Grafiki: Trilinos-based Software for High-Performance Distributed Graph-based Algorithms

Nathan Ellingwood

Thanks to Jon Berry, Danny Dunlavy, Rich Lehoucq, Mike Eydenberg, Alex Foss, Renee Gooding, Carolyn Mayer, Derek Tucker, Michael Weylandt

Nov. 1, 2023

SAND2023-11247C

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Grafiki Motivation

Questa needs to solve linear systems on HPC systems

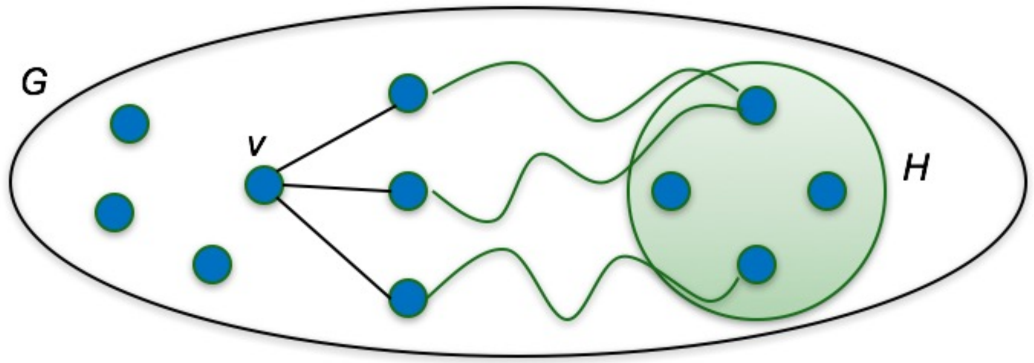
- **Current focus**: Scalable computation of hitting time moments for large scale data science problems
- Leverage DOE/Sandia experience with numerical linear algebra and performance portability

Grafiki: Software for high-performance graph-based algorithms

- Built on **Trilinos**, Sandia/DOE software with distributed linear algebra-based solvers designed for multi-physics engineering/science problems
- Current computational capabilities include:
  - Graph hitting times (today's focus)
  - Spectral clustering (hypergraph support)
  - Eigenvector centrality (hypergraph support)
- Hitting time capabilities discussed today based on methods developed by Questa team **Rich Lehoucq, Jon Berry, Danny Dunlavy**

Brief Background

Hitting time: *A* random variable describing the number of steps for a random walk from vertex *v* to a vertex in subset *H* of graph *G*



- **Use case - seed set expansion:** given a set of "seed" vertices, find others related or grouped with them
- **Applications**: Information Retrieval, Social Network Analysis, Neuroscience…

Brief Background - Linear System Setup

Graph mean hitting times can be computed by linear system

- Derived from Markov chains applied to graph analysis
- Initial assumption: unweighted, symmetric graph – traversal to any neighbor equally likely

General relationship

$$\boldsymbol{x_i = ones + \sum_{i \neq j} p_{ij}\, x_j}$$

- $\boldsymbol{x_i}$: hitting time starting from node *i*
- $\boldsymbol{p_{ij}}$: transition probability from *j* to *i*
- $\boldsymbol{p_{ij} \leftarrow A_{ij}/d_i}$: valid based on symmetric, unweighted graph assumption

Linear system in matrix form:

$$\boldsymbol{(I - D^{-1}A) * x = ones}$$

- Non-seed vertices included
- $\boldsymbol{A}$: adjacency matrix representing graph
- $\boldsymbol{D}$: diagonal matrix of vertex degrees
- $\boldsymbol{ones}$: vector of 1's

Challenges

Achieve performance-portability for distributed computing and GPU by addressing challenges from data science such as:

- **Laplacian (implicit) operators**
  - Custom operators to avoid data duplication
- **Multiple components**
  - “Discounting” avoids bookkeeping, data duplication
- **Directed, non-symmetric graphs**
  - Extend data manipulation via composite operators
- **Verification and debugging by small-example**
- **Different sparsity patterns**
  - Heuristics may require tuning; 2D partitioning

Laplacians and Iterative Solvers: CG ↔ PCG

Initial assumptions:

- *A* is an adjacency matrix for an undirected, symmetric, connected graph
- *D* is a diagonal matrix of vertex degrees, invertible

Laplacian used for linear system:

The normalized Laplacian:

$$\begin{aligned} \boldsymbol{L_{norm}} &= \boldsymbol{I} - \boldsymbol{D^{-1/2}AD^{-1/2}} \\ &= \boldsymbol{D^{-1/2}(D-A)D^{-1/2}} \end{aligned}$$

The combinatorial Laplacian:

$$\boldsymbol{L = D - A}$$

These methods yield equivalent results for x^*

Preconditioned Conjugate Gradient

- using combinatorial Laplacian $\boldsymbol{L}$, with
  - $Preconditioner$: $\boldsymbol{D}$

$$\boldsymbol{(D-A) * x = b}$$

Conjugate Gradient (CG)

- using $\boldsymbol{L_{norm}}$, no preconditioning

$$\boldsymbol{D^{-1/2}(D-A)D^{-1/2} * y = D^{-1/2} * b}$$
$$\boldsymbol{y = D^{1/2} * x}$$

* *"Golub, Gene H.; Van Loan, Charles F. (2013). Matrix Computations (4th ed.). Johns Hopkins University Press. sec. 11.5.2. ISBN 978-1-4214-0794-4."*

Hitting time linear system

Given the hitting set *H* of vertices, we seek to solve the linear system below for mean hitting times:

$$\boldsymbol{\Pi}^{\mathbf{T}}(\boldsymbol{D}-\boldsymbol{A})\boldsymbol{\Pi} * \widehat{\boldsymbol{x}} = \widehat{\boldsymbol{b}}$$

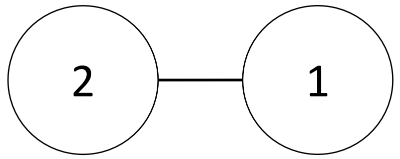
- $\boldsymbol{\Pi}^{\mathbf{T}}$: as left − multiply operator, removes rows corr. to H indices
- $\boldsymbol{\Pi}$: as right − multiply operator removes cols corr. to H indices
- $\boldsymbol{D} = \boldsymbol{diag}(\boldsymbol{A} * \boldsymbol{ones})$; $\boldsymbol{rhs}$: $\widehat{\boldsymbol{b}} = \boldsymbol{\Pi}^{\mathbf{T}}(\boldsymbol{D} * \boldsymbol{ones})$
- $\widehat{\boldsymbol{x}}$: mean hitting time moments from vertices of $G\backslash H$ to H

Solving the formulation above requires either

- Explicit creation of the “reshaped” Laplacian, rhs and lhs for each set *H*, or
- Repeat application of Π^{T} and Π (recreate for each *H*) during PCG, or
- Disruptive logic in our mat-mat, mat-vec mult. operations (e.g. added comm.)

Tiny example

Example: Consider a simple two node undirected graph with corresponding matrices:



$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},\ D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ \mathrm{L} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

If we select node “1” as our hitting set *H*, this gives

$$\Pi = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\ \Pi^T = \begin{bmatrix} 0 & 1 \end{bmatrix},\ \Pi^T \mathrm{L} \Pi = [1]$$ (row 1 and col 1 removed)

And the resulting linear system $\boldsymbol{\Pi}^{\mathbf{T}} \boldsymbol{L} \boldsymbol{\Pi} * \widehat{\boldsymbol{x}} = \widehat{\boldsymbol{b}}$ is

$$[1] * \widehat{\boldsymbol{x}} = [1]$$

Hitting time linear system - reformulated

An alternative formulation to the hitting time linear system for more performant computation:

$$\boldsymbol{\Pi\Pi^{\mathrm{T}}(D - A)\Pi\Pi^{\mathrm{T}} * \Pi\hat{x} = \Pi\Pi^{\mathrm{T}}b}$$

- $\boldsymbol{\Pi\Pi^{\mathrm{T}}}$: diagonal matrix, projector onto vertices *not* in H
  - **Left multiplication**: zero out corr. rows to H
  - **Right multiplication**: zero out corr. cols to H
- $\mathbf{D} = \boldsymbol{diag(A * ones)}$
- $\boldsymbol{b = A * ones}$
- $\boldsymbol{\hat{x}}$: mean hitting time moments from nodes of $G \backslash H$ to H

Hitting time linear system - reformulated (cont.)

A refinement to the left-hand side for convenience

$$\mathbf{\Pi\Pi^T}(\boldsymbol{D}-\boldsymbol{A})\mathbf{\Pi\Pi^T} * \mathbf{\Pi\Pi^T}\boldsymbol{x} = \mathbf{\Pi\Pi^T}\boldsymbol{b}$$

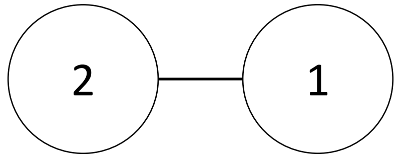
We make the identification:

$$\mathbf{\Pi}\hat{\boldsymbol{x}} = \mathbf{\Pi\Pi^T}\boldsymbol{x}\; where\; \mathbf{\Pi\Pi^T}\boldsymbol{x}(\boldsymbol{i}) = \begin{cases} \hat{\boldsymbol{x}}(\boldsymbol{i})\; for\; \boldsymbol{i} \in G\backslash H \\ \mathbf{0}\; for\; \boldsymbol{i} \in H \end{cases}$$

- The components of $\mathbf{\Pi}\hat{\boldsymbol{x}}$ in the null space of $\mathbf{\Pi\Pi^T}$ are those corresponding to linear combinations of columns of *I* associated with indices of *H* – we choose 0 for the "free parameters" for the solution to the associated homogenous equation; that is, we set $\boldsymbol{x}(\boldsymbol{i}) = 0$ for $\boldsymbol{i} \in H$
- The components of $\mathbf{\Pi}\hat{\boldsymbol{x}}$ orthogonal to the null space of $\mathbf{\Pi\Pi^T}$ are the uniquely defined components of interest, the mean hitting times for $G\backslash H$

Tiny example revisited

Example: Consider a simple two node undirected graph with corresponding matrices:



$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathrm{L} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

If we select node “1” as our hitting set, this gives

$$\Pi\Pi^T = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \Pi\Pi^T \mathrm{L} \Pi\Pi^T = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$ (row/col 1 are “zeroed” out)

The resulting linear system$\mathbf{\Pi\Pi^T}\boldsymbol{L}\mathbf{\Pi\Pi^T} * \mathbf{\Pi\Pi^T}\boldsymbol{x} = \mathbf{\Pi\Pi^T b}$ is

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Grafiki: Custom linear operators

Performance considerations:

Avoid double-storage due to explicit construction of Laplacian – do not construct L!

- Use implicit Laplacian operators - requires only matrix-vector and element-wise multiplication or assignment!

Avoid matrix-matrix operations

- Store diagonal matrix and preconditioner as a vector with custom operators for element-wise operations – perfectly SIMD, all operations local to proc. owning the row

Avoid matrix "reshaping" for each hitting set (large setup costs, requires distributed communication)

- Use "hitting set projectors" stored as vectors in place of Π, Π^T operators - perfectly SIMD, all operations local to proc. owning the row

Avoid communication for hitting set updates

- Hitting set updates require only local update by owning proc. to the vector (no distributed communication needed)

Grafiki: Custom linear operators

Example: Demonstrate details of operator implementation

$$A = \begin{bmatrix} 0 & b & a \\ b & 0 & c \\ a & c & 0 \end{bmatrix} \qquad \mathrm{D} = \begin{bmatrix} a+b & 0 & 0 \\ 0 & b+c & 0 \\ 0 & 0 & a+c \end{bmatrix}$$

Graph adjacency matrix **Row sums of graph adjacency matrix**

$$L = D - A = \begin{bmatrix} a+b & -b & -a \\ -b & b+c & -c \\ -a & -c & a+c \end{bmatrix}$$

Combinatorial Laplacian (explicit)

Grafiki: Operator setup

Compute *D*, but store as a vector $\boldsymbol{d} = \boldsymbol{A} * \boldsymbol{ones}$

$$\boldsymbol{d} \quad \begin{bmatrix} a+b \\ b+c \\ a+c \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & b & a \\ b & 0 & c \\ a & c & 0 \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\boldsymbol{ones}}$$

Create "hitting set" projectors $\boldsymbol{\Pi\Pi^T}$, but store as vector *p*

- Element-wise multiply acts like a "mask", zeros out entries corr. to *H*

Example: take as hitting set *H* the singleton {2}

$$\boldsymbol{p} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Distributed computing notes:

- Distribution of rows of vector across proc.'s matches matrix
- Element-wise multiplication:
  - Each proc. does work local to its owned rows
- Element-wise assignment:
  - During initialization of "hitting set mask", each proc. searches list of hitting set indices, assigns 0 if it owns the row corresponding to the index

Grafiki: Apply operator

$$\begin{bmatrix} x_1 \\ 0 \\ x_3 \end{bmatrix} \leftarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} .* \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$Step\ 1$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \leftarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} - \begin{bmatrix} 0 & b & a \\ b & 0 & c \\ a & c & 0 \end{bmatrix} * \begin{bmatrix} x_1 \\ 0 \\ x_3 \end{bmatrix}$$

$Step\ 3$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \leftarrow \begin{bmatrix} a+b \\ b+c \\ a+c \end{bmatrix} .* \begin{bmatrix} x_1 \\ 0 \\ x_3 \end{bmatrix}$$

$Step\ 2$

$$\begin{bmatrix} y_1 \\ 0 \\ y_3 \end{bmatrix} \leftarrow \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} .* \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$Step\ 4$

Pseudo-code for $\mathbf{\Pi\Pi^T}(\boldsymbol{D} - \boldsymbol{A})\mathbf{\Pi\Pi^T} * \mathbf{\Pi\Pi^T}\boldsymbol{x}$:

```
Step 1: x = $p .* x$  (element-wise multiplication)
Step 2: y = $d .* x$  (element-wise multiplication)
Step 3: y = $y - A * x$  (fused spmv with vector subtraction)
Step 4: y = $p .* y$  (element-wise multiplication)
```

Multiple components

Finding and bookkeeping of multiple components is inefficient and adds costly setup time

Discounting: instead of incrementing by “1” per ”step” of the simulation, increment by

$$\boldsymbol{\alpha}:\ \mathbf{0} < \boldsymbol{\alpha} \leq \mathbf{1}$$

- $\boldsymbol{\alpha}$ referred to as the “discount factor”
- Sum of steps results in geometric series with upper bound $1/(1-\alpha) = \sum_{i=1}^{\infty} \alpha^i$ (for $0 < \alpha < 1$)
- “Origins in the relationship between potentials and Markov chains” – Rich Lehoucq

The linear system update is trivial (as is the corresponding operator implementation):

$$\boldsymbol{\Pi\Pi}^{\mathbf{T}}(\boldsymbol{D} - \boldsymbol{\alpha A})\boldsymbol{\Pi\Pi}^{\mathbf{T}} * \boldsymbol{\Pi\Pi}^{\mathbf{T}}\boldsymbol{x} = \boldsymbol{\Pi\Pi}^{\mathbf{T}}\boldsymbol{b}$$

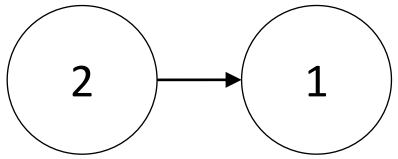
Directed graphs

Directed graphs introduce new complications for the solver:

- Row of zeros: singular coefficient matrix
  - *D* no longer valid as preconditioner
- Asymmetry: Breaks PCG requirements

Example: Consider the two node directed graph

and corresponding matrices:



$$A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \mathrm{L} = \begin{bmatrix} 0 & 0 \\ -1 & 1 \end{bmatrix}$$

If node “1” is our hitting set, we attain a valid result

But if we select node “2” as our hitting set, the solver fails

Directed graphs

To support these added cases we use:

Biconjugate gradient stabilized (BiCGStab)

- Iterative method for solution of nonsymmetric systems
- Available in Matlab (bicgstab) and Trilinos through Belos

A new formulation to the linear system

- Equivalent to previous system for undirected graphs
- Same computational benefits as the previous linear system
- Similar relationship between non-preconditioned use with L_{norm}, and use of L with preconditioning

Linear system - reformulated for directed graphs

To reformulate the linear system…

Recall: the equations derived from the Markov chain,

$(\boldsymbol{I} - \boldsymbol{S}) * \boldsymbol{x} = \mathbf{1}$, where $\boldsymbol{S}$ is a Markov transition matrix

We set $\boldsymbol{S} \leftarrow \mathbf{D^{-1}A}$ under previous assumptions about $\boldsymbol{A}$

If *A* has row(s) of zeros (e.g. row *i*), we set the corresponding row(s) of *S* as

$$\boldsymbol{S}(\boldsymbol{i},\boldsymbol{j}) = \begin{cases} \mathbf{1}\ if\ \boldsymbol{i} = \boldsymbol{j} \\ \mathbf{0}\ if\ \boldsymbol{i} \neq \boldsymbol{j} \end{cases}$$

This motivates defining $\widetilde{\boldsymbol{D}}$:

$$\boldsymbol{e}_i^T \widetilde{\boldsymbol{D}} \boldsymbol{e}_i = \begin{cases} \boldsymbol{e}_i^T \boldsymbol{D} \boldsymbol{e}_i\ for\ \boldsymbol{e}_i^T \boldsymbol{D} \boldsymbol{e}_i > \mathbf{0} \\ \mathbf{1}\ for\ \boldsymbol{e}_i^T \boldsymbol{D} \boldsymbol{e}_i = \mathbf{0} \end{cases}$$

And the updated operator $\boldsymbol{D} \widetilde{- \boldsymbol{\alpha}} \boldsymbol{A}$:

$$\boldsymbol{e}_i^T (\boldsymbol{D} \widetilde{- \boldsymbol{\alpha}} \boldsymbol{A}) \boldsymbol{e}_i = \begin{cases} \boldsymbol{e}_i^T (\boldsymbol{D} - \boldsymbol{\alpha} \boldsymbol{A}) \boldsymbol{e}_i\ for\ \boldsymbol{e}_i^T \boldsymbol{D} \boldsymbol{e}_i > \mathbf{0} \\ \mathbf{1} - \boldsymbol{\alpha}\ for\ \boldsymbol{e}_i^T \boldsymbol{D} \boldsymbol{e}_i = \mathbf{0} \end{cases}$$

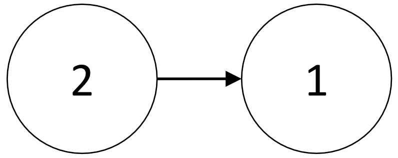
Linear system - reformulated for directed graphs…

The updated linear system for directed graphs:

$$\boldsymbol{\Pi\Pi^{\mathrm{T}}\left(\widetilde{D-\alpha}A\right)\Pi\Pi^{\mathrm{T}} * \Pi\Pi^{\mathrm{T}}x = \Pi\Pi^{\mathrm{T}}b}$$

- Where $\boldsymbol{b} = \boldsymbol{\widetilde{D}} * \boldsymbol{ones}$,
- solved with **BiCGStab**
- and (right) preconditioner $\boldsymbol{M} = \boldsymbol{\widetilde{D}}$
- produces the desired mean hitting times $\boldsymbol{x}$, where
- results corresponding to zero-rows are set to $1/(1-\alpha)$

Tiny example revisited again



Example: Consider a simple two node directed graph with corresponding matrices:

$$A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, D = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \widetilde{D} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \ \widetilde{D - \alpha A} = \begin{bmatrix} 1-\alpha & 0 \\ \alpha & 1 \end{bmatrix}$$

If we select node "2" as our hitting set, this gives

$$\Pi\Pi^{\mathrm{T}} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \Pi\Pi^{\mathrm{T}}\left(\widetilde{D - \alpha A}\right) = \begin{bmatrix} 1-\alpha & 0 \\ 0 & 0 \end{bmatrix}$$ (row 2 and col 2 "zeroed" out)

The resulting linear system $\boldsymbol{\Pi\Pi^{\mathrm{T}}}\left(\widetilde{\mathbf{D} - \boldsymbol{\alpha}\mathbf{A}}\right) * \boldsymbol{\Pi\Pi^{\mathrm{T}}}\mathbf{x} = \boldsymbol{\Pi\Pi^{\mathrm{T}}}\mathbf{b}$ is

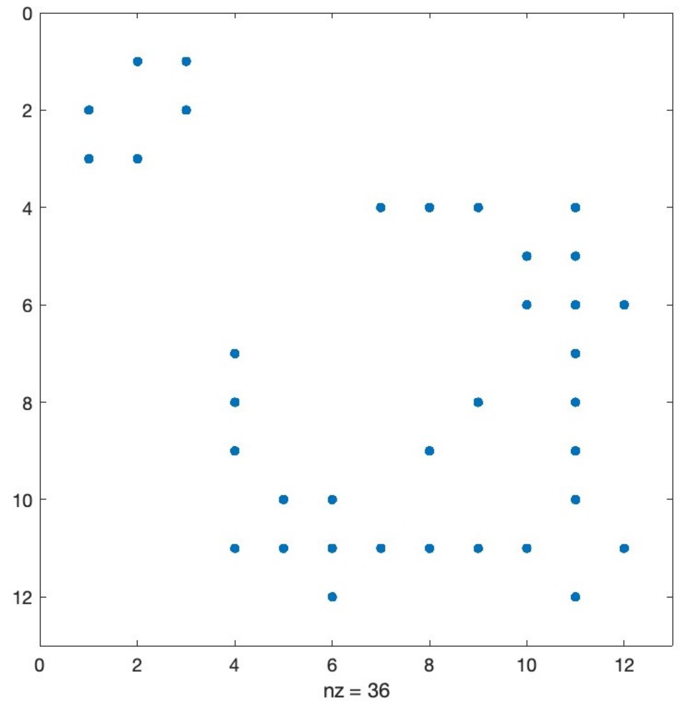
$$\begin{bmatrix} 1-\alpha & 0 \\ 0 & 0 \end{bmatrix} * \begin{bmatrix} x_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Application Requirement: Verification

Solving the same problem with different software/tools

- Research conducted via Matlab/Octave – fast prototyping and testing
- Scale up implementation for large data sets – Grafiki
- Agreement of results essential to confidence in tools

Example: 12 x 12, nnz = 36



Relative difference of mean ht

CG tol = 10e-4, H = {4}

-2.72777540692822e-07
-2.72777540692822e-07
-2.72777540692822e-07
N/A
-6.1877621860528e-08
9.04881754882386e-08
8.69025533663561e-08
-5.48118673616131e-08
2.53207324572656e-07
9.69602156157076e-08
-1.45566037785778e-08
-5.21284232600952e-08

Application Requirement: Deployment

Application research requires deployment of tools to different and unfamiliar compute environments

- Grafiki, through Trilinos, supports MPI builds for distributed computation + threaded and/or accelerator options: **Serial, OpenMP, Cuda, Hip (Experimental)**

Docker containers available for:

- MPI + (multi) NVIDIA GPU
- MPI + OpenMP
- OpenMP

Spack

- In-progress (MPI + OpenMP)

Build from source

- Requires building dependencies
  - BLAS/LAPACK
  - Trilinos

Application Requirement: Multiple GPUs

Performance Results: Grafiki Mean Hitting Times

Data set: BrainGraph.mtx

18808797 x 18808797

Non-zeros (edges): 486315743

System info:

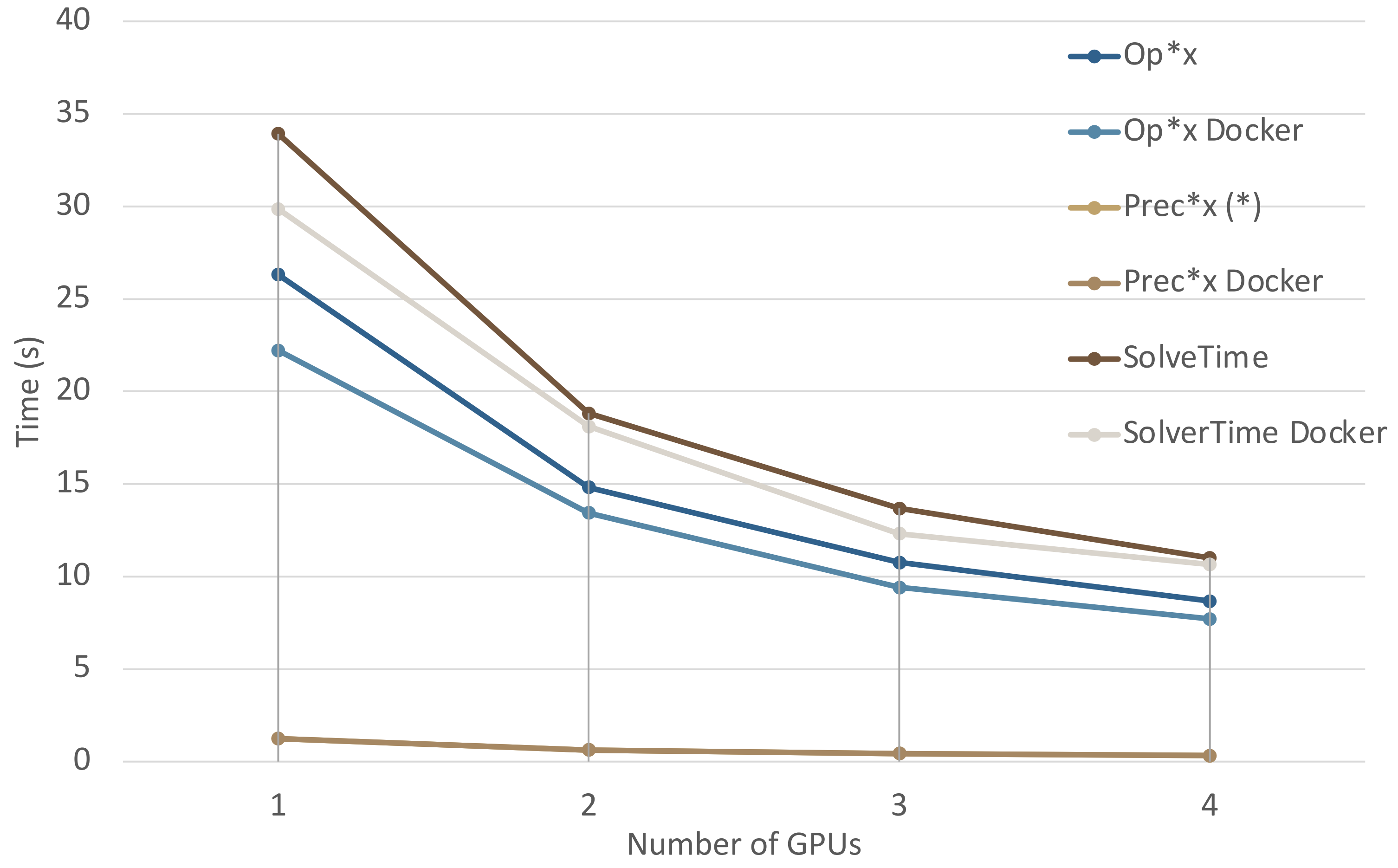
DGX System NVIDIA V100 GPUs

NVIDIA driver: 470.199.02

CUDA version: 11.1

Docker CUDA version: 11.2

OpenMPI: 4.1.1



Ongoing and Follow-up Work

- Verification continued
  - Larger data sets – subset of Wikipedia
  - Increased unification between tool usage

- Software and Research
  - Testing on larger graphs
  - Improved tool interoperability
  - Explore Python interoperability – improved user experience, productivity
  - Custom file I/O improvements for specialized customer data formats