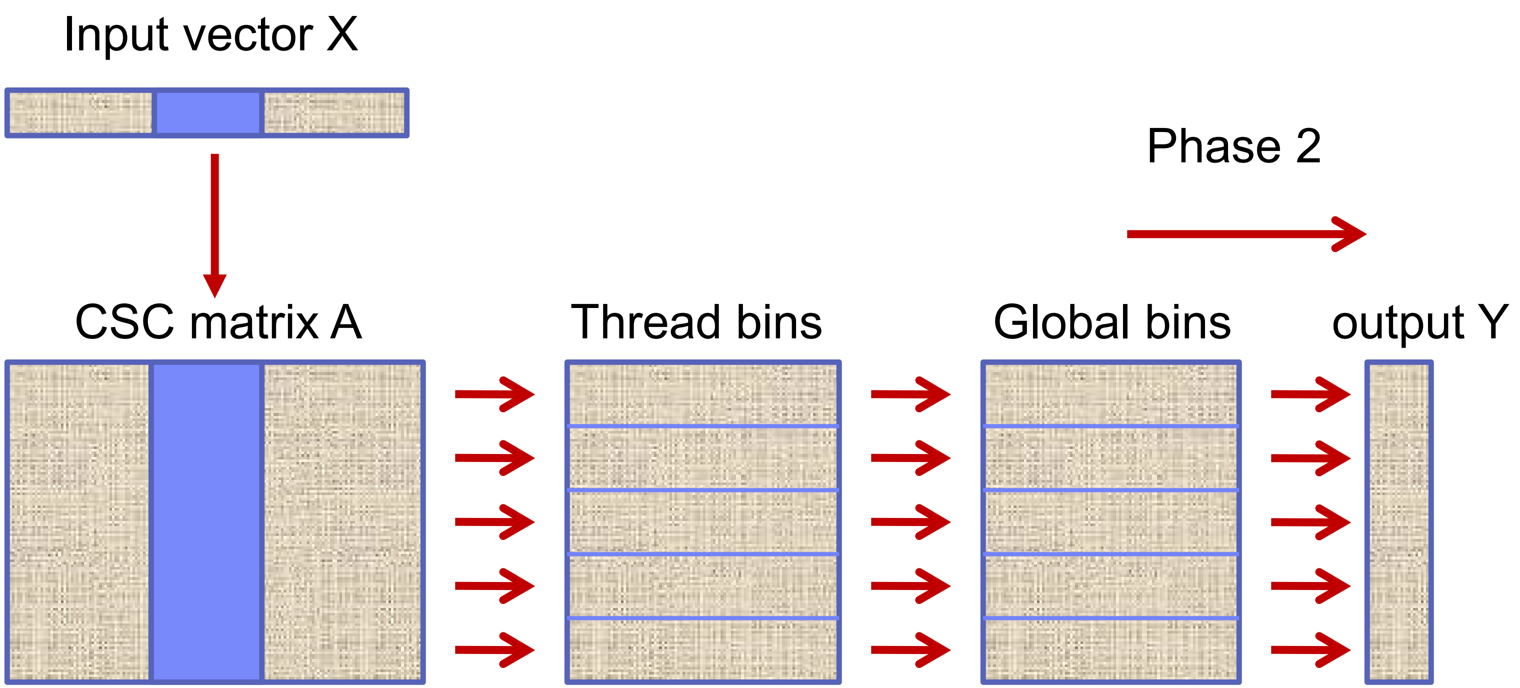
Efficient Implementation of sparse matrix – sparse vector multiplication for large scale graph analytics

Mauricio J. Serrano

- **Naïve algorithms have poor memory access patterns… not cache friendly.**
- **Prior work (references 1 and 2):**
  - SpMV/SpMSpV addresses memory latency by splitting `y=Ax` into two phases:
    - Phase 1: Scaling phase `A'=S(A,x)` using a temporary matrix
    - Phase 2: Reduction phase `y=R(A')`
    - This is more cache/memory friendly , in spite of the extra work/memory needed.
- **Prior papers:**

1. D. Buono et al, "Optimizing sparse matrix-vector multiplication for large scale analytics", ICS 2016, Presented results for IBM POWER8, **best algorithm for SpMV**.
2. A. Azad, A. Buluc, "A work-efficient parallel sparse

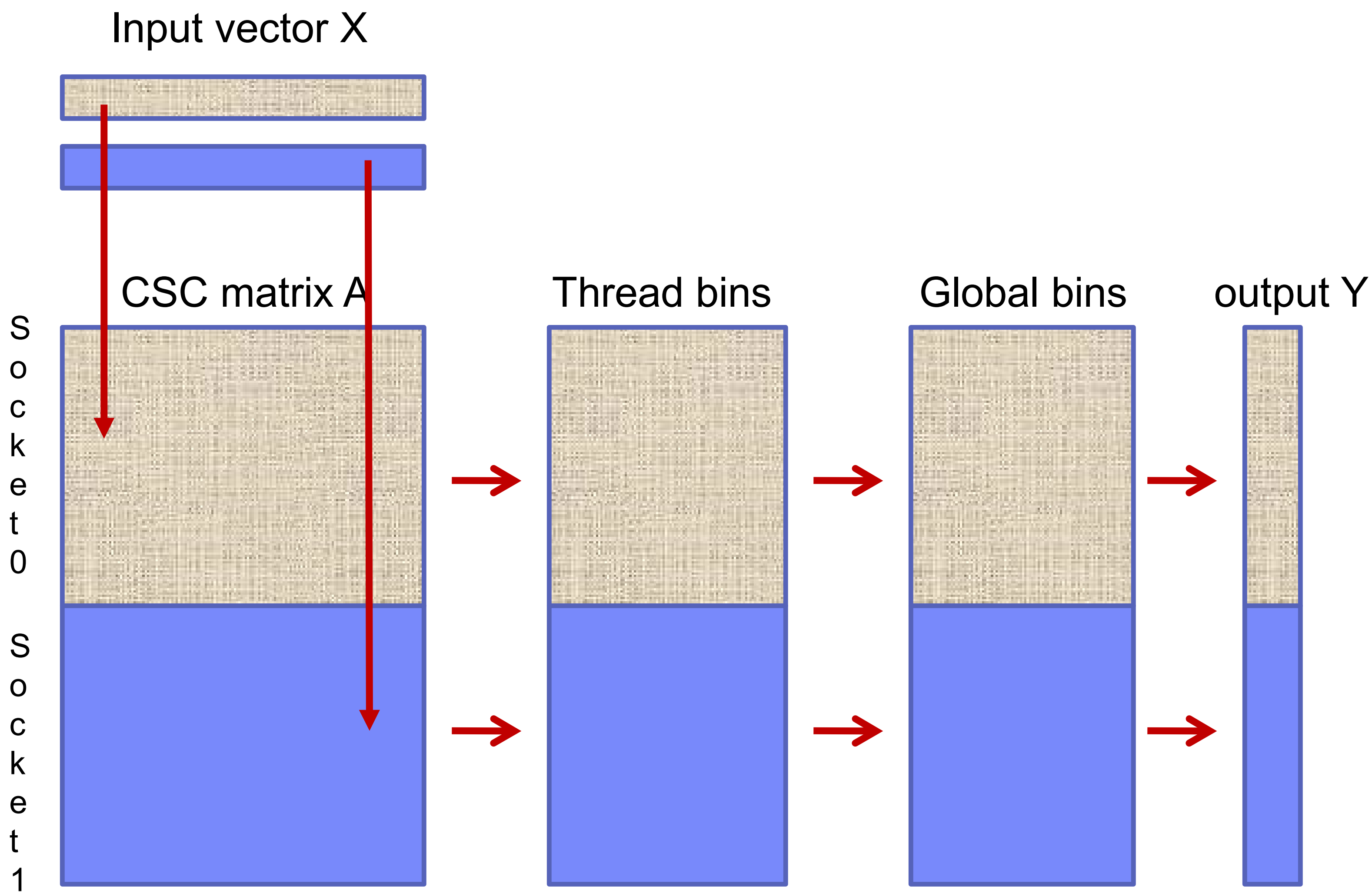
Our algorithm



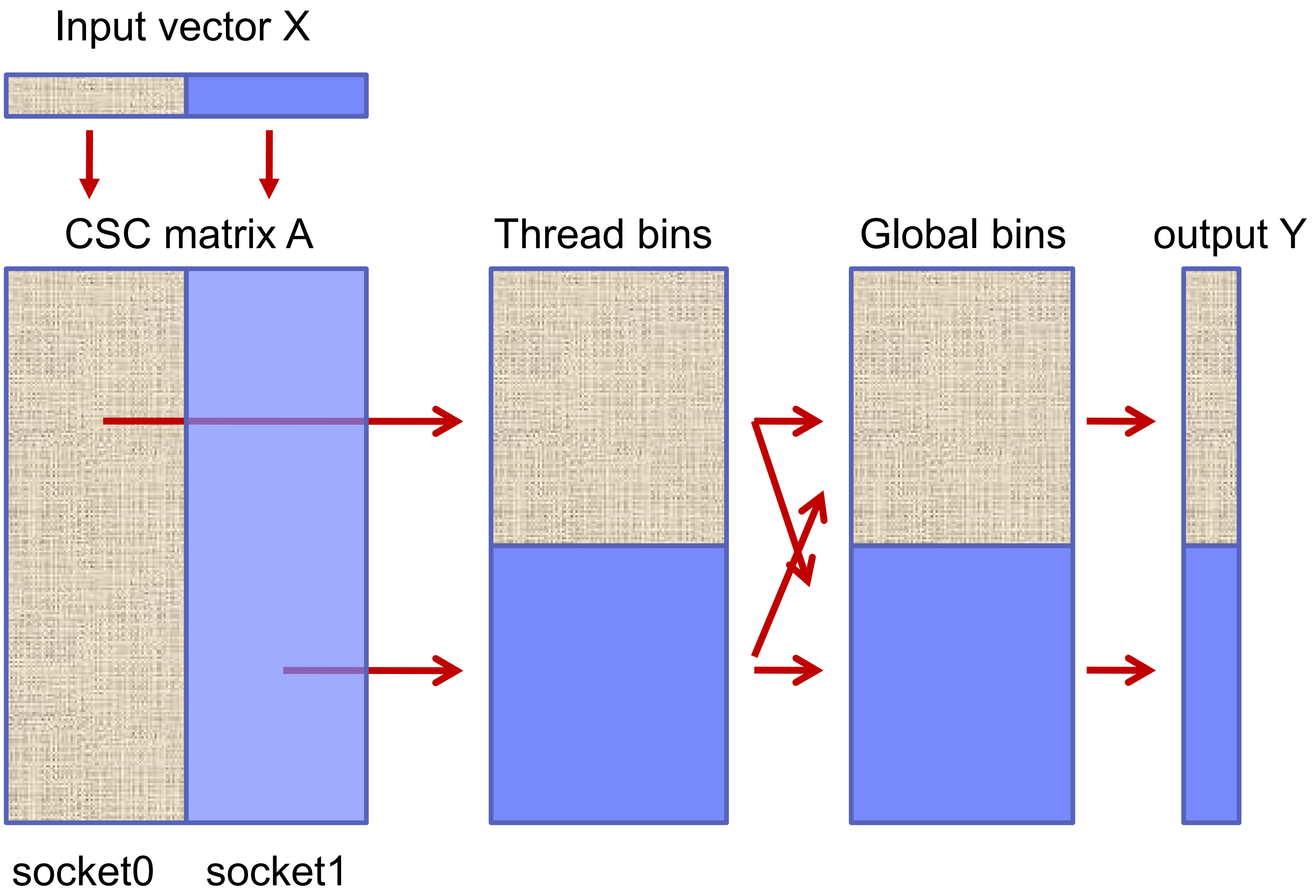
We present a new approach for SpMSpV y=Ax

- **Phase 1: A' = S(A,x)**
  - –Each thread scans a portion of the input vector `x`
  - –Each thread maintains a collection of small fixed capacity bins (or buckets)
    - •Each bucket captures accesses to a limited portion of the output vector `y`
  - –Each thread obtains `[row,product=weight * x[col]]` where weight is obtained from the CSC representation of the matrix
  - –Each thread performs `bucket = row/number_of_rows_per_bucket`
  - –Each thread deposits `[row,product]` in the corresponding bin (or bucket)
    - • A bucket counter is incremented for each operation.
  - –When the small fixed capacity bucket is full, contents transferred to a global bucket

NUMA strategy for significant number of non-zeros (example has two sockets).



Numa strategy for very sparse input vectors



When to Choose SpMV instead of SpMSpV

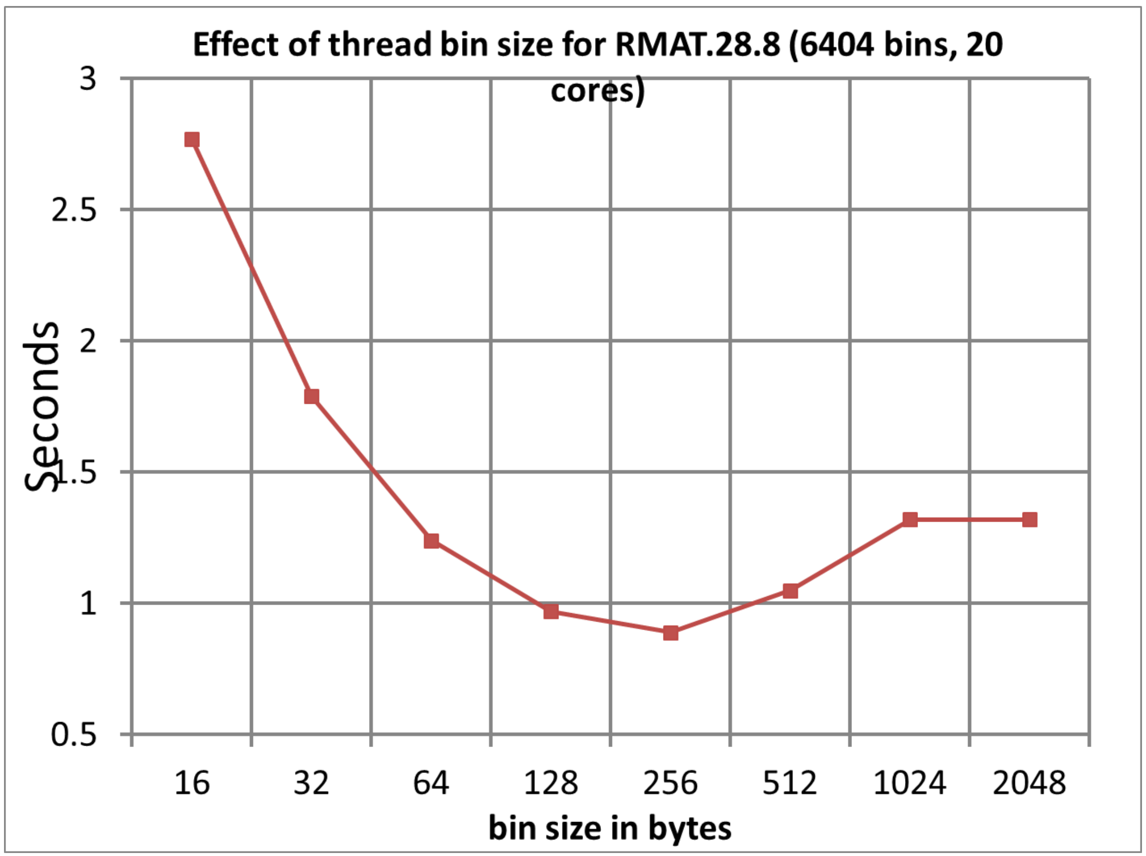
- **SpMV can be performed more efficiently by Buono et. al algorithm, because bucket information can be precomputed (input vector is dense),**
  - no need for a runtime bucket technique.
- **In some cases it is more efficient to perform SpMSpV as SpMV, in spite of the extra work needed to convert input/output vectors from sparse/dense and viceversa.**
- **We used the heuristic shown below: estimate the number of nonzeros that the operation will involve**
  - Our re e SpMV.

```
Procedure SpMV(M,X):
nnz(X,M) = 0;
for every nonzero position i in vector X
    nnz(X,M) += edgePtr[i+1] - edgePtr[i]
if ((nnz(X,M) / nnz(M)) > threshold)
    use SpMV instead of SpMSpV
```

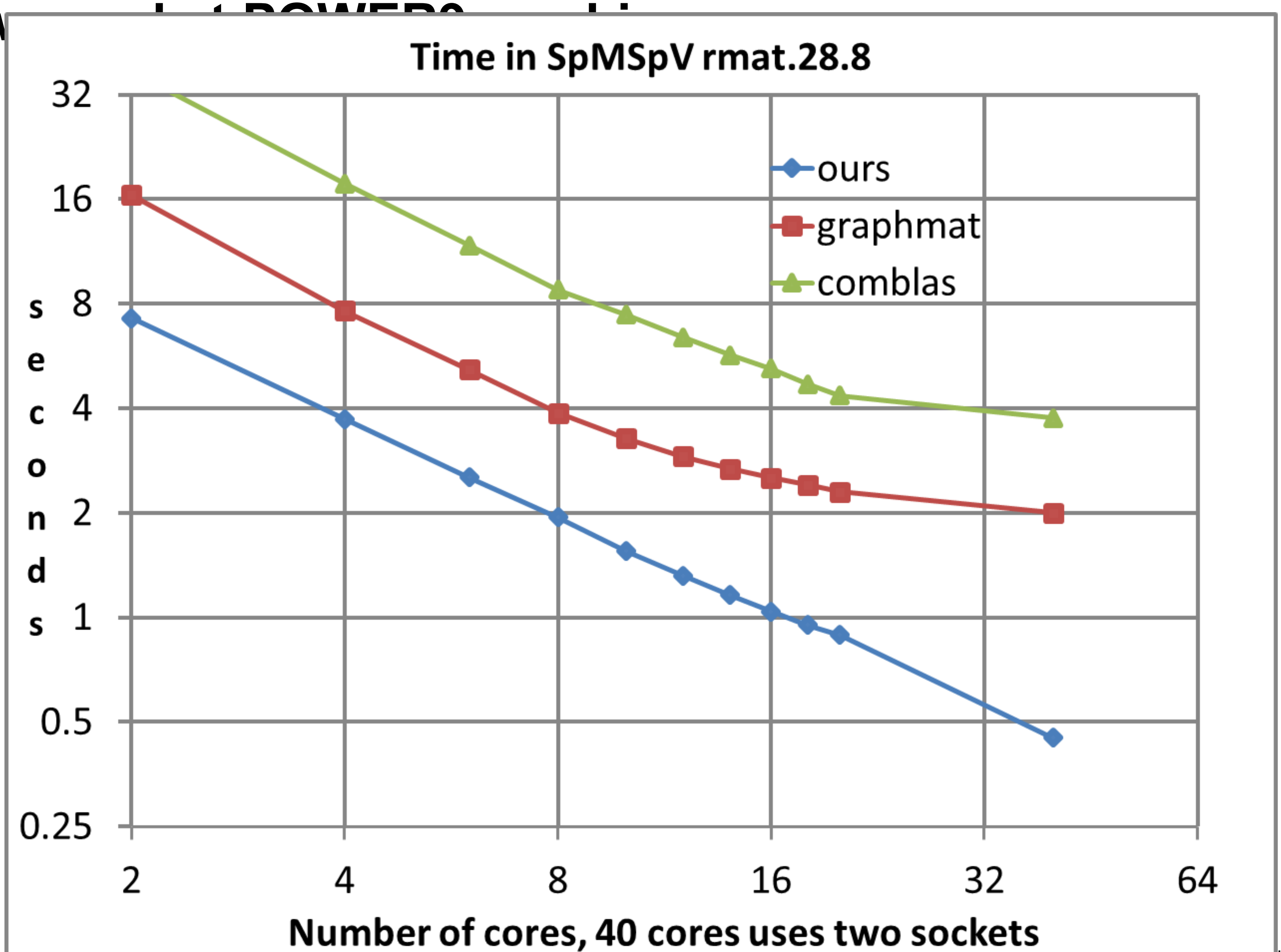
Fig. 5. Choosing SpMV based on nonzero density

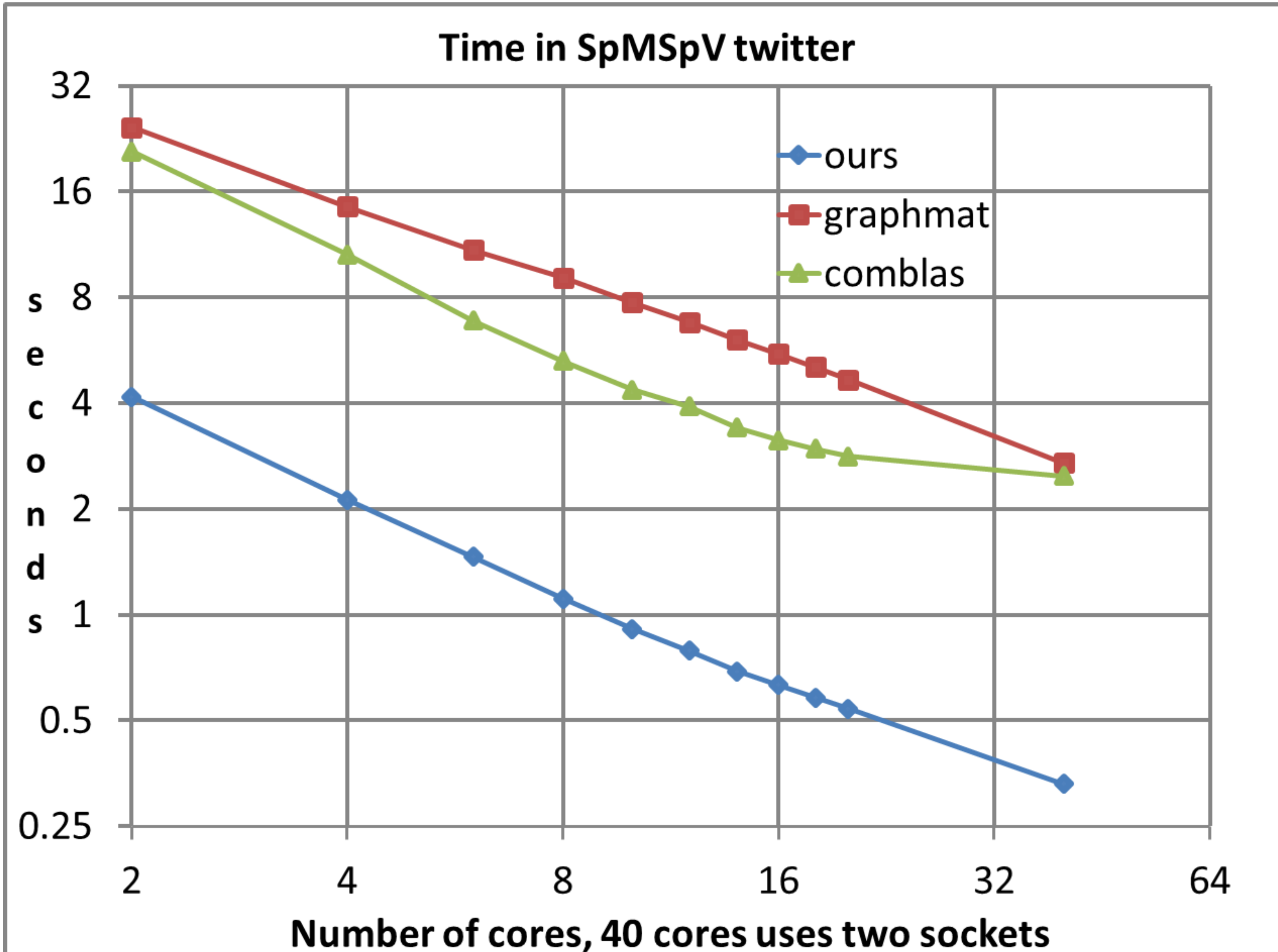
Optimal Thread Bin Size

- **If bucket is too small, frequent transfers to global bin increase synchronization overhead**
- **If bucket is too large: cache footprint exceeds L3 cache size**
- **Optimal size found to be 256 bytes for RMAT 28.8**
  – Bucket counter can be a single byte



- **Our results show from 2x to 5x better performance than COMBLAS and GRAPHMAT when used with an AC922 tw**




Time in SpMSpV twitter
32
16
8
4
2
1
0.5
0.25
seconds
ours
graphmat
comblas
2
4
8
16
32
64
Number of cores, 40 cores uses two sockets

- **Questions ?**

- **Thank You !**

Naïve CSR algorithm

Algorithm 1: Sequential CSR Algorithm.

Input: $A = (rowstart, colidx, val)$: $n \times n$ CSR matrix;
x: input vector.

Output: b: output vector, initialized to 0.

for $i \leftarrow 0$ ***to*** $n - 1$ **do**
    for $j \leftarrow rowstart[i]$ ***to*** $rowstart[i+1] - 1$ **do**
        $k \leftarrow colidx[j]$;
        $b[i] \leftarrow b[i] + (val[j] * x[k])$;