Mosaic: Processing a Trillion-Edge Graph on a Single Machine

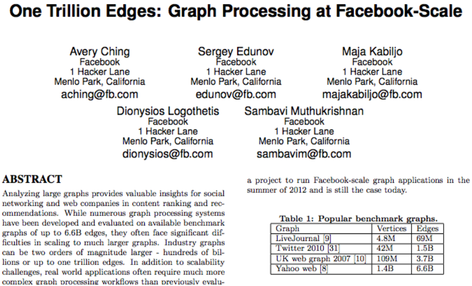
Steffen Maass, Changwoo Min, Sanidhya Kashyap, Woonhak Kang, Mohan Kumar, Taesoo Kim

Georgia Institute of Technology

Best Student Paper @ EuroSys'17

June 8, 2017

Large-scale graph processing is ubiquitous



One Trillion Edges: Graph Processing at Facebook-Scale

Avery Ching
Facebook
1 Hacker Lane
Menlo Park, California
aching@fb.com

Sergey Edunov
Facebook
1 Hacker Lane
Menlo Park, California
edunov@fb.com

Maja Kabiljo
Facebook
1 Hacker Lane
Menlo Park, California
majakabiljo@fb.com

Dionysios Logothetis
Facebook
1 Hacker Lane
Menlo Park, California
dionysios@fb.com

Sambavi Muthukrishnan
Facebook
1 Hacker Lane
Menlo Park, California
sambavim@fb.com

ABSTRACT

Analyzing large graphs provides valuable insights for social networking and web companies in content ranking and recommendations. While numerous graph processing systems have been developed and evaluated on available benchmark graphs of up to 6.6B edges, they often face significant difficulties in scaling to much larger graphs. Industry graphs can be two orders of magnitude larger - hundreds of billions or up to one trillion edges. In addition to scalability challenges, real world applications often require much more complex graph processing workflows than previously evalu-

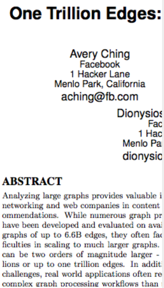
a project to run Facebook-scale graph applications in the summer of 2012 and is still the case today.

Table 1: Popular benchmark graphs.

Graph	Vertices	Edges
LiveJournal [9]	4.8M	69M
Twitter 2010 [31]	42M	1.5B
UK web graph 2007 [10]	109M	3.7B
Yahoo web [8]	1.4B	6.6B

Social networks

Large-scale graph processing is ubiquitous


One Trillion Edges:

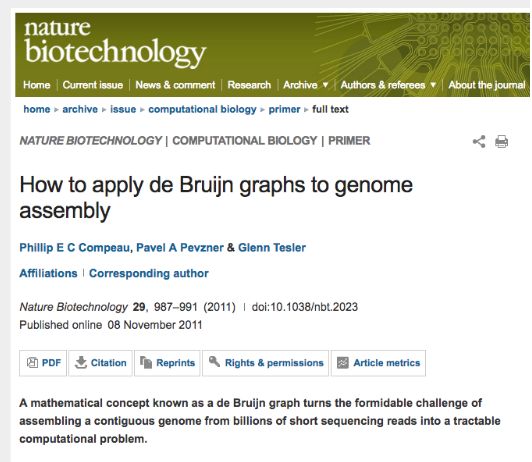
Avery Ching
Facebook
1 Hacker Lane
Menlo Park, California
aching@fb.com

Dionysios
Fac
1 Hac
Menlo Pa
dionysio

ABSTRACT

Analyzing large graphs provides valuable i
networking and web companies in content
ommendations. While numerous graph pr
have been developed and evaluated on avai
graphs of up to 6.6B edges, they often fac
ficulties in scaling to much larger graphs.
can be two orders of magnitude larger -
lions or up to one trillion edges. In addit
challenges, real world applications often re
complex graph processing workflows than

Social networks


nature biotechnology

Home | Current issue | News & comment | Research | Archive ▼ | Authors & referees ▼ | About the journal

home ▸ archive ▸ issue ▸ computational biology ▸ primer ▸ full text

NATURE BIOTECHNOLOGY | COMPUTATIONAL BIOLOGY | PRIMER

How to apply de Bruijn graphs to genome assembly

Phillip E C Compeau, Pavel A Pevzner & Glenn Tesler

Affiliations | Corresponding author

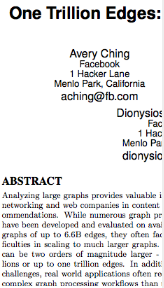
Nature Biotechnology **29**, 987–991 (2011) | doi:10.1038/nbt.2023
Published online 08 November 2011

PDF | Citation | Reprints | Rights & permissions | Article metrics

A mathematical concept known as a de Bruijn graph turns the formidable challenge of assembling a contiguous genome from billions of short sequencing reads into a tractable computational problem.

Genome analysis

Large-scale graph processing is ubiquitous



One Trillion Edges:

Avery Ching
Facebook
1 Hacker Lane
Menlo Park, California
aching@fb.com

Dionysios
Fac
1 Hac
Menlo Pa
dionysio

ABSTRACT

Analyzing large graphs provides valuable i
networking and web companies in content
ommendations. While numerous graph pr
have been developed and evaluated on avai
graphs of up to 6.6B edges, they often fac
ficulties in scaling to much larger graphs.
can be two orders of magnitude larger -
lions or up to one trillion edges. In addit
challenges, real world applications often re
complex graph processing workflows than

Social networks

NATURE BIOTECHNOLOGY | CO

How to apply de B
assembly

Phillip E C Compeau, Pavel A P

Affiliations | Corresponding aut

Nature Biotechnology **29**, 987–9
Published online 08 November 20

PDF | Citation | Reprint

**A mathematical concept known
assembling a contiguous genor
computational problem.**

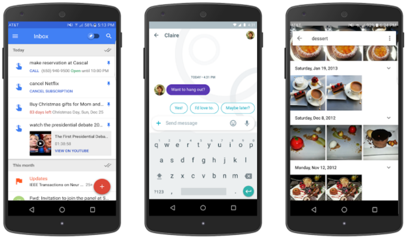
Genome analysi

Graph-powered Machine Learning at Google

Thursday, October 06, 2016

Posted by Sujith Ravi, Staff Research Scientist, Google Research

Recently, there have been significant advances in Machine Learning that enable computer systems to solve complex real-world problems. One of those advances is Google's large scale, graph-based machine learning platform, built by the Expander team in Google Research. A technology that is behind many of the Google products and features you may use everyday, graph-based machine learning is a powerful tool that can be used to power useful features such as reminders in Inbox and smart messaging in Allo, or used in conjunction with deep neural networks to power the latest image recognition system in Google Photos.



Learning with Minimal Supervision

Graphs enable Machine Learning

sgi The Trusted Leader in High Performance Computing

Products | Solutions | Services | Partners | About SGI

More sockets. More memory. More SAP HANA.

by Cori Pasinetti on July 29, 2015

Tweet 0 | Share 0 | Share 1 | +1 2

SGI UV 300H 20-Socket Appliance Certified by SAP to Run SAP HANA® Under Controlled Availability
Announcing the first 20-socket SAP HANA-certified in-memory server!

SGI announced today that the SGI® UV™ 300H is now SAP®-certified to run the SAP HANA® platform in controlled availability at 20-sockets–delivering up to 15 terabytes (TB) of in-memory computing capacity in a single node. Asserting the value of key enhancements in support package stack 10 (SPS10) for SAP HANA and SAP's close collaboration with system providers, SGI UV 300H delivers outstanding single-node performance and simplicity for enterprises moving to SAP HANA to gain business breakthroughs.

SGI UV 300H is a specialized offering in the SGI® UV™ server line for in-memory computing that enables enterprises to further unlock value from information in real-time, boost innovation, and lower IT costs with SAP HANA. Featuring a highly differentiated single-node architecture, the system delivers significant performance advantages for businesses running SAP® Business Suite 4 SAP HANA (SAP S/4HANA) and complex analytics at extreme scale. The single-node simplicity also helps enterprises eliminate overhead associated with clustered environments, streamline high availability, and scale-up seamlessly as data volumes grow with near-linear performance.

Integrated with the recently announced SAP HANA SPS10, SGI UV 300H capitalizes on deep collaboration between SAP, Intel and SGI to optimize SAP HANA-based workloads on multicore NUMA (non-uniform memory access) systems. This enables enterprises to leverage single-node systems with very large memory capacity and

Terabytes of RAM on multiple sockets

Powerful, heterogeneous machines

sgi The Trusted Leader in High Performance Computing

Products Solutions

More sockets. More

by Cori Pasinetti on July 29, 2015

Tweet 0 Share 0

SGI UV 300H 20-Socket Applian
Announcing the first 20-socket

SGI announced today that the SGI
controlled availability at 20-sockets
single node. Asserting the value of
SAP's close collaboration with syste
simplicity for enterprises moving to

SGI UV 300H is a specialized offerir
server line for in-memory computin
enterprises to further unlock value
real-time, boost innovation, and lov
HANA. Featuring a highly differentia
architecture, the system delivers si
advantages for businesses running
4 SAP HANA (SAP S/4HANA) and cc
extreme scale. The single-node sim
enterprises eliminate overhead ass
environments, streamline high avai
seamlessly as data volumes grow w
performance.

Integrated with the recently annour
SPS10, SGI UV 300H capitalizes on
between SAP, Intel and SGI to opti
workloads on multicore NUMA (non
access) systems. This enables ente

Intel Unveils Plans for Knights Mill, a Xeon Phi for Deep Learning

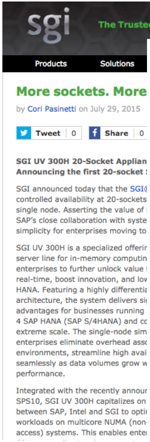
Michael Feldman | August 18, 2016 01:33 CEST

At the Intel Developer Forum (IDF) this week in San Francisco, Intel revealed it is working on a new Xeon Phi processor aimed at deep learning applications. Diane Bryant, executive VP and GM of Intel's Data Center Group, unveiled the new chip, known as Knights Mill, during her IDF keynote address on Wednesday.

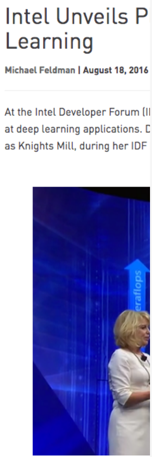
Terabytes of sockets

Powerful many-core coprocessors

Powerful, heterogeneous machines



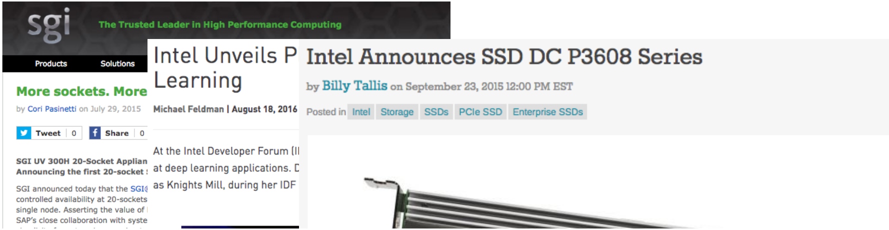
Terabytes of sockets



Powerful ma

Fast, large-capacity Non-volatile Memory

Powerful, heterogeneous machines


sgi The Trusted Leader in High Performance Computing

Products Solutions

More sockets. More

by Cori Pasinetti on July 29, 2015

Tweet 0 Share 0

SGI UV 300H 20-Socket Applian
Announcing the first 20-socket

SGI announced today that the SGI
controlled availability at 20-sockets
single node. Asserting the value of
SAP's close collaboration with syste

Intel Unveils P
Learning

Michael Feldman | August 18, 2016

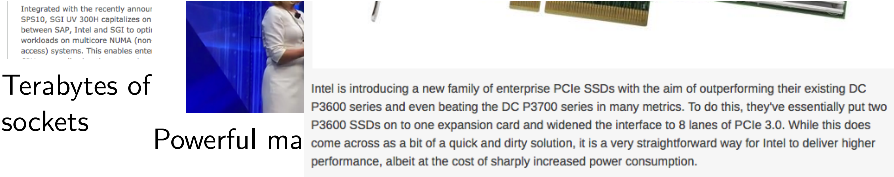
At the Intel Developer Forum (I
at deep learning applications. D
as Knights Mill, during her IDF

Intel Announces SSD DC P3608 Series

by Billy Tallis on September 23, 2015 12:00 PM EST

Posted in Intel Storage SSDs PCIe SSD Enterprise SSDs

Take advantage of heterogeneous machine to process tera-scale graphs


Integrated with the recently annour
SPS10, SGI UV 300H capitalizes on
between SAP, Intel and SGI to opti
workloads on multicore NUMA (non
access) systems. This enables ente

Intel is introducing a new family of enterprise PCIe SSDs with the aim of outperforming their existing DC P3600 series and even beating the DC P3700 series in many metrics. To do this, they've essentially put two P3600 SSDs on to one expansion card and widened the interface to 8 lanes of PCIe 3.0. While this does come across as a bit of a quick and dirty solution, it is a very straightforward way for Intel to deliver higher performance, albeit at the cost of sharply increased power consumption.

Terabytes of sockets

Powerful ma

Fast, large-capacity Non-volatile Memory

Table of contents

Graph Processing: Applications

- Community Detection
- Find Common Friends
- Find Shortest Paths
- Estimate Impact of Vertices (webpages, users, . . .)
- . . .

Example: Pagerank

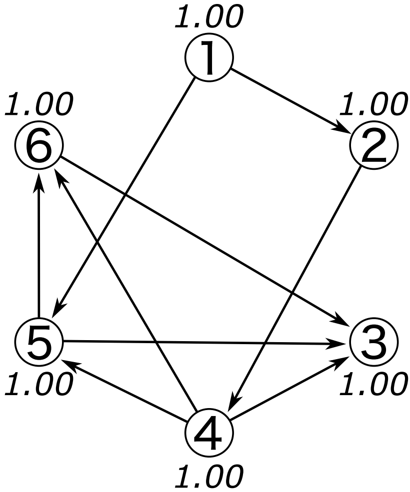
- Calculate impact of each vertex

$$Pagerank_v = \alpha * \left(\sum_{u \in Neighborhood(v)} \frac{Pagerank_u}{degree_u} \right) + (1 - \alpha)$$

- Simple Algorithm:
  - In each iteration, distribute current impact along out-edges, weighted by degree
  - Sum up all incoming impacts $\Rightarrow$ new impact for next iteration
  - Weight new impact with regularization factor $\alpha = 0.85$
  - Repeat until no changes

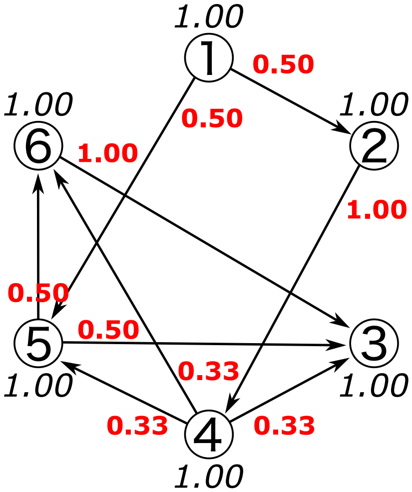
Pagerank: Graphical example

1) Initialize



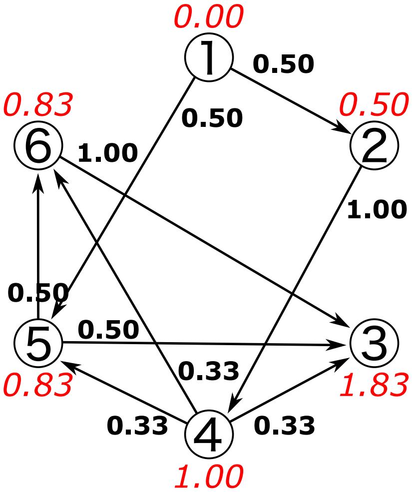
Pagerank: Graphical example

2) Propagate along outgoing edges



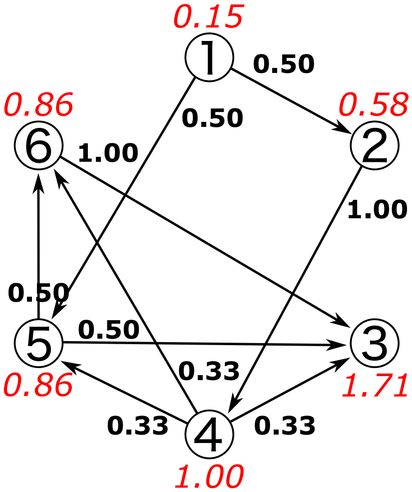
Pagerank: Graphical example

3) Sum up incoming contributions



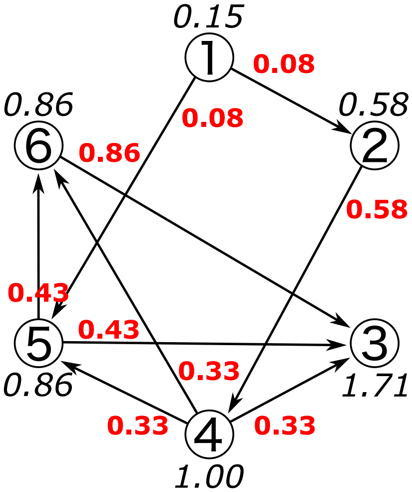
Pagerank: Graphical example

4) Apply regularization: $x * 0.85 + 0.15$



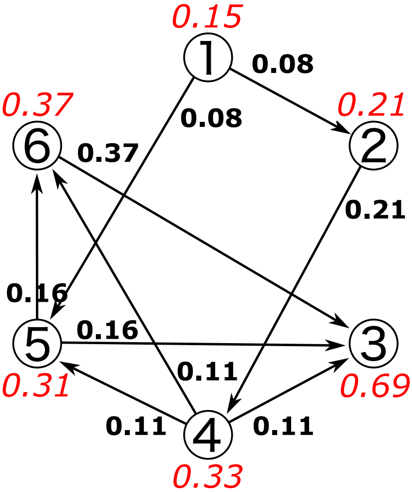
Pagerank: Graphical example

5) Update outgoing edges for second iteration



Pagerank: Graphical example

6) Repeat until stabilized



Mosaic: Design space

Graph Processing has many faces:

- Single Machine
  - Out-of-core
  - In memory
- Cluster
  - Out-of-core
  - In memory

Mosaic: Design space

Graph Processing has many faces:

- Single Machine
  - Out-of-core $\Rightarrow$ Cheap, but potentially slow
  - In memory $\Rightarrow$ Fast, but limited graph size
- Cluster
  - Out-of-core $\Rightarrow$ Large graphs, but expensive & slow
  - In memory $\Rightarrow$ Large graphs & fast, but *very* expensive

Mosaic: Design space

Graph Processing has many faces:

- Single Machine
  - Out-of-core $\Rightarrow$ Cheap, but potentially slow
  - In memory $\Rightarrow$ Fast, but limited graph size
- Cluster
  - Out-of-core $\Rightarrow$ Large graphs, but expensive & slow
  - In memory $\Rightarrow$ Large graphs & fast, but *very* expensive

$\Rightarrow$ Single machine, out-of-core is most cost-effective

$\Rightarrow$ Goal: Good performance and large graphs!

Mosaic: Design goals

Goal

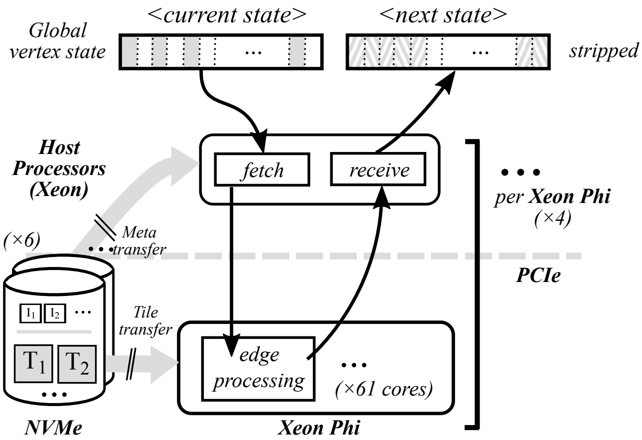
Run *algorithms* on *very large graphs* on *a single machine* using *coprocessors*

Enabled by:

- Common, familiar API (vertex/edge-centric)
- Encoding: Lossless compression
- Cache locality
- Processing on isolated subgraphs

Architecture of Mosaic

- Usage of *Xeon Phi & NVMe*
- Involvement of *Host*



Graph encoding: Idea

Compression

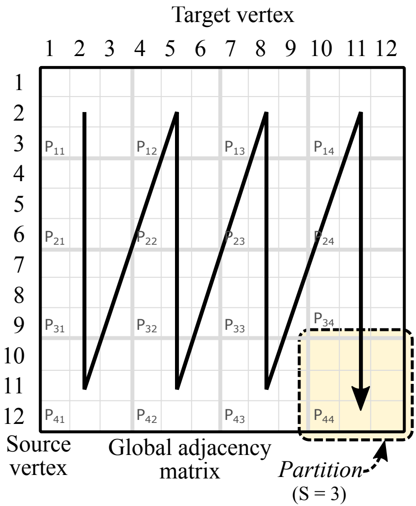
Split graph into subgraphs, use local (short) identifiers

Cache locality

- Inside subgraphs: Sort by access order
- Between subgraphs: Overlap vertex sets

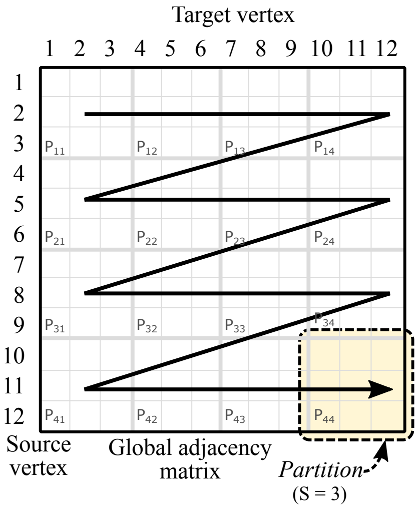
Background: Column first

- Locality for *write*
- Multiple sequential *reads*



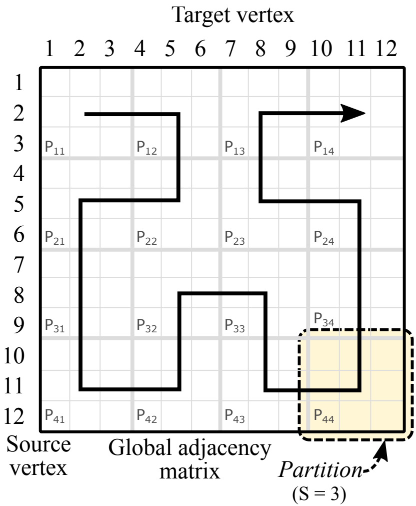
Background: Row first

- Locality for *read*
- Multiple sequential *writes*



Background: Hilbert order

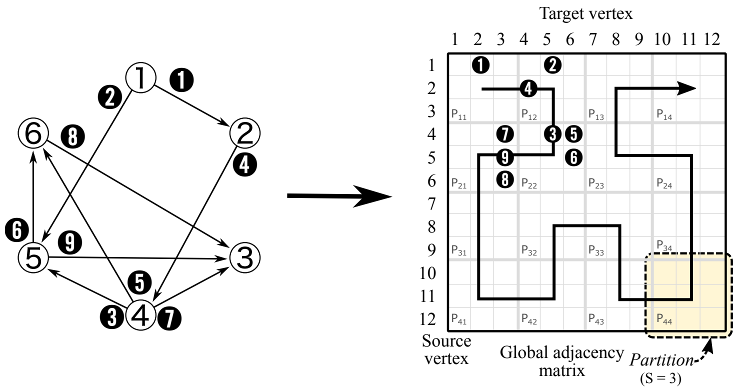
- Space-filling curve
- Provides locality between adjacent data points



From global to local: Tiles

- Convert graph to set of *tiles*

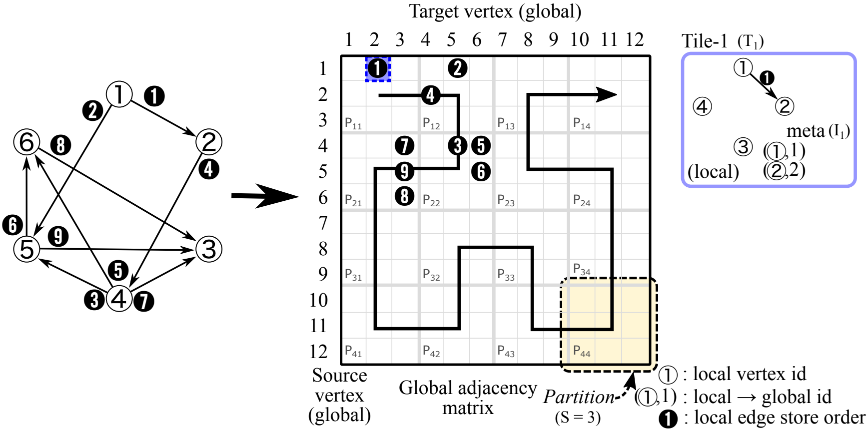
1) Start with adjacency Matrix:



From global to local: Tiles

- Convert graph to set of *tiles*

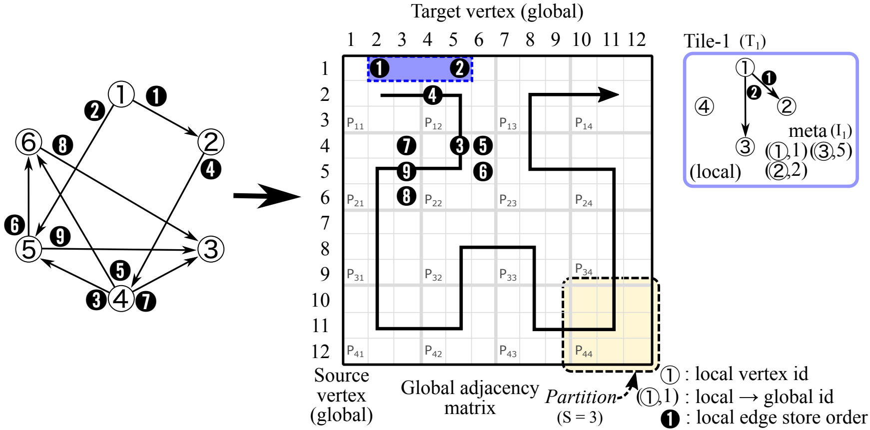
2) Use first edge in tile T_1:



From global to local: Tiles

- Convert graph to set of *tiles*

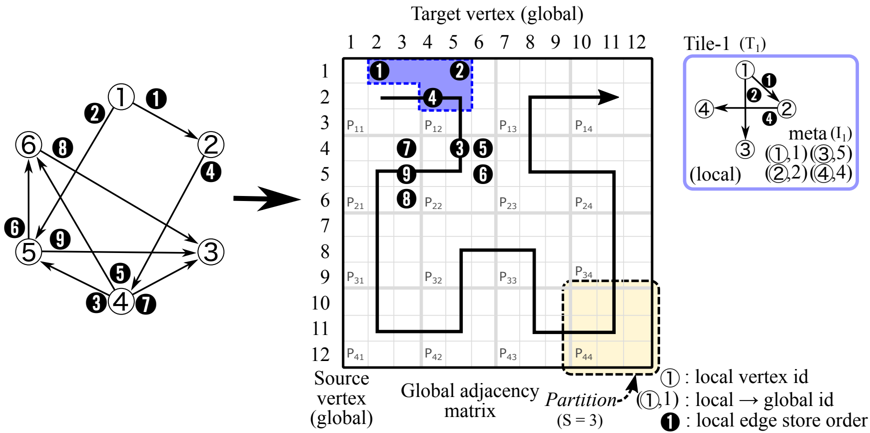
3) Use more edges in tile T_1:



From global to local: Tiles

- Convert graph to set of *tiles*

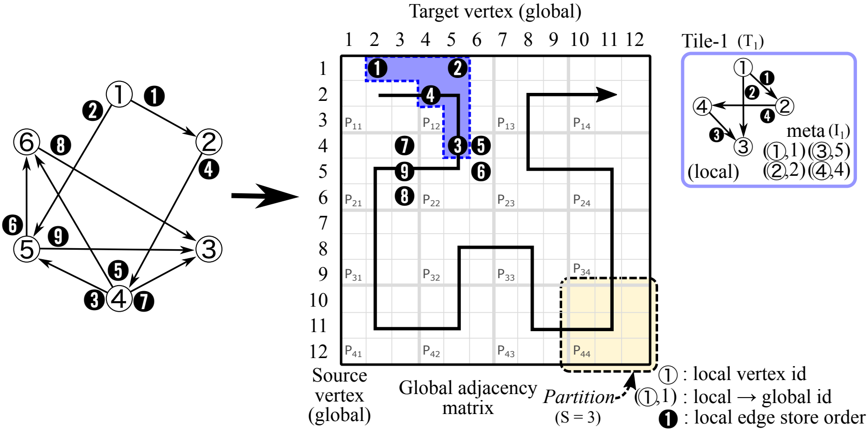
4) Use more edges in tile T_1:



From global to local: Tiles

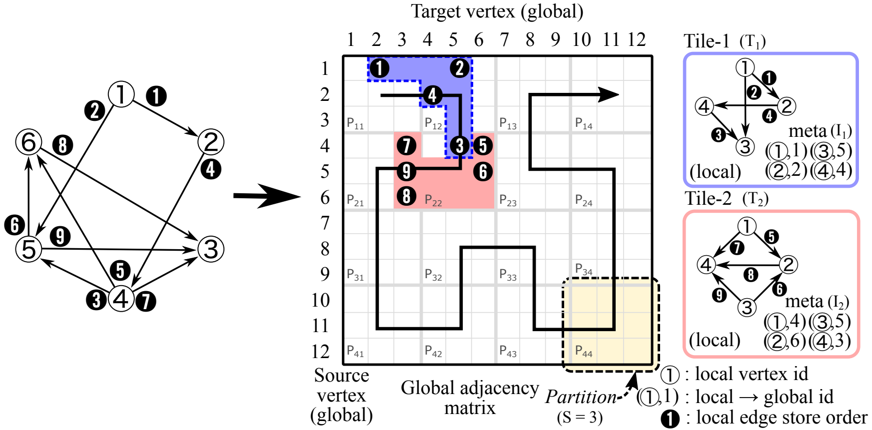
- Convert graph to set of *tiles*

5) Use more edges in tile T_1:



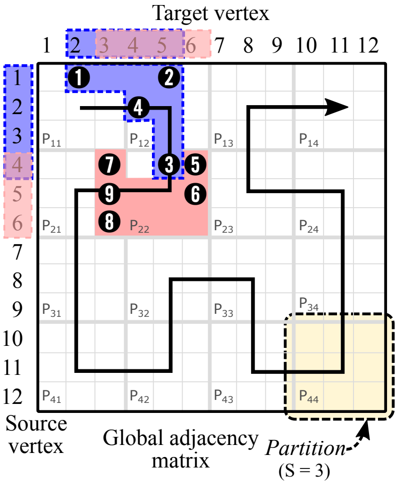
- Convert graph to set of *tiles*

6) Next edges do not fit in T_1 anymore, construct T_2:



Locality with Hilbert-ordered tiles

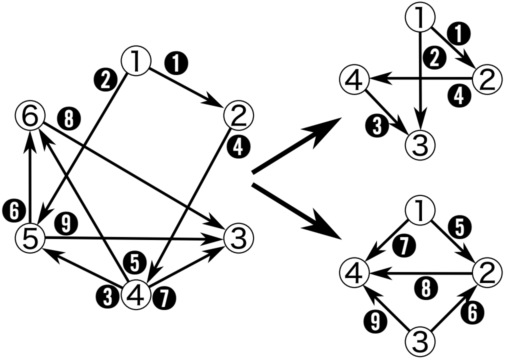
- Overlapping sets of *sources* and *targets*



⇒ Better locality than row-first or column-first

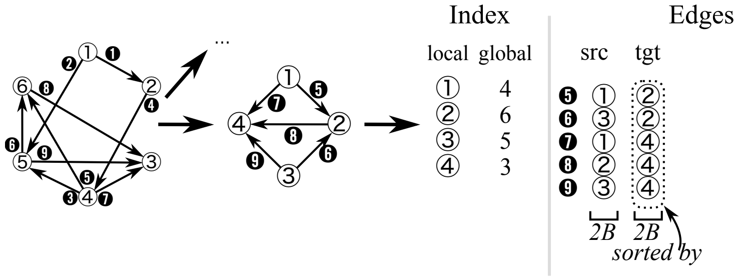
From global to local: Data structure

1) Split original graphs into two subgraphs:



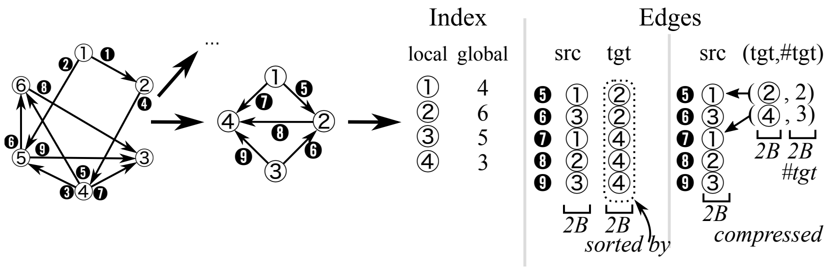
From global to local: Data structure

2) Internal data structure of T_2:



From global to local: Data structure

3) Compress edges: *Compressed sparse rows*



⇒ Efficient, local encoding, sequentially accessed

From global to local: Summary

- Better locality
- Efficient encoding of local graphs
- Effect: up to 68% reduction in data size:

Graph	#vertices	#edges	Raw data	Mosaic size (red.)
***rmat24**	16.8 M	0.3 B	2.0 GB	1.1 GB (−45.0%)
twitter	41.6 M	1.5 B	10.9 GB	7.7 GB (−29.4%)
***rmat27**	134.2 M	2.1 B	16.0 GB	11.1 GB (−30.6%)
uk2007-05	105.8 M	3.7 B	27.9 GB	8.7 GB (−68.8%)
hyperlink14	1,724.6 M	64.4 B	480.0 GB	152.4 GB (−68.3%)
***rmat-trillion**	4,294.9 M	1,000.0 B	8,000.0 GB	4,816.7 GB (−39.8%)

API: Pagerank example

- *Pull*: Gather per edge information
- *Reduce*: Combine results from multiple subgraphs
- *Apply*: Calculate non-associative regularization

Edge-centric operation

Local graph processing on Tile

```
// On edge processor (co-processor)
// Edge e = (Vertex src, Vertex tgt)
def Pull(Vertex src, Vertex tgt):
    return src.val / src.out_degree
// On edge processor/global reducers (both)
def Reduce(Vertex v1, Vertex v2):
    return v1.val + v2.val
// On global reducers (host)
def Apply(Vertex v):
    v.val = (1 - α) + α × v.val
```

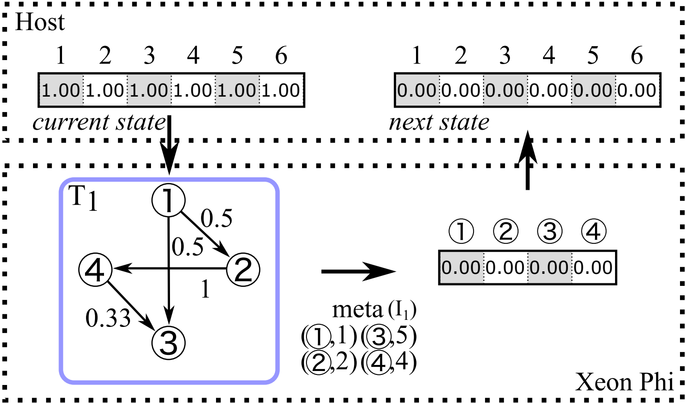
Global graph processing

Vertex-centric operation

Formula: $Pagerank_v = \alpha * \left(\sum_{u \in Neighborhood(v)} \frac{Pagerank_u}{degree_u} \right) + (1 - \alpha)$

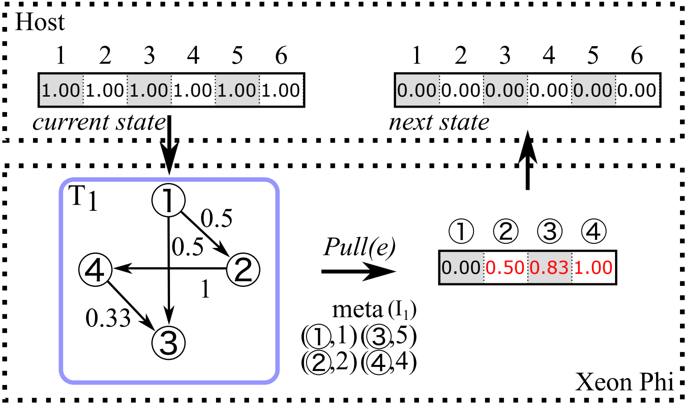
API: Pagerank example

1) Start with T_1



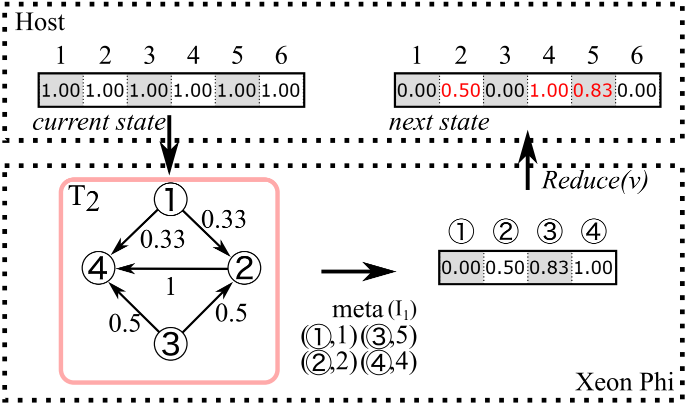
API: Pagerank example

2) Execute *Pull* along all edges in T_1



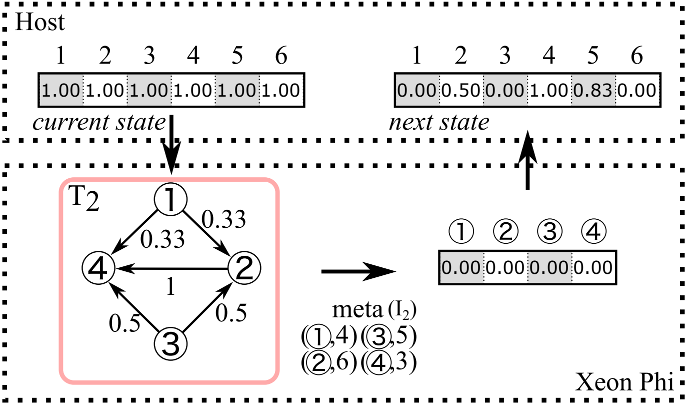
API: Pagerank example

3) *Reduce* all updates from T_1 onto *next state*



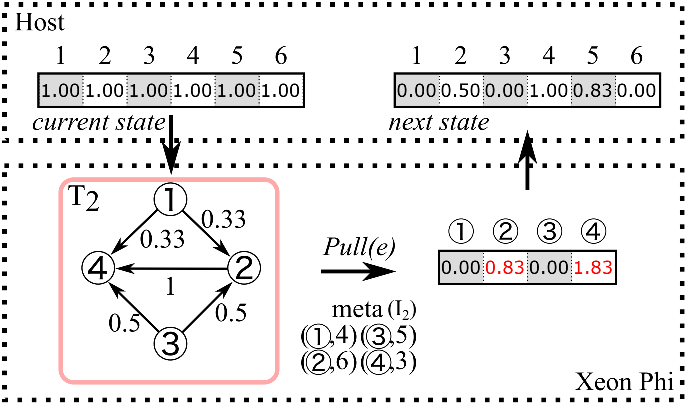
API: Pagerank example

4) Switch to T_2



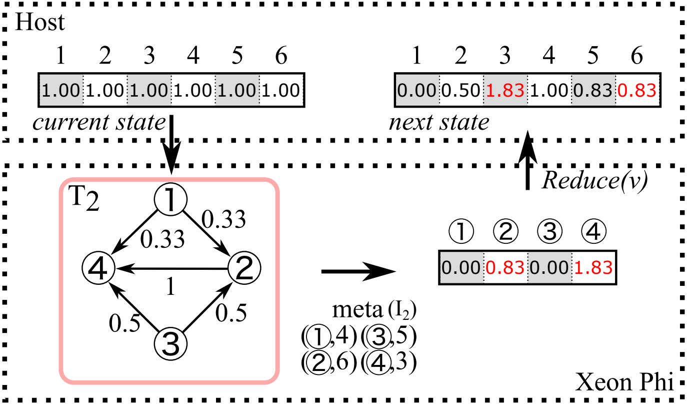
API: Pagerank example

5) *Pull* all updates



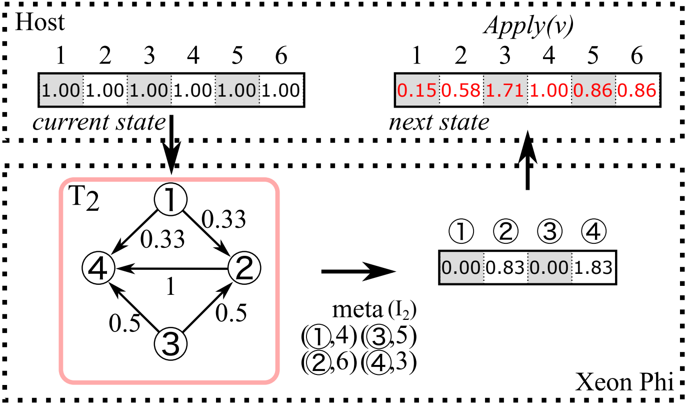
API: Pagerank example

6) *Reduce* updates from T_2

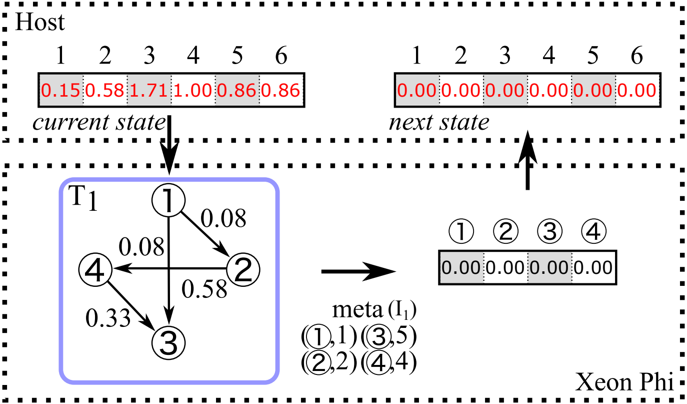


API: Pagerank example

7) All tiles processed, *Apply* processed updates



8) Switch *current* and *next* state, clear *next* state for second iteration



Evaluation

Questions:

- Preprocessing Cost
- Performance (in comparison)
- Impact of Design Decisions
- Scalability

Evaluation - Setup

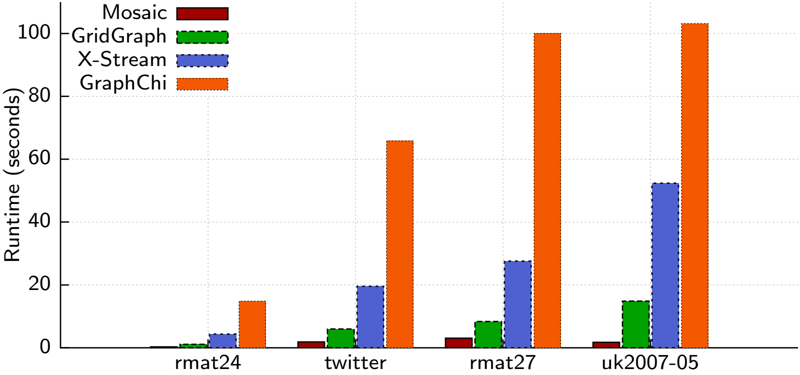
- Single Server:
  - 2 sockets, 12 cores each
  - 768GB RAM
  - 4 Xeon Phi (KNC, 61 cores)
  - 6 NVMe (1.2TB each)
- 7 Algorithms
- 6 Datasets (3 real world, 3 synthetic)

Preprocessing

- Mosaic needs explicit preprocessing step
- 2-4 min for small datasets, 51 minutes for webgraph, 31 hours for trillion edges
- But: Can be amortized during execution:
  - GridGraph: Mosaic faster after
    - twitter: 20 iterations
    - uk2007: 8 iterations
  - X-Stream: Mosaic faster after
    - twitter: 8 iterations
    - uk2007: 5 iterations

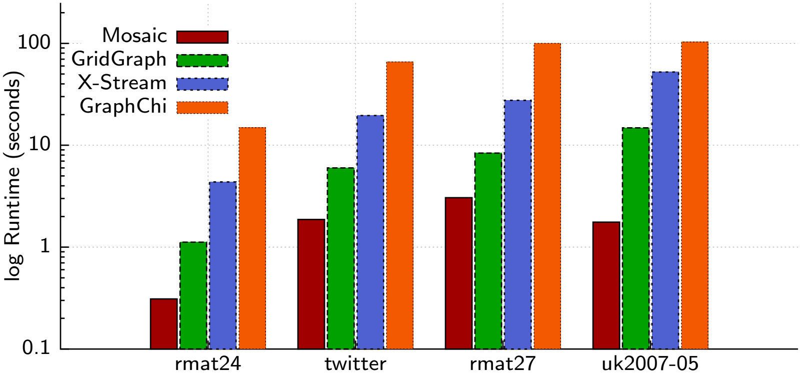
Performance comparison

- Comparison to other single machine engines with Pagerank:



Performance comparison

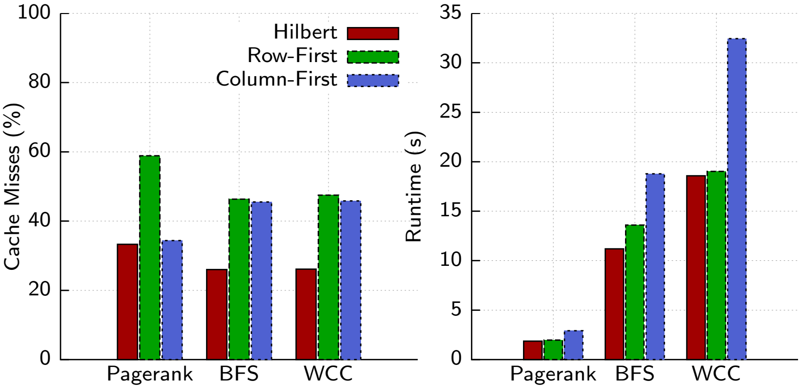
- Comparison to other single machine engines with Pagerank:



$\Rightarrow$ Mosaic outperforms other system by 2.7×to 58.6×

Hilbert-ordered tiles: Cache locality

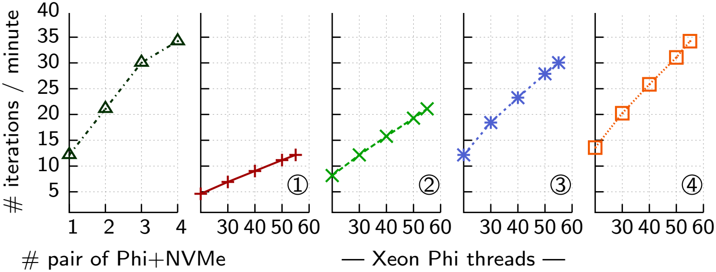
- Cache misses and execution times for three different strategies



$\Rightarrow$ Hilbert-ordered tiles have up to 45% better cache locality, up to 43% reduction in runtime

Evaluation - Scalability

- Dimensions
  - Add Xeon Phis/NVMes
  - Add threads on each Xeon Phi



⇒ Mosaic scales well when adding threads/Xeon Phis

Conclusion

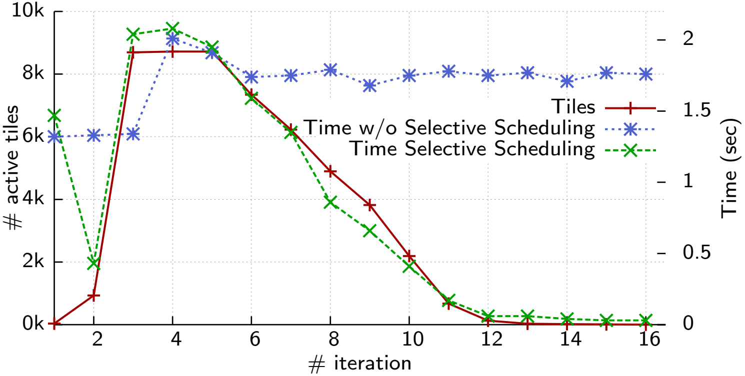
- Mosaic, a graph processing engine for trillion edge graphs *on a single machine*
- Hilbert-ordered tiles allow:
    - Enable localized processing on *coprocessors*
    - Optimizes *cache locality*
    - Enables *compression*

Code is open-source: `https://github.com/sslab-gatech/mosaic`

Thank you!

Evaluation - Selective scheduling

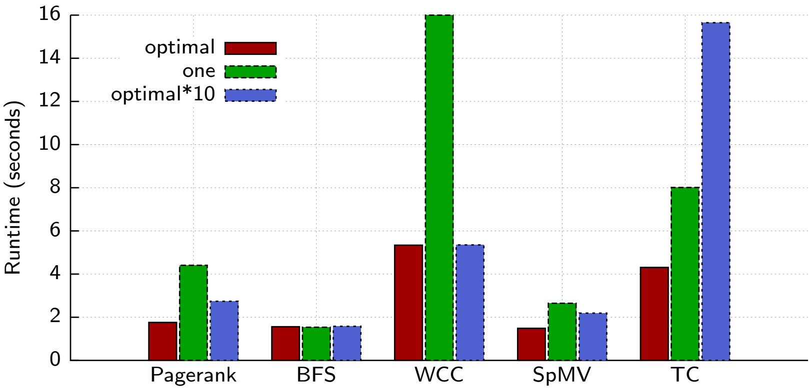
- Skip subgraphs without active vertices
- Especially useful for traversal algorithms: BFS, Connected Components, ...



⇒ 2.2×speedup for BFS on `twitter` graph

Evaluation - Dynamic load balancing

- Effect of choosing the wrong load balancing scheme
- Mosaic has light-weight balancing scheme



⇒ Up to 5.8×improvement in running time by choosing correct load balancing scheme

Mosaic architecture

- Host and Xeon Phi component
- Streaming-based design