Linear Algebraic Primitives for Parallel Computing on Large Graphs

Aydın Buluç
Ph.D. Defense
March 16, 2010

Committee:
John R. Gilbert (Chair)
Ömer Eğecioğlu
Fred Chong
Shivkumar Chandrasekaran
Sources of Massive Graphs

Graphs naturally arise from the internet and social interactions

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

(WWW snapshot, courtesy Y. Hyun)  (Yeast protein interaction network, courtesy H. Jeong)
Types of Graph Computations

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

Fuzzy intersection
Examples: Clustering,
Algebraic Multigrid

Tightly coupled

Filtering based

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

Tool: Graph Traversal

Tool: Map/Reduce

Diagram showing the categorization of types of graph computations and their examples.
Tightly Coupled Computations

A. Computationally intensive graph/data mining algorithms.
   (e.g. graph clustering, centrality, dimensionality reduction)
B. Inherently latency-bound computations.
   (e.g. finding s-t connectivity)

Let the special purpose architectures handle (B),
but what can we do for (A) with the commodity architectures?

<table>
<thead>
<tr>
<th>Huge Graphs</th>
<th>Expensive Kernels</th>
<th>High Performance and Massive Parallelism</th>
</tr>
</thead>
</table>

We need memory !  Disk is not an option !
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”
“...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”
"...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?
Parallel Graph Software

<table>
<thead>
<tr>
<th>Packet</th>
<th>Parallelism</th>
<th>Abstraction</th>
<th>Offering</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBGL</td>
<td>Distributed</td>
<td>Visitor</td>
<td>Algorithms</td>
<td>Limited</td>
</tr>
<tr>
<td>GAPDT</td>
<td>Distributed</td>
<td>Sparse Matrix</td>
<td>Both</td>
<td>Limited</td>
</tr>
<tr>
<td>MTGL</td>
<td>Shared</td>
<td>Visitor</td>
<td>Algorithms</td>
<td>Unknown</td>
</tr>
<tr>
<td>SNAP</td>
<td>Shared</td>
<td>Various</td>
<td>Both</td>
<td>Good</td>
</tr>
<tr>
<td>Combinatorial BLAS</td>
<td>Distributed</td>
<td>Sparse Matrix</td>
<td>Kernels</td>
<td>Good</td>
</tr>
</tbody>
</table>

- Shared memory scales, not it pays off to target productivity
- Distributed memory should be targeting \( p > 100 \) at least
- Shared and distributed memory complement each other.
- **Coarse-grained parallelism** for distributed memory.
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

```plaintext
for k=1:n  // the induction sequence
    for i = 1:n
        for j = 1:n
            if(w(i→k)+w(k→j) < w(i→j))
                w(i→j) := w(i→k) + w(k→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
Recursive All-Pairs Shortest Paths

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

+ is “min”, × is “add”

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

\[
A = A^*; \quad \% \text{recursive call} \\
B = AB; \quad C = CA; \\
D = D + CB; \\
D = D^*; \quad \% \text{recursive call} \\
B = BD; \quad C = DC; \\
A = A + BC;
\]
Conclusions:

High performance is achievable but not simple

Carefully chosen and optimized **primitives** will be key
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
Sparse Adjacency Matrix and Graph

- Every graph is a sparse matrix and vice-versa
- Adjacency matrix: sparse array w/ nonzeros 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.</td>
<td>Fixed communication patterns.</td>
</tr>
<tr>
<td>Unpredictable communication.</td>
<td></td>
</tr>
<tr>
<td>Irregular and unstructured.</td>
<td>Operations on matrix blocks.</td>
</tr>
<tr>
<td>Poor locality of reference</td>
<td>Exploits memory hierarchy</td>
</tr>
<tr>
<td>Fine grained data accesses.</td>
<td>Coarse grained parallelism.</td>
</tr>
<tr>
<td>Dominated by latency</td>
<td>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. (·, +), (and, or), (+, min)
Applications of Sparse GEMM

- Graph clustering (MCL, 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 ...

Loop until convergence
\[
A = A \times A; \\
A = A .\times 2; \\
sc = 1./\text{sum}(A, 1); \\
A = A \times \text{diag}(sc); \\
A(A < 0.0001) = 0;
\]
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, but not linearly

\[ E = \frac{W}{p(T_{\text{comp}} + T_{\text{comm}})} = \frac{\gamma c^2 n}{(\gamma + \beta)cn\sqrt{p} + \gamma c^2 n\lg\left(\frac{c^2 n}{p}\right) + \alpha p\sqrt{p}} \]
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:
\( \Theta( \text{flops} \cdot \lg ni + \text{nnz}(A) + \text{nzr}(B) ) \)

- 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 (<10% imbalance).
- Bulk synchronous algorithms may still suffer due to imbalance within the stages. *But this goes away as matrices get larger*
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

```cpp
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>

Each *may* support an iterator concept
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

Betweenness Centrality (BC)

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

Brandes’ algorithm

A typical software stack for an application enabled with the Combinatorial BLAS

Applications
- Community Detection
- Network Vulnerability Analysis

Combinatorial Algorithms
- Betweenness Centrality
- Graph Clustering
- Contraction

Parallel Combinatorial BLAS
- SpGEMM
- SpRef/SpAsgn
- SpMV
- SpAdd
Betweenness Centrality using Sparse GEMM

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

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

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

Pure MPI-1 version.
No reliance on any particular hardware.
More scaling and sensitivity (Betweenness Centrality)

Sensitivity to batch processing

Scaling beyond hundreds

Input is an RMat graph of scale 22

Experiment with 512 batch nodes
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
Our parallel algorithms for \( y \leftarrow Ax \) and \( y' \leftarrow A^\top x' \) using the new **compressed sparse blocks (CSB)** layout have

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

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

Hierarchical parallelism
So long, grad school…

› What did I learn about parallel graph computations?

› No silver bullets.
  › Graph computations are not homogenous.

› No free lunch.
  › If communication >> computation for your algorithm, overlapping them will NOT make it scale.
  › Scale your algorithm, not your implementation.

› Any sustainable speedup is a success!
  › Don’t get disappointed with poor parallel efficiency.
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…