Exaflop/s Biomedical Knowledge Graph Analytics

Ramakrishnan Kannan*, Piyush Sao*, Hao Lu*, Jakub Kurzak$, Gundolf Schenk#, Yongmei Shi#, Seung-Hwan Lim*, Sharat Israni#, Vijay Thakkar+, Guojing Cong*, Robert Patton*, Sergio Baranzini#, Richard Vuduc+, Thomas Potok*
* Oak Ridge National Laboratory, Oak Ridge, TN, USA
# University of California San Francisco, CA, USA
+ Georgia Institute of Technology, Atlanta, GA, USA
$ Advanced Micro Devices, Inc., USA

***Abstract*—We are motivated by newly proposed methods for mining large-scale corpora of scholarly publications (e.g., full biomedical literature), which consists of tens of millions of papers spanning decades of research. In this setting, analysts seek to discover relationships among concepts. They construct graph representations from annotated text databases and then formulate the relationship-mining problem as an all-pairs shortest paths (APSP) and validate connective paths against curated biomedical knowledge graphs (e.g., SPOKE). In this context, we present COAST (Exascale Communication-Optimized All-Pairs Shortest Path) and demonstrate 1.004 EF/s on 9,200 Frontier nodes (73,600 GCDs). We develop hyperbolic performance models (HYPERMOD), which guide optimizations and parametric tuning. The proposed COAST algorithm achieved the memory constant parallel efficiency of 99% in the single-precision tropical semiring. Looking forward, COAST will enable the integration of scholarly corpora like PubMed into the SPOKE biomedical knowledge graph.**

***Index Terms*—Shortest Path Problem, High-Performance Computing, Parallel Algorithms**

I. GORDON BELL JUSTIFICATION

We computed all-pairs shortest path (APSP) on a graph with 7.06 million vertices using 9,200 Frontier nodes (73,600 GCDs) in 11.7 minutes at 1.004 exaflop/s (single-precision, 99% memory constant parallel efficiency and 75% machine-peak for tropical semi-ring GEMM). This computation would be the first exaflop-level demonstration of a graph algorithm and the first scientific study on the integration of SPOKE (Scalable Precision Medicine Open Knowledge Engine) with publication information by using paths.

II. OVERVIEW OF THE PROBLEM

The scientific context of this work is SPOKE [1],[1] an ongoing effort of biomedical researchers to organize and represent the knowledge embedded across a wide variety of disparate biomedical datasets as a unified graph representation, or *knowledge graph*. The many millions of graph nodes represent biological and biomedical concepts and entities (e.g., genes, diseases, pathways, proteins, symptoms). The edges represent known relationships among these nodes, such as "cell expresses gene," "compound causes side effect," or "nirmatrelvir/ritonavir treats COVID-19." The known relationships among different biomedical fields (e.g., genomics, physiology, cellular behavior, clinical observations) and even environmental factors (e.g., "lead compounds") are highly complex, as illustrated in Figure 1. In response, graph databases have recently gained popularity as a practical solution to integrating such disparate sources of information. The SPOKE graph database is meant to facilitate the discovery of new knowledge by enabling users to explore the structure of this graph and run a variety of analytical queries against it.

The remarkable variety of downstream tasks envisioned for SPOKE include a pharmaceutical lab's discovery of candidate drug compounds for a given disease, an environmental agency's identification of a toxin that may be associated with a condition, or a doctor's querying for potential diagnoses based on observations of their patient. SPOKE has also been applied to the study of COVID-19. With the sequencing of the SARS-CoV-2 virus, its effects on the human molecular framework and systems were evident [2], [3]. The chain of events that cause dangerous cytokine and bradykinin storms in patients, the effects on human respiratory pathways, and the potential treatments (e.g., dexamethasone, corticosteroids) were all laid out in the SPOKE biomedical knowledge network [4].

In this paper, we consider the comparison of knowledge encoded within SPOKE, which is largely human curated, against concept relationships that might be mined automatically from a scholarly database (i.e., the papers in Pubmed/MEDLINE).

TABLE I: Gordon Bell Performance Attributes.

Attributes	Category	Details
Category	Peak performance	1.004 exaflop/s
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	98% of Frontier (9,200 nodes, 73,600 GCDs)
Measurement mechanism	Timers	Direct timer instrumentation

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the US 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).

[1]https://spoke.ucsf.edu/data-tools

978-1-6654-5444-5/22/$31.00

Such an analysis would be a major step toward comparing and integrating these two forms of biomedical knowledge. The paradigm for analysis is based on examining paths in the graph and the distances between concepts [5]–[7]. The bottleneck of this analysis is the APSP computation.

Because the graphs of interest have many millions of vertices, the comparison requires leadership-class computing resources and appropriate scaling of our methods. Our scalable method for APSP is the Exascale Communication-Optimized All-Pairs Shortest Path (COAST) algorithm, which implements a GPU-accelerated, distributed-memory parallel version of the Floyd-Warshall (FW) algorithm. Our experiment targets the Oak Ridge National Laboratory's Frontier system, the first exascale supercomputer. Our approach performs well because FW relies on matrix-multiplication-like (i.e., level-3 BLAS-like) operations, which are well suited to distributed-memory systems [8] and GPUs [9].

We apply this algorithm to a dataset of 18.5 million vertices and 213 million relations that represent biomedical concepts, publications, and their connections extracted from the new COVID-19 Open Research Dataset (CORD-19) [11] and the Pubmed database. We also investigate whether the paths determined in Pubmed by using COAST are meaningful by investigating their presence in the SPOKE knowledge graph.

A. Challenges of Frontier over Summit

Scaling APSP to an exascale machine is considerably more challenging than our previous experiences. We previously demonstrated 136 petaflop/s in single-precision on a pre-exascale machine—Summit [10]. However, lessons from pre-exascale systems do not always translate to similar efficiency on an exascale machine. The difficulty is that various components in the computation can behave in unexpected ways and lead to non-intuitive observations. This puzzle has several facets: (1) most components of the computation have more than one algorithm; (2) each candidate algorithm's performance depends strongly on the input size and scale of the machine; and (3) when combining optimized components, performance does not necessarily compose.

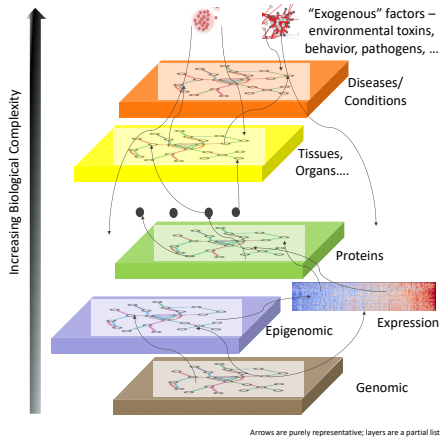


Fig. 1: Hierarchical organization of human biology, genomics, and up. Each layer has interconnections (e.g., a protein may be *related* to another), and the layers are connected via specific vertices (e.g., a specific gene expresses a specific protein).

To better understand this difficulty, suppose there are two variants of the APSP implementation: naïve and optimized. Each component in the optimized variant is asymptotically faster than its naïve counterpart, whether it is the broadcast algorithm, rank placement, or GPU compute kernel. When run on the entire Frontier supercomputer, the naïve algorithm can still outperform the optimized variant for many problem sizes despite using asymptotically slower components (Figure 8).

Additionally, performance variations across nodes can cause significant performance differences from what a simple asymptotic projection might otherwise predict. There are many sources of such performance variability, including complex combinations of random events (e.g., thermal performance throttling, clock speed fluctuations, skew, operating system noise), that are common at scale. Hence, we seek to design algorithms that gracefully tolerate such random events (e.g., minimize their impact on performance) and reduce performance variance across multiple runs of the same algorithm.

Targeting Frontier, we developed multiple optimized candidates for different performance aspects (e.g., computation, communication, rank placement). We then built performance models by using various benchmarking runs of these optimized candidates on the Summit and Frontier systems. During large-scale executions, these models can quickly estimate the performance of different components, and this information then enables better-informed designs that can achieve the desired performance on the target system.

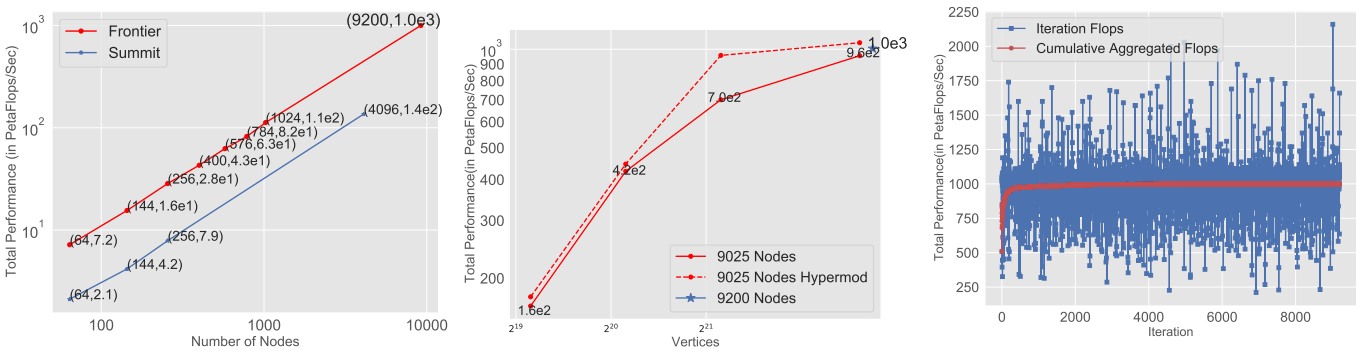
The key enabling ideas in our overall approach are as follows.

- We develop a 2D-distributed FW algorithm that uses GPUs and optimized communication, that scales APSP to inputs with millions of vertices on 9,200 nodes of Frontier.
- For Frontier, we used packed 32-bit floating point (FP32) semi-ring based GEMM (OPTSSRGEMM-NT) on AMD's MI250X GPU along with the ring-2 modified communication strategy to scale to 9,200 nodes.
- Targeting high-performance computing (HPC) architectures with GPUs, we develop a novel performance model based on the hyperbolic performance model (HYPERMOD). We propose a parameter estimation heuristic to determine the model parameters via experiment. HYPERMOD can determine the component-candidates that meet a desired performance target of COAST on Frontier.
- We show that APSP algorithms can uncover novel relations from biomedical knowledge graphs at scale by investigating these paths on the SPOKE knowledge graph.
- On Frontier, we achieve 1.004 EF/s for a graph analysis algorithm based on state-of-the-art practices.

III. CURRENT STATE OF THE ART

A. Summit and Frontier architectures

The Oak Ridge Leadership Computing Facility (OLCF) currently hosts two leadership-class supercomputers: Summit and Frontier (Table III).



(a) (b) (c)

Fig. 2: Path toward exaflops: Figure 2a compares COAST performance on Frontier vs. Summit. Every point is annotated (Nodes, Performance). Figure 2b shows the experimental runs to validate the projection from HYPERMOD while finding the smallest size to achieve 1 exaflop/s. Figure 2c shows the per iteration and the cumulative aggregated flops of the experiment run on 9,200 nodes for 702 seconds and also shows the achieved 1.004 EF/s. Note that the 2 MPI ranks share a graphics complex die (GCD), and therefore the instantaneous flops estimated from MPI Rank 0 can fluctuate between 250 petaflop/s and 2 exaflop/s.

TABLE II: Gordon Bell justification. Refer to Table III for metrics. To demonstrate that we could use a graph 3× larger than the one used for Summit, we ran the entire 2018 PubMed graph for 15 minutes and achieved 1.008 exaflop/s.

Nodes	MPI (P)	Row (P_x)	Col (P_y)	Vertices (n)	Memory (TB)	Time (s)	Exaflops (single precision)	Fraction of peak
Summit's fastest run [10]								
4,096	24,576	192	256	4.43×10^6	157	1.28×10^3	1.36×10^{-1}	70%
Frontier's fastest run (Section VII-A)								
9,200	147,200	368	400	7.06×10^6	399	702	1.004	75%
Frontier's largest run (7% completion)								
9,025	144,400	380	380	18.6×10^6	2,768	913	1.008	75%

	Summit	**Frontier**
Nodes	4,608	9,408
Processor	POWER9	3rd-gen EPYC
GPU	NVIDIA V100	AMD MI250X
GCDs per node	6	8
Memory per GPU/GCD	16 GB	64 GB
FP16/FP64 (GCD)	125/7.8 TF/s	298/54.5 TF/s
GPU interconnect	NVLINK	Infinity Fabric
NICs	2× Mellanox EDR IB	4× Slingshot-11
NIC bandwidth per node	12.5 + 12.5 GB/s	25 + 25 GB/s
SSRGEMM Performance metrics per node		
Theoretical peak	47.1 TF/s	144 TF/s

TABLE III: Key architectural specifications for Summit and Frontier using sustained frequency for Frontier (SSRGEMM peak, not boost frequency).

The proposed COAST algorithm leverages certain Frontier-specific features that impact the choice of parameters, optimization, and ultimately the performance of the code at scale. When comparing the two systems, we consider a single NVIDIA Volta GPU on Summit against a single graphics complex die (GCD) of an AMD MI250X on Frontier. The GPU memory dictates the maximum size of the problem that can be solved. The GCD count per node impacts the number of MPI ranks per node. Frontier's AMD GPUs also have 4× the memory per GCD over Summit and can thus handle larger problems.

Most importantly, GCD performance correlates with the performance of COAST. Frontier has 3× more performance per node in single-precision SRGEMM (SSRGEMM) and more than 2× as many nodes as Summit. Frontier is expected to see more than a 7× COAST performance improvement over Summit at a full scale, which closely matches the 8× computational power increase of Frontier over Summit in 64-bit floating point (FP64) arithmetic.

Considering communication costs, the time required to the exchange the matrix correlates with the number of network interface controllers (NICs) and their performance. The Frontier NIC is directly connected to the GPU, and this direct connection enhances the performance of GPU-aware MPI because the memory contents can be transferred between the GPUs without offloading through other components, thus reducing latency.

B. SPOKE

SPOKE is an evolving biomedical knowledge network that integrates over 40 data sources into a graph with more than 50 million vertices (of 20 types) and more than 100 million edges (of 55 types) [1]. This data includes genomic associations with disease, chemical compounds and their binding targets, and

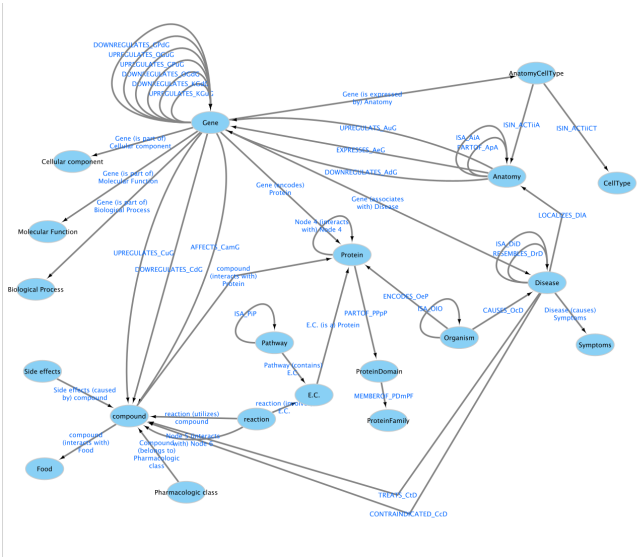


Fig. 3: SPOKE metagraph: Vertices denote biological concepts, and edges show relationships among the concepts.

metabolic reactions from select bacterial organisms of relevance to human health (Figure 3).

SPOKE draws from seven standard biomedical ontologies in the National Center for Biomedical Ontology's BioPortal repository. Several of the key concepts are mapped to biomedical ontologies to provide an organizational framework and facilitate user navigation. SPOKE also uses ontologies to mark up the datasets coming into the knowledge graph for consistent linking and contributes to and aligns with the Biolink biomedical semantic standard.[2]

C. Pubmed and CORD-19 Dataset as a Graph

As mentioned in Section II, our Pubmed graph was constructed using the Semantic MEDLINE database [12]. Here, we briefly summarize the construction of the graph from the Pubmed and CORD-19 datasets. The graph is composed of two types of vertices: *concepts* and *abstracts*. The vertices can be connected in three different ways: (1) concept-to-concept relations (co-occurrences), (2) concept-to-abstract relations (occurrences), and (3) abstract-to-abstract relations (citations). Additional details appear in Appendix XI-D and with the dataset itself [13].

IV. INNOVATIONS REALIZED

APSP is the simultaneous computation of the shortest paths between all pairs of vertices in a graph. During the computation of APSP, FW algorithms maintain and update a 2D array of distances, Dist. FW uses a dynamic programming approach to computing APSP by initializing Dist with the input weights, W. Then, in the k-th iteration, it checks for all pairs of vertices, v_i and v_j, to determine if there is a shorter path between them via the intermediate vertex, v_k. If so, FW updates $\text{Dist}[i,j]$. Therefore, $\text{Dist}[i,j]$ after k steps, which we denote by $\text{Dist}^k[i,j]$, may be defined recursively as

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

[2]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6440568/

Algorithm 1 COAST.

```
def Coast (A, P = P_r × P_c):
# A is distributed in block cyclic fashion
# my process Id is p_id
# On each MPI process p_id do in parallel:
for k in {1, 2..., n_b}:
  if p_id= p_{k,k}:
    A(k,k) ← FW(A(k,k)) #Diagonal Update
    GPUAwareRingBCast(A(k,k), P_r(k))
    GPUAwareRingBCast(A(k,k), P_c(k))
  if p_id ∈ P_r(k):
    Receive(A(k,k), p_{k,k})
    A(k,:) ← A(k,:) ⊕ A(k,k)⊗A(k,:) #Panel Update
    GPUAwareRing2MBCast(A(k,:), P_c(p_id))
  else :
    Receive(A(k,:))
  if p_id ∈ P_r(c):
    Receive(A(k,k), p_{k,k})
    A(:,k) ← A(:,k) ⊕ A(:,k)⊗A(k,k) #Panel Update
    GPUAwareRing2MBCast(A(:,k), P_r(p_id))
  else :
    Receive(A(k,:))
  # Min-plus outer product
  if p_id owns A(i,j):
    # A(i,j) ← A(i,j) ⊕ A(i,k)⊗A(k,j)
    A(i,j) ← OptsSrgemm-Nt (A(i,k), A(k,j)^T, A(i,j))
```

This can be computed in place by overwriting $\text{Dist}^k[i,j]$ in place of $\text{Dist}^{k-1}[i,j]$.

APSP may be understood algebraically as computing the matrix closure of the weight matrix, W, defined over the tropical semi-ring [14]. 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

$$C_{ij} \leftarrow \min_k \left(A_{ik} + B_{kj}\right).$$

This product is an analogue of matrix-matrix multiplication over the reals.

This computation may also be blocked. Divide Dist into $n_b \times n_b$ blocks, each of size $b \times b$ (i.e., $n_b = \frac{n}{b}$). Let A_{ij} denote the (i,j) block of A, where $1 \le i,j \le n_b$. A blocked version of FW, called BLOCKEDFW, can carry out the same APSP computation as FW in the following three steps.

(1) **Diagonal Update (DIAGUPDATE):** Perform the classic FW algorithm on a diagonal block, A_{kk}.

(2) **Panel Update (PANELUPDATE):** Update the k-th block row and column. For any block, $A(k,j),\ j{\neq}k$ in the block row, the update is a MIN-PLUS multiply with A_{kk} from the left. For block $A(i,k)$ on the k-th block column, it is MIN-PLUS multiply with A_{kk} from right:

$$A(k,j) \leftarrow A(k,j) \oplus\ A(k,k)\otimes A(k,j) \qquad j \neq k$$
$$A(i,k) \leftarrow A(i,k) \oplus\ A(i,k)\otimes A(k,k) \qquad i \neq k$$

Here, $a \oplus b$ denotes $\min\{a,b\}$, and $a \otimes b$ denotes $a+b$.

(3) **MinPlus Outer Product**: Perform the outer product of k-th block row and block column and update all the remaining blocks of matrix A:

$$A(i,j) \leftarrow A(i,j) \oplus\ A(i,k)\otimes A(k,j) \quad i,j \neq k.$$

This step is analogous to a Schur-complement update in an LU or Cholesky factorization.

A. Parallel FW Algorithm on a 2D Process Grid

We call the BLOCKEDFW's parallel distributed memory realization by using MPI as PARALLELFW. The MPI processes

are logically arranged in a 2D process grid of dimension $p_x \times p_y$. On this 2D process grid, the distance input matrix A is distributed in a block-cyclic fashion such that a block $A_{i,j}$ resides in a process with coordinates $(i \mod P_r,\ j \mod P_c)$.

The outer loop of PARALLELFW proceeds as the BLOCKEDFW. However, we have additional communication steps, namely DIAGBCAST and PANELBCAST, so that all processes can perform the PANELUPDATE and OUTERUPDATE.

After process P_{kk} performs DIAGUPDATE on block A_{kk}, the process P_{kk} broadcasts A_{kk} across its process row $P_r(k)$ and its process column $P_c(k)$. In the panel update step, each process in $P_r(k)$ performs left multiplies of $A_{k:}$ with A_{kk}, and each process in $P_c(k)$ performs right multiplies of $A_{:k}$ with A_{kk}. Subsequent to the panel update, each process in $P_r(k)$ broadcasts blocks of $A_{k:}$ to its process column, and each process in the $P_c(k)$ broadcasts blocks of $A_{:k}$ to its process row. Finally, upon receiving $A_{k:}$ and $A_{:k}$, each process performs OUTERUPDATE on its local copy.

B. Communication Optimizations

The PARALLELFW algorithm performs a high volume of communication in the $O\left(\frac{n^2}{P_x} + \frac{n^2}{P_y}\right)$ (see Section IV-D). As such, more time is spent in communication than computation when scaling to many nodes. To alleviate potential communication issues, we consider several techniques:

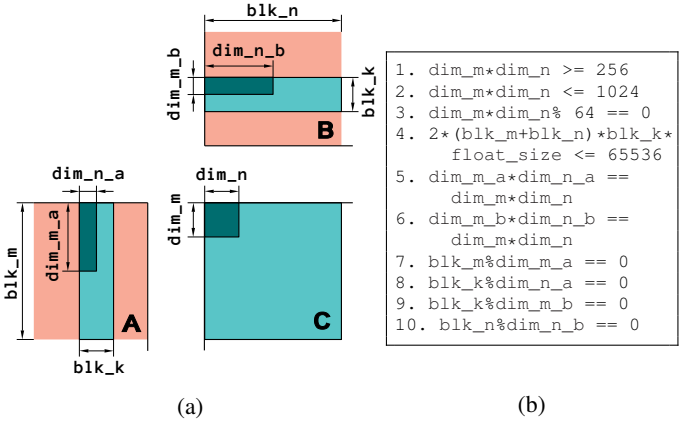
- **Look-ahead:** The look-ahead technique overlaps the two operations. We leverage the idea of breaking the dependency between k and $k+1$ iterations to achieve the overlap.
- **Ring broadcast:** The traditional library provided in MPI_BCast may not be most efficient for our application because it uses a *kd-tree* pattern (also called hyper-cube algorithm). Such an algorithm balances latency and bandwidth costs. To this end, we exercise *Ring* and *Ring-2 modified* at full scale on Frontier.
- **GPU-aware MPI:** Unlike Summit, Frontier's NICs are directly connected to GPUs. Hence, communicating the GPU buffers with GPU-aware MPI directly from GPU to GPU will be faster than communicating through the CPU.

On Frontier, every node has four separate NICs connected directly to the Slingshot network. For COAST to utilize all four NICs concurrently, we use the rank placements provided via the Cray MPICH library. Appendix XI-B details these optimizations.

C. Acceleration on AMD MI250X GPUs

Frontier boasts AMD's MI250X accelerator, which is based on the CDNA architecture and contains two GCDs with 110 compute units (CUs) each [15]. Each CU contains four single instruction, multiple data engines, which are 16-lanes wide, for a total of 64 operations per cycle. The MI250X supports packed FP32 instructions, which operate on pairs of FP32 operands, and fused multiply add (FMA) instructions, which perform multiplication and addition in one step and effectively bring the throughput to 256 floating-point operations per cycle.

Unfortunately, we cannot use the FMA instruction for the *min-plus* $C_{ij} \leftarrow min(A_{ik} + B_{kj}, C_{ij})$ operation required by the algorithm. Instead, we use the packed FP32 `add` instruction and the FP32 `min3` instruction. The packed FP32 `add` sums two pairs of operands, and FP32 `min3` finds the minimum of three operands simultaneously. This combination takes two input values of A and two input values of B and produces one output value of C using one packed FP32 `add` and the FP32 `min3`. This design yields 128 operations per cycle.



(a) (b)

Fig. 4: (a) The min-plus kernel blocking factors and (b) constraints used to trim the tuning space.

	bm	bn	bk	dm	dn	dma	dna	dmb	dnb
NN	256	64	4	32	8	64	4	4	64
NT	64	256	4	8	32	64	4	64	4
TN	128	192	4	32	8	4	64	4	64
TT	64	256	4	8	32	4	64	64	4

TABLE IV: Tuned blocking factors for OPTSSRGEMM-NT; **b** represents blk, e.g., **bk** stands for **blk_k**; **d** represents dim, e.g., **dmb** stands for **dim_m_b**.

The MAGMA (Matrix Algebra on GPU and Multicore Architectures) [16] kernel was the starting point for a fast implementation of the min-plus kernel, OPTSSRGEMM-NT. We found that the MAGMA SSRGEMM kernel delivers 5.6 TF/s on one GCD of the MI250X GPU, which is much less than we expected based on a theoretical peak of 18 TF/s. After applying vectorization, pipelining, and autotuning to produce a faster implementation, we achieved performance levels of up to 15.3 TF/s for the NT version. Here, 'N' stands for *no transpose*, and 'T' stands for *transpose*. For example, the SSRGEMM NT version is performed as $A \times B^T$. The other versions perform slightly worse owing to less favorable memory access patterns: 14.8 TF/s for NN, 12.2 TF/s for TN, and 15.1 TF/s for TT. This is of little concern because the code is structured to leverage the NT kernel. The details of these optimizations are presented in Appendix XI-C.

D. COAST Analysis

1) Computation Cost

In the blocked FW algorithm, the total number of floating-point operations is $2n^3$ distributed among P processes. Let γ be the cost of one floating-point operation. The time to perform

these floating-point operations on P processes is

$$T_{\text{comp}} = \frac{2n^3}{P}\gamma. \tag{1}$$

2) Communication Cost

If b is the block size used for block-cyclic data distribution, then Algorithm 1 performs the $\frac{n}{b}$ outer loop iterations. In each of the iterations, each process participates in two broadcasts: $\frac{nb}{P_r}$ across the process row and $\frac{nb}{P_c}$ across the process column. In the ring broadcast, the total cost of the two broadcasts is $2\alpha + \beta\left(\frac{nb}{P_r} + \frac{nb}{P_c}\right)$, where α is the setup cost of sending a message, and β is the cost of sending a unit-float word. Because the outer iteration runs for $\frac{n}{b}$ iterations, the total communication cost is $2\frac{n}{b}\alpha + \beta(\frac{n^2}{P_r} + \frac{n^2}{P_c})$.

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

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

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

In the worst case, communication and computation will not overlap at all, in which case the total cost is given by

$$T_{\text{worst}} = T_{\text{comp}} + T_{\text{comm}} = \frac{2n^3}{P}\gamma + 2\frac{n}{b}\alpha + \beta\left(\frac{n^2}{P_r} + \frac{n^2}{P_c}\right). \tag{4}$$

These models act as guidelines to validate the scaling experiments. Most of the experiments were cross-validated against this model, and we present the application of the model on the weak scaling experiments in Section V.

E. Hyperbolic Performance Model (HyPerMod)

The preceding asymptotic model is only accurate for large values of problem parameters. Such models can break down because increasing the number of processes can make other parameters (e.g., problem size per process) small, which results in less accurate predictions.

Instead, we use *hyperbolic* performance models, which are functions, $y = f(x)$, the shapes of which are hyperbolas in the xy-plane. If η is a performance parameter, then hyperbolic models can be understood as a combination of performance models that are asymptotic to η and $\frac{1}{\eta}$. A general hyperbolic model, $y = f(\eta)$, is given by

$$y := f(\eta) = \frac{a\eta + b}{c\eta + d} \qquad a,\ c,\ ad - bc \neq 0$$

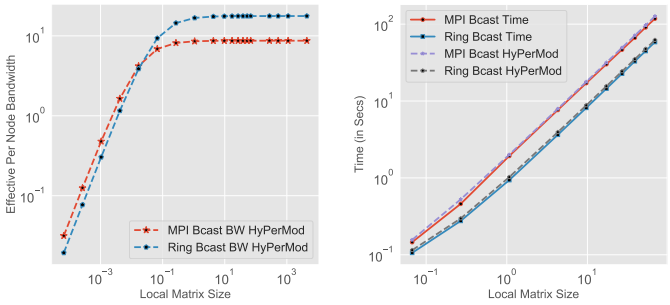
The function $f(\eta)$ has two asymptotes: $y = \frac{a}{c}$ as $\eta \to \infty$ and $y = \frac{b}{d}$ as $\eta \to 0$. In this paper, we assume $b = 0$ so that

$$y := f(\eta) = \frac{a\eta}{c\eta + d} \qquad a,\ c,\ ad \neq 0. \tag{5}$$

Without loss of generality, we can further assume $c = 1$; thus

$$y := f(\eta) = \frac{a\eta}{\eta + d} \qquad a,\ ad \neq 0. \tag{6}$$

This model captures asymptotes on both sides of the parameters instead of just a single side as in the traditional model.



(a) (b)

Fig. 5: Performance models for communication. Figure 5a shows effective per-node bandwidth estimated by our models. We show the actual and predicted time for MPI BCast and ring-based BCast in Figure 5b.

F. Parameter Estimation of HyPerMod

We can estimate a and d of HyPerMod using a least-squares fit. Assume $g(\eta_i)$ is the observed performance value that we are fitting against our model, $f(\eta_i)$, where function f is defined in Equation (6). Then, the least-squares objective is

$$\arg\min_{a,d} \sum_i \left(g(\eta_i) - f(\eta_i)\right)^2.$$

The above function is a biased estimator because least squares favor minimizing the error of larger η_i. The resulting model will behave similarly to a traditional asymptotic model. To overcome this problem, we reformulate the objective for HyPerMod to be

$$\arg\min_{a,d} \sum \frac{1}{\eta_i} \cdot \left(\log \frac{f(\eta_i)}{g(\eta_i)}\right)^2. \tag{7}$$

This optimization problem is challenging. We use an approximation heuristic for parameter estimation.

$$a \approx \max_i g(\eta_i) \tag{8}$$

$$d \approx \arg\min_{\eta_i} \left| g(\eta_i) - \frac{a}{2} \right|. \tag{9}$$

In many cases, Equation (9) provides a good solution—especially when the observations are not noisy. Otherwise, we sweep the values of d within a tiny approximate neighborhood to choose the best fit for $g(\eta_i)$.

The sustainable floating-point operations per second rates for sSrgemm with different parameters are recorded in Figures 6a and 6b. The effective per-node bandwidth and communication time for different broadcasts are recorded in Figures 5a and 5b. They show the proximity of our hyperbolic performance model against the real world experiments. The parameters for our HyPerMod are recorded in Table V.

V. How Performance Was Measured in Experiments

First, we look at Coast's single-node efficiency compared to other Apsp baselines (Section XI-D1) on a variety of real-world graphs that fit within a node (Section XI-D2). Appendix XI-D provides complete details of the real-world experiments. These experiments help establish whether node-local code is a good building block for the distributed implementation. We conduct weak-scaling experiments in which we fix the per-node problem size of $O(n^3/p)$. Weak scaling matches aspects of a growing biomedical literature and should be an easy case for Coast.

Performance models		a	d
Input	**Output**		
K	sSRGEMM performance	15.56 TF/s	K = 70
Local matrix size	sSRGEMM performance	15 TF/s	4.00 MB
Local matrix Size	Effective per-node ring bandwidth	17.7 GB/s	0.06 GB
Local matrix size	Effective per-node MPI bandwidth	8.7 GB/s	0.018 GB

TABLE V: Parameter estimation of HYPERMOD for different computation and communication performance levels.

Subsequently, we present the projection and the experiments conducted on Frontier at full scale. Lastly, we analyze the paths determined in Pubmed against the SPOKE graph in Section VIII.

1) Environment

The software environment on Summit comes from IBM and differs from Frontier, which is an HPE/Cray system. On Summit, the software versions used are GCC 9.1.0, IBM Spectrum MPI 10.4.0.3, and CUDA 11.4.243. Summit's *jsrun* tool is used to launch applications. On Frontier, the software versions used are GCC 11.2.0, Cray-MPICH 8.1.17, ROCM 5.1.0, and MAGMA 2.6.1. No other proprietary software was used.

VI. PERFORMANCE RESULTS AND OBSERVATIONS

A. Block and Input Size Effect on sSRGEMM Performance

Figures 6a and 6b show OPTSSRGEMM-NT runs on one of Frontier's AMD MI250X, MAGMA sSRGEMM runs on a single MI250X GCD, and cuASR runs on one of Summit's NVIDIA Volta GPUs. The plots show that OPTSSRGEMM-NT significantly outperforms MAGMA on MI250X and cuASR on NVIDIA Volta for block sizes greater than 256. OPTSSRGEMM -NT on MI250X is 2.2× faster than cuASR on NVIDIA Volta.

We observed an increase in performance as the block size increased from 1 to 128 due to the compute intensity associated with OPTSSRGEMM-NT. Beginning at block size 128, we achieve peak performance, which then flattens out. Similarly, for input sizes above 8,192, the performance flattens.

Higher block sizes result in fewer global iterations of COAST. Therefore, we obtain the best communication performance at block size 4,096. However, the diagonal panel update time is higher for a larger K, and COAST's best overall performance is achieved for $K = 768$. Hence, we choose a block size of 768 for our experiments on Frontier.

B. Real-World Experiments on a Single Node

We compared the FW algorithm based on sSRGEMM against other single-node implementations using the datasets from Table VII, which were taken from the SuiteSparse Matrix Collection [17]. We either developed or obtained open-source baselines for these experiments.

Both Summit and Frontier have fat nodes with 512 GB of CPU-attached memory per compute node. However, the AMD EPYC CPUs on Frontier have higher floating-point throughput and increased bandwidth. Hence, in most cases, the speedup of MI250X over the CPU on Frontier is not as significant as it is on Summit. The details of the experiments are presented in Appendix XI-D. This experiment established whether node-local code is a good building block for a distributed implementation.

C. PARALLELFW Variations

PARALLELFW is a naïve 2D algorithm without any optimization and uses MAGMA's realization of the sSRGEMM kernel as a baseline. We benchmarked six variants of PARALLELFW by varying the communication-optimization strategies discussed in Section IV-B using the tuned OPTSSRGEMM-NT kernel. The PARALLELFW with look ahead is called LA. The variant LA+R is PARALLELFW +LA with ring communication on the CPU. The GMPI and GMPI+R are GPU-aware MPI realizations of LA and LA+R, respectively. Per Section IV-C, performing sSRGEMM over $A \times B^T$ as the NT kernel is faster than the NN kernel with $A \times B$. GMPI+NT+R is the variant that exercises the NT kernel. Finally, we present COAST, which utilizes all optimizations (e.g., look ahead, ring, and pipelined ring-2 modified broadcasts) directly by using GPU-aware MPI along with the NT kernel.

D. Single-Node Performance of COAST

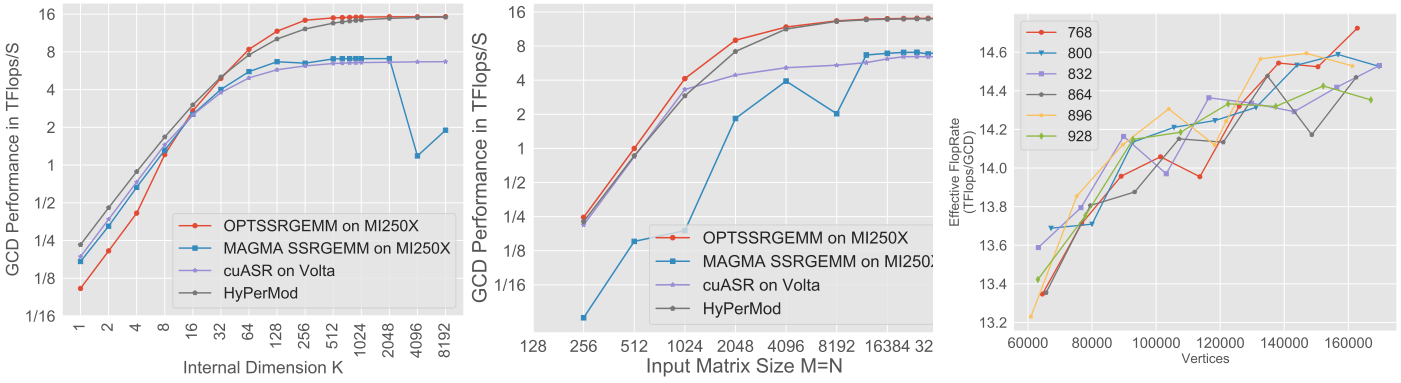
In a previous development, we placed two MPI ranks per GCD to fully saturate the GCD's computing power. We scanned through several batch sizes, B, and vertices, N, to build the baseline performance of our single node result (Figure 6c). Our code achieved 14.7 teraflop/s/GCD (81% of theoretical peak) at sizes N = 162,816 and 13.7 teraflop/s per GCD (76% of peak) at N = 76,800.

E. Weak Scaling

For the weak-scaling experiment, we keep the compute-per-MPI rank fixed. That is, we maintain $O(n^3/p)$ floating point operations per rank across the different runs. We present two key observations from Figure 7. (1) The COAST algorithm is 2.53× more efficient over the naïve PARALLELFW on 1,024 nodes. We see the incremental performance improvement for every communication optimization (e.g., look ahead, ring, and GPU-aware MPI). The GPU-aware MPI and ring broadcast using OPTSSRGEMM-NT exhibits the best performance. The pipelined ring broadcast improvement in the fully optimized COAST lowers the performance variance in the case of many nodes. (2) The naïve implementation achieves 62%, and the proposed COAST achieves at least 97% efficiency when weak scaling to 64× as many processes. We remain compute-bound throughout the weak-scaling experiments.

F. Ring vs. Off-the-Shelf MPI Broadcast

Figure 5a shows the effective per-node bandwidth by sweeping the local nodes' matrix size for the Cray-MPICH broadcast against the ring broadcast discussed in Section IV-B. For per-node matrix sizes less than 20 MB, the off-the-shelf MPI broadcast offers higher bandwidth. For larger matrices



(a) (b) (c)

Fig. 6: Single Node Performance Comparison. Figure 6a shows sweeping K with fixed $M = N$ =32768, whereas in Figure 6b, we sweep M with fixed K =512. Single node performance with different block values of COAST is shown in Figure 6c.

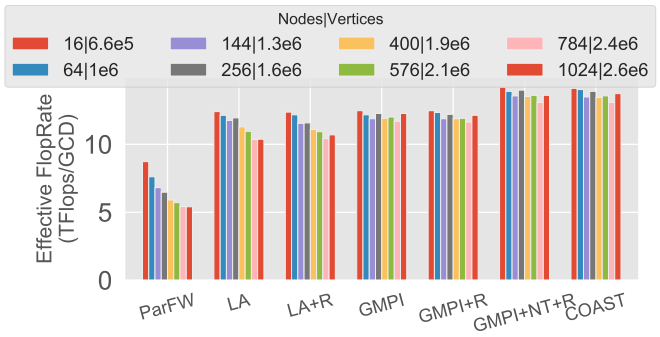


Fig. 7: Weak scaling on Frontier: R stands for Ring, NT is SSRGEMM on $A \times B^T$, and GMPI is GPU-aware MPI.

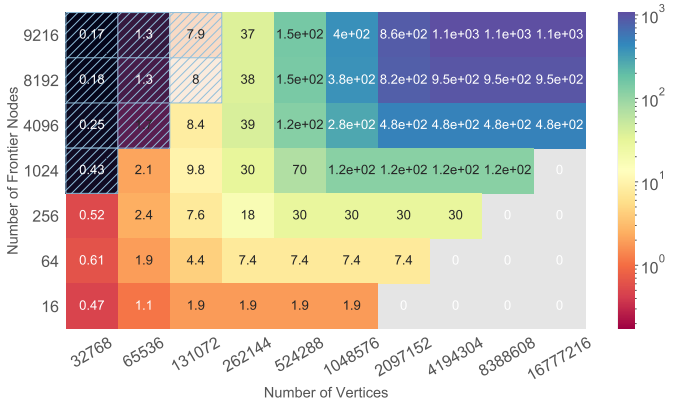


Fig. 8: The COAST algorithm uses HYPERMOD to determine the ideal implementations for obtaining the best performance. Zero performance denotes running out of memory. The best performance in the hatched cells is obtained through slower naïve components over the solid cells that use optimized variant.

from 70 MB and up, ring broadcast offer 2× the effective bandwidth over MPI broadcast. In Figure 5b, we see a clear 2× advantage for ring broadcast in wall time for large local matrix sizes.

VII. SCALING ON FRONTIER

In this section, we present the challenges and our mitigation strategies for scaling COAST to nearly the full Frontier system. Additionally, we provide the actual run result to compare with our HYPERMOD projection model and provide lessons learned from running applications on an exascale machine.

A. Performance Projection on Full Frontier

Finally, we combine the SSRGEMM and effective bandwidth HYPERMOD models along with the Equation (3) for performance results on Frontier. For both the naïve and the optimized PARALLELFW, we project the performance of Frontier by using our models and parameters from Table V. The optimized COAST consists of components that are asymptotically faster than naïve PARALLELFW implementations. Figure 8 shows the heat map for the projected performance on Frontier. In these plots, the values on every column from the bottom row to the top row represent strong scaling. Similarly, the diagonal values capture the weak memory scaling. The optimized implementation is projected to achieve 1.1 exaflop/s on 9,216 nodes with 4.1×10^6 vertices or more. For peak performance, the optimized implementation COAST is 3× faster than the naïve PARALLELFW. The models capture real-world characteristics in Figure 8, including out of memory (bottom right), strong scaling performance that wanes on many nodes with smaller matrix sizes (top left), and best performance on many nodes for bigger graphs (top right).

B. Anticipated Challenges – HYPERMOD to Frontier

Despite our best effort to include every parameter in the performance projection, we would like to discuss the limitations of our projections and potential circumstances that could pose a threat to achieving 1 exaflop/s as predicted.

Hardware faults: The HYPERMOD model does not account for hardware faults such as a node or network failure. We can alleviate the hard faults by using checkpoint/restart from local NVMe storage, but this comes with a nonzero overhead. If we checkpoint every 30 minutes, then we will incur a performance overhead of 2.5% and drop our projection to 1.07 exaflop/s. If we checkpoint more frequently, say every 6 minutes or less, then we will miss the 1 exaflop/s goal. In contrast to hard faults, dealing with soft faults is significantly

Scenarios	Pubmed path	SPOKE path
Scenario 1: Exact match	C1448177 TNF protein → C5203670 COVID-19 → C1699239 IL7 protein	P01375 TNFA_HUMAN (Protein) → DOID:0080600 COVID-19 (Disease) → P13232 IL7_HUMAN (Protein)
Scenario 2: All concepts in a Pubmed path map to SPOKE concept		
Scenario 2a: Pubmed path has a partial path in SPOKE	C0027059 Myocarditis → C0042769 Virus Diseases → C0024312 Lymphopenia	DOID:820 Myocarditis (Disease) → DOID:934 Viral Infectious Disease (Disease) but no direct link between DOID:934 Viral Infectious Disease (Disease) and DOID:614 Lymphopenia (Disease).
Scenario 2b: All the concepts in the Pubmed path could map to SPOKE concepts, but none of the edges exist in SPOKE shortest paths	C0021745 Interferon Type II → C0042774 Virus Replication → C0004364 Autoimmune Diseases → C0003469 Anxiety Disorders → C3887665 Thrombopoietin	None of the edges exist in SPOKE shortest paths
Scenario 3: Some concepts in the Pubmed paths cannot map to SPOKE concepts		
Scenario 3a: The collapsed paths are also found in the SPOKE shortest paths	C1699239 IL7 protein → C5203670 COVID-19 → G0000159 Expression → C1698754 IL6 protein	After removing G0000159 expression, the full path is P13232 IL7_HUMAN (Protein) → DOID: 0080600 COVID-19 (Disease) → P05231 IL6_HUMAN (Protein)
Scenario 3b: The collapsed paths have a partial match in the SPOKE shortest paths	C0035222 Respiratory Distress Syndrome → C0021400 Influenza → C0007634 Cells → C1171892 VEGF protein	After removing C0007634 Cells, only a partial path exists in SPOKE, since there is no edge between DOID:8469 influenza (Disease) and P15692 VEGFA_HUMAN (Protein)
Scenario 3c: Even after collapsing no full or partial match in SPOKE	C3887665 Thrombopoietin → C0003469 Anxiety Disorders → C0004364 Autoimmune Diseases → C0012854 DNA → C0032136 Plasmids → C1447107 HGF protein	After removing the two unmapped concepts, C0012854 DNA and C0032136 Plasmids, the collapsed path is not in SPOKE
Scenario 4: Path exists in SPOKE, and no path exists in the Pubmed co-occurrence graph	Does not exists in Pubmed	(1) DOID:11394 Adult Respiratory Distress Syndrome (Disease) → CHEMBL3989798 FORMOTEROL FUMARATE (Compound) → P07900 HS90A_HUMAN (Protein) → P09341 GROA_HUMAN (Protein) and (2) P09341 GROA_HUMAN (Protein) → P01579 IFNG_HUMAN

TABLE VI: Example paths on Pubmed and SPOKE.

more challenging because we do not have algorithm-based fault tolerance techniques [18].

Frequent random events: Random events such as network jitters can affect the accuracy of our model. In addition to this, we do not model the effects of random events such as staggered processors, GPUs and network links, thermal performance throttling, clock speed fluctuations, skew, or operating system noise. These random events can affect performance in unpredictable ways.

C. Communication Cost at Large Scale

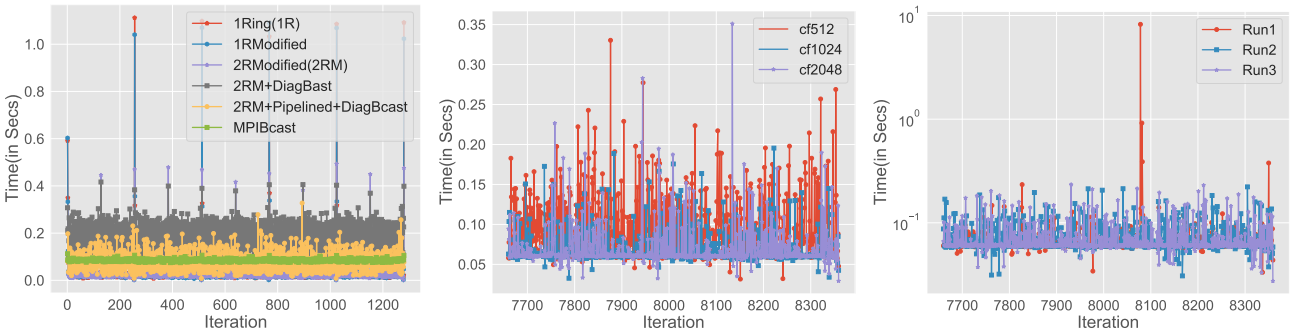
During our Summit development, we observed that panel broadcast cost was mostly hidden by overlapping with the computation. However, when we deployed our code on Frontier, we observed that ring broadcast bandwidth/latency is insufficient to maintain a fully computationally bound run. Communication dominated computation for the processors located toward the tail of the ring.

We introduce two separate strategies to cope with this issue: (1) we keep the diagonal broadcast with the standard library broadcast to reduce the latency on the critical path, and (2) we pipeline the ring broadcast by breaking the messages into smaller 4 MB chunks. Recall that our COAST algorithm is already pipelined by design, and this additional messages chunking pipeline targets the excess communication delays at tail of the ring.

A slow link in the chain of network components can cause a bottleneck, thereby forcing all processes to wait, and Figure 9a shows the impact of a bottleneck as it affects ring and 2-ring broadcasts. The ring-type broadcast often yields more performance but is more susceptible to such issues and exhibits higher performance variance. The implicit barrier in the library broadcast can address this wait at the end of the tail but at a cost of higher overall execution time. Our novel semi-synchronous pipelined ring broadcast algorithm outperforms the vendor-optimized library solutions while maintaining lower variance. Figure 9b shows our experiment using different chunking sizes and the best overall performance with a chunk size of 4 MB.

D. From 1 to 9,200 Nodes

At the time of this writing, Frontier was still an early access system and therefore more prone to the hardware faults and frequent random events. To avoid these problems, we started our large-scale experiments from a lower number of vertices, N = 583,680, to search for the smallest N for which we could achieve 1 exaflop/s. Figure 2b shows that our runs closely match



(a) (b) (c)

Fig. 9: Scaling Challenges on Frontier: Figure 9a shows the communication time per iteration recorded on the last rank using 2,809 nodes for $P_r = P_c$ = 212 and N = 6,946,816. The time taken on the last 700 iterations with different chunk sizes and the variations among three different runs under same settings are shown in Figures 9b and 9c, respectively.

our projection for 9,216 nodes and finally achieve 1 exaflop/s at a vertex size of N = 7,065,600 on 9,200 nodes, which results in 75% of peak and 99% memory-constant parallel efficiency when compared with same memory usage on a single-node run. At any particular iteration, the total performance on that iteration and the cumulative aggregated performance on 9,200 nodes are shown in Figure 2c. The cumulative aggregate performance achieved a stable region of 97% of 1 exaflop/s at the 500^{th} iteration, and at around 52% of the total run, the machine achieved 1.004 EF/s.

E. Lesson Learned from Large-Scale Runs

We described the challenges of efficiently utilizing a large HPC system in Section VII-B. Hardware faults and random events cause performance reduction or even termination for some faults. Figure 9c shows that three consecutive runs with the same parameters all experience different network delays, which causes a 5% variation of performance.

Frontier consists of more than 80,000 cables, 9,400 CPUs, and more than 37,000 GPUs. There are millions of components. Even with the high reliability of a single component, system-level hardware faults are not uncommon when you multiply the relatively low failure rate by millions of components.

Beyond the possibility of a parts failure, the staggered network links and processes in a ring-based collective affect MPI clusters negatively due to the long linear chains. Because any staggered link along this chain could cause a significant slowdown in the pipeline, MPI clusters will suffer either from rerouting costs or slower routes.

In future, the high-performance codes may rely on performance-sensitive checkpoints and link failures tolerant MPI collective communication. One-sided communication and checkpoint strategies that use NVMe, in-memory file systems, or both are worth exploring for long-running jobs.

VIII. Use Case

As described in Section III-C, we utilize co-occurrence and occurrence within a paper along with the citations while processing the Pubmed literature graph. However, SPOKE does not have any paper-related information. Hence, we use only co-occurrences to demonstrate the power of the integration between the SPOKE biomedical knowledge network and the Pubmed literature graph. Owing to the extant pandemic, we focused on biomedical concepts related to COVID-19. COVID-19-related concepts that appear in SPOKE v3 as vertices were identified by using SPOKE's Neighborhood Explorer [19]. A total of 54 connected vertices, of which 6 are of type *Disease* and 48 are of type *Protein*, are selected as sample data. We validate each path determined from the Pubmed co-occurrence graph against its presence on the SPOKE v3 graph. To compare paths found in the two graphs, mappings between concepts are applied by using the Unified Medical Language System's (UMLS's) Concept Unique Identifiers (CUIs) [20].

Given that one SPOKE identifier could map to multiple UMLS CUIs, the selected 54 COVID-19 concepts in SPOKE were mapped to 68 UMLS CUIs. However, the possible discrepancy on the coverage of concept types and the missing concept mappings imply that not all concepts appearing in the Pubmed paths could be mapped to SPOKE concepts.

Four scenarios could be seen based on the presence or absence of each path or parts of each path in both graphs. Table VI presents the different scenarios and the example paths, and Figure 10 summarizes the counts for each scenario.

IX. Implications

COAST is the first demonstration of a graph analysis running at just over 1 exaflop/s. The entire PubMed graph with co-occurrences, occurrences, and citations is completely connected, which means a path always exists between any pair of CUIs through paper citations. A biomedical research study built around a hypothesis and with finite available resources, by its inherent nature, is of limited scope. Thus, the ability to establish paths between any pair of vertices within the richness of PubMed in a reasonable time has the potential to revolutionize biomedical research and apply national research funds more effectively. Furthermore, the existence of paths in SPOKE and its non-existence in the co-occurrence graph of PubMed will naturally lead to a deeper inquiry of most published research studies. This observation also supports the relevance of literature-based discovery, which is a process

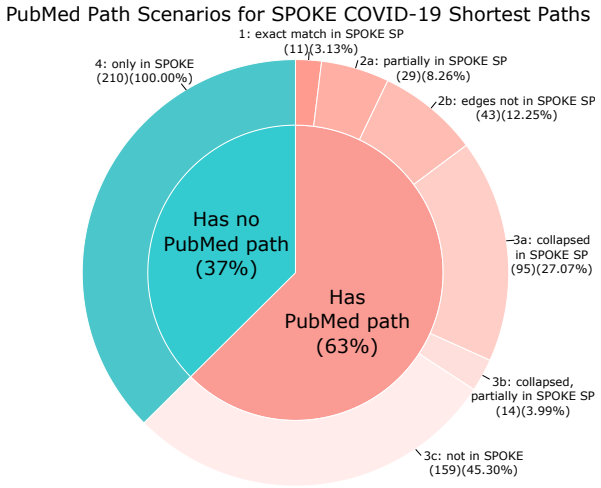


Fig. 10: Mapping the Pubmed discovered paths using only co-occurrence to the SPOKE knowledge graph for different scenarios based on the presence or absence of each path or parts of each path in both graphs.

concerned with uncovering previously unknown relationships between existing concepts from the scientific literature.

SPOKE and PubMed present two different perspectives of relations among medical concepts. It is an open problem to determine a scientific way to integrate the knowledge from PubMed into SPOKE. However, it is also accepted that a large corpus like PubMed can enhance scientific datasets and SPOKE because the latter are human curated. On the algorithmic front, for truly sparse problems with larger separators, we envision a distributed, communication-avoiding supernodal realization of COAST based on prior results for sparse Gaussian elimination.

X. ACKNOWLEDGEMENTS

This material is based upon work supported by the US Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research (Robinson Pino, program manager) under contract DE-AC05-00OR22725 and by the National Science Foundation (NSF) under award number 1710371. SPOKE development was funded in substantial part by the NSF Convergence Accelerator awards 1937160 and 12033569. This research used resources of the OLCF which is a DOE Office of Science User Facility supported under contract DE-AC05-00OR22725.

REFERENCES

[1] S. E. Baranzini, K. Börner, J. Morris, C. A. Nelson, K. Soman, E. Schleimer, M. Keiser, M. Musen, R. Pearce *et al.*, "A biomedical open knowledge network harnesses the power of AI to understand deep human biology," *AI Magazine*, vol. 43, no. 1, pp. 46–58, 2022.

[2] T. A. Tummino, V. V. Rezelj, B. Fischer, A. Fischer, M. J. O'Meara, B. Monel, T. Vallet, K. M. White, Z. Zhang, A. Alon *et al.*, "Drug-induced phospholipidosis confounds drug repurposing for SARS-CoV-2," *Science*, vol. 373, no. 6554, pp. 541–547, 2021.

[3] D. E. Gordon, G. M. Jang, M. Bouhaddou, J. Xu, K. Obernier, K. M. White, M. J. O'Meara, V. V. Rezelj, J. Z. Guo, D. L. Swaney *et al.*, "A sars-cov-2 protein interaction map reveals targets for drug repurposing," *Nature*, vol. 583, no. 7816, pp. 459–468, 2020.

[4] M. R. Garvin, C. Alvarez, J. I. Miller, E. T. Prates, A. M. Walker, B. K. Amos, A. E. Mast, A. Justice, B. Aronow, and D. Jacobson, "A mechanistic model and therapeutic interventions for covid-19 involving a ras-mediated bradykinin storm," *elife*, vol. 9, p. e59177, 2020.

[5] 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.

[6] 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.

[7] 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.

[8] E. Solomonik, A. Buluç, and J. Demmel, "Minimizing communication in all-pairs shortest paths," in *Proceedings of the 27th IPDPS*, 5 2013.

[9] 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.

[10] R. Kannan, P. Sao, H. Lu, D. Herrmannova, V. Thakkar, R. Patton, R. Vuduc, and T. Potok, "Scalable knowledge graph analytics at 136 petaflop/s," in *SC20*. IEEE, 2020, pp. 1–13.

[11] 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. [Online]. Available: https://www.semanticscholar.org/cord19

[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] 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

[14] 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.

[15] AMD. AMD CDNA 2 white paper. AMD. [Online]. Available: https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf

[16] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, "Accelerating numerical dense linear algebra calculations with gpus," *Numerical Computations with GPUs*, pp. 1–26, 2014.

[17] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM TOMS*, vol. 38, no. 1, p. 1, 2011.

[18] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. 100, no. 6, pp. 518–528, 1984.

[19] S. Huang, J. Morris, and S. E. Baranzini, "Neighborhood explorer." [Online]. Available: https://spoke.rbvi.ucsf.edu/

[20] O. Bodenreider, "The Unified Medical Language System (UMLS): integrating biomedical terminology," *Nucleic Acids Research*, vol. 32, pp. D267–D270, 2004.

[21] P. Sao, H. Lu, R. Kannan, V. Thakkar, R. Vuduc, and T. Potok, "Scalable all-pairs shortest paths for huge graphs on multi-GPU clusters," in *Proceedings of the 30th International Symposium on HPDC*. ACM, 2021, p. 121–131.

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

[23] J. Kurzak, Y. M. Tsai, M. Gates, A. Abdelfattah, and J. Dongarra, "Massively parallel automated software tuning," in *Proceedings of the 48th ICPP*, 2019, pp. 1–10.

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

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

[26] 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.

[27] "NCBI Gene." [Online]. Available: https://www.ncbi.nlm.nih.gov/gene/

[28] J. Chambers, M. Davies, A. Gaulton, A. Hersey, S. Velankar, R. Petryszak, J. Hastings, L. Bellis, S. McGlinchey, and J. P. Overington, "Unichem: a unified chemical structure cross-referencing and identifier tracking system," *Journal of Cheminformatics*, vol. 5, p. 3, 2013.

[29] "The National Cancer Institute Thesaurus." [Online]. Available: https://ncit.nci.nih.gov

[30] S. Jupp, T. Burdett, C. Leroy, and H. E. Parkinson, "A new ontology lookup service at EMBL-EBI," in *SWAT4LS*, 2015.

XI. Appendix

A. Data Pipeline

A graph is a universal language that can describe many physical or social phenomena by representing relationships between constituents. Graph-based analysis relies on knowledge graphs and graph-structured data to integrate information from multiple domains (e.g., biology, chemistry, natural language processing). In this study, we focus on running the expensive computations required for a knowledge graph on HPC resources. As shown in Figure 11, we integrate the text data from Pubmed and CORD-19 publications along with the SPOKE knowledge graph on MongoDB, a key-value storage solution. We represent the integrated data as a knowledge graph in a sparse matrix representation by using vertex and edge encoding techniques as detailed in Section III-C. This data is stored on the GPFS/Lustre network storage connected to the HPC computing systems. This matrix is consumed by COAST code, and the output is typically hundreds of terabytes using ADIOS/MPIIO. Because this is a massive and dense dataset, tuning graph databases that are designed for holding sparse information can be difficult. Hence, we will directly expose service APIs to query this dense output on GPFS/Lustre.

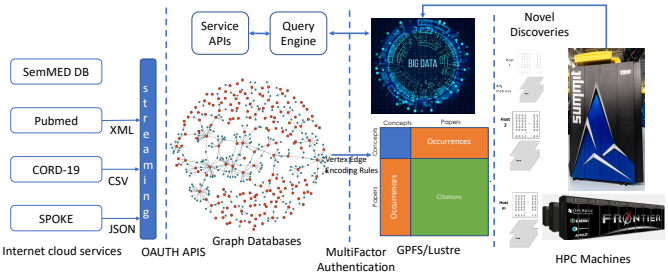


Fig. 11: Data pipeline and workflow.

B. Communication Optimizations

The COAST algorithm performs a high volume of communication in the $O(\frac{n^2}{P_x} + \frac{n^2}{P_y})$. The communication complexity is explained in Section IV-D. This means that during scaling to large nodes, we will spend more time in communication than in computation. To mitigate this issue, we propose (1) look ahead to accelerate the critical path of diagonal update and broadcast, (2) ring broadcast protocol, (3) rank mapping, and (4) optimizing intranode communication.

1) Look-Ahead Technique

Recall that the most expensive communication operation is PANELBCAST, and the most expensive compute operation is OUTERUPDATE in any iteration of Listing 1. Look-ahead technique overlaps the two operations. We use the idea of breaking the dependency between the k and $k{+}1$ iterations to achieve the overlap.

Consider the execution of the k-th iteration of Listing 1. Let us assume that we have performed the PANELBCAST(k). Each process has received k-th horizontal and vertical panels and is ready to perform the OUTERUPDATE(k). Then, any process in $P_r(k{+}1)$ and $P_c(k{+}1)$, except $P_{k+1,k+1}$, will first perform the OUTERUPDATE(k) on the $k{+}1$-th panels and wait for the DIAGBCAST($k{+}1$). Then, the process $P_{k+1,k+1}$ will perform OUTERUPDATE(k) on the block $A_{k+1,k+1}$, followed by DIAGUPDATE($k{+}1$) and DIAGBCAST($k{+}1$). Following that, we perform the OUTERUPDATE(k) on the $k{+}1$-th panels.

After DIAGBCAST($k{+}1$), processes in $P_r(k{+}1)$ and $P_c(k{+}1)$ can perform the PANELUPDATE($k{+}1$) and initiate PANELBCAST($k{+}1$). Subsequently, all the processes on $P_r(k{+}1)$ and $P_c(k{+}1)$ perform the OUTERUPDATE(k) on the remaining matrix. Meanwhile, all other processes initiate OUTERUPDATE(k) on the GPU, and the host CPU waits for the PANELBCAST($k{+}1$). At the end of this step, all processes have finished PANELBCAST($k{+}1$) and OUTERUPDATE(k). We perform PANELBCAST($k{+}2$) and OUTERUPDATE($k{+}1$) concurrently in the next iteration.

The look-ahead COAST requires some small block SRGEMM to be invoked and also additional buffer management. Sao et al. [21] present the algorithmic realization of the PARALLELFW algorithm with a look-ahead technique.

2) Ring Broadcast over Tree-Based Broadcast

The traditional library provided by MPI_Bcast may not be the most efficient for our application because it uses a *kd-tree* pattern (a.k.a., 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 a ring pattern that costs $(P-1)\alpha + w\beta$ is optimal for bandwidth but worse for latency. 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 COAST. In this paper, we also focus on the Ring-2 Modified (R2M) implementation to help scale to Frontier's 9,400 nodes. In the case of R2M, the p processes are split into two groups. As one may expect, first $0 \rightarrow 1$, and then the leftover $p-1$ processes are divided into two equal parts. After the division, processes 2 and $p/2$ act as sources for two rings. That is, 2 sends data to 3, 3 sends data to 4, and so on until $p/2-2$ sends data to $p/2-1$. Simultaneously, another process communicates concurrently in its own ring as process $p/2$ sends data to $p/2+1$, $p/2+1$ sends data to $p/2+2$, and so on until $p-1$ sends data to p.

3) Optimal Rank Placement

When creating a 2D logical process grid using MPI, all the MPI ranks within a node will be placed in a process row or process column by default. However, this rank placement is suboptimal when considering data transfer via the 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 the case of Frontier, every node has four separate disjoint NICS connected directly to the Slingshot network. For COAST to utilize all four NICS and

communicate concurrently, we use the optimal rank placements that respect *latin square* ordering—any process rows or process columns will have all NICs working to make effective use of the ring broadcast. At any given instance, every GCD will be participating in a communication directly through the Slingshot. Every process owns the portion of the global data in such a way that it optimally participates in the ring broadcast.

4) Optimizing Intranode Communication

Unlike Summit, Frontier's NICs are directly connected to GPUs. Hence, communicating the GPUs buffers directly through GPU-aware MPI will be faster than communicating through the CPU because this will avoid unnecessary GPU-CPU-GPU data movement. Hence, in this paper, we explore the GPU-aware MPI support provided by Cray-MPICH=8.1.2. Frontier enables GPU-aware MPI through a dedicated module called `craype-accel-amd-gfx90a`. We can enable or disable GPU-aware MPI through an environment variable. We observe that GPU-aware MPI exhibits improved performance over the CPU-based ring broadcast Section VI-E.

C. OPTSSRGEMM-NT:

We took the MAGMA kernel as the starting point and produced a fast implementation of the min-plus kernel OPTSSRGEMM-NT by using vectorization, pipelining, and autotuning.

1) Vectorization

First, we restructured the kernel to use the `float2` data type. This makes it easier for the compiler to map the computation to packed FP32 instructions. Figure 12 shows the device function that implements the innermost loops of our kernel. In this case, the `float2` type was enough to produce the packed `add` instruction. At the same time, the compiler successfully reduced the two calls to `fminf` to a single `min3` instruction. This can be verified by inspection of the assembly code produced when compiling with the `--save-temps` flag.

```
__device__
void compute_minplus(float2 a[THR_M],
                     float2 b[THR_N],
                     float  c[THR_N][THR_M])
{
    for (int j = 0; j < THR_N; ++j)
        for (int i = 0; i < THR_M; ++i)
            c[j][i] = fminf(fminf((a[i]+b[j]).x,
                                  (a[i]+b[j]).y),
                            c[j][i]);
}
```

Fig. 12: Device function implementing the vectorized loops of the min-plus kernel.

2) Pipelining

Next, we structured our code to allow for software pipelining, which is a solution popularized by the CUTLASS library [22]. We refrained from pipelining the accesses to shared memory. Instead, we relied on having multiple waves in flight. By scheduling shared memory instructions from some waves simultaneously with arithmetic instructions from other waves, the CU can hide both the cost of issuing shared memory instructions and the cost of the associated access latency. At the same time, we did implement explicit pipelining of reads from the global memory because the associated latency is large and more difficult to hide. The solution is implemented by maintaining two sets of local arrays and two sets of shared memory buffers and alternating accesses between even and odd iterations. We consider this to be a fairly balanced implementation in terms of performance and complexity trade-offs.

TABLE VII: Real-world datasets from SuiteSparse matrix collection [17].

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
Scientific Problems				
`Zd_Jac3`	ZD	2.28×10^4	1.92×10^6	Simulation
`nd6k`	ND6K	1.80×10^4	6.90×10^6	2D/3D mesh
`pkustk07`	PS7	1.69×10^4	2.42×10^6	Structural

3) Autotuning

Being derived from MAGMA, the kernel inherits MAGMA's blocking factors (Figure 4a), and these factors must be tuned for maximum performance. We did this by applying simple, brute-force tuning (i.e., by generating a large number of kernels and testing them for best performance). We trimmed the search space presented by the blocking factors by applying a set of simple constraints listed in Figure 4b.

Constraints 1–3 force the number of work items per group to be between 256 and 1,024, and be divisible by the wave size, which is 64 in the CDNA™ 2 architecture. Constraint 4 prevents us from exceeding the 64 KB shared memory size of the CDNA™ 2 devices. The remaining constraints (5–10) force mutual divisibility of dimensions to eliminate any potential branching/divergence situations.

The result is 25,894 viable kernels, which must be compiled and timed. We tune for one combination of Trans(T)/NoTrans(N) arguments at a time. To speed up the tuning process, we dispatch each sweep to a moderate number of nodes of the Frontier system (between 16 and 32) using a HIP port of the Benchtuning OpeN Software Autotuning Infrastructure software [23]. The largest job launched in the course of this work took 7 minutes using 32 nodes (256 GCDs). Table IV shows the tuned values of the blocking factors for the packed FP32 SRGEMM kernel, OPTSSRGEMM-NT.

D. Realworld Dataset Experiments on Single Node:

1) 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 (FW-CPU)** : This implementation is an efficient multithreaded OpenMP variant that 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 here. Hence, we

consider this as the baseline in Figure 13 for comparison against other baseline algorithms.

- **BOOSTDIJKSTRA (Boost-D)**: This APSP implementation uses DIJKSTRA's algorithm from the popular Boost Graph Library (BGL) [24]. BGL also provides a BFW that is slower than BOOSTDIJKSTRA, so we omit it.
- **Δ-STEP (Galois)**: We use the parallel Δ-stepping variant of DIJKSTRA's algorithm [25] for computing the single-source-shortest path in Johnson's algorithm. We use the parallel Δ-stepping algorithm from the Galois Graph library [26]. The Δ-stepping requires tuning a Δ parameter for each input graph. Our Δ-STEP-based APSP is autotuned—we try different values of Δ during the first few SSSP calls and pick the best Δ for the rest of the execution.

2) Test Graphs

Our graph datasets for the single-node experiments use the real-world graphs shown in Table VII. We chose the graphs to be sufficiently large while running in a reasonable time on a single node. The DIJKSTRA 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.

3) Single Node Performance Evaluation

We compared SSRGEMM against other single-node implementations using the datasets from Table VII, which were taken from the Suite Sparse Matrix Collection [17]. 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. The CPU algorithms were executed on a single Frontier node with 64 OpenMP threads and on a single Summit node with 42 OpenMP threads. Similar to other sections, we used only one GCD of Frontier and one GPU on Summit. We are reporting the wall clock time in seconds. The results appear in Figure 13.

Despite the difference in the floating-point operation rates between CPU and GPU, we believe the comparison reflects the availability of the current state-of-the-art algorithms on different architectures—AMD EPYC 7A53 vs. POWER9, MI250X vs. NVIDIA Volta, and Frontier vs. Summit.

None of the other implementations are competitive with COAST. For the **FW-CPU** with the `mycielskian15` dataset, COAST is 23× faster on Frontier and COAST on Frontier is 1.59× faster over Summit.

DIJKSTRA uses a priority queue-based implementation that performs equally well on POWER9 and AMD EPYC. We expect **Δ-STEP (Galois)** to be the slowest because it is neither work optimal nor scalable. Both **Δ-STEP (Galois)** and **FW-CPU** perform better on the AMD EPYC 7A53 than on the POWER9 owing to increased bandwidth and throughput on the AMD CPU.

Similarly, BOOSTDIJKSTRA is not competitive to our implementation of DIJKSTRA for Frontier or Summit. DIJKSTRA can be better than 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. Our DIJKSTRA implementation is CPU based, and we could not find an efficient priority queue implementation for GPUs to implement DIJKSTRA on MI250X and NVIDIA Volta.

Both Summit and Frontier have fat nodes with large amounts of memory—512 GB in each node. In most cases, the speedup of MI250X over CPU on Frontier is not as significant as it is on Summit. This is because the AMD EPYC CPUs on Frontier have higher throughput and increased bandwidth.

E. Example Paths on Pubmed and SPOKE

The biomedical concepts in the Pubmed graph use CUIs as defined by the UMLS, whereas SPOKE uses identifiers from multiple ontologies and taxonomies.

The Pubmed graph is composed of two types of vertices.

(1) **Concept vertices** represent unique biomedical terms (e.g., drugs, genes, diseases, symptoms) and are standardized using the UMLS [20]. There are 127 different concept types and over 290,000 unique concepts mapped to their CUIs. (2) **Abstract vertices** represent the Pubmed and CORD-19 abstracts. The vertices can be connected in three different ways.

a) Concept-to-concept relations (co-occurences)

The connections between concepts represent relationships described in these abstracts. For the shortest path computation, we assign these edges a weight, which is calculated as the number of times the two concepts appear together in a predication divided by the total number of predictions in which these concepts appear.

b) Concept-to-abstract relations (occurences)

The connections between abstracts and concepts represent occurrence of concepts in abstracts. For the shortest path computation, we assign these edges a weight that represents the number of times a concept, c, appears in an abstract, p, divided by the total number of concepts that appear in p.

c) Abstract-to-abstract relations (citations)

The connections between abstracts represent citation relations between them. 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.

For the shortest path calculation, we apply $-\log$ to have weights where lower values represent higher importance. This results in 213 million relations: 14 million relations between concepts, 196 million relations between papers and concepts, and 3 million citation relations between papers. Additional information about how the graph was produced is provided in our data readme [13].

Although, the SPOKE v3 knowledge graph has 19 vertex types, the mappings to CUIs can be built for 13 vertex types: Anatomy, BiologicalProcess, CellType, CellularComponent, Compound, Disease, Gene, MolecularFunction, Organism, PharmacologicClass, Protein, SideEffect, and Symptom. Three methods are used to create the mappings based on the vocabularies of the vertex types:

(1) No mapping needed: CUIs are the vertex identifiers in SPOKE. The vertex type in this category is SideEffect.

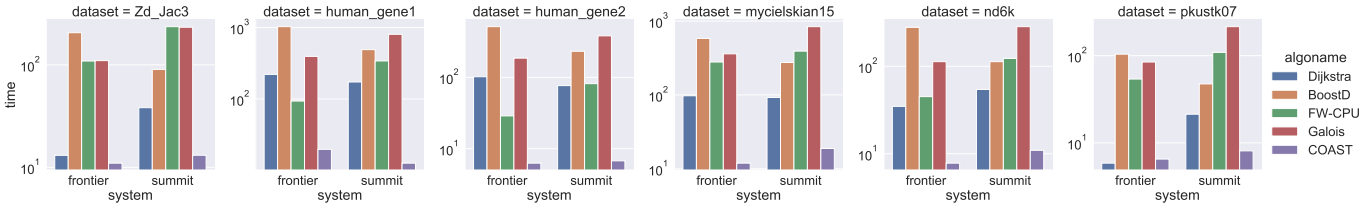


Fig. 13: Time to solution for different algorithms on real-world datasets on Frontier and Summit.

(2) Direct mapping: SPOKE identifiers or vertex properties have the vocabularies covered by UMLS. The vertex types that have identifiers in SPOKE that use vocabularies covered by UMLS (e.g., MeSH, Gene Ontology, NCBI Taxonomy) are BiologicalProcess, CellularComponent, MolecularFunction, Organism, PharmacologicClass, and Symptom. Also, some Compound vertices use the Drug Bank IDs as vertex identifiers. Vertex properties in SPOKE sometimes provide vocabularies covered by UMLS. Some disease vertices in SPOKE have MeSH IDs.

(3) Indirect mapping: Cross-references are used to map the other vertex identifiers to vocabularies covered by UMLS. For vertex type Gene, NCBI's `Homo_sapiens.gene_info` file is used to map GeneID to HGNC [27]. For Compound, UniChem is used to map ChEMBL IDs to either Drug Bank or RxNorm [28]. For Protein, NCI Thesaurus's NCIt-SwissProt mapping is used to map UniProt IDs to NCIt [29]. Finally, for vertex types Anatomy and CellType, the Ontology Lookup Service is used to map UBERON and Cell Ontology IDs to CUIs, NCIts, or FMA [30].

In Table VI, we compare the Pubmed and the SPOKE paths for the different scenarios based on the presence and absence of each path or parts of each path in both graphs.