CREST 戦略的創造研究推進事業
Core Research for Evolutional Science and Technology
JST Japan Science and Technology Agency

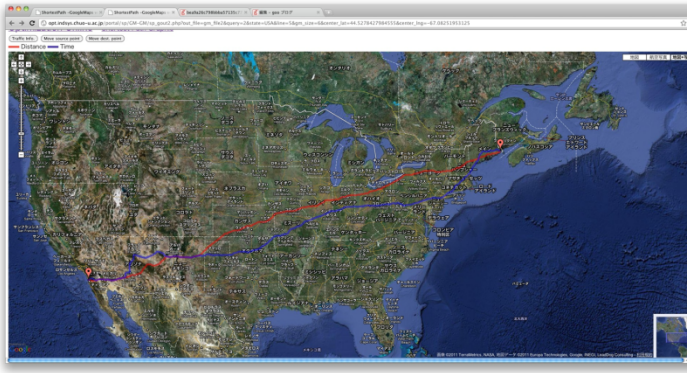
How to win Graph500
-- A Challenge to Graph500 Benchmark --

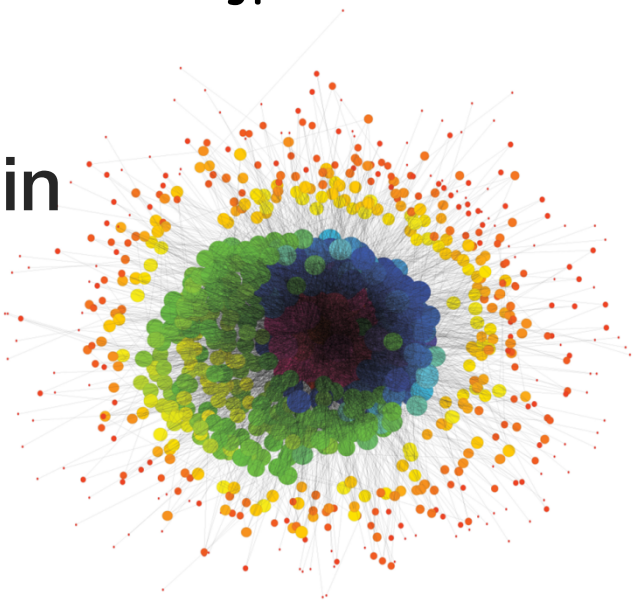
Katsuki Fujisawa

The Institute of Mathematics for Industry,
Kyushu University, Fukuoka, Japan
e-mail : fujisawa@imi.kyushu-u.ac.jp

October 2, 2015

Co@work, ZIB, Berlin



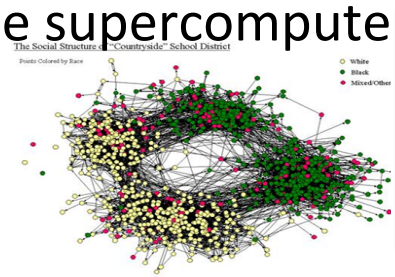


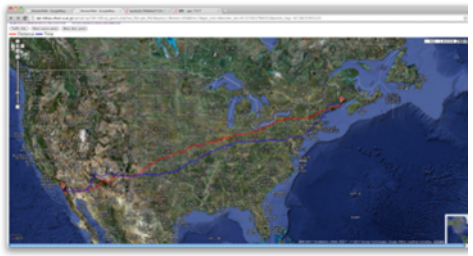
Contents

- Applications of Graph Analysis
- Graph500 and Green Graph500 benchmarks and our Achievements
- Tutorials (Using Software packages)
  - Graph500 (Optimized implementation)
  - Netal (NETwork Analysis Library)
- Future of Graph Analysis

Advanced Computing and Optimization Infrastructure for Extremely Large-Scale Graphs on Post Peta-Scale Supercomputers

- **JST**(Japan Science and Technology Agency) **CREST**(Core Research for Evolutionally Science and Technology) **Project** (Oct, 2011 ～ March, 2017)
- **Winner of 8^{th} and 10^{th} Graph 500 benchmarks, and 1^{st} ～ 5^{th} Green Graph 500 benchmarks**
- **3 groups, over 60 members**
  1. Fujisawa-G (Kyushu University) : Large-scale Mathematical Optimization
  2. Suzumura-G (University College Dublin, Ireland) : Large-scale Graph Processing
  3. Sato-G (Tokyo Institute of Technology) : Hierarchical Graph Store System
- **Innovative Algorithms and implementations**
  - Optimization, Searching, Clustering, Network flow, etc.
- Extreme Big Graph Data for emerging applications
  - **2^{30} ~ 2^{42} nodes** and **2^{40} ~ 2^{46} edges**
  - **Over 1M threads** are required for real-time analysis
- Many applications on post peta-scale supercompute
  - Cyber security and social networks
  - Optimizing smart grid networks
  - Health care and medical science

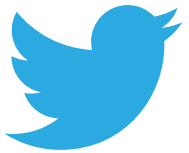




Background

- **The extremely large-scale graphs that have recently emerged in various application fields**
  - US Road network : 58 **million** edges
  - Twitter fellow-ship : 1.47 **billion** edges
  - Neuronal network : 100 **trillion** edges

Social network



Twitter

61.6 million nodes & 1.47 billion edges

- **Fast and scalable graph processing** by using **HPC**

Neuronal network @ Human Brain Project

89 billion nodes & 100 trillion edges

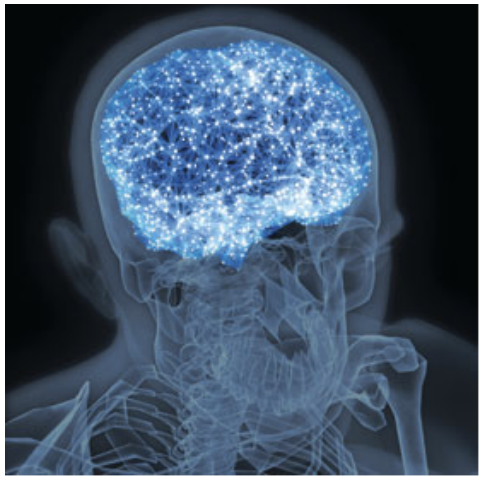
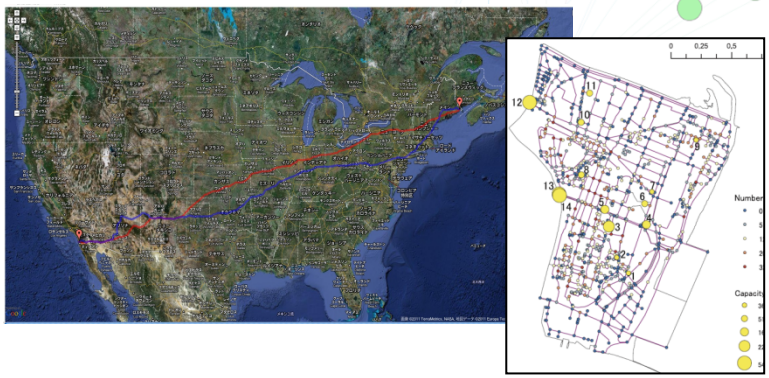


Image: Illustration by Mirko Ilic

US road network

24 million nodes & 58 million edges

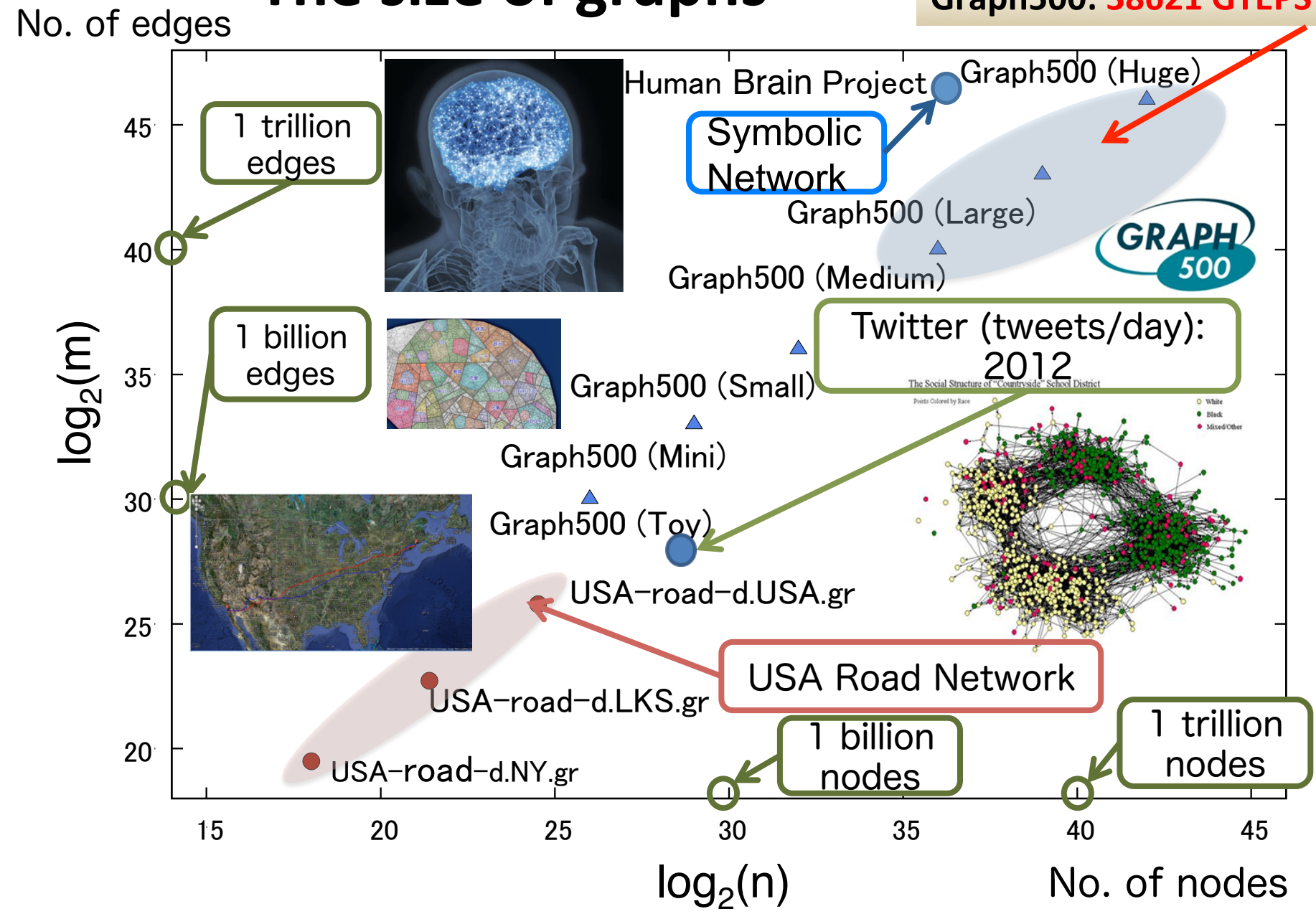


Cyber-security

15 billion log entries / day

The size of graphs

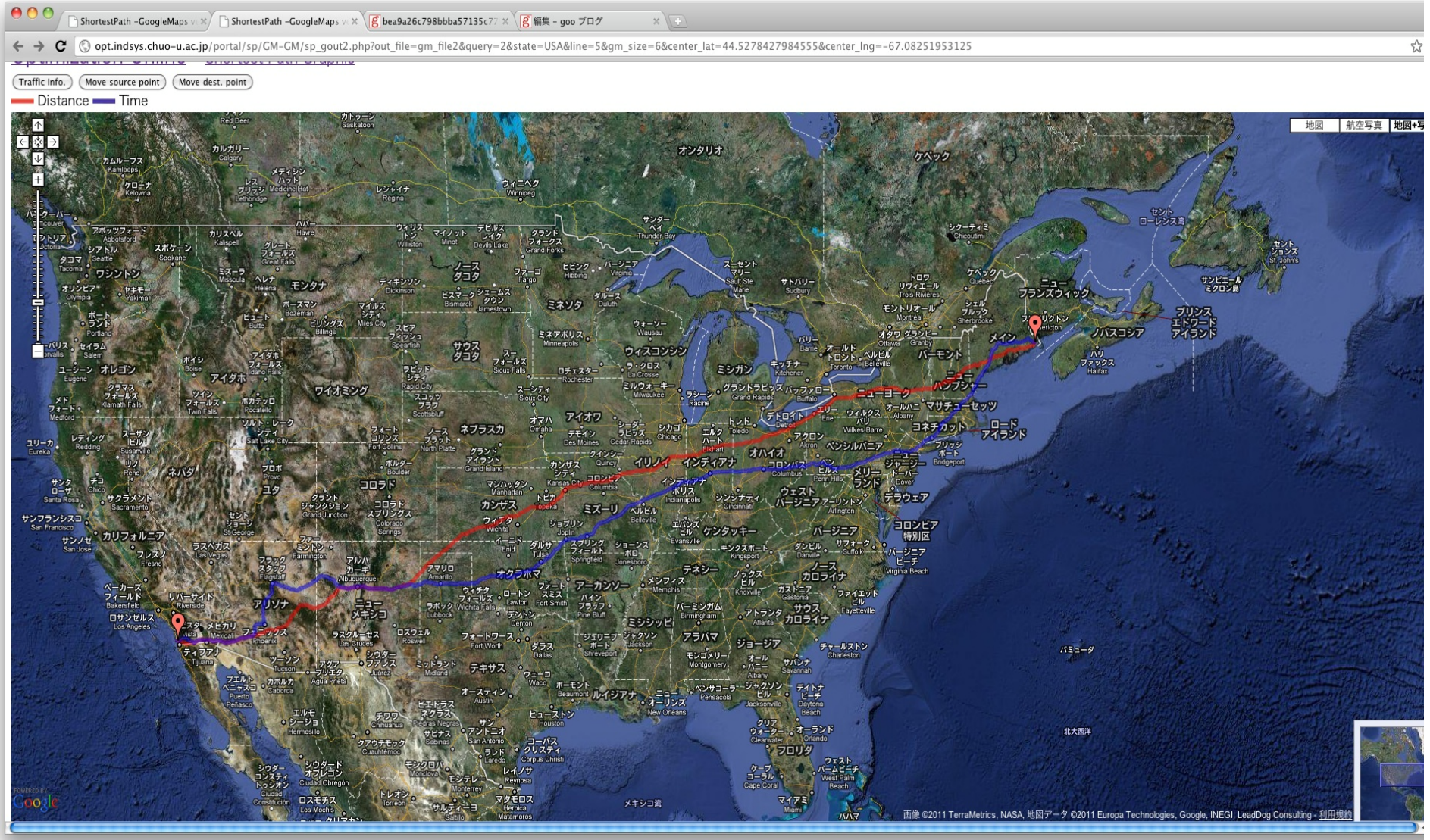
K computer: 82944 nodes
Graph500: 38621 GTEPS



USA road network : **24 million vertices & 58 million edges**

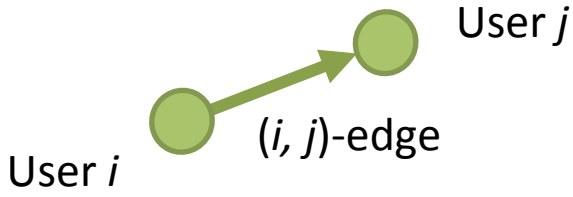
A shortest path from San Diego to Augusta

(**Red** : distance、**Blue** : time) : **http://opt.imi.kyushu-u.ac.jp/**



Twitter network (Application of BFS)

Follow-ship network 2009



41 million vertices and 1.47 billion edges

Our NUMA-optimized BFS
on 4-way Xeon system

70 ms / BFS

⇒ 21.28 GTEPS

Six-degrees of separation

Frontier size in BFS

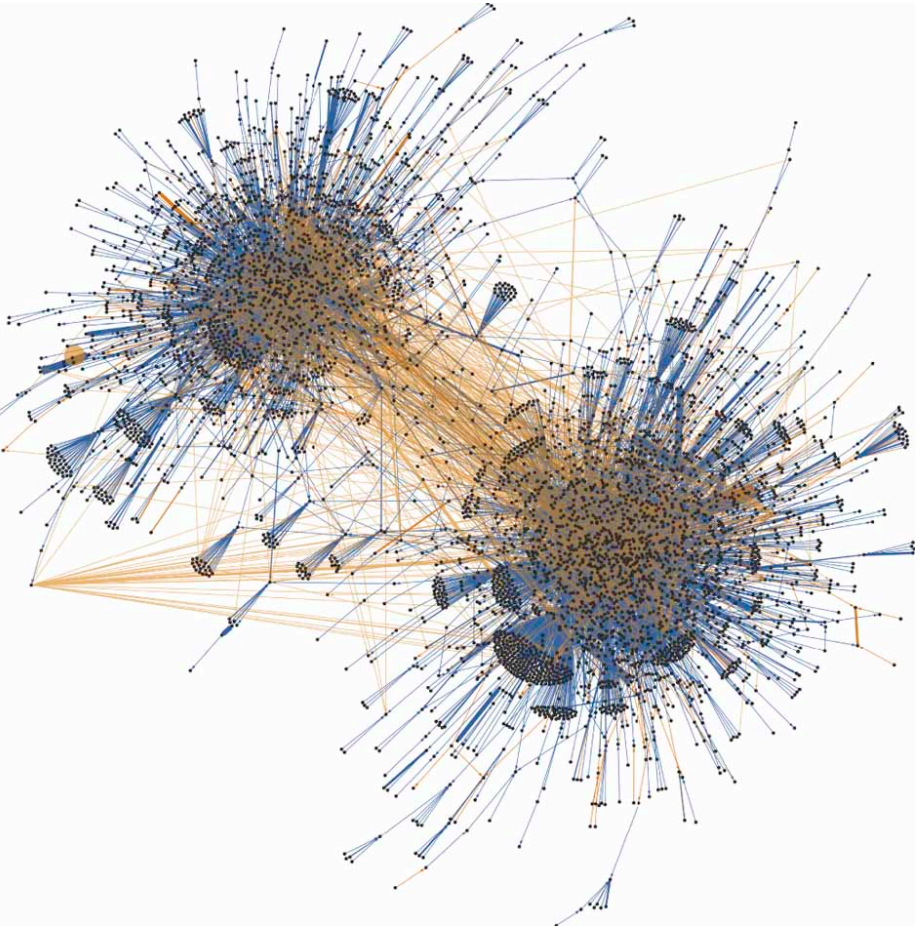
with source as User 21,804,357

Lv	Frontier size	Freq. (%)	Cum. Freq. (%)
0	1	0.00	0.00
1	7	0.00	0.00
2	6,188	0.01	0.01
3	510,515	1.23	1.24
4	29,526,508	70.89	72.13
5	11,314,238	27.16	99.29
6	282,456	0.68	99.97
7	11536	0.03	100.00
8	673	0.00	100.00
9	68	0.00	100.00
10	19	0.00	100.00
11	10	0.00	100.00
12	5	0.00	100.00
13	2	0.00	100.00
14	2	0.00	100.00
15	2	0.00	100.00
Total	41,652,230	100.00	-

Twitter network (Application of BFS)

41 million vertices and 1.47 billion edges

Six-degrees of separation



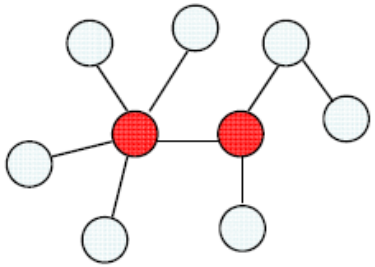
Frontier size in BFS

with source as User 21,804,357

Lv	Frontier size	Freq. (%)	Cum. Freq. (%)
0	1	0.00	0.00
1	7	0.00	0.00
2	6,188	0.01	0.01
3	510,515	1.23	1.24
4	29,526,508	70.89	72.13
5	11,314,238	27.16	99.29
6	282,456	0.68	99.97
7	11536	0.03	100.00
8	673	0.00	100.00
9	68	0.00	100.00
10	19	0.00	100.00
11	10	0.00	100.00
12	5	0.00	100.00
13	2	0.00	100.00
14	2	0.00	100.00
15	2	0.00	100.00
Total	41,652,230	100.00	-

Centrality : vertex measures its relative importance within a graph

$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$



- σ_{st} : Number of shortest paths between vertices s and *t*
- $\sigma_{st}(v)$: Number of shortest paths between vertices s and *t* passing through *v*

$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}$$ *closeness centrality* (Sabidussi, 1966)

$$C_G(v) = \frac{1}{\max_{t \in V} d_G(v,t)}$$ *graph centrality* (Hage and Harary, 1995)

$$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$$ *stress centrality* (Shimbel, 1953)

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$ *betweenness centrality* (Freeman, 1977; Anthonisse, 1971)

Betweenness centrality (BC)

- **Definition**

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

σ_{st} : # of (s,t)-shortest paths

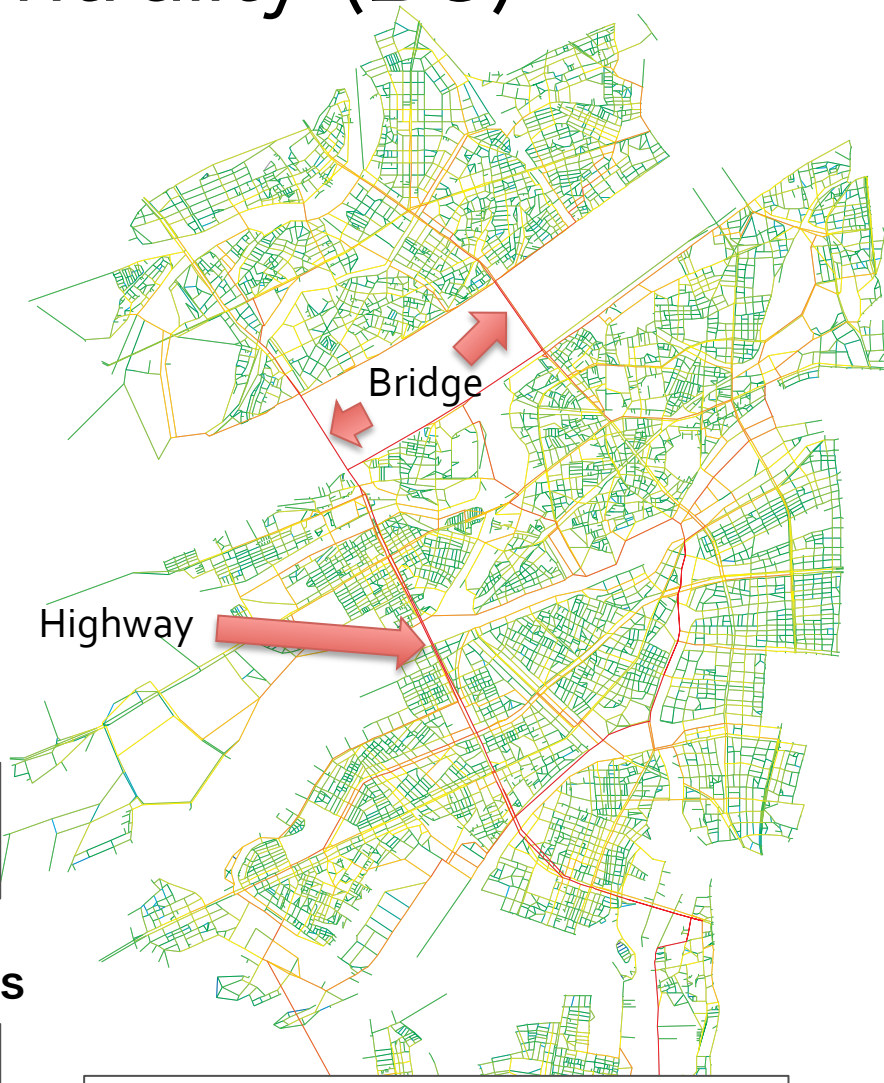
$\sigma_{st}(v)$: # of (s,t)-shortest paths passing throw v

- **BC measures important vertices and edges without coordinates**

High score vertex/edge = Important place
c.g.) Highway, Bridge

- **BC requires the all-to-all shortest paths**

- BFS => one-to-all
- <#vertices> times BFS => all-to-all



Osaka road network
13,076 vertices and 40,528 edges

=> **13,076** times BFS computations

Fukuoka road network(1.5 million people)

Betweenness centrality

Computation time
2m 30s (180 CPU cores)

HP ProLiant m710
Server cartridge

Graph

# of nodes:	314,571
# of edgs	694,906

Betweenness centrality

Tokyo Area in Japan
40 million people

Open Street Map
https://mapzen.com/
metro-extracts

Graph

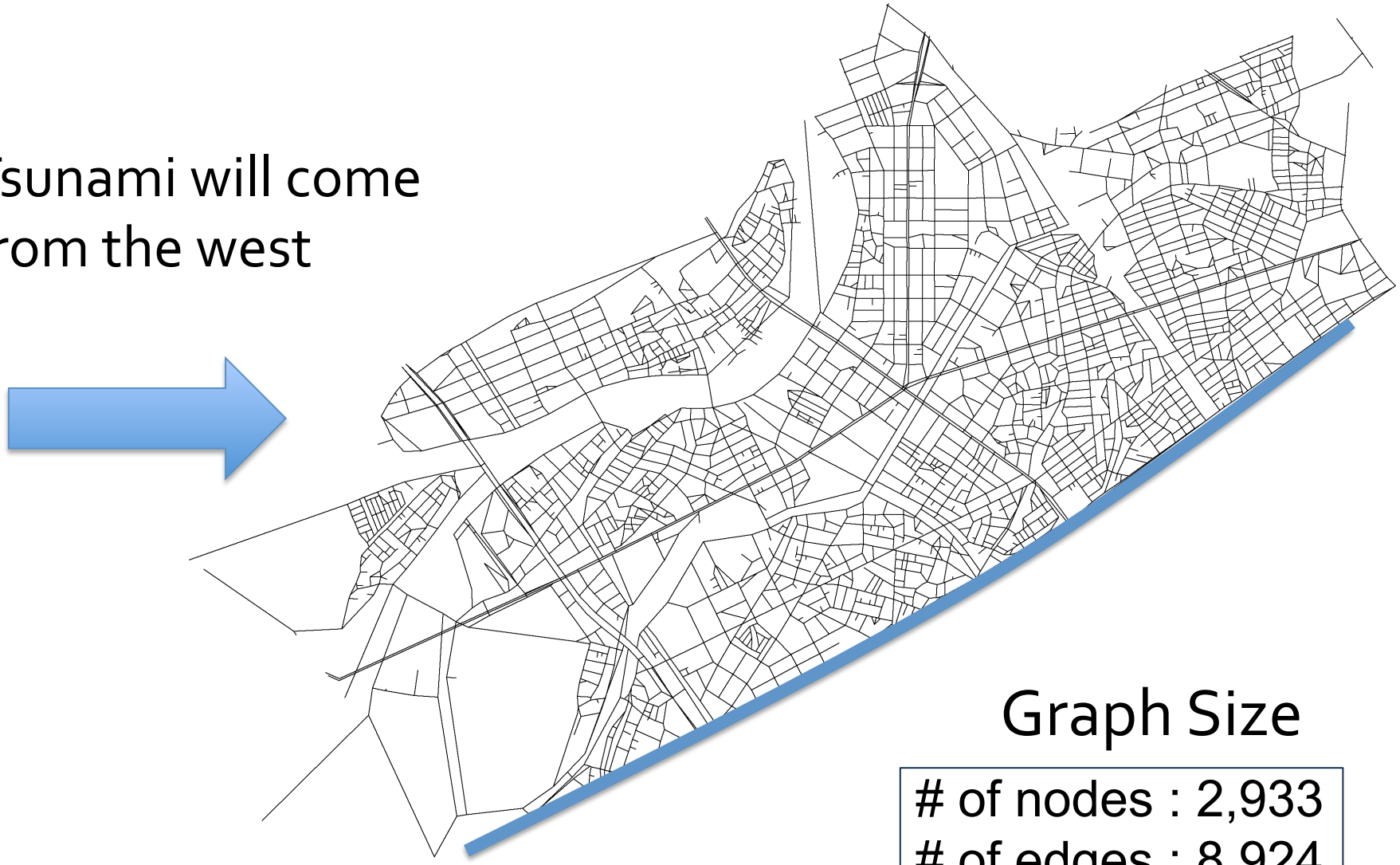
nodes 6,509,809
edges 14,460,834

Computation Time
98h 27m 37s

Huawei RH5885H V3
CPU ：Intel Xeon E7-4890 x 4
Memory ：2.0TB (32GB LRDIMM x 64 DIMMs)

North area of Osaka City, Japan

Tsunami will come from the west



Graph Size

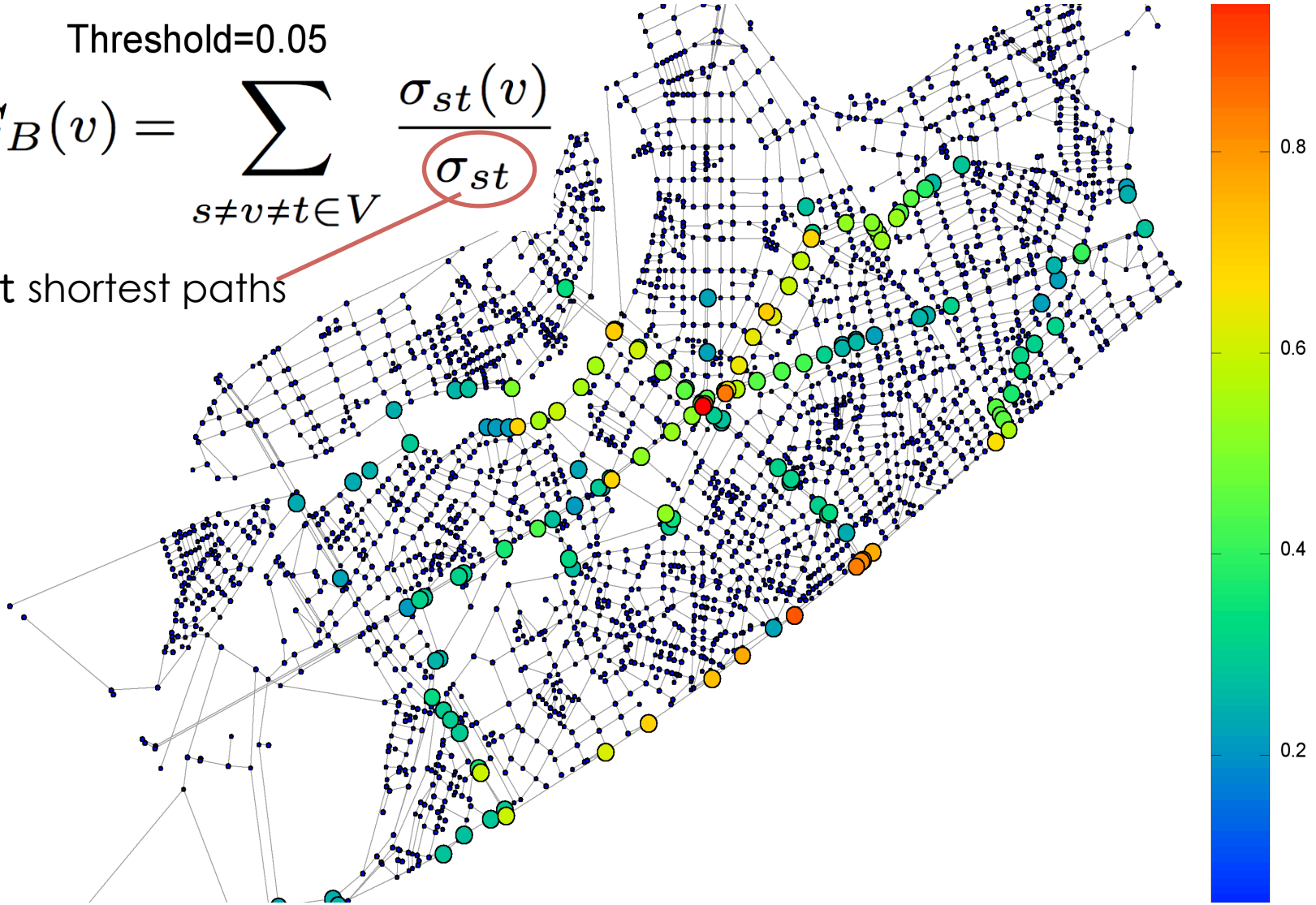
# of nodes : 2,933
# of edges : 8,924

Betweenness centrality; BC

Threshold=0.05

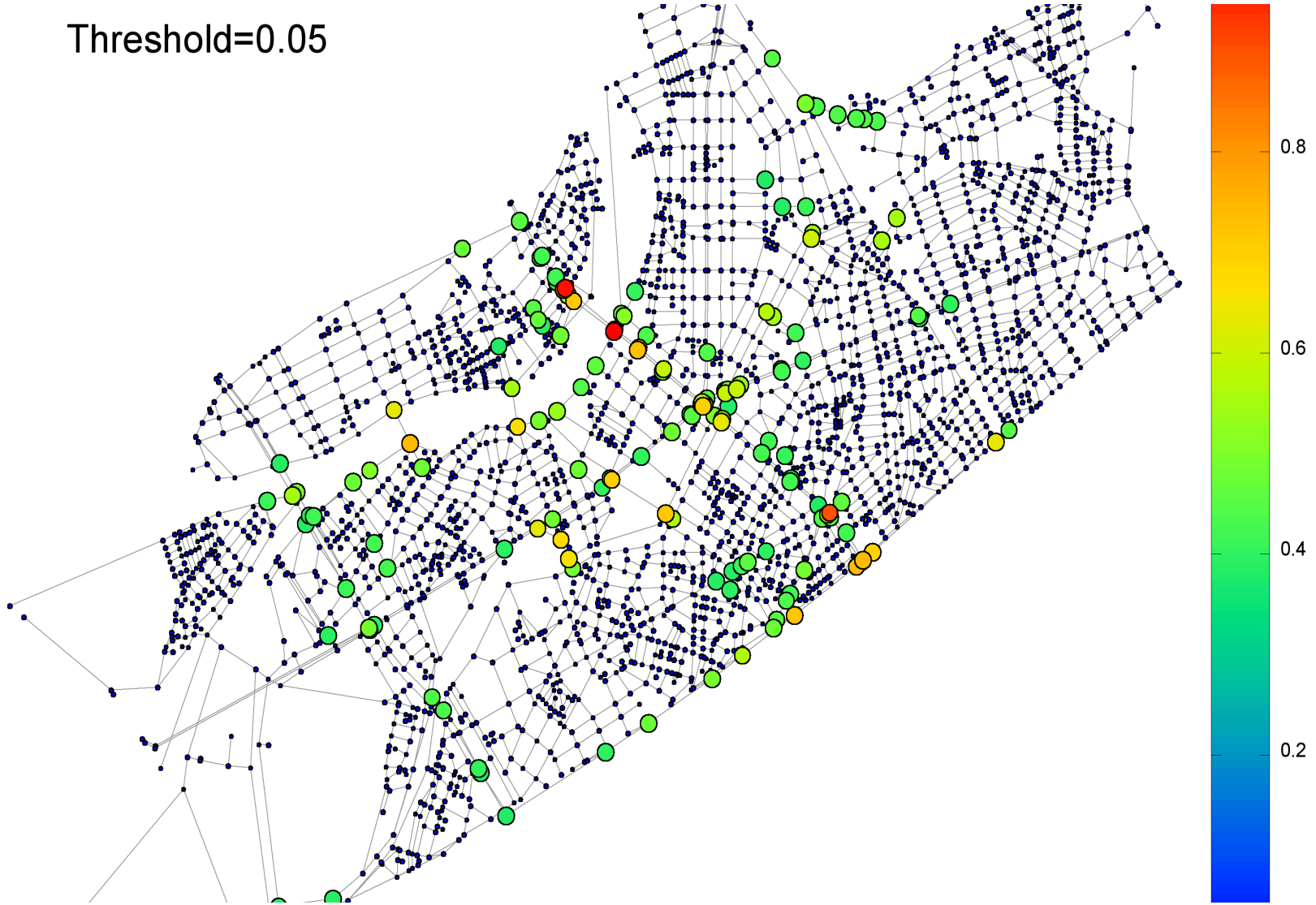
$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

s-t shortest paths



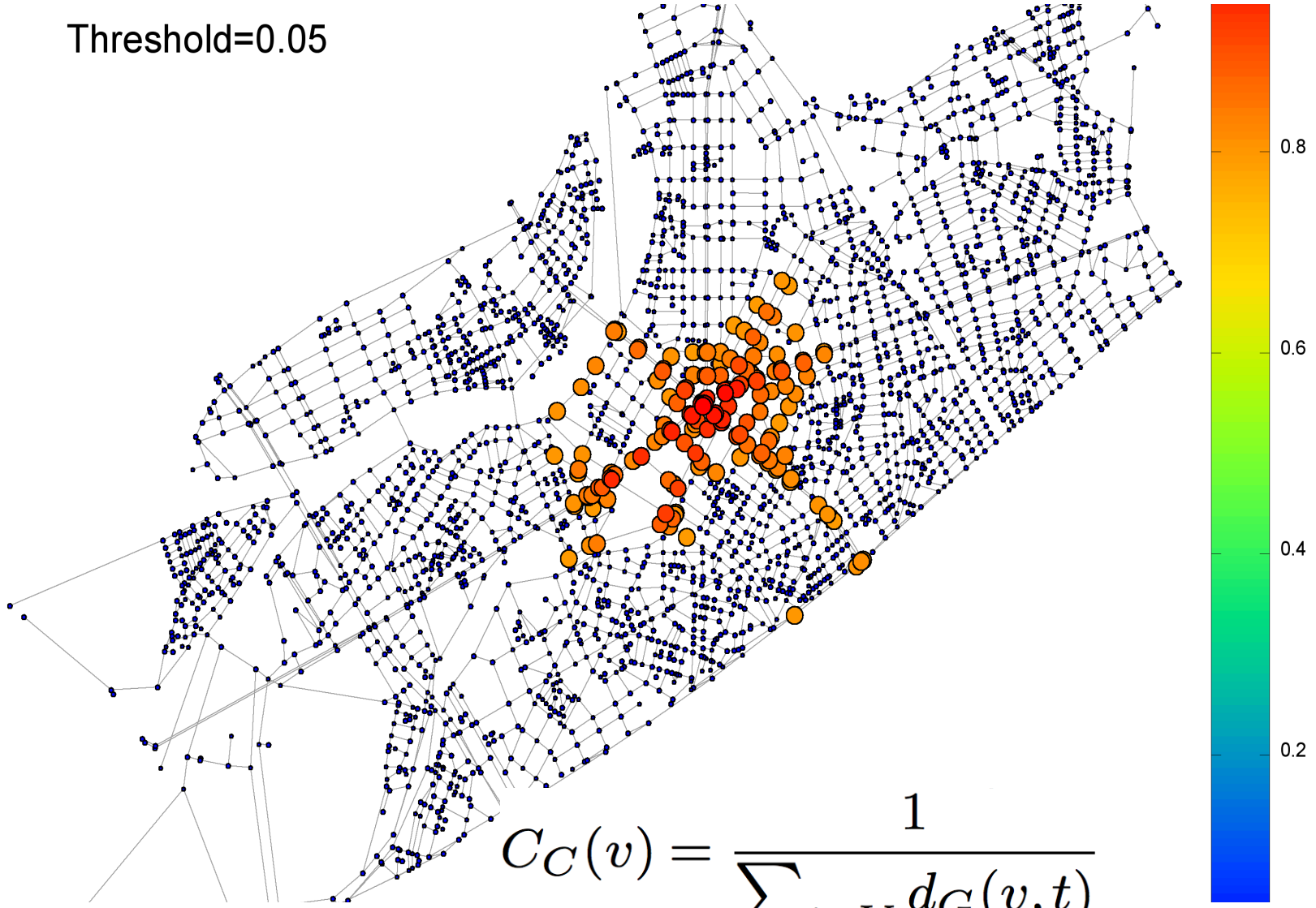
Betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v:

Current_flow_betweenness_centrality



Current-flow betweenness centrality uses an electrical current model for information spreading in contrast to betweenness centrality which uses shortest paths.

closeness_centrality



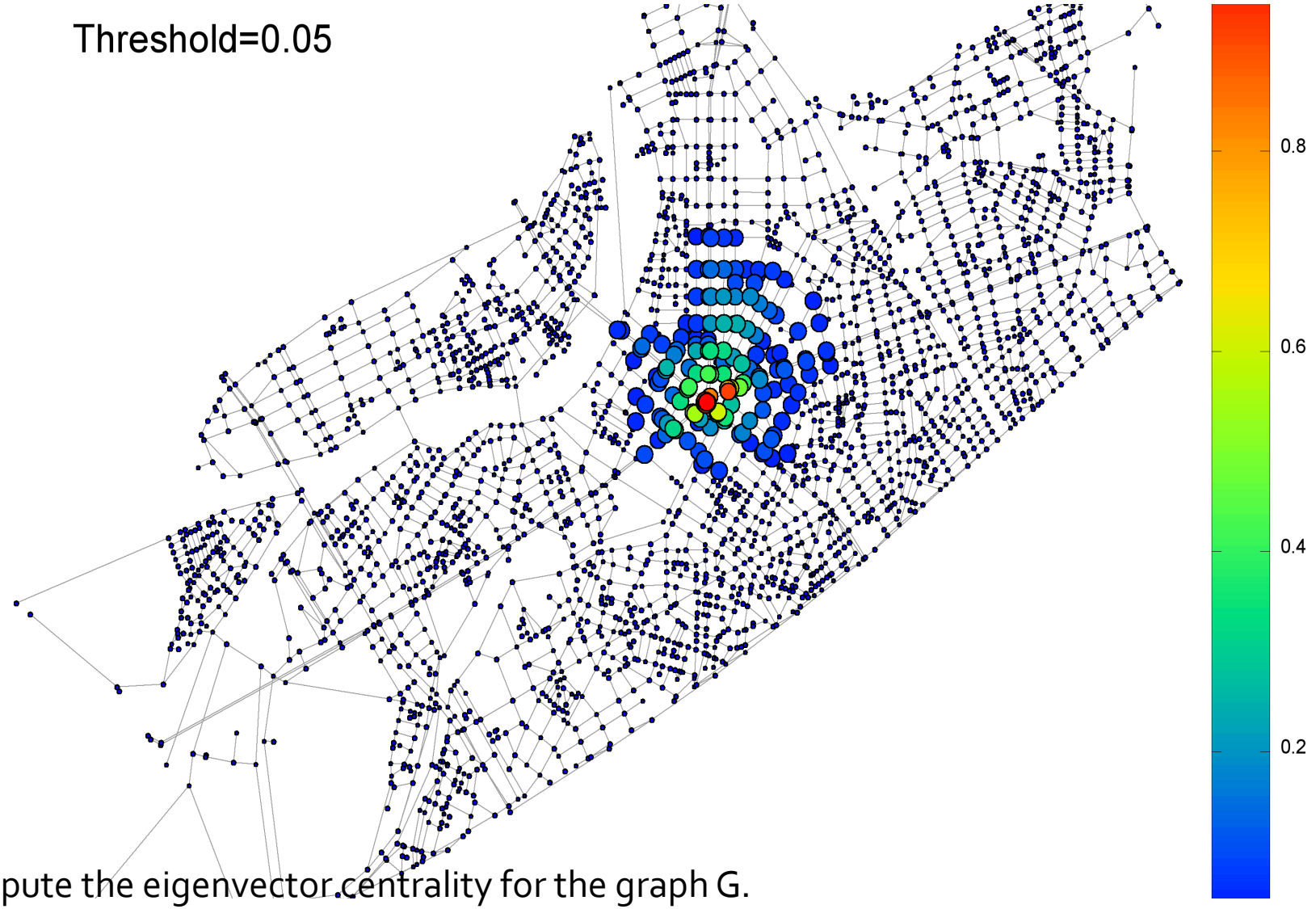
$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}$$

Closeness centrality at a node is 1/average distance to all other nodes.

communicability_centrality

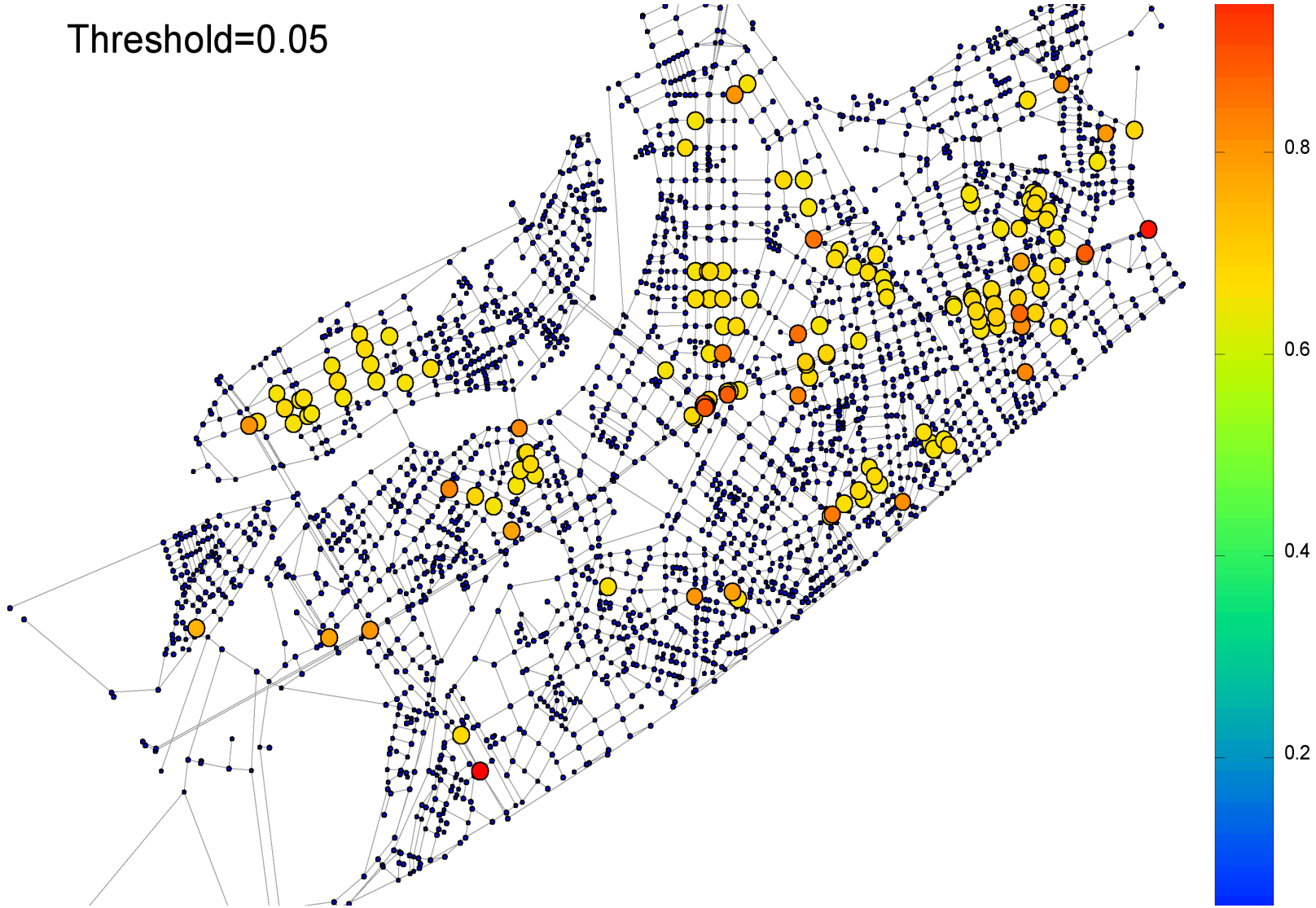
Communicability centrality, also called subgraph centrality, of a node n is the sum of closed walks of all lengths starting and ending at node n.

eigenvector_centrality



Compute the eigenvector centrality for the graph G.
Uses the power method to find the eigenvector for the largest eigenvalue of the adjacency matrix of G.

katz_centrality



Katz centrality computes the relative influence of a node within a network by measuring the number of the immediate neighbors (first degree nodes).

pagerank

All pairs shortest path problems for USA road network

No. of nodes = 23,947,347, No. of edges m = 58,333,344

4-way Opteron 6174 2.3 GHz (12 cores x 4), 256 GB, GCC-4.6.0

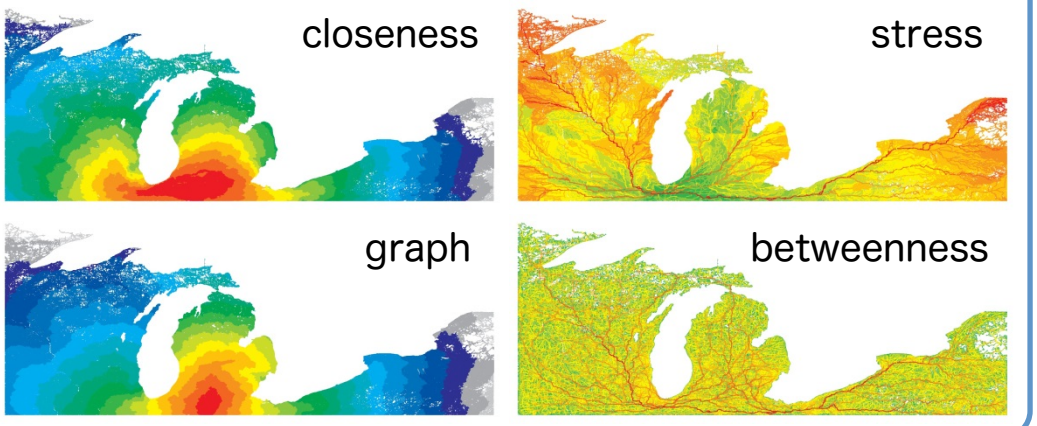
Algorithm	Comp. Time	ratio
Dijkstra's algorithm	512 Years	--
Δ-stepping	9.1 Years	60
Multi-Level Buckets	4.9 Years	110
MSLC + NUMA optimized	7.8 Days	24,000

Computation of 4 centralities for USA road network (Great Lakes)

No. of nodes = 2,758,119, No. of edges m = 6,885,658

4-way Opteron 6174 2.3 GHz (12 cores x 4), 256 GB, GCC-4.6.0

Computation time : 19.35 hours



GraphCT : Georgia Tech
20.6 Days (Only BC)

Graph 500 and Green Graph500 Benchmarks

- **New Graph Search Based Benchmarks for Ranking Supercomputers**
- BFS (Breadth First Search) from a single node on a static, undirected Kronecker graph with average vertex degree edgegactor (=16).
- No. of Nodes = 2^{SCALE}, Average degree = 16
- Performance Metrics：

  TEPS(Traversed Edges per Second) : Graph 500

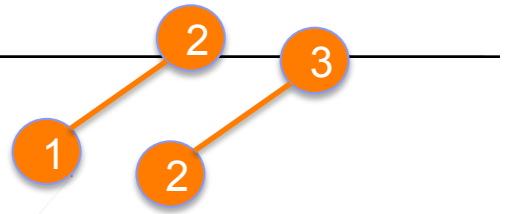
  TEPS/W (Traversed Edges per Second / Watt) : Green Graph500

Step.

1. Generate edgelist
2. Construct Graph（CSR format）
3. BFS
4. Validation

} Repeat 64 times for randomly selected source vertices

Benchmark Flow

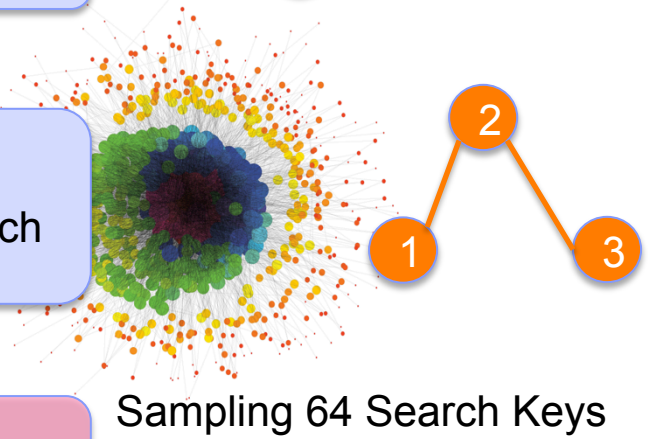


Graph (Edge List) Generation

Kernel 1

Graph Construction

(Conversion to any space-efficient format such as CSR or CSC)



Sampling 64 Search Keys

Kernel 2

Breadth First Search

(Reference or Customized Impl.)

Validation for BFS tree (5 rules)

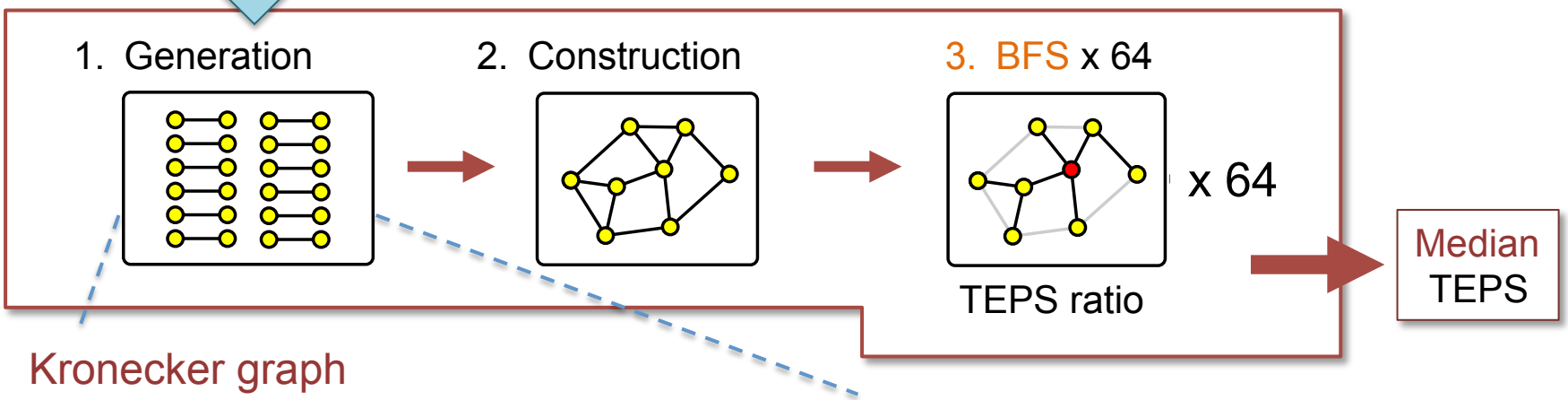
64 times iterations

Graph500 Benchmark www.graph500.org

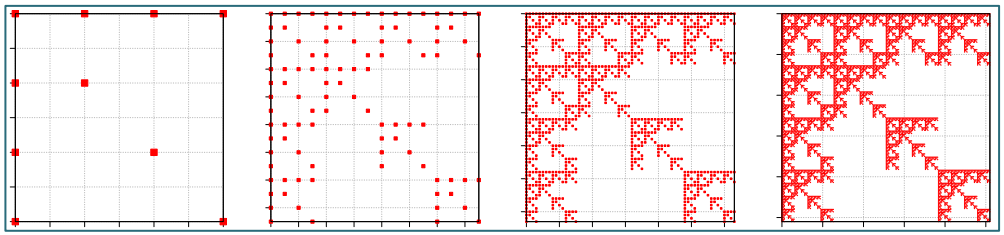
- Measures a performance of irregular memory accesses
- TEPS score (# of Traversed edges per second) in a BFS

Input parameters for problem size

SCALE & edgefactor (=16)

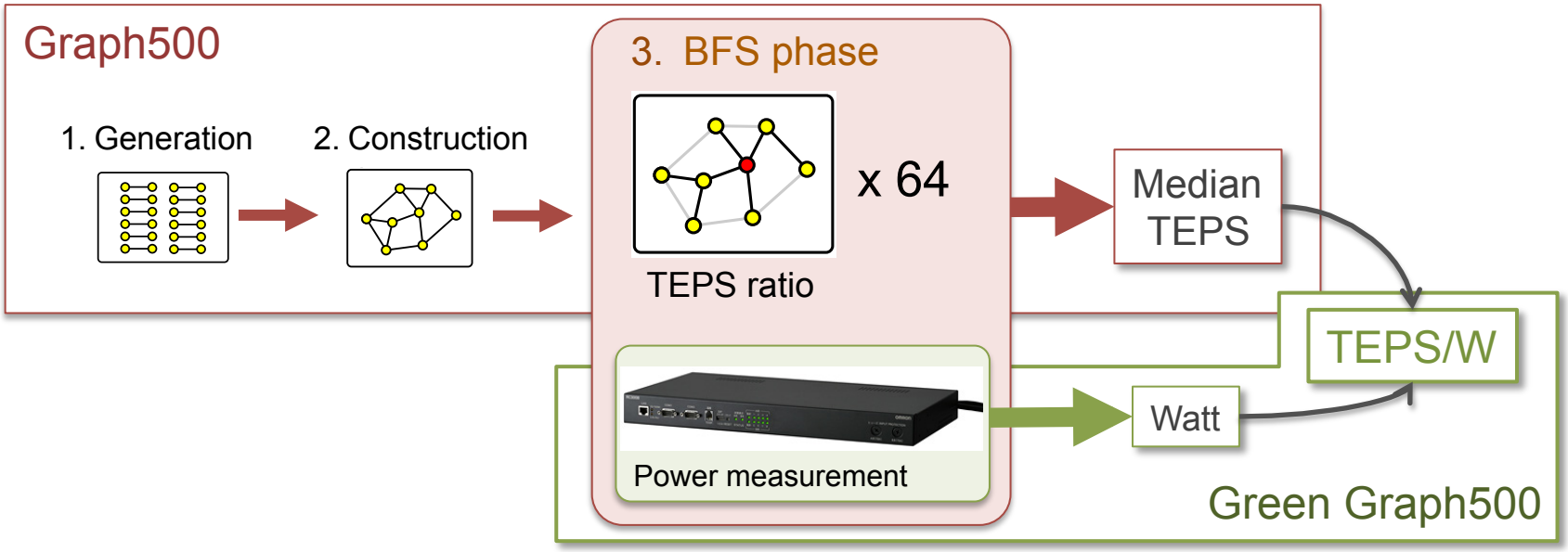


- Generates synthetic scale-free network with 2^{SCALE} *vertices* and $2^{SCALE} \times edgefactor$ *edges* by using ***SCALE****-times* the Recursive Kronecker products



Green Graph500 Benchmark http://green.graph500.org

- Measures power-efficiency using TEPS/W score
- Our results on various systems such as **SGI UV series** and **Xeon servers**, **Android devices**



Five Business Areas --- Graph500

Graph CREST

Cybersecurity

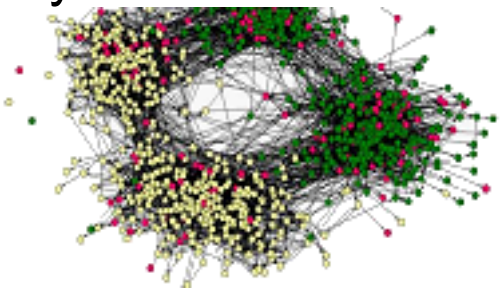
- 15 billion log entries/day
- Full data scan required

Medical Informatics

- 50 million patient records, with 20 to 200 records per patient, billions of individuals
- Entry resolution important

Social Networks

- 例） facebook twitter
- Nearly unbounded dataset size



Data Enrichment

- Easily PB of data
- 例）Maritime Domain Awareness
  - Hundreds of Millions of Transponders
  - Tens of Thousands of Cargo Ships
  - Tens of Millions of Pieces of Bulk Cargo
  - May involve additional data

Symbolic Network

- 例) the Human Brain
- 25 billion neurons
- 7,000+ connections per Neuron

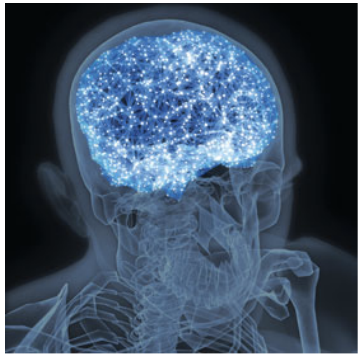
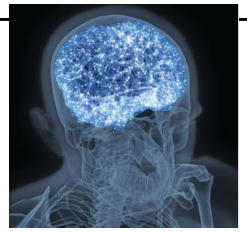


Image: Illustration by Mirko Ilic

Target networks



Human Project

Human Brain
89 B, 100 T

Graph500 (Huge)

Graph500 (Large)

Graph500 (Medium)

1trillion

Twitter2009

Graph500 (Small)

1blillion

Graph500 (Mini)

twitter-rv

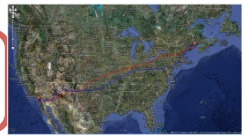
Graph500 (Toy)

soc-LiveJournal1

USA-road-d.USA.gr

cit-Patents

US road network



USA-road-d.NY.gr

1blillion

1trillion

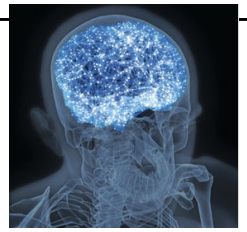
45
40
35
30
25
20

# of edges (in logscale)

15
20
25
30
35
40
45

# of vertices (in logscale)

Target networks on Smartphone



Human Project

Human Brain
89 B, 100 T

Graph500 (Huge)

Graph500 (Large)

Graph500 (Medium)

1trillion

Twitter2009

Graph500 (Small)

1blillion

Graph500 (Mini)

twitter-rv

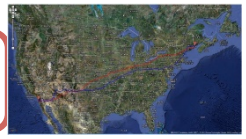
Graph500 (Toy)

soc-LiveJournal1

USA-road-d.USA.gr

cit-Patents

US road network



USA-road-d.NY.gr

1blillion

1trillion

Smartphone

Graph500 (SCALE20)
・Smartphone (4 cores)

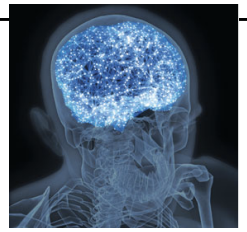
45
40
35
30
25
20

# of edges (in logscale)

20
25
30
35
40
45

# of vertices (in logscale)

Target networks on Single-server



Human Project

Human Brain
89 B, 100 T

Graph500 (Huge)

Graph500 (Large)

Graph500 (Medium)

1trillion

Twitter2009

Graph500 (Small)

1blillion

Graph500 (Mini)

twitter-rv

Graph500 (Toy)

Single server

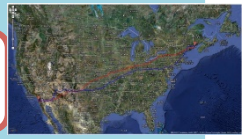
Graph500 (SCALE29)
・4-way Intel Xeon (64 cores)

soc-LiveJournal1

USA-road-d.USA.gr

cit-Patents

US road network



USA-road-d.NY.gr

1blillion

1trillion

Smartphone

Graph500 (SCALE20)
・Smartphone (4 cores)

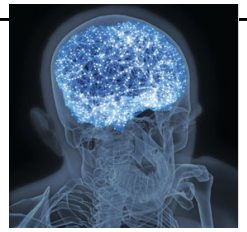
45
40
35
30
25
20

20
25
30
35
40
45

# of edges (in logscale)

# of vertices (in logscale)

Target networks on UV2000



Human Project

Human Brain
89 B, 100 T

Graph500 (Huge)

Graph500 (Large)

Graph500 (Medium)

1trillion

UV 2000

Graph500 (SCALE32)
・UV2000 (1rack, 640 cores, 16TB memory)

Twitter2009

Graph500 (Small)

1blillion

Graph500 (Mini)

twitter-rv

Graph500 (Toy)

Single server

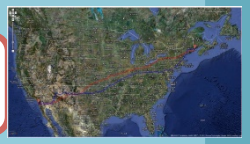
Graph500 (SCALE29)
・4-way Intel Xeon (64 cores)

soc-LiveJournal1

USA-road-d.USA.gr

cit-Patents

US road network



USA-road-d.NY.gr

1blillion

1trillion

Smartphone

Graph500 (SCALE20)
・Smartphone (4 cores)

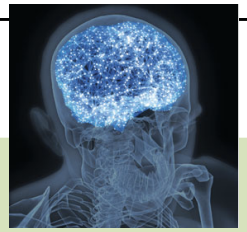
45
40
35
30
25
20

# of edges (in logscale)

20
25
30
35
40
45

# of vertices (in logscale)

Target networks on Supercomputer



Human Project

Human Brain
89 B, 100 T

Graph500 (Huge)

Graph500 (Large)

1trillion

UV 2000

Graph500 (SCALE32)
・UV2000 (1rack, 640 cores)

Graph500 (Medium)

K and Sequoia

Graph500 (SCALE40)
・BlueGene/Q (64K nodes)
・K computer (64K nodes)

Twitter2009

Graph500 (Small)

1blillion

Graph500 (Mini)

twitter-rv

Graph500 (Toy)

Single server

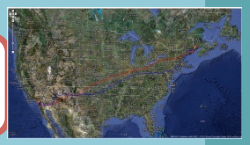
Graph500 (SCALE29)
・4-way Intel Xeon (64 cores)

soc-LiveJournal1

USA-road-d.USA.gr

cit-Patents

US road network



USA-road-d.NY.gr

1blillion

1trillion

Smartphone

Graph500 (SCALE20)
・Smartphone (4 cores)

45
40
35
30
25
20

20 25 30 35 40 45

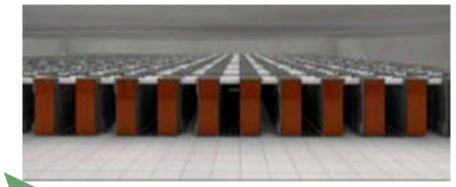
# of edges (in logscale)

# of vertices (in logscale)

K computer Specifications

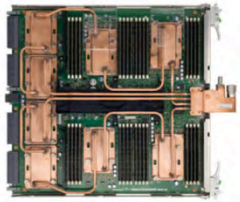
CPU *(SPARC64 VIIIfx)*	Cores/Node	8 cores (@2GHz)
	Performance	128GFlops
	Architecture	SPARC V9 + HPC extension
	Cache	L1(I/D) Cache : 32KB/32KB L2 Cache : 6MB
	Power	58W (typ. 30 C)
	Mem. bandwidth	64GB/s.
Node	Configuration	1 CPU / Node
	Memory capacity	16GB (2GB/core)
System board(SB)	No. of nodes	4 nodes /SB
Rack	No. of SB	24 SBs/rack
System	Nodes/system	> 80,000

Inter-connect	Topology	6D Mesh/Torus
	Performance	5GB/s. for each link
	No. of link	10 links/ node
	Additional feature	H/W barrier, reduction
	Architecture	Routing chip structure (no outside switch box)
Cooling	CPU, ICC*	Direct water cooling
	Other parts	Air cooling

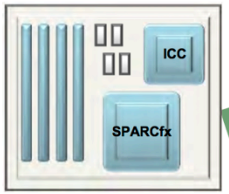


System
LINPACK 10 PFlops
over 1PB mem.
800 racks
80,000 CPUs
640,000 cores

Rack
12.3 TFlops
15TB memory

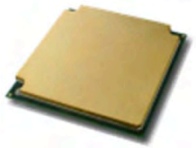


System Board
512 GFlops
64 GB memory



Node
128 GFlops
16GB Memory
64GB/s Memory band width

CPU
128GFlops
SPARC64™ VIIIfx
8 Cores@2.0GHz



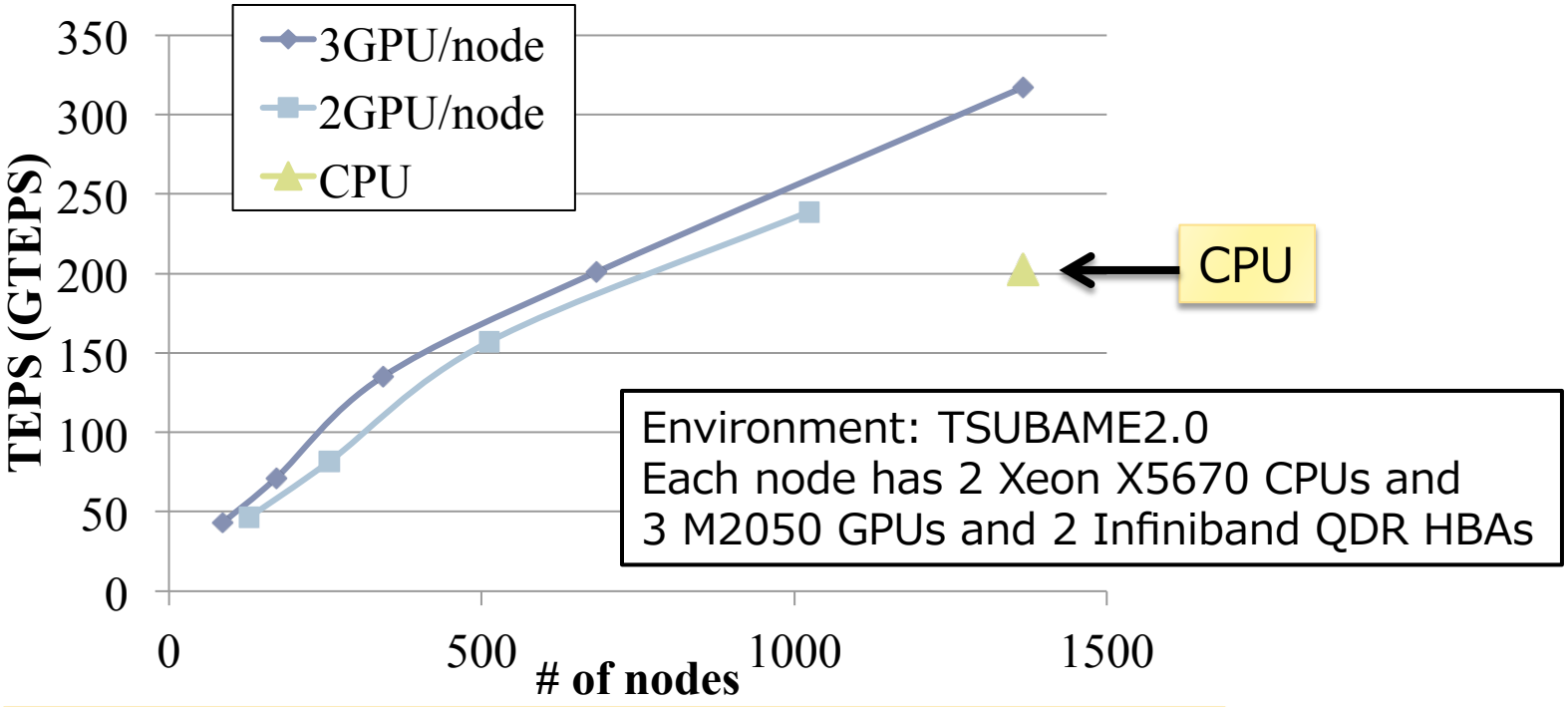
* ICC : Interconnect Chip

Our implementations for Graph500 and Green Graph500

- Multiple nodes and multiple processes
  - Multiple CPUs (K computer, TSUBAME 2.0, Fujitsu FX10, TSUBAME KFC)
    - 38621.4GTEPS, 1st position in 10th Graph500 list
  - Multiple CPUs and GPUs (TSUBAME 2.0 & TSUBAME KFC)
    - 317GTEPS, 4th position in 4th Graph500 list
    - 6.72GTEPS/kW, 1st position in 2nd Green Graph500 list (Big)
- Single node and single process
  - NUMA optimized and multiple threads (SGI Altix UV2000, 4-way Intel Xeon server)
    - 174.7GTEPS, 40th position in 10th Graph500 list (world fastest implementation for single node
  - Multiple threads and power efficient (Android tablet, Android based smart phone)
    - 153.2 GTEPS/kW, 1st position in 2nd Green Graph500 list (Small)

Overall Performance of Distributed BFS

- We propose an optimized method based on **2D based partitioning** and other various optimization methods such as communication compression and vertex sorting.
- We developed CPU implementation and GPU implementation.
- Our optimized GPU implementation can solve BFS (Breadth First Search) of large-scale graph with 2^{35} (34.4 billion) vertices and 2^{39} (550 billion) edges for 1.733 seconds with 1366 nodes and 4096 GPUs on TSUBAME 2.0 ⇒ **317 GTEPS**
- **1.5** times faster than CPU implementation.



Weak Scaling (2^{23} vertices per GPU)

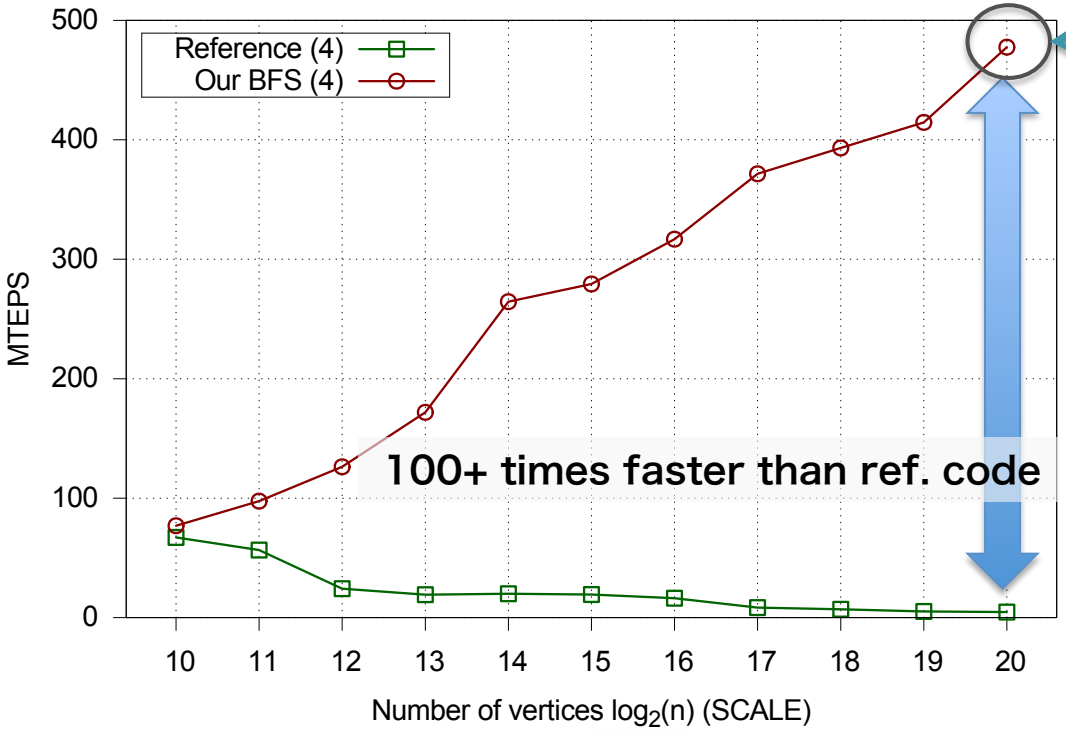
Our implementations for Graph500 and Green Graph500

- Multiple nodes and multiple processes
  - Multiple CPUs (K computer, TSUBAME 2.0, Fujitsu FX10, TSUBAME KFC)
    - 38621.4GTEPS, 1st position in 10th Graph500 list
  - Multiple CPUs and GPUs (TSUBAME 2.0 & TSUBAME KFC)
    - 317GTEPS, 4th position in 4th Graph500 list
    - 6.72GTEPS/kW, 1st position in 2nd Green Graph500 list (Big)
- Single node and single process
  - NUMA optimized and multiple threads (SGI Altix UV2000, 4-way Intel Xeon server)
    - 174.7GTEPS, 40th position in 10th Graph500 list (world fastest implementation for single node
  - Multiple threads and power efficient (Android tablet, Android based smart phone)
    - 153.2 GTEPS/kW, 1st position in 2nd Green Graph500 list (Small)

Green Graph500 on Xperia-A-SO-04E

fast and energy-efficiency

1,048,576 (= 2^{20}) vertices, 16,777,216 (= 2^{24}) edges



Smartphone
SONY Xperia-A-SO-04E
CPU : 4-core Snapdragon
RAM : 2 GB

p is # of threads

Same amount of Power-consumption

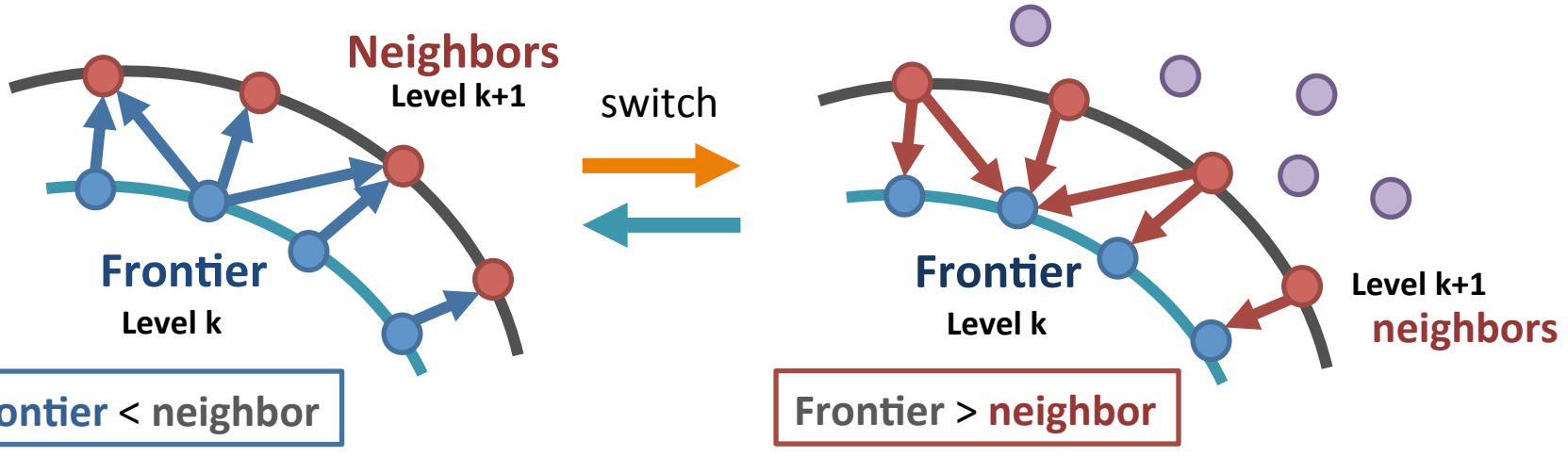
Implementation	SCALE	MTEPS	watt	MTEPS/W
Reference $(p = 1)$	20	3.25	3.15	1.03
Reference $(p = 4)$	20	4.58	3.22	1.42
This study $(p = 1)$	20	136.29	3.23	42.25
This study $(p = 2)$	20	248.08	2.99	82.92
This study $(p = 4)$	20	477.63	3.12	153.17

Our implementations for Graph500 and Green Graph500

- Multiple nodes and multiple processes
  - Multiple CPUs (K computer, TSUBAME 2.0, Fujitsu FX10, TSUBAME KFC)
    - 38621.4GTEPS, 1st position in 10th Graph500 list
  - Multiple CPUs and GPUs (TSUBAME 2.0 & TSUBAME KFC)
    - 317GTEPS, 4th position in 4th Graph500 list
    - 6.72GTEPS/kW, 1st position in 2nd Green Graph500 list (Big)
- Single node and single process
  - NUMA optimized and multiple threads (SGI Altix UV2000, 4-way Intel Xeon server)
    - 174.7GTEPS, 40th position in 10th Graph500 list (world fastest implementation for single node
  - Multiple threads and power efficient (Android tablet, Android based smart phone)
    - 153.2 GTEPS/kW, 1st position in 2nd Green Graph500 list (Small)

Hybrid BFS for low-diameter graph

- Efficient for Low-diameter graph **[Beamer2011, 2012]**
  - **scale-free** and/or **small-world** property such as social network.
- At higher ranks in Graph500 benchmark
- Hybrid algorithm
  - combines top-down algorithm and bottom-up algorithm
  - reduces unnecessary edge traversal



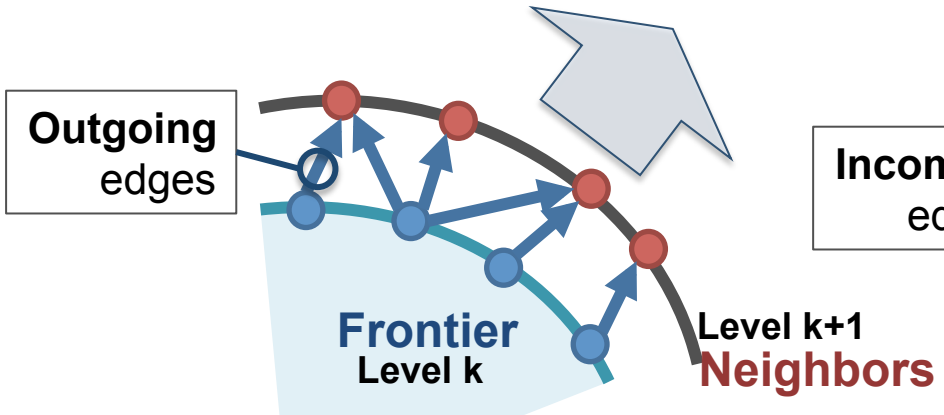
Direction-optimizing BFS

Beamer2012 @ SC2012

Chooses **direction** from *Top-down* or *Bottom-up*

Top-down direction

- Efficient for small-frontier
- Uses out-going edges



Input : *Directed graph* $G = (V, A^{\mathcal{F}})$, *Queue* Q^F
Data : *Queue* Q^N, *visited*, *Tree* $\pi(v)$

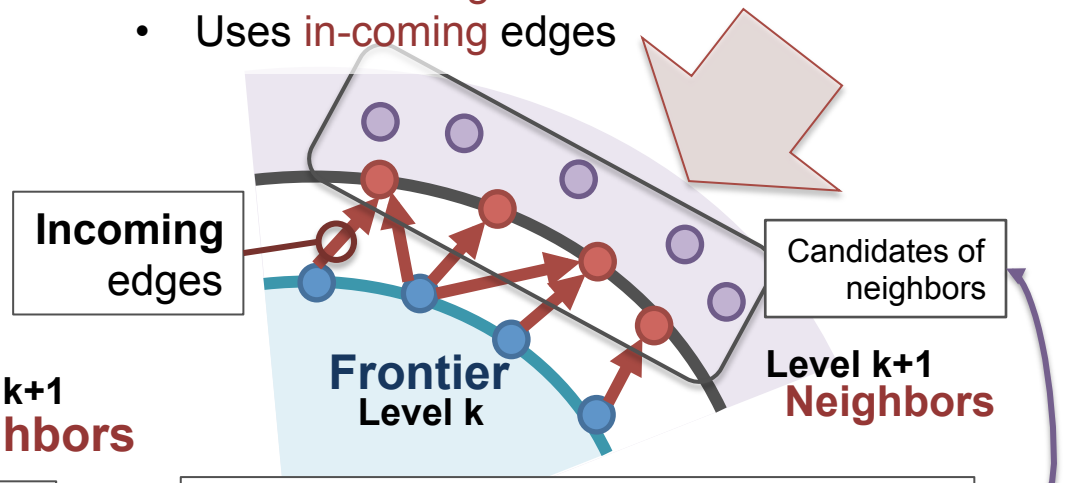
$Q^N \leftarrow \emptyset$
for $v \in Q^F$ **in parallel do**
  for $w \in A^{\mathcal{F}}(v)$ **do**
    if $w \notin$ *visited* **atomic then**
      $\pi(w) \leftarrow v$
      visited $\leftarrow$ *visited* $\cup \{w\}$
      $Q^N \leftarrow Q^N \cup \{w\}$
$Q^F \leftarrow Q^N$

Current frontier

Unvisited neighbors

Bottom-up direction

- Efficient for large-frontier
- Uses in-coming edges



Input : *Directed graph* $G = (V, A^{\mathcal{B}})$, *Queue* Q^F
Data : *Queue* Q^N, *visited*, *Tree* $\pi(v)$

$Q^N \leftarrow \emptyset$
for $w \in V \setminus$ *visited* **in parallel do**
  for $v \in A^{\mathcal{B}}(w)$ **do**
    if $v \in Q^F$ **then**
      $\pi(w) \leftarrow v$
      visited $\leftarrow$ *visited* $\cup \{w\}$
      $Q^N \leftarrow Q^N \cup \{w\}$
      break
$Q^F \leftarrow Q^N$

Candidates of neighbors

Current frontier

Skips unnecessary edge traversal

Direction-optimizing BFS

Beamer @ SC12

Chooses direction from *Top-down* or *Bottom-up*

for **small** frontier — for **large** frontier

# of traversal edges of Kronecker graph with SCALE 26

$|V| = 2^{26}$, $|E| = 2^{30}$

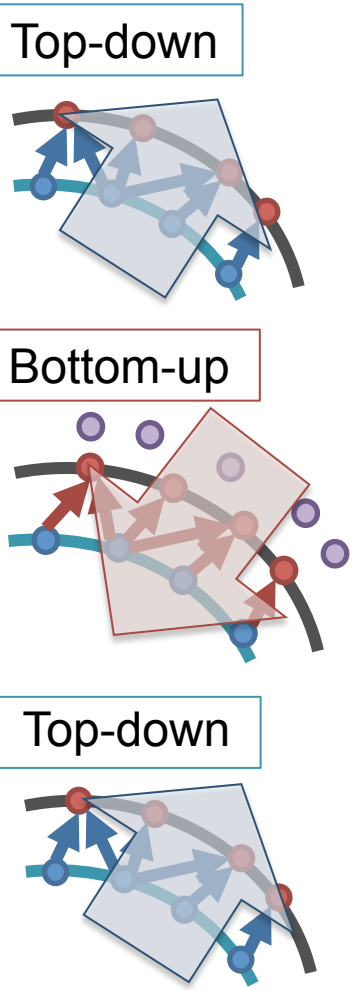
Distance from source

Level	Top-down	Bottom-up	Hybrid
0	2	2,103,840,895	2
1	66,206	1,766,587,029	66,206
2	346,918,235	52,677,691	52,677,691
3	1,727,195,615	12,820,854	12,820,854
4	29,557,400	103,184	103,184
5	82,357	21,467	21,467
6	221	21,240	227
Total	2,103,820,036	3,936,072,360	65,689,631
Ratio	100.00%	187.09%	3.12%

Large frontier

= $|E|$

Hybrid-BFS reduces unnecessary edge traversals



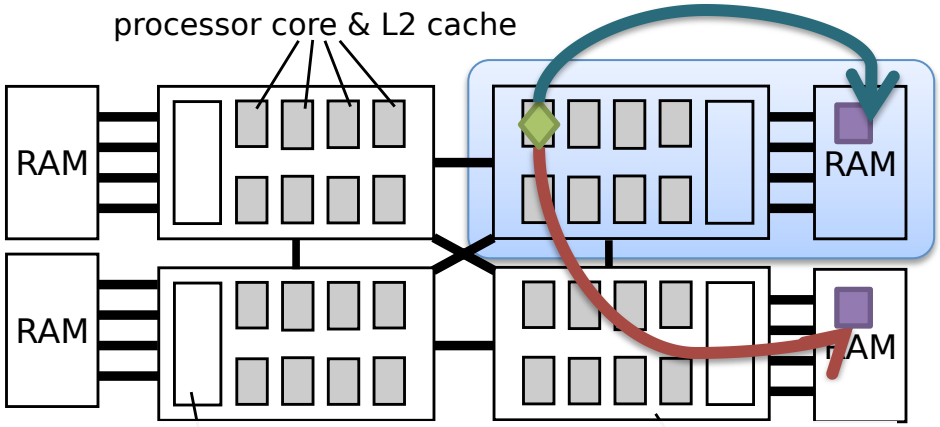
NUMA architecture

- 4-way Intel Xeon E5-4640 (Sandybridge-EP)
  - 4 (# of CPU sockets)
  - 8 (# of physical cores per socket)
  - 2 (# of threads per core)

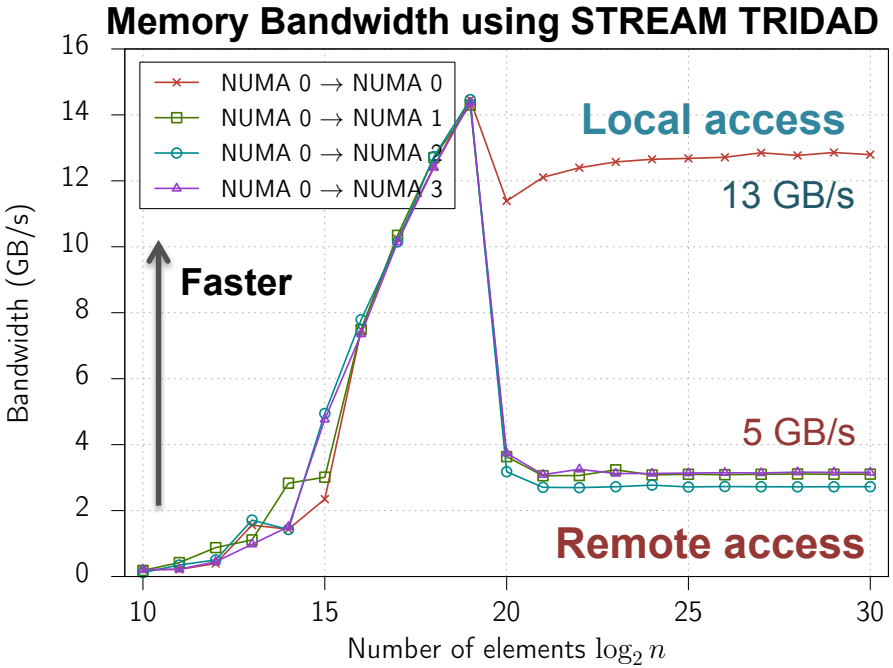
Max.
4 x 8 x 2 = 64 threads

NUMA node

Memory access for Local RAM (Fast)



Memory access for Remote RAM (Slow)

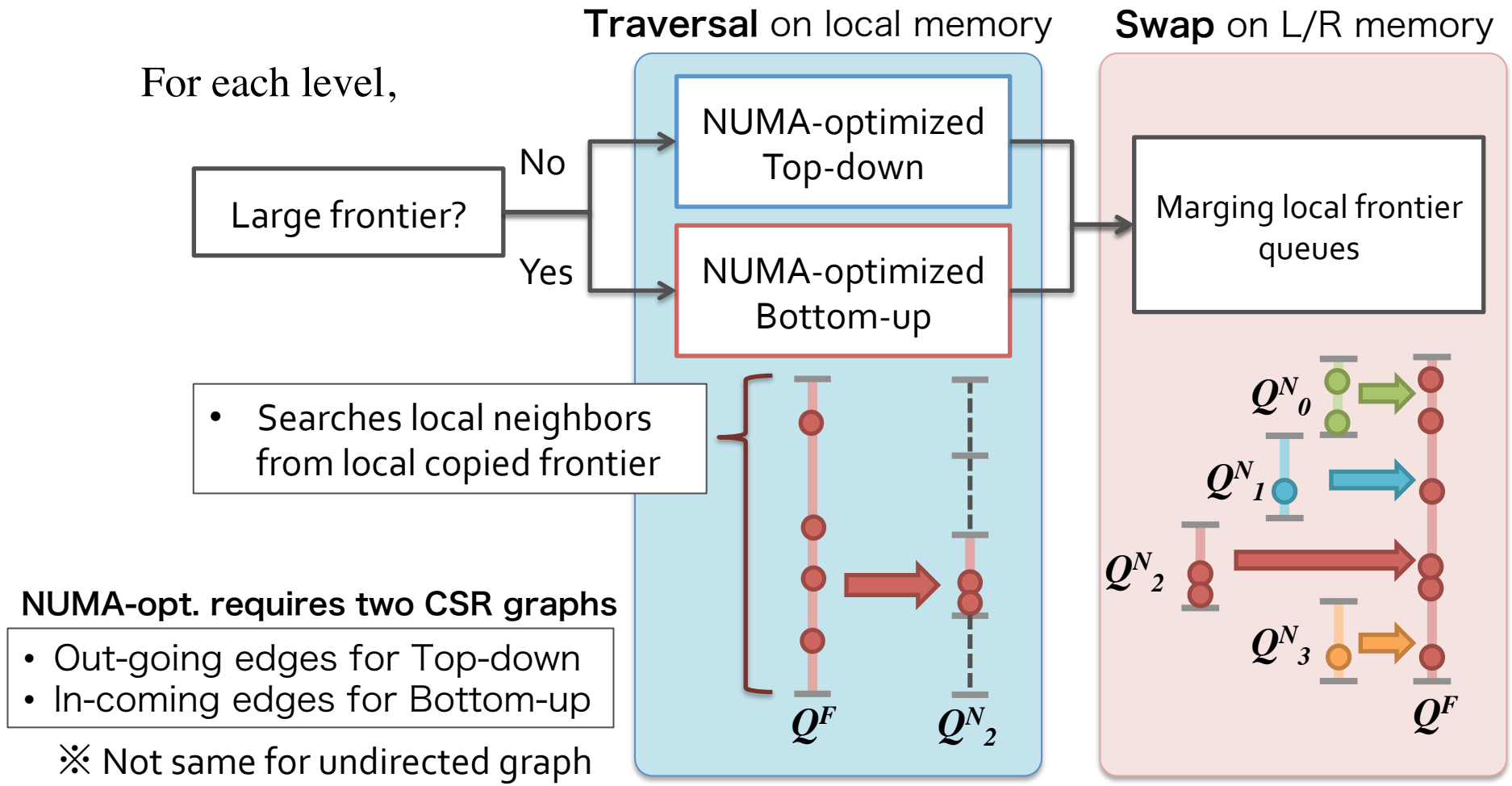


NUMA-aware (optimized) computation

- Reduces and avoids memory accesses for Remote RAM

NUMA-optimized BFS

- Clearly separated to accessing for local and remote memory
  - Edge traversal on Local RAM
  - All-gather of local queues and bitmaps for Remote RAM



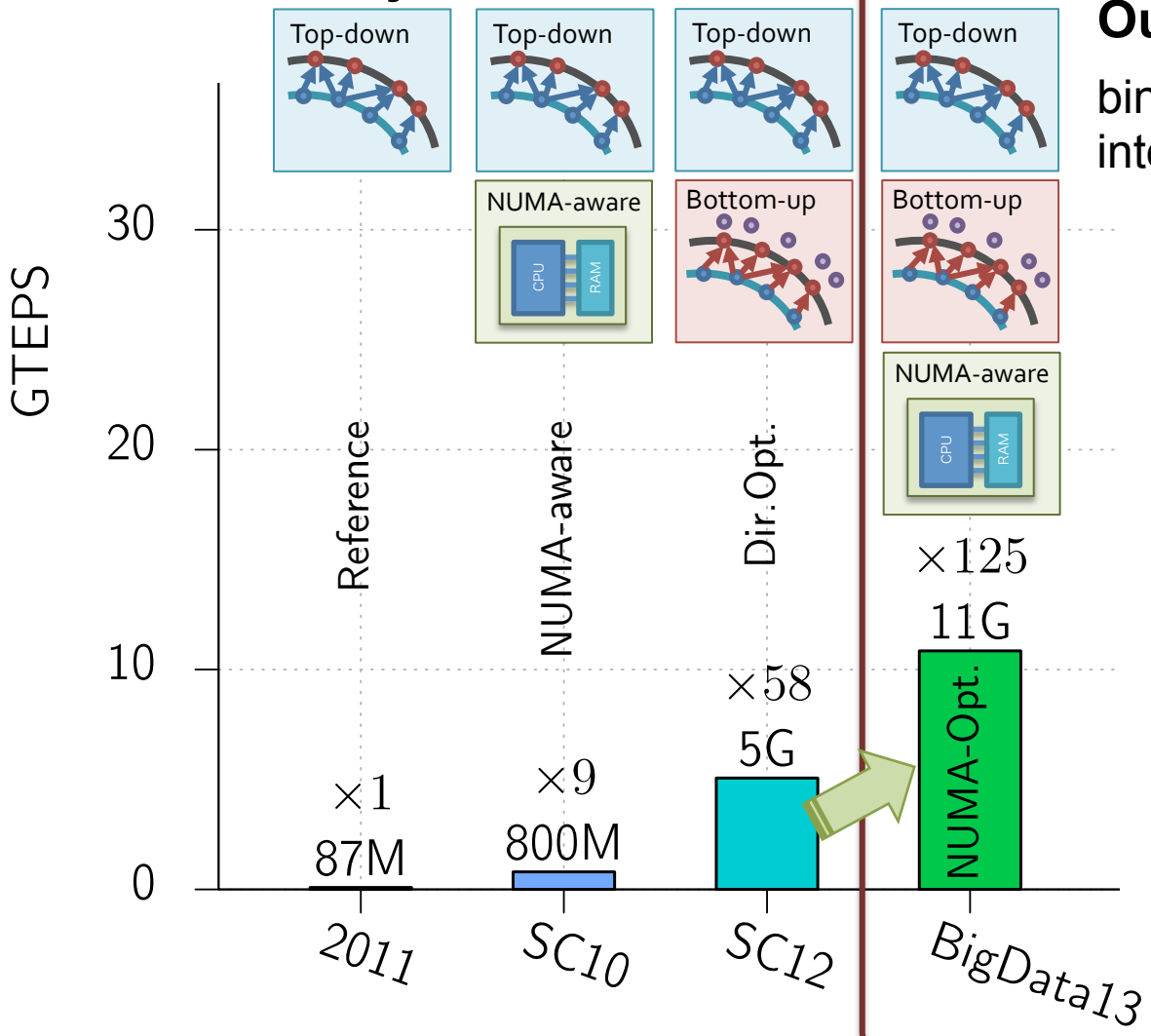
NUMA-opt. requires two CSR graphs

- Out-going edges for Top-down
- In-coming edges for Bottom-up

※ Not same for undirected graph

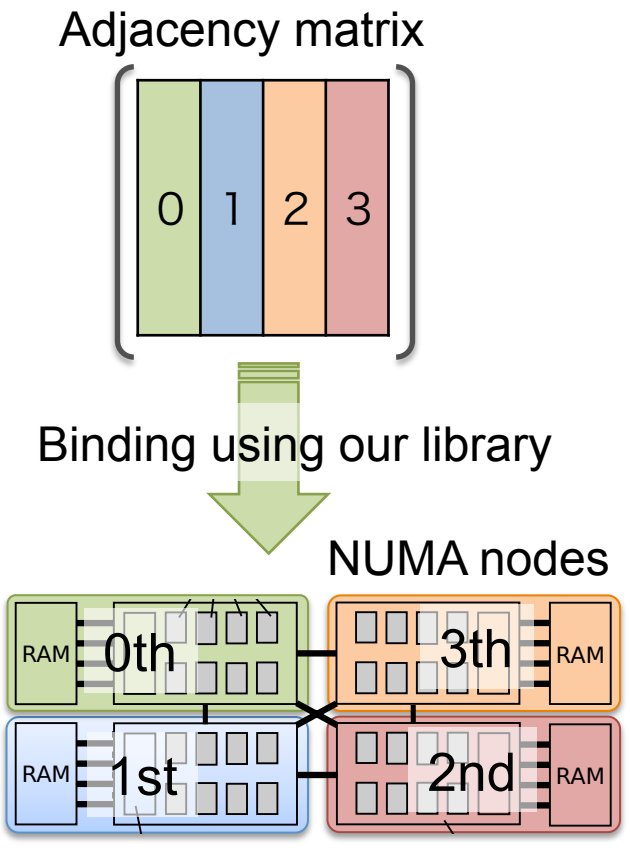
NUMA-Opt. + Dir. Opt. BFS

- Manages memory accesses on a NUMA system carefully.



Our previous result (2013)

binds a partial adjacency matrix into each NUMA nodes.



Proposal; *Sorting Vertex Indices by Out-degree*

- Kronecker graph has many zero-degree (isolated) vertices
  - c.g.) SCALE32; more than half of all vertices

- **Combines** following techniques;
  - Yasui et al. (ISC'14)'s the ***degree aware optimizations***
    - Ignoring the zero-degree vertices
    - sorting vertices in each adjacent list
  - Ueno et al. (HPDC'12)'s the ***vertex sorting***
    - renumbering vertex indices by degree

Graph500 benchmark on a 4-way Intel Xeon

Implementation	**SCALE**	**TEPS**
Dir. Opt. (baseline) [4], [5]	28	5.1 G
+ NUMA Opt. [35]	27	10.9 G
+ NUMA Opt. + Degree Opt. [36]	27	29.0 G
This paper (*DG-V*)		
+ NUMA Opt. + Degree Opt. + Vertex Sorting	27	42.5 G

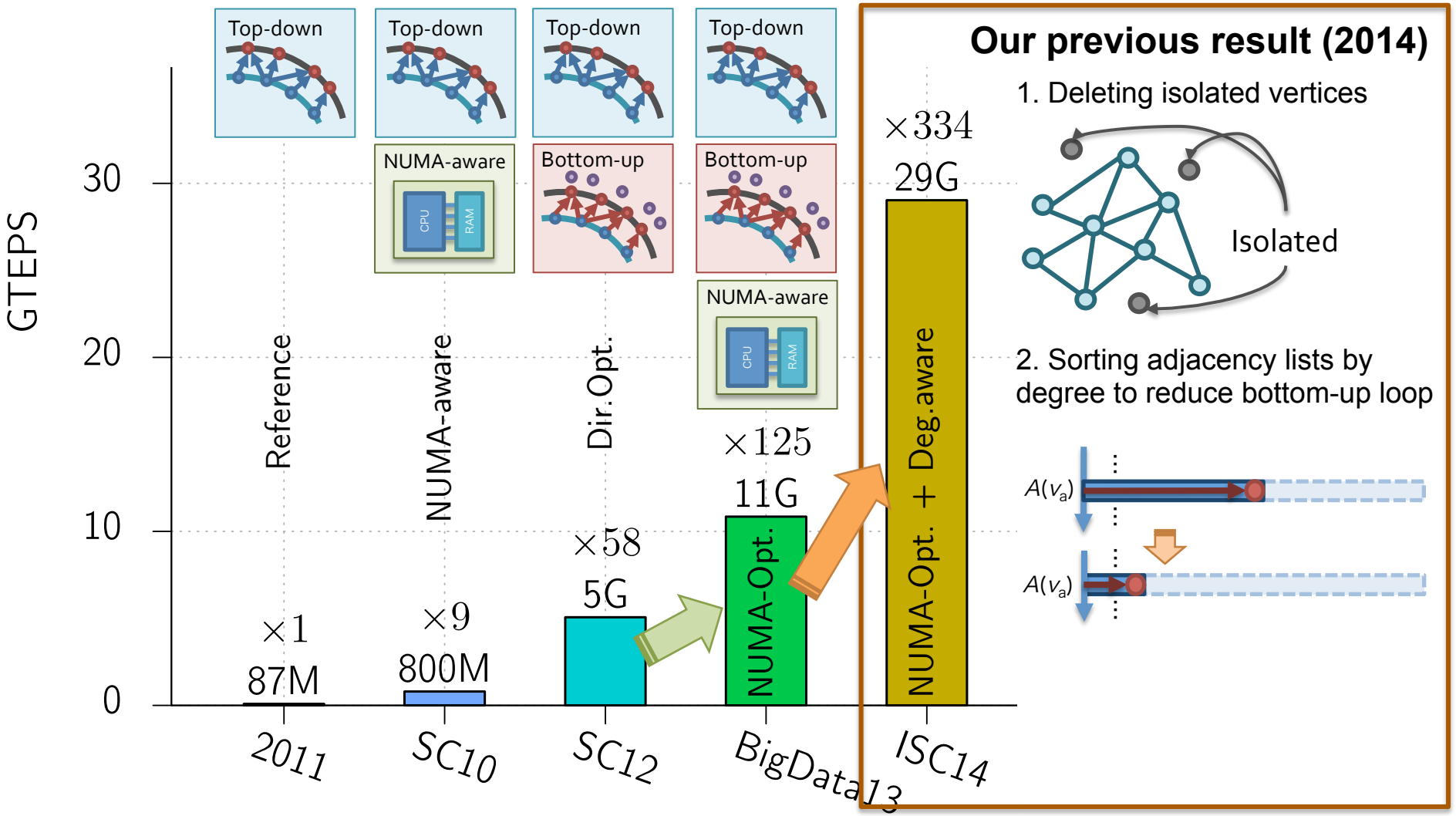
NUMA

2.66X by Yasui's

1.47X by Ueno's

Degree-aware + NUMA-opt. + Dir. Opt. BFS

- Manages memory accesses on NUMA system
  - Each NUMA node contains CPU socket and local memory



Graph500 benchmark

TUTORIAL

How to run the Graph500 benchmark

1. Installation

```
$ tar zxvf graph500_numa_opt_2013_coatwork.tar.gz
$ cd graph500_numa_opt_2013_coatwork
$ make
```

2. Execute the Graph500 benchmark

```
$ OMP_NUM_THREADS=2 ./graph500 -s 16
```

Usage

- Environment variables
  - OMP_NUM_THREADS=INT ... Number of threads
- Options
  - -s INT … SCALE
  - -e INT … Edgefactor (default 16)

Execution log

0. NETAL header

```
$ OMP_NUM_THREADS=4 ./graph500 –s 16
------------------------------------------------------------
NETAL-BD13
------------------------------------------------------------------------
Implementation:         NETAL-BD13
ULIBC version:          ULIBC (version 1.10, dummy)-HWLOC
RAM size:               0.00 GB
Number CPU procs:       4 (= 1 packages x 4 cores x 1 SMTs)
COMPILER:               GCC 5.2.0
------------------------------------------------------------------------
Graph instance:         Kronecker2010
Scale, Edgefactor:      16 16
Energy-loop:            disable
Number of threads:      4
Number of NUMA nodes:   1
Affinity(major:minor):  disable:disable
Dir. parameters:        64 4
queue buffer size:      16384
------------------------------------------------------------------------
```

Experiment settings

This implementation is the updated version for following paper;

Y. Yasui, K. Fujisawa, and K. Goto: NUMA-optimized Parallel Breadth-first Search on Multicore Single-node System, IEEE BigData 2013 (Acceptance rate 17.37%).

Execution log

1. Generating edge list

```
## ------------------------------------------------------
## Kernel-0: Generating or Parsing edge list and Sampling source vertices
## ------------------------------------------------------
   [NUMA00 on Socket000] size(E): 1.05e+06, 0.01 GB (total: 0.01 GB)
Generating 'Kronecker2010' edges with (A,B,C,D) = (5700,1900,1900,500)
generating kronecker edge list ...
   Ite. 01 of 10 generates 104858 edges (total: 104858) (0.019603 s, 5.349075e+06 E/s)
   Ite. 02 of 10 generates 104858 edges (total: 209716) (0.016570 s, 6.328149e+06 E/s)
   Ite. 03 of 10 generates 104858 edges (total: 314574) (0.019584 s, 5.354285e+06 E/s)
   Ite. 04 of 10 generates 104858 edges (total: 419432) (0.015809 s, 6.632779e+06 E/s)
   Ite. 05 of 10 generates 104858 edges (total: 524290) (0.016792 s, 6.244588e+06 E/s)
   Ite. 06 of 10 generates 104858 edges (total: 629148) (0.015598 s, 6.722503e+06 E/s)
   Ite. 07 of 10 generates 104857 edges (total: 734005) (0.017009 s, 6.164788e+06 E/s)
   Ite. 08 of 10 generates 104857 edges (total: 838862) (0.018647 s, 5.623277e+06 E/s)
   Ite. 09 of 10 generates 104857 edges (total: 943719) (0.014448 s, 7.257581e+06 E/s)
   Ite. 10 of 10 generates 104857 edges (total: 1048576) (0.014475 s, 7.243953e+06 E/s)
done. (generated 1048576 edges)
Generated 'Kron2010' undirected 1048576 edges (SCALE 16.00, edgefactor 16.00: n=65536,
m=1048576)
Takes 0.169004 seconds
```

Generating

....

Execution log

2. Constructing graph representation

```
## ------------------------------------------------------------
## Kernel-1: graph construction
## ------------------------------------------------------------
Constructing graph representation w/o self-loops and duplicated-edges ...
Detecting graph size (0.007903 seconds)
   Number of subgraphs    is 1
   Number of vertices     is 65536
   Number of edges        is 2096194
   Number of local edges  is {
      E[0] has 65536 nodes and 2096194 edges,
   }
   Number of self-loops   is 479
   Number of broken-edges is 0

Allocating NUMA graph representation ...
   [NUMA00 on Socket000] G[0] 0.03 GB (total: 0.04 GB)
done (0.000091 seconds)

G (N= 65536 nodes, M= 2096194 edges) = {
   G[0] = (FG: n=65536, m=2096194), (BG: n=65536 (off=   0), m=2096194 (n/N=100.0%, m/M=100.0%))
}

Constructing degree table ...
   ell:                          1
   number_of_nodes:              65536
   number_of_nonzero_nodes:      46802
   number_of_zero_nodes:         18734
   nonzero/total:                71.41418457 %
   number_of_edges:              2096194
   chunksize:                    65536

   max( degree[  0] ):           25640

   G[  0] n: 65536, m: 2096194, nz(V): 46802, z(V): 18734

done (0.017838 seconds)
```

Getting statistics

Execution log

2. Constructing graph representation (continued)

....

```
Constructing CSR indices ... done. (0.000364 seconds)
Computing Prefix-sum for CSR index ... done. (0.003655 seconds)
Constructing NUMA-aware CSR graph ... done. (0.068494 seconds)
Sorting CSR values without duplication ...
  found and extracts 276312 (0.132 %) duplicated edges in Forward-graphs
  found and extracts 276312 (0.132 %) duplicated edges in Backward-graphs
done. (0.053454 seconds)
G = {
   G[0] = (FG: n=65536, m=1819882), (BG: n=65536 (off=   0), m=1819882 (n/N=100.0%, m/M=86.8%))
} = (N= 65536 nodes, M= 2096194 edges)
done.

Sorting edge list ...
   Ite. 01 of 10 sorting 104858 edges (Elapsed: 0.008706 s, 0.008706 s, 1.204421e+07 E/s)
   Ite. 02 of 10 sorting 209716 edges (Elapsed: 0.017373 s, 0.008660 s, 2.421708e+07 E/s)
   Ite. 03 of 10 sorting 314574 edges (Elapsed: 0.026051 s, 0.008665 s, 3.630363e+07 E/s)
   Ite. 04 of 10 sorting 419432 edges (Elapsed: 0.034673 s, 0.008612 s, 4.870232e+07 E/s)
   Ite. 05 of 10 sorting 524290 edges (Elapsed: 0.044799 s, 0.010107 s, 5.187374e+07 E/s)
   Ite. 06 of 10 sorting 629148 edges (Elapsed: 0.053712 s, 0.008876 s, 7.088315e+07 E/s)
   Ite. 07 of 10 sorting 734005 edges (Elapsed: 0.062484 s, 0.008750 s, 8.388665e+07 E/s)
   Ite. 08 of 10 sorting 838862 edges (Elapsed: 0.071179 s, 0.008686 s, 9.657825e+07 E/s)
   Ite. 09 of 10 sorting 943719 edges (Elapsed: 0.079881 s, 0.008687 s, 1.086355e+08 E/s)
   Ite. 10 of 10 sorting 1048576 edges (Elapsed: 0.088565 s, 0.008674 s, 1.208886e+08 E/s)
done. (0.088585 seconds)

Finished construction (0.313303 seconds).
MemoryUsage = 0.04 GB (47710208 bytes)
LocalMemoryUsages = {
  NUMA[0] 45.5 MB
}
```

Constructing CSR graph

Sorting edge list for speed-up of validation

Execution log

3. Executing 64 BFSs

```
NUMA-optimized BFS (alpha=64, beta=4) ...
   BFS( G(n=       65536, m=      2096194), s=        1469 )
#  i= 01, s= 1469, maxdist= 5, time_s= 0.002456, trav_e= 1048569, teps= 4.269091e+08
# Lv  algorithm   trav(ms)  swap(ms)        QF        Trav    Trav/QF
* -1  init(min)     0.064     0.015
* -1  init(max)     0.064     0.015
*  0   TopDown      0.078     0.116         1          25        25
*  1   BottomUp     0.834     0.026        25      528009    21120.4
*  2   BottomUp     0.402     0.031      8344       38855    4.65664
*  3   BottomUp     0.111     0.065     36611        1821  0.0497391
*  4   TopDown      0.135     0.000      1801        2055    1.14103
*  5   TopDown      0.159     0.000         6           6          1
$        Total      1.719     0.238     46788      570771
! trav: 1.719 ms, 70.0 %, swap: 0.238 ms, 9.7 %, init: 0.079 ms, 3.2 %, other: 0.421 ms, 17.1 %
#
# Local-neighbor-queue
# Lv  algorithm      QN[000]
*  0   TopDown           25
*  1   BottomUp        8344
*  2   BottomUp       36611
*  3   BottomUp        1801
*  4   TopDown            6
*  5   TopDown            0
#         total       46787
#
# Local-traversal-edges
# Lv  algorithm    Trav[000]
*  0   TopDown           25
*  1   BottomUp      528009
*  2   BottomUp       38855
*  3   BottomUp        1821
*  4   TopDown         2055
*  5   TopDown            6
#         total       570771
validating BFS tree ... (Tree:0.011s) ... (Tree2:0.000s) ... done (0.01 seconds)
#  i= 01, s= 1469, maxdist= 5, time_s= 0.002456, trav_e= 1048569, teps= 4.269091e+08
```

- Search-direction
- CPU times and traverse and swap
- frontier-size and #traversed-edges
  on each NUMA node at each level

Number of next frontier nodes
on each NUMA node at each level

Number of traversed edges
on each NUMA node at each level

BFS-ID, Source vertex, Maximum level,
CPU time of BFS in seconds, #Traversed-edges, **TEPS**

TEPS = trav_e / time_s

Execution log

4. Graph500 results

```
SCALE:                  16
edgefactor:             16
nvtx:                   65536
terasize:               1.67772159999999997e-05
A:                      5.70000000000000021e-02
B:                      1.89999999999999995e-02
C:                      1.89999999999999995e-02
D:                      5.00000000000000010e-03
generation_time:        1.69003963470458984e-01
construction_time:      3.13302993774414062e-01
nbfs:                   64

....
```

Parameters

- SCALE and edgefactor (=16)
- Initial matrix (A,B,C,D) for Kronecker generator
- Number of sources (=64)

```
....

min_time:               0.0023191
firstquartile_time:     0.00245941
median_time:            0.00280106
thirdquartile_time:     0.00987935
max_time:               0.0562453
mean_time:              0.0100472
stddev_time:            0.0143069
min_nedge:              1048569
firstquartile_nedge:    1048569
median_nedge:           1048569
thirdquartile_nedge:    1048569
max_nedge:              1048569
mean_nedge:             1048569
stddev_nedge:           0
min_TEPS:               1.86428e+07
firstquartile_TEPS:     1.06137e+08
median_TEPS:            3.74347e+08
thirdquartile_TEPS:     4.2635e+08
max_TEPS:               4.52145e+08
harmonic_mean_TEPS:     1.04364e+08
harmonic_stddev_TEPS:   1.87233e+07
min_validate:           0.0108559
firstquartile_validate: 0.0109969
median_validate:        0.0111861
thirdquartile_validate: 0.0133055
max_validate:           0.0163701
mean_validate:          0.0121689
stddev_validate:        0.00148899
```

BFS Times

Traversd edges

TEPS scores

Validation times

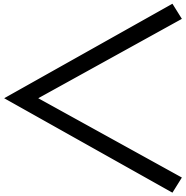
Submit to Graph500 with

- SCALE = 16
- edgefactor=16
- **Medtian TEPS = 374.3 MTEPS**

Performance comparison

- Median GTEPS (=10^9 TEPS) on a NUMA-based server
  - GCC 4.4.7
  - 4-way Intel Xeon E5-4640 2.40GHz and 512 GB RAM

Scale	*Reference code*	*Graph500-BD2013*	*Speedup*
20	0.358	4.578	12.8x
21	0.233	6.317	27.1x
22	0.101	8.433	83.7x
23	0.129	10.023	77.6x
24	0.123	10.829	88.3x
25	0.107	11.155	104.4x
26	0.097	11.149	114.9x
27	0.087	10.854	124.5x
28	–	9.887	–
29	–	9.393	–



100+ faster!

Segmentation fault

NETAL for Graph Analysis

TUTORIAL

Tutorial of NETAL (1)

0. Build on Linux and OSX

```
$ tar zxvf netal-1.51-coatwork.tar.gz && cd netal-1.51-coatwork && make
```

※ We strongly recommend to use Homebrew-GCC or Macports-GCC on OSX

1. Computes Betweenness centrality

```
$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
```

Tutorial of NETAL (1)

0. Build on Linux and OSX

```
$ tar zxvf netal-1.51-coatwork.tar.gz && cd netal-1.51-coatwork && make
```

※ We strongly recommend to use Homebrew-GCC or Macports-GCC on OSX

1. Computes Betweenness centrality

```
$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
```

sample.gr (DIMACS format graph)

```
p sp 9 12
a 1 2 2
a 1 3 5
a 1 4 3
a 2 5 8
a 3 7 1
a 4 7 2
a 4 9 6
a 5 3 4
a 5 6 6
a 6 8 7
a 7 8 9
a 9 8 2
```

This graph contains
9 nodes and 12 edges

Each line describes
a directed weighted edge;
(Tail-ID, Head-ID, Weight)

※ Index starts from 0

http://www.dis.uniroma1.it/challenge9/format.shtml

Tutorial of NETAL (1)

0. Build on Linux and OSX

```
$ tar zxvf netal-1.51-coatwork.tar.gz && cd netal-1.51-coatwork && make
```

※ We strongly recommend to use Homebrew-GCC or Macports-GCC on OSX

1. Computes Betweenness centrality

```
$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
```

u … unweighted
B … Betweenness centrality

Accuracy
1.00 … exact

sample.gr (DIMACS format graph)

```
p sp 9 12
a 1 2 2
a 1 3 5
a 1 4 3
a 2 5 8
a 3 7 1
a 4 7 2
a 4 9 6
a 5 3 4
a 5 6 6
a 6 8 7
a 7 8 9
a 9 8 2
```

This graph contains
9 nodes and 12 edges

Each line describes
a directed weighted edge;
(Tail-ID, Head-ID, Weight)

※ Index starts from 0

http://www.dis.uniroma1.it/challenge9/format.shtml

Tutorial of NETAL (1)

0. Build on Linux and OSX

```
$ tar zxvf netal-1.51-coatwork.tar.gz && cd netal-1.51-coatwork && make
```

※ We strongly recommend to use Homebrew-GCC or Macports-GCC on OSX

1. Computes Betweenness centrality

```
$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
```

u … unweighted
B … Betweenness centrality

Accuracy
1.00 … exact

sample.gr (DIMACS format graph)

```
p sp 9 12
a 1 2 2
a 1 3 5
a 1 4 3
a 2 5 8
a 3 7 1
a 4 7 2
a 4 9 6
a 5 3 4
a 5 6 6
a 6 8 7
a 7 8 9
a 9 8 2
```

This graph contains
9 nodes and 12 edges

Each line describes
a directed weighted edge;
(Tail-ID, Head-ID, Weight)

※ Index starts from 0

http://www.dis.uniroma1.it/challenge9/format.shtml

out_uB.sp (NETAL format)

```
p sp 9 12
l Betweenness
L edge Betweenness
c
d 1 0
d 2 2
d 3 2.83333
d 4 2.16667
...
e 1 2 3
e 1 3 1.83333
e 1 4 3.16667
e 2 5 7
e 3 7 4.83333
...
```

This graph contains
9 nodes and 12 edges

(Node-ID, BC-score)

(Tail-ID, Head-ID, Edge-BC-score)

Tutorial of NETAL (2)

Betweenness centrality（Exact）

```
$ ./netal -i data/sample.gr -t dimacs -z cent uB 1.00 -o out_uB.sp
```

Betweenness centrality（10%-approx. using random sampling of source vertices）

```
$ ./netal -i data/sample.gr -t dimacs -z cent uB 0.10 -o out_uB.sp
```

Multiple centrality

Specify number of threads (default: #threads=logical cores on a system)

```
$ OMP_NUM_THREADS=8 ./netal -i data/sample.gr -z cent wmCGSBD 1.00 -o out.sp
```

C	*Closeness* [1]	$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}$
G	*Graph* [2]	$C_G(v) = \frac{1}{\max_{t \in V} d_G(v,t)}$
S	*Stress* [3]	$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$
B	*Betweenness* [4]	$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$
D	*Degree* [5]	$C_D(v) = \sum_{v \in V} \deg_G(v)$

Options

m … computes maximum connected components※1 only

u … computes unweighted centrality

w … computes weighted centrality

※1 Using Shiloach-Vishkin algorithm

Tutorial of NETAL (3)

```
$ OMP_NUM_THREADS=8 ./netal -i data/sample.gr -z cent wmCGSBD 1.00 -o out.sp
```

out.sp

```
c file name: out.sp
c instance name: sample.gr
c created: Mon Feb  9 23:51:36 2015
c
p sp 9 12
c
c centrality (mwBCDGS, acc=100.000000%, weighted, random sampling) (|Vs|=9, exact)
c diameter is 21 (s=1)
c
l Closeness
l Degree
l Graph
l Betweenness
l Stress
L edge Betweenness
L edge Stress
c
d 1 1.18033 3 0.5 0 0
d 2 1.05882 1 0.380952 3 3
d 3 6.54545 1 0.8 6 6
d 4 4.5 2 1 2 2
...
e 1 2 0 0
e 1 3 0 0
e 1 4 3 3
e 2 5 6 6
e 3 7 8 8
e 4 7 0 0
e 4 9 4 4
...
```

c-line is a comment line

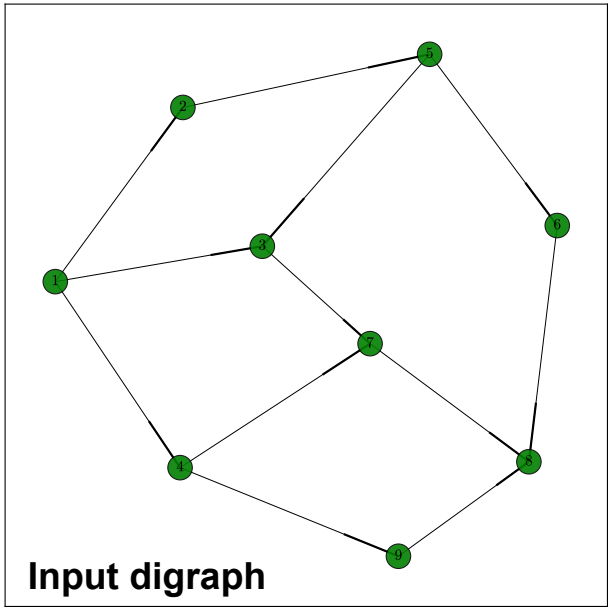
l-line describes **node** label

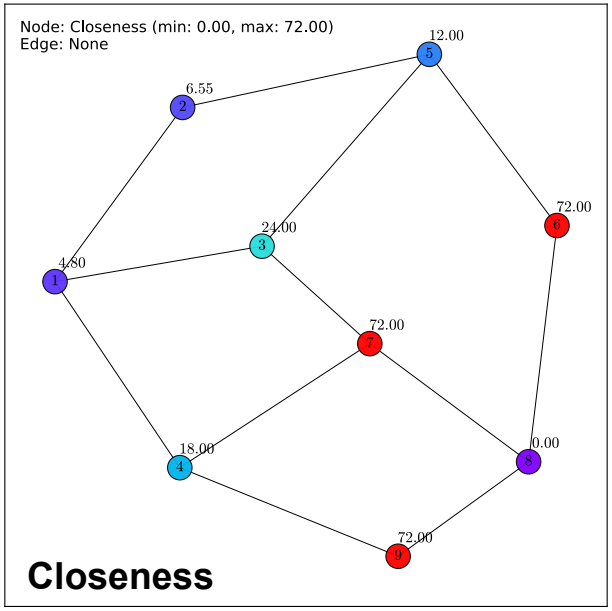
L-line describes **edge** label

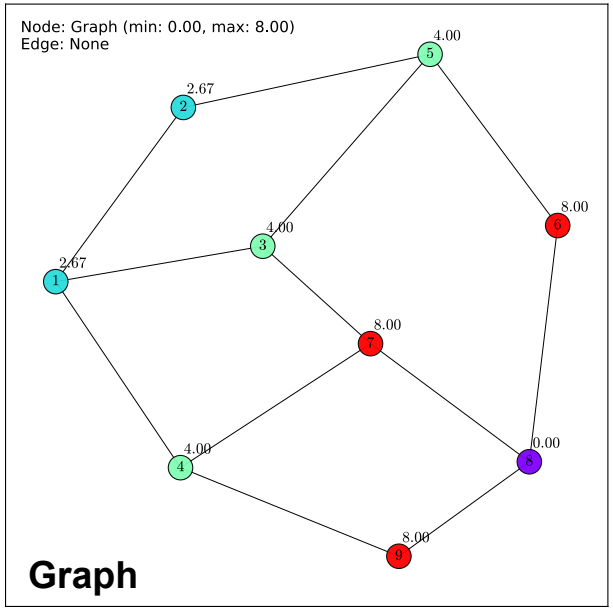
Centrality scores for each node
d <Node-ID> Closeness Degree Graph Betweenness Stress

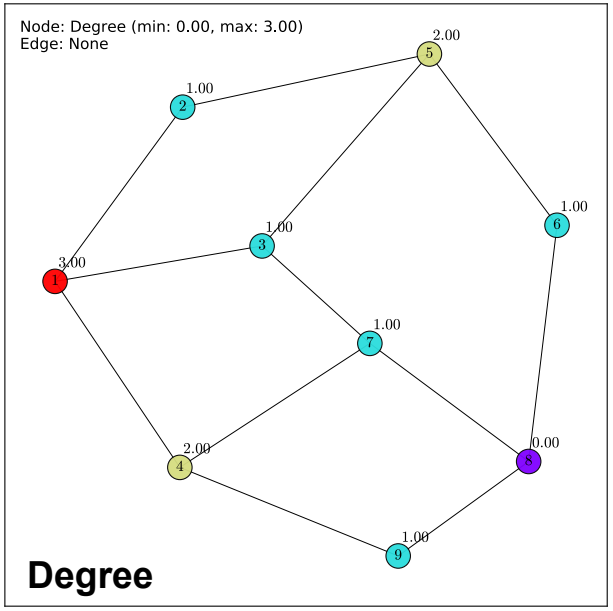
Centrality scores for each edge
e <Tail-Node> <Head-Node> edge-Betweenness edge-Stress

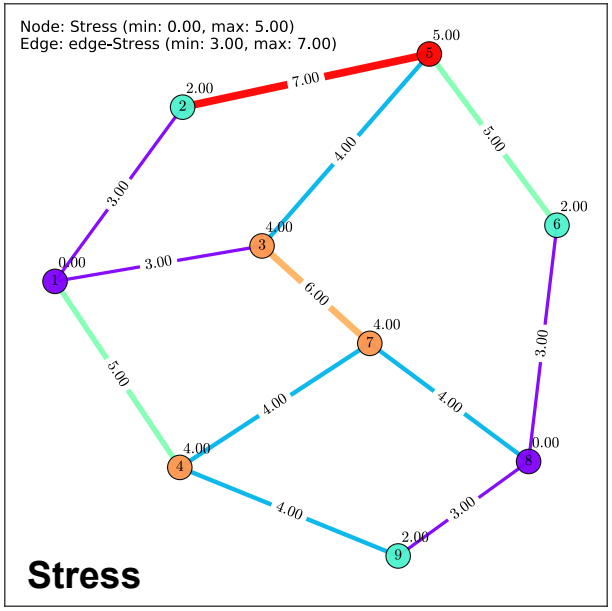
Examples: Digraph with 9 nodes and 12 edges

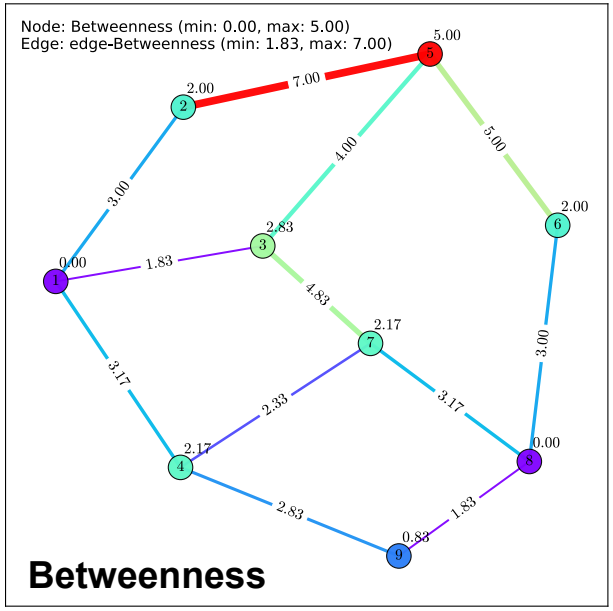




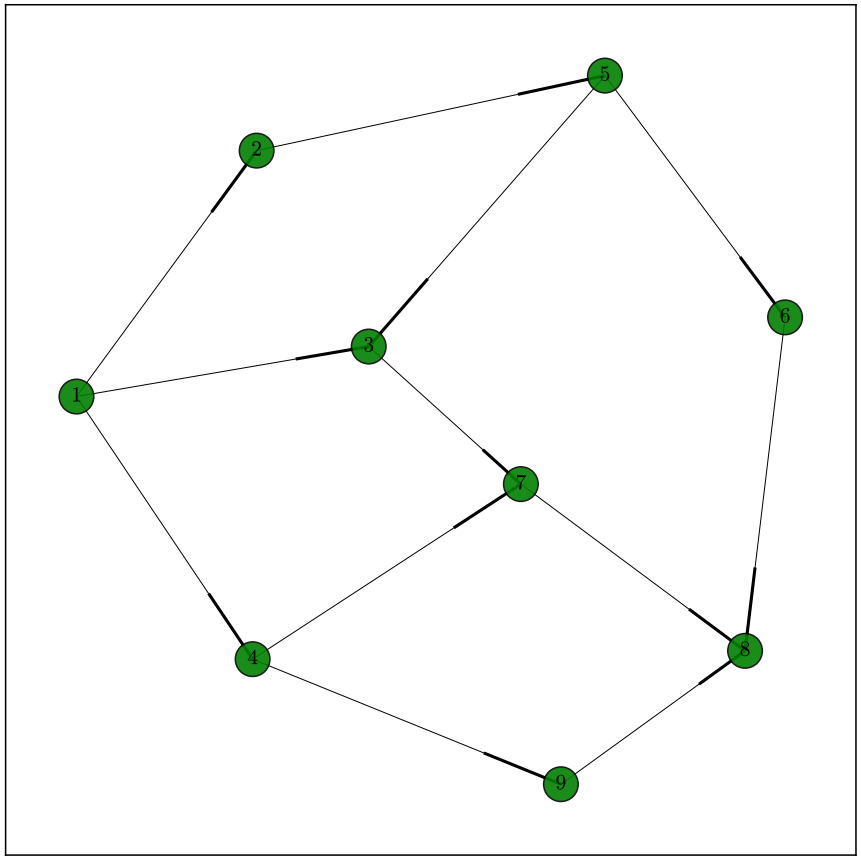


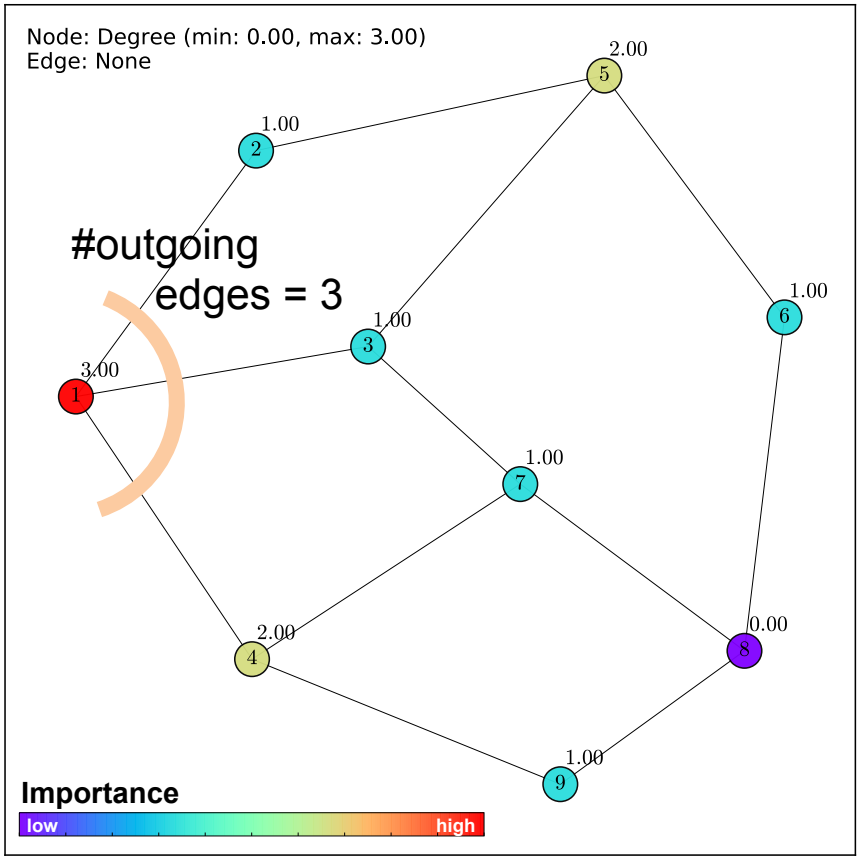






Degree centrality





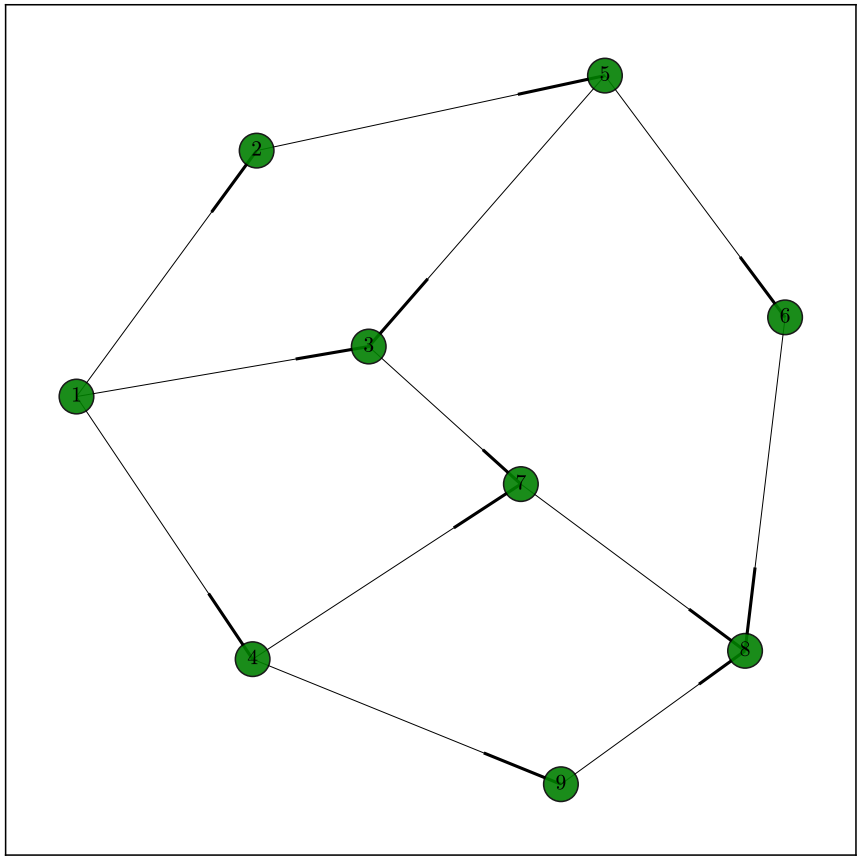
Input digraph

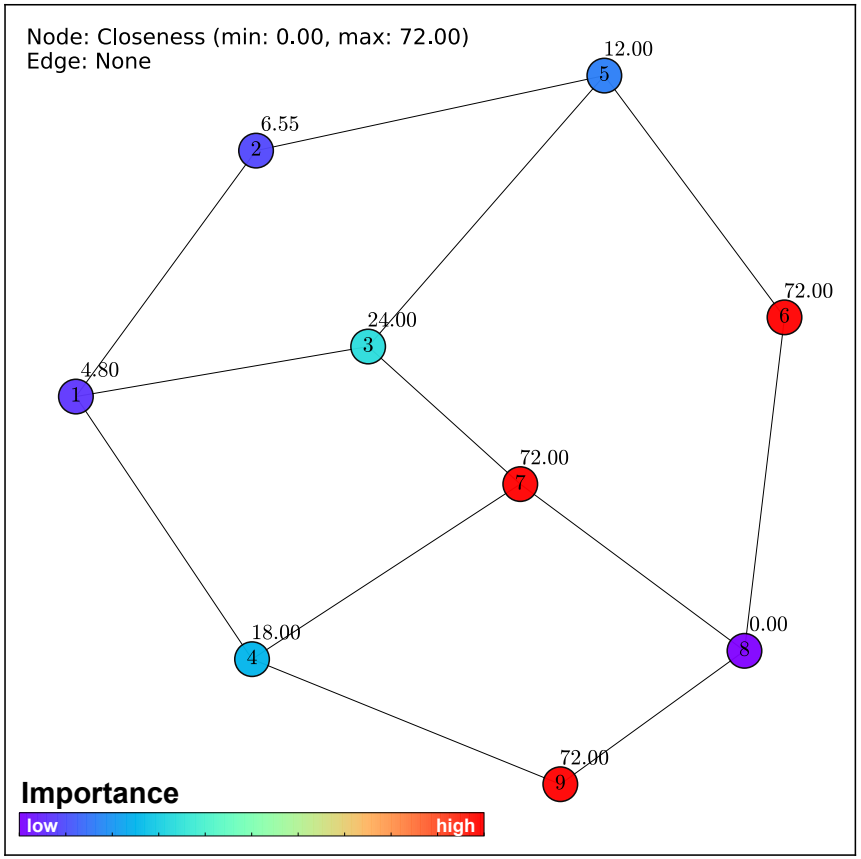
- 9 nodes and 12 edges

Degree centrality

- Number of outgoing edges

Closeness centrality





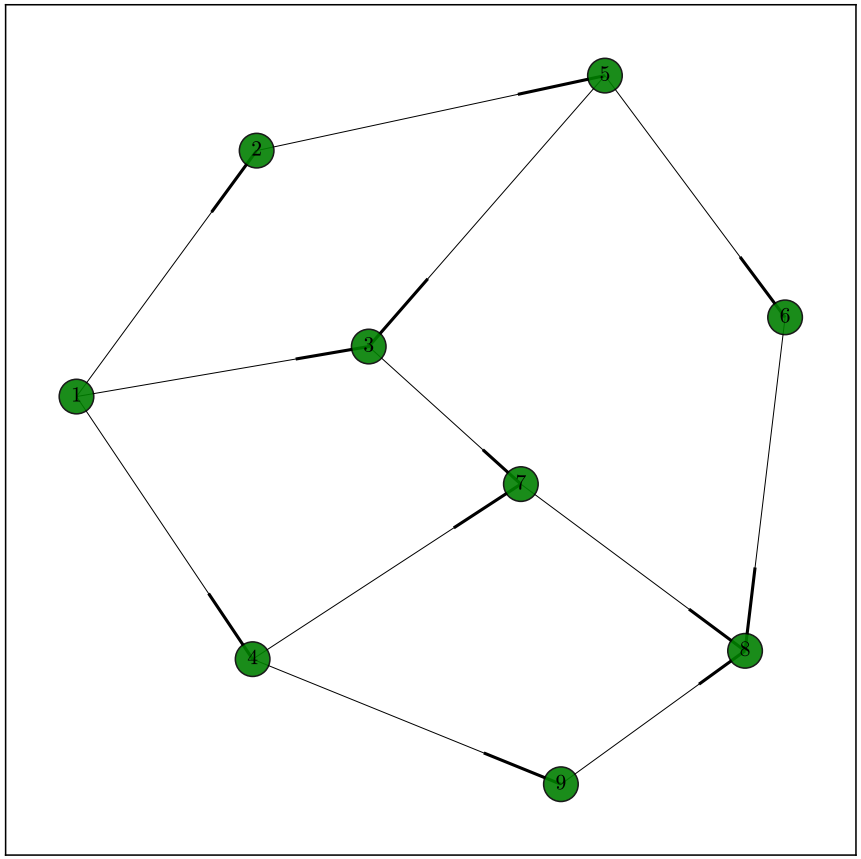
Input digraph

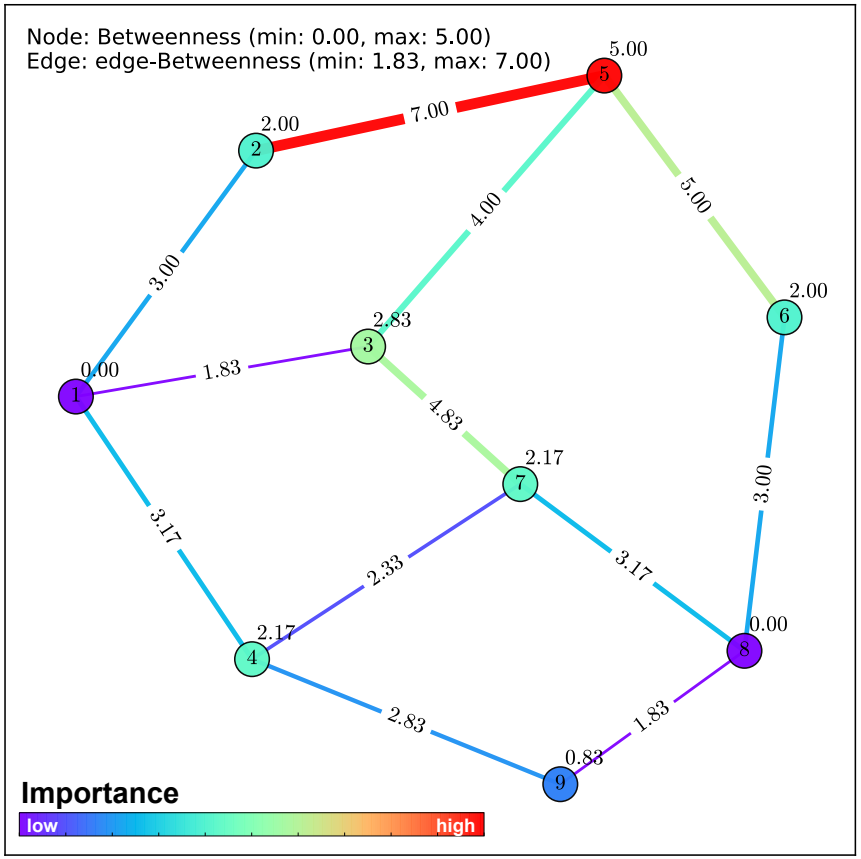
- 9 nodes and 12 edges

Closeness centrality

$$C_C(v) = \frac{n(n-1)}{\sum_{t \in V} d_G(v,t)}$$

Betweenness centrality





Input digraph

- 9 nodes and 12 edges

Betweenness centrality

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Appendix for ULIBC

TUTORIAL

ULIBC on Virtual- and Real-Machine

- ULIBC
  - Detects processor topology at runtime
  - Constructs CPU and memory affinities automatically
  - Available at https://bitbucket.org/yuichiro_yasui/ulibc

- How to build
  - on Virtual Machine (CO@Work)

```
$ make CC=gcc-5 ULIBCDIR=./ULIBC_v1.10_dummy/
```

  - on Real Machine

```
$ make CC=gcc-5 ULIBCDIR=./ULIBC_v1.10/
```

Our implementations for Graph500 and Green Graph500

- Multiple nodes and multiple processes
  - Multiple CPUs (K computer, TSUBAME 2.0, Fujitsu FX10, TSUBAME KFC)
    - 38621.4GTEPS, 1st position in 10th Graph500 list
  - Multiple CPUs and GPUs (TSUBAME 2.0 & TSUBAME KFC)
    - 317GTEPS, 4th position in 4th Graph500 list
    - 6.72GTEPS/kW, 1st position in 2nd Green Graph500 list (Big)
- Single node and single process
  - NUMA optimized and multiple threads (SGI Altix UV2000, 4-way Intel Xeon server)
    - 174.7GTEPS, 40th position in 10th Graph500 list (world fastest implementation for single node
  - Multiple threads and power efficient (Android tablet, Android based smart phone)
    - 153.2 GTEPS/kW, 1st position in 2nd Green Graph500 list (Small)

2D Hybrid BFS [Beamer, ’13]

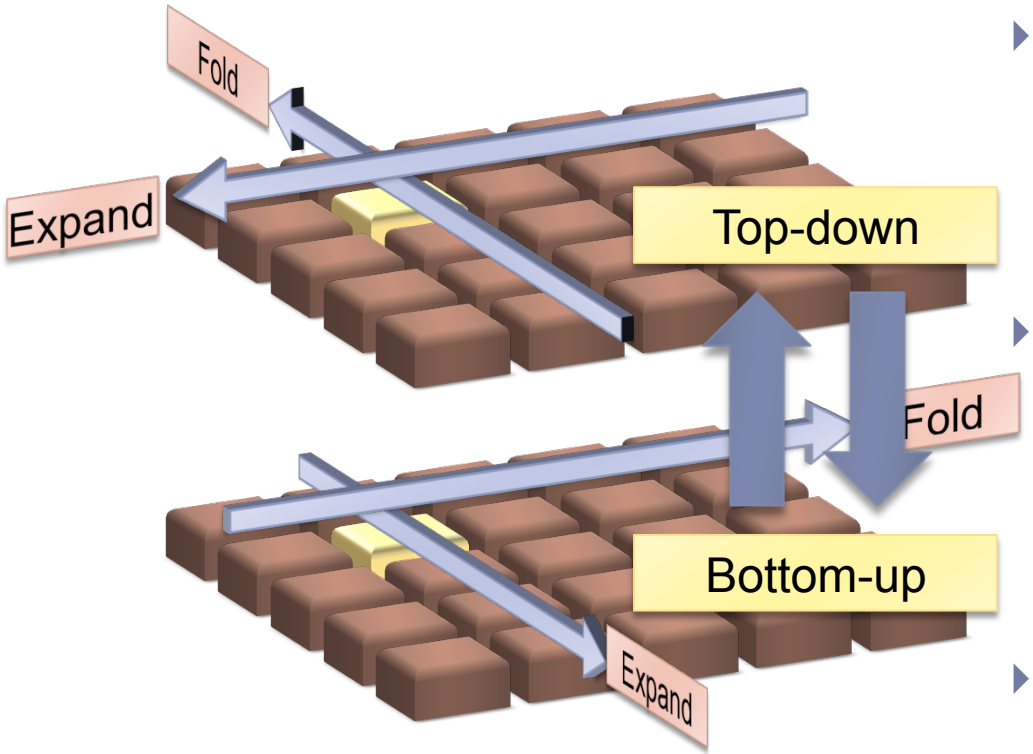
- 2D partitioning of the adjacency matrix for the graph

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,C} \\ \vdots & \ddots & \vdots \\ A_{R,1} & \cdots & A_{R,C} \end{pmatrix}$$

- Each partitioned region is assigned to a processor places on the 2D mesh.
  - Locality is better in 2D partitioning
  - 1D partitioning requires all-to-all communication among all the processes.
- Our implementation is based on this method.

[Beamer, ’13] Scott Beamer, et. al. Distributed Memory Breadth-First Search Revisited: Enabling Bottom-Up Search. IPDPSW '13.

Efficient Hybrid Search with 2D Partitioning

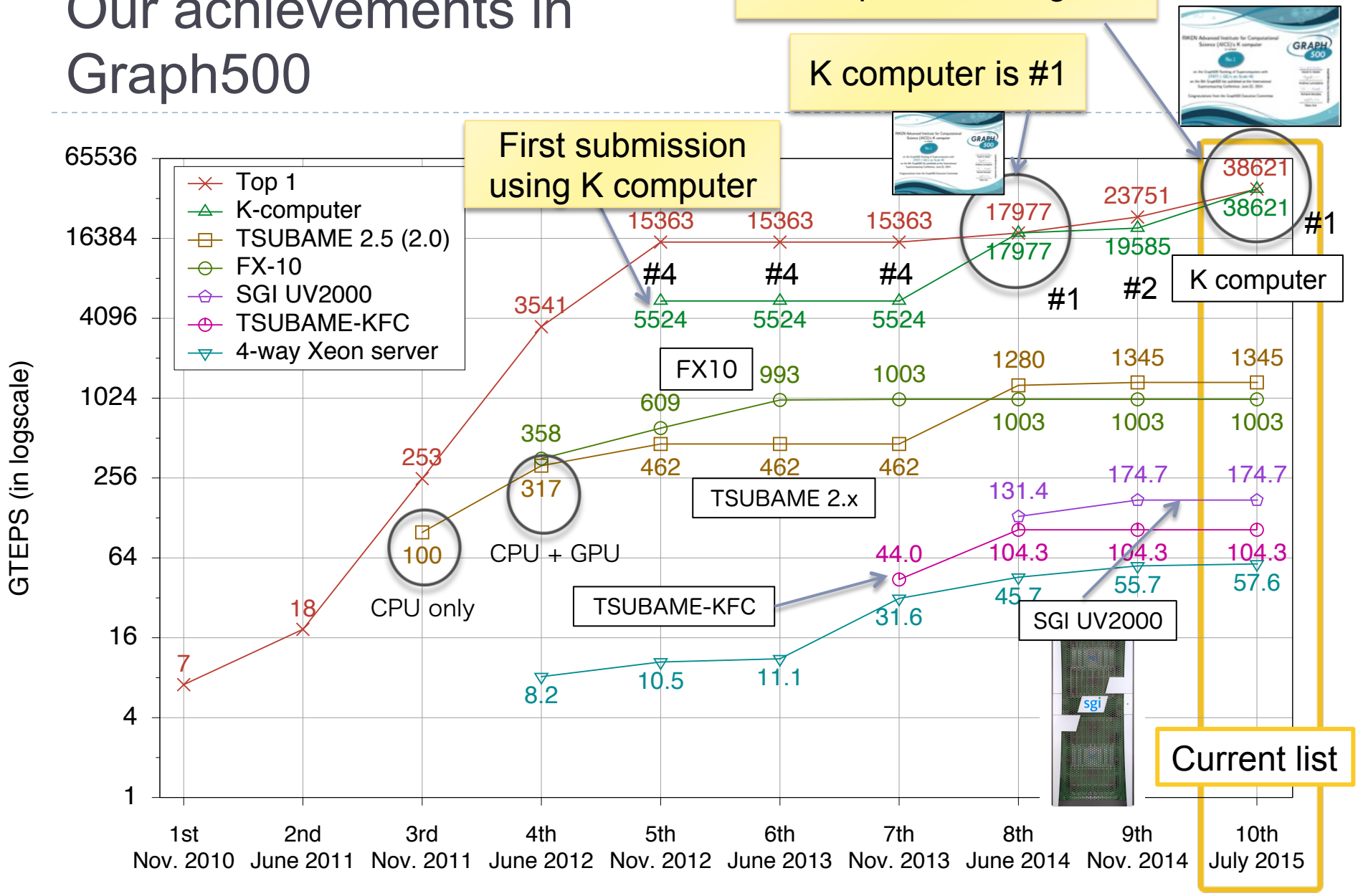


: Graph data owned by a single processor

Sharing the same graph data between two direction search.

- Our approach is based on the Top-down and bottom-up hybrid search BFS. [Beamer2011, 2012]
- We realized the hybrid search without any increase of memory footprint.
  - Graph data is shared between two search directions.
- Overlapped communication with computation.
  - Both top-down and bottom-up utilize overlapping communication.

Our achievements in Graph500



The Graph500

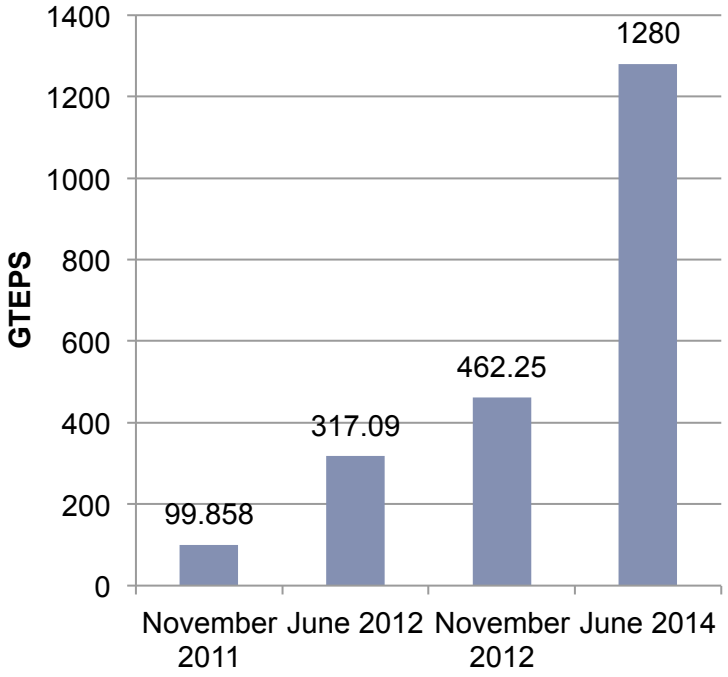
K Computer and TSUBAME 2.0 & 2.5

Graph500 ranking history for TSUBAME2.0 and 2.5

List	Rank	GTEPS	Implementation
November 2011	4	99.858	Top-down only
June 2012	4	317.09	CPU+ GPU
November 2012	20	462.25	CPU + GPU
July 2015	12	1345	**Efficient hybrid**

*Every score is obtained using TSUBAME2.0 1366 nodes or TSUBAME 2.5 1024 nodes

BFS performance on TSUBAME2.0 and 2.5

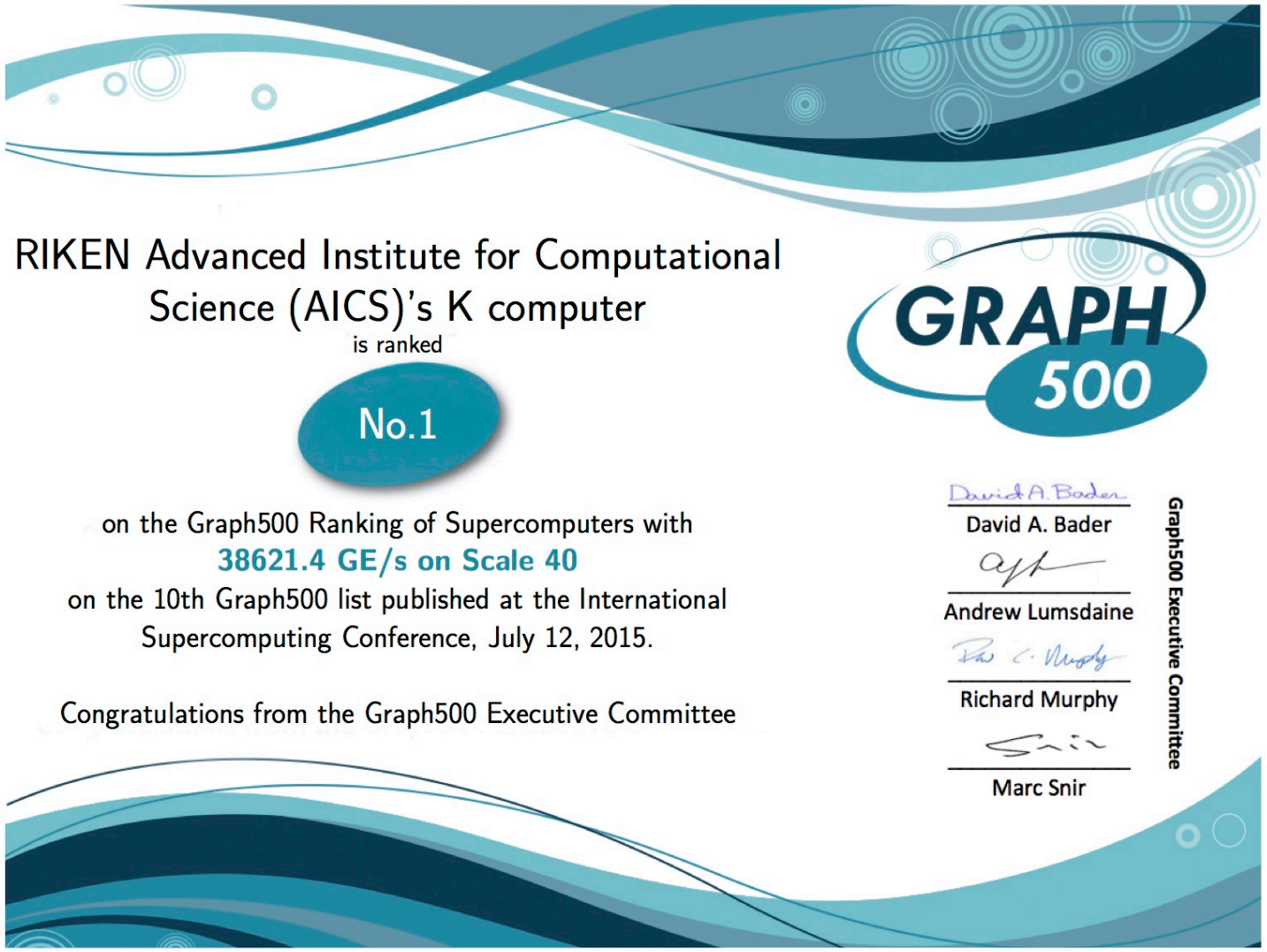


Graph500 ranking history for K Computer

List	Rank	GTEPS	Implementation
November 2013	4	5524.12	Top-down only
June 2014	1	17977.05	Efficient hybrid
July 2015	1	38632.4	Efficient hybrid & Zero-degree suppression & Vertex sorting

The 10th Graph500 List : The winner is K computer

Scale 40 : 38621.4 GTEPS (July 2015)



We have published a press release about Graph500 benchmark :
2015/07/14 : http://www.riken.jp/en/pr/topics/2015/20150715_1/

News

Previous Index Next

July 15, 2015

K computer takes first place in Graph 500 supercomputer ranking

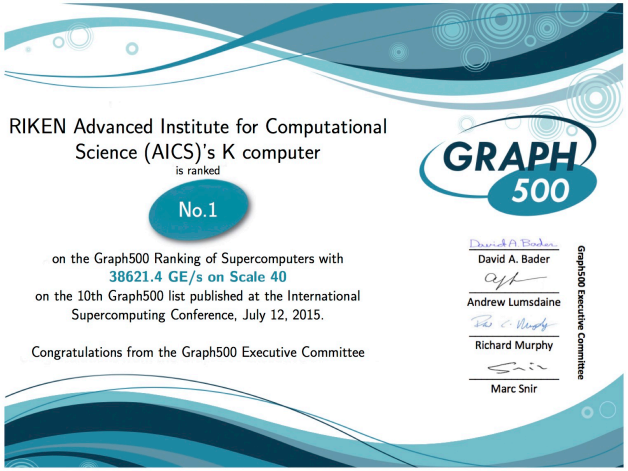
A collaboration between RIKEN, the Tokyo Institute of Technology, University College Dublin, Kyushu University, and Fujitsu has again won top place for the K computer in the June 2015 Graph 500 supercomputer ranking, demonstrating the Japanese supercomputer's prowess in the area of data-intensive processing. The results were announced on July 13 at the international conference on high-performance computing (ISC2015) in Frankfurt, Germany.

The Graph 500 ranking is a relatively new benchmark, first issued in 2010, which seeks to gauge the ability of supercomputers on data-intensive loads rather than simple speed, with the goal of improving computing involving complex data problems in areas such as cybersecurity, medical informatics, data enrichment, social networks,

News & Media
Press Center
Press Releases
News
2015
2014
2013
2012
2011
2010
2009
2008
2007
2006

The 10th Graph500 List : The winner is K computer

Scale 40 : 38621.4 GTEPS (July 2015)




```
============ Result ==============
SCALE:                          40
edgefactor:                     16
NBFS:                           64
graph_generation:               197.379
num_mpi_processes:              82944
construction_time:              609.395
min_time:                       0.395321105141
firstquartile_time:             0.409624118358
median_time:                    0.455501377815
thirdquartile_time:             0.566996186739
max_time:                       1.95167534612
mean_time:                      0.562004323256
stddev_time:                    0.311699826145
min_nedge:                      1.7592103987e+13
firstquartile_nedge:            1.7592103987e+13
median_nedge:                   1.7592103987e+13
thirdquartile_nedge:            1.7592103987e+13
max_nedge:                      1.7592103987e+13
mean_nedge:                     1.7592103987e+13
stddev_nedge:                   0
min_TEPS:                       9.01384752422e+12
firstquartile_TEPS:             3.10268470903e+13
median_TEPS:                    3.86214067477e+13
thirdquartile_TEPS:             4.29469437914e+13
max_TEPS:                       4.45007963348e+13
harmonic_mean_TEPS:             3.1302435335e+13
harmonic_stddev_TEPS:           2.18728188393e+12
min_validate:                   43.201660905
firstquartile_validate:         43.4925568579
median_validate:                44.3293765394
thirdquartile_validate:         45.4055157886
max_validate:                   50.040661654
mean_validate:                  44.6539914012
stddev_validate:                1.39768976422
```

Green Graph 500

Kyushu University's GraphCREST-SandybridgeEP-2.4GHz

is ranked

No. 1

in the **Big Data** category of the Green Graph 500 Ranking of Supercomputers with

62.93 MTEPS/W on Scale 30

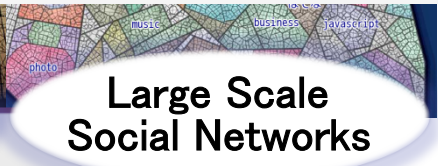
on the fifth Green Graph 500 list published at the International Supercomputing Conference, July 13, 2015.

Congratulations from the Green Graph 500 Chair

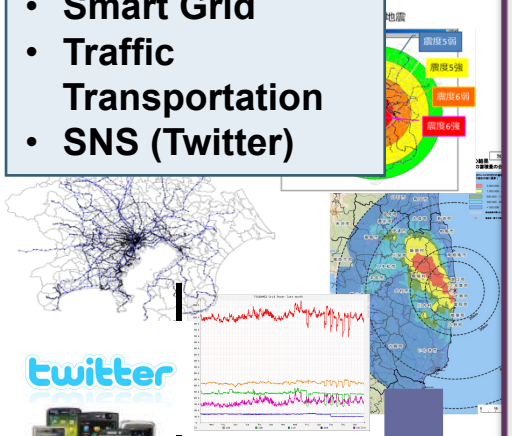
Torsten Hoefler

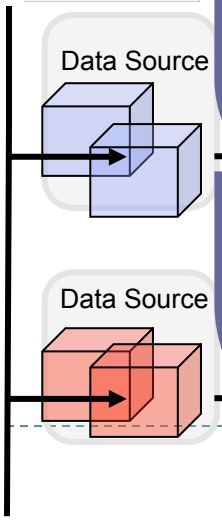
GreenGraph500 Chair

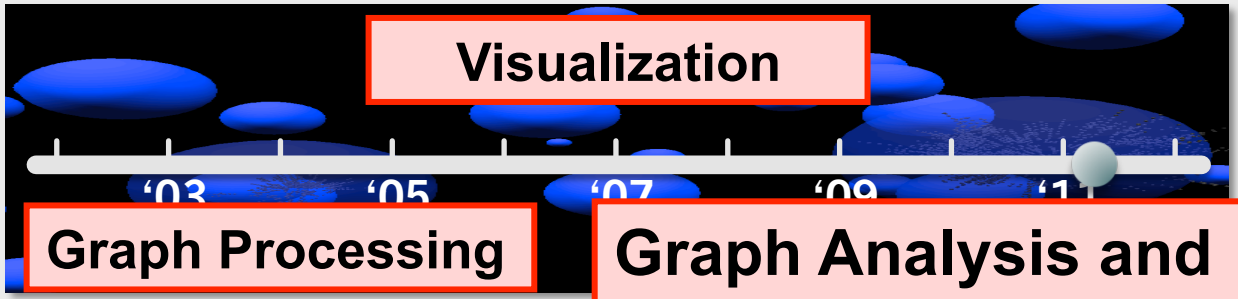
Extremely Large-scale Graph Analysis System



Large Sensor
- **Monitoring Data**
- **Smart Grid**
- **Traffic Transportation**
- **SNS (Twitter)**







Graph Processing

Graph Analysis and Optimization Library

Centrality | **Shortest Path** | **Connected Component**

Indexing | **Clustering** | **Page Rank** | **Mathematical Optimization**

Streaming Processing System

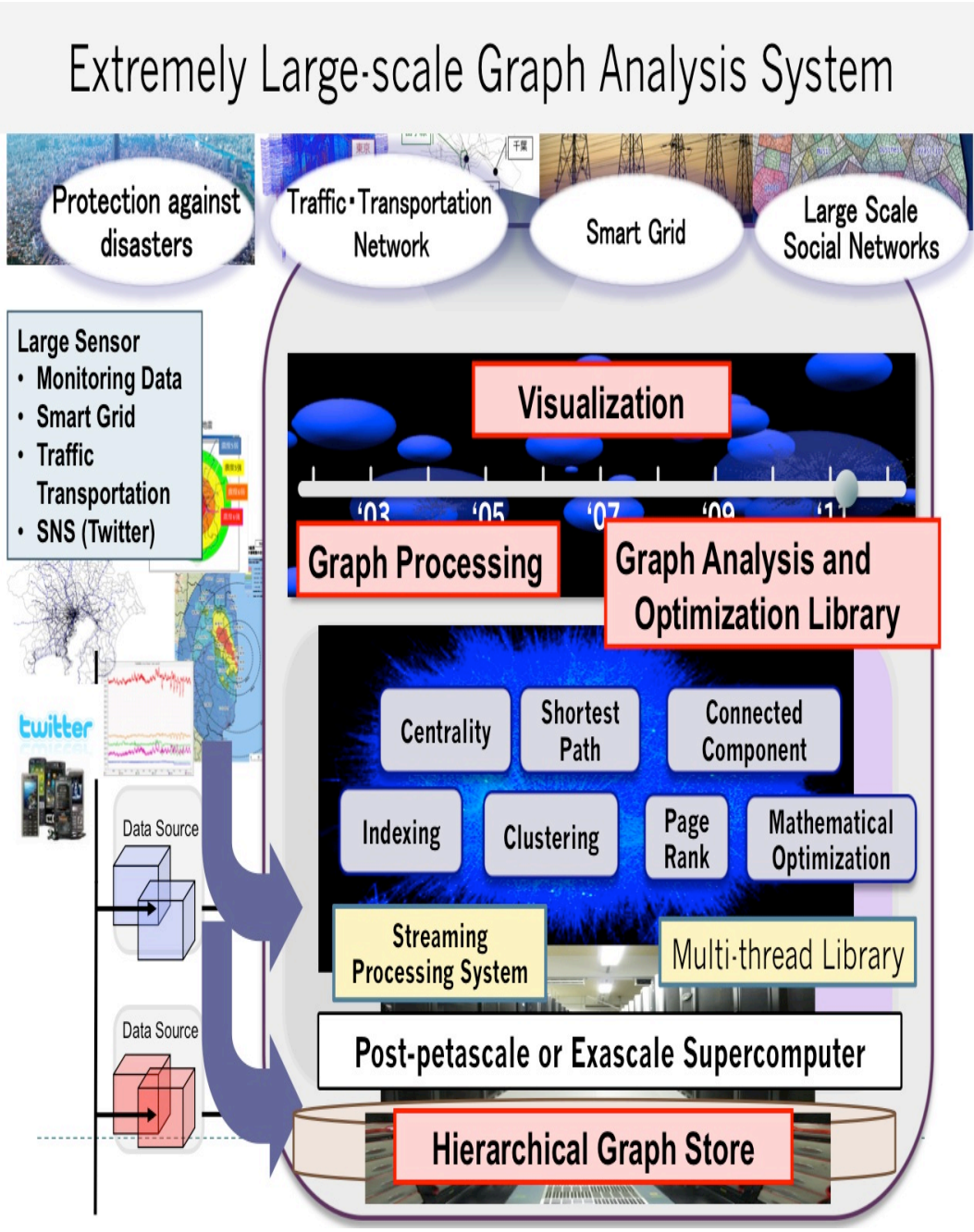
Multi-thread Library

Post-petascale or Exascale Supercomputer

Hierarchical Graph Store

Software stacks for an extremely large-scale graph analysis system

- **Hierarchal Graph Store:**
- Utilizing emerging NVM devices as extended semi-external memory volumes for processing extremely large-scale graphs that exceed the DRAM capacity of the compute nodes
- Design highly efficient and scalable data offloading techniques, PGAS-based I/O abstraction schemes, and optimized I/O interfaces to NVMs.
- **Graph Analysis and Optimization Library:**
- Perform graph analysis and search algorithms, such as the BFS kernel for Graph500, on multiple CPUs and GPUs. Implementations, including communication-avoiding algorithms and techniques for overlapping computation and communication, are needed for these libraries.
- Finally, we can make a BFS tree from an arbitrary node and find a shortest path between two arbitrary nods on extremely large-scale graphs with tens of trillions of nodes and hundreds of trillions of edges.
- **Graph Processing and Visualization:**
- We aim to perform an interactive operation for large-scale graphs with hundreds of million of nodes and tens of billion of edges.



Software Collections in GraphCREST

High-Performance General Solver for Large-scale Optimization Problems

SDPARA is a parallel implementation on multiple CPUs and GPUs for solving extremely large-scale **Semidefinite programming problems**. SDPARA can also perform parallel Cholesky factorization using thousands of GPUs and techniques to overlap computation and communication. SDPARA also achieved **1.713 PFlops** in double precision for large-scale Cholesky factorization using **4,080 GPUs on TSUBAME 2.5 supercomputer.**

`http://www.graphcrest.jp/eng/`

ScaleGraph Library

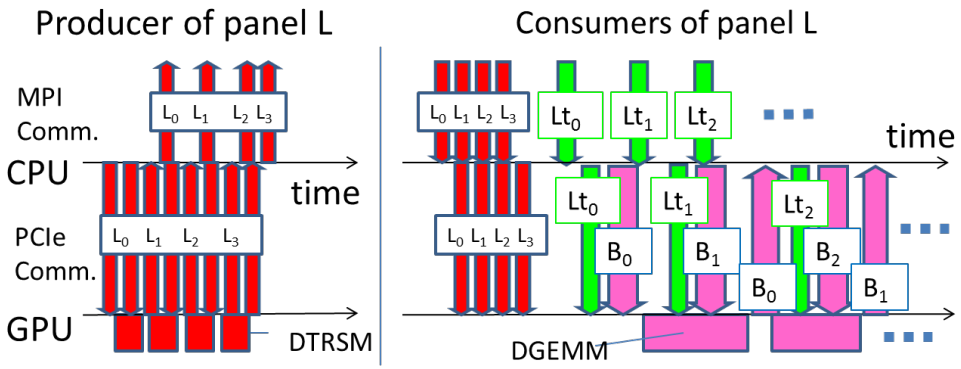
Highly Scalable Large Scale Graph Analytics Library beyond the scale of billions of vertices and edges on Distributed Systems

- Based on our extended X10
  - X10 is a new parallel distributed programming language.
- Fully utilizing MPI collective communication
- Native support for hybrid (MPI and multi-threading) parallelism

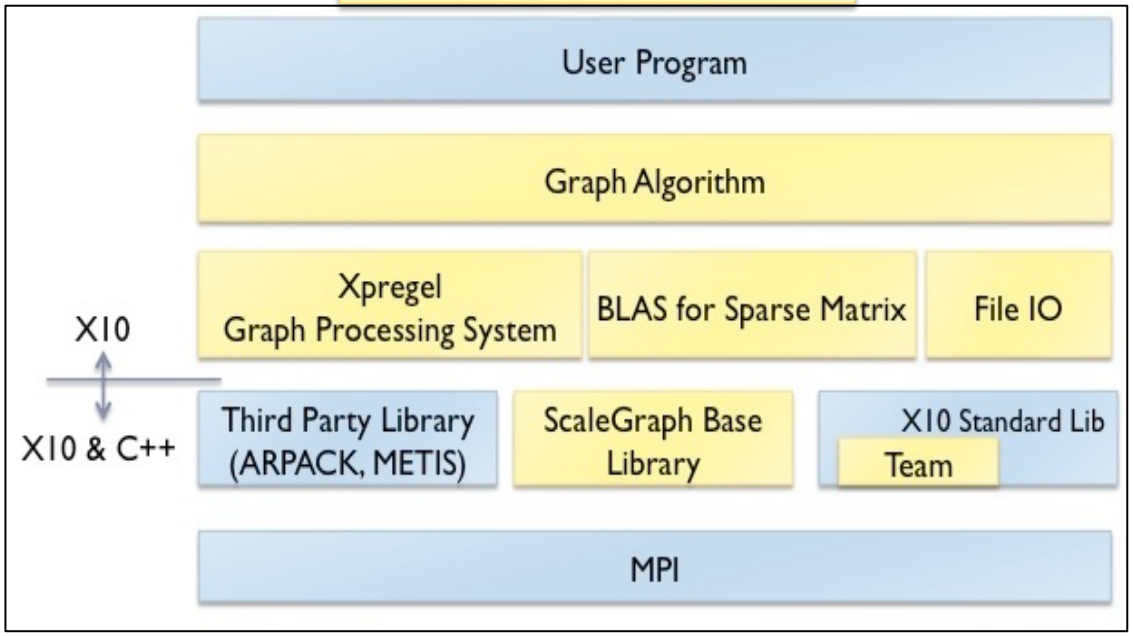
`http://www.scalegraph.org/`

Parallel Algorithm of Cholesky Factorization

GPU computation, PCI-e communication, and MPI communication are overlapped



Software stack

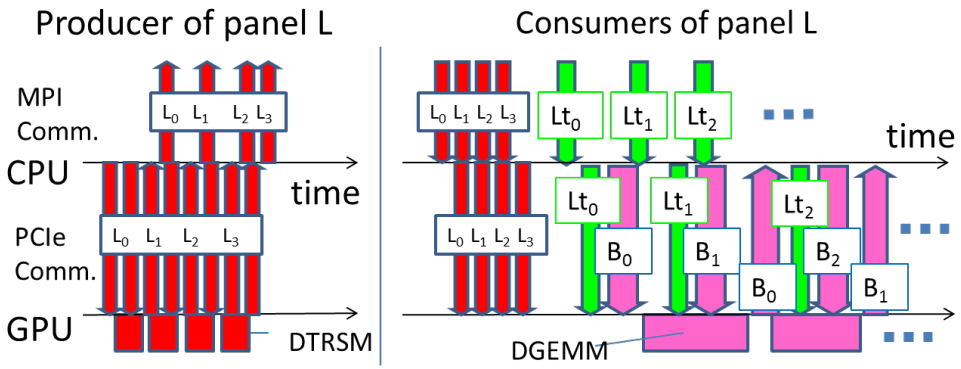


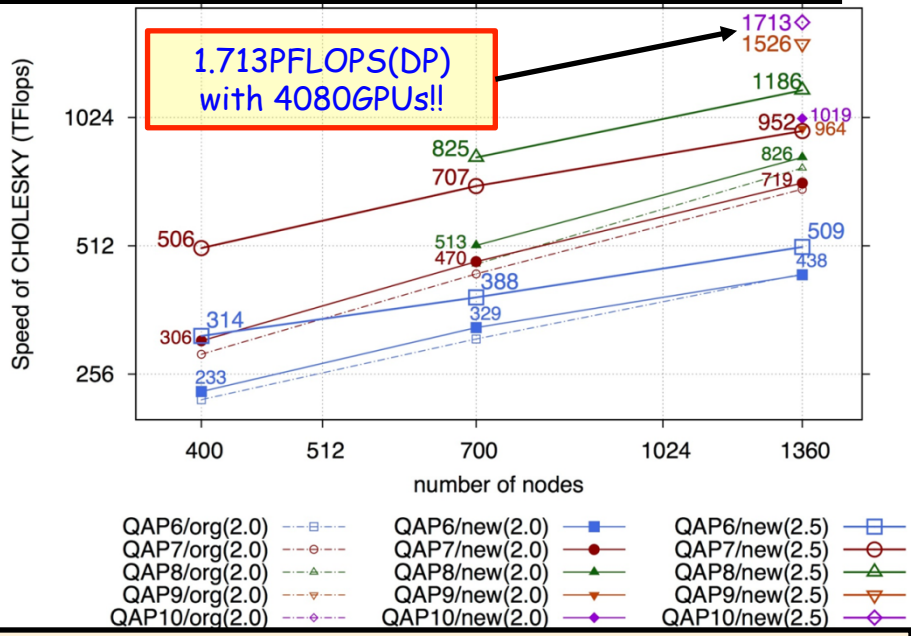
High-Performance General Solver for Extremely Large-scale Semidefinite Programming Problems

1. Mathematical Programming : one of the most important mathematical programming
2. Many Applications : combinatorial optimization, control theory, structural optimization, quantum chemistry, sensor network location, data mining, etc.

Parallel Algorithm of Cholesky Factorization

GPU computation, PCI-e communication, and
MPI communication are overlapped

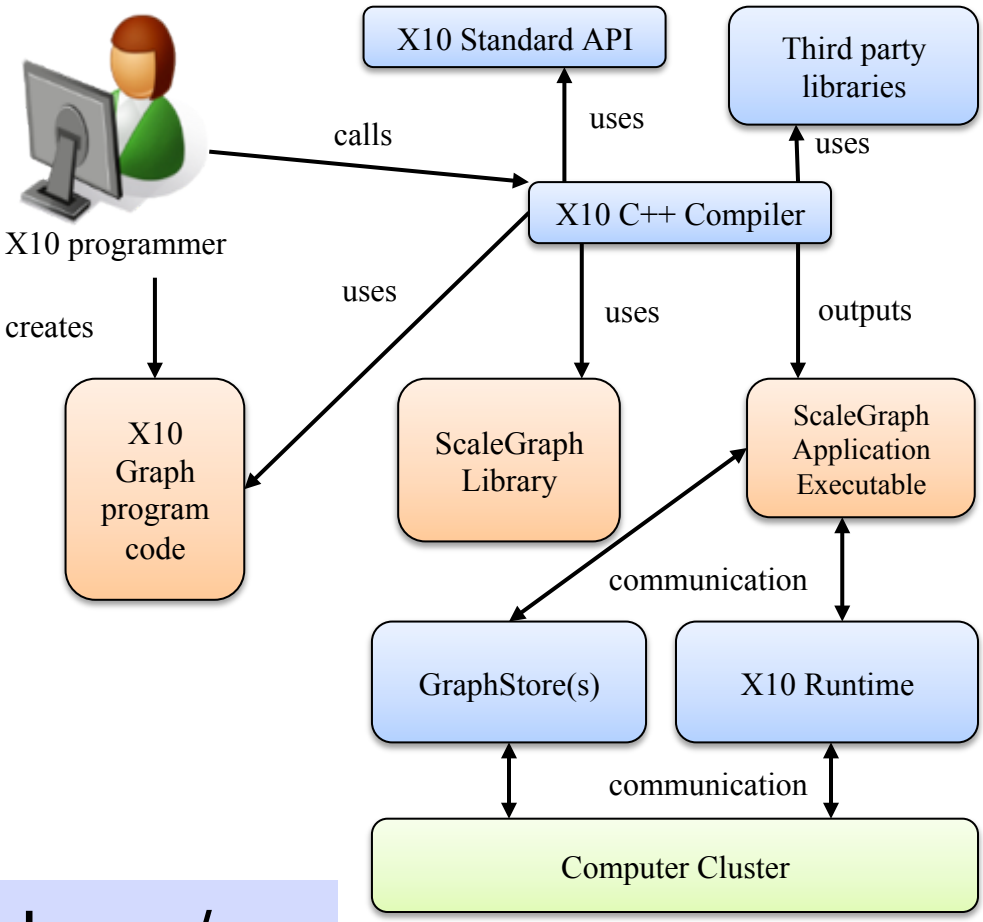




- SDPARA is a parallel implementation of the interior-point method for Semidefinite Programming Parallel computation for two major bottlenecks
  - ELEMENTS ⇒ Computation of Schur complement matrix (SCM)
  - CHOLESKY ⇒ Cholesky factorization of Schur complement matrix (SCM)
- SDPARA could attain high scalability using 16,320 CPU cores on the TSUBAME 2.5 supercomputer and some techniques of processor affinity and memory interleaving when the computation of SCM (ELEMENTS) constituted a bottleneck.
- With 4,080 NVIDIA GPUs on the TSUBAME 2.0 & 2.5 supercomputer, our implementation achieved 1.019 PFlops(TSUBAME 2.0) & 1.713PFlops(TSUBAME 2.5) in double precision for a large-scale problem (CHOLESKY) with over two million constraints.

ScaleGraph : Large-Scale Graph Analytics Library

- **Aim** - Create an open source **X10-based Large Scale Graph Analytics Library** beyond the scale of billions of vertices and edges.
- **Objectives**
  - To define concrete abstractions for Massive Graph Processing
  - To investigate use of X10 (I.e., PGAS languages) for massive graph processing
  - **To support significant amount of graph algorithms (E.g., structural properties, clustering, community detection, etc.)**
  - To create well defined interfaces to Graph Stores
  - To evaluate performance of each measurement algorithms and applicability of ScaleGraph using real/synthetic graphs in HPC environments.



URL: http://www.scalegraph.org/

Graph Analysis & High-Performance Computing Techniques for Realizing Urban OS

Katsuki Fujisawa

Hisato Matsuo

KYUSHU UNIVERSITY

Urban Issues/Challenges

Traffic Congestion

Energy Supply-Demand Balance

- Traffic problems
- Means of mobility
- Information access

Information Flood

- Energy issues
- Environment, water, sanitation & hygiene
- Disaster control

Environment Preservation

- Finance
- Industrial promotion
- Gov. administration
- Education/Welfare
- Innovation

- Globalization
- Diversity
- Information divide

- Population decline
- Low birth rate & rapid aging
- Urban concentration

Diversified Society

Aging

Depopulation

Decline of Community

Disaster Response

Urban OS provides three Mobility's

Anyone can access … anytime, anywhere

Three Mobility's lead sustainable society

Improve the citizens' QOL and activity

Vitalize community by citizens' participation and autonomy

Cyclic and low carbon society

Smart Mobility (People/Goods Mobility)

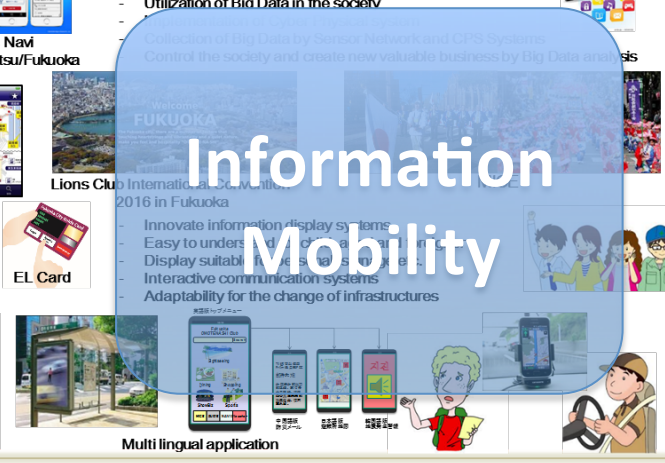
- Seamless service from mass transportation to personal
- Care
- Co
- O
- C
- F
- P
- A

People/ Material Mobility

Ubiquitous (Information Mobility)

- Utilization of Big Data in the society
- Innovate information display systems
- Easy to understand
- Display suitable
- Interactive communication systems
- Adaptability for the change of infrastructures

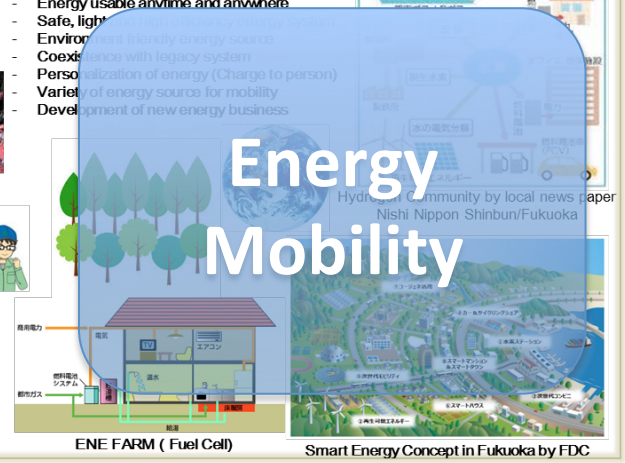
Information Mobility



Smart Energy (Energy Mobility)

- Energy usable anytime and anywhere
- Safe, ligh
- Environ
- Coexis
- Perso
- Variet
- Devel

Energy Mobility



Make city operation efficient and optimize

Create valuable community services by Big & Open Data

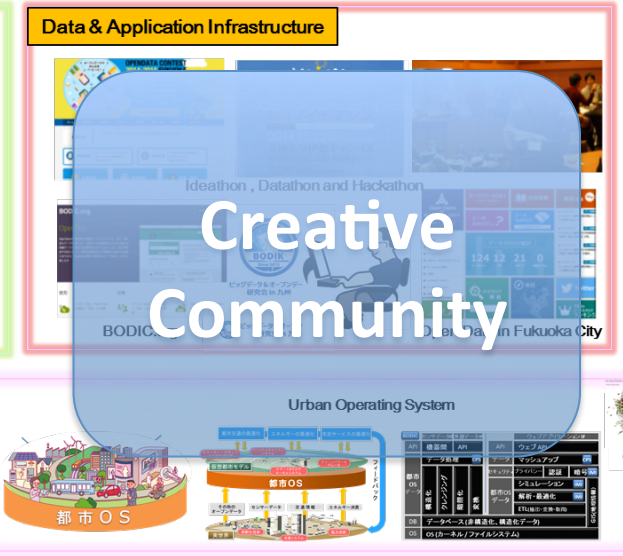
Use dispersion energy efficiently

Infrastructure for City design and operation

Efficient & optimized Infrastructure

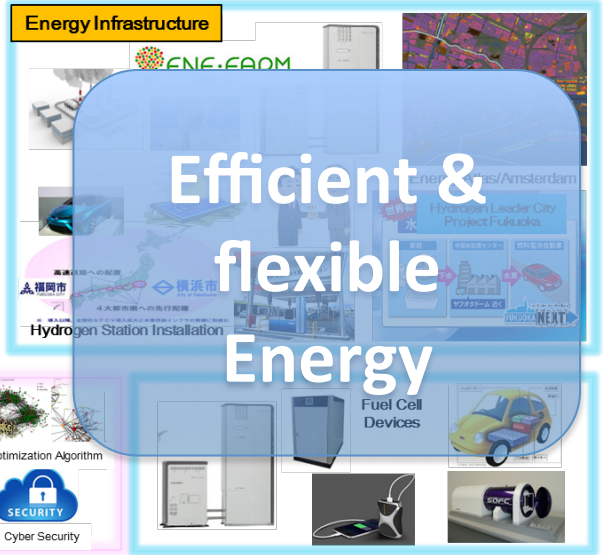
Data & Application Infrastructure

Creative Community



Energy Infrastructure

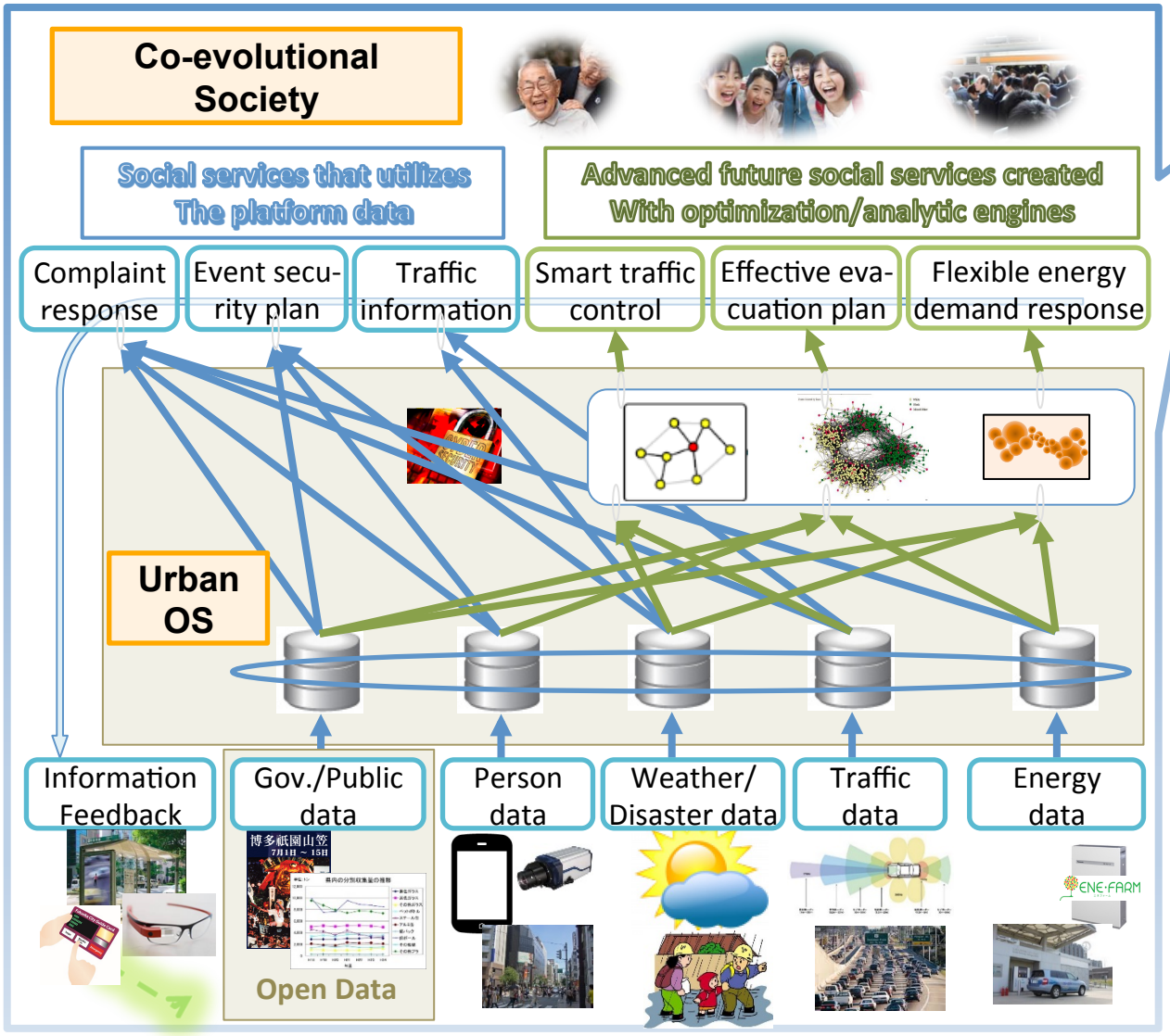
Efficient & flexible Energy



Software Sensor Device

Urban OS Functions

Open platform for advanced urban services



Application Service

Open platform for social/public/commercial applications

BODIC.org

Optimization/Analytic

Automatic optimization, control & bottleneck analysis

Data Store

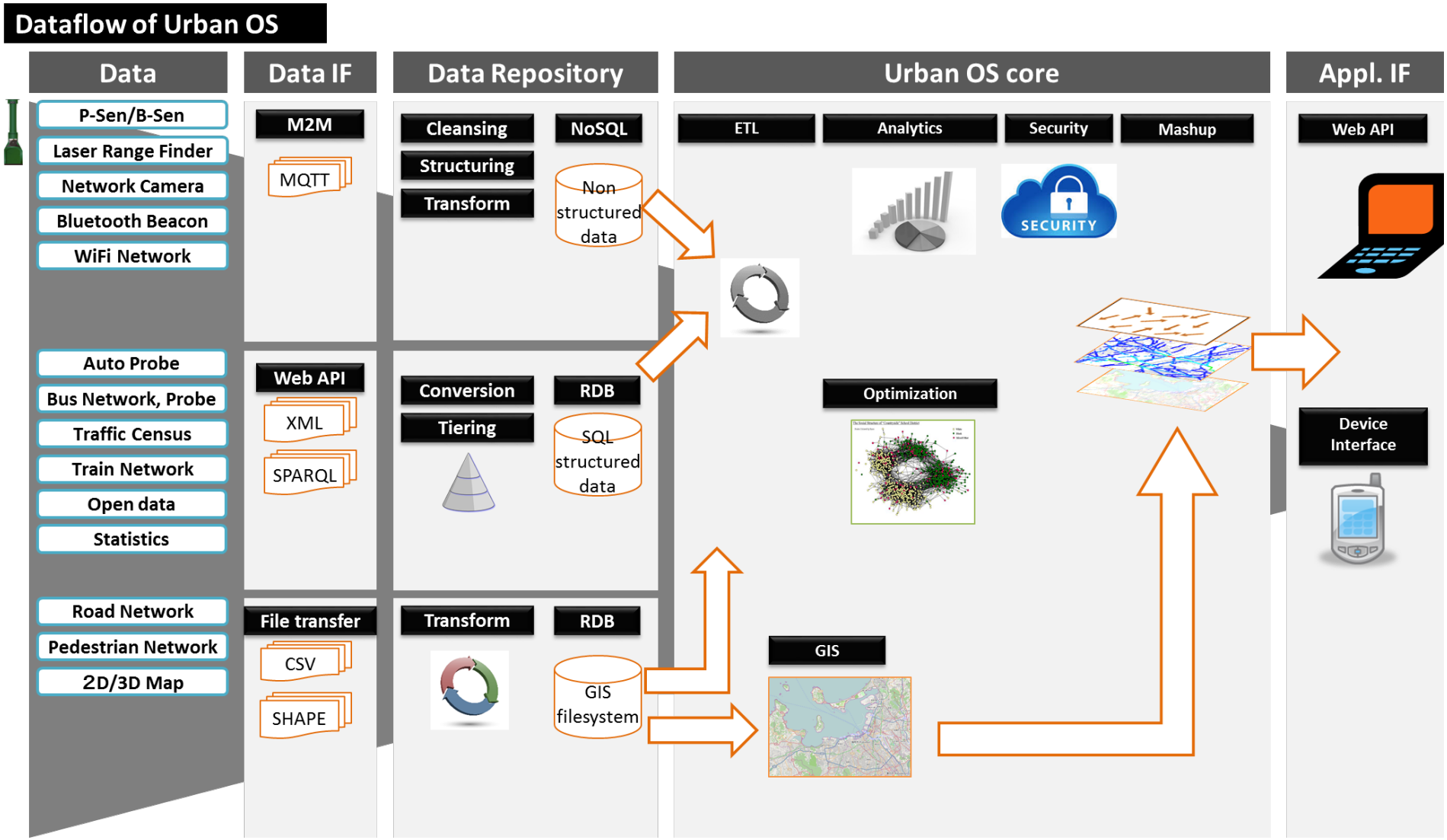
Cross-utilization of various data

BODIC.org

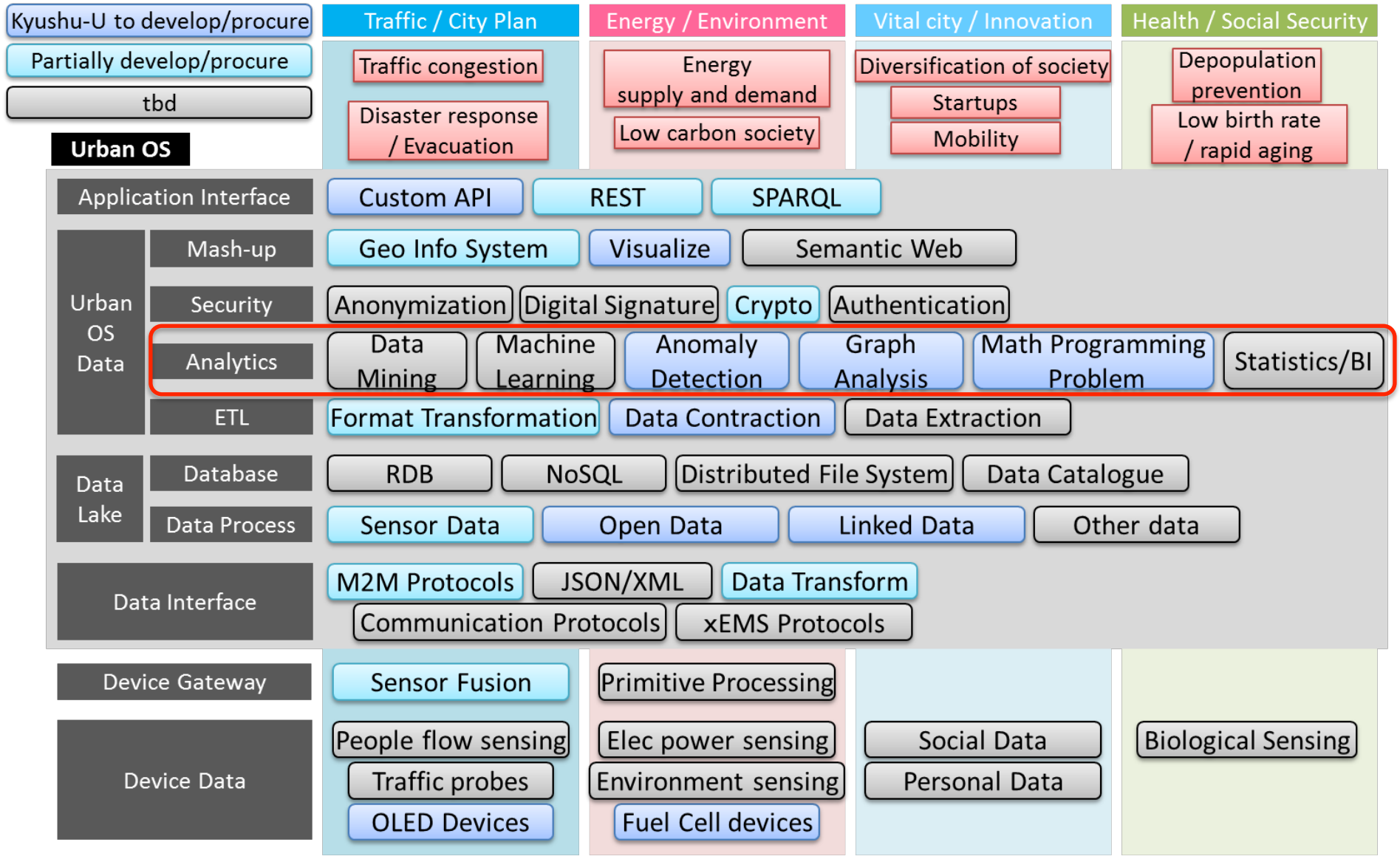
Data

Big data / Open data
Sensor Network

How data are processed in Urban OS



Required Technology for Urban OS



Real-time Emergency Evacuation Planning

- catastrophic disasters by massive earthquakes are increasing in the world, and disaster management is required more than ever

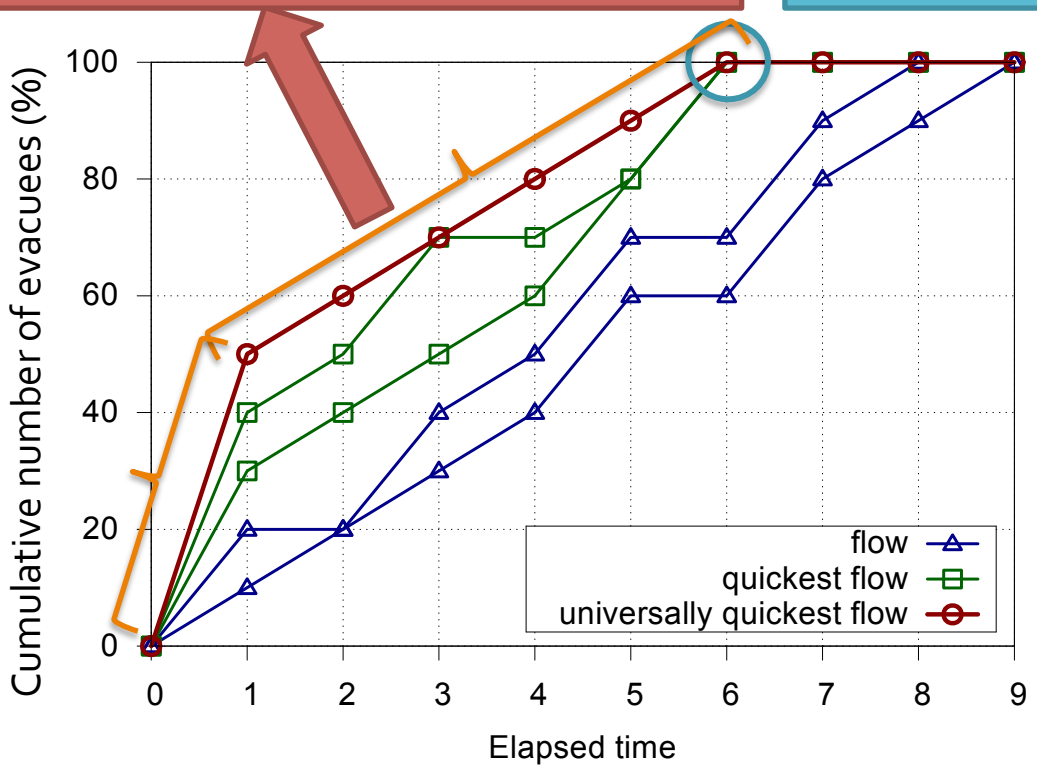
Universally Quickest Flow(UQF) → Not simulation But Optimization Problem

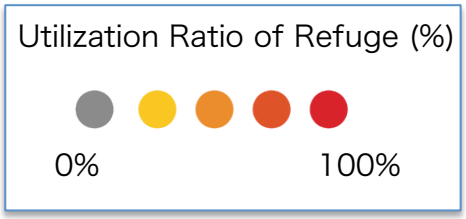
UQF simultaneously maximizes the cumulative number of evacuees at an arbitrary time.

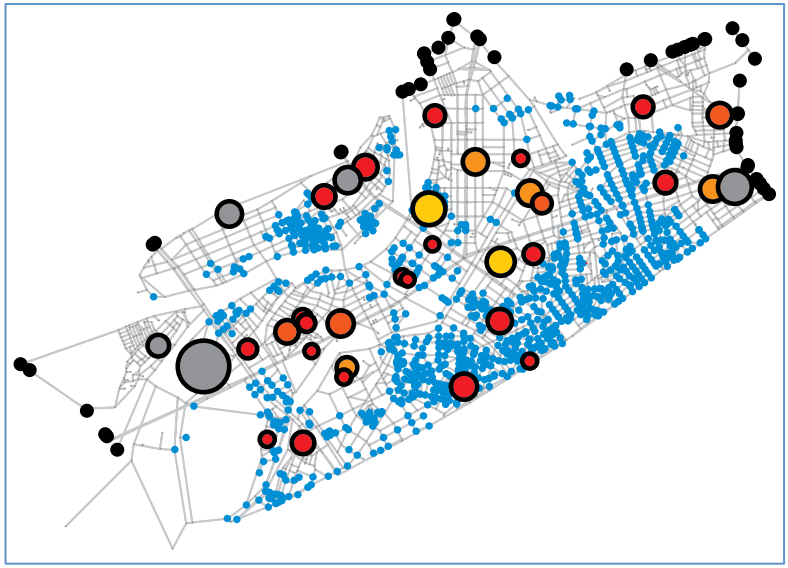
Evacuation planning can be reduced to UQF of a given dynamic network.

maximizes the cumulative number of evacuees

Quickest Evacuation

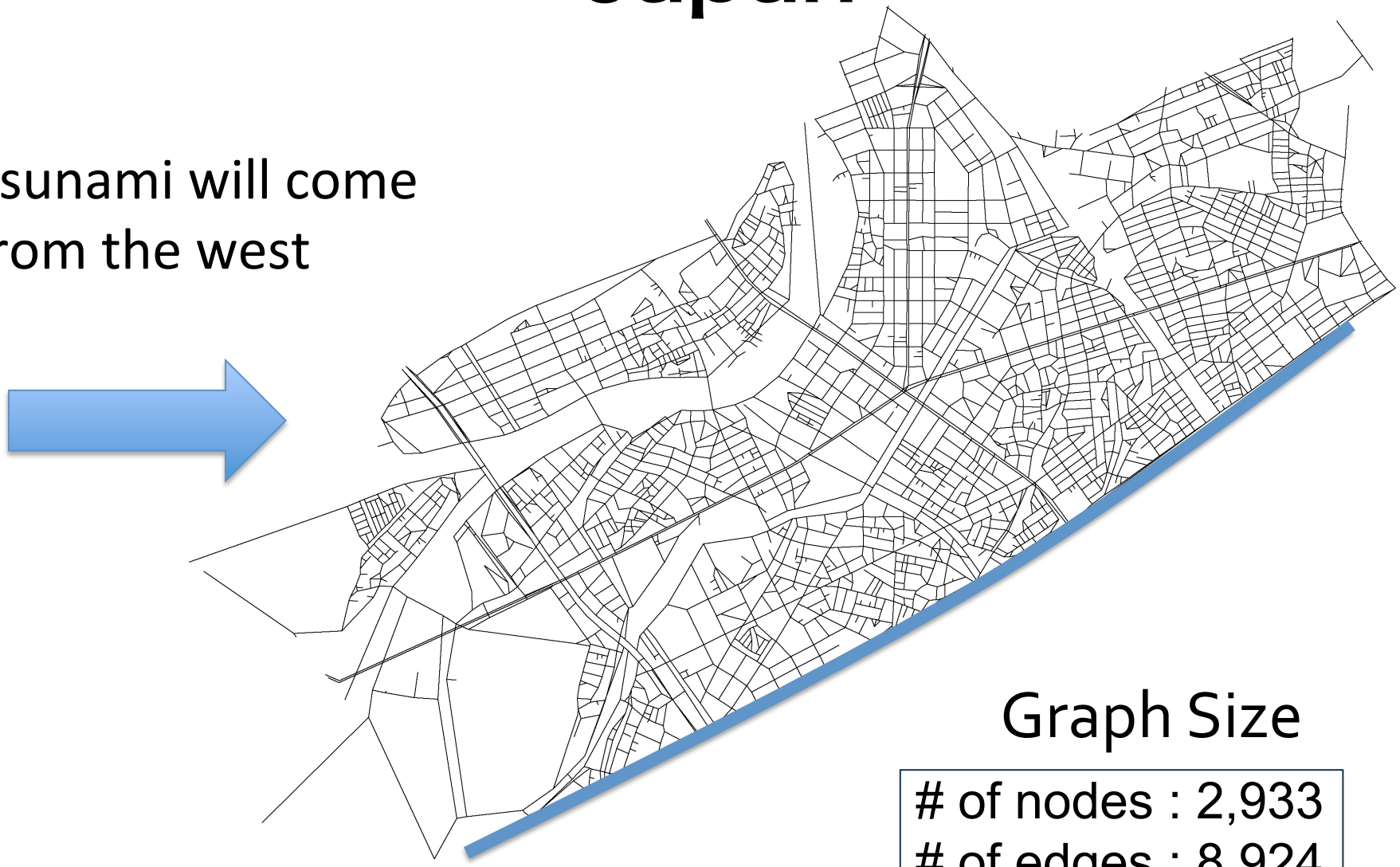






North area of Osaka City, Japan

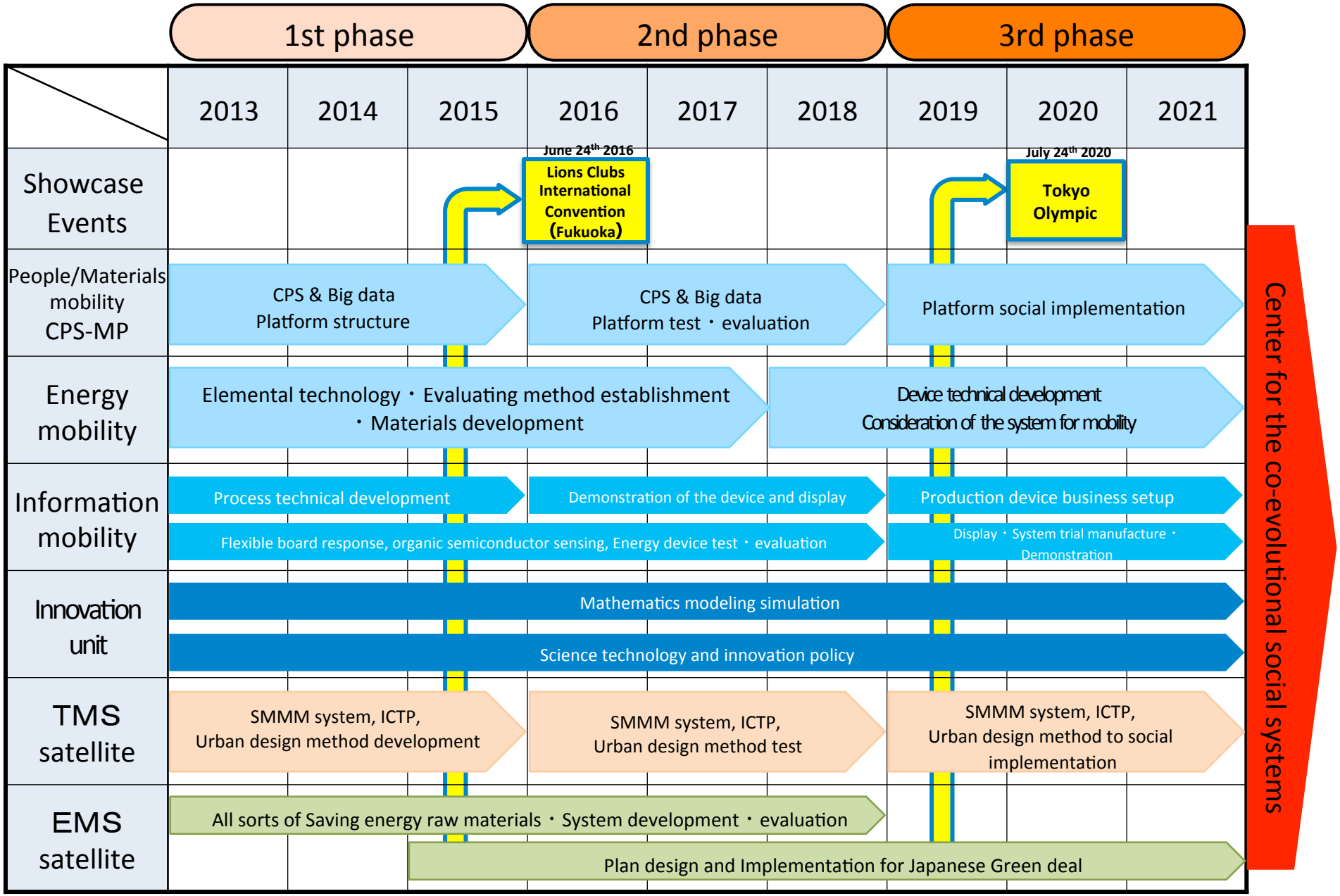
Tsunami will come from the west



Graph Size

# of nodes : 2,933
# of edges : 8,924

Project Schedule



Partnership

Kaiserslautern

Fukuoka

Yokohama

FUNAI

TOYOBO
Ideas & Chemistry

KONICA MINOLTA

保土谷化学

JX日鉱日石金属

KOMURA-TECH

JARI
一般財団法人日本自動車研究所
Japan Automobile Research Institute

Relations Inc.

一般財団法人 計量計画研究所

HITACHI
Inspire the Next

FUJITSU

ShinEtsu

NISSAN

エネルギー・フロンティア
TOKYO GAS

NEC

東京大学
THE UNIVERSITY OF TOKYO

MIZUHO
みずほ情報総研

HARIMA

THE NEW VALUE FRONTIER
KYOCERA

NTT

福岡市
FUKUOKA CITY

Nishitetsu

ISIT

ふくおか
IST

YNU
横浜国立大学

OPEN
YOKOHAMA

DENSO

Japan Display

くらしを考え、明日をつくる。
SHIZGAS
静岡ガス

esri ジャパン

TOSHIBA MACHINE
Best Partner of Leading Industries

HINO

Microsoft

Our Goal

- Urban OS as an open platform of data aggregation
  - big data / open data / sensor data / linked data
- Urban OS as an advanced optimization / analytic platform utilizing HPC based graph analysis experience
- Urban OS as an application platform to delightedly support start-ups.