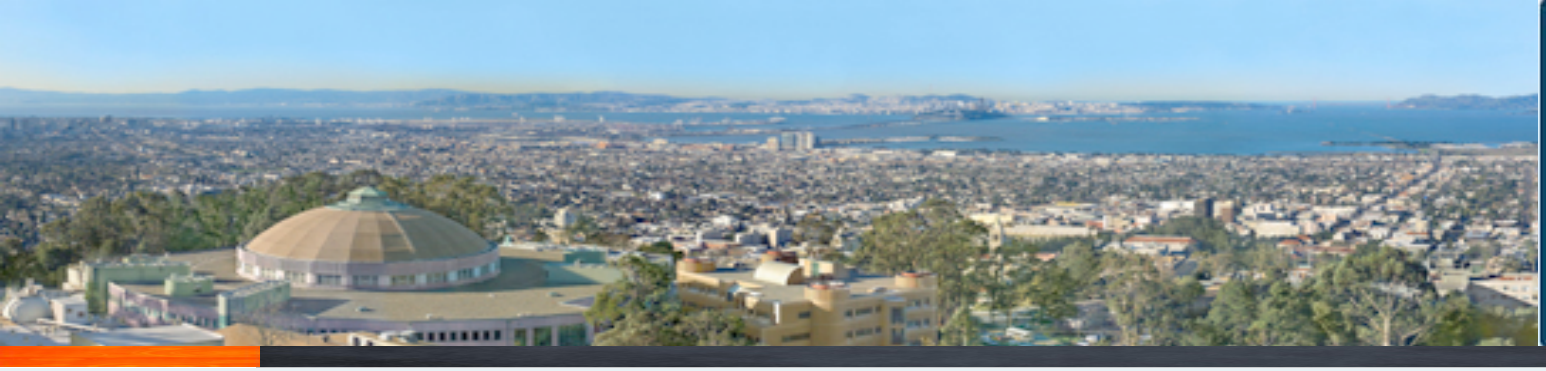


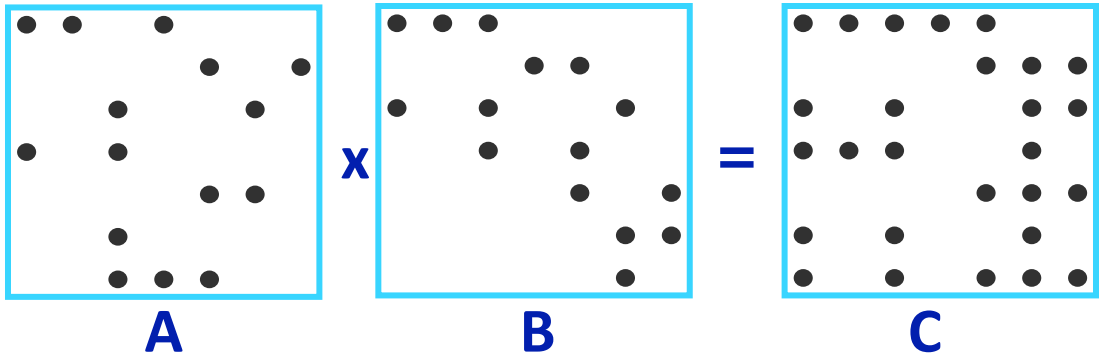
Parallel Sparse Matrix-Matrix Multiplication and Its Use in Triangle Counting and Enumeration

Ariful Azad
Lawrence Berkeley National Laboratory

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

SIAM ALA 2015, Atlanta

Sparse Matrix-Matrix Multiplication (**SpGEMM**)



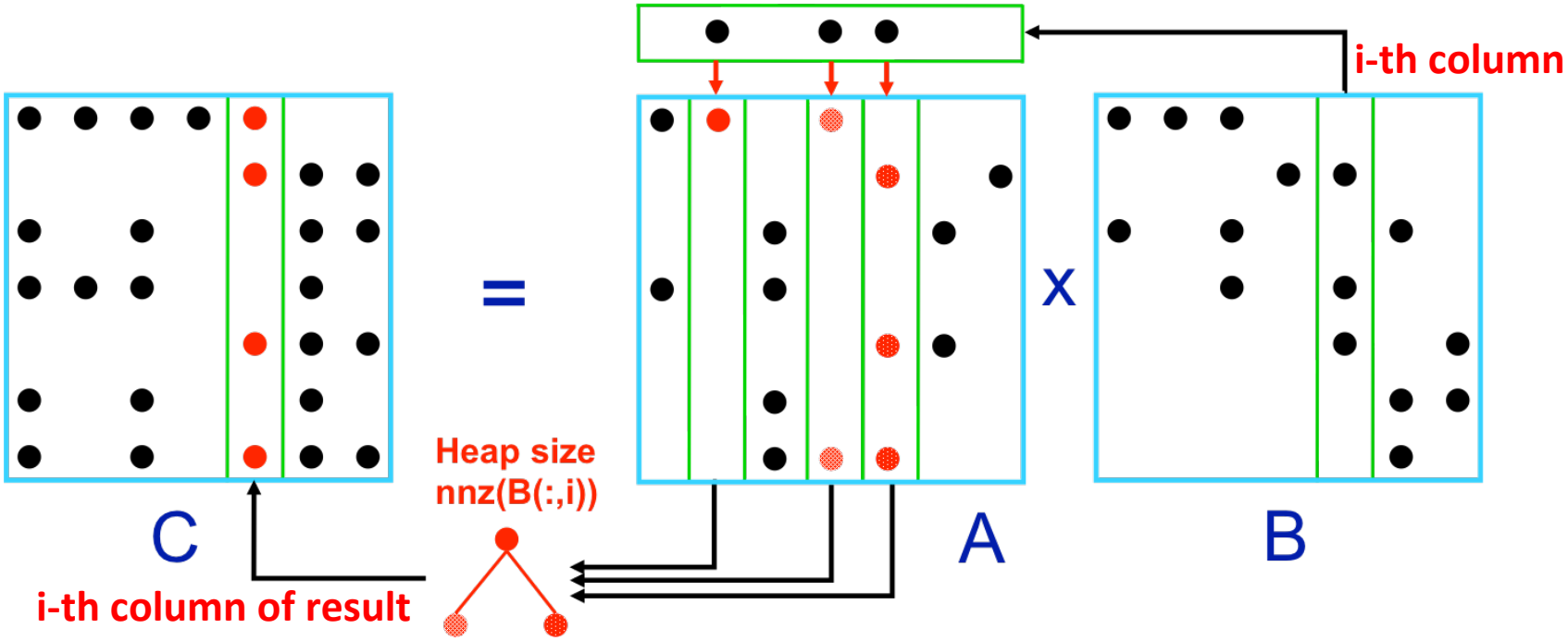
- A, B and C are sparse.
- Why sparse matrix-matrix multiplication?
  - Algebraic multigrid (AMG),
  - Graph clustering,
  - Betweenness centrality,
  - Graph contraction,
  - Subgraph extraction
  - Quantum chemistry
  - Triangle counting/enumeration

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

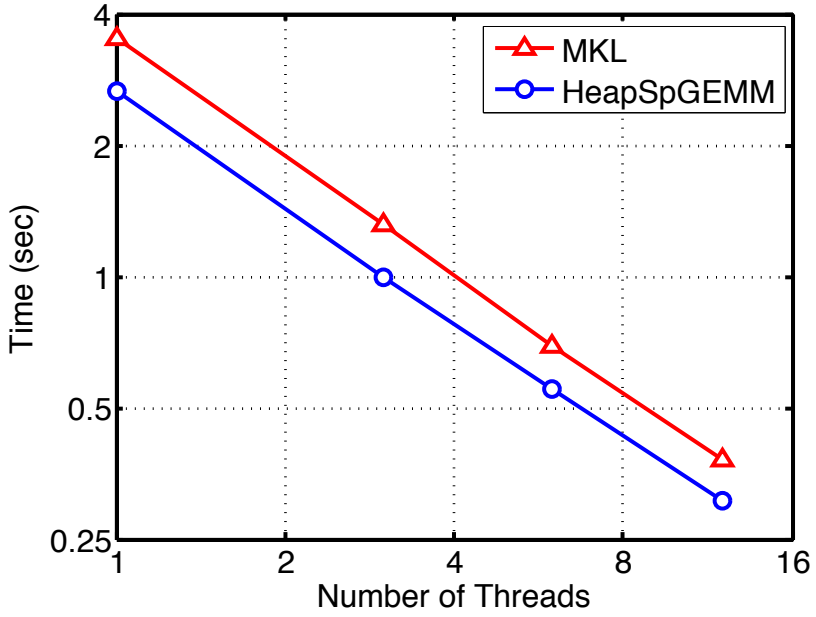
Shared-Memory SpGEMM



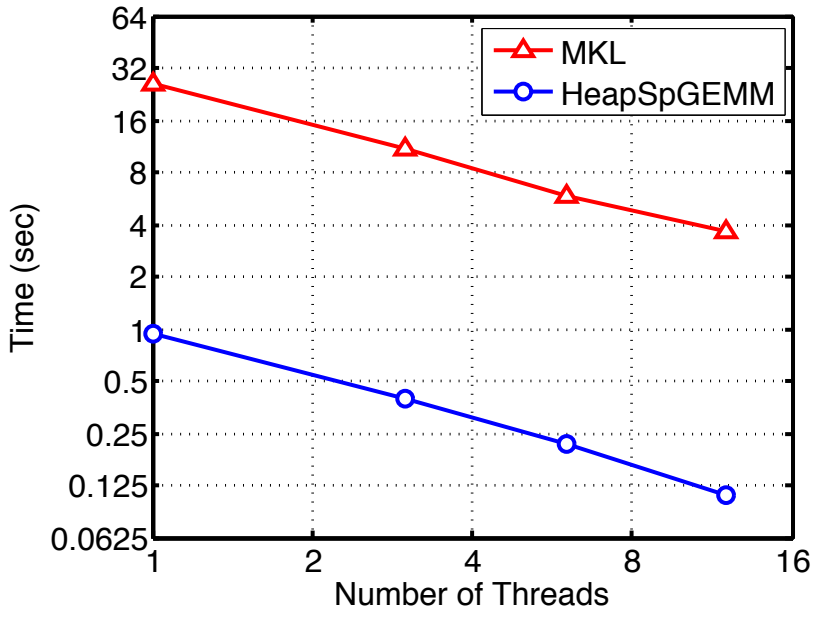
- Heap-based column-by-column algorithm [Buluc and Gilbert, 2008]
- Easy to parallelize in shared-memory via multithreading
- Memory efficient and scalable (i.e. temporary per-thread storage is asymptotically negligible)

Performance of Shared-Memory SpGEMM

Faster than Intel's MKL (mkl_csrmultcsr)
(when we keep output sorted by indices)



(a) cage12 x cage12

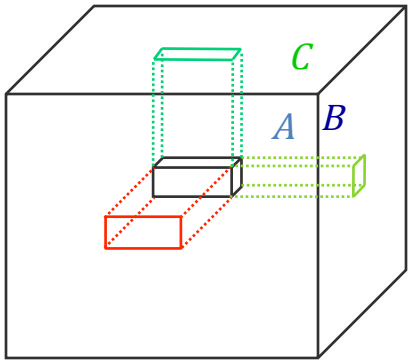


(b) Scale 16 G500 x G500

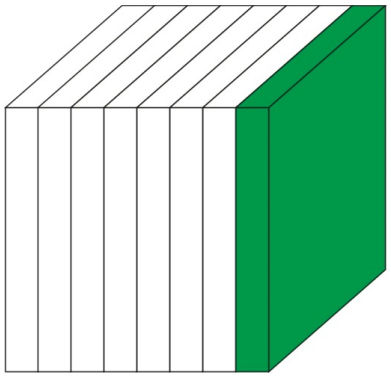
A. Azad, G. Ballard, A. Buluc, J. Demmel, L. Grigori, O. Schwartz, S. Toledo, S. Williams. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. arXiv preprint arXiv:1510.00844.

Variants of Distributed-Memory SpGEMM

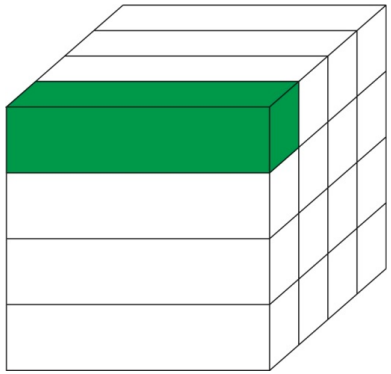
Matrix multiplication: $\forall (i,j) \in n \times n, \quad C(i,j) = \Sigma_k A(i,k)B(k,j),$



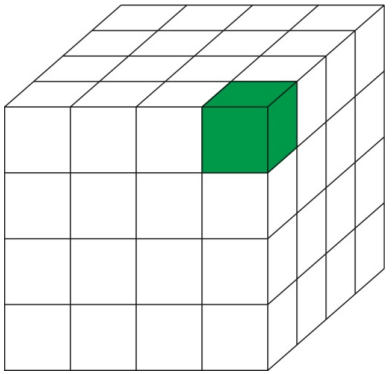
Sparsity independent algorithms:
assigning grid-points to processors is independent of sparsity structure.
[Ballard et al., SPAA 2013]



1D algorithms
Communicate A or B



2D algorithms
Communicate A and B

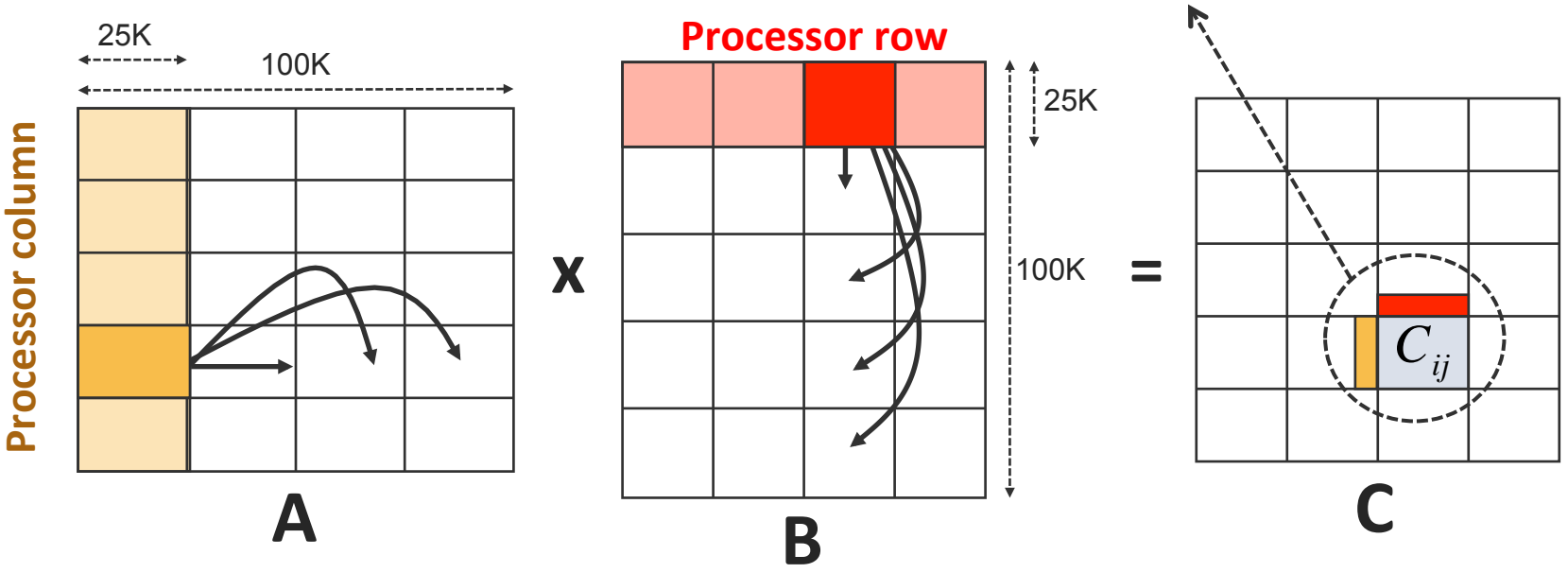


3D algorithms
Communicate A, B and C

2D Algorithm: Sparse SUMMA

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

C_{ij} += LocalSpGEMM(A^{recv}, B^{recv})

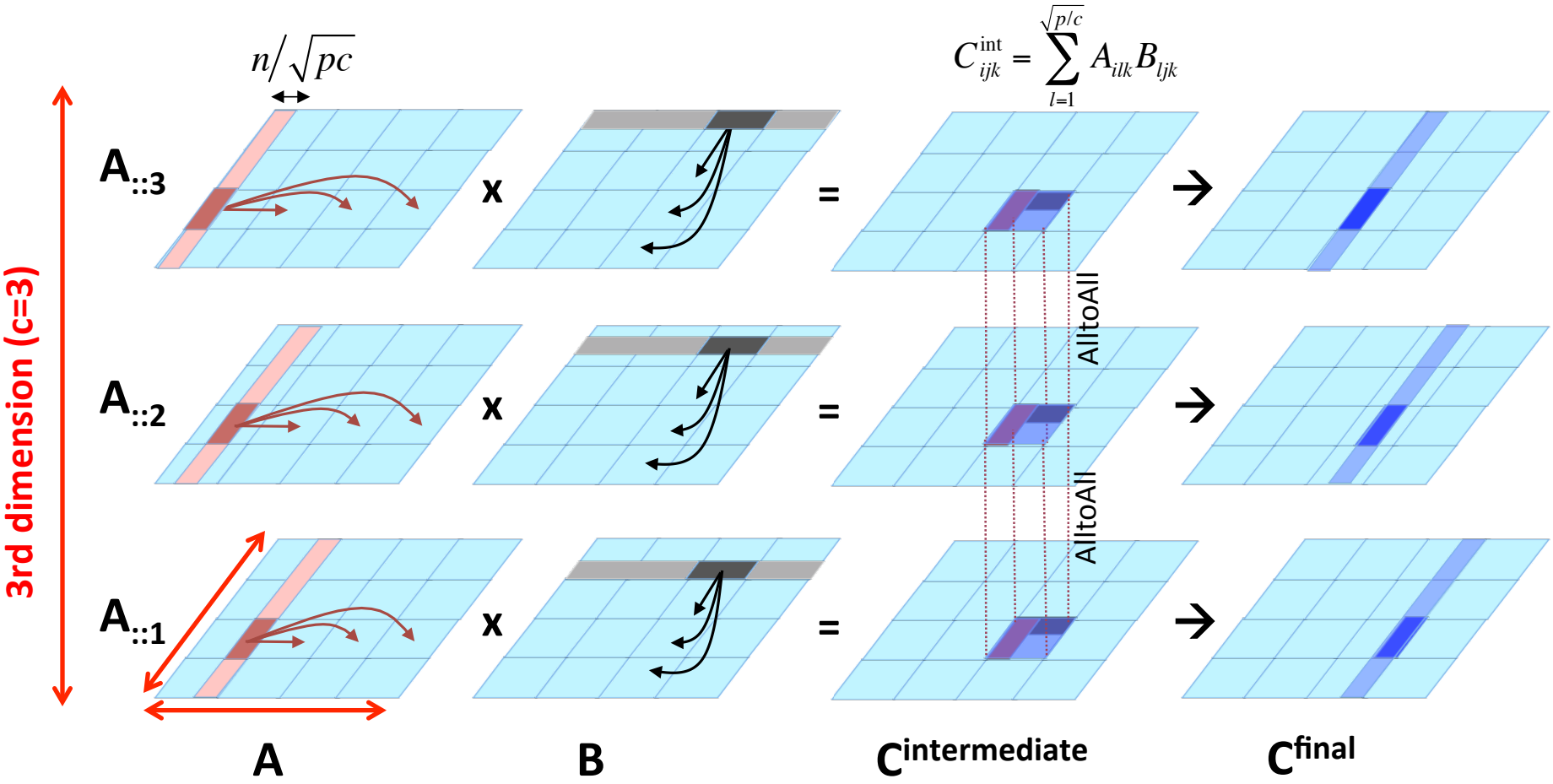


- 2D Sparse SUMMA (Scalable Universal Matrix Multiplication Algorithm) was the previous state of the art. It becomes communication bound on high concurrency [Buluc & Gilbert, 2012]

3D Parallel SpGEMM in a Nutshell

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

Input storage does not increase



A. Azad, G. Ballard, A. Buluc, J. Demmel, L. Grigori, O. Schwartz, S. Toledo, S. Williams. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. arXiv preprint arXiv:1510.00844.

3D Parallel SpGEMM in a Nutshell

Communication

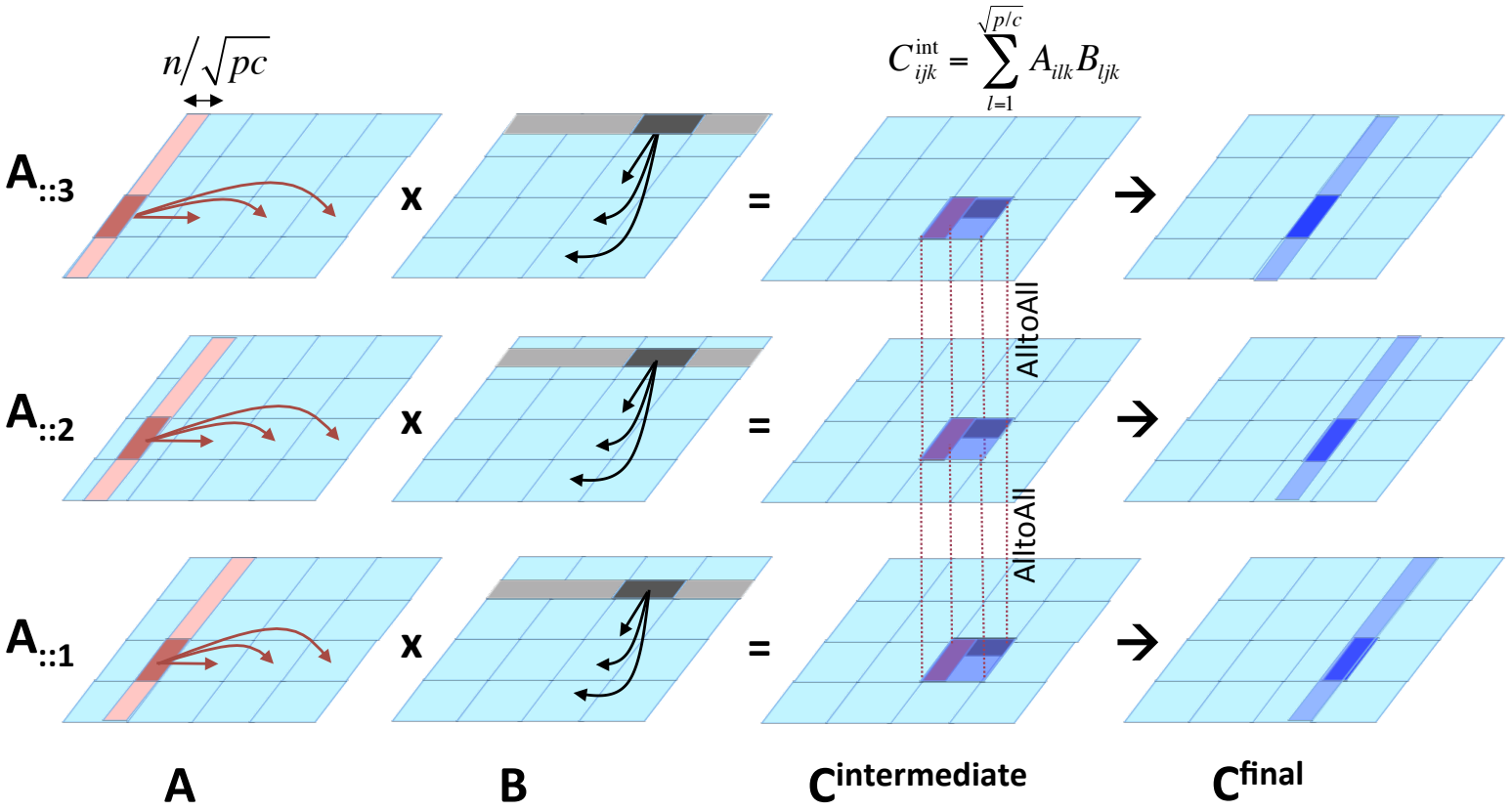
Broadcast (row/column)

AlltoAll (layer)

Computation (multithreaded)

Local multiply Multiway merge

Multiway merge



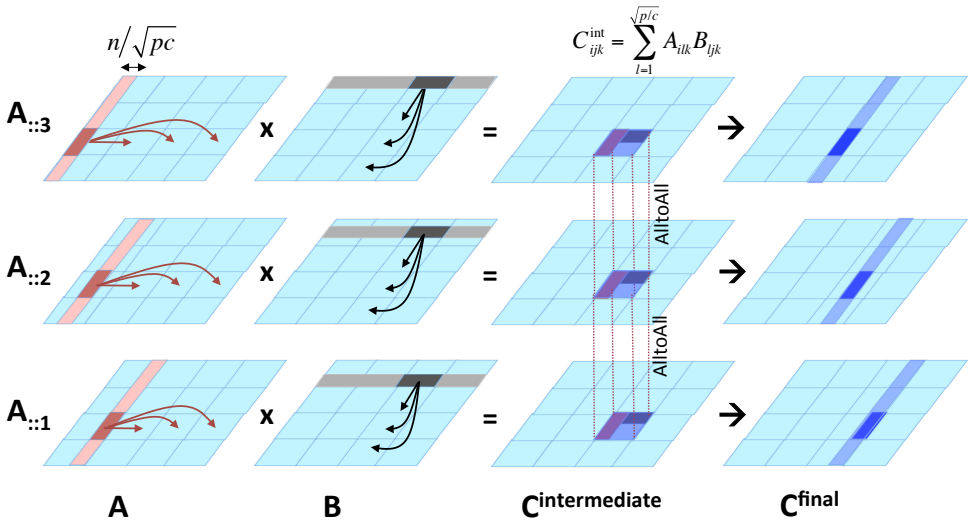
Communication Cost

Communication **Broadcast (row/column)** Most expensive step

#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)	$nnz/\sqrt{pc}$

layers

threads



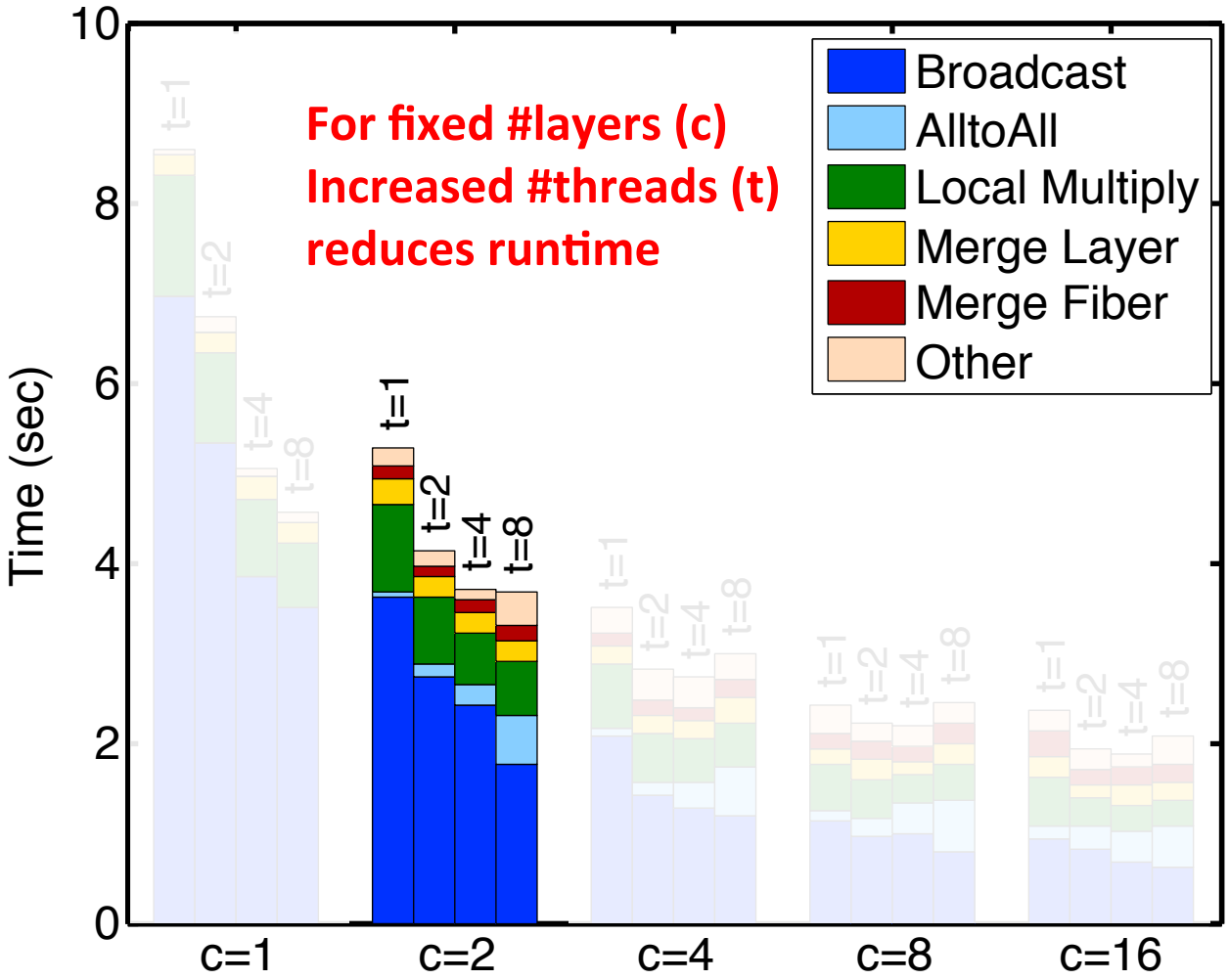
Threading decreases Network card contention

$$\sqrt{p/c} \times \sqrt{p/c} \times c$$

Processor Grid

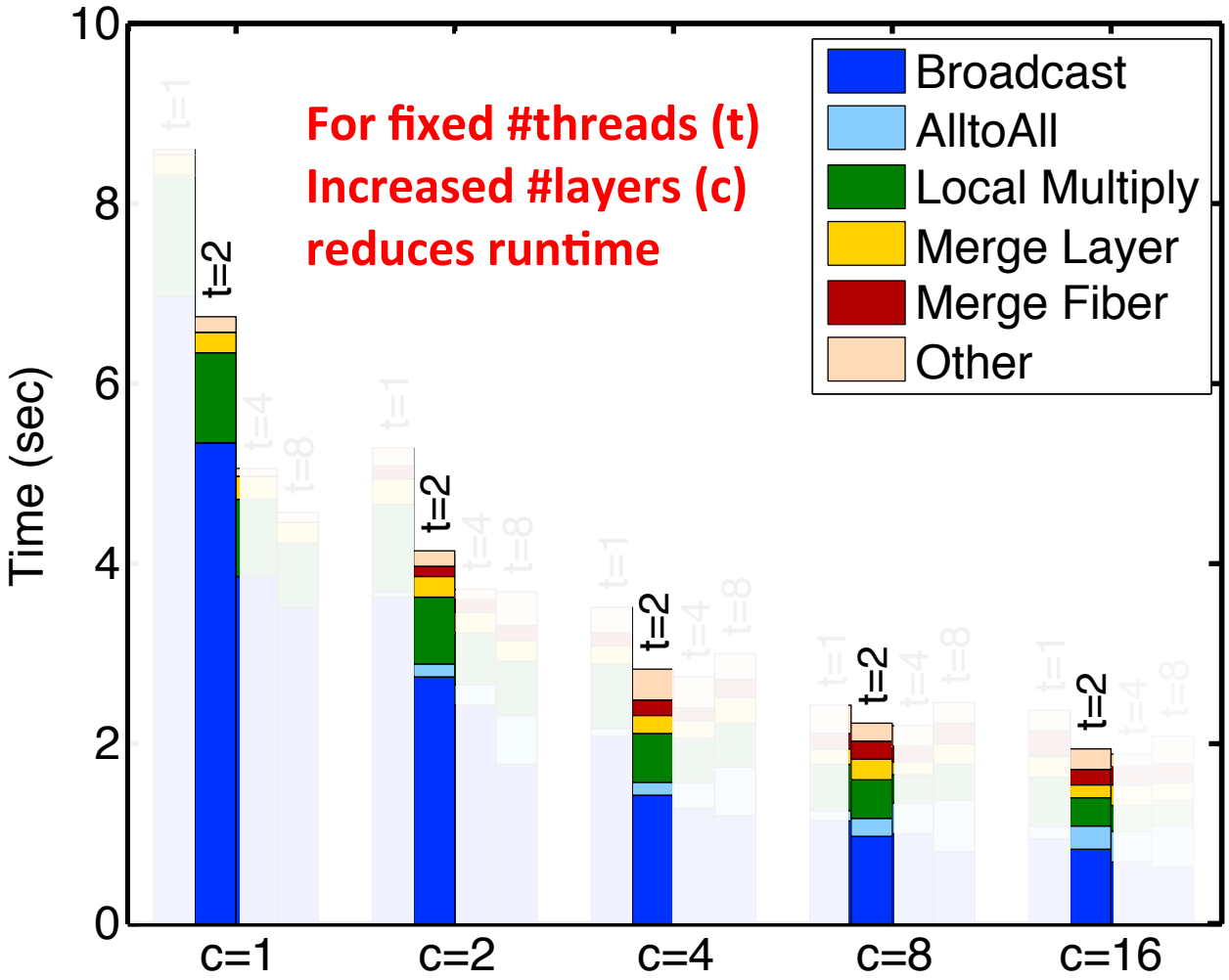
How do 3D algorithms gain performance?

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



How do 3D algorithms gain performance?

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

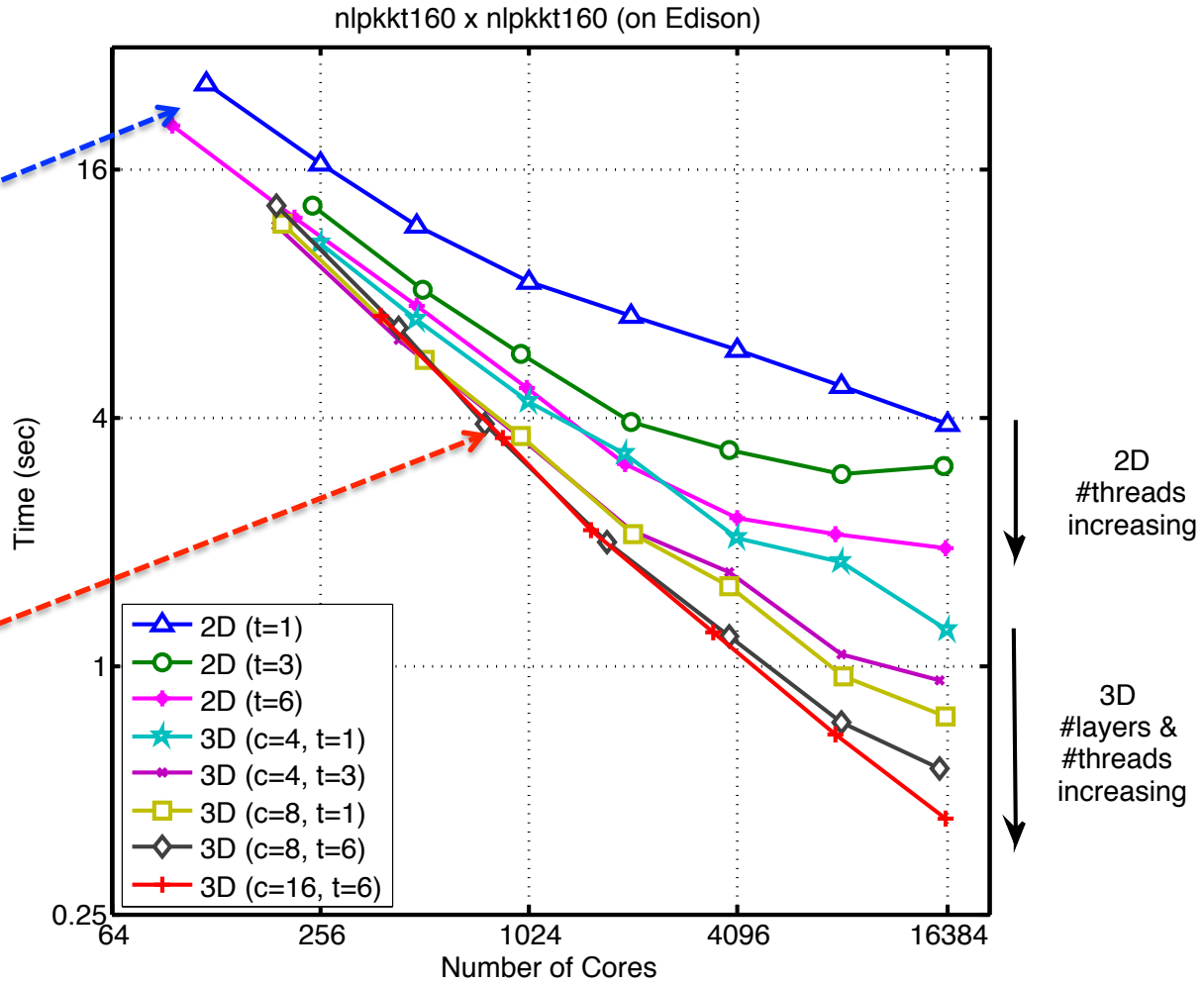


3D SpGEMM performance (matrix squaring)

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

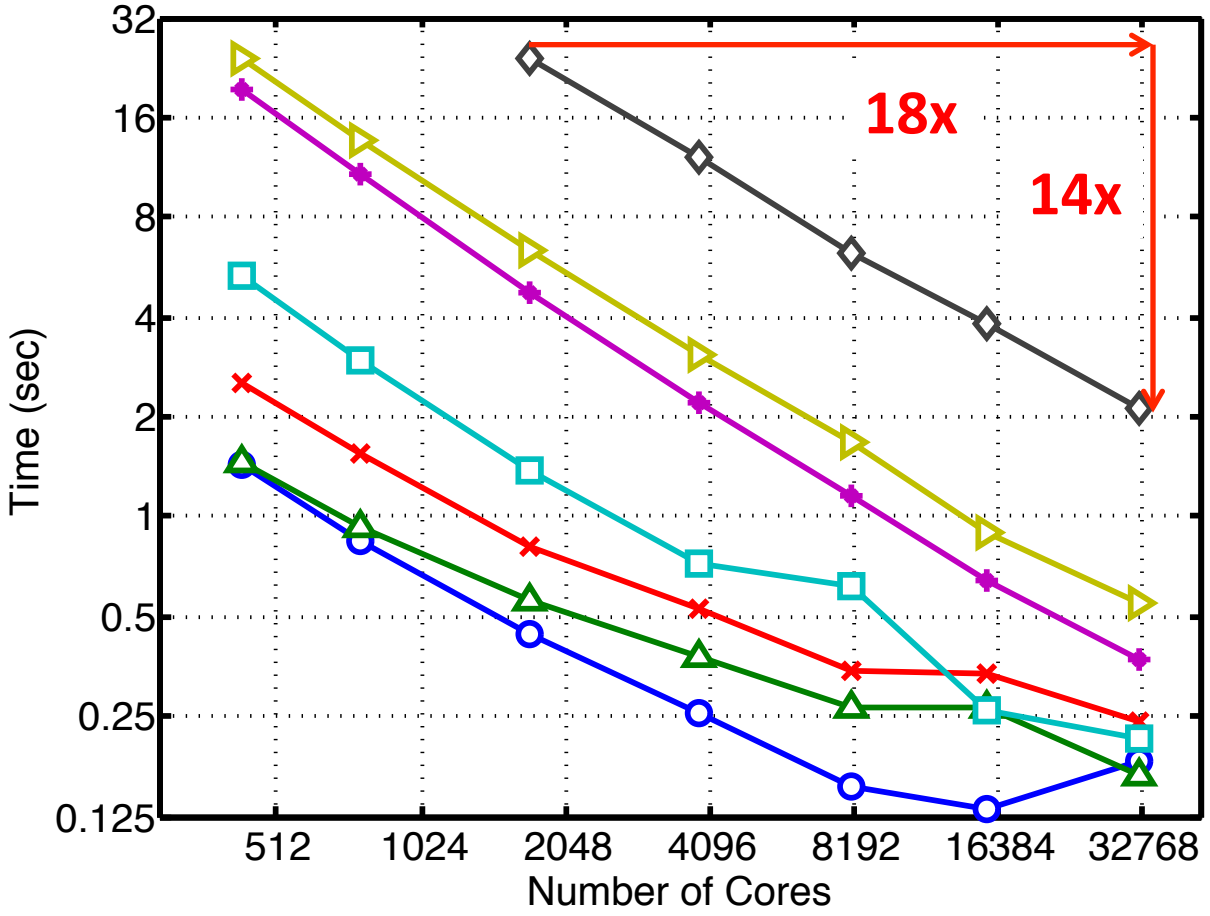
2D (non-threaded) is the previous state-of-the art

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



3D SpGEMM performance (matrix squaring)

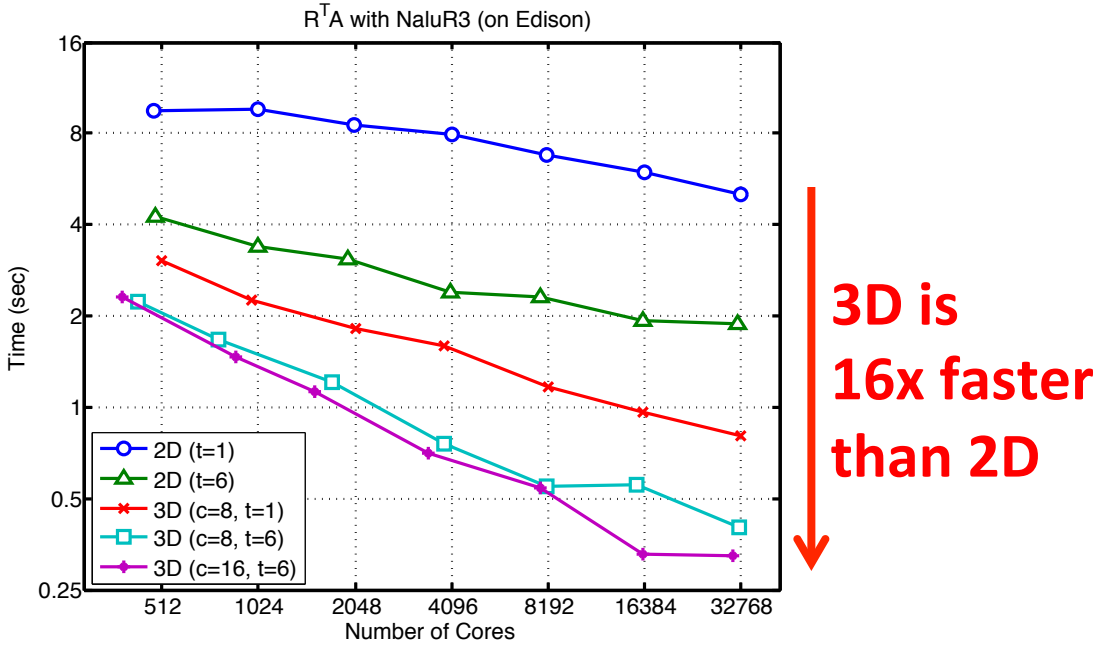
3D SpGEMM with c=16, t=6 on Edison



It-2004 (web crawl)
nnz(A) = 1.1 billion
nnz(A^2) = 14 billion

3D SpGEMM performance (AMG coarsening)

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

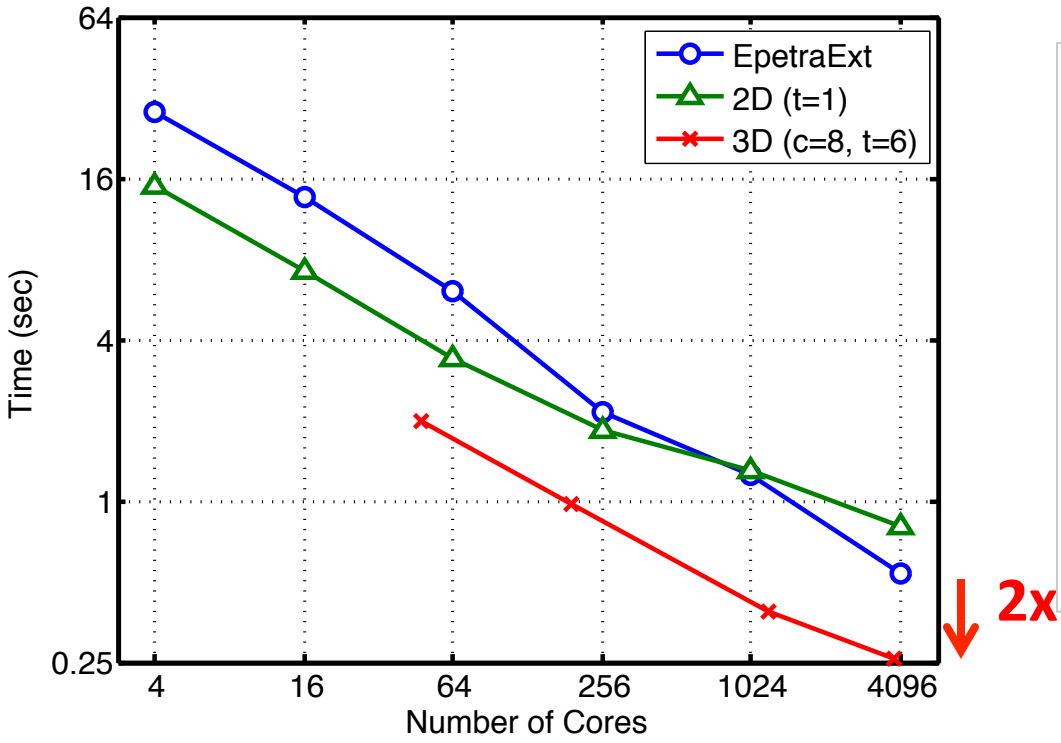


Only showing the first product (R^TA)

3D SpGEMM performance (AMG coarsening)

Comparing performance with EpetraExt package of Trilinos

AR computation with nlpkkt160 on Edison



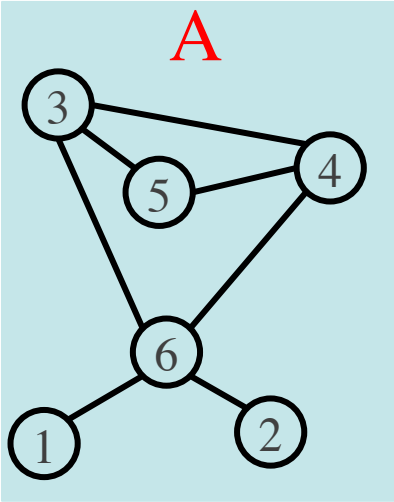
Notes:

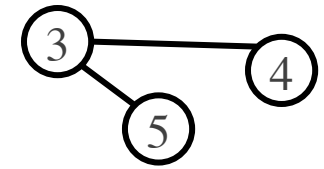
- EpetraExt runs up to 3x faster when computing AR, 3D is less sensitive
- 1D decomposition used by EpetraExt performs better on matrices with good separators.

Application of SpGEMM
Triangle Counting/Enumeration

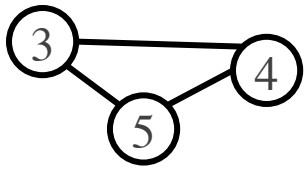
[A.Azad, A. Buluc, J. Gilbert. Parallel Triangle Counting and Enumeration using Matrix Algebra. IPDPS Workshops, 2015]

Counting Triangles





Wedge: a path of length two

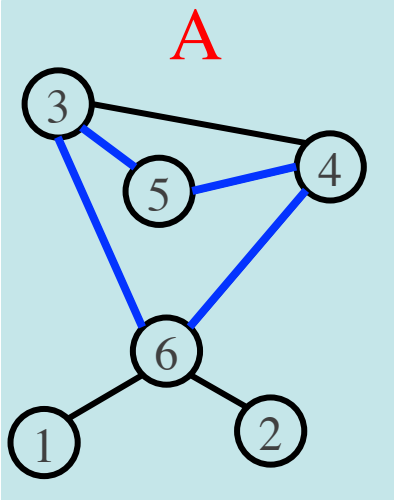


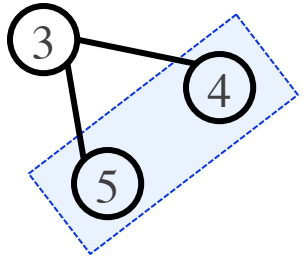
A triangle has three wedges

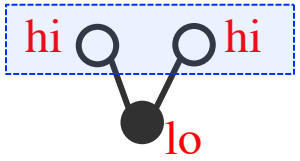
Clustering coefficient:

- Pr (wedge i-j-k makes a triangle with edge i-k)
- 3 * # triangles / # wedges
- 3 * **2** / **13** = **0.46** in example
- may want to compute for each vertex j

Triangle Counting in Matrix Algebra

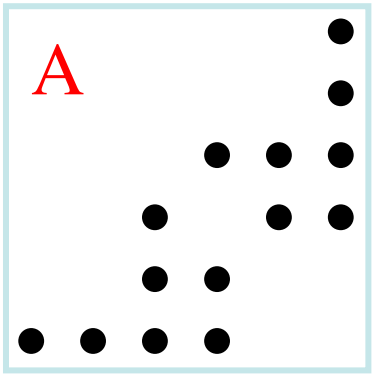




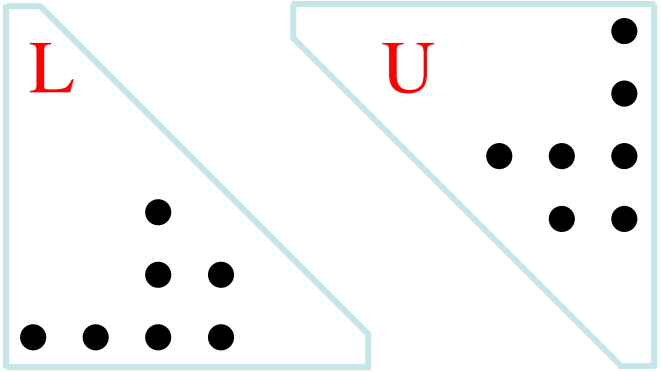


Step1: Count wedges from a pair of vertices (u,v) by finding a path of length 2. Avoid repetition by counting wedges with low index of the middle vertex.

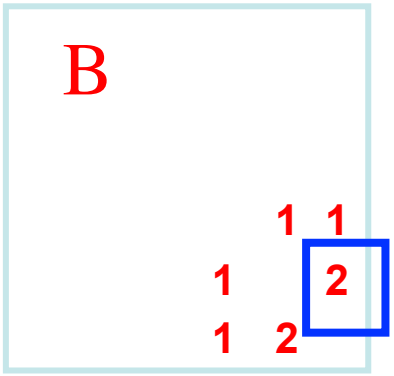
2 wedges between vertex 5 and 6



Adjacency matrix

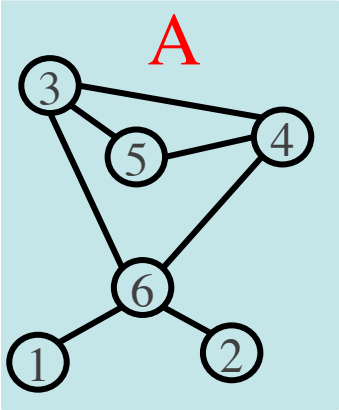


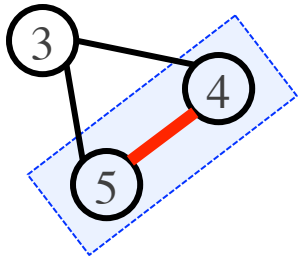
$A = L + U$ (hi->lo + lo->hi)



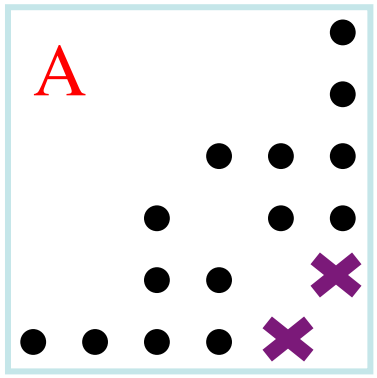
$L \times U = B$ (wedges)

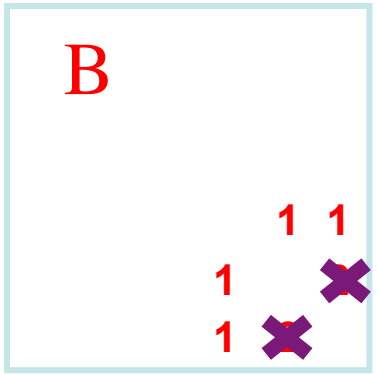
Triangle Counting in Matrix Algebra

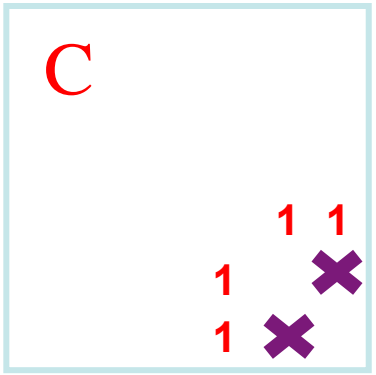




Step2: Count wedges as triangles if there is an edge between the selected pair of vertices.







#triangles
=sum(C)/2

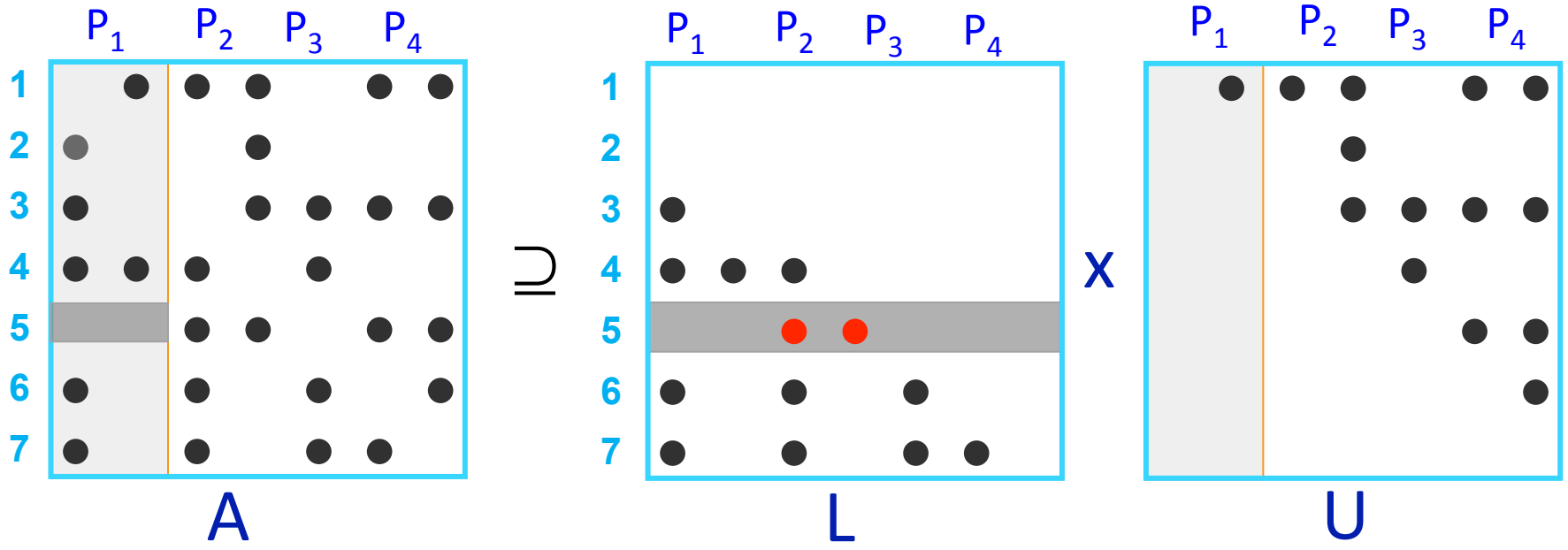
$L \times U = B$ (wedges)

$A \wedge B = C$ (closed wedge)

Goal: Can we design a parallel algorithm that communicates less data exploiting this observation?

Masked SpGEMM (1D algorithm)

Special SpGEMM: The nonzero structure of the product is contained in the original adjacency matrix A



In this example, nonzeros of **L(5,:)** (marked with red color) are not needed by P_1 because $\mathbf{A_1(5,:)}$ does not contain any nonzeros.

So, we can mask rows of L by A to avoid communication.

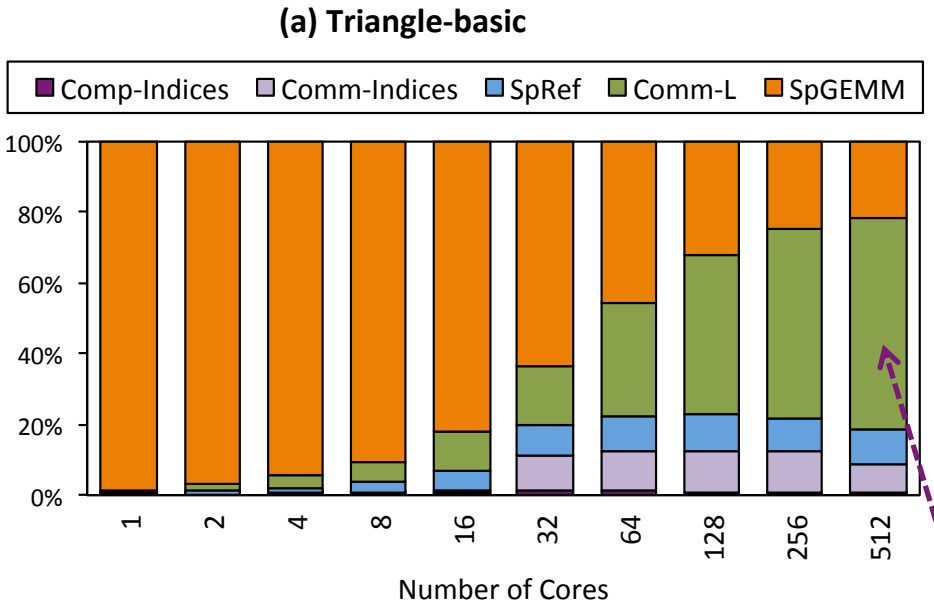
Benefits and Overheads of Masked SpGEMM

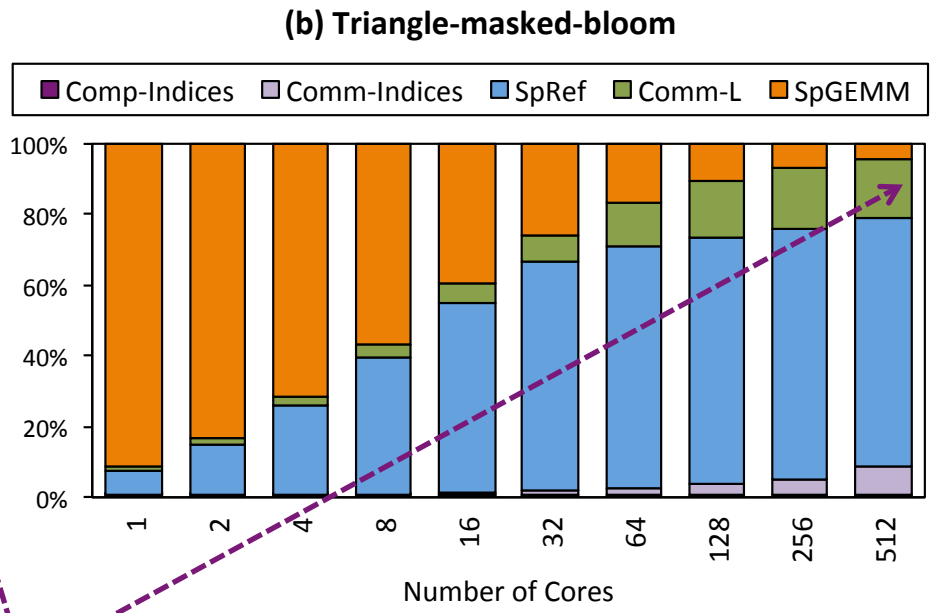
- Benefit
  - **Reduce communication** by a factor of **p/d** where d is the average degree (good for large p)

- 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?

CopapersDBLP on NERSC/Edison

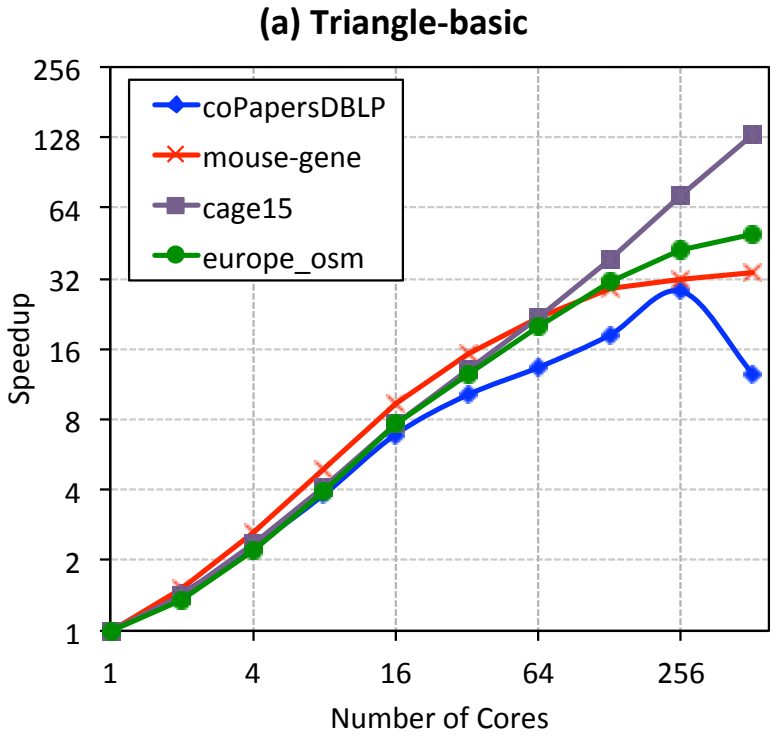


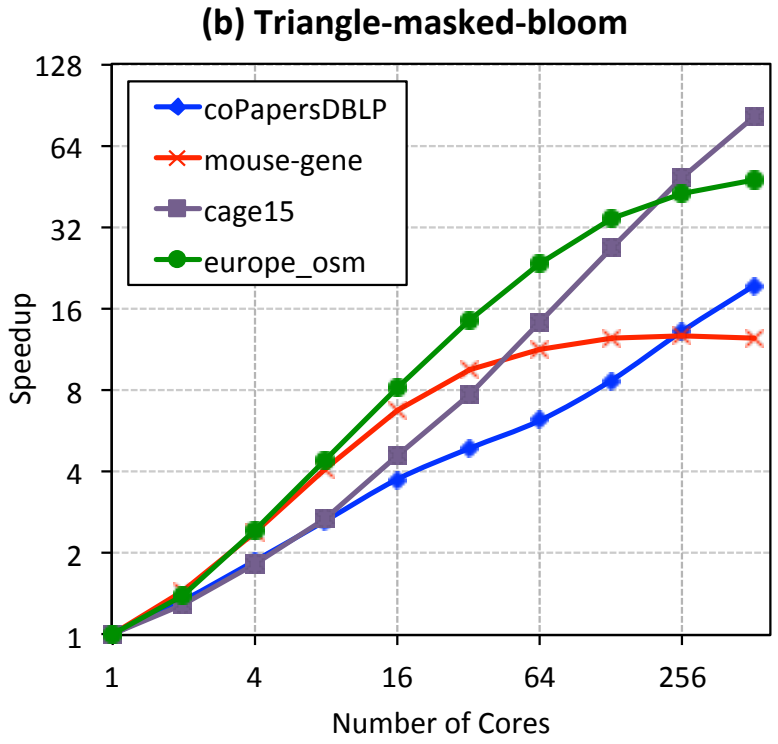


As expected, Masked SpGEMM reduces cost to communicate L

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

Strong Scaling of Triangle Counting

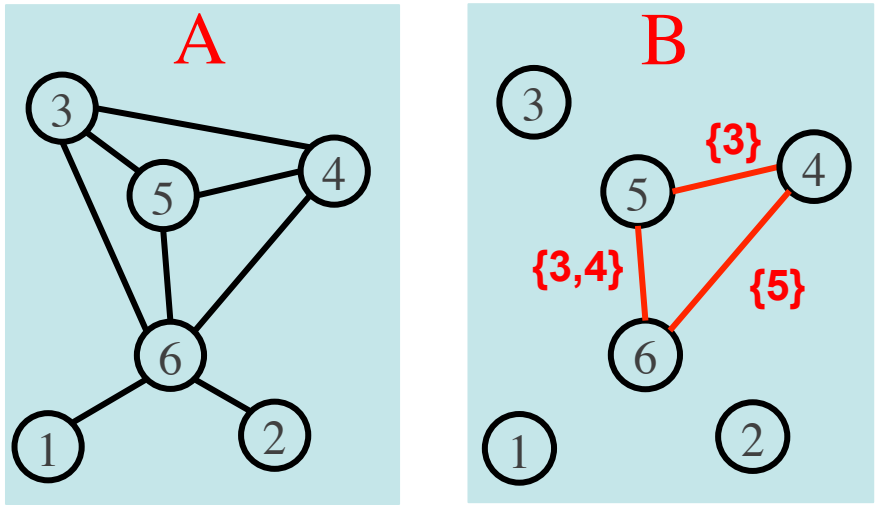




- Decent scaling to modest number of cores (p=512 in this case)
- Further scaling requires 2D/3D SpGEMM (ongoing/future work)

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), Aydin Buluc (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

U.S. DEPARTMENT OF ENERGY | Office of Science

Thanks for your attention

?

Supporting slides

- How do dense and sparse GEMM compare?

Dense:	**Sparse:**
Lower bounds match algorithms.	Significant gap
Allows extensive data reuse	Inherent poor reuse?

Communication Lower Bound

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

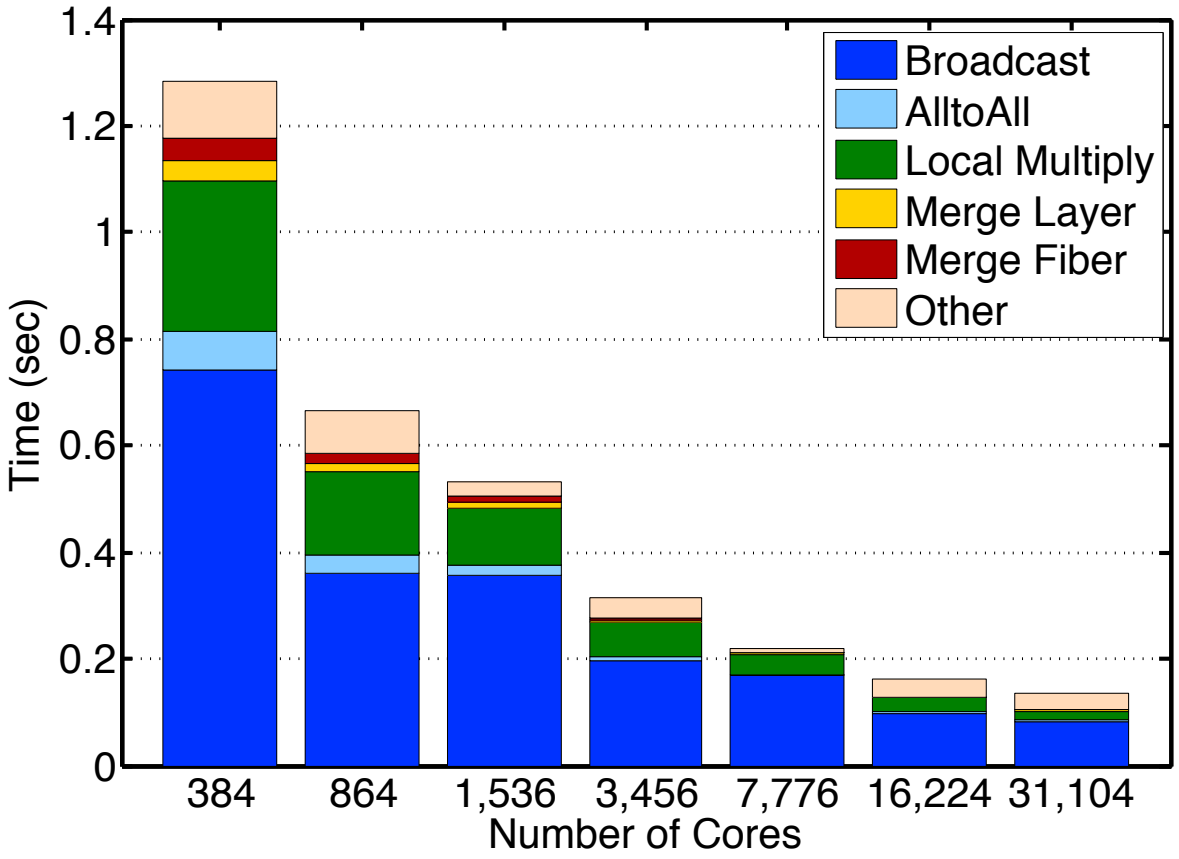
$$\Omega\left(\min\left\{\frac{dn}{\sqrt{P}}, \frac{d^2 n}{P}\right\}\right)$$

(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?

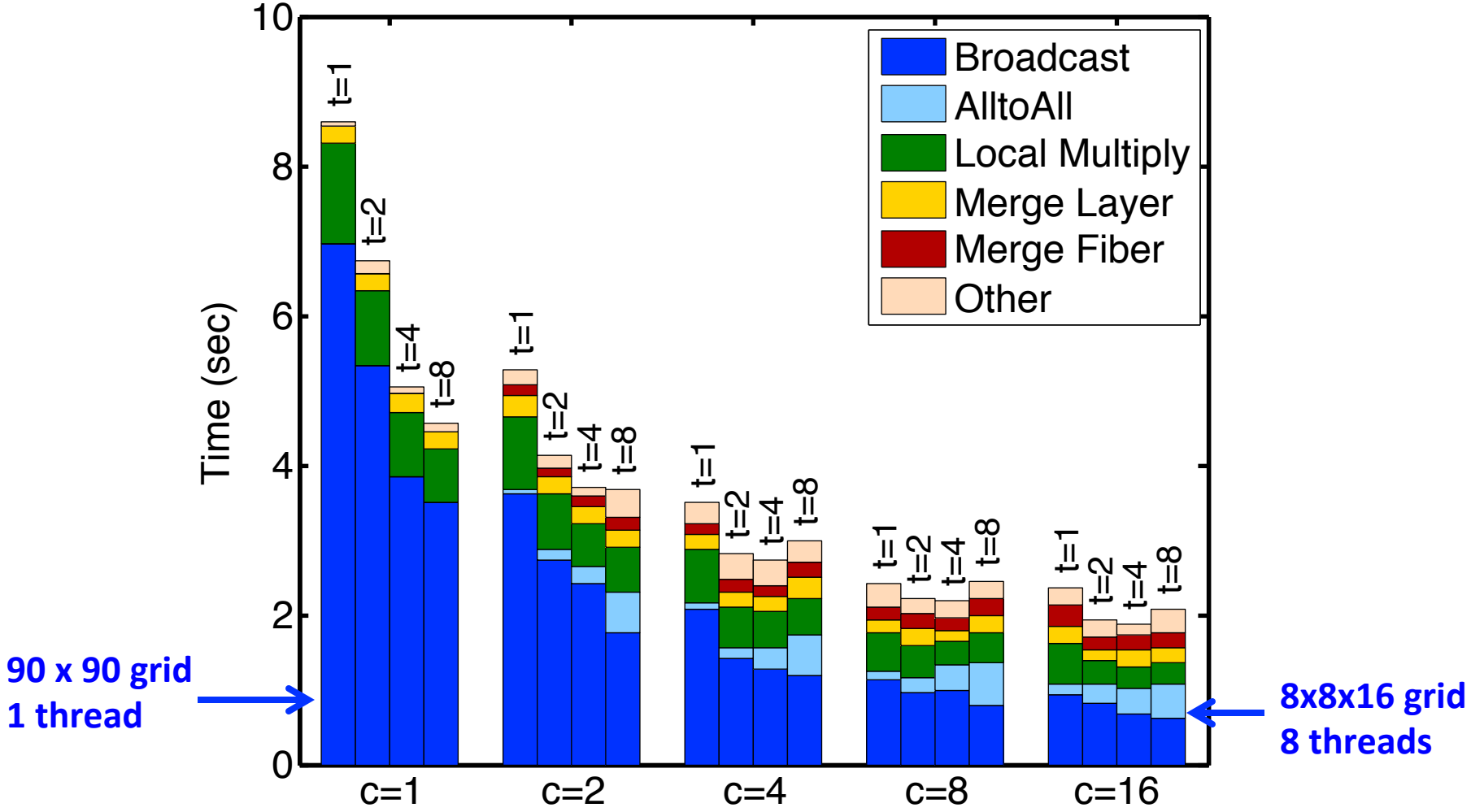
Broadcast dominates on all concurrency



Squaring nlpkkt160 on Edison

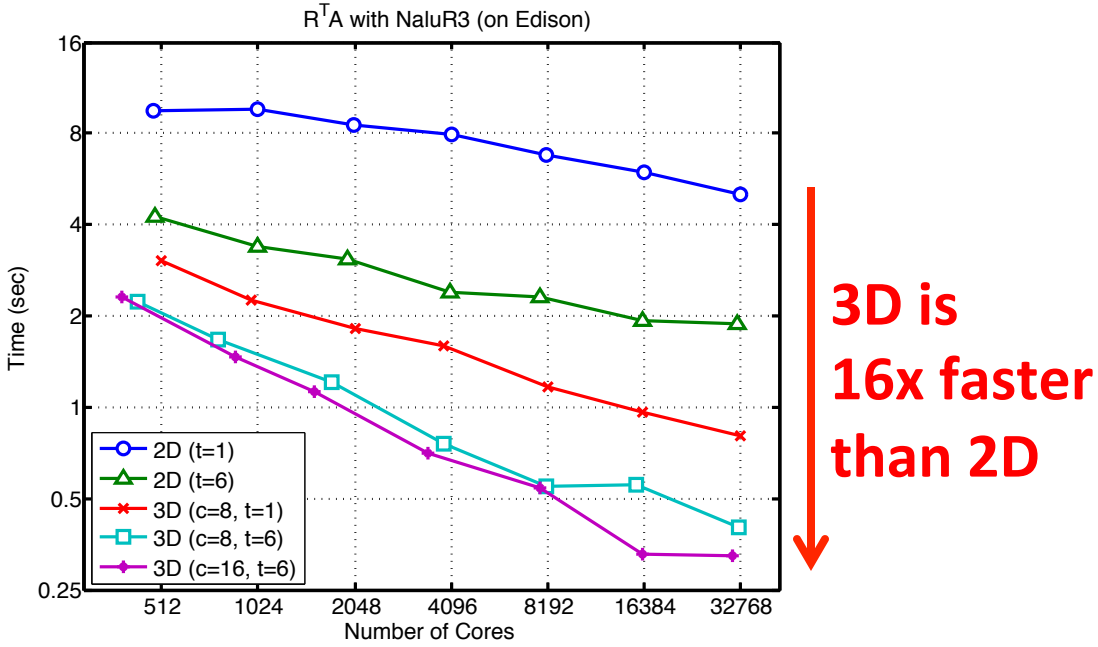
How do 3D algorithms gain performance?

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)
- $\mathbf{A_{coarse} = R^T A_{fine} R}$ where R is the restriction matrix
- R constructed via distance-2 maximal independent set (MIS)



3D is 16x faster than 2D

Limited scalability of 3D in computing R^TA

NaluR3	
nnz(A)	474 million
nnz(A^2)	2.1 billion
nnz(R^TA)	**77 million**

Not enough work on high concurrency

Only showing the first product (R^TA)

Communication Reduction in Masked SpGEMM

- **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: **nd/p**
  - Number of columns of L needed by a processor: **nd/p**
  - Data received per processor: **$O(nd^3/p^2)$**
- If no output masking is used ("improved 1D")
  - **$O(nd^2/p)$** [Ballard, Buluc, et al. SPAA 2013]
- reduction of a factor of **p/d** over "improved 1D"
  - good for large p

Experimental Evaluation

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

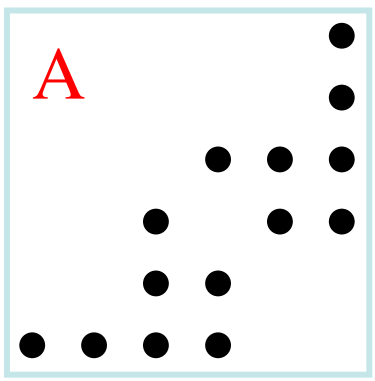
Higher nnz(B) / nnz(A.*B) ratio => *more potential communication reduction* due to masked-spgemm for ***triangle counting***

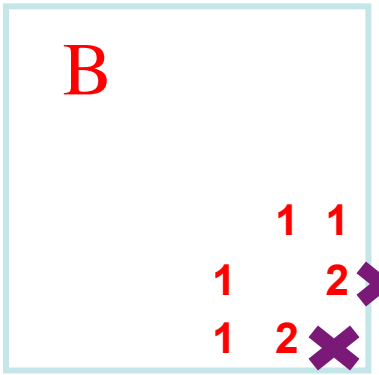
Ranges between 1.7 and 500

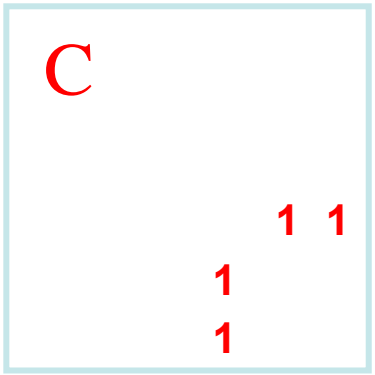
Higher Triads / Triangles ratio => *more potential communication reduction* due to masked-spgemm for ***triangle enumeration***

Ranges between 5 and 1000

Serial Complexity







sum(C)/2

2 triangles

$L \times U = B$ (wedges)

$A \wedge B = C$ (closed wedge)

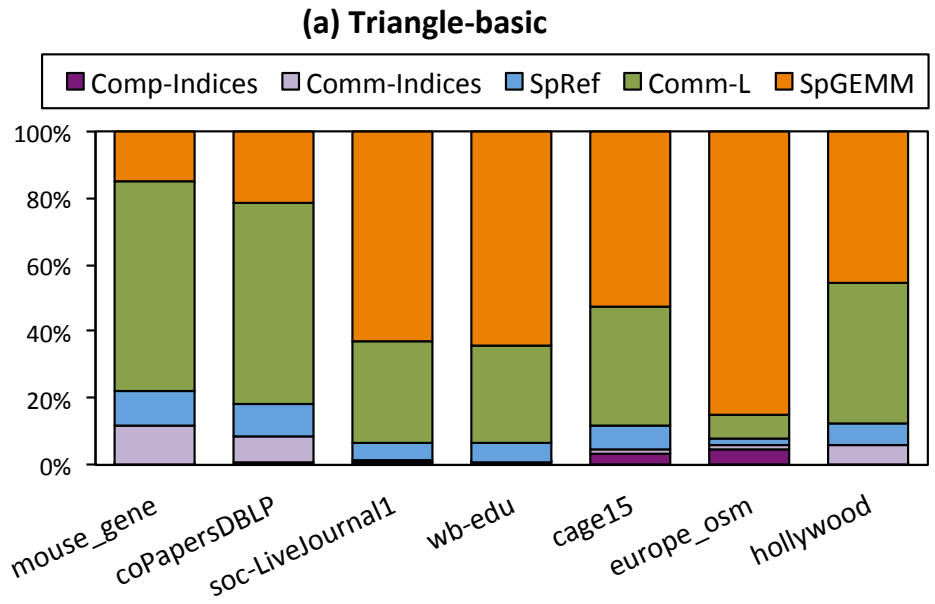
Model: *Erdős-Rényi(n,d)* graphs aka *G(n, p=d/n)*

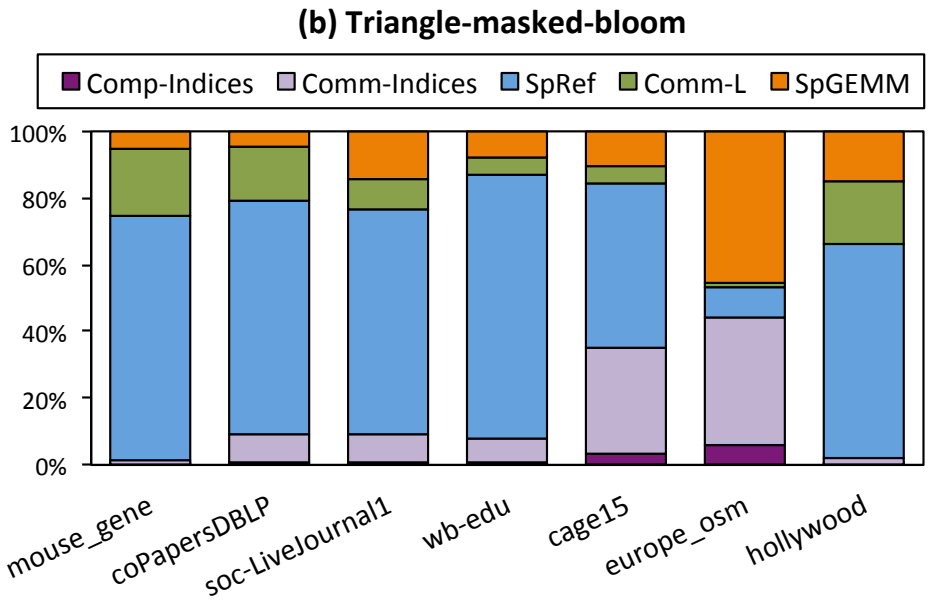
Observation: The multiplication B = L· U costs **$O(d^2n)$** operations. However, the ultimate matrix C has at most **2dn** nonzeros.

Goal: Can we design a parallel algorithm that communicates less data exploiting this observation?

Where do algorithms spend time?

On 512 cores of NERSC/Edison





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

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