Faster and More Scalable Sparse Matrix-Matrix Multiplication

Aydın Buluç
Lawrence Berkeley National Laboratory

In collaboration with
Grey Ballard (Sandia), Ariful Azad (LBNL), James Demmel (UC Berkeley),
John Gilbert (UCSB), Laura Grigori (INRIA), Oded Schwartz (Hebrew University),
Sivan Toledo (Tel Aviv University), Samuel Williams (LBNL)

SIAM PP 2016, Paris
Sparse Matrix-Matrix Multiplication (SpGEMM)

A, B and C are sparse.

Why sparse matrix-matrix multiplication?
- Algebraic multigrid (AMG),
- Graph clustering,
- Betweenness centrality,
- Graph contraction,
- Subgraph extraction
- Quantum chemistry
- Triangle counting/enumeration
Outline of the talk

- **3D SpGEMM** [A. Azad et al. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. arXiv preprint arXiv:1510.00844. ]
  - Scalable shared-memory SpGEMM
  - Distributed-memory 3D SpGEMM

- **Parallel triangle counting and enumeration using SpGEMM** [A. Azad, A. Buluc, J. Gilbert. Parallel Triangle Counting and Enumeration using Matrix Algebra. IPDPS Workshops, 2015]
  - Masked SpGEMM to reduce communication
  - 1D parallel implementation
Shared-Memory SpGEMM

• **Heap-based column-by-column algorithm** [Buluc and Gilbert, 2008]
• Easy to parallelize in shared-memory via **multithreading**
• Memory efficient and scalable (i.e. temporary per-thread storage is asymptotically negligible)
Performance of Shared-Memory SpGEMM

Faster than Intel’s MKL (mkl_csrmmultcsr) (when we keep output sorted by indices)

(a) cage12 x cage12

(b) Scale 16 G500 x G500

Matrix multiplication: \( \forall (i,j) \in n \times n, \quad C(i,j) = \sum_k A(i,k)B(k,j), \)

**Sparsity independent algorithms:**
assigning grid-points to processors is independent of sparsity structure.
[Ballard et al., SPAA 2013]
2D Algorithm: Sparse SUMMA

$\sqrt{p \times p}$ Processor Grid

$C_{ij} += \text{LocalSpGEMM}(A_{\text{recv}}, B_{\text{recv}})$

- 2D Sparse SUMMA (Scalable Universal Matrix Multiplication Algorithm) was the previous state of the art. It becomes communication bound on high concurrency [Buluc & Gilbert, 2012]
3D Parallel SpGEMM in a Nutshell

$\sqrt{\frac{p}{c}} \times \sqrt{\frac{p}{c}} \times c$ Processor Grid

Input storage does not increase

$C_{ijk}^{\text{int}} = \sum_{l=1}^{p/c} A_{ilk} B_{ljk}$

3D Parallel SpGEMM in a Nutshell

Communication

Broadcast (row/column)

Computation (multithreaded)

AlltoAll (layer)

Local multiply Multiway merge

Multiway merge

\[
n/\sqrt{pc}
\]

\[
A::3 \times x = C_{\text{intermediate}}
\]

\[
A::2 \times x = C_{\text{intermediate}}
\]

\[
A::1 \times x = C_{\text{intermediate}}
\]

\[
C_{ij} = \sum_{l=1}^{p/c} A_{ilk} B_{ljk}
\]

\[
C_{\text{final}}
\]
# Communication Cost

### Communication

- **Broadcast (row/column)**
  - #broadcasts targeted to each process (synchronization points / SUMMA stages): $\sqrt{p/c}$
  - #processes participating in each broadcasts (communicator size): $\sqrt{p/c}$
  - Total data received in all broadcasts (process): $\frac{nnz}{\sqrt{pc}}$

### Broadcast (row/column)

**Most expensive step**

- Threading decreases
- Network card contention

### Diagram

- A::3
- A::2
- A::1

**A** $\times$ **B** = **C**

- **C** intermediate
- **C** final

**Communication**

**Broadcast** (row/column)

**Most expensive step**

- Threading decreases
- Network card contention

- Processor Grid

- $\sqrt{p/c} \times \sqrt{p/c} \times c$
On 8,192 cores of Titan when multiplying two scale 26 G500 matrices.

How do 3D algorithms gain performance?

For fixed #layers (c) 
Increased #threads (t) reduces runtime
How do 3D algorithms gain performance?

On 8,192 cores of Titan when multiplying two scale 26 G500 matrices.

For fixed #threads (t)
Increased #layers (c)
reduces runtime
3D SpGEMM performance (matrix squaring)

Squaring nlpkkt160 on Edison: 1.2 billion nonzeros in the $A^2$

2D (non-threaded) is the previous state-of-the-art

3D (threaded) – first presented here – beats it by 8X at large concurrencies
3D SpGEMM performance (matrix squaring)

3D SpGEMM with c=16, t=6 on Edison

It-2004 (web crawl)

nnz(A) = 1.1 billion
nnz(A^2) = 14 billion

Number of Cores

Time (sec)

18x

14x
3D SpGEMM performance (AMG coarsening)

- Galerkin triple product in Algebraic Multigrid (AMG)
- \( A_{\text{coarse}} = R^T A_{\text{fine}} R \) where \( R \) is the restriction matrix
- \( R \) constructed via distance-2 maximal independent set (MIS)

Only showing the first product \((R^T A)\)

3D is 16x faster than 2D
3D SpGEMM performance (AMG coarsening)

Comparing performance with EpetraExt package of Trilinos

AR computation with nlpkkt160 on Edison

Notes:
- EpetraExt runs up to 3x faster when computing AR, 3D is less sensitive
- 1D decomposition used by EpetraExt performs better on matrices with good separators.
Application of SpGEMM
Triangle Counting/Enumeration

[A.Azad, A. Buluc, J. Gilbert. Parallel Triangle Counting and Enumeration using Matrix Algebra. IPDPS Workshops, 2015]
Counting Triangles

**Wedge:** a path of length two

A triangle has three wedges

**Clustering coefficient:**

- Pr (wedge i-j-k makes a triangle with edge i-k)
- $3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}}$
- $3 \times \frac{2}{13} = 0.46$ in example
- may want to compute for each vertex j
Triangle Counting in Matrix Algebra

Step 1: Count wedges from a pair of vertices \((u,v)\) by finding a path of length 2. Avoid repetition by counting wedges with low index of the middle vertex.

2 wedges between vertex 5 and 6

Adjacency matrix

\[ A = L + U \quad (hi\rightarrow lo \ + \ lo\rightarrow hi) \]

\[ L \times U = B \quad (wedges) \]
Triangle Counting in Matrix Algebra

**Step 2:** Count wedges as triangles if there is an edge between the selected pair of vertices.

\[
L \times U = B \quad \text{(wedges)} \quad A \land B = C \quad \text{(closed wedge)}
\]

**Goal:** Can we design a parallel algorithm that communicates less data exploiting this observation?
In this example, nonzeros of $L(5,:)$ (marked with red color) are not needed by $P_1$ because $A_1(5,:)$ does not contain any nonzeros. So, we can mask rows of $L$ by $A$ to avoid communication.
Benefits and Overheads of Masked SpGEMM

- **Benefit**
  - Reduce communication by a factor of \( \frac{p}{d} \) where \( d \) is the average degree (good for large \( p \)).

- **Overheads**
  - Increased index traffic for requesting subset of rows/columns.
    - Partial remedy: Compress index requests using bloom filters.
  - Indexing and packaging the requested rows (SpRef).
    - Remedy (ongoing work): data structures/algorithms for faster SpRef.
Where do algorithms spend time with increased concurrency?

As expected, Masked SpGEMM reduces cost to communicate $L$

Expensive indexing undermines the advantage More computation in exchange of communication. [Easy to fix]
Strong Scaling of Triangle Counting

- Decent scaling to modest number of cores (p=512 in this case)
- Further scaling requires 2D/3D SpGEMM (ongoing/future work)
What about triangle enumeration?

Data access patterns stay intact, only the scalar operations change!

B(i,k) now captures the set of wedges (indexed solely by their middle vertex because i and k are already known implicitly), as opposed to just the count of wedges.
Future Work

- Develop data structures/algorithms for faster SpRef
- Reduce the cost of communicating indices by a factor of $t$ (the number of threads) using **in-node multithreading**
- Use **3D decomposition/algorithms**: The requested index vectors would be of shorter length.
- Implement **triangle enumeration** via good serialization support. Larger performance gains are expected using masked-SpGEMM
Acknowledgements

Grey Ballard (Sandia), Ariful Azad (LBNL),
James Demmel (UC Berkeley),
John Gilbert (UCSB), Laura Grigori (INRIA),
Oded Schwartz (Hebrew University),
Sivan Toledo(Tel Aviv University), Samuel Williams(LBNL)

Work is funded by
Thanks for your attention
Supporting slides
How do dense and sparse GEMM compare?

<table>
<thead>
<tr>
<th>Dense:</th>
<th>Sparse:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower bounds match algorithms.</td>
<td>Significant gap</td>
</tr>
<tr>
<td>Allows extensive data reuse</td>
<td>Inherent poor reuse?</td>
</tr>
</tbody>
</table>
Communication Lower Bound

Lower bound for Erdős-Rényi($n,d$) [Ballard et al., SPAA 2013]:

$$\Omega\left(\min\left\{ \frac{dn}{\sqrt{P}}, \frac{d^2n}{P} \right\}\right)$$

(Under some technical assumptions)

- Few algorithms achieve this bound
- **1D algorithms** do not scale well on high concurrency
- **2D Sparse SUMMA** (Scalable Universal Matrix Multiplication Algorithm) was the previous state of the art. But, it still becomes communication bound on several thousands of processors. [Buluc & Gilbert, 2013]
- **3D algorithms** avoid communications
What dominates the runtime of the 3D algorithm?

Broadcast dominates on all concurrency

Squaring nlpkkt160 on Edison
How do 3D algorithms gain performance?

On 8,192 cores of Titan when multiplying two scale 26 G500 matrices.
3D SpGEMM performance (AMG coarsening)

- Galerkin triple product in Algebraic Multigrid (AMG)
- $A_{\text{coarse}} = R^T A_{\text{fine}} R$ where $R$ is the restriction matrix
- $R$ constructed via distance-2 maximal independent set (MIS)

![Graph showing $R^T A$ with NaluR3 (on Edison)]

Limited scalability of 3D in computing $R^T A$

<table>
<thead>
<tr>
<th>NaluR3</th>
<th>nnz($A$)</th>
<th>474 million</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nnz($A^2$)</td>
<td>2.1 billion</td>
</tr>
<tr>
<td></td>
<td>nnz($R^T A$)</td>
<td>77 million</td>
</tr>
</tbody>
</table>

3D is 16x faster than 2D

Only showing the first product ($R^T A$)

Not enough work on high concurrency
Communication Volume per Processor for Receiving $L$

- Assume $Erdős$-$Rényi(n,d)$ graphs and $d < p$
- Number of rows of $L$ needed by a processor: $\frac{nd}{p}$
- Number of columns of $L$ needed by a processor: $\frac{nd}{p}$
- Data received per processor: $O(\frac{nd^3}{p^2})$

- If no output masking is used ("improved 1D")
  - $O(\frac{nd^2}{p})$ [Ballard, Buluc, et al. SPAA 2013]

- Reduction of a factor of $\frac{p}{d}$ over "improved 1D"
  - Good for large $p$
## Experimental Evaluation

### Higher nnz(B) / nnz(A.*B) ratio => more potential communication reduction due to masked-spgemm for triangle counting

Ranges between 1.7 and 500

### Higher Triads / Triangles ratio => more potential communication reduction due to masked-spgemm for triangle enumeration

Ranges between 5 and 1000

<table>
<thead>
<tr>
<th>Graph</th>
<th>nnz(A)</th>
<th>(nnz(B = L \cdot U))</th>
<th>(nnz(A \cdot * B))</th>
<th>Triads</th>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>mouse_gene</td>
<td>28,922,190</td>
<td>138,568,561</td>
<td>26,959,674</td>
<td>25,808,000,000</td>
<td>3,619,100,000</td>
</tr>
<tr>
<td>coPaperDBLP</td>
<td>30,491,458</td>
<td>48,778,658</td>
<td>28,719,580</td>
<td>2,030,500,000</td>
<td>444,095,058</td>
</tr>
<tr>
<td>soc-LiveJournal1</td>
<td>85,702,474</td>
<td>480,215,359</td>
<td>40,229,140</td>
<td>7,269,500,000</td>
<td>285,730,264</td>
</tr>
<tr>
<td>wb-edu</td>
<td>92,472,210</td>
<td>57,151,007</td>
<td>32,441,494</td>
<td>12,204,000,000</td>
<td>254,718,147</td>
</tr>
<tr>
<td>cage15</td>
<td>94,044,692</td>
<td>405,169,608</td>
<td>48,315,646</td>
<td>895,671,340</td>
<td>36,106,416</td>
</tr>
<tr>
<td>europe_osm</td>
<td>108,109,320</td>
<td>57,983,978</td>
<td>121,816</td>
<td>66,584,124</td>
<td>61,710</td>
</tr>
<tr>
<td>hollywood-2009</td>
<td>112,751,422</td>
<td>1,010,100,000</td>
<td>104,151,320</td>
<td>47,645,000,000</td>
<td>4,916,400,000</td>
</tr>
</tbody>
</table>
Serial Complexity

Model: \textit{Erdős-Rényi}(n,d) graphs aka \(G(n, p=d/n)\)

Observation: The multiplication \(B = L \cdot U\) costs \(O(d^2 n)\) operations. However, the ultimate matrix \(C\) has at most \(2dn\) nonzeros.

Goal: Can we design a parallel algorithm that communicates less data exploiting this observation?
Where do algorithms spend time?

80% time spent in communicating L And local multiplication. (increased communication)

70% time spent in communicating index requests and SpRef (expensive indexing)

On 512 cores of NERSC/Edison