Efficient Large Scale Sparse Irregular Computations

Somesh Singh

Post-doctoral Researcher, ROMA team, LIP, ENS de Lyon

ROMA team candidate

{ High Performance Computing (HPC), Graph computing, Tensor computations }

23rd May, 2024

Background

Legend

- Tensor computation
- Graph computing
- Parallelization
- GPU
- Application

PhD and Master's | July 2014 – June 2021

Indian Institute of Technology Madras, India

- parallel inexact graph computing on graphics processing unit (GPU)

Post-doctoral researcher | September 2021 – now

ROMA team at LIP, ENS de Lyon

- high-performance parallel sparse tensor computations

2014 2021 now

Industry collaborations

- approximate nearest neighbor search (ANNS) on a graph index using a GPU } Research intern | Microsoft Research India (2020)
- parallelizing a particle-tracking library used for the Large Hadron Collider (LHC) } Google Summer of Code participant | CERN (2018)

Background

Update since application:
The work "**Efficient Parallel Sparse Tensor Contraction**" is submitted to the "IEEE Transactions on Parallel and Distributed Systems (TPDS)"
- Code released (link)

Legend
- Tensor computation
- Graph computing
- Parallelization
- GPU
- Application

PhD and Master's | July 2014 – June 2021
Indian Institute of Technology Madras, India
- parallel inexact graph computing on graphics processing unit (GPU)

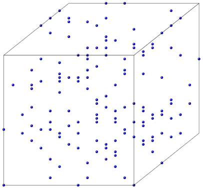
Post-doctoral researcher | September 2021 – now
ROMA team at LIP, ENS de Lyon
- high-performance parallel sparse tensor computations

2014 2021 now

Industry collaborations
- approximate nearest neighbor search (ANNS) on a graph index using a GPU } Research intern | Microsoft Research India (2020)
- parallelizing a particle-tracking library used for the Large Hadron Collider (LHC) } Google Summer of Code participant | CERN (2018)

Research Context and Motivation

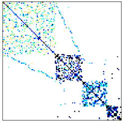
Computations on Graphs and Sparse Tensors



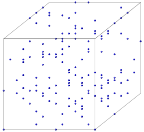
Graphs and sparse tensors

- Tensors are multi-dimensional arrays and model multi-dimensional data

1D Tensor / Vector



2D Tensor / Matrix



3D Tensor / Cube

- Sparse tensors have zero-value elements in majority

- Graphs model connected data
  - nodes: entities of interest
  - edges: relationships among entities

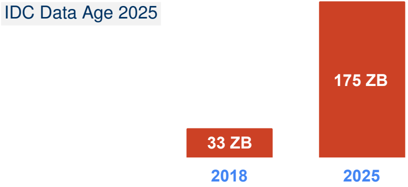
Social Network

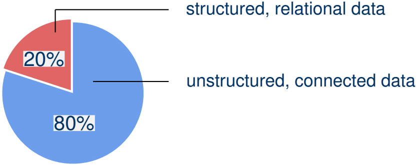
node: person

edge: friendship

Need for high-performance computing on graphs and sparse tensors

- Age of big data: generating large volumes of data at an accelerating rate

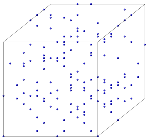




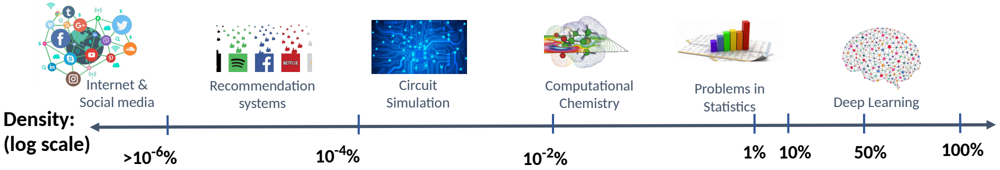
- Graphs and sparse tensors model unstructured data arising in **diverse disciplines**
  - machine learning, scientific computing, bioinformatics, quantum computing, finance, cybersecurity, . . .
- Time to solution: perform high-speed analysis of data to obtain solution in time for it to be useful

* 1 ZB = 10^{12} GB

Challenges and opportunities



- Sparsity enables scalability: enhances computational and storage efficiency
- Sparsity disrupts locality: storing only the nonzeros gives rise to unpredictable access patterns
- Tensors vary in dimensionality, sparsity and size



- A single data structure and algorithm cannot achieve peak performance on the full spectrum of sparsity

Graphics Source: Google Images

Past Research Work

Graph Computing

Sparse Tensor Computations

Parallelization

PhD research

- parallel graph computing on GPU
- inexact computations on graphs
- performance-accuracy trade-off
- GPU-aware and graph algorithm-specific techniques

TMSCS 2018, PPoPP 2019, ICPP 2020, TODAES 2021, GECCO 2023

Ranking vertices using betweenness centrality (BC)

Speedup	Accuracy
$> 2\times$	$> 95\%$

Baseline: vertex ranks computed using exact BC scores from the state-of-the-art GPU-method[1]

Industry collaborations

- Research internship at Microsoft Research India (**continued collaboration**)
  - approximate nearest neighbor search (ANNS) on a graph index using a single GPU

  manuscript (arxiv 2024) (under review); patent application filed (2024)
- Google Summer of Code participant with CERN
  - parallelizing **SixTrackLib**, a particle-tracking library used for the Large Hadron Collider (LHC)

  ICAP 2018, IJMPA 2019

ANNS on a 128 dimensional billion points dataset

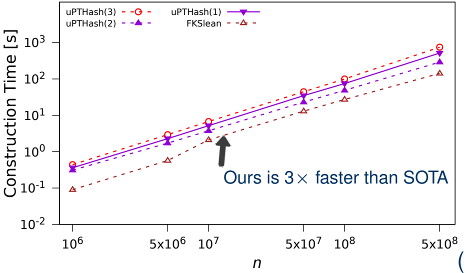
Throughput (queries per second)	Accuracy
$> 10^5$	$\sim 96\%$

Baseline: exact nearest neighbors of points

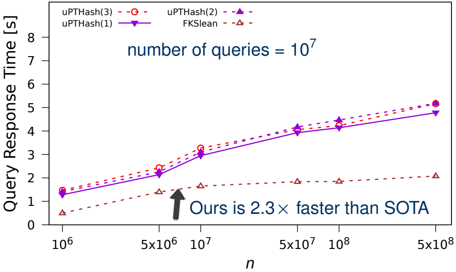
[1] Wang *et al.*, "Gunrock: a high-performance graph processing library on the GPU", PPoPP 2016

Postdoc research (at ROMA)

- Querying for zeros / nonzeros in a sparse tensor
  - Given a d-dimensional sparse tensor $\mathcal{T}$, efficiently answer queries: "$\mathcal{T}[i_1, \ldots, i_d] = 0$ or $\neq 0$?"
  - A subproblem in a tensor decomposition method, used for finding patterns in massive datasets
  - Our method: A space efficient static hashing method with a worst case $O(1)$ lookup



(Lower is better)



- Efficient parallel sparse tensor contraction (⇒ developed in the next slides)

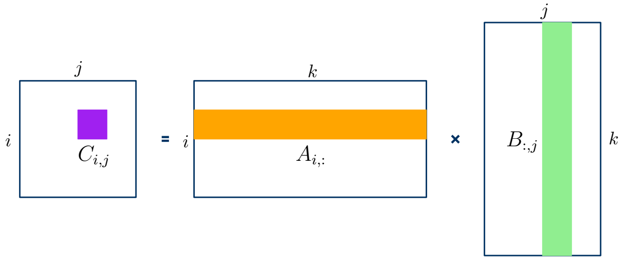
GrAPL 2022 (IPDPSW), JEA 2022, ESA 2023, manuscript (2024; under review)

Efficient Parallel Sparse Tensor Contraction

- Tensor contraction is a higher-dimensional analog of matrix-matrix multiplication

- Multiplication of two matrices: $\mathbf{A} \in \mathbb{R}^{I \times K}$ and $\mathbf{B} \in \mathbb{R}^{K \times J}$

$$\mathbf{C}_{ij} = \sum_k \mathbf{A}_{ik} \cdot \mathbf{B}_{kj}$$



k is the contraction index

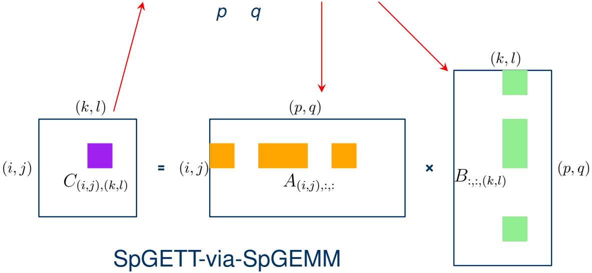
Sparse Tensor Contraction (SpGETT)

Contraction of sparse tensors: $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I \times J \times P \times Q}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{P \times Q \times K \times L}$ with p,q as contraction indices

$$\boldsymbol{\mathcal{C}}_{ijkl} = \sum_p \sum_q \boldsymbol{\mathcal{A}}_{ijpq} \cdot \boldsymbol{\mathcal{B}}_{pqkl}$$

Critical performance hurdles in SpGETT

- unpredictable number of nonzeros in output tensor
- unpredictable sparsity structure of tensors
- different number of operations per thread
- SpGETT-via-SpGEMM: space and time overhead



SpGETT-via-SpGEMM

- Performance issues get aggravated in SpGETT compared to SpGEMM

Key question: How to mitigate the performance challenges in SpGETT?

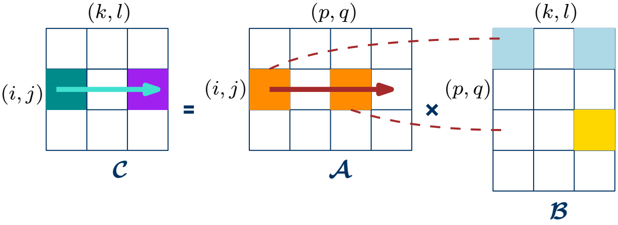
* SpGEMM: sparse matrix-sparse matrix multiplication

Our formulation of SpGETT

- Approach: Perform SpGETT natively on tensors, without explicit conversion to matrices

Contraction of sparse tensors: $\mathcal{A} \in \mathbb{R}^{I \times J \times P \times Q}$ and $\mathcal{B} \in \mathbb{R}^{P \times Q \times K \times L}$ with p,q as contraction indices

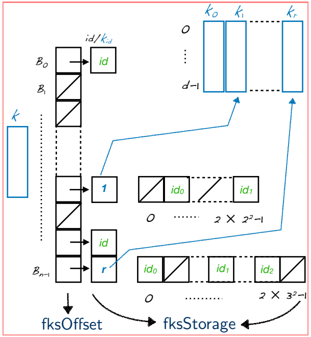
```
parfor nonempty subtensor 𝒜[i,j,:,:] do
    for nonzero 𝒜[i,j,p,q] in subtensor 𝒜[i,j,:,:] do
        for nonzero ℬ[p,q,k,l] in subtensor ℬ[p,q,:,:] do
            v ← 𝒜[i,j,p,q] · ℬ[p,q,k,l]
            if 𝒞[i,j,:,:].lookup(⟨k,l⟩) = True then
                𝒞[i,j,k,l] ← 𝒞[i,j,k,l] + v
            else
                𝒞[i,j,:,:].insert(⟨k,l⟩)
                𝒞[i,j,k,l] ← v
```



$$\mathcal{C}_{ij::} = \sum_p \sum_q \mathcal{A}_{ijpq} \cdot \mathcal{B}_{pq::}$$

- A high number of lookups and insertions necessitates efficient lookup and insert operations
- A dense array has a worst case $O(1)$ lookup, but cannot handle a large number of nonzeros
- The classical hashing methods guarantee $O(1)$ lookup only in the average case

Our hashing method powering SpGETT



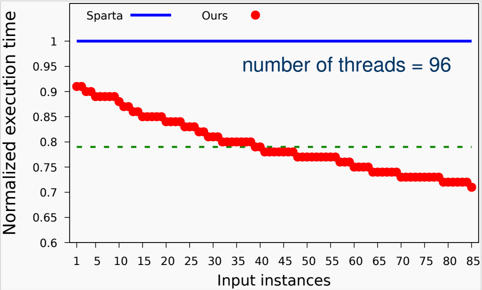
- Space-efficient hashing method with worst-case $O(1)$ lookup per query
- Total storage space $O(n)$; less than $5n$ in practice
- Construction time is $O(n)$, in expectation
- Supports dynamic insertions, and multiple items may have identical hashing indices
- Our hashing method has utility beyond SpGETT
  - can be used for other computations on sparse tensors, and graph algorithms

* **Somesh Singh**, Bora Uçar, "Efficient Parallel Sparse Tensor Contraction" (under review)

** J. Bertrand, F. Dufossé, **S. Singh** and B. Uçar, "Algorithms and Data Structures for Hyperedge Queries", ACM Journal of Experimental Algorithmics (JEA), vol. 27, no. 1, 2022 [**ACM Results Reproduced Badge**]

Experimental evaluation

- Operation considered for evaluation: $\mathcal{C} = \mathcal{A}\mathcal{A}^T$
- Baseline: Sparta§, the state-of-the-art for SpGETT
- Ours* is always faster than Sparta, on real-world sparse tensors
  - Sequential execution: 25% faster on average
  - Parallel execution: 21% faster on average



* **Somesh Singh**, Bora Uçar, “Efficient Parallel Sparse Tensor Contraction” (under review)

§ Liu *et al.*, “Sparta: High-Performance, Element-Wise Sparse Tensor Contraction on Heterogeneous Memory”, PPoPP 2021

Research Program
and
Integration to ROMA & INRIA

Efficient Large Scale Sparse Irregular Computations

{ High Performance Computing (HPC), Graph computing, Tensor computations }

Research Program

Goal: Enable efficient big data analysis for large-scale AI computing and scientific computing

Approach: Design novel parallel algorithms and parallelization techniques for accelerating computations on graphs and sparse tensors using heterogeneous parallel architectures

- **Software** (**open-source**):
  - develop sparse tensor computation libraries and graph processing frameworks for heterogeneous architectures
  - contribute to improving the performance of tensor computations in machine learning libraries like PyTorch and TensorFlow
- **Potential applications** (include, but not limited to): training and inference of deep neural networks, computer vision, cybersecurity, finance, medicine, electronic design automation (EDA), path planning and logistics, . . .

Efficient Large Scale Sparse Irregular Computations

- Graph analytics and mining for queries on graph databases using accelerators (Short Term)
- Parallel sparse tensor contraction using accelerators (Long Term)
- Parallel dynamic graph processing using accelerators (Long Term)

Expertise pertinent to the research program =

$$\left\{\begin{array}{c}\text{Combinatorial scientific computing}\\ \text{Memory-/Energy-aware algorithms}\\ \text{Task scheduling}\end{array}\right\} \bigcup \left\{\begin{array}{c}\text{Irregular computations on GPUs}\\ \text{Sparse tensor computations}\\ \text{Graph computing (approximate?)}\end{array}\right\}$$

Sub-areas in ROMA

My expertise

Parallel sparse tensor contraction using accelerators (1/2)

Motivation

- Sequence of sparse tensor contractions arise in training and inference of neural networks
- Tensor decomposition involves tensor contraction, where one of the operands is sparse

Challenges

- Contraction of sparse tensors suffers from high data movement due to irregular memory access
- Volume of data movement depends on
  - sparsity structure and number of nonzeros in tensors
  - memory layout (data structure) of tensors
  - underlying architecture
- For sequence of tensor contractions, tensor contraction order is crucial $(\mathcal{A}\mathcal{B})\mathcal{C}$ v/s $\mathcal{A}(\mathcal{B}\mathcal{C})$
  - affects memory footprint of intermediate results, and number of operations
  - finding optimal order of dense tensor contractions is NP-hard,[2] sparse tensors add further complexity

[2] Chi-Chung *et al.*, “Optimization of a Class of Multi-Dimensional Integrals on Parallel Machines”, Parallel Processing Letters, 1997

Parallel sparse tensor contraction using accelerators (2/2)

Approach

- Design architecture-aware methods for parallel tensor contraction for GPU
  - leverage GPU's massive multithreading and high memory bandwidth
- Minimize data movement through GPU's memory hierarchy
  - improve **locality** and **data reuse**: adaptive tiling, reordering of sparse tensors
    - tiling is a key technique for optimizing data movement in dense matrix / tensor computations
    - tiling of sparse tensors is challenging due to higher dimensionality and sparsity patterns
    - reordering approaches for sparse matrices are inadequate for sparse tensors due to higher dimensionality and sparsity patterns
  - model data movement and develop performance-cost models to analyze volume of data movement
- The techniques developed will also be applicable to other sparse tensor computations

Aligned sub-areas in ROMA: combinatorial scientific computing, memory-aware algorithms, scheduling

Parallel processing of dynamic graphs using accelerators (1/2)

Motivation

- Real-world graphs are dynamic: they change over time

- Timely completion of complex graph analytics on large dynamic graphs having frequent updates

Challenges

- Running full static analysis on large dynamic graphs, after every update, is infeasible
- Memory requirement of the graph algorithm changes due to updates to the graph structure
- Load imbalance among threads more severe than on static graphs due to structural updates

Parallel processing of dynamic graphs using accelerators (2/2)

Approach

- Design efficient concurrent dynamic data structures (hash map, set, . . .) for GPUs
  - crucial for (dynamic) graph algorithms, sparse tensor computations, and other irregular algorithms
  - parallel data structures on CPU are not a good fit for the GPU due to its massive multithreading
  - current dynamic data structures for GPU do not utilize the GPU's resources efficiently
- Perform batch-updates for more parallelism
- Design adaptive thread scheduling strategies for reducing load imbalance among threads

Aligned sub-areas in ROMA: memory-aware algorithms, scheduling

Parallel processing of dynamic graphs using accelerators (2/2)

Approach

- Design efficient concurrent dynamic data structures (hash map, set, . . .) for GPUs
  - crucial for (dynamic) graph algorithms, sparse tensor computations, and other irregular algorithms
  - parallel data structures on CPU are not a good fit for the GPU due to its massive multithreading
  - current dynamic data structures for GPU do not utilize the GPU's resources efficiently
- Perform batch-updates for more parallelism
- Design adaptive thread scheduling strategies for reducing load imbalance among threads

Aligned sub-areas in ROMA: memory-aware algorithms, scheduling

Next years at ROMA: plenty of exciting research to do in high performance sparse irregular applications !

Update since application: The work "**Efficient Parallel Sparse Tensor Contraction**" is submitted to the "IEEE Transactions on Parallel and Distributed Systems (TPDS)"

- Code released (link) [∼4.5K loc, C/C++]

Service

- Reviewer for journals [over **5**]: IEEE TPDS (since 2021), ACM TACO (since 2022), . . .
- Review committee & external review committee: SC 2024, ECOOP (2023, 2022), GrAPL 2023 (IPDPSW)
- Artifact-evaluation committee [over **10**]: PPoPP (2021, 2018), PLDI (2024, 2022), . . .

Selected Software

- Algorithms and Data Structures for Hyperedge Queries
  - ∼3K loc, C/C++ (open-source)
  - **ACM Results Reproduced** badge
  - **State-of-the-art** for the hyperedge-existence query problem
- Optimize and Integrate Standalone Tracking Library (SixTrackLib)
  - ∼13K loc, C/C++, OpenCL (open-source)
  - Contributed code largely **integrated into the SixTrackLib library maintained by and used at CERN**

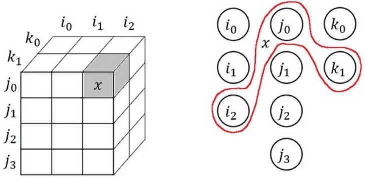
Backup Slides

Tensor	d	Dimensions	n
chicago_crime (T-1)	4	$6{,}186 \times 24 \times 77 \times 32$	5,330,673
vast-2015-mc1-3d (T-2)	3	$165{,}427 \times 11{,}374 \times 2$	26,021,854
vast-2015-mc1-5d (T-3)	5	$165{,}427 \times 11{,}374 \times 2 \times 100 \times 89$	26,021,945
enron (T-4)	4	$6{,}066 \times 5{,}699 \times 244{,}268 \times 1{,}176$	54,202,099
nell-2 (T-5)	3	$12{,}092 \times 9{,}184 \times 28{,}818$	76,879,419
flickr-3d (T-6)	3	$319{,}686 \times 28{,}153{,}045 \times 1{,}607{,}191$	112,890,310
flickr-4d (T-7)	4	$319{,}686 \times 28{,}153{,}045 \times 1{,}607{,}191 \times 731$	112,890,310
delicious-3d (T-8)	3	$532{,}924 \times 17{,}262{,}471 \times 2{,}480{,}308$	140,126,181
delicious-4d (T-9)	4	$532{,}924 \times 17{,}262{,}471 \times 2{,}480{,}308 \times 1{,}443$	140,126,181
nell-1 (T-10)	3	$2{,}902{,}330 \times 2{,}143{,}368 \times 25{,}495{,}389$	143,599,552

Input tensors from FROSTT dataset

Hyperedge Queries in Hypergraphs

- Tensors are multi-dimensional arrays
- A d-dimensional sparse tensor corresponds to a special class of hypergraphs



3D Tensor

Hypergraph

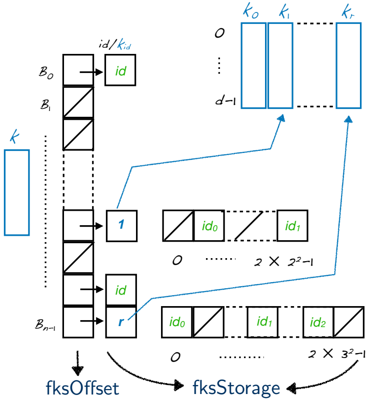
The problem

Given:	A d-dimensional sparse tensor $\mathcal{T}$
Goal:	Answer queries of the form: "Is $\mathcal{T}[i_1, \ldots, i_d]$ zero or nonzero?"
Motivation:	*Tensor decomposition*[§]: used for finding patterns in massive data

§ T. G. Kolda and D. Hong, "Stochastic gradients for large-scale tensor decomposition," SIAM Journal on Mathematics of Data Science 2(2020)

Hyperedge Queries in Hypergraphs

Our proposal: Space-efficient hashing-based method with worst-case $O(1)$ lookups



Hash data structure

- All nonzeros are available at the start
- All nonzeros have unique coordinates
- A two-level hash data structure
- First level hash function: $h(\mathbf{k}, \mathbf{x}, p, n)$
  - > Map nonzeros to buckets
- Second level hash function: $h(\mathbf{k}_i, \mathbf{x}, p, 2b_i^2)$
  - > Determine unique positions for all nonzeros in each bucket