Scalable Knowledge Graph Analytics at 136 Petaflop/s

Ramakrishnan Kannan*, Piyush Sao*, Hao Lu*, Drahomira Herrmannova*, Vijay Thakkar+, Robert Patton*, Richard Vuduc+, Thomas Potok*
* Oak Ridge National Laboratory, Oak Ridge, TN, USA
+ Georgia Institute of Technology, Atlanta, GA, USA

***Abstract*—We are motivated by newly proposed methods for data mining large-scale corpora of scholarly publications, such as the full biomedical literature, which may consist of tens of millions of papers spanning decades of research. In this setting, analysts seek to discover how concepts relate to one another. They construct graph representations from annotated text databases and then formulate the relationship-mining problem as one of computing all-pairs shortest paths (APSP), which becomes a significant bottleneck. In this context, we present a new high-performance algorithm and implementation of the Floyd-Warshall algorithm for distributed-memory parallel computers accelerated by GPUs, which we call DSNAPSHOT (Distributed Accelerated Semiring All-Pairs Shortest Path). For our largest experiments, we ran DSNAPSHOT on a connected input graph with millions of vertices using** 4,096 nodes (24,576 GPUs) **of the Oak Ridge National Laboratory's Summit supercomputer system. We find DSNAPSHOT achieves a sustained performance of** 136×10^{15} **floating-point operations per second** (136 petaflop/s) **at a parallel efficiency of** 90 % **under weak scaling and, in absolute speed,** 70 % **of the best possible performance given our computation (in the single-precision tropical semiring or "min-plus" algebra). Looking forward, we believe this novel capability will enable the mining of scholarly knowledge corpora when embedded and integrated into artificial intelligence-driven natural language processing workflows at scale.**

***Index Terms*—Shortest path problem, High Performance Computing, Parallel Algorithms**

I. Gordon Bell Justification

We computed All-Pairs Shortest Path on a graph with 4.43 million vertices using 4,096 Summit nodes (24,576 GPUs) in 21.3 minutes or 136 petaflop/s (90 % parallel efficiency and 70 % machine-peak for single-precision tropical semiring GEMM). We also processed a 6 million vertex graph, constructed from medical documents between 2010-2015, in 80 minutes.

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

TABLE I
Gordon Bell Performance Attributes

Attributes	Category	Details
Category of achievement	Peak Performance Scalability	136 petaflop/s Weak scaling to 6 million vertices
Type of method used	N/A	N/A
Results reported on the basis of	Whole application except I/O and Init	End-to-end APSP
Precision	Single (FP32)	
System scale	Measured	90 % of Summit (4,096 nodes 24,576 GPUs)
Measurement mechanism	Timers	Direct timer instrumentation

II. Overview of the Problem

The scientific literature is expanding at incredible rates, which were recently estimated to be in the millions of new articles per year [1] and growing exponentially [2]. Extracting information from such vast stores of knowledge is an urgent need, as exemplified by the recent open release of materials relevant to the current SARS-CoV-2 pandemic [3]. Given that the volume of information is easily beyond the capacity of any one person, analysts have been strongly motivated to develop automated knowledge-mining methods and extraction tools [4]–[6].

In this context, our work seeks to develop highly efficient algorithms and software for the analysis and mining of *knowledge graphs* at scale. Knowledge graph construction concerns the acquisition and integration of information into an ontology from which new information can be uncovered. The work in this paper is based more specifically on the process of literature-based discovery [7], [8]. It has been shown that previously unknown relationships exist in the scientific literature that can be uncovered by finding concepts that link disconnected entities [7]–[9]. This process, called *Swanson Linking* [10], is based on the familiar idea of transitivity: if there is no known direct relation between entities A and C, but there are published relations between A and B, and B and

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

C, then one can hypothesize that there is a plausible, novel, yet unpublished indirect relation A-C [8]. For instance, in 1986, Swanson applied this concept to propose a connection between dietary fish oil (A) and Raynaud's disease (C) through high blood viscosity (B), which fish oil reduces [11]. This connection was validated in a clinical trial three years later.

To begin replicating this idea at the full scale of a large corpus, we have created a graph dataset based on Semantic MEDLINE [12], a dataset of biomedical concepts and relations between them extracted from the PubMed database of biomedical literature maintained by the U.S. National Library of Medicine. We have enriched the graph with data extracted from the new COVID-19 Open Research Dataset[1] (CORD-19) [13], of research literature on COVID-19, SARS-CoV-2, and other coronaviruses, which was released by the White House in March of this year [3]. This dataset consists of 18.5 million vertices representing over 290 thousand unique biomedical concepts and the publication abstracts from which these concepts were extracted. The roughly 213 million relations between these vertices represent 1) existing published relations between biomedical concepts, 2) relations between concepts and publication abstracts in which they appear, and 3) citation relations between the abstracts.

The analysis of this dataset presents several challenges. The graph is not only extremely large, but is also constantly growing. For example, adding research articles on COVID-19 published just this year enlarged the graph by several thousand vertices and tens of thousands of new edges. To aid the analysis of such a large graph, we are developing an enterprise architecture for biomedical knowledge graph analytics whose overall workflow appears in Figure 1. It incorporates (a) streaming publication data from different sources and the knowledge graph representation of these biomedical texts; (b) various client services like natural language processing and visualization; and (c) a backend query engine for operating on the knowledge graph, which invokes operations such as All-Pairs Shortest Path (APSP), which is the bottleneck of this paper's application case study (Section VI-C).

For this paper's application, the central bottleneck is APSP. We develop a scalable algorithm for it, which we call the Distributed Accelerated Semiring All-Pairs Shortest Path (DSNAPSHOT) algorithm. It implements a GPU-accelerated, distributed-memory parallel version of the Floyd-Warshall (FW) algorithm targeted at the Summit supercomputer system at Oak Ridge National Laboratory. Our approach can perform well because at the heart of FW lies matrix-multiplication-like (level-3 BLAS-like) operations, which are well-suited to GPUs [14] and distributed memory systems [15].

In its best configuration, DSNAPSHOT achieves 136 Petaflop/s in single-precision. It takes 21.3 minutes to process a large graph composed of 4.43 million vertices, which required execution of 170 exa–floating–point operations. Based on our performance model (Section V-C) and experiments, we estimate that for the 18 million vertex biomedical knowledge graph, it will take about 18 hours to compute the APSP.

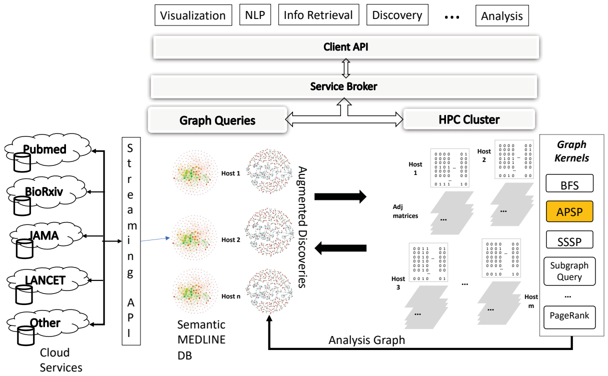


Fig. 1. Workflow for biomedical knowledge graph analytics

The key enabling ideas in our work are as follows.

- Our approach leverages GPU acceleration and communication optimizations for a two-dimensional distributed FW algorithm, which enabled scaling of APSP to inputs with millions of vertices.
- We show, for the first time, performance of 136 PF/s on the world's fastest supercomputer, Summit, for a graph analysis algorithm based on state-of-the-art practices.
- We model the Semantic MEDLINE dataset as a matrix suitable for HPC algorithms, thereby enabling the extraction of useful information at scale.
- We show that APSP algorithms offer the potential to uncover novel relations from biomedical knowledge graphs at scale.

The data [16] used in these experiments are being publicly released in an open-source form.

III. CURRENT STATE-OF-THE-ART

Summaries of closely related state-of-the-art in algorithms for biomedical knowledge graphs and APSP appear below.

A. *Biomedical Knowledge Graph Mining*

In the biomedical field, the relations between biomedical concepts—such as drugs, diseases, symptoms, proteins, and genes—play an important role in tasks like drug discovery and repurposing [17], [18]. Much previous work has focused on leveraging natural language models for speeding up these tasks [9], [19], [20]. However, these methods are often black boxes that require significant additional work to provide an explanation [20]. Furthermore, these methods are often limited to providing results for a specific query (for example, a specific drug and a disease) [9], [21], significantly limiting their scope and increasing the work and background knowledge required to use them.

Consequently, there is interest in mining knowledge graphs created from scholarly literature. Many different approaches

[1] https://www.semanticscholar.org/cord19

TABLE II
Gordon Bell Justification. Fastest and Biggest runs of dSnapshot on 4096 Summit nodes. For Metrics refer Section VI-3

Nodes	MPI (P)	Row (P_x)	Col (P_y)	Vertices (n)	Memory(in TB)	Time (in Secs)	PFlop/s	Fraction of Peak	Parallel Efficiency
Biggest Real World Graph (See VI-C1)									
4,096	147,456	384	384	5.95×10^6	393	4.79×10^3	89.96	46 %	60 %
Fastest Run									
4,096	24,576	192	256	4.43×10^6	209	1.28×10^3	135.9	70 %	90 %

have been applied with the goal of performing automated hypothesis generation and literature-based discovery, such as statistical methods and pattern matching [7], link prediction [22], association rule mining [23], manually constructed queries [24], and graph statistics [25]. A recent survey reviews literature-based discovery and hypothesis generation [5].

Several previous works have explored using shortest-path computations for discovering novel connections between biomedical concepts [21], [26], [27]. Like our method, these approaches use shortest path calculation to discover pathways between pairs of entities that do not otherwise have a direct connection between them. However, these approaches need a starting query (a pair of entities) [21]; or reduce the size of the graph to specific types of nodes and connections [26] or to specific topics [27]. To the best of our knowledge, dSnapshot is the first method capable of calculating shortest path between all pairs of entities in a biomedical knowledge graph, thereby enabling the discovery of meaningful relations across the whole of biomedical knowledge.

B. Distributed Semiring-based All-Pairs Shortest Path

The seminal work of Carre and others establishes the equivalence between finding shortest paths and solving a system of linear equations [28], [29]. There are several modern treatments of this subject as well [30]–[32].

While Apsp is the semiring-equivalent of matrix inversion, no truly sub-cubic (Strassen-like) algorithm for Apsp is known. The best known complexity of Apsp for the dense case is $\mathcal{O}\left(n^{3-o(1)}\right)$ [33] and $\mathcal{O}\left(\frac{mn}{\log n}\right)$ for sparse graphs [34]. For the parallel case, the complexity is $\mathcal{O}(\log n)$ due to Tishkin [35].

A recent distributed 2.5-dimensional Apsp for a CPU-only cluster exhibited excellent strong scaling [15]. But in absolute performance, it achieved only about 10–25 % of peak, with maximum tested problem sizes of $n = 65{,}536$. A distributed GPU Apsp showed good performance for smaller clusters [36]. However, possibly due to their centralized communication scheme, they do not scale beyond clusters of 64 GPUs.

IV. Background

Many path problems in graph analysis can be described succinctly in a semiring algebra. We review this formalism and the resulting classical and blocked FW algorithms for Apsp, below.

TABLE III
Notation

Symbol Type	Symbol	Description
Process	P	Number of MPI Processes
	P_x, P_y	Row and Column Processes
	$P_x(k)$	($k \mod P_x$)-th Process Row
	$P_y(k)$	($k \mod P_x$)-th Process Column
	N	Number of physical nodes
Matrix	A	Adjacency matrix of a graph
	$A(:, k)$	$A(k : n, k)$
	$A(k, :)$	$A(k, k+1 : n)$: k-th A panels
Graphs	G	Input graph as in Figure 3
	V	Vertex set
	E	Edge set
	n	Number of vertices
	m	Number of edges

a) Notation and terminology: Let $G = \{V, E, W\}$ be an undirected weighted graph with a vertex set V containing $n = |V|$ vertices or nodes, edge set E with $m = |E|$ edges, and weights W, defined below. Say, $e_{i,j}$ is the edge between i-th vertex by v_i and an edge between v_i and v_j. The weights are represented by W, a sparse symmetric matrix and whose entry $w_{i,j}$ denotes the distance between vertices v_i and v_j if $e_{i,j} \in E$; otherwise, $w_{i,j} = \infty$. We are also presenting the relevant notations used in the paper frequently in Table III.

A. *Classical* FW *algorithm*

Algorithm 1 FW algorithm for Apsp

```
function FloydWarshall($G = (V, E)$):
    Let $n \leftarrow \dim(V)$
    Let $\text{Dist}[i, j] = \begin{cases} w_{i,j} & \text{if}(i, j) \in E \\ \infty & \text{otherwise} \end{cases}$
    for $k = \{1, 2 \ldots, n\}$ do:
        for $i = \{1, 2 \ldots, n\}$ do:
            for $j = \{1, 2 \ldots, n\}$ do:
                $\text{Dist}[i,j] = \min\{\text{Dist}[i,j], \text{Dist}[i,k] + \text{Dist}[k,j]\}$
    Return Dist
```

During the computation of Apsp, FW maintains and updates a 2-D array of distances, Dist. At any k^{th} iteration, FW maintain the invariance that $\text{Dist}[i, j]$ holds the current shortest distance between v_i and v_j with all intermediate vertices $k \in (v_1, v_2, \cdots, v_k)$ so far. This is realized by the following update equation in FW.

$$\mathrm{Dist}^k[i,j] \leftarrow \min\left\{\mathrm{Dist}^{k-1}[i,j], \mathrm{Dist}^{k-1}[i,k] + \mathrm{Dist}^{k-1}[k,j]\right\}.$$

The above invariance is always satisfied when there are no cycles of negative weight sum. In the case of negative cycles, it is trivial to say that the shortest path length will be ∞.

When we have explored all paths between any two pairs of vertices v_i to v_j with all the vertices as intermediaries (i.e., $\forall k \in V$), then $\mathrm{Dist}[i,j]$ will be the APSP. The computation can be done in-place, obviating the need for two separate copies, Dist^{k-1} and Dist^k. This is realized as Algorithm 1.

For analysis purposes, we will assume G is a single connected component, i.e., that there exists a path between any pair of two vertices v_i and v_j resulting in a fully dense Dist matrix.

B. MIN-PLUS *Matrix Multiply or "SemiRing GEMM"*

APSP may be understood algebraically as computing the matrix closure of the weight matrix, W, defined over the tropical semiring [37]–[39]. In more basic terms, let $\oplus$ and $\otimes$ denote the two binary scalar operators

$$\begin{aligned} x \oplus y &:= \min(x,y) \\ x \otimes y &:= x + y, \end{aligned}$$

where x and y are real values or ∞. Next, consider two matrices $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$. The MIN-PLUS product C of A and B is

$$C_{ij} \leftarrow \sum_k^{\oplus} A_{ik} \otimes B_{kj} = \min_k \left(A_{ik} + B_{kj}\right). \qquad (1)$$

Sao et al. [32] discuss in detail the connections between the semiring GEMM and the APSP.

C. Baseline: A 2D Distributed Floyd Warshall

We can design a baseline FW algorithm using the Message Passing Interface (MPI) for expressing the distributed memory parallelism. The MPI processes are logically arranged in a two-dimensional (2D) process grid. On this 2D process grid, DSNAPSHOT distributes the input matrix A in a block cyclic fashion.

In the Figure 2, we show various MPI communicator and communication patterns involved in the Algorithm 2. We show the k-th process row $P_x(k)$ and process column $P_y(k)$ by dotted rectangles in the Figure 2. In k-th diagonal update step, process p_{kk} performs a local FW computation and broadcast it across $P_y(k)$ and $P_x(k)$. In k-th panel broadcast, each process in $P_x(k)$ broadcast calculated $A(:,k)$ panel to their process column $P_y(k)$, and similarly each process in $P_x(k)$ broadcast the calculated $A(k,:)$ panel to their process row $P_x(k)$.

On obtaining the k-th panels, $A(k,:)$ and $A(:,k)$, a process can update the blocks of $A(k+1:n_s, k+1:n_s)$ that it owns. This update is the MIN-PLUS Outer Product, and it invokes the proposed Semi-Ring GEMM kernel (Section V-A) on GPU.

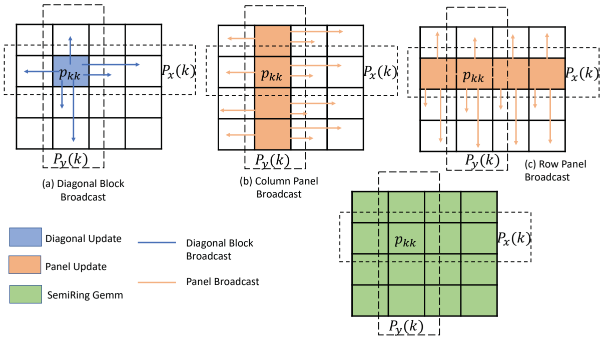


Fig. 2. Communication patterns in DSNAPSHOT

Algorithm 2 A Baselines 2D Distributed Block FW

```
Input: Distributed sparse matrix $A$;
On each MPI process $p_{id}$ do in parallel:
for $k = 1, 2, 3 \ldots n_b$ do
    Synchronize all processes
  Diagonal Update
    if $p_{id}$ owns $A(k,k)$ then
        $A(k,k) \leftarrow \mathrm{FW}(A(k,k))$
        Send $A(k,k)$ to $P_x(k)$ and $P_y(k)$
  Panel Update
    if $p_{id} \in P_y(k)$ then
        Wait for $A(k,k)$
        $A(k,:) \leftarrow A(k,:) \bigoplus A(k,k) \otimes A(k,:)$
        Send $A(k,:)$ blocks to needed processes in $P_x(:)$
    else
        Receive $A(k,:)$ blocks if needed
    if $p_{id} \in P_x(k)$ then
        Wait for $A(k,k)$
        $A(:,k) \leftarrow A(:,k) \bigoplus A(:,k) \otimes A(k,k)$
        Send $A(:,k)$ blocks to required processes in $P_y(:)$
    else
        Receive $A(:,k)$ blocks if required
  MinPlus Outer Product
    for $i = \{1, 2 \ldots, n_b\}, i \neq k$ do:
        for $j = \{1, 2 \ldots, n_b\}, j \neq k$ do:
            $A(i,j) \leftarrow A(i,j) \bigoplus A(i,k) \otimes A(k,j)$
```

D. Biomedical Knowledge Graph

As mentioned in Section II, our biomedical knowledge graph was constructed using the Semantic MEDLINE[2] database [12]. The latest version of Semantic MEDLINE (`semmedVER40_R` as of April 28, 2020) contains nearly 98 million predications (concept-to-concept relations) extracted using the SemRep library[3] from over 18 million biomedical abstracts. We have enriched the dataset with concepts and relations extracted using SemRep from the CORD-19 dataset[4] of publications on COVID-19, SARS-CoV-2, and other coronaviruses [3]. Specifically, we have used the version from June 30, 2020, which contains over 130 thousand publication abstracts. Here we describe the construction of the graph from the two datasets.

The graph is composed of two types of nodes:
1) Concept nodes: represent unique biomedical terms, for example, drugs, genes, diseases, and symptoms (there are 127

[2] https://skr3.nlm.nih.gov/
[3] https://semrep.nlm.nih.gov/
[4] https://www.semanticscholar.org/cord19

different concept types). There are over 290 thousand unique concept nodes.

2) Abstract nodes: represent the 18 million PubMed abstracts and 130 thousand CORD-19 abstracts.

As shown in Figure 3, the nodes can be connected in three different ways:

1) Concept to concept relations: The connections between concepts represent relationships described in these abstracts, for example, the sentence, "Zika virus is a member of the family Flaviviridae," would result in a "part of" relation between "Zika" and "Flavivirida," while both concepts would be tagged with "virus" label. In the Semantic MEDLINE database, these relations are represented as predications. There are 14 million unique concept to concept relations in the graph which were extracted from the 98 million Semantic MEDLINE predications. For the shortest path computation, we assign these edges Jaccard similarity score of the connected concepts, which is calculated as the number of times the two concepts appear together in a predication divided by the total number of predications these concepts appear in. These connections are represented in blue in Figure 3.

2) Concept to abstract relations: The connections between abstracts and concepts represent occurrence of concepts in abstracts. For example, if the above sentence, "Zika virus is a member of the family Flaviviridae," appeared in abstract with PubMed ID 111, there would be a connection between the abstract node "PMID111" and concepts "Zika" and "Flaviviridae." There are 196 million unique concept to paper connections in the graph. For the shortest path computation, we assign these edges a weight representing the number of times a concept c appears in abstract p divided by the total number of concepts appearing in p. These connections are represented in red in Figure 3.

3) Abstract to abstract relations: The connections between abstracts represent citation relations between them. For example, if an abstract with PubMed ID 111 cited an abstract with PubMed ID 222, there would be a connection between those two abstracts in the graph. There are 3 million citation relations in the graph. For the shortest path computation, we treat citations as undirected edges and assign them a weight calculated as $1/(N_{p_1} + N_{p_2})$, where N represents the total number of citation relations of p. These relations are represented in yellow in Figure 3.

In total, the graph is composed of nearly 18.5 million nodes (over 290 thousand unique concepts and over 18 million publications), and 213 million edges. Figure 3, which was extracted from the graph, shows both types of nodes and all three types of relations. Additional information about how the graph was produced is provided in our data read me [16].

V. Innovations Realized: Distributed Accelerated Semiring All-Pairs Shortest Path (dSnapshot)

Our algorithm, Distributed Accelerated Semiring All-Pairs Shortest Path, is the 2-D distributed-memory FW variant of Algorithm 2 that offloads semiring GEMM computations to the GPU. We perform computations on the GPU or CPU based on their nature and offloading costs. The SemiRing GEMM (SRGEMM) of Section IV-B has the same acceleration opportunities of classical GEMM due to its high arithmetic intensity and identical data access pattern. However, the diagonal update is performed on smaller portions of the matrix and involves sequential computation that are difficult to accelerate on a GPU. Hence, all panel updates and the Min-Plus outer product are performed on the GPU using the SRGEMM described in Section V-A.

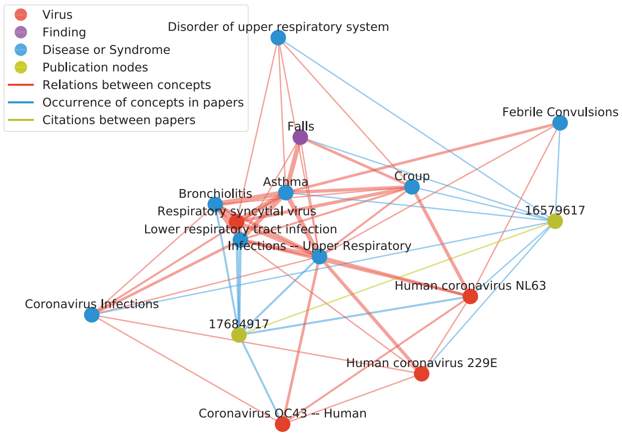

● Virus
● Finding
● Disease or Syndrome
● Publication nodes
Relations between concepts
Occurrence of concepts in papers
Citations between papers
Disorder of upper respiratory system
Febrile Convulsions
Falls
Croup
Asthma
Bronchiolitis
Respiratory syncytial virus
Lower respiratory tract infection
Infections -- Upper Respiratory
16579617
Human coronavirus NL63
Coronavirus Infections
17684917
Human coronavirus 229E
Coronavirus OC43 -- Human

Fig. 3. Example knowledge graph generated from Semantic MEDLINE. In the figure, yellow nodes represent publication abstracts, which are identified by their PubMed ID. All the other nodes are concepts, with concepts of different types (e.g. virus, syndrome) distinguished by color. The thickness of the edges represents edge weight.

dSnapshot is flexibly implemented so that it can perform communication from both CPU and GPU. In Algorithm 3, we explain the conventional model of communication from the CPU. Even though the panel updates are performed on the GPU, to broadcast it on the process rows $P_x(k)$ or $P_y(k)$, there is a memory transfer to CPU involved In the experiments, we call this variant of algorithm as dSnapshot and any additional communication optimizations are named appropriately.

A. GPU Acceleration

As detailed in Section IV, the primary compute kernel in our blocked Floyd-Warshall algorithm is matrix multiplication over the tropical semiring. Although this $(\mathsf{min}, +)$ SRGEMM kernel is semantically different from the traditional level-3 BLAS multiply-accumulate GEMM operation, it lends itself to the same acceleration opportunities due to its very high arithmetic intensity and identical data access pattern. We implement our Min-Plus SRGEMM by extending the NVIDIA Cutlass open-source linear algebra framework [40].

We wish to modify the traditional BLAS GEMM of the form $\mathbf{C} = \alpha\mathbf{AB}+\beta\mathbf{C}$ with semiring GEMM. We made the following modifications to Cutlass to be able to implement our SRGEMM kernel,

Algorithm 3 Distributed Accelerated Semiring All-Pairs Shortest Path (dSnapshot)

```
Input: Distributed sparse matrix $A$;
On each MPI process $p_{id}$ do in parallel:
for $k = 1,\ 2,\ 3 \dots\ n_b$ do
  Synchronize all processes
  Diagonal Update
  if $p_{id}$ owns $A(k,k)$ then
    Copy Diagonal Blocks from GPU $A_{\text{GPU}}(k,k)$ to CPU $A(k,k)$
    $A(k,k) \leftarrow$ FW $(A(k,k))$
    Send $A(k,k)$ to $P_x(k)$ and $P_y(k)$
  Panel Update
  if $p_{id} \in P_y(k)$ then
    Wait for $A(k,k)$
    Copy $A(k,k)$ from CPU to GPU $A_{\text{GPU}}(k,k)$
    SRGEMM_GPU($A_{\text{GPU}}(k,k), A_{\text{GPU}}(k,:), A_{\text{GPU}}(k,:)$)   ▷ // $A_{\text{GPU}}(k,:) \leftarrow A_{\text{GPU}}(k,:) \bigoplus\ A_{\text{GPU}}(k,k) \otimes A_{\text{GPU}}(k,:)$
    Copy $A_{\text{GPU}}(k,:)$ from GPU to CPU $A(k,:)$
    Send $A(k,:)$ blocks to needed processes in $P_x(:)$
  else
    Receive $A(k,:)$ blocks if needed
  if $p_{id} \in P_x(k)$ then
    Wait for $A(k,k)$
    Copy $A(k,k)$ from CPU to GPU $A_{\text{GPU}}(k,k)$
    SRGEMM_GPU($A_{\text{GPU}}(:,k), A_{\text{GPU}}(k,k), A_{\text{GPU}}(:,k)$)   ▷ // $A_{\text{GPU}}(:,k) \leftarrow A_{\text{GPU}}(:,k) \bigoplus\ A_{\text{GPU}}(:,k) \otimes A_{\text{GPU}}(k,k)$
    Copy $A_{\text{GPU}}(:,k)$ from GPU to CPU $A(:,k)$
    Send $A(:,k)$ blocks to required processes in $P_y(:)$
  else
    Receive $A(:,k)$ blocks if required
  MinPlus Outer Product
  for $i = \{1,\ 2 \dots, n_b\},\ i \neq k$ do:
    for $j = \{1,\ 2 \dots, n_b\},\ j \neq k$ do:
      SRGEMM_GPU($A_{\text{GPU}}(i,k), A_{\text{GPU}}(k,j), A_{\text{GPU}}(i,j)$)   ▷ // $A(i,j) \leftarrow A(i,j) \bigoplus\ A(i,k) \otimes A(k,j)$
```

- **Semiring Operators in Matrix Multiplication**: Support for overriding the ring operators in matrix multiply for other semiring operators in a composable fashion.
- **Identity Values:** Support for custom initialization and padding values as identity values of ring operations. This involves initializing registers with ∞ instead of the default zero.
- **Epilogue Operator**: Addition of semiring BLAS epilogue operator in the tropical semiring $(\min, +)$ for an element-wise min with the **C** matrix.

Performance of GPU SRGEMM Kernel: Fused Multiply Accumulate instruction such as `FFMA` on Volta perform the GEMM multiply and accumulate operations in a single instruction. Volta micro-architecture does not support a similar fused min-plus instruction in hardware. As a result, both ring operations must be issued individually, halving the peak possible flop rate from 15.6 TF/s to 7.83 TF/s at single precision for an SXM2 V100. Our SRGEMM implementation achieves 6.81 TF/s at single precision, corresponding to 87% of the peak performance on our target hardware for this workload.

B. Communication Optimizations

The dSnapshot Algorithm 3, performs high volume of communication in the $O(\frac{n^2}{P_x} + \frac{n^2}{P_y})$. The communication complexity is explained in the next section V-C. This means, during scaling to large nodes, we will spend more time in communication over computation. In order to tame this communication, we propose the alleviation techniques (a) Lookahead - accelerating the critical path of diagonal update and broadcast (b) Ring broadcast protocol (c) Rank Mapping and (d) optimizing intranode communication.

1) Lookahead Technique: The main objective of the lookahead technique is to accelerate the execution of the critical path in the computation and to overlap communication with computation. The diagonal and the panel update lies in this critical path. When the $A(k,:)$ in iteration k has been updated by the column processes P_y and broadcast to row processes P_x as in Figure 2(b), the globally next urgent job is to perform both the diagonal and panel update and communication of the of $A(k+1, k+1), A(k+1,:)$ by the next column process P_{y+1}. A similar argument is also applicable for row process P_x for block $A(:, k+1)$.

In the case of lookahead dSnapshot, as soon as $P_{k+1,k+1}$ receive the $A(k,:)$ and $A(:,k)$ panels, $P_{k+1,k+1}$ perform a SRGEMM using these panels to obtain $A(k+1, k+1)$. The diagonal update on $A(k+1, k+1)$ is performed and broadcast to row and column processes in P_{x+1} and P_{y+1} respectively. Then for, P_{x+1} and P_{y+1}, has the panels $A(k,:)$ and $A(:,k)$ and along with the updated diagonal block $A(k+1, k+1)$ to perform the panel updates/broadcast $A(k+1,:), A(:,k+1)$ using SRGEMM before the Min-Plus Outer Product on k^{th} iteration. By this, the diagonal and panel broadcast can be advanced and the global critical path is accelerated.

The lookahead dSnapshot, requires some small block SRGEMM to be invoked and also additional buffer management.

2) Ring Broadcast over Tree Based Broadcast: The traditional library provided MPI_BCast may not be most efficient for our application since it uses a *kd-tree* pattern (also called hyper-cube algorithm) that costs $\log P(\alpha + w\beta)$ for broadcasting w units of data among P processes. Such an algorithm balances latency (α term) and bandwidth (β term) costs. In contrast, bandwidth is of greater concern for our application thus a broadcast based on ring-pattern that costs $(P-1)\alpha + w\beta$, which is optimal in bandwidth and worst in latency costs. The latency cost in the ring broadcast can be hidden by pipelining broadcasts from different iterations. We implemented a non-blocking version of the ring-broadcast that we use in the dSnapshot.

3) Optimal Rank Placement: When creating a two-dimensional logical process grid using MPI, by default all the MPI ranks within a node will be placed in a process row or process column. However, this rank placement is not optimal considering data transfer via the network interface card (NIC) in a single node. To minimize the data transfer via NIC, we must have ranks within a node arranged in a 2D grid with the same aspect ratio as the logical process grid. In other words, if $Q_r \times Q_c$ ranks per physical node with a total of $P_x \times P_y$ logical MPI processes, then NIC data transfer is minimized when $\frac{Q_r}{P_x} \approx \frac{Q_c}{P_y}$. In Summit supercomputer, we achieve this using the so-called explicit resource file (ERF). The optimal Rank placement benefited all dSnapshot variants except LA+Ring. We observed that

ERF increased the synchronization time for diagonal and panel update of dSnapshot negating the benefits of ring broadcast.

4) Optimizing Intra-Node communication: To minimize data transfer via NIC, it is imperative that we efficiently exploit architectural features for intranode communication. For optimal intranode GPU to GPU transfer, we must use NVlink, and within node CPU to CPU transfer, we use hyper transport. We use GPUDirect for efficient intranode communication. We also observed that GPUDirect did not improve performance over CPU based ring broadcast. This is because GPUDirect does not work well with look-ahead and limits amount of communication and computation overlap.

C. dSnapshot *Analysis*

We present the computation, communication and total cost analysis of dSnapshot in this section.

1) Computation Cost: In the blocked FW algorithm, the total number of floating-point operations is $2n^3$ distributed among P processes. Since the computation is uniform and load-balanced, hence the cost of floating-point operations is $\frac{2n^3}{P}\gamma$, where γ is the cost of unit floating-point operations.

$$T_{\text{comp}} = \frac{2n^3}{P}\gamma \tag{2}$$

2) Communication Cost: If b is the block-size used for block-cyclic data distribution, then algorithm-2 performs the $\frac{n}{b}$ outer loop iterations. In each of the iterations, each process participates in two broadcasts $\frac{nb}{P_x}$ across process row and $\frac{nb}{P_y}$ across process column. In the ring broadcast, the total cost of the two broadcast is $2\alpha + \beta(\frac{nb}{P_x} + \frac{nb}{P_y})$, where α is the setup cost of sending a message and β is cost of sending a unit float word. Since the outer iteration runs for $\frac{n}{b}$ iterations, hence the total communication cost is $2\frac{n}{b}\alpha + \beta(\frac{n^2}{P_x} + \frac{n^2}{P_y})$.

$$T_{\text{comm}} = 2\frac{n}{b}\alpha + \beta n^2 \left(\frac{1}{P_x} + \frac{1}{P_y}\right) \tag{3}$$

3) Total Cost: Depending on n and P, either T_{comp} or T_{comm} will dominate the total cost of computation. In the ideal case, we can completely overlap the communication with computation or vice-versa. In that case, the total cost is given by:

$$T_{\text{ideal}} = \max\left\{\frac{2n^3}{P}\gamma,\ 2\frac{n}{b}\alpha + \beta\left(\frac{n^2}{P_x} + \frac{n^2}{P_y}\right)\right\}. \tag{4}$$

On the other hand of the spectrum, in the worst case, communication and computation will not overlap at all, in which case the total cost is given by:

$$T_{\text{worst}} = T_{\text{comp}} + T_{\text{comm}} = \frac{2n^3}{P}\gamma + 2\frac{n}{b}\alpha + \beta\left(\frac{n^2}{P_x} + \frac{n^2}{P_y}\right). \tag{5}$$

These models acted as guidelines to validate the scaling experiments. Most of the experiments was cross validated against this model and in Section VI-B6, we present the application of the model on the weak scaling experiments.

VI. How Performance Was Measured - Experiments

The aim of our experiments is to understand dSnapshot performance in the context of the growth trends in scientific articles, which is generally accepted to grow exponentially with time [2]. This number doubles about every nine years [2], and in 2014, it has been estimated that the number of English scientific articles in existence is at least 114 million [41].

To assess dSnapshot's ability to respond to this growth, we consider several categories of experiments. First, we look at dSnapshot's single-node efficiency compared to other Apsp baselines (Sections VI-2) on a variety of real-world graphs that fit within a node (Section VI-4). These experiments help establish whether node-local code is a good building block for the distributed implementation. Secondly, we conduct strong-scaling experiments, which allows us to estimate how efficiently dSnapshot can process a fixed-size corpus with increasing numbers of processing nodes and GPUs. Lastly, we conduct weak-scaling experiments, in which we fix the per-node problem size of $O(n^3/p)$. These correspond to the growing literature and the efficiency with which dSnapshot can scale to accommodate that growth.

1) Test Bed: The Summit system consists of 4,608 nodes nodes. Each node has two 22-core IBM POWER9 processors and six NVIDIA Volta V100 GPUs, connected by NVLINK-2, which has a peak performance bidirectional bandwidth of 100 GB/s. V100 GPU has 5,120 cores operating at 1.53 GHz, which translates to theoretical peak of 7.85 TF/s and with FMA, the single precision peak is 15.7 TF/s. The peak memory bandwidth of each V100 GPU is 900 GB/s. Each node contains 512 GB main memory, while each GPU contains 16 GB HBM2 memory. The nodes are connected with a Mellanox Infiniband fat-tree interconnect. Each node is equipped with a 1.6 TB NVMe burst buffer device.

2) Apsp *Baselines:* There are no off-the-shelf distributed scalable FW algorithms available in the public domain. Hence, we compare the single-node performance of our algorithm with the following single-node implementations:

- **BlockedFw-CPU (BFW-CPU)** : This implementation is an efficient multithreaded OpenMP variant, which performs n^3 operations.
- **Dijkstra**: This algorithm performs a single-pair shortest path from all the vertices. It has the lowest asymptotic complexity of all the methods considered herein. Hence, we consider this as the baseline in Figure 5 for comparison against other baseline algorithms.
- **BoostDijkstra (Boost-D)**: This Apsp implementation uses Dijkstra's algorithm from the popular Boost Graph Library (BGL) [42]. BGL also provides a BFW that is slower than BoostDijkstra, so we omit it.
- **Δ-Step**: We use the parallel Δ-stepping variant of Dijkstra's algorithm [43] for computing the single-source-shortest path in Johnson's algorithm. We use the parallel Δ-stepping algorithm from the Galois Graph library [44]. The Δ-stepping requires tuning a Δ parameter for each

TABLE IV
Real World Datasets from SuiteSparse Matrix Collection [45]

Dataset	Label	Rows	NNZ	Kind
		Weighted Graphs		
`human_gene1`	HG1	2.23×10^4	2.47×10^7	Undirected
`mycielskian15`	MY	2.46×10^4	1.11×10^7	Undirected
`human_gene2`	HG2	1.43×10^4	1.81×10^7	Undirected
`appu`	APPU	1.40×10^4	1.85×10^6	Directed
`vsp_msc`	VSP	2.20×10^4	2.44×10^6	Random
		Scientific Problems		
`pkustk08`	PS8	2.22×10^4	3.23×10^6	Structural
`tsyl201`	TSY	2.07×10^4	2.45×10^6	Structural
`trdheim`	TRD	2.21×10^4	1.94×10^6	Structural
`Zd_Jac3`	ZD	2.28×10^4	1.92×10^6	Simulation
`crystk03`	C03	2.47×10^4	1.75×10^6	Materials
`nd6k`	ND6K	1.80×10^4	6.90×10^6	2D/3D Mesh
`pkustk07`	PS7	1.69×10^4	2.42×10^6	Structural

input graph. Our Δ-Step-based Apsp is autotuned, i.e., we try different values of Δ during the first few SSSP calls and pick the best Δ for rest of the execution.

- **cuGraph**: The NVIDIA RAPIDS cuGraph library is a collection of GPU accelerated graph algorithms. We implemented Apsp based on the RAPIDS reference documentation.[5]

3) Metrics: Theoretical Peak on 4,096 nodes *of Summit:* Given that, we can execute no more than 7.85 TF/s FP32 min-plus operations per GPU, so on 4,096 nodes with 24,576 GPUs, the theoretical peak will be 193 PF/s. We report the performance relative to this value as *Fraction of Theoretical Peak*.

Parallel Efficiency: We report the parallel efficiency relative to best performance achieved on 16 nodes at 6.15 TF/s per GPU shown in Figure 6c. The efficiency can be defined as $(\text{achieved flop rate})/(N \times 6 \times 6.15)$.

4) Test Graphs: Our graph datasets for the single-node experiments use the real-world graphs shown in Table IV. We have chosen the graphs to be sufficiently large while running in a reasonable time on a single node. The Djikstra's and Δ-Step algorithms work on graphs with positive edge weights, so we modify the adjacency matrices from real world and synthetic graphs to have only positive entries.

A. Environment

The software versions used are GCC 6.4.0, IBM Spectrum MPI 10.2.0.0, and CUDA 10.1.243. Summit's `jsrun` tool is used for application launch. No other proprietary software was used in the execution. The entire software stack for the experiments are shown in Figure 4.

B. Performance Results and Observations

1) Effect of Blocksize on SRGEMM Performance: Figure 6a shows the relative performance of single GPU dSnapshot runs, normalized to the observed peak of 6.5 TF/s. We observe increase in performance as block size increases from 1 to 32 due to increase in the compute intensity for SRGEMM. Beginning at block size 64, we achieve peak performance and it flattens out. We obtain the best performance at block size 256, which is used for rest of the experiments. As expected, we observe slight degradation in performance for block size beyond 256, as SRGEMM performs better for outer product type matrix multiplication ($k \ll m, n$). Higher block sizes result in fewer global iteration of dSnapshot, however, since latency is not the bottleneck here, we choose block size of 256 for our experiments.

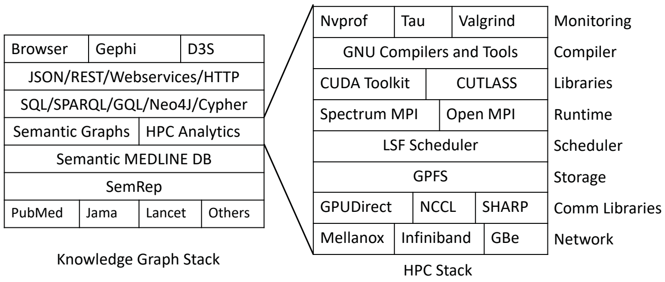


Fig. 4. dSnapshot software stack

2) Single Node Performance Evaluation: We compared SRGEMM against other single-node implementations using the datasets of Table IV, which were taken from the SuiteSparse Matrix Collection [45]. We either developed or obtained open-source baselines for these experiments. The baselines were chosen from different categories, such as sparse algorithms on CPU, and dense algorithms on CPU or GPU implementations. The CPU algorithms were executed on a full single node of Summit with 42 OpenMP threads and the GPUs used only one GPU. We are reporting the wallclock time in seconds. The results appear in Figure 5.

None of the other implementations are competitive with dSnapshot. Over the BFW-CPU, it is 32× faster on `human_gene1`, with even the lowest speedup being 12× for `appu` and `human_gene2`. We expect **Δ-Step** to be slowest as it is neither work optimal nor scalable. Similarly, BoostDijkstra is not competitive to our own implementation of Dijkstra. Dijkstra can be better over FW for sparse graphs. In our case, most of the datasets are relatively dense and Dijkstra does not perform as well as GPU accelerated FW– dSnapshot. While our implementation of Dijkstra is CPU-based, we could not find an efficient priority queue implementation on GPUs to realize GPU Dijkstra. We observed poor performance from cuGraph SSSP relative to other SSSP baselines. Perhaps this finding can be attributed to the beta release nature of the library. Despite the difference in the flop rates between CPU and GPU, we believe the comparison reflects the availability of the current state-of-the-art on different architectures.

3) dSnapshot *Variations:* Having established the single-node dSnapshot as a strong baseline, we next benchmarked three different variants of dSnapshot. dSnapshot is the algorithm explained in the Listing 3. LA is extension of dSnapshot with the lookahead (LA) as detailed in Section V-B. Additionally,

[5] https://docs.rapids.ai/api/cugraph/stable/api

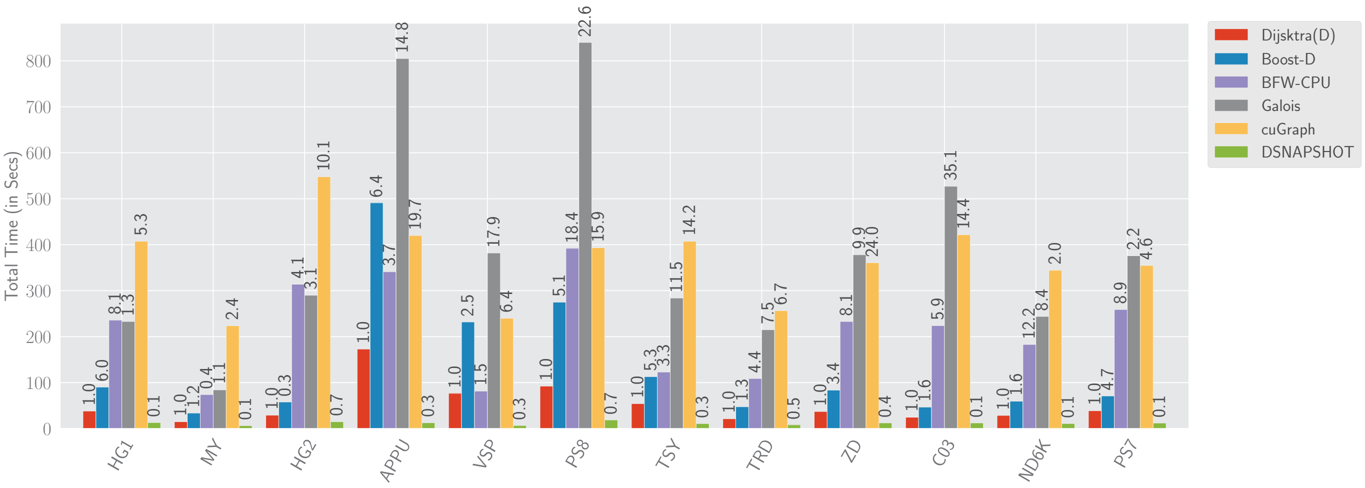


Fig. 5. Time to solution for different algorithms on real world datasets. The text label over each bar denotes the speedup over the reference algorithm, Djikstra. Datasets are labeled according to Table IV.

for MPI broadcast, we leveraged the ring broadcast called as LA+Ring. In all three variants, DSNAPSHOT, LA, and LA+Ring, all MPI communications were performed using host side buffers. Any GPU device buffers required for communication were staged through host memory via a device to host memory copy. In the next two variants, we leveraged CUDA-aware MPI communication, performing communication directly from GPU device buffers. In both the cases, to take advantage of GPUDirect communication, we directly pass the GPU buffer as input for the MPI broadcast to the CUDA-aware IBM spectrum MPI implementation. CUDA-aware MPI does not take a separate GPU stream as input. Hence, to maintain parallelism the SRGEMM kernel was invoked in a dedicated stream. We refer to these two, CUDA-aware MPI broadcast, and CUDA-aware MPI ring broadcast variants of DSNAPSHOT as GPUDirect and GPUDirect+Ring respectively.

For both the weak and strong scaling up to 256 nodes, every node had 12 MPI ranks and one GPU for every two ranks. Each MPI rank was allocated six cores and the ratio of $p_r : p_c$ was maintained as 4:3 per node.

4) Strong Scaling: In the case of strong scaling experiment, we ran with $V = 300{,}000$ vertices. Overall, LA+Ring shows the best strong scaling even though LA may be slightly faster for smaller numbers of processors. GPUDirect did not improve performance over CPU based ring broadcast. This is because GPUDirect with asynchronous communication does not work well on IBM Spectrum MPI and limits amount of communication and computation overlap.

5) Weak Scaling: In the case of the weak scaling experiment, we kept the compute per MPI rank fixed. That is, we maintain $O(n^3/p)$ flops per rank across different run.

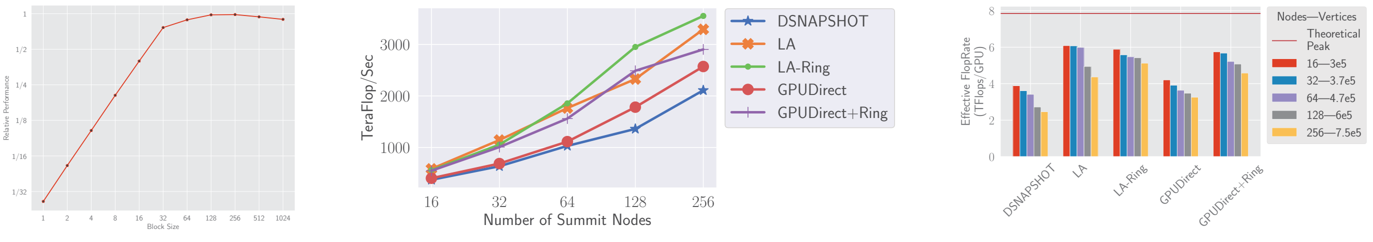
All of the implementations achieve at least 75 % efficiency when weak-scaling to 16× as many processes. In contrast to the strong-scaling experiment, where LA+Ring achieved the best scaling, for weak-scaling the LA implementation achieves marginally better efficiency on fewer nodes.

Summit's relatively fat nodes can achieve petaflop/s on as few as 20 nodes, which should be favorable for both strong and weak scaling. Throughout the weak scaling experiments, we always remain compute-bound. By contrast, at a scale of 256 nodes, strong scaling achieved only 35 % of the efficiency of the semiring GEMM kernel, even when using the best grid configuration. This observation is consistent with the cost model of Section V-C. In this challenging scaling regime, where even minimal extra communication can hurt the performance, we have achieved 136 PF/s on 4,096 nodes of Summit (or 24,576 GPUs), achieving 90 % of the parallel efficiency (i.e., out of 151 PF/s). If we consider parallel efficiency relative to the smallest 16-node run, which achieves 6.12 TF/s per GPU, DSNAPSHOT attains 92 % parallel efficiency at 256 nodes.

6) Cost Model Validation: We compared the experiments against the cost model discussed in Section V-C. Let the best case flop rate be $\Gamma_{\text{ideal}} = \frac{2n^3}{T_{\text{ideal}}}$ and the worst case flop rate be $\Gamma_{\text{worst}} = \frac{2n^3}{T_{\text{worst}}}$, where the best- and worst-case execution times, T_{ideal} and T_{worst}, are as defined in Equation (4) and Equation (5), respectively. The experimental flop rate is computed as $\Gamma = \frac{2n^3}{T}$, where T is measured execution time. Table V presents these three flop rates for our weak scaling experiments. Observe that the experiment always lies in between the best and the worst case rates, and effectively demonstrates the degree to which we were able to overlap computation and communication. Therefore, we can use this model for estimating the running time on the entire 18.5 million vertex biomedical knowledge graph.

C. Case study

We perform two further experiments on the biomedical knowledge graph described in Section IV-D. We share the complete DSNAPSHOT results obtained on the CORD-19 dataset online



(a) (b) (c)

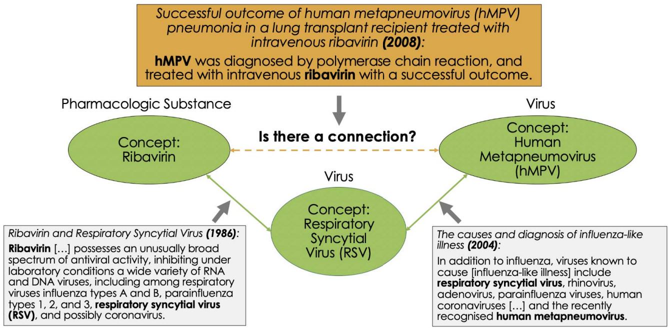
Fig. 6. Figure 6a plots the relative performance of SRGEMM normalized to 6.5 TF/s for different block size parameters. Figure 6b shows strong scaling results with V = 300,000. This plots the performance in teraflop/s as the node count increases. Figure 6c Weak Scaling Experiment. As we increase the number of nodes, we keep the per GPU workload constant. Effective GPU FlopRate is a ratio of the overall flop/s to the number of GPUs.

TABLE V
COST MODEL VALIDATION FOR A WEAK-SCALING EXPERIMENT. THE Γ, Γ_{worst}, AND Γ_{ideal} VALUES ARE SHOWN IN PF/S; SEE SECTION VI-B6.

N	P	P_x	P_y	n	Γ	Γ_{worst}	Γ_{ideal}
16	192	12	16	300,000	0.55	0.4	0.7
32	384	16	24	378,624	1.05	0.8	1.3
64	768	24	32	476,928	2.06	1.6	2.7
128	1536	32	48	600,576	4.07	3.0	5.4
256	3072	48	64	756,480	7.69	5.7	10.8

TABLE VI
SIZE OF GRAPHS BASED ON CORD-19 DATA.

Graph version	2005	2020
Concept vertices	16,427	56,029
Paper vertices	12,863	155,771
Concept-concept edges	25,297	172,254
Concept-paper edges	140,095	1,535,064
Paper-paper edges	1,638	18,666


Pharmacologic Substance
Concept: Ribavirin
Is there a connection?
Virus
Concept: Human Metapneumovirus (hMPV)
Virus
Concept: Respiratory Syncytial Virus (RSV)

*Successful outcome of human metapneumovirus (hMPV) pneumonia in a lung transplant recipient treated with intravenous ribavirin **(2008)**:*
hMPV was diagnosed by polymerase chain reaction, and treated with intravenous **ribavirin** with a successful outcome.

*Ribavirin and Respiratory Syncytial Virus **(1986)**:*
Ribavirin [...] possesses an unusually broad spectrum of antiviral activity, inhibiting under laboratory conditions a wide variety of RNA and DNA viruses, including among respiratory viruses influenza types A and B, parainfluenza types 1, 2, and 3, **respiratory syncytial virus (RSV)**, and possibly coronavirus.

*The causes and diagnosis of influenza-like illness **(2004)**:*
In addition to influenza, viruses known to cause [influenza-like illness] include **respiratory syncytial virus**, rhinovirus, adenovirus, parainfluenza viruses, human coronaviruses [...] and the recently recognised **human metapneumovirus**.

Fig. 7. Example potential path found in 2005 CORD-19 subset.

for use by the community[6] [16].

1) Large knowledge graph: In the first experiment, we use DSNAPSHOT to process a biomedical knowledge graph extracted from research articles published between 2010 to 2015. This graph is composed of 5,953,712 vertices and 445,236,526 edges. To show that APSP can uncover unestablished concept relations, we want to compare historic data with a newer version of the dataset where some new relations appear only in the later literature. To this end, we compute APSP on data from 2010-2015 and plan to use 2015-2020 for validation in a future study. DSNAPSHOT's performance on this biomedical knowledge graph is described in Table II.

2) CORD-19 graph: In the second experiment, we analyze the shortest paths obtained for a knowledge graph built from the CORD-19 dataset [3] (Section IV-D). After removing duplicates (these were identified using their DOI indices), there are 155,771 articles in the dataset. We processed this dataset using SemRep and created a graph from it using the procedure described in Section IV-D. We split the dataset into two subsets: research articles published up until 2005 (12,863 articles) and a version containing all articles published up until now. The size of both graphs appears in Table VI. Next, we study how many direct new connections between pairs of concepts have formed since 2005, and measure the characteristics of the shortest paths between these pairs of concepts in the 2005 version of the graph. Specifically, we investigate whether shorter paths indicate direct connections forming in the future. For this analysis, we consider only those concepts that existed in the 2005 version of the dataset. (From Table VI, it can be seen that more than 28 thousand new concepts were added since 2005.)

Between 2005 and 2020, there appeared 42,077 new direct edges between all 16,427 concepts that existed in 2005. Figure 7 presents an example path that was not present in our dataset as of 2005 and was established later in a paper published in 2007. We would like to note that this example is based on the CORD-19 dataset; it is possible that the A-C connection presented in Figure 7 represents common knowledge which hasn't been recorded in writing, or that other literature not included in our dataset exists that established this connection earlier. The figure serves to demonstrate what an example path from DSNAPSHOT looks like.

To understand whether shorter paths are indicative of future connections, we extract the following two sets of shortest paths from the 2005 graph: 1) shortest paths between pairs of concepts that do not have an edge between them in the 2005 version of the graph but do have a direct edge in the 2020 version of the graph (there are 42,077 such paths); and

[6] https://doi.org/10.13139/OLCF/1646608

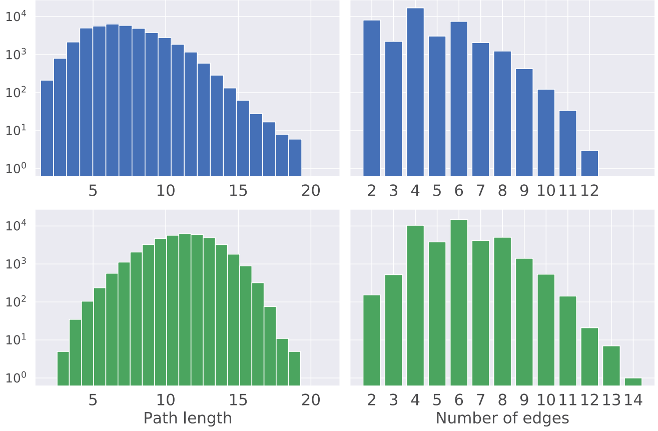


Fig. 8. Path length distribution for node pairs which have formed a direct connection between 2005 and 2020 (top) and pairs of nodes which have not (bottom).

2) shortest paths between random pairs of concepts that do not have a direct connection between them in either version of the graph (we select 42,077 such paths to match the size of the first set). Figure 8 shows the distribution of path length and number of edges in both sets. It can be seen the first set is slightly skewed towards shorter paths, while the second set is skewed in the opposite direction. Furthermore, very short paths do not appear in the second set. To confirm that the two distributions are different, we apply the Kolmogorov-Smirnov test for two samples, which can be used to test whether two probability distributions differ. The K-S statistic obtained from the test is 0.6323 and the p-value is 0.0; therefore, we reject the null hypothesis that the samples are drawn from the same distribution.

These results help to validate the approach used in previous work [21], [26], [27] and indicate that shortest path information may aid in uncovering novel relations between concepts. In contrast to previous work, our hope is that DSNAPSHOT will enable the discovery of meaningful relations across the whole of biomedical knowledge. As future work, we plan to further investigate the properties of shortest paths in this graph. For example, we are interested in studying whether the types of nodes in the shortest paths (in addition to path length) are indicative of novel relations. For instance, in pharmaceutical research, chemical similarity of drugs can be indicative of similar therapeutic properties, and we want to investigate whether such similarity would be captured in the shortest path. We are also interested in studying whether path information can be used as features for training machine learning models for predicting the most promising future relations.

In Section VI-B6, we validated the model discussed of Section V-C against the weak scaling experiment. Based on the cost model using T_{worst} and the experimental data, we estimate that it will take at most 18 hours to compute the APSP for the entire 18.5 million Biomedical Knowledge Graph.

VII. Implications

DSNAPSHOT is highly performant and scalable, achieving an absolute performance of 136 PF/s, setting a new high-watermark for FW-based APSP at the full-scale of Summit. Looking forward, numerous additional challenges remain, including I/O: the raw shortest paths output might consume upwards of 324 TB for the biomedical knowledge graph needed to mine for candidate molecules in the COVID-19 dataset. On the algorithmic front, for truly sparse problems with larger separators, we envision a distributed, communication-avoiding supernodal realization of DSNAPSHOT based on prior results for sparse Gaussian elimination [46]. We provide the results obtained on the COVID-19 data for broader use in the scientific community [16].

VIII. Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Robinson Pino, program manager, under contract number DE-AC05-00OR22725, as well as by the National Science Foundation under Grant No. 1710371. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This material is based upon work supported by the U.S. National Science Foundation (NSF) Award Numbers 1533768 and 1710371. We would like to thank Dr. Oded Green for his help in analysing the performance of our GPU kernels.

References

[1] E. Landhuis, "Scientific literature: information overload," *Nature*, vol. 535, no. 7612, pp. 457–458, 2016.

[2] L. Bornmann and R. Mutz, "Growth rates of modern science: A bibliometric analysis based on the number of publications and cited references," *Journal of the Association for Information Science and Technology*, vol. 66, no. 11, pp. 2215–2222, 2015.

[3] Office of Science and Technology Policy, "Call to action to the tech community on new machine readable COVID-19 dataset," Online, 2020, accessed: 2020-04-18.

[4] H.-T. Yang, J.-H. Ju, Y.-T. Wong, I. Shmulevich, and J.-H. Chiang, "Literature-based discovery of new candidates for drug repurposing," *Briefings in bioinformatics*, vol. 18, no. 3, pp. 488–497, 2017.

[5] M. Thilakaratne, K. Falkner, and T. Atapattu, "A systematic review on literature-based discovery: General overview, methodology, & statistical analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–34, 2019.

[6] B. D. T. Consortium *et al.*, "Toward a universal biomedical data translator," *Clinical and translational science*, vol. 12, no. 2, p. 86, 2019.

[7] D. R. Swanson and N. R. Smalheiser, "An interactive system for finding complementary literatures: a stimulus to scientific discovery," *Artificial Intelligence*, vol. 91, no. 2, pp. 183–203, apr 1997.

[8] D. R. Swanson, N. R. Smalheiser, and V. I. Torvik, "Ranking indirect connections in literature-based discovery: The role of medical subject headings," *Journal of the American Society for Information Science and Technology*, vol. 57, no. 11, pp. 1427–1439, 2006.

[9] V. Tshitoyan, J. Dagdelen, L. Weston, A. Dunn, Z. Rong, O. Kononova, K. A. Persson, G. Ceder, and A. Jain, "Unsupervised word embeddings capture latent knowledge from materials science literature," *Nature*, vol. 571, no. 7763, pp. 95–98, Jul. 2019. [Online]. Available: https://www.nature.com/articles/s41586-019-1335-8

[10] J. Stegmann and G. Grohmann, "Hypothesis generation guided by co-word clustering," *Scientometrics*, vol. 56, no. 1, pp. 111–135, 2003.

[11] D. R. Swanson, "Fish oil, raynaud's syndrome, and undiscovered public knowledge," *Perspectives in biology and medicine*, vol. 30, no. 1, pp. 7–18, 1986.

[12] T. C. Rindflesch, H. Kilicoglu, M. Fiszman, G. Rosemblat, and D. Shin, "Semantic medline: An advanced information management application for biomedicine," *Information Services & Use*, vol. 31, no. 1-2, pp. 15–21, 2011.

[13] L. L. Wang, K. Lo, Y. Chandrasekhar, R. Reas, J. Yang, D. Eide, K. Funk, R. Kinney, Z. Liu, W. Merrill *et al.*, "Cord-19: The covid-19 open research dataset," *arXiv preprint arXiv:2004.10706*, 2020.

[14] A. Buluç, J. R. Gilbert, and C. Budak, "Solving path problems on the gpu," *Parallel Computing*, vol. 36, no. 5-6, pp. 241–253, 2010.

[15] E. Solomonik, A. Buluç, and J. Demmel, "Minimizing communication in all-pairs shortest paths," in *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Boston, MA, USA, 5 2013.

[16] D. Herrmannova, R. Kannan, P. K. Sao, H. Lu, R. M. Patton, T. E. Potok, V. Thakkar, and R. W. Vuduc, "Scalable knowledge-graph analytics at 136 petaflop/s," Aug. 2020, DOI: 10.13139/OLCF/1646608. [Online]. Available: https://zenodo.org/record/3980252

[17] A. P. Chiang and A. J. Butte, "Systematic evaluation of drug–disease relationships to identify leads for novel drug uses," *Clinical Pharmacology & Therapeutics*, vol. 86, no. 5, pp. 507–510, 2009.

[18] A. Gottlieb, G. Y. Stein, E. Ruppin, and R. Sharan, "Predict: a method for inferring novel drug indications with application to personalized medicine," *Molecular systems biology*, vol. 7, no. 1, 2011.

[19] D. L. Ngo, N. Yamamoto, V. A. Tran, N. G. Nguyen, D. Phan, F. R. Lumbanraja, M. Kubo, and K. Satou, "Application of word embedding to drug repositioning," *Journal of Biomedical Science and Engineering*, vol. 9, no. 1, pp. 7–16, 2016.

[20] J. Wawrzinek, S. Hussaini, O. Wiehr, J. González Pinto, and W.-T. Balke, "Explainable word-embeddings for medical digital libraries – a context-aware approach," 08 2020.

[21] J. Sybrandt, M. Shtutman, and I. Safro, "Moliere: Automatic biomedical hypothesis generation system," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1633–1642.

[22] A. Kastrin, T. C. Rindflesch, and D. Hristovski, "Link prediction on the semantic medline network," in *International Conference on Discovery Science*. Springer, 2014, pp. 135–143.

[23] D. Hristovski, J. Stare, B. Peterlin, and S. Dzeroski, "Supporting discovery in medicine by association rule mining in medline and umls," *Studies in health technology and informatics*, no. 2, pp. 1344–1348, 2001.

[24] M. J. Cairelli, C. M. Miller, M. Fiszman, T. E. Workman, and T. C. Rindflesch, "Semantic medline for discovery browsing: using semantic predications and the literature-based discovery paradigm to elucidate a mechanism for the obesity paradox," in *AMIA Annual Symposium Proceedings*, vol. 2013. American Medical Informatics Association, 2013, p. 164.

[25] B. Wilkowski, M. Fiszman, C. M. Miller, D. Hristovski, S. Arabandi, G. Rosemblat, and T. C. Rindflesch, "Graph-based methods for discovery browsing with semantic predications," in *AMIA annual symposium proceedings*, vol. 2011. American Medical Informatics Association, 2011, p. 1514.

[26] Y. H. Kim, S. H. Beak, A. Charidimou, and M. Song, "Discovering new genes in the pathways of common sporadic neurodegenerative diseases: a bioinformatics approach," *Journal of Alzheimer's Disease*, vol. 51, no. 1, pp. 293–312, 2016.

[27] S. H. Baek, D. Lee, M. Kim, J. H. Lee, and M. Song, "Enriching plausible new hypothesis generation in pubmed," *PloS one*, vol. 12, no. 7, 2017.

[28] B. A. Carré, "An algebra for network routing problems," *IMA Journal of Applied Mathematics*, vol. 7, no. 3, pp. 273–294, 1971.

[29] R. C. Backhouse and B. A. Carré, "Regular algebra applied to path-finding problems," *IMA Journal of Applied Mathematics*, vol. 15, no. 2, pp. 161–186, 1975.

[30] M. Gondran and M. Minoux, *Graphs, dioids and semirings: new models and algorithms*. Springer Science & Business Media, 2008, vol. 41.

[31] G. L. Litvinov, "Idempotent/tropical analysis, the hamilton–jacobi and bellman equations," in *Hamilton-Jacobi equations: approximations, numerical analysis and applications*. Springer, 2013, pp. 251–301.

[32] P. Sao, R. Kannan, P. Gera, and R. W. Vuduc, "A supernodal all-pairs shortest path algorithm," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'20)*. ACM, 2020, pp. 250–261.

[33] R. R. Williams, "Faster all-pairs shortest paths via circuit complexity," *SIAM Journal on Computing*, vol. 47, no. 5, pp. 1965–1985, 2018.

[34] T. M. Chan, "All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time," in *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2006, pp. 514–523.

[35] A. Tiskin, "All-pairs shortest paths computation in the bsp model," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2001, pp. 178–189.

[36] H. Djidjev, S. Thulasidasan, G. Chapuis, R. Andonov, and D. Lavenier, "Efficient multi-GPU computation of all-pairs shortest paths," in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, ser. IPDPS '14. USA: IEEE Computer Society, 2014, p. 360–369. [Online]. Available: https://doi.org/10.1109/IPDPS.2014.46

[37] J. T. Fineman and E. Robinson, "Fundamental graph algorithms," in *Graph Algorithms in the Language of Linear Algebra*, J. Kepner and J. Gilbert, Eds. Philadelphia, PA, USA: Society of Industrial and Applied Mathematics, 2011, ch. 5, pp. 45–58.

[38] J. Kepner, P. Aaltonen, D. Bader, A. Buluç, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke *et al.*, "Mathematical foundations of the GraphBLAS," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2016, pp. 1–9.

[39] T. A. Davis, "Graph algorithms via SuiteSparse: GraphBLAS: triangle counting and k-truss," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–6.

[40] A. Kerr, "Cutlass: Cuda templates for linear algebra subroutines," Nov. 2019. [Online]. Available: https://github.com/NVIDIA/cutlass

[41] M. Khabsa and C. L. Giles, "The number of scholarly documents on the public web," *PloS one*, vol. 9, no. 5, 2014.

[42] J. Siek, A. Lumsdaine, and L.-Q. Lee, *The boost graph library: user guide and reference manual*. Addison-Wesley, 2002.

[43] U. Meyer and P. Sanders, "δ-stepping: A parallel single source shortest path algorithm," in *European symposium on algorithms*. Springer, 1998, pp. 393–404.

[44] K. Pingali, "High-speed graph analytics with the galois system," in *Proceedings of the first workshop on Parallel programming for analytics applications*. ACM, 2014, pp. 41–42.

[45] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[46] P. Sao, X. S. Li, and R. Vuduc, "A communication-avoiding 3D LU factorization algorithm for sparse matrices," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Vancouver, BC, Canada, May 2018.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We ran scaling experiments on Summit using gcc/6.4.0, cuda/10.1, spectrum MPI, cmake. The code is behind ORNL's firewall. But for evaluation purpose, we have shared the code over a dropbox link. The source code zip file has a README.pdf that shows step-by-step process for recreating the experiments.

ARTIFACT AVAILABILITY

Software Artifact Availability: Some author-created software artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: All author-created data artifacts are maintained in a public repository under an OSI-approved license.

Proprietary Artifacts: None of the associated artifacts, author-created or otherwise, are proprietary.

Author-Created or Modified Artifacts:

```
Persistent ID: https://code.ornl.gov/gordon-bell-aps⌟
↪  p/distfw2d.git
Artifact name: DSNAPSHOT

Persistent ID: https://www.dropbox.com/s/dc0ytm9ytw4⌟
↪  08kl/snapshot-adae.zip?dl=0
Artifact name: SNAPSHOT Code
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Summit, Nvidia Voltas, PowerPC

Operating systems and versions: Redhat Linux

Compilers and versions: gcc/6.4.0

Applications and versions: None

Libraries and versions: spectrum MPI, cuda/10.1

Key algorithms: Floyd Warshall

Input datasets and versions: Suite sparse matrix collection

URL to output from scripts that gathers execution environment information.

```
https://www.dropbox.com/s/sd79sdmtzig7z2c/summitenv.⌟
↪  txt?dl=0
```