UCSB

A Toolbox for High-Performance Graph Computation

John R. Gilbert
University of California, Santa Barbara

UC Berkeley ParLab
September 15, 2011

Support: Intel, Microsoft, DOE Office of Science, NSF

Team (2010-2011)

- PI: JRG
- UCSB PhD students: Adam Lugowski, Lijie Ren
- UCSB MS students: Yun Teng
- UCSB undergrads: Chris Lock, Drew Waranis
- Berkeley Lab: Aydın Buluç
- Microsoft Corporation: Steve Reinhardt, David Alber

- Research support: Intel Corporation, Microsoft Corporation, DOE Office of Science, NSF

UCSB

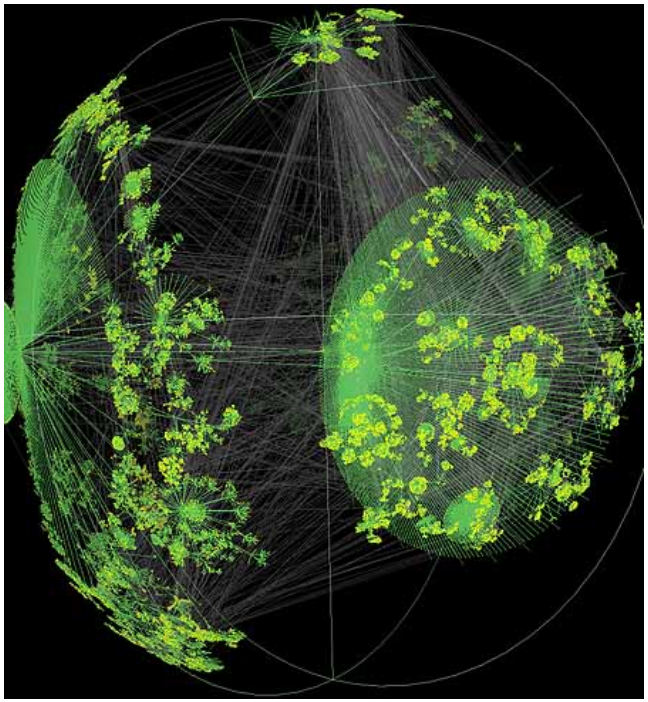
Outline

- Motivation
- Libraries: CombBLAS and KDT
- Algorithms
- Plans

UCSB

Large graphs are everywhere…

- Internet structure
- Social interactions



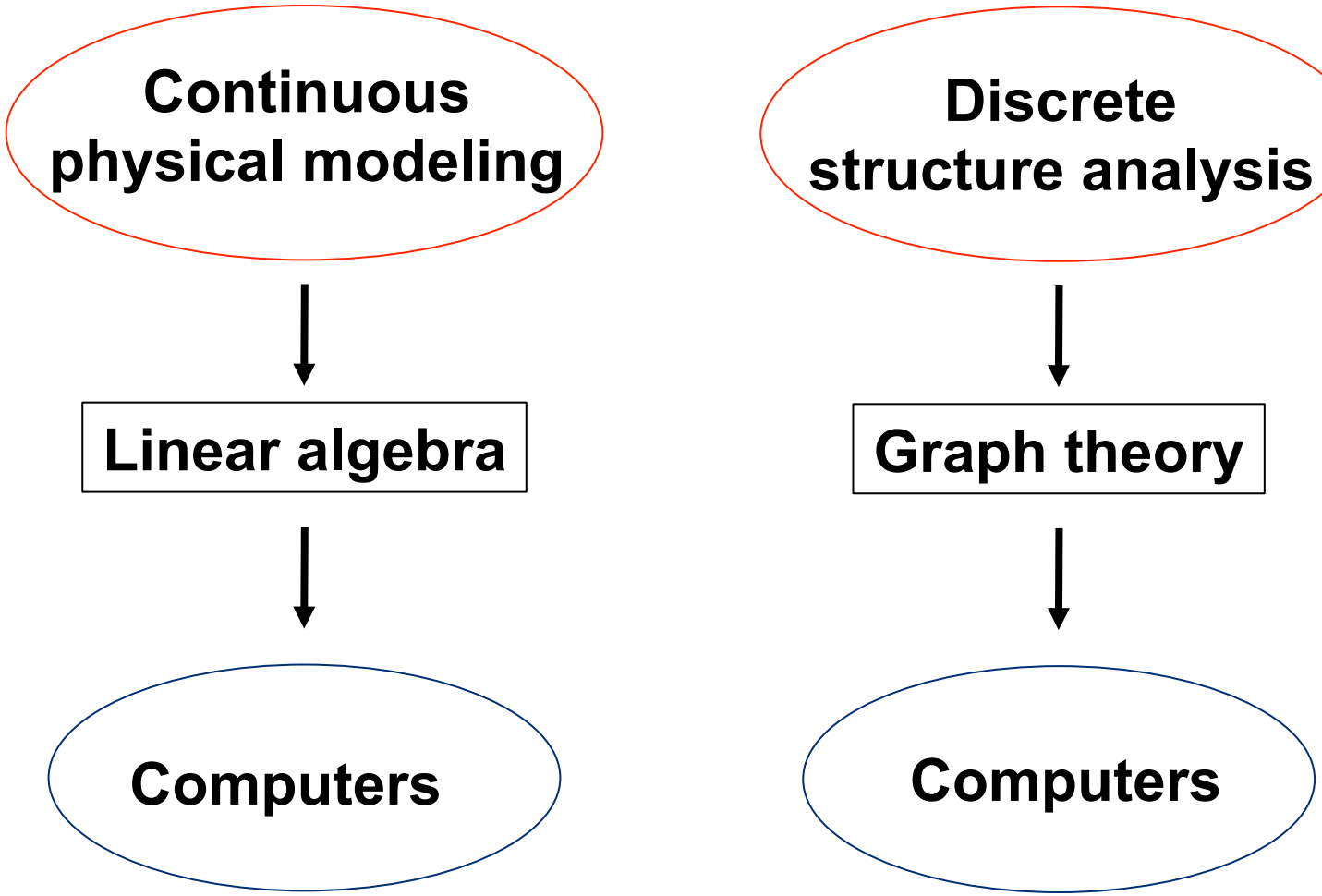
WWW snapshot, courtesy Y. Hyun

- Scientific datasets: biological, chemical, cosmological, ecological, …

Yeast protein interaction network, courtesy H. Jeong

UCSB

An analogy?

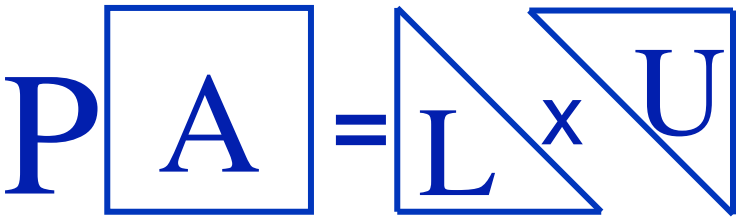


UCSB

Top 500 List (June 2011)

Top500 Benchmark:

Solve a large system of linear equations by Gaussian elimination

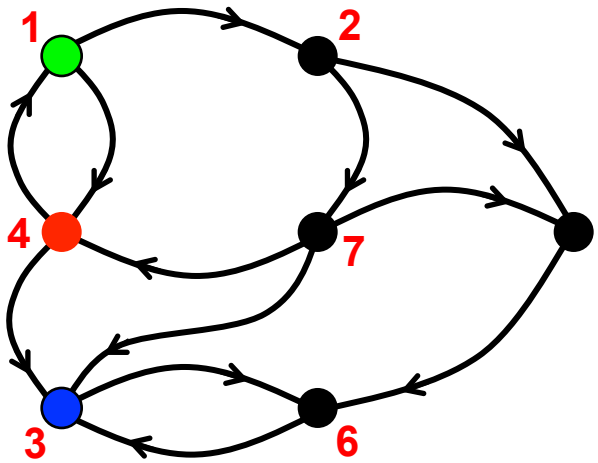


Rank	Site	Computer/Year Vendor	Cores	R_{max}
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	548352	8162.00
2	National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368	2566.00
3	DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00
4	National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640	1271.00
5	GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00
6	DOE/NNSA /LANL/SNL United States	Cielo - Cray XE6 8-core 2.4 GHz / 2011 Cray Inc.	142272	1110.00
7	NASA/Ames Research Center/NAS United States	Pleiades - SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband / 2011 SGI	111104	1088.00
8	DOE/SC /LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408	1054.00

Graph 500 List (June 2011)

Graph500 Benchmark:

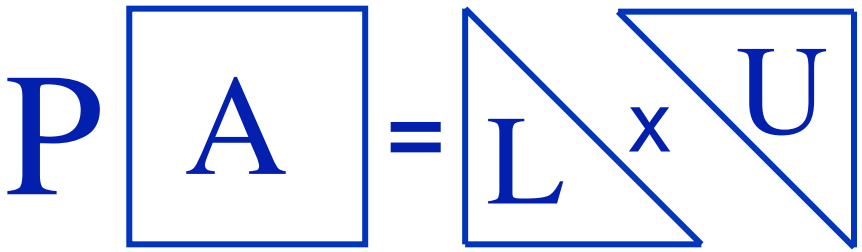
Breadth-first search in a large power-law graph



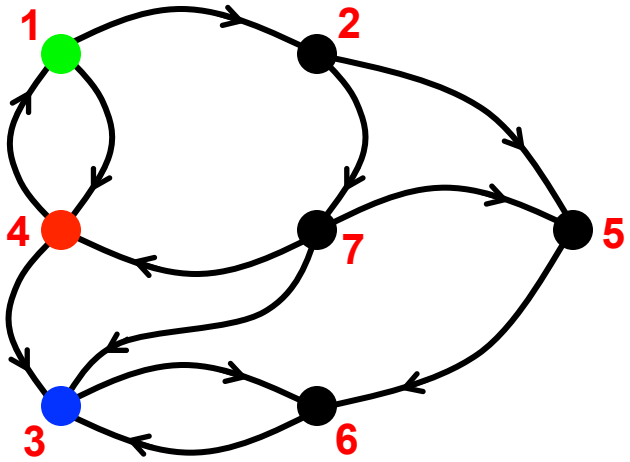
Rank	Machine	Owner	Problem Size	TEPS
1	Intrepid (IBM Blue Gene/P, 32,768 nodes / 131,072 cores)	ANL	38	18,508,000,000
2	Jugene (IBM Blue Gene/P, 32,768 nodes / 131,072 cores)	Forschungszentrum Jülich	38	18,416,700,000
3	Lomonosov (MPP, 4096 nodes / 8192 cores)	Moscow State University	37	43,471,500,000
4	Hopper (Cray XE6, 1800 nodes / 43,200 cores)	LBL	37	25,075,200,000
5	Franklin (Cray XT4, 4000 nodes / 16,000 cores)	LBL	36	19,955,100,000
6	Lonestar (Dell PowerEdge M610, 512 nodes / 6144 cores)	TACC	34	8,080,000,000
7	Kraken (Appro, 1 node / 32 cores / Fusion I/O)	LLNL	34	55,948,453
8	Red Sky (Sun, 512 nodes / 4096 cores)	SNL	33	9,470,000,000
9	Endeavor (Westmere X5670, 256 processors / 3072 cores)	Intel	33 (Toy MPI Simple)	6,860,000,000
10	SGI Altix UV 1000 (2048 cores)	SGI	32	10,161,300,000
11	IBM BlueGene/P, 2048 nodes / 8192 cores	Moscow State University	32	6,930,560,000
12	Blacklight (SGI Altix UV 1000, 512 processors)	PSC	32 (Small)	4,452,270,000

Floating-Point vs. Graphs, June 2011

8.1 Petaflops



43 Gigateps



8.1 Peta / 43 Giga is about 190,000!

UCSB

Outline

- Motivation
- Libraries: CombBLAS and KDT
- Algorithms
- Plans

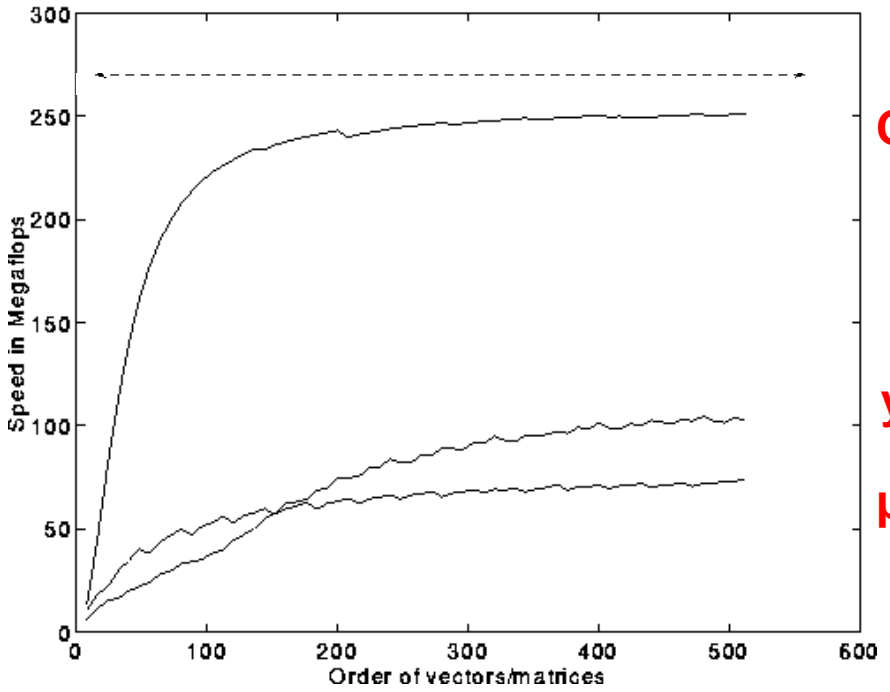
UCSB

The challenge of the software stack

- By analogy to numerical scientific computing. . .

- What should the combinatorial BLAS look like?

Basic Linear Algebra Subroutines (BLAS): Speed (MFlops) vs. Matrix Size (n)



C = A*B

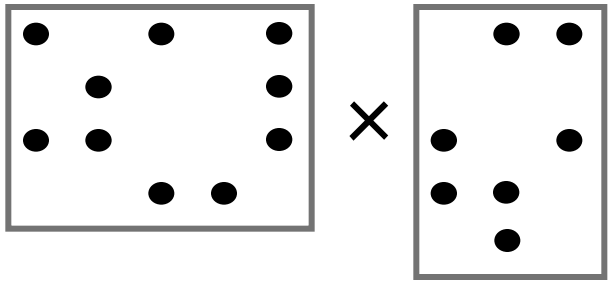
y = A*x

μ = x^T y

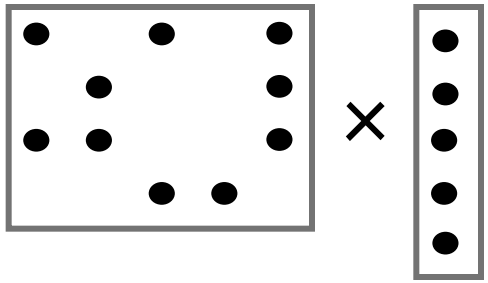
UCSB

Sparse array-based primitives

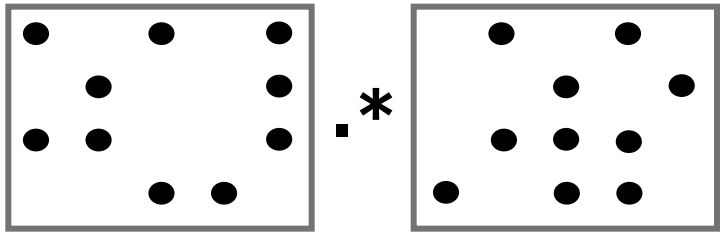
Sparse matrix-matrix multiplication (SpGEMM)



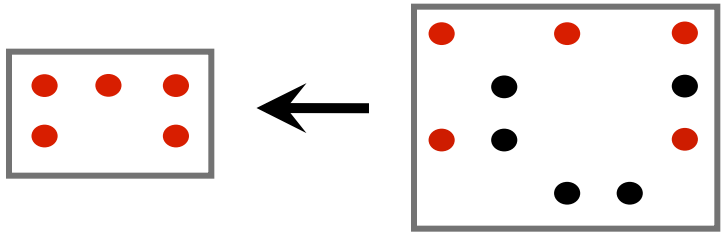
Sparse matrix-dense vector multiplication



Element-wise operations



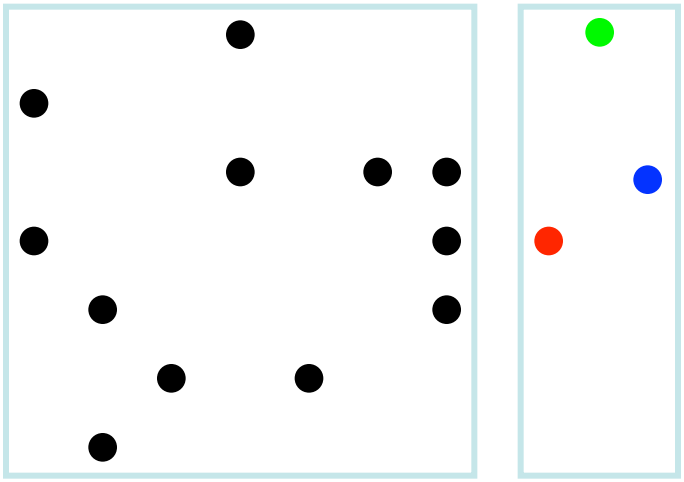
Sparse matrix indexing

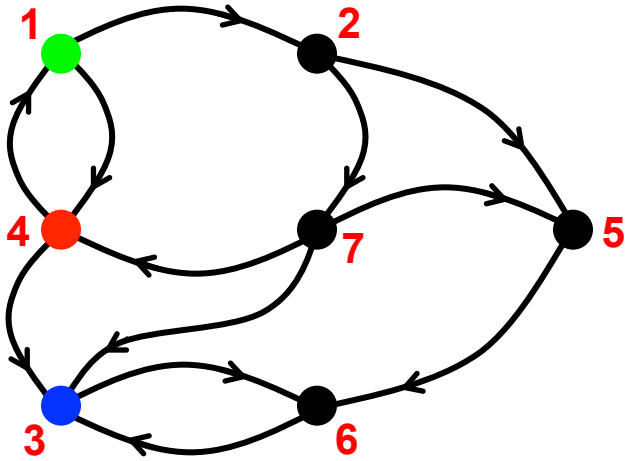


Matrices on various semirings: (x, +) , (and, or) , (+, min) , …

UCSB

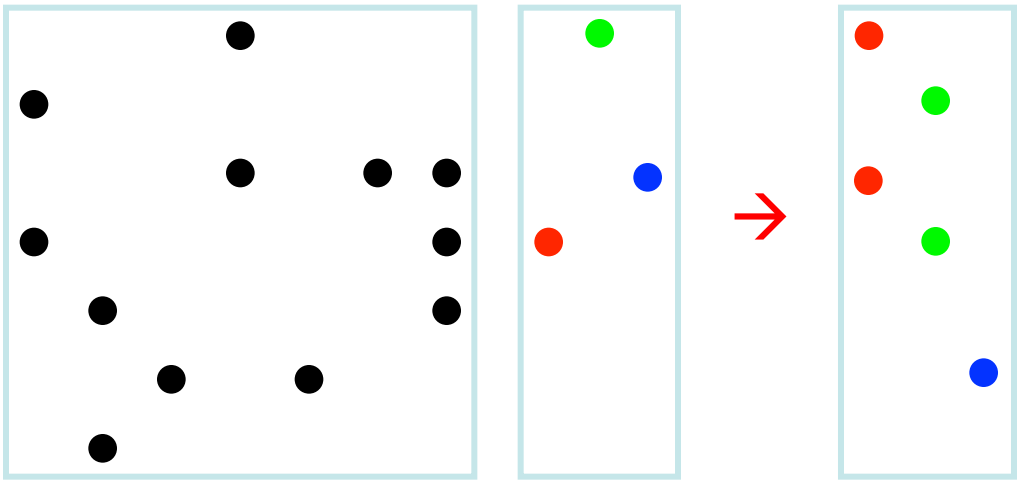
Multiple-source breadth-first search

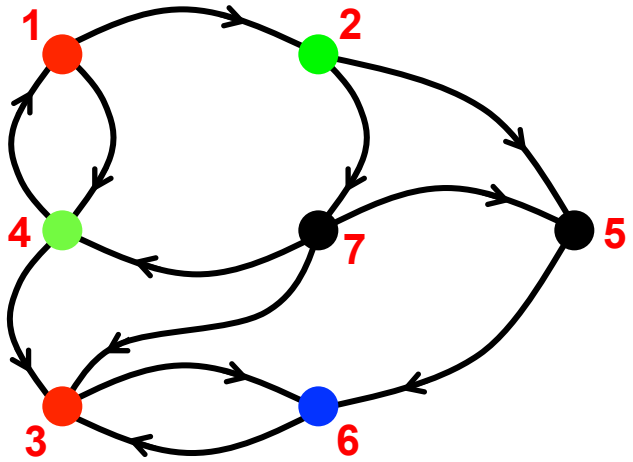




UCSB

Multiple-source breadth-first search





- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

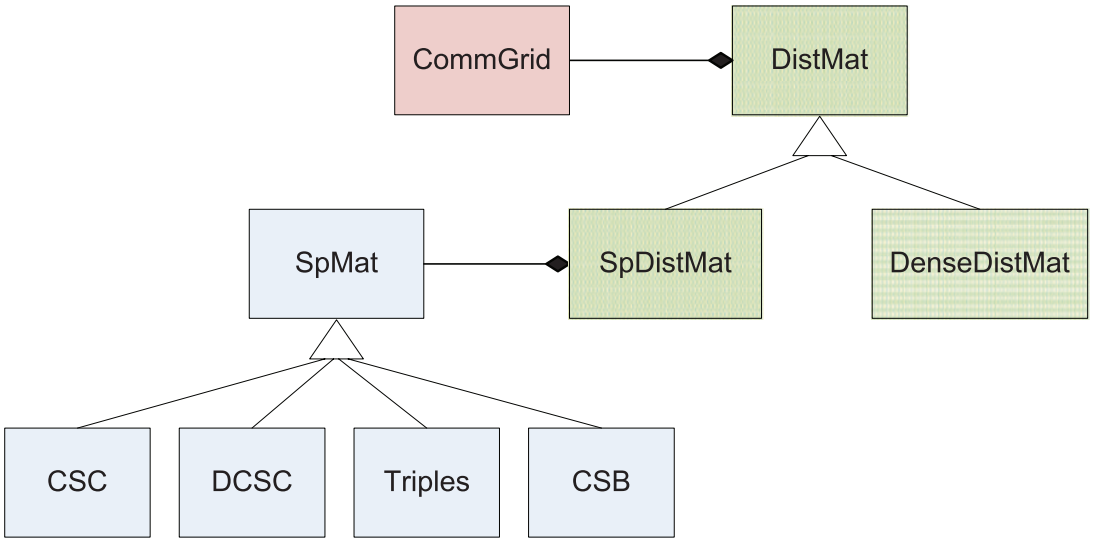
UCSB

The case for sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

Traditional graph computations	Graphs in the language of linear algebra
Data driven, unpredictable communication.	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained parallelism, bandwidth limited

Combinatorial BLAS: A matrix-based graph library



Architecture of matrix classes

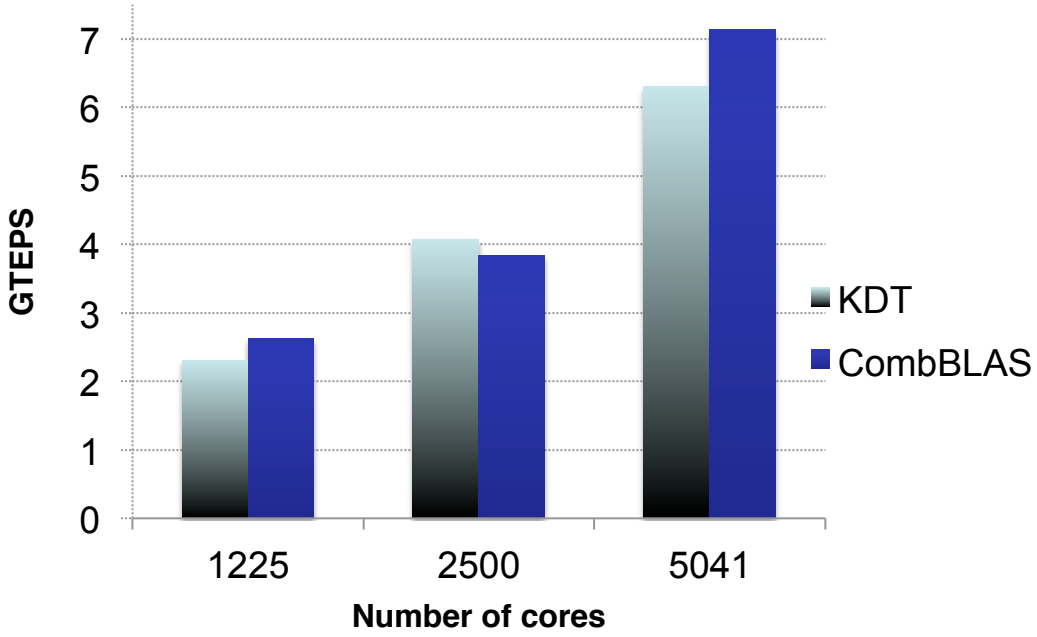
- Also sparse & dense vectors, distributed and local
- Matrix operations over user-defined (and some built-in) semirings
- Highly templated C++
- Reference implementation in MPI

UCSB

Some Combinatorial BLAS functions

Function	Applies to	Parameters		Returns	Matlab Phrasing
SpGEMM	Sparse Matrix (as friend)	**A**, **B**: trA: trB:	sparse matrices transpose **A** if true transpose **B** if true	Sparse Matrix	$\mathbf{C} = \mathbf{A} * \mathbf{B}$
SpMV	Sparse Matrix (as friend)	**A**: **x**: trA:	sparse matrices sparse or dense vector(s) transpose **A** if true	Sparse or Dense Vector(s)	$\mathbf{y} = \mathbf{A} * \mathbf{x}$
SpEWiseX	Sparse Matrices (as friend)	**A**, **B**: notA: notB:	sparse matrices negate **A** if true negate **B** if true	Sparse Matrix	$\mathbf{C} = \mathbf{A} * \mathbf{B}$
Reduce	Any Matrix (as method)	dim: *binop*:	dimension to reduce reduction operator	Dense Vector	sum(**A**)
SpRef	Sparse Matrix (as method)	**p**: **q**:	row indices vector column indices vector	Sparse Matrix	$\mathbf{B} = \mathbf{A}(\mathbf{p}, \mathbf{q})$
SpAsgn	Sparse Matrix (as method)	**p**: **q**: **B**:	row indices vector column indices vector matrix to assign	none	$\mathbf{A}(\mathbf{p}, \mathbf{q}) = \mathbf{B}$
Scale	Any Matrix (as method)	**rhs**:	any object (except a sparse matrix)	none	Check guiding principles 3 and 4
Scale	Any Vector (as method)	**rhs**:	any vector	none	none
Apply	Any Object (as method)	*unop*:	unary operator (applied to non-zeros)	None	none

BFS in “vanilla” MPI Combinatorial BLAS



- Graph500 benchmark at scale 29, C++ (or KDT) calling CombBLAS
- NERSC “Hopper” machine (Cray XE6)
- [Buluç & Madduri]: New hybrid CombBLAS MPI + OpenMP gets 17.8 GTEPS at scale 32 on 40,000 cores of Hopper

UCSB

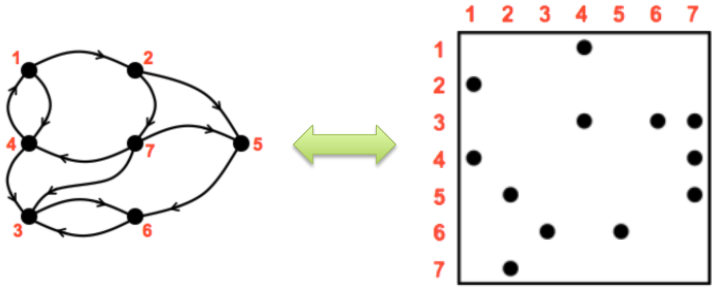
Outline

- Motivation
- Libraries: CombBLAS and KDT
- Algorithms
- Plans

UCSB

Knowledge Discovery Toolbox

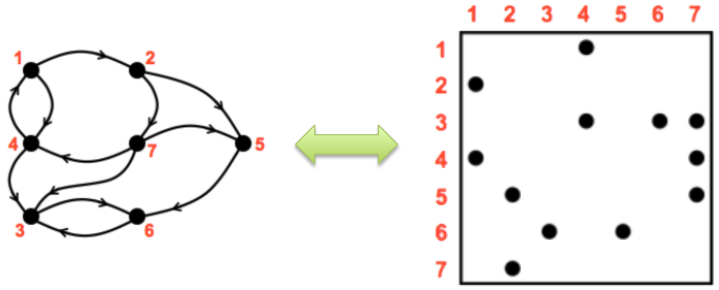
http://kdt.sourceforge.net/



A general graph library with operations based on linear algebraic primitives

Knowledge Discovery Toolbox

http://kdt.sourceforge.net/

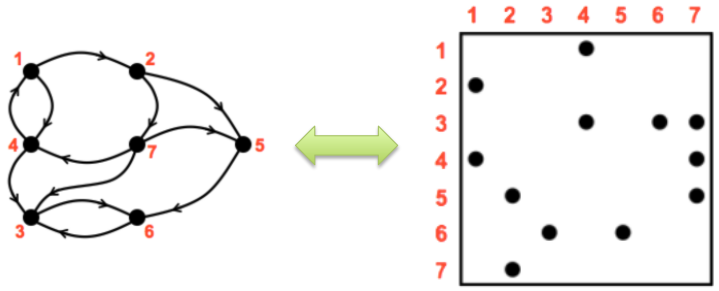


A general graph library with operations based on linear algebraic primitives

- Aimed at domain experts who know their problem well but don't know how to program a supercomputer
- Easy-to-use Python interface
- Runs on a laptop as well as a cluster with 10,000 processors

Knowledge Discovery Toolbox

http://kdt.sourceforge.net/



A general graph library with operations based on linear algebraic primitives

- Aimed at domain experts who know their problem well but don't know how to program a supercomputer
- Easy-to-use Python interface
- Runs on a laptop as well as a cluster with 10,000 processors

- A collaboration among UCSB, Lawrence Berkeley National Lab, and Microsoft Technical Computing
- Open source software, released under New BSD license
- v0.1 released March 2011; v0.2 expected October 2011

Domain Expert vs. Graph Expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank

- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

Domain Expert vs. Graph Expert

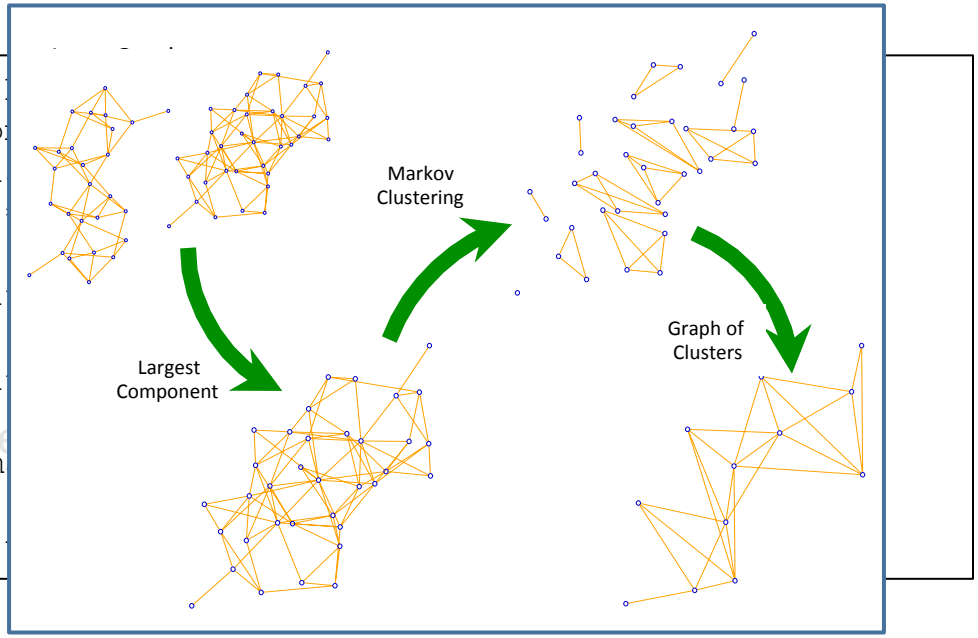
- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank

```
# bigG contains the input graph
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)

clus = G.cluster('Markov')

clusNedge = G.nedge(clus)

smallG = G.contract(clus)

# visualize
```

- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

Domain Expert vs. Graph Expert

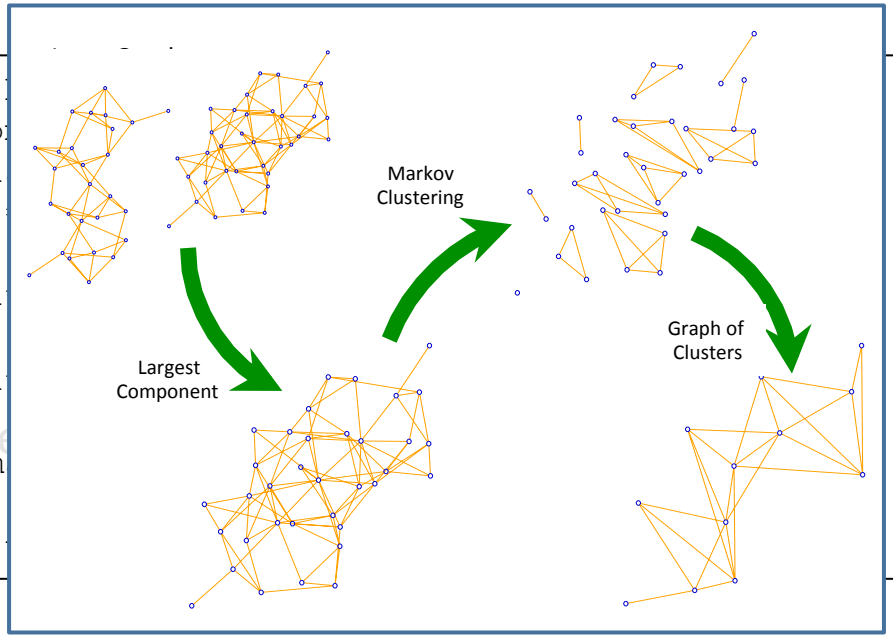
- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank



- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

Domain Expert vs. Graph Expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority: betwee
  centrality, PageRank

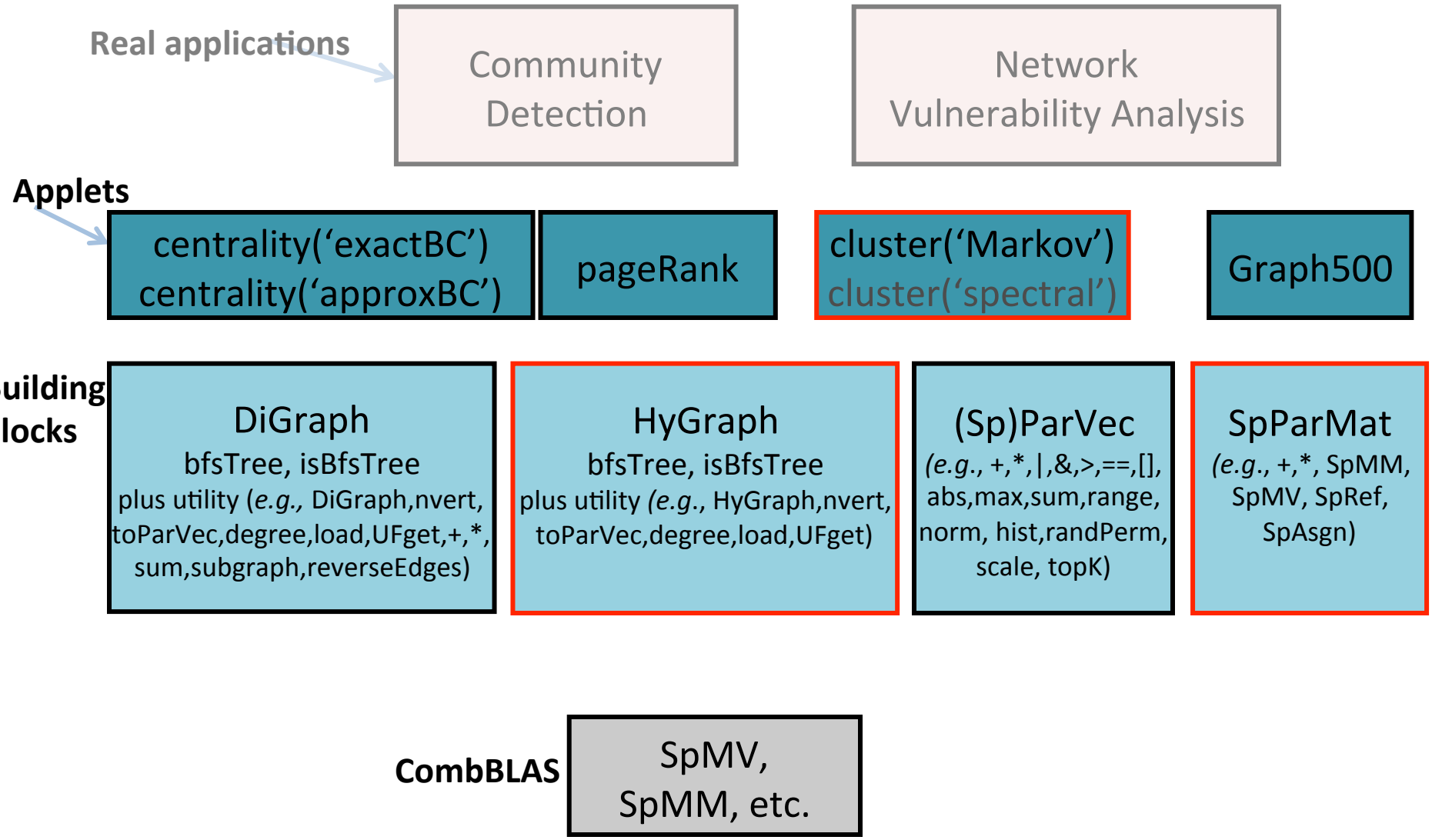


- Hypergraphs and sparse matric
- Graph primitives (*e.g.*, `bfsTree`)
- SpMV / SpGEMM on semirings

```
[…]
L = G.toSpParMat()
d = L.sum(kdt.SpParMat.Column)
L = -L
L.setDiag(d)
M = kdt.SpParMat.eye(G.nvert()) - mu*L
pos = kdt.ParVec.rand(G.nvert())
for i in range(nsteps):
    pos = M.SpMV(pos)
```

Graph API (v0.2)

New for v0.2



A few KDT applications

Markov Clustering

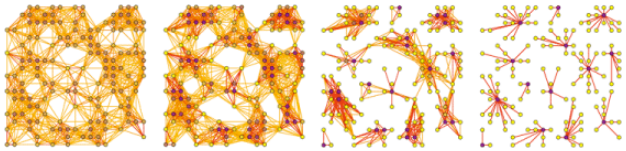
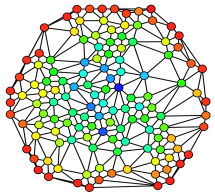


image courtesy Stijn van Dongen

Markov Clustering (MCL) finds clusters by postulating that a random walk that visits a dense cluster will probably visit many of its vertices before leaving.

We use a Markov chain for the random walk. This process is reinforced by adding an inflation step that uses the Hadamard product and rescaling.

Betweenness Centrality



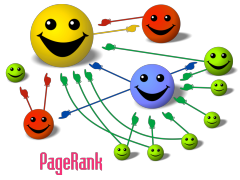
$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

image courtesy Claudio Rocchini

Betweenness Centrality says that a vertex is important if it appears on many shortest paths between other vertices. An exact computation requires a BFS for every vertex. A good approximation can be achieved by sampling starting vertices.

PageRank

PageRank says a vertex is important if other important vertices link to it.

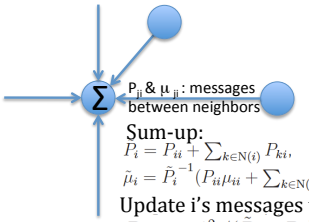


courtesy Felipe Micaroni Lalli

Each vertex (webpage) votes by splitting its PageRank score evenly among its out edges (links). This broadcast (an SpMV) is followed by a normalization step (ColWise). Repeat until convergence.

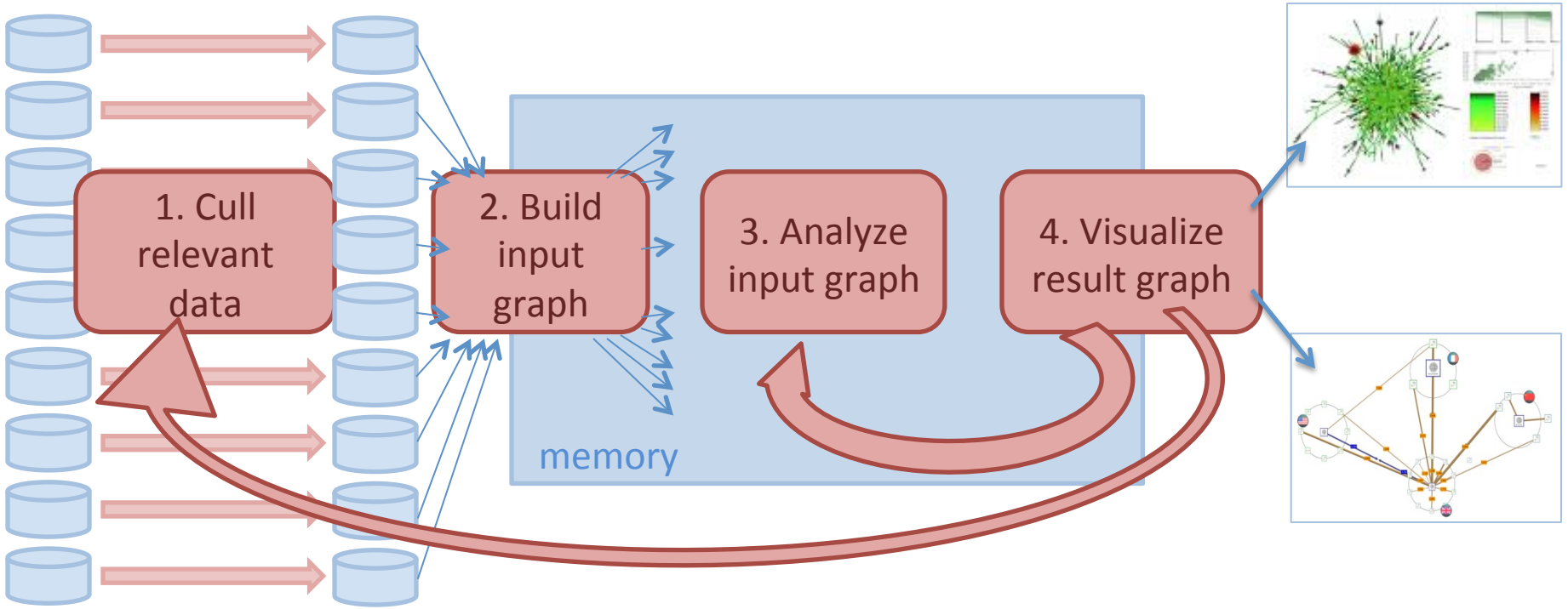
PageRank is the stationary distribution of a Markov Chain that simulates a "random surfer".

Belief Propagation



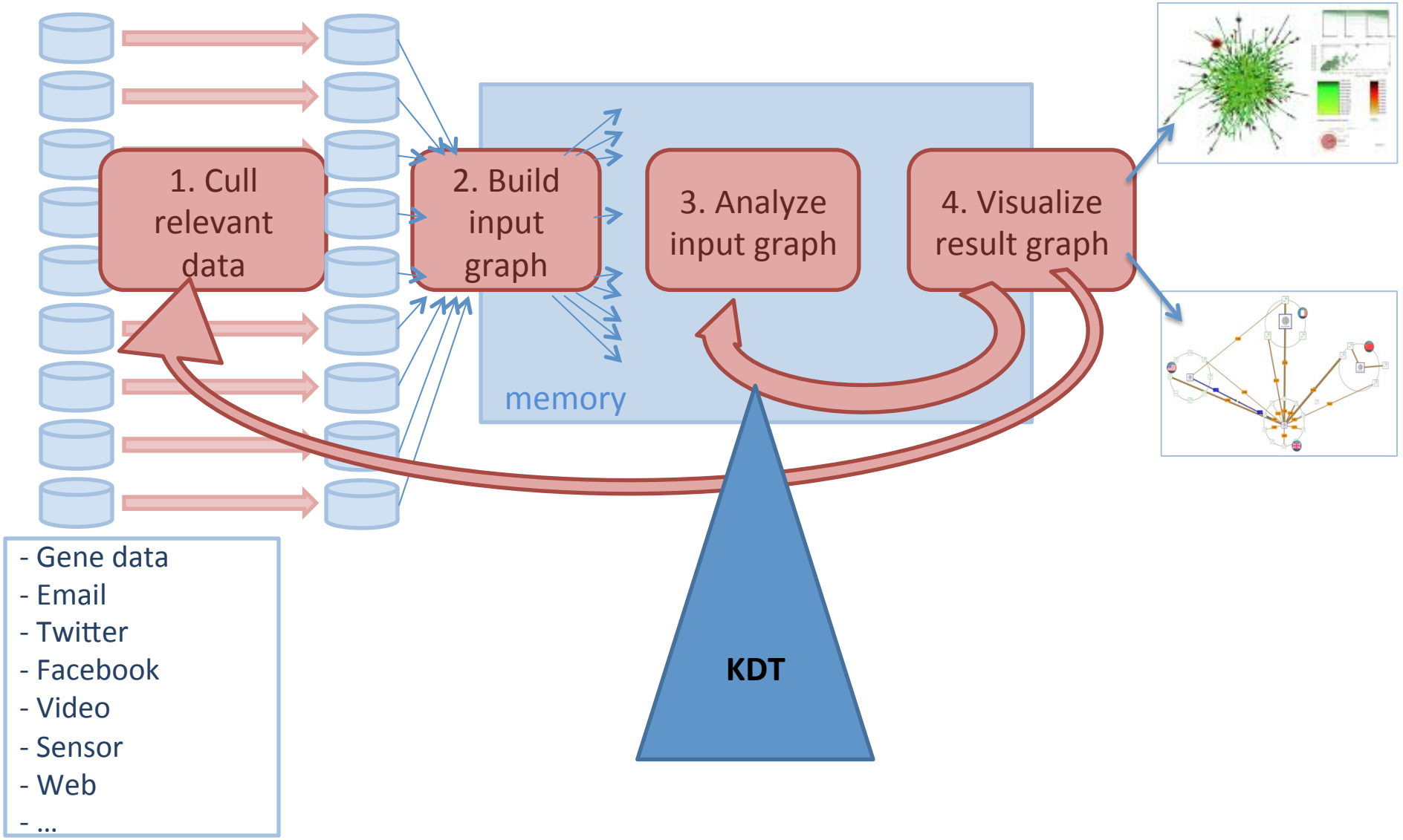
Gaussian belief propagation (GaBP) is an iterative algorithm for solving the linear system of equations Ax = b, where A is symmetric positive definite.
GaBP assumes each variable follows a normal distribution. It iteratively calculates the precision P and mean value μ of each variable; the converged mean-value vector approximates the actual solution.

KNOWLEDGE DISCOVERY WORKFLOW

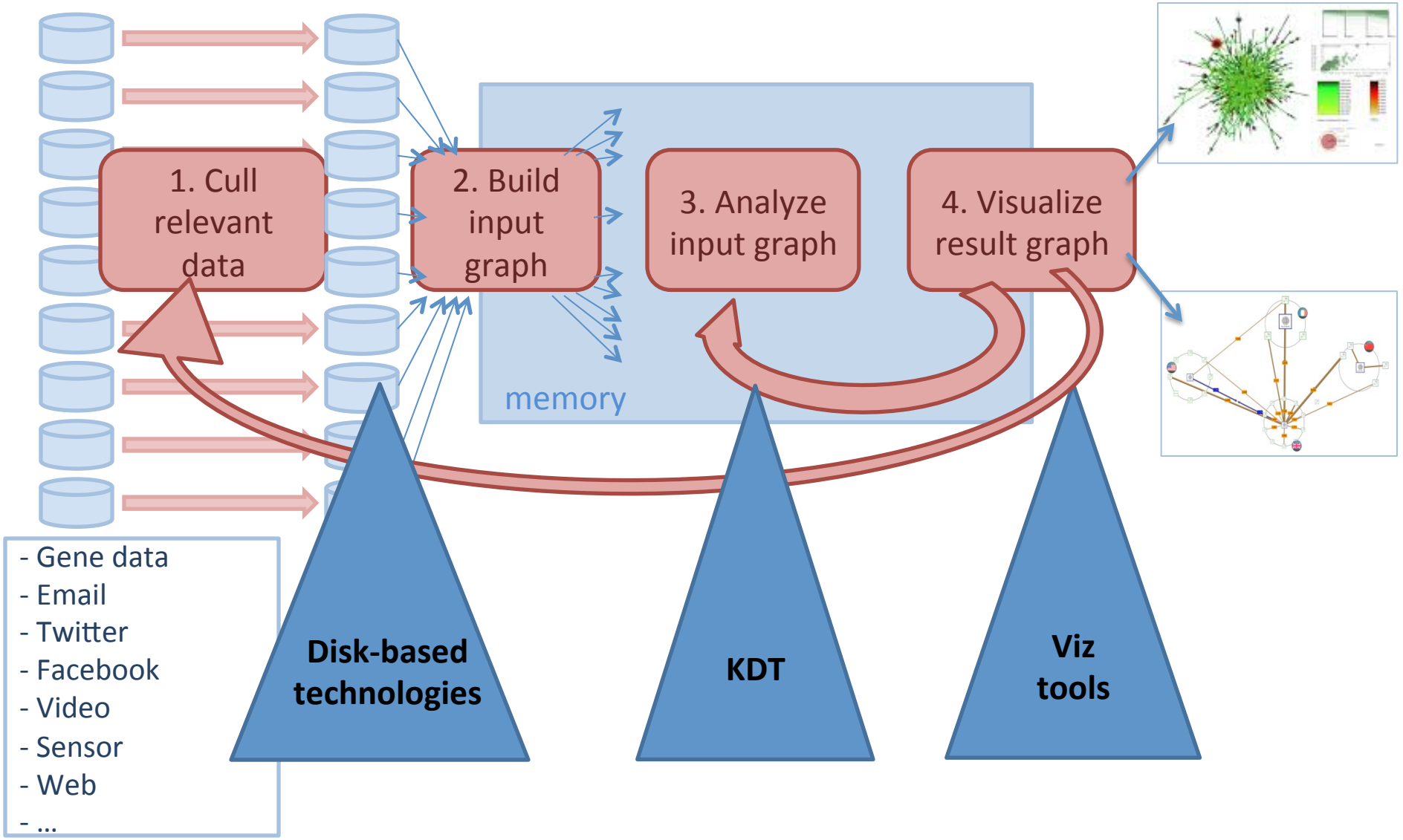


- Gene data
- Email
- Twitter
- Facebook
- Video
- Sensor
- Web
- …

KNOWLEDGE DISCOVERY WORKFLOW



KNOWLEDGE DISCOVERY WORKFLOW

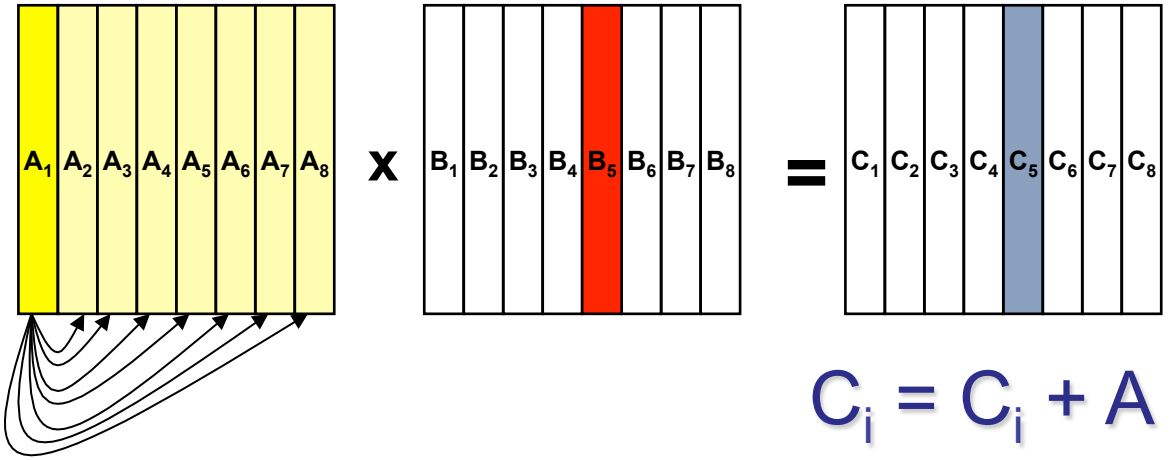


Outline

- Motivation
- Libraries: CombBLAS and KDT
- Algorithms
- Plans

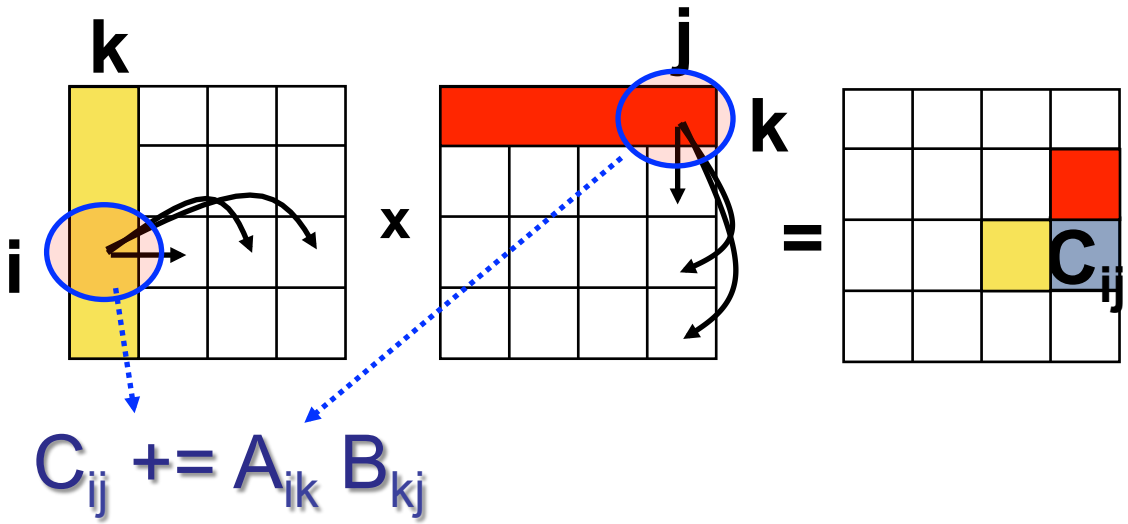
UCSB

Two versions of sparse GEMM



$C_i = C_i + A\,B_i$

1D block-column distribution

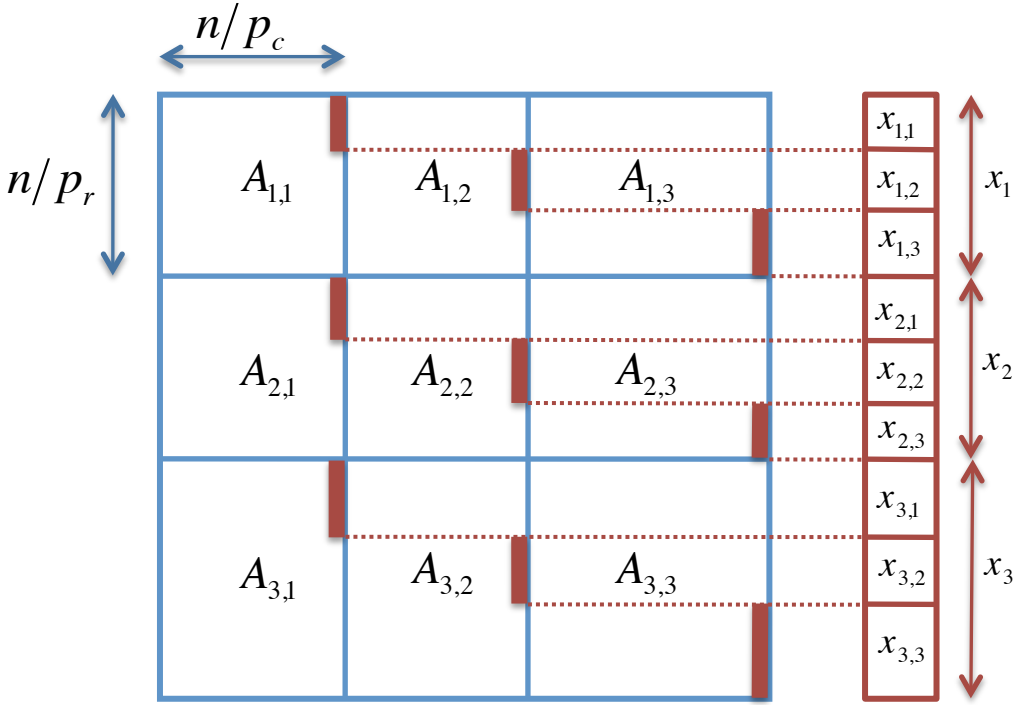


$C_{ij} \mathrel{+}= A_{ik}\,B_{kj}$

2D block checkerboard distribution

UCSB

2D layout for sparse matrices & vectors



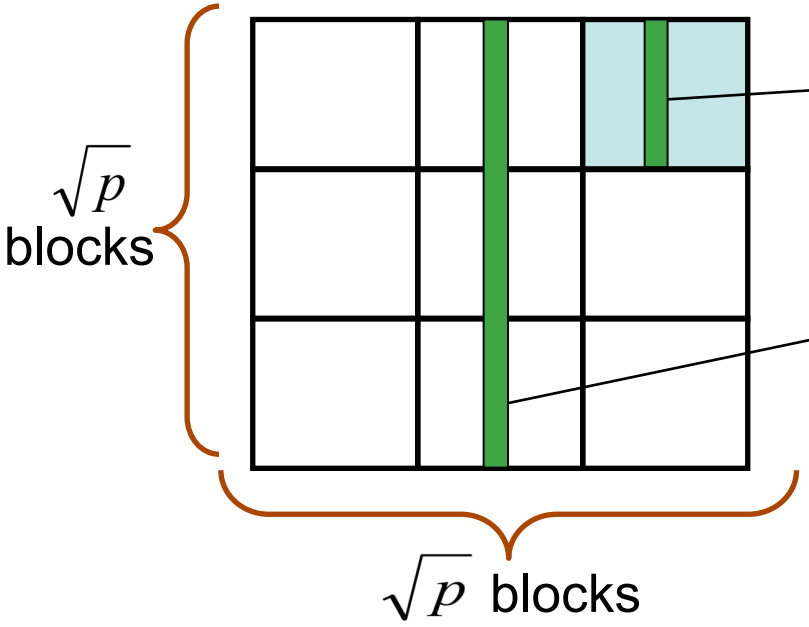
Matrix/vector distributions, interleaved on each other.

Default distribution in **Combinatorial BLAS**.

- 2D matrix layout wins over 1D with large core counts and with limited bandwidth/compute
- 2D vector layout sometimes important for load balance
- Scalable with increasing number of processes

Node-level considerations

Submatrices are "*hypersparse*" *(i.e. nnz << n)*



$$nnz' = \frac{c}{\sqrt{p}} \rightarrow 0$$

Average of c nonzeros per column

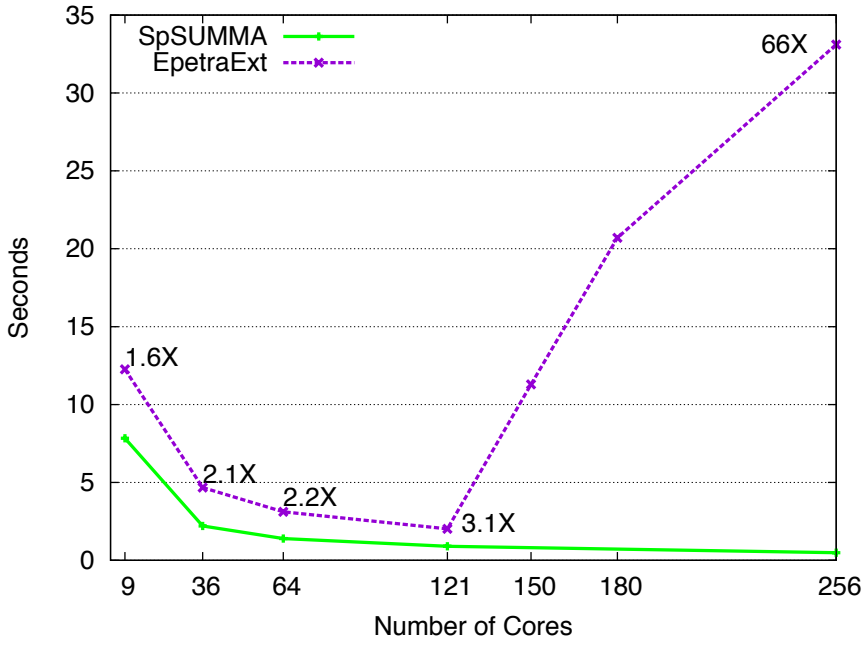
Total Storage:

$$O(n + nnz) \Rightarrow O(n\sqrt{p} + nnz)$$

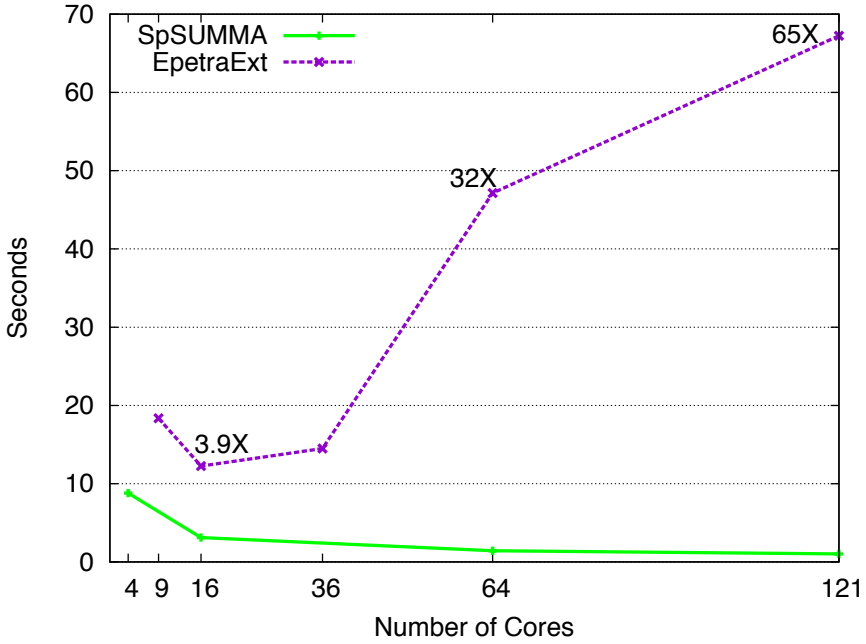
- A data structure or algorithm that depends on matrix dimension n (e.g. CSR or CSC) is asymptotically too wasteful for submatrices
- Use doubly-compressed (DCSC) or compressed sparse block (CSB) data structures instead.

UCSB

Comparison of SpGEMM implementations



(a) R-MAT × R-MAT product (scale 21).



(b) Multiplication of an R-MAT matrix of scale 23 with the restriction operator of order 8.

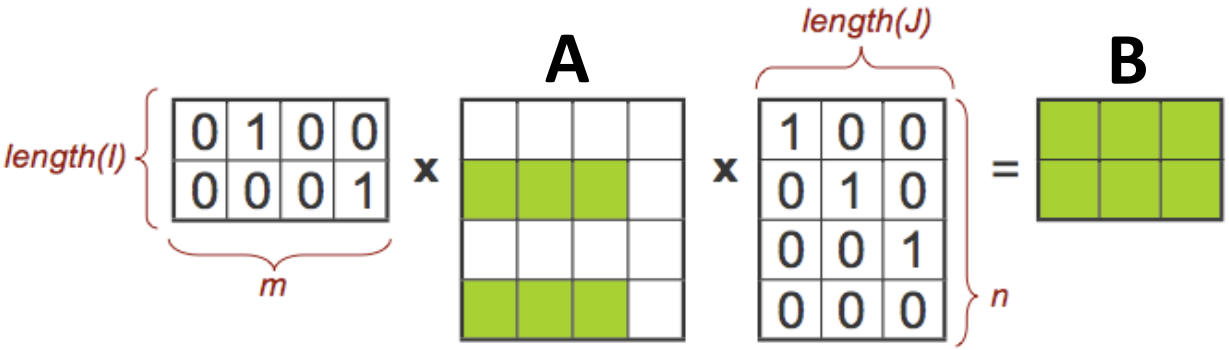
- SpSUMMA = 2-D data layout (Combinatorial BLAS)
- EpetraExt = 1-D data layout (Trilinos)

UCSB

Indexing sparse arrays in parallel
(extract subgraphs, coarsen grids, etc.)

SpRef:	`B = A(I,J)`
SpAsgn:	`B(I,J) = A`
SpExpAdd:	`B(I,J) += A`

`A,B:` sparse matrices
`I,J:` vectors of indices



SpRef using mixed-mode sparse matrix-matrix multiplication (**SpGEMM**). Ex: B = A([2,4], [1,2,3])

UCSB

Sequential SpRef and SpAsgn

```
function B = spref(A,I,J)
  R = sparse(1:length(I),I,1,length(I),size(A,1));
  Q = sparse(J,1:length(J),1,size(A,2),length(J));
  B = R*A*Q;
```

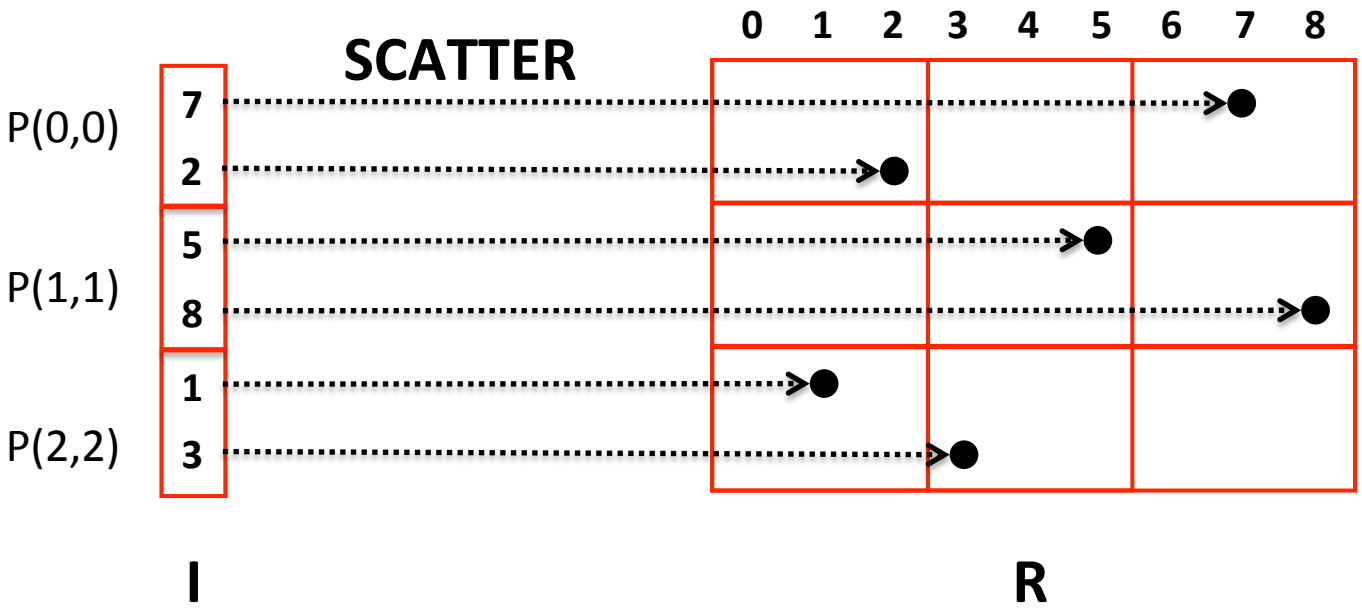
```
function C = spasgn(A,I,J,B)
  [ma,na] = size(A);
  [mb,nb] = size(B);
  R = sparse(I,1:mb,1,ma,mb);
  Q = sparse(1:nb,J,1,nb,na);
  S = sparse(I,I,1,ma,ma);
  T = sparse(J,J,1,na,na);
  C = A + R*B*Q - S*A*T;
```

$$A + \begin{pmatrix} 0 & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 \\ 0 & A(I,J) & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

UCSB

Parallel algorithm for SpRef

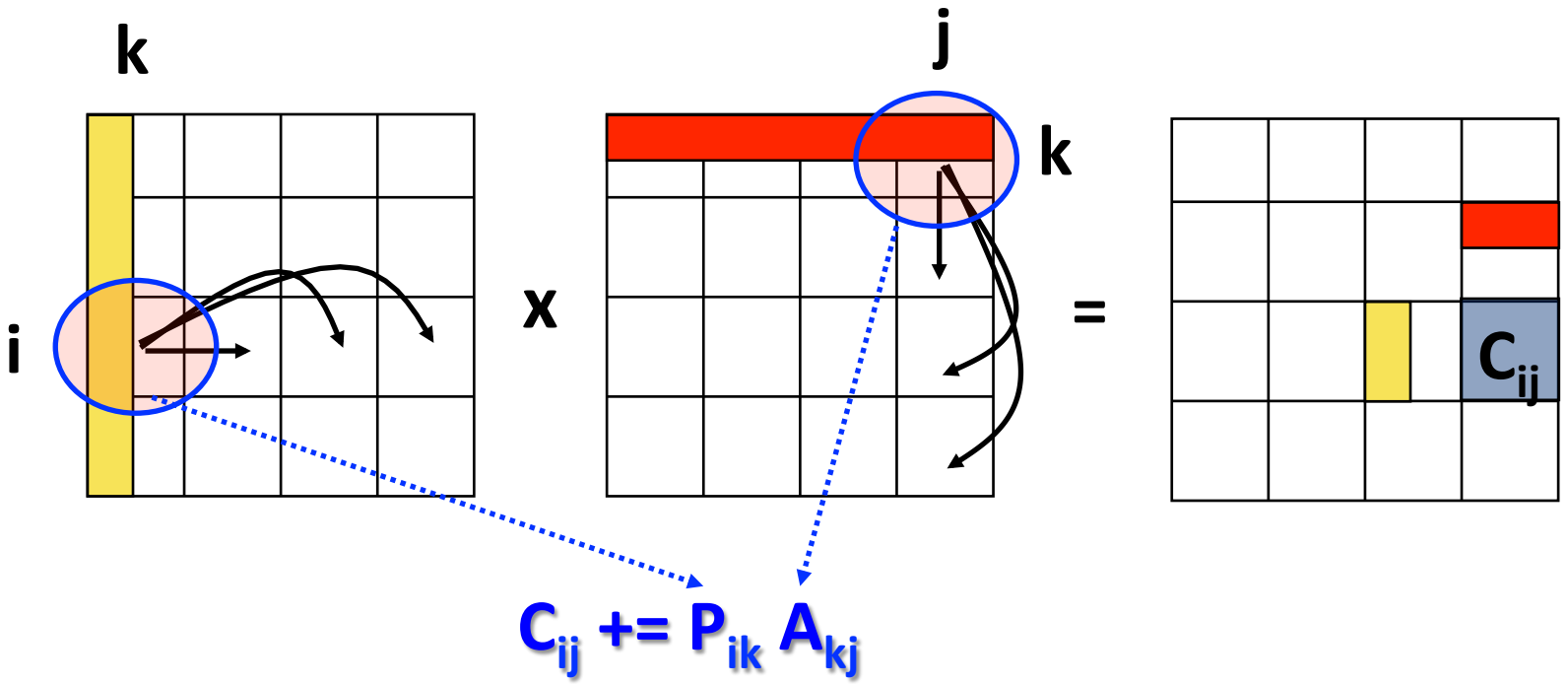
1. Form R from I in parallel, on a 3x3 processor grid



UCSB

Parallel algorithm for SpRef

2. **SpGEMM** using memory-efficient Sparse SUMMA.

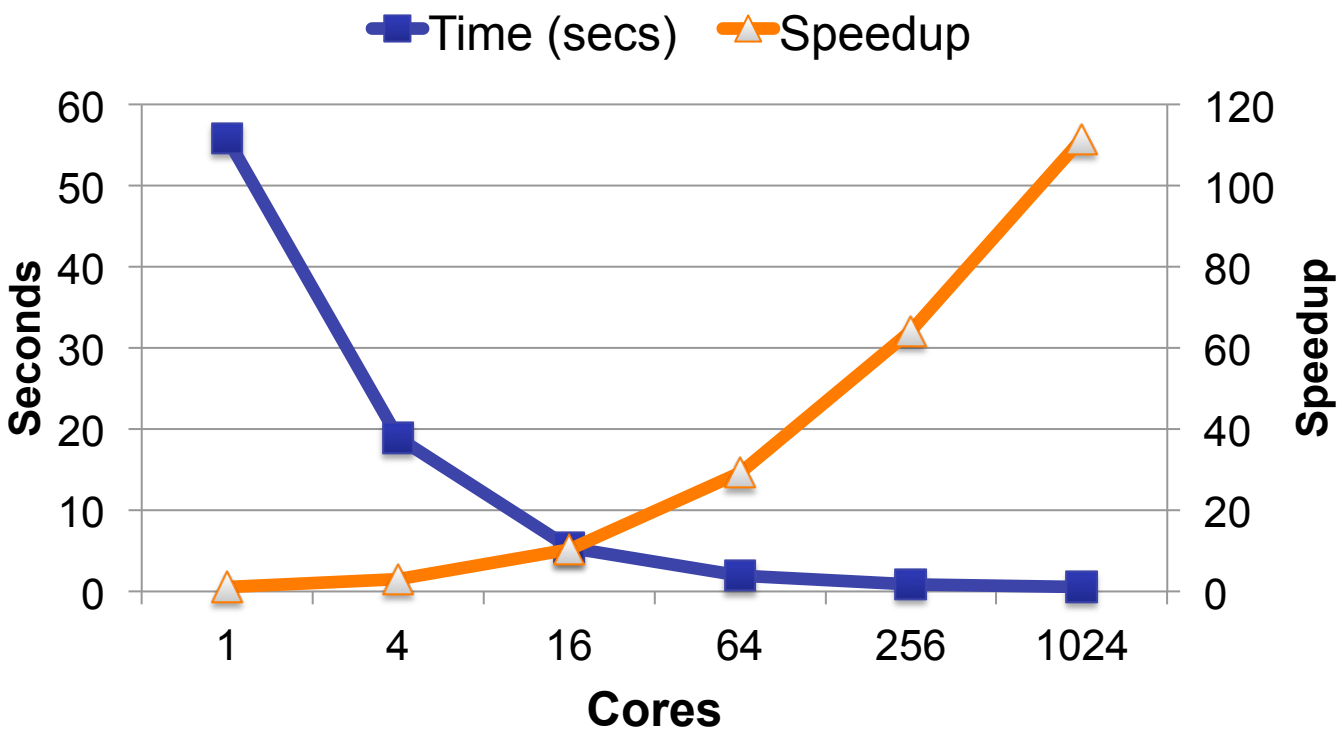


Minimize temporaries by:

- Splitting local matrix, and broadcasting multiple times
- Deleting P (and A if in-place) after forming C=P*A

UCSB

Strong scaling of SpRef

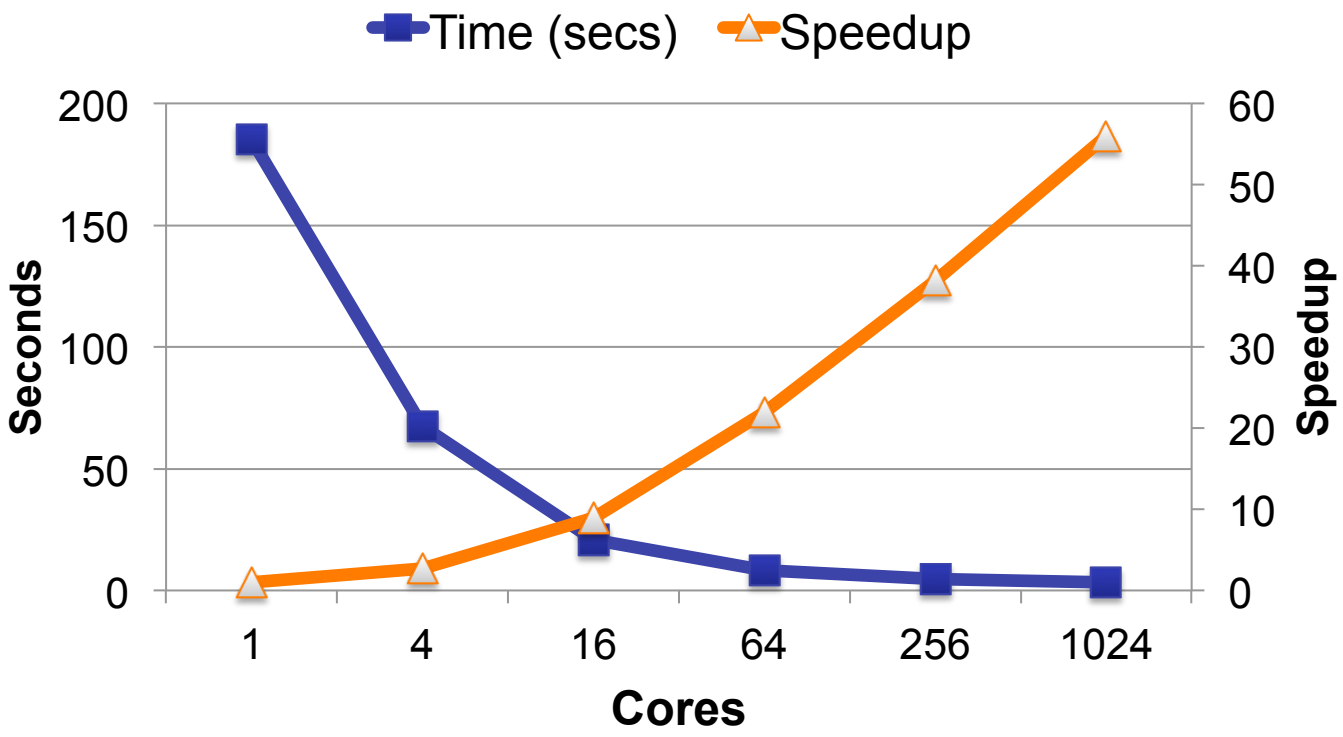


random symmetric permutation ⇔ relabeling graph vertices

- RMAT Scale 22; edge factor=8; a=.6, b=c=d=.4/3
- Franklin/NERSC, each node is a quad-core AMD Budapest

UCSB

Strong scaling of SpRef



Extracts 10 random (induced) subgraphs, each with |V|/10 vert.
Higher span → Decreased parallelism → Lower speedup

UCSB

Outline

- Motivation
- Libraries: CombBLAS and KDT
- Algorithms
- Plans

UCSB

Coming in v0.2: Attributed Semantic Graphs and Filters

Example:

- Vertex types: Person, Phone, Camera
- Edge types: PhoneCall, TextMessage, CoLocation
- Edge attributes: StartTime, EndTime

- Calculate centrality just for PhoneCalls and TextMessages between times sTime and eTime

```
def vfilter(self, vTypes):
    return self.type in vTypes

def efilter(self, eTypes, sTime, eTime):
    return ((self.type in eTypes) and
            (self.sTime > sTime) and
            (self.eTime < eTime))

wantedVTypes = (People)
wantedETypes = (PhoneCall, TextMessage)
start = dt.now() - dt.timedelta(hours=1)
end = dt.now()
bc = G.centrality('approxBC',filter=
        ((vfilter, wantedVTypes),
         (efilter, wantedETypes,
          start, end)))
```

Options and issues in implementing filters

- Prefilter to extract the relevant subgraph
  - Simplest solution
  - Too much memory or time for some applications

Options and issues in implementing filters

- Prefilter to extract the relevant subgraph
  - Simplest solution
  - Too much memory or time for some applications
- Write filters as semiring ops in C++, wrap in Python
  - Can get good performance at CombBLAS level
  - Inflexible, hard to write new filters

Options and issues in implementing filters

- Prefilter to extract the relevant subgraph
  - Simplest solution
  - Too much memory or time for some applications
- Write filters as semiring ops in C++, wrap in Python
  - Can get good performance at CombBLAS level
  - Inflexible, hard to write new filters
- Write filters in Python, call back from CombBLAS
  - Very flexible
  - But slow

Options and issues in implementing filters

- Prefilter to extract the relevant subgraph
  - Simplest solution
  - Too much memory or time for some applications
- Write filters as semiring ops in C++, wrap in Python
  - Can get good performance at CombBLAS level
  - Inflexible, hard to write new filters
- Write filters in Python, call back from CombBLAS
  - Very flexible
  - But slow
- Need a better way! SEJITS?

Plans

- KDT:
  - Release V0.2 soon (semantic graphs & attribute filters)
  - Evolve front end to include other parallel graph libraries
  - Selective, embedded JIT specialization to accelerate KDT/CombBLAS: Fox, Kamil et al.
  - Collectives and autotuning for discrete primitives: Williams, Oliker et al.
- More algorithms work (multicore, hybrid)
- More applications (time-dependent path planning, ….)