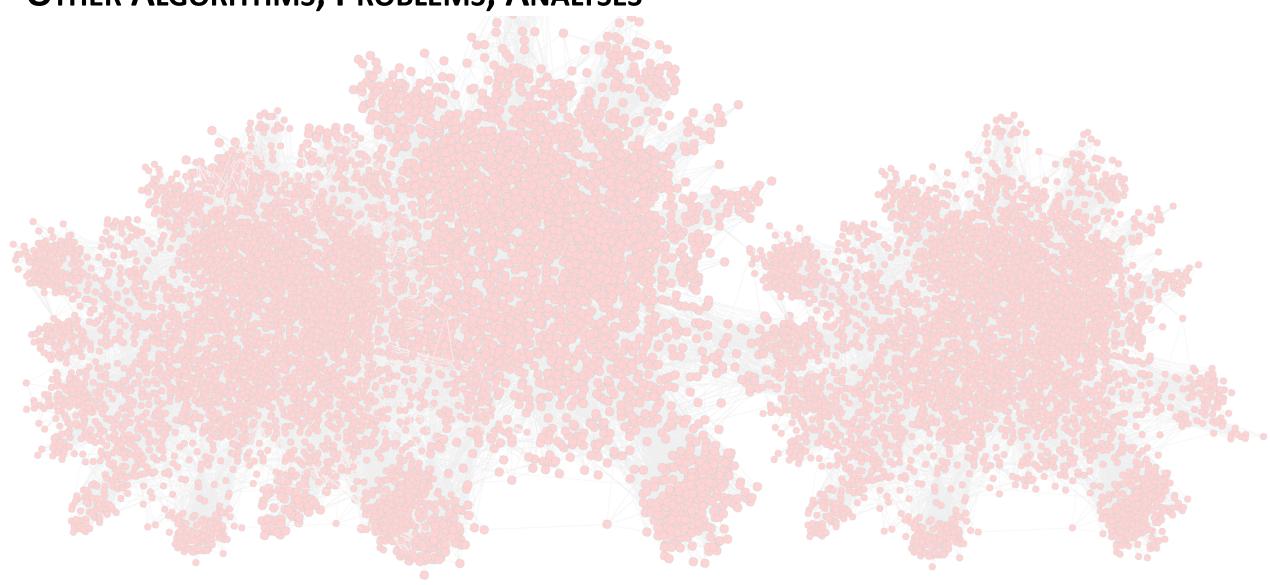








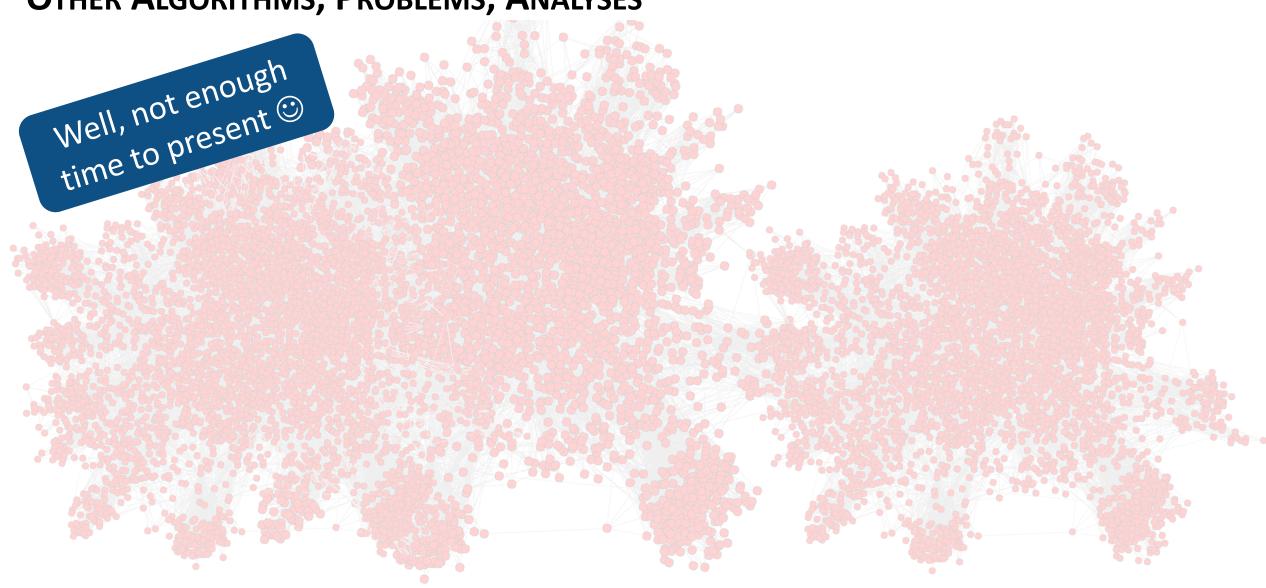


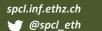




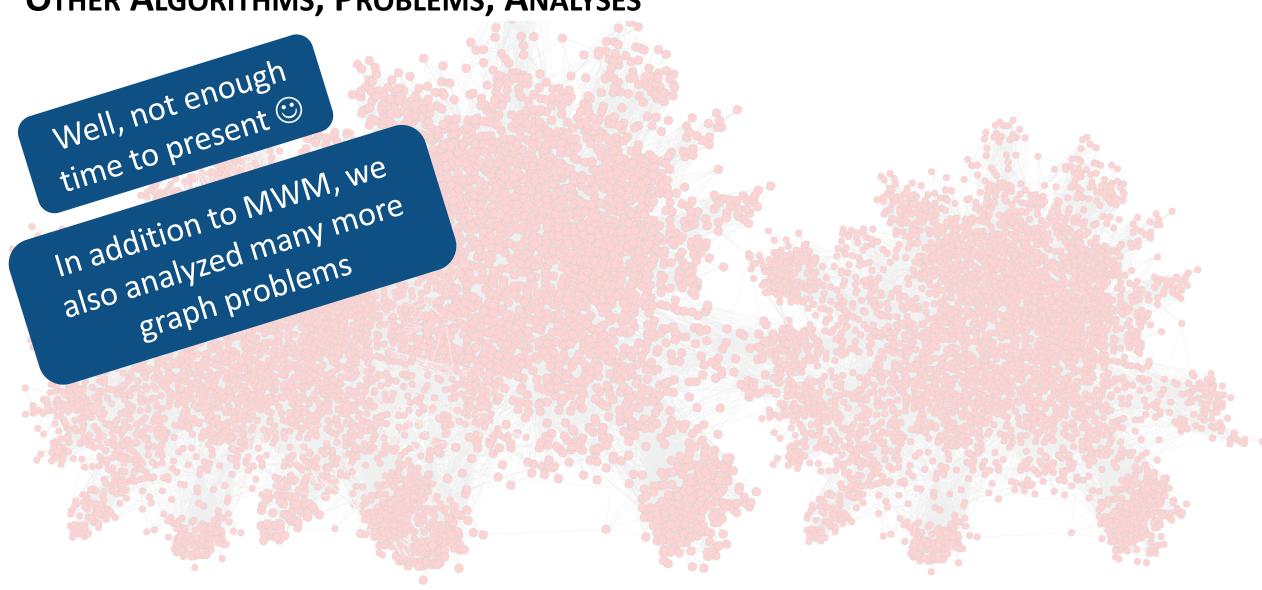












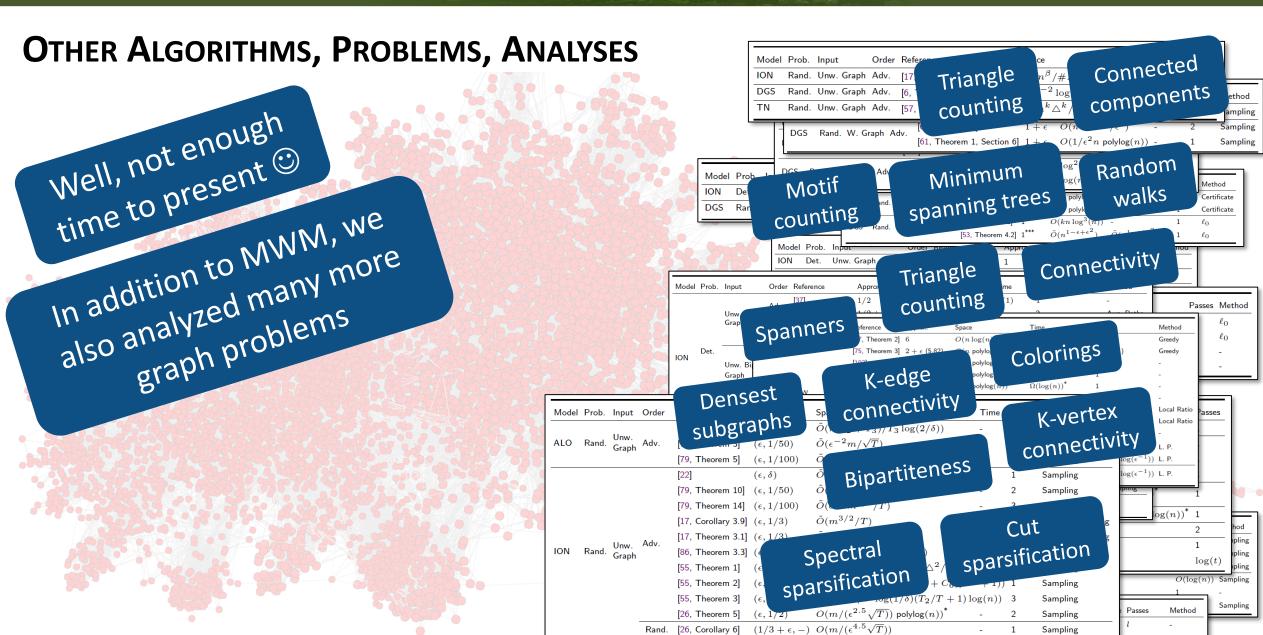






 $\tilde{O}(\sqrt{l/\alpha})$ Sampling

 $O(n\alpha + \sqrt{l/\alpha})$

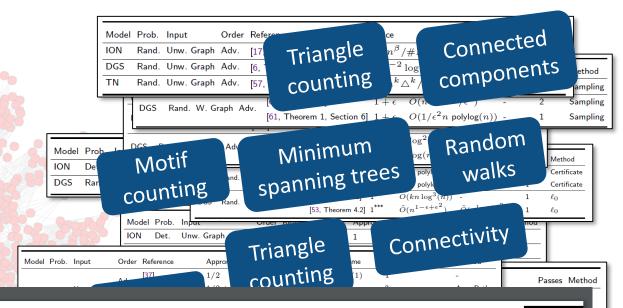








Well, not enough
time to present ime to present well, we
In addition to MWM, we
also analyzed many more
graph problems



Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Storage

MARC FISCHER, Department of Computer Science, ETH Zurich MACIEJ BESTA, Department of Computer Science, ETH Zurich TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

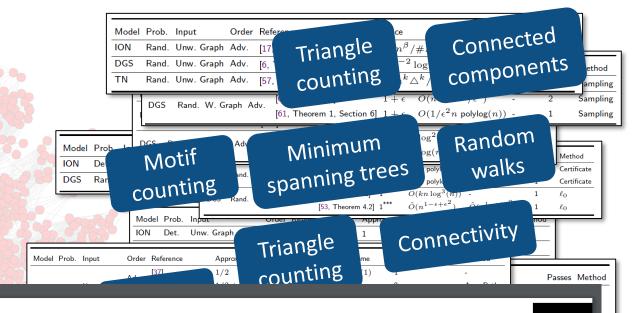
Graph processing has become an important part of various areas of computer science, including machine learning, social network analysis, computational sciences, and others. Two key challenges that hinder accelerating







Well, not enough time to present © In addition to MWM, we also analyzed many more graph problems



Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Again, the most relevant Towards Understanding of Modern Graph Processing and Stor

parts are in the FPGA paper, the rest in this survey MARC FISCHER, Department of Computer (ready in ~1-2 months) MACIEJ BESTA, Department of Computer TAL BEN-NUN, Department of Computer S TORSTEN HOEFLER, Department of Comp

Graph processing has become an important part of va ing, social network analysis, computational sciences, graph processing are (1) sizes of input detects reach and (2) the growing rate of graph



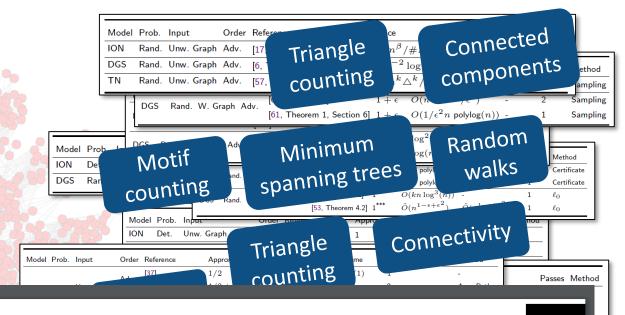




Well, not enough time to present ©

In addition to MWM, we also analyzed many more graph problems

Preliminary substreamcentric designs and results



Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Again, the most relevant Towards Understanding of Modern Graph Processing and Stor

parts are in the FPGA paper, the rest in this survey MARC FISCHER, Department of Computer MACIEJ BESTA, Department of Computer TAL BEN-NUN, Department of Computer S TORSTEN HOEFLER, Department of Comp

Graph processing has become an important part of va ing, social network analysis, computational sciences, graph processing are (1) sizes of input detects reach

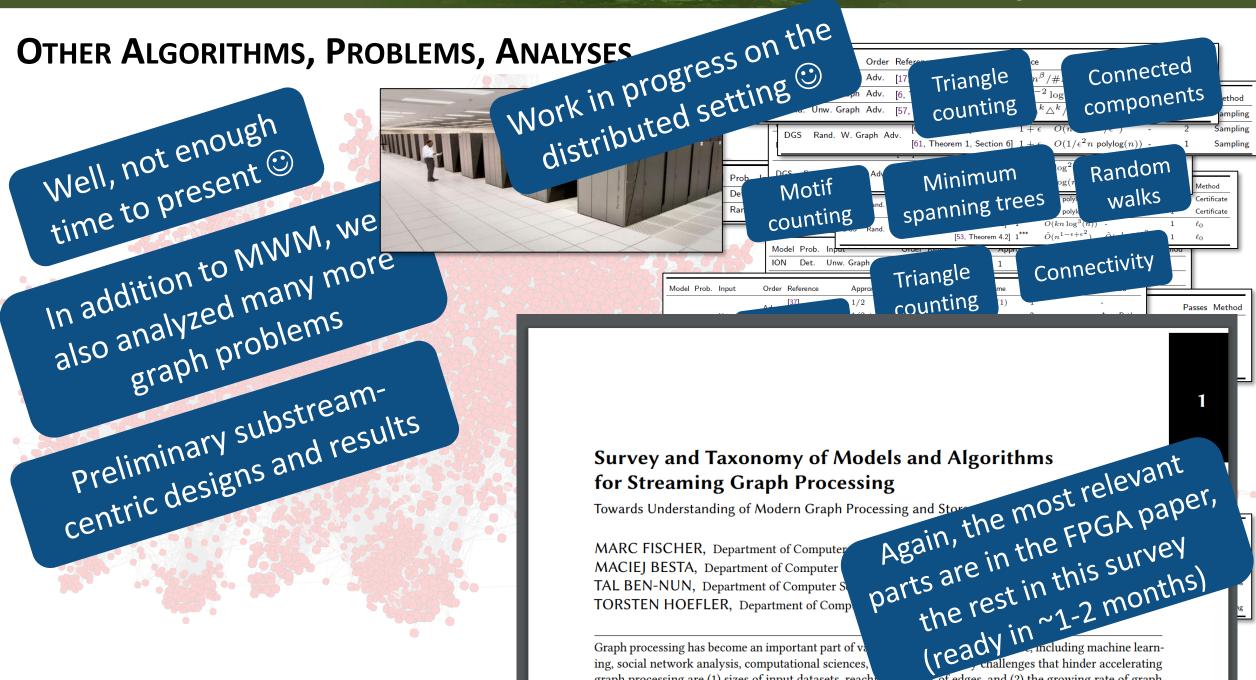
(ready in ~1-2 months) and (2) the graving rate of graph





and (2) the graving rate of graph





ing, social network analysis, computational sciences, graph processing are (1) sizes of input detects reach





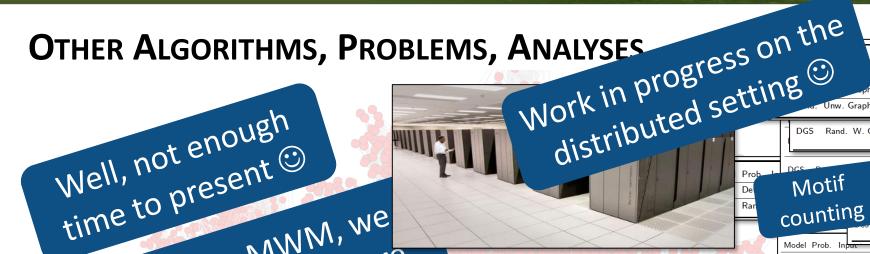


Connected

components

Random

walks



In addition to MWM, we also analyzed many more graph problems -- substream-

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Stor

Again, the most relevant parts are in the FPGA paper, the rest in this survey MARC FISCHER, Department of Computer (ready in ~1-2 months) MACIEJ BESTA, Department of Computer TAL BEN-NUN, Department of Computer S TORSTEN HOEFLER, Department of Comp

Graph processing has become an important part of va ing, social network analysis, computational sciences, graph propaging are (1) sizes of input datasets, reach and (2) the growing rate of graph

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

-cults

Model Prob. Input

Motif

counting

ION Det. Unw. Graph

DGS Rand, W. Graph Adv.

Order Refe

Triangle counting

Triangle

counting

[61, Theorem 1, Section 6] 1 ±

Minimum

spanning trees

Connectivity

Passes Method







Passes Method

Connected

components

Random

walks

Connectivity



Well, not enough time to present © In addition to MWM, we also analyzed many more graph problems

-- substream-

https://arxiv.org/abs/...

Graph Processing on FPGAs: Taxonomy, Survey, Challenges

Towards Understanding of Modern Graph Processing, Storage, and Analytics

MACIEJ BESTA*, DIMITRI STANOJEVIC*, Department of Computer Science, ETH Zurich JOHANNES DE FINE LICHT, TAL BEN-NUN, Department of Computer Science, ETH Zurich TORSTEN HOEFLER, Department of Computer Science, ETH Zurich

Survey and Taxonomy of Models and Algorithms for Streaming Graph Processing

Towards Understanding of Modern Graph Processing and Stor

Order Refe

DGS Rand, W. Graph Adv.

Motif

counting

ION Det. Unw. Graph

Model Prob. Input

Triangle

counting

[61, Theorem 1, Section 6] 1 ±

Minimum

spanning trees

Triangle

counting

Again, the most relevant parts are in the FPGA paper, the rest in this survey MARC FISCHER, Department of Computer (ready in ~1-2 months) MACIEJ BESTA, Department of Computer TAL BEN-NUN, Department of Computer S TORSTEN HOEFLER, Department of Comp

Graph processing has become an important part of va ing, social network analysis, computational sciences, and (2) the growing rate of graph graph propaging are (1) signs of input datacate reach







An incoming edge...













An incoming edge... e = (u, v, weight)

Vertices + matchings (correctness)







An incoming edge... e = (u, v, weight)

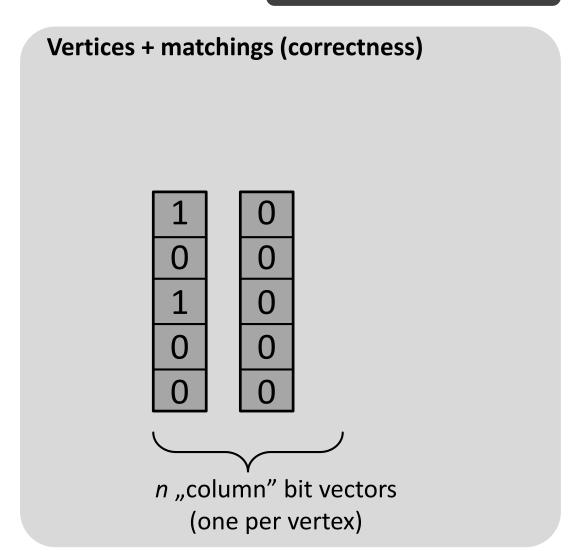
Vertices + matchings (correctness)







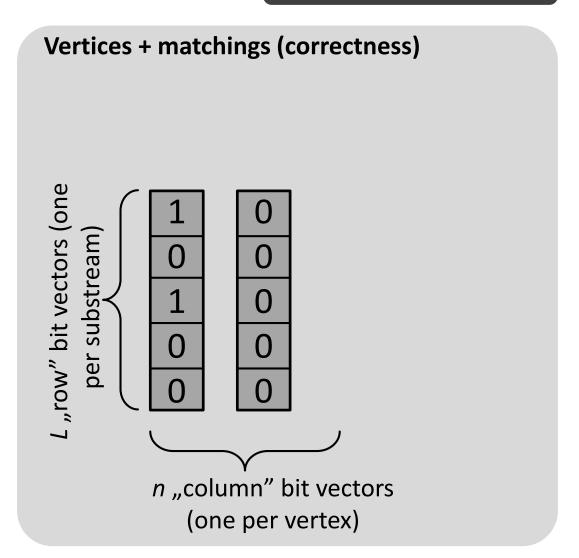
An incoming edge...
$$e = (u, v, weight)$$







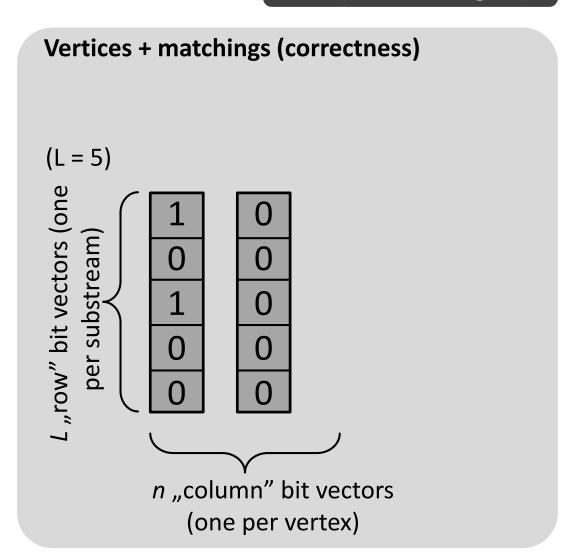








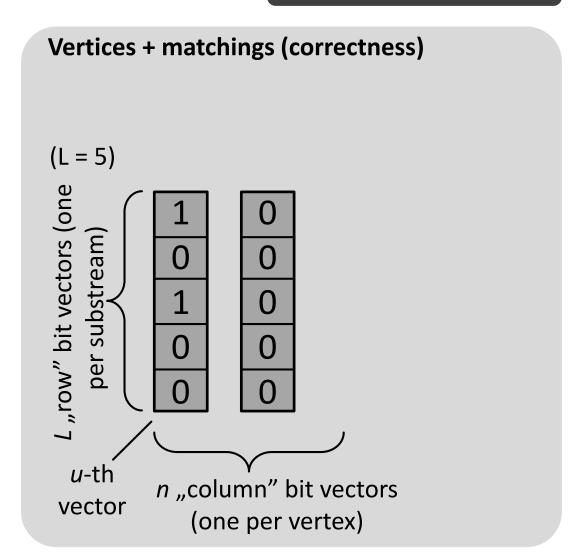








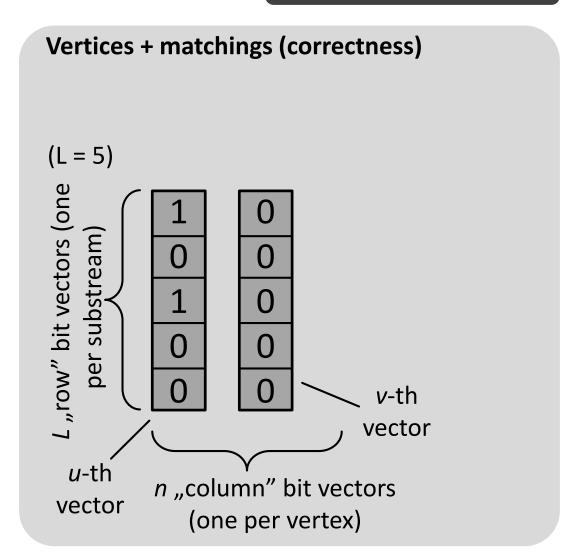








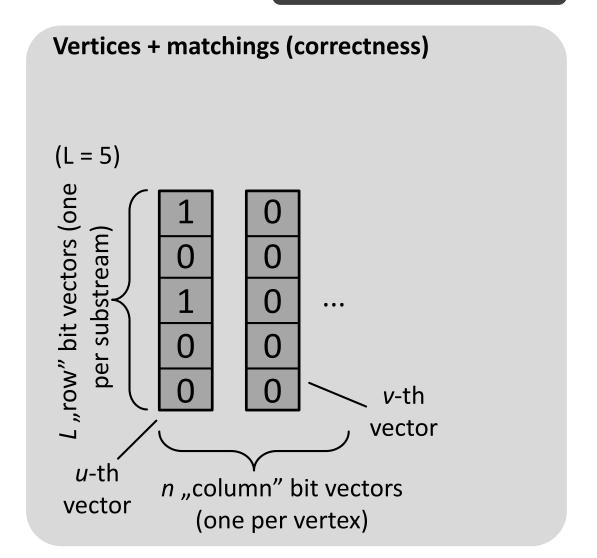








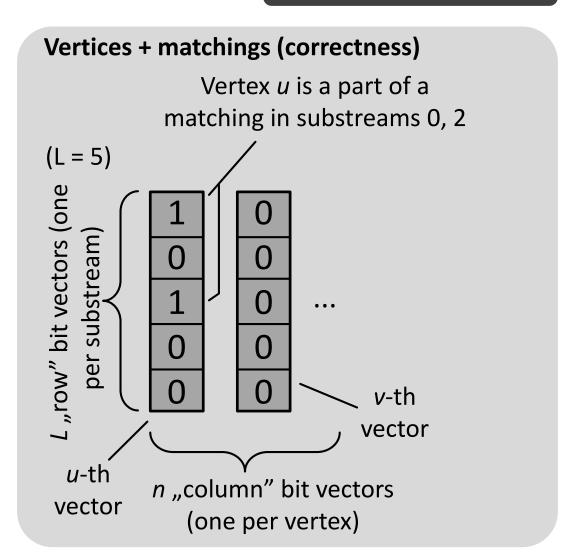








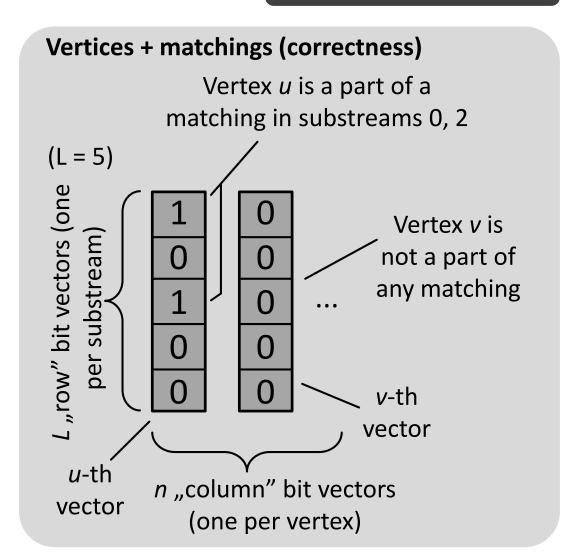










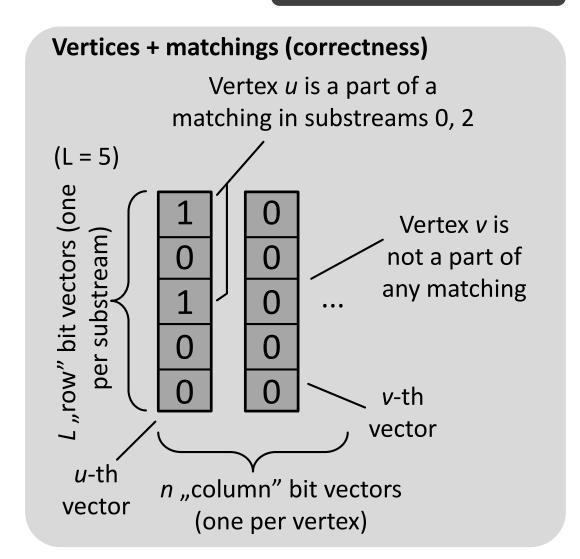








An incoming edge... e = (u, v, weight)



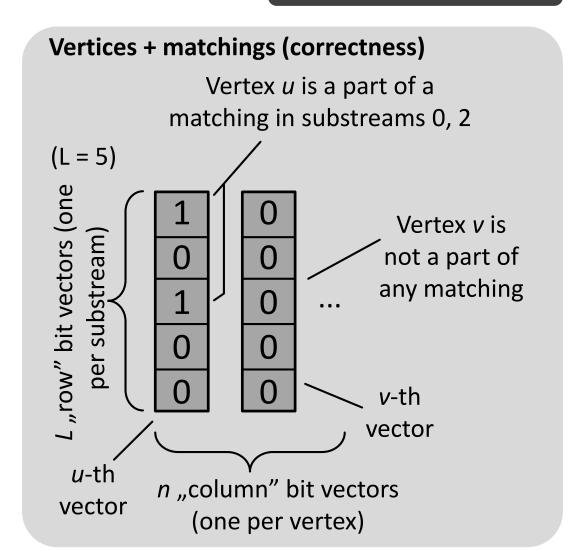
Edges + matchings (more performance)







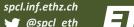
An incoming edge... e = (u, v, weight)



Edges + matchings (more performance)

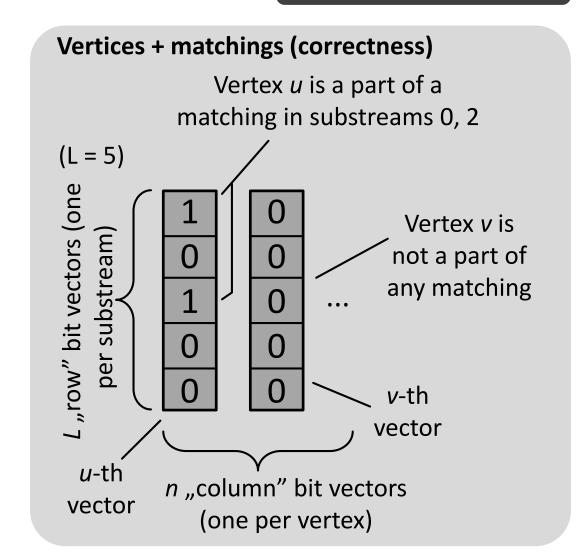


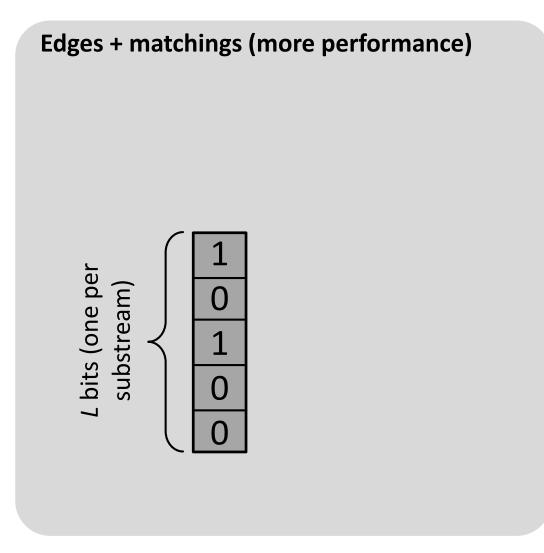




ETH zürich

MATCHING BITS: KEY DATA STRUCTURES FOR MAINTAINING INFORMATION ON MATCHINGS

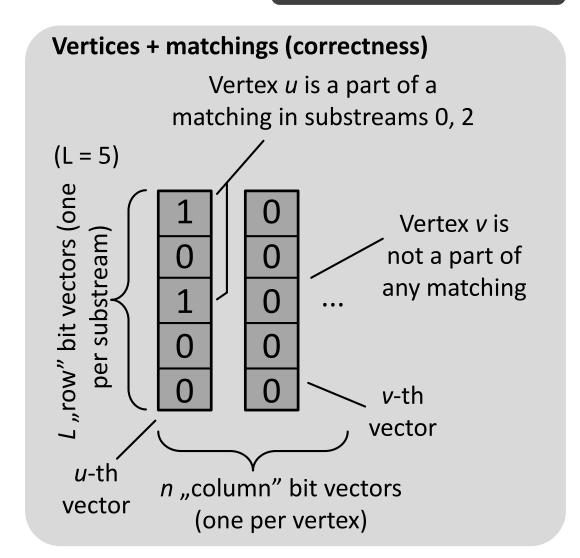


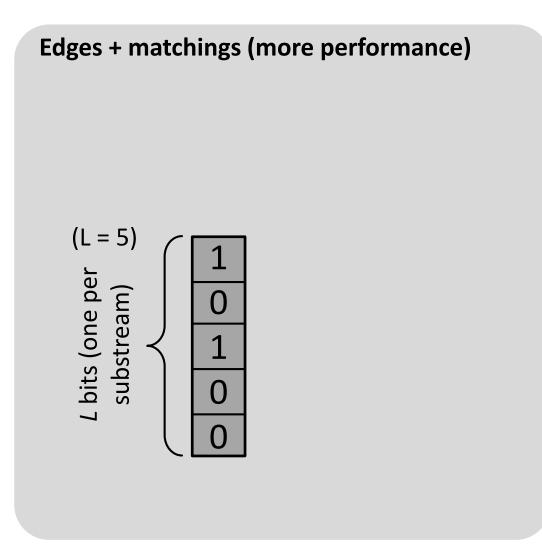






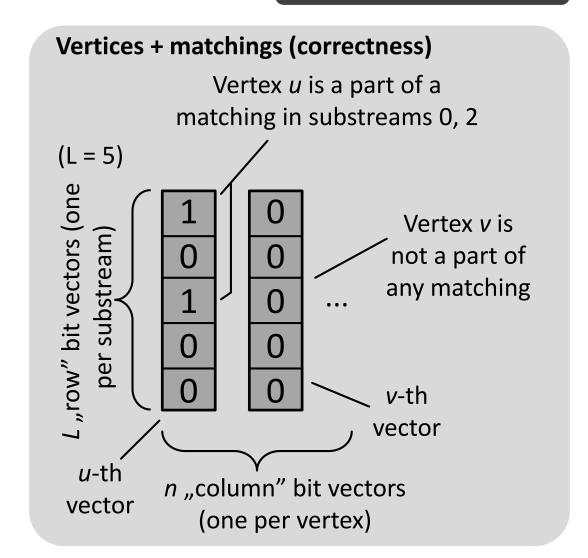


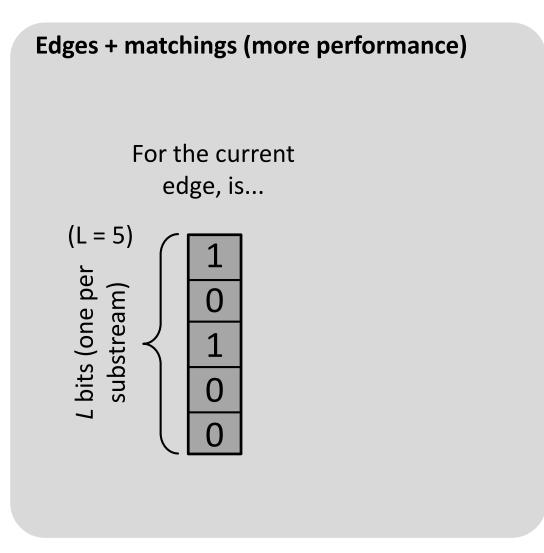






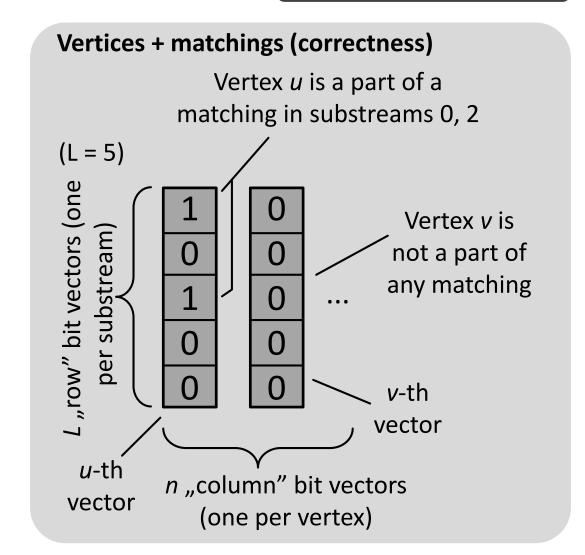


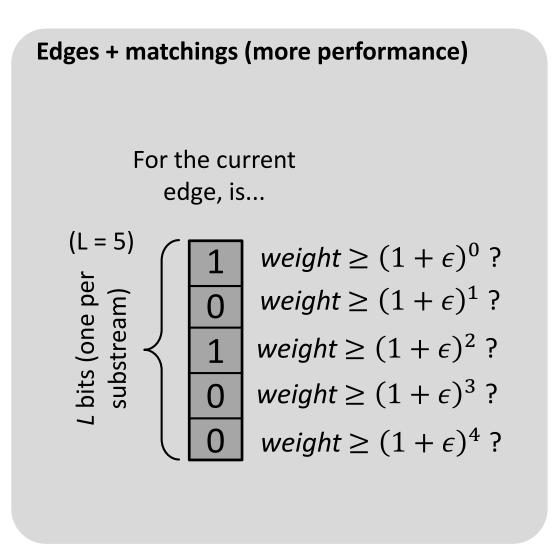


















PERFORMANCE ANALYSIS VARIOUS #SUBSTREAMS (L)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

Blocking size (K) = 32, #threads = 4, #edges = 16M (Kronecker), ε = 0.1

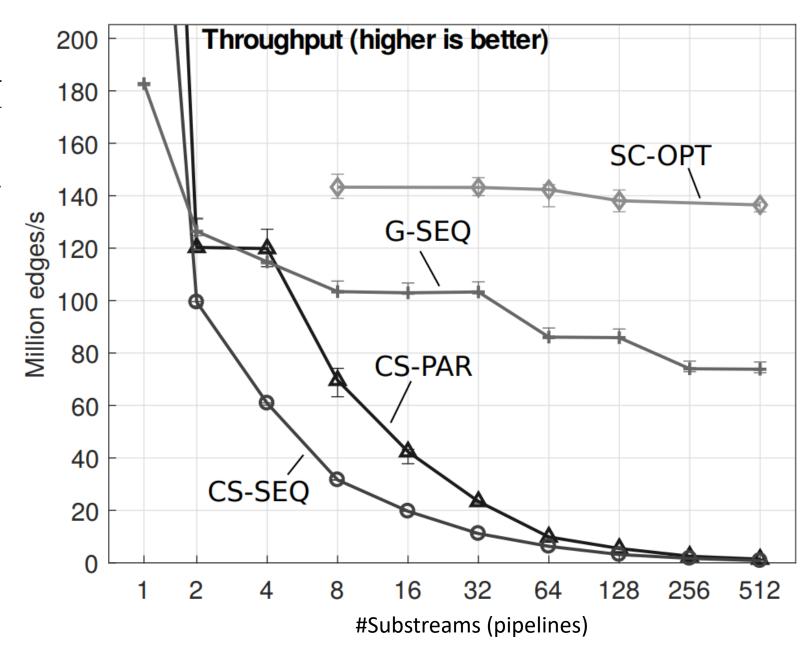


PERFORMANCE ANALYSIS VARIOUS #SUBSTREAMS (L)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

Blocking size (K) = 32, #threads = 4, #edges = 16M (Kronecker), ε = 0.1



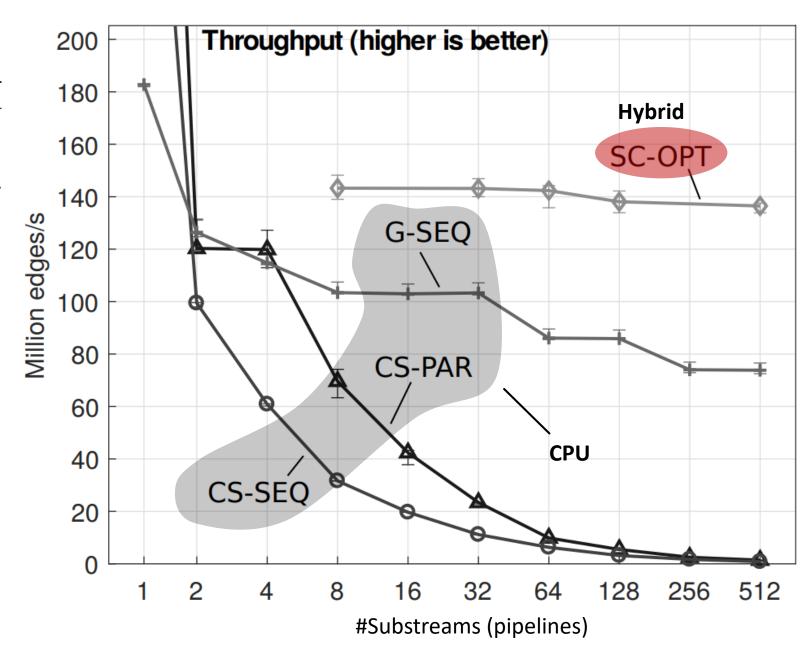


PERFORMANCE ANALYSIS VARIOUS #SUBSTREAMS (L)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

Blocking size (K) = 32, #threads = 4, #edges = 16M (Kronecker), ε = 0.1





PERFORMANCE ANALYSIS VARIOUS #SUBSTREAMS (L)

Algorithm	Platform
Crouch et al. [1] Sequential (CS-SEQ)	CPU
Crouch et al. [1] Parallel (CS-PAR)	CPU
Ghaffari [2] Sequential (G-SEQ)	CPU
Substream-Centric (SC-OPT)	Hybrid

Parameters:

Blocking size (K) = 32, #threads = 4, #edges = 16M (Kronecker), ε = 0.1

SC-OPT secures
highest performance
for all considered
values of parameters

