Scalable Parallel Primitives for Massive Graph Computation

Aydın Buluç
University of California, Santa Barbara
Sources of Massive Graphs

Graphs naturally arise from the internet and social interactions

(WWW snapshot, courtesy Y. Hyun)

Many scientific (biological, chemical, cosmological, ecological, etc) datasets are modeled as graphs.

(Yeast protein interaction network, courtesy H. Jeong)
Types of Graph Computations

Examples:
- Centrality
- Shortest paths
- Network flows
- Strongly Connected Components

Tightly coupled

Filtering based

Fuzzy intersection

Examples:
- Clustering,
  Algebraic Multigrid

Examples:
- Loop and multi edge removal
- Triangle/Rectangle enumeration

Tool: Graph Traversal

Tool: Map/Reduce
Tightly Coupled Computations on Sparse Graphs

- Many graph mining algorithms are computationally intensive. (e.g. graph clustering, centrality)
- Some computations are inherently latency-bound. (e.g. finding shortest paths)
- Interesting graphs are sparse, typically $|edges| = O(|vertices|)$

Huge Graphs + Expensive Kernels $\rightarrow$ High Performance and Massive Parallelism

Sparse Graphs/Data $\rightarrow$ Sparse Data Structures (Matrices)
“...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do”
Software for Graph Computation

“...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do”

Dealing with software is hard!
“...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do”

Dealing with software is hard!

High performance computing (HPC) software is harder!
“…my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do”

Dealing with software is hard!

High performance computing (HPC) software is harder!

Deal with parallel HPC software?
Outline

- The Case for Primitives
- The Case for Sparse Matrices
- Parallel Sparse Matrix-Matrix Multiplication
- Software Design of the Combinatorial BLAS
- An Application in Social Network Analysis
- Other Work
- Future Directions
All-Pairs Shortest Paths

- **Input:** Directed graph with “costs” on edges
- Find least-cost paths between all reachable vertex pairs
- Classical algorithm: Floyd-Warshall

for \( k = 1 \) to \( n \)  // the induction sequence
  for \( i = 1 \) to \( n \)
    for \( j = 1 \) to \( n \)
      if \( w(i \rightarrow k) + w(k \rightarrow j) < w(i \rightarrow j) \)
        \( w(i \rightarrow j) := w(i \rightarrow k) + w(k \rightarrow j) \)

- Case study of implementation on multicore architecture:
  - graphics processing unit (GPU)
GPU characteristics

Powerful: two Nvidia 8800s > 1 TFLOPS
Inexpensive: $500 each

But:

• Difficult programming model:
  One instruction stream drives 8 arithmetic units
• Performance is counterintuitive and fragile:
  Memory access pattern has subtle effects on cost
• Extremely easy to underutilize the device:
  Doing it wrong easily costs 100x in time
Based on R-Kleene algorithm

Well suited for GPU architecture:
- Fast matrix-multiply kernel
- In-place computation => low memory bandwidth
- Few, large MatMul calls => low GPU dispatch overhead
- Recursion stack on host CPU, not on multicore GPU
- Careful tuning of GPU code

\[
\begin{align*}
A &= A^*; & \text{% recursive call} \\
B &= AB; & C = CA; \\
D &= D + CB; \\
D &= D^*; & \text{% recursive call} \\
B &= BD; & C = DC; \\
A &= A + BC; \\
\end{align*}
\]
The Case for Primitives

Lifting Floyd-Warshall to GPU

Conclusions:
High performance is achievable but not simple
Carefully chosen and optimized primitives will be key
The Case for Primitives

The Case for Sparse Matrices

Parallel Sparse Matrix-Matrix Multiplication

Software Design of the Combinatorial BLAS

An Application in Social Network Analysis

Other Work

Future Directions
Sparse Adjacency Matrix and Graph

- Every graph is a sparse matrix and vice-versa
- Adjacency matrix: sparse array w/ nonzerros for graph edges
- Storage-efficient implementation from sparse data structures
Many irregular applications contain sufficient coarse-grained parallelism that can ONLY be exploited using abstractions at proper level.

<table>
<thead>
<tr>
<th>Traditional graph computations</th>
<th>Graphs in the language of linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data driven. Unpredictable communication.</td>
<td>Fixed communication patterns.</td>
</tr>
<tr>
<td>Irregular and unstructured. Poor locality of reference</td>
<td>Operations on matrix blocks. Exploits memory hierarchy</td>
</tr>
<tr>
<td>Fine grained data accesses. Dominated by latency</td>
<td>Coarse grained parallelism. Bandwidth limited</td>
</tr>
</tbody>
</table>
Linear Algebraic Primitives

Sparse matrix-matrix Multiplication (SpGEMM)

Element-wise operations

Sparse matrix-vector multiplication

Sparse Matrix Indexing

Matrices on semirings, e.g. (\cdot, +), (\text{and}, \text{or}), (+, \text{min})
Applications of Sparse GEMM

- Graph clustering (Markov, peer pressure)
- Subgraph / submatrix indexing
- Shortest path calculations
- Betweenness centrality
- Graph contraction
- Cycle detection
- Multigrid interpolation & restriction
- Colored intersection searching
- Applying constraints in finite element computations
- Context-free parsing ...
Outline

- The Case for Primitives
- The Case for Sparse Matrices
- **Parallel Sparse Matrix-Matrix Multiplication**
- Software Design of the Combinatorial BLAS
- An Application in Social Network Analysis
- Other Work
- Future Directions
Two Versions of Sparse GEMM

1D block-column distribution

\[ C_i = C_i + A B_i \]

Checkerboard (2D block) distribution

\[ C_{ij} = A_{ik} B_{kj} \]
Projected performances of Sparse 1D & 2D

In practice, 2D algorithms have the potential to scale, if implemented correctly. Overlapping communication, and maintaining load balance are crucial.
Compressed Sparse Columns (CSC): A Standard Layout

- Stores entries in column-major order
- Dense collection of “sparse columns”
- Uses $O(n + nnz)$ storage.
Submatrices are “hypersparse” (i.e. \( nnz << n \))

\[
nnz' = \frac{c}{\sqrt{p}} \to 0
\]

Average of \( c \) nonzeros per column

Total Storage:
\[
O(n + nnz) \Rightarrow O(n\sqrt{p} + nnz)
\]

- A data structure or algorithm that depends on the matrix dimension \( n \) (e.g. CSR or CSC) is asymptotically too wasteful for submatrices
Sequential Hypersparse Kernel

Standard algorithm’s complexity:
\( \Theta( \text{flops} + \text{nnz}(B) + n + m ) \)

New hypersparse kernel:
\( \Theta( \text{flops} \cdot \lg ni + \text{nz}(A) + \text{nzr}(B) ) \)

- Strictly \( O(\text{nnz}) \) data structure
- Outer-product formulation
- Work-efficient
Scaling Results for SpGEMM

- RMat X RMat product (graphs with high variance on degrees)
- Random permutations useful for the overall computation.
- Bulk synchronous algorithms may still suffer due to imbalance within the stages.
- Asynchronous algorithm to avoid the curse of synchronicity
- One sided communication via RDMA (using MPI-2)
- Results obtained on TACC/Lonestar for graphs with average degree 8
Outline

- The Case for Primitives
- The Case for Sparse Matrices
- Parallel Sparse Matrix-Matrix Multiplication
- **Software Design of the Combinatorial BLAS**
- An Application in Social Network Analysis
- Other Work
- Future Directions
Generality, of the numeric type of matrix elements, algebraic operation performed, and the library interface.

Without the language abstraction penalty: C++ Templates

```c++
template <class IT, class NT, class DER>
class SpMat;
```

- Achieve mixed precision arithmetic: Type traits
- Enforcing interface and strong type checking: CRTP
- General semiring operation: Function Objects

- Abstraction penalty is not just a programming language issue.
- In particular, view matrices as indexed data structures and stay away from single element access (Interface should discourage)
Software design of the Combinatorial BLAS

**Extendability**, of the library while maintaining compatibility and seamless upgrades.

⇒ Decouple parallel logic from the sequential part.

Commonalities:
- Support the sequential API
- Composed of a number of arrays

Any parallel logic: asynchronous, bulk synchronous, etc

```
SpPar<Comm, SpSeq>
```

```
CSC  DCSC  Tuples
```

```
SpSeq
```

```
SpSeq
```
Outline

- The Case for Primitives
- The Case for Sparse Matrices
- Parallel Sparse Matrix-Matrix Multiplication
- Software Design of the Combinatorial BLAS
- An Application in Social Network Analysis
- Other Work
- Future Directions
Social Network Analysis

Applications and Algorithms

**Betweenness Centrality (BC)**

$C_B(v)$: Among all the shortest paths, what fraction of them pass through the node of interest?

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

Brandes’ algorithm

A typical software stack for an application enabled with the Combinatorial BLAS
Betweenness Centrality using Sparse GEMM

\[ A^T \times X \xrightarrow{\text{}} (A^T X)_{*\leftarrow X} \]

- Parallel breadth-first search is implemented with sparse matrix-matrix multiplication
- Work efficient algorithm for BC
BC Performance on Distributed-memory

Input: RMAT scale N
2^N vertices
Average degree 8

• TEPS: Traversed Edges Per Second
• Batch of 512 vertices at each iteration
• Code only a few lines longer than Matlab version

Pure MPI-1 version.
No reliance on any particular hardware.
Outline

- The Case for Primitives
- The Case for Sparse Matrices
- Parallel Sparse Matrix-Matrix Multiplication
- Software Design of the Combinatorial BLAS
- An Application in Social Network Analysis
- Other Work
- Future Directions
SpMV on Multicore

Our parallel algorithms for $y \leftarrow Ax$ and $y' \leftarrow A^T x'$ using the new **compressed sparse blocks (CSB)** layout have:

- $\Theta(\sqrt{n} \lg n)$ span, and $\Theta(n_{nz})$ work,
- yielding $\Theta(n_{nz}/\sqrt{n} \lg n)$ parallelism.

![Graph showing performance of different algorithms](image.png)
Outline

- The Case for Primitives
- The Case for Sparse Matrices
- Parallel Sparse Matrix-Matrix Multiplication
- Software Design of the Combinatorial BLAS
- An Application in Social Network Analysis
- Other Work
- **Future Directions**
Future Directions

- Novel scalable algorithms
- Static graphs are just the beginning. Dynamic graphs, Hypergraphs, Tensors
- Architectures (mainly nodes) are evolving
  Heterogeneous multicores
  Homogenous multicores with more cores per node

Hierarchical parallelism

TACC Lonestar (2006) 4 cores / node
TACC Ranger (2008) 16 cores / node
SDSC Triton (2009) 32 cores / node
XYZ Resource (2020)
New Architectural Trends

LANL / IBM Roadrunner

NVIDIA Tesla

A unified architecture?

Cray XMT

Intel 80-core chip
Remarks

- Graph computations are pervasive in sciences and will become more so.
- High performance software libraries improve productivity.
- Carefully chosen and implemented primitive operations are key to performance.
- Linear algebraic primitives:
  - General enough to be widely useful
  - Compact enough to be implemented in a reasonable time.
Related Publications

- Hypersparsity in 2D decomposition, sequential kernel.
  B., Gilbert, "On the Representation and Multiplication of Hypersparse Matrices“, IPDPS’08

- Parallel analysis of sparse GEMM, synchronous implementation
  B., Gilbert, "Challenges and Advances in Parallel Sparse Matrix-Matrix Multiplication, ICPP’08

- The case for primitives, APSP on the GPU

- SpMV on Multicores
  B., Fineman, Frigo, Gilbert, Leiserson, "Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication using Compressed Sparse Blocks“, SPAA’09

- Betweenness centrality results
  B., Gilbert, “Parallel Sparse Matrix-Matrix Multiplication and Large Scale Applications”

- Software design of the library
  B., Gilbert, “Parallel Combinatorial BLAS: Interface and Reference Implementation“
Acknowledgments…