Efficiently Multiplying Sparse Matrix - Sparse Vector for Social Network Analysis

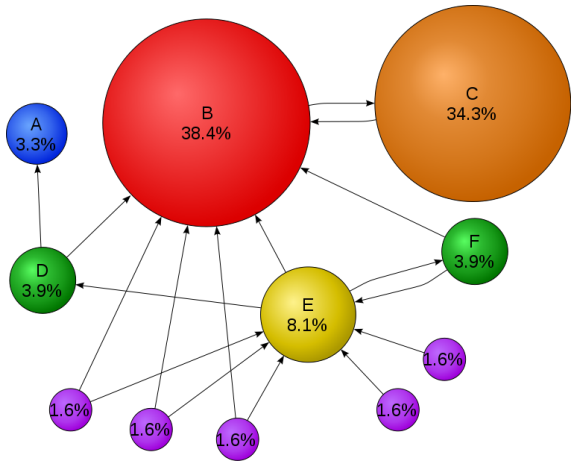
Ashish Kumar
Mentors - David Bader, Jason Riedy, Jiajia Li

Indian Institute of Technology, Jodhpur

6 June 2014

Problem Motivation: PageRank

PageRank- An algorithm which ranks the nodes of a graph on their importance –



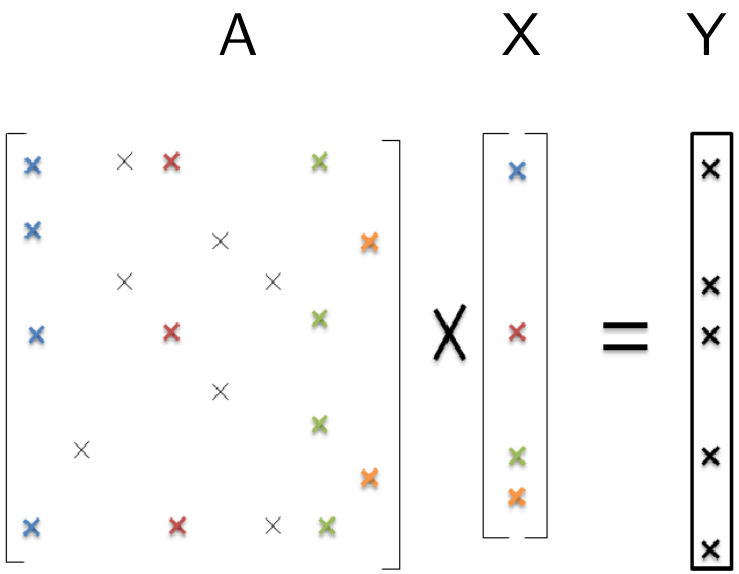
1. Involves sparse matrix - Dense vector multiplication.
2. Considering only the changes on a streaming graph, we encounter sparse matrix- sparse vector multiplication instead.
3. We thus explore the implementation space of sparse matrix - sparse vector multiplication.

Contribution

1. We implement a performance portable version of sparse matrix - sparse vector multiplication, which performs comparably, or better than:
   1. The dense vector product
   2. Implementations relying on atmoic operations, even on architectures supporting efficient atomic operations.
2. On pequin server and for batch size of 80,000 insertions our implementation runs
   1. Almost 6-times faster than dense multiplication on 32 threads (competing with Batch Atomic)and,
   2. Around 4-times faster than dense multiplication on 64 threads (whereas Batch Atomic is only 2 times faster)

Problem Description

We are given a Sparse Matrix A (in CSC Format), and a sparse vector X (in compressed vector format), and we have to compute the product:



where the output vector Y should also be in compressed vector format.

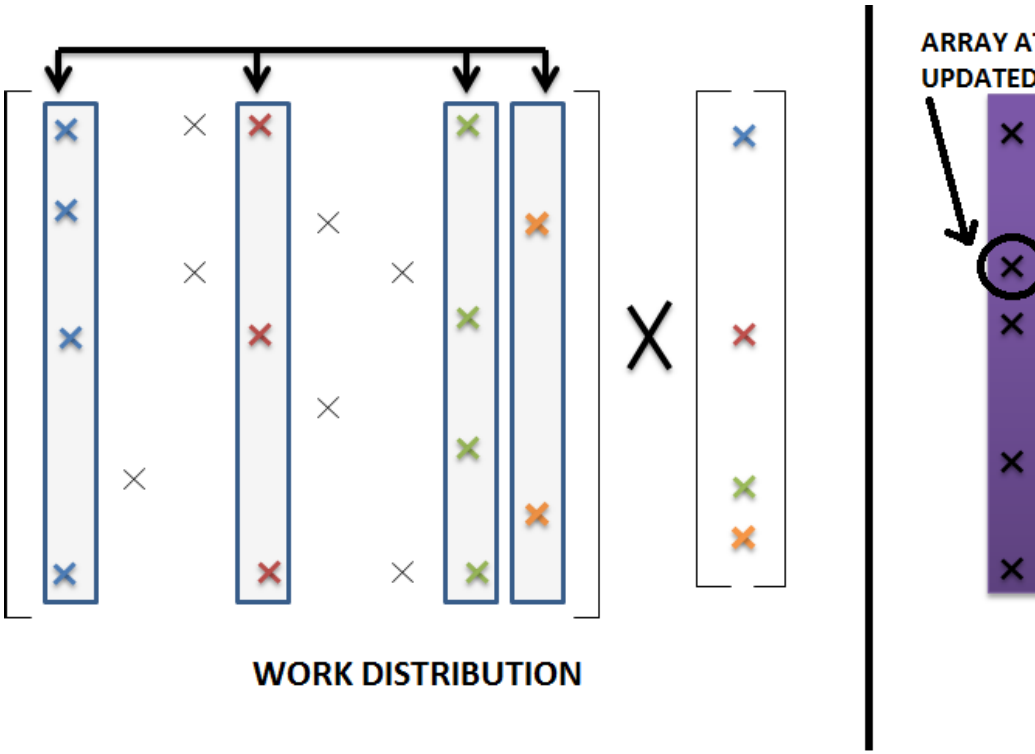
High Level Implementation Details

Variants Implemented :

1. Atomic
2. Batch Atomic
3. Sort and Merge
4. Parallel Reduction
5. Load Balance

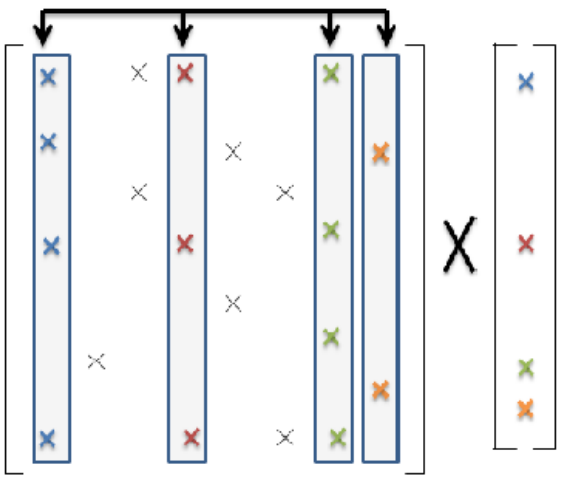
Atomic and Batch Atomic

Global output array is atomically updated by threads, and then sequentially compressed.

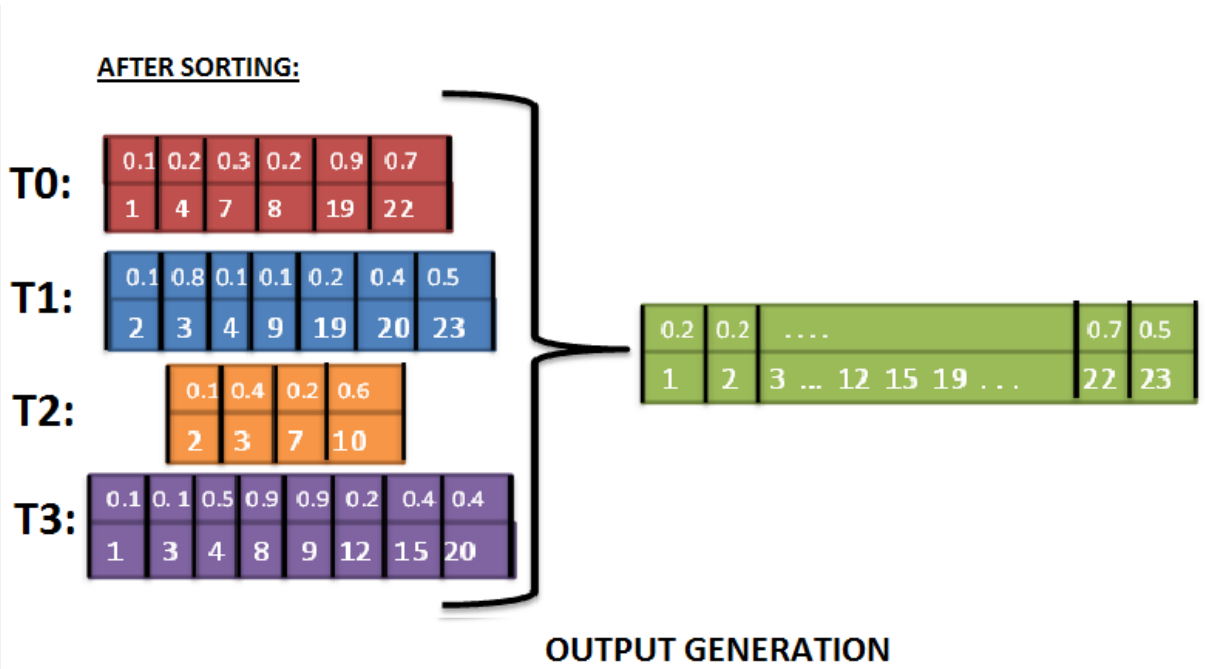


Sort and Merge

Each Thread generates a partial sorted output in compressed form, which is then uniquely merged into compressed output array.



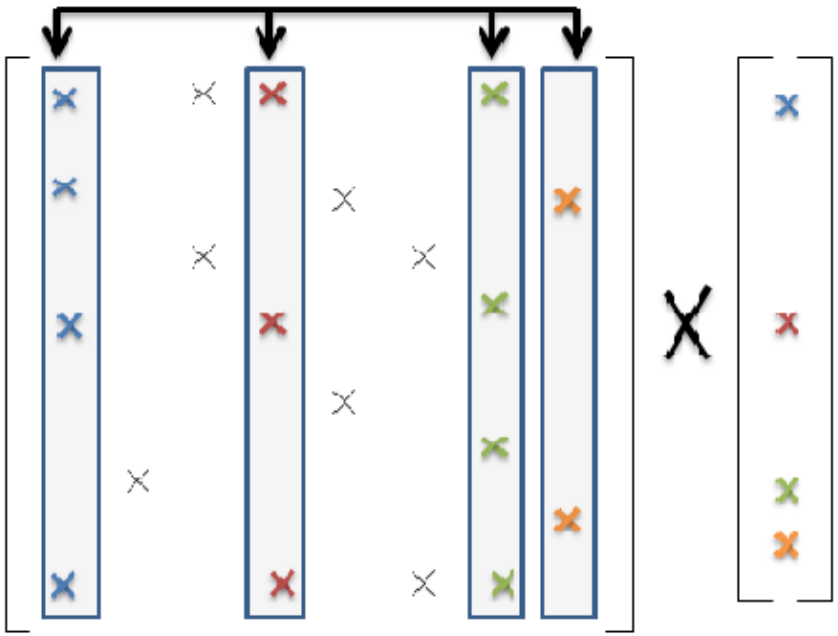
WORK DISTRIBUTION
(SAME AS IN ATOMIC)



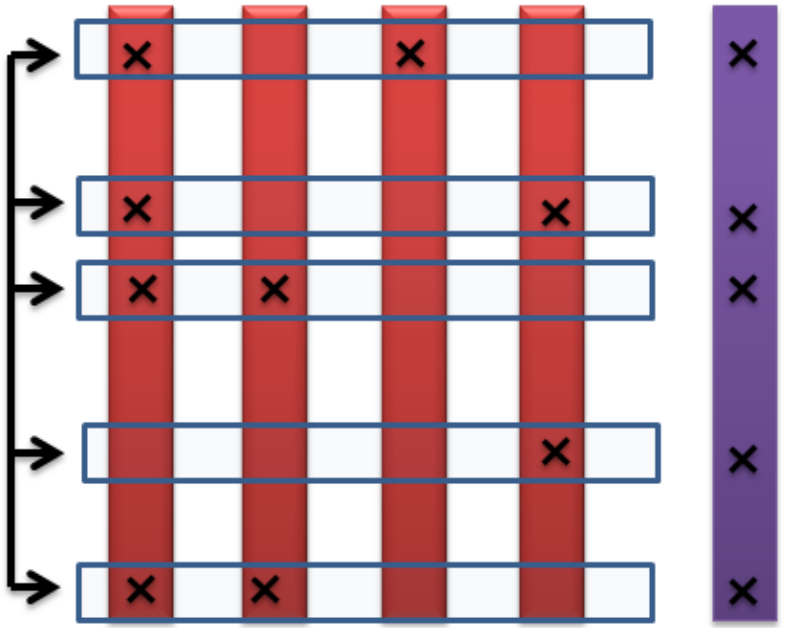
OUTPUT GENERATION

Parallel Reduction

Each thread generates partial outputs in indexed arrays, which are then reduced in parallel to generate the output vector. It is then sequentially compressed.



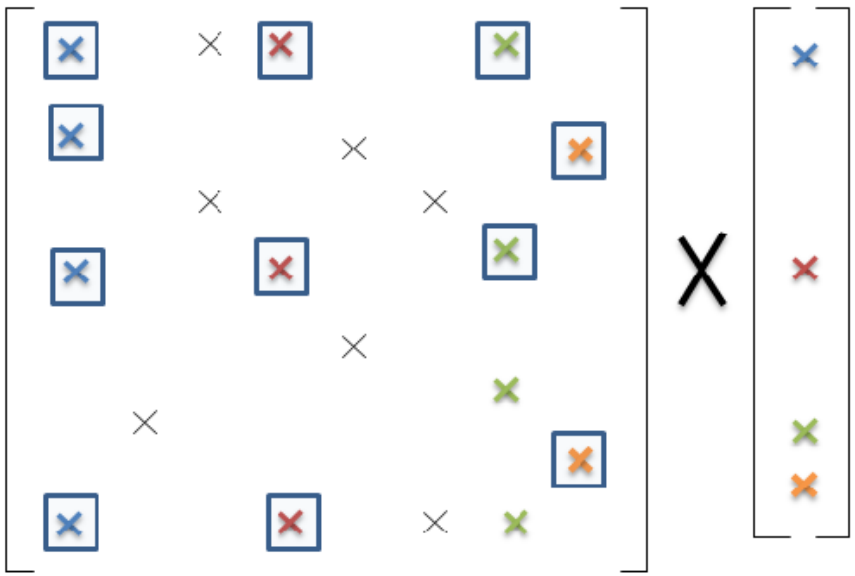
WORK DISTRIBUTION
(SAME AS ATOMIC)



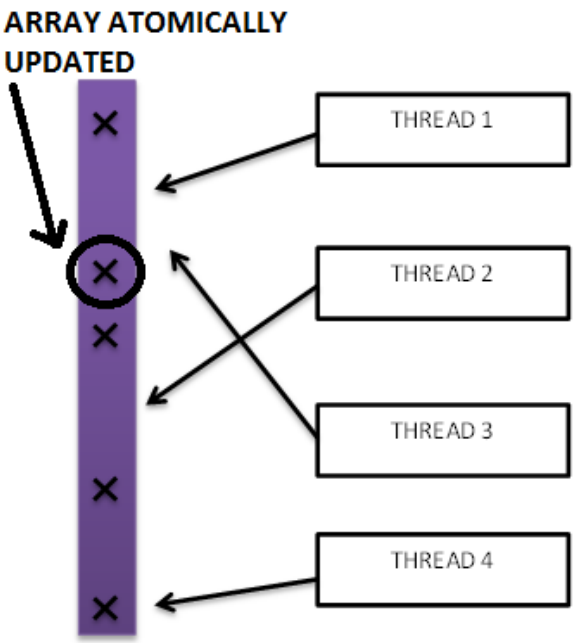
OUTPUT GENERATION

Load Balanced

Same as the atomic implementation except for the units of work which are generated, which are finer than the atomic version.



WORK DISTRIBUTION

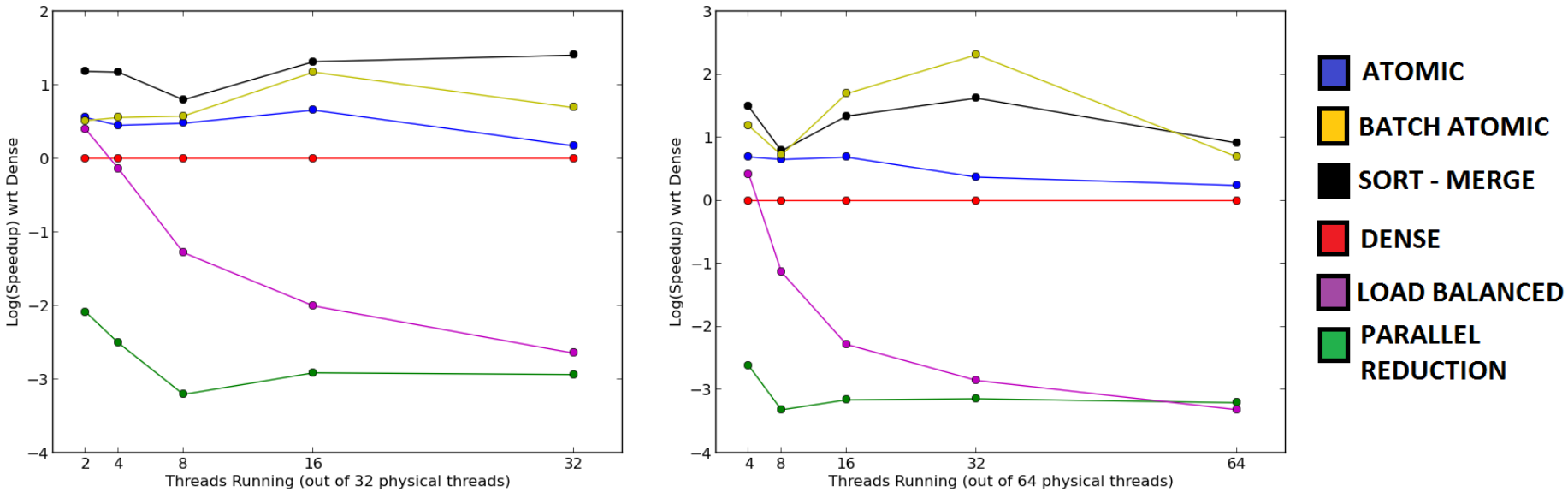


OUTPUT GENERATION

Comparisons:

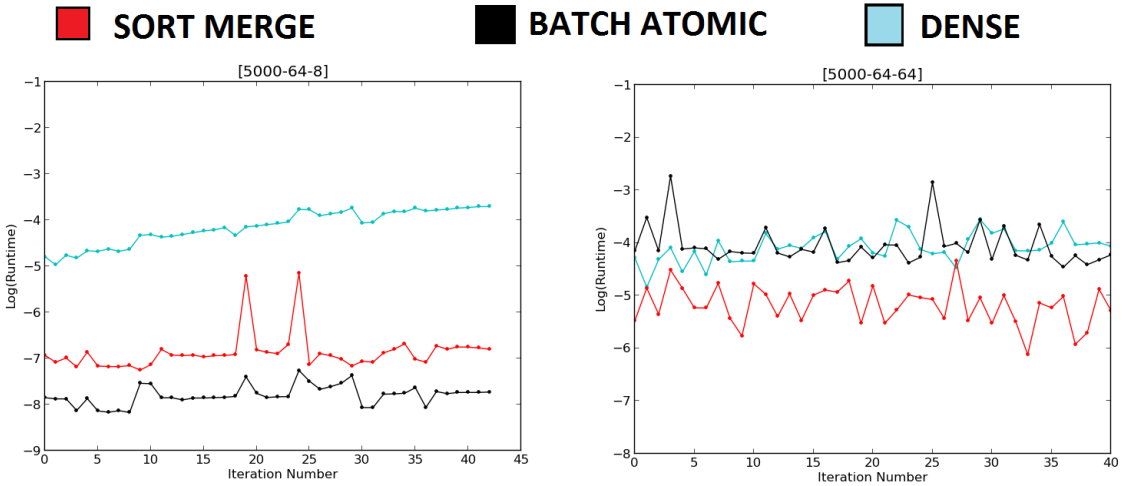
1. Platform - Both are multisocket, eight-core Intel Xeons using the Ivy Bridge micro-architecture.
   1. Serrano: 2 socket - hence a total of 16 physical cores and 32 hardware threads
   2. Pequin: 4 socket - hence a total of 32 physical cores and 64 hardware threads
2. Graphs -
   1. Results on : in-2004.graph # nodes=1382908 # edges=13591473
   2. Verified on : eu-2005.graph # nodes=862664 # edges=16138468
3. Streaming Behaviour
   1. Tested on Stinger - For handling edge insertions in dynamic graphs.
   2. Insertions - 5000, 20000, 50000, 80000, 100000

Comparison with Dense Multiplication - Speedups

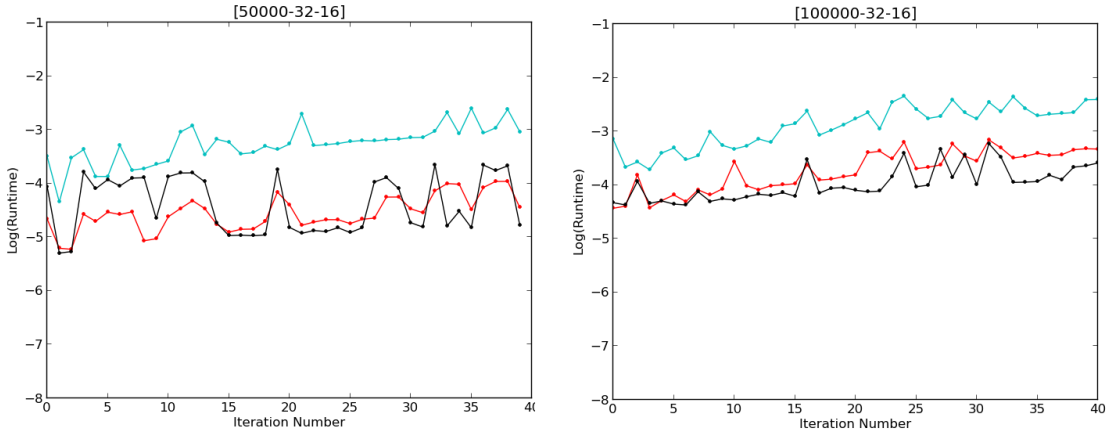


1. Atomic, Batch Atomic and Sort and Merge are the only ones which beat Dense.
2. Load Balance and Parallel Reduction versions have very poor performance.

Sort and Merge vs Batch Atomic vs Dense Multiplication

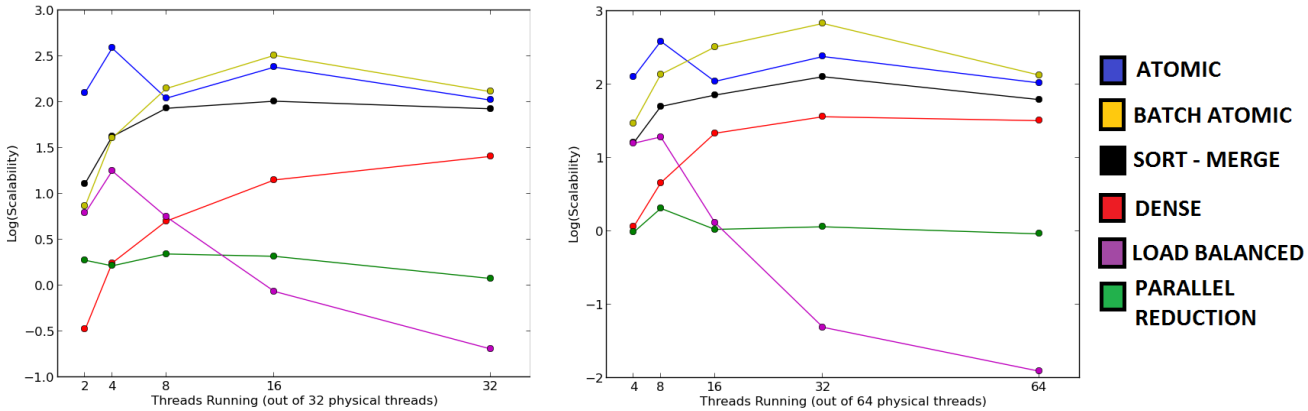


More threads have more atomic conflicts, and larger time for merging



Large number of insertions slowdown sort step

Strong Scalability Plots



1. Dense shows a consistent strong scalability.
2. Strong scalability curves of Batch Atomic and Atomic are similar and are hampered by atomic conflicts.
3. Strong scalability of Sort and Merge is hampered by the sequential merge step.

Conclusions

1. PageRank will achieve better performance by exploring the dynamic features, using the sparsity of each vector.
2. Of all, the sort and merge and the batch atomic implementations are the best.
3. It is necessary to choose different algorithms according to the input features, e.g. sparsity, # insertions, # threads.

Future Work

The sort and merge can be improved by:

1. Using radix sort , as we would always be sorting integers. Here, we tradeoff space for time.
2. A parallel merge step for larger number of threads.

THANKYOU