Exploiting GPU Caches in Sparse Matrix Vector Multiplication

Yusuke Nagasaka

Tokyo Institute of Technology

Sparse Matrix

- Generated by FEM, being as the graph data
  - Often require solving sparse linear equation fast
- Iterative method : CG method, BiCG method
  - Level-1 BLAS (Dot product + AXPY)
    - Sequential memory access
  - Sparse matrix vector multiplication (SpMV)
    - Using sparse matrix format
    - Random memory access

      Performance depends on cache hit rate

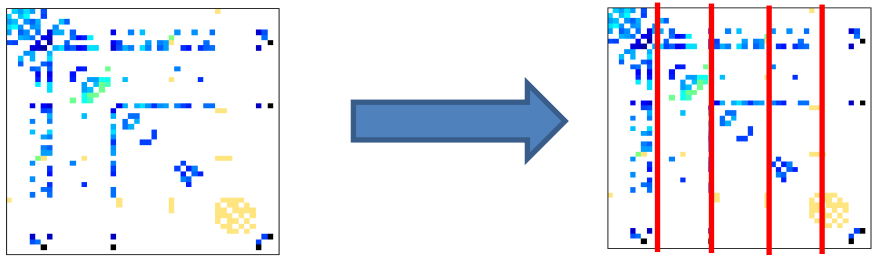
SpMV computation on GPU

- High memory bandwidth and parallelism enable high performance
- Latency is hidden with SMT
- Available cache per thread is small
  - Controlling the cache is difficult
  - => Lower cache hit rate compared to CPU

	Intel Xeon Processor E5-2620 v2	**NVIDIA Tesla K20X**
Cache size	L1 cache : 192KB (instruction / data) L2 cache : 1.5MB, L3 cache : 15MB	Read-only cache : 12KB * 4 / SMX L2 cache : 1.5MB
Max threads	12 threads	28672 threads

Contribution

- We propose a family of cache-aware formats for GPU
  - Segmentation along the column
  - Segmented formats, Non-Uniformly Segmented formats
    - 2 ways of SpMV computation
  - Achieve speedups of up to
    - x2.1 for real datasets and x3.2 for synthetic matrices in SpMV
    - x1.15 in CG

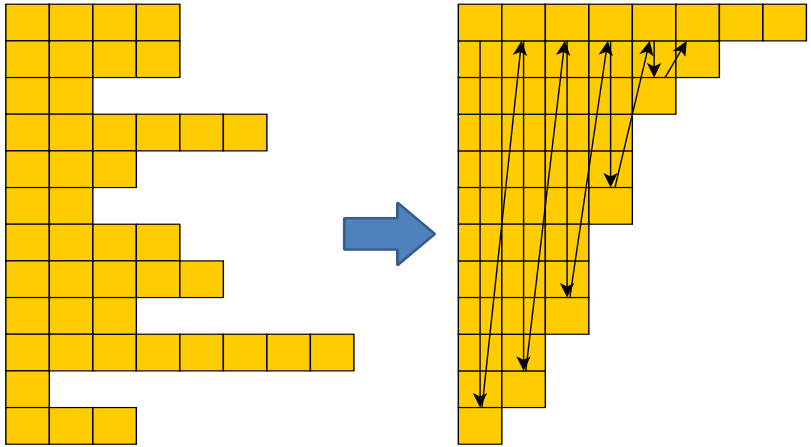


Sparse Format

- Compressing the needless zero elements
  - Reduce memory usage
  - Eg.) COO, CSR
- Efficient memory access to matrix data depends on architecture
  - Vector machine, GPU : column major format
    - JDS, ELLPACK, SELL-C-σ

(Existing Sparse Format)
JDS

- Reordering the rows by the number of non-zero elements per row
  - Generate column major format
    - Favorable for vector machine and many core architectures



(Existing Sparse Format)
SpMV kernel of JDS format

```
__constant__ int jds_ptr[];
__global__ void KernelJDS(float *out,
                          const float* __restrict__ vector
                          int *jds_col, float *jds_val,
                          int  M, int nnz_max) {
   //Calculate i-th row
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int j=0; float answer = 0;   int index = i + jds_ptr[i];
   while (index < jds_ptr[j + 1] && j < nnz_max) {
      answer += jds_val[index] * vector[jds_col[index]];
      index = I + jds_ptr[++j];
   }
   out[i] = answer;
}
```

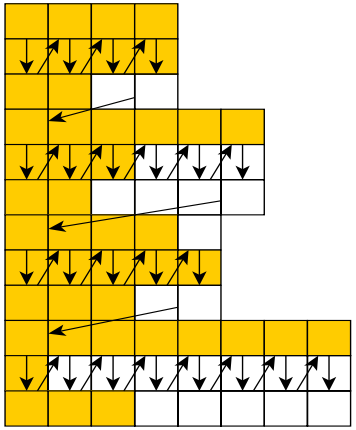
Using Constant cache
when nnz_max * sizeof(float) < 16KB

Read-only cache
for input vector

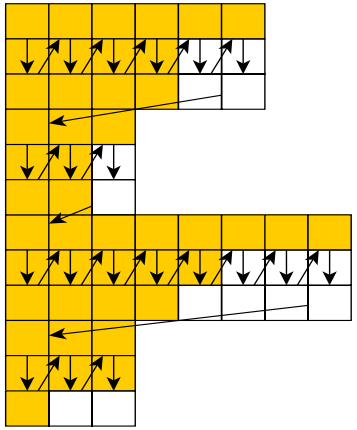
Using inline PTX assembly not to pollute the cache
jds_val[index]=>ld.global.cv.f32
jds_col[index]=>ld.global.cv.s32
out[i]=>st.global.cs.f32

(Existing Sparse Format) SELL-C-σ [Kreutzer, 2013]

- Converting ELLPACK each row block (Sliced ELLPACK)
  - Reduce the zero filling
  - C is block size
    - C = WARP size
- Sorting each σ rows
  - Tradeoff between the zero fill and the cost of sorting



SELL-3-1



SELL-3-6

Cache Hit Rates of Existing Sparse Formats

- NVIDIA Tesla K20X
- Dataset : University of Florida Sparse Matrix Collection
- JDS format
  - Input vector is assigned to read-only cache
  - Coalesced access to matrix data

Matrix	Size	L2 Cache Hit Rate [%]	Read-only Cache Hit Rate [%]
Audikw_1	943,695	82.864	51.420
Crankseg_2	63,838	98.338	66.540
mouse_gene	45,101	99.912	8.298

PROPOSED FORMATS

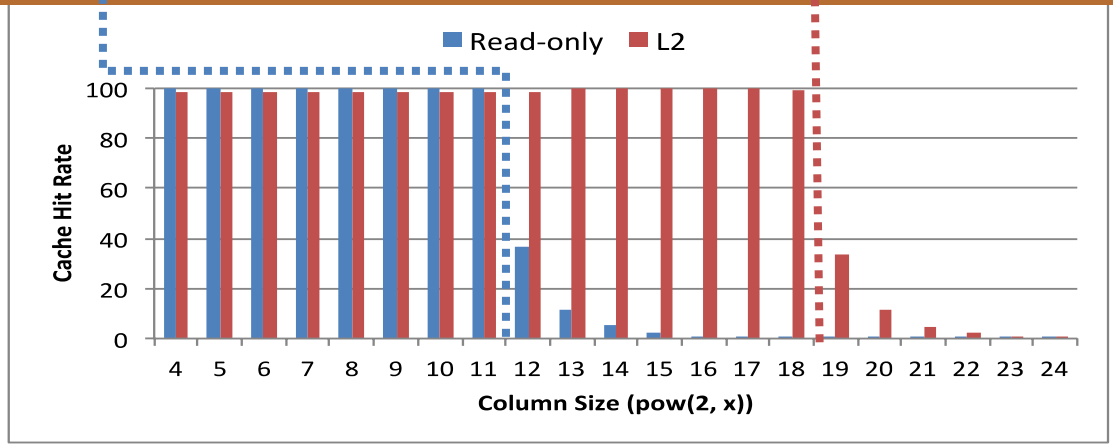
Column size and cache hit rate

Column size where the cache hit rate drops corresponds to each cache size

- Read-only cache : 12KB
- L2 cache : 1.5MB

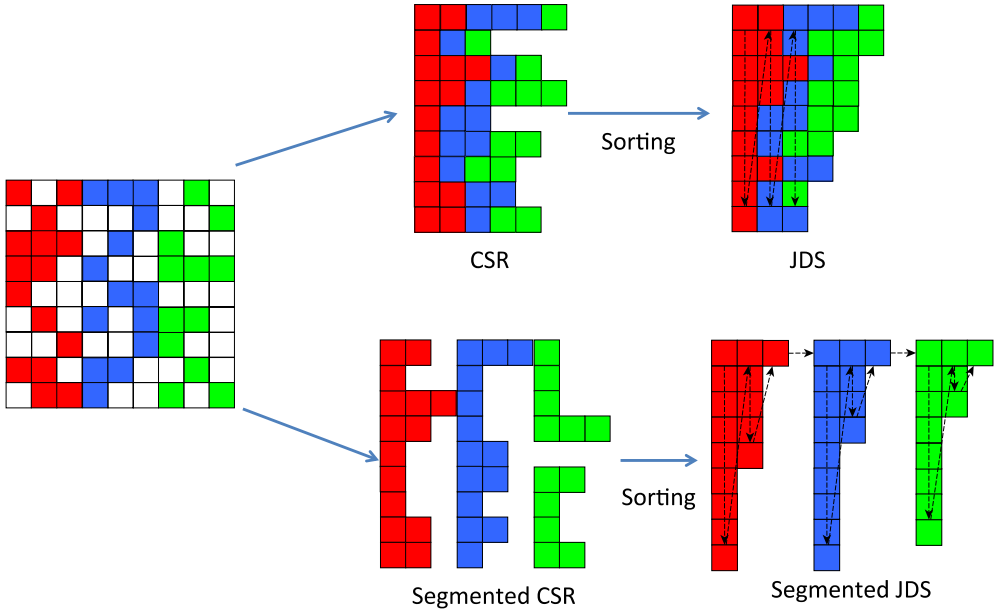
⇒ Segmenting the matrix and the input vector enable to achieve high cache hit rate

- Single precision
- Using JDS format



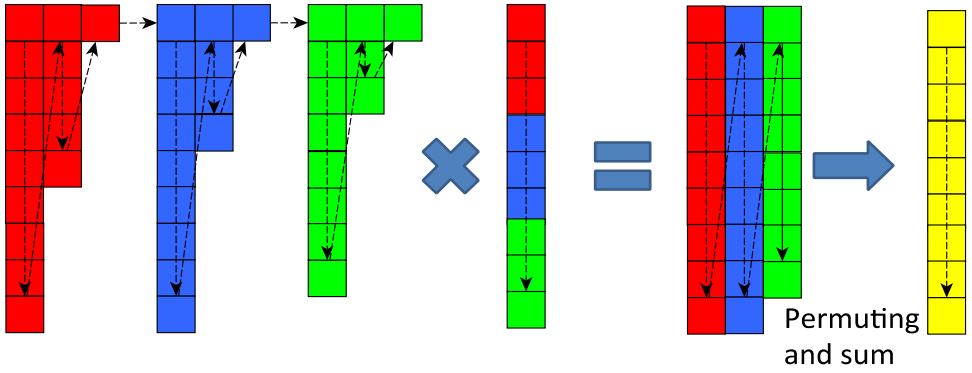
Segmented Formats

- Column-wise segmentation
  - Each segment is converted to JDS or SELL-C-σ



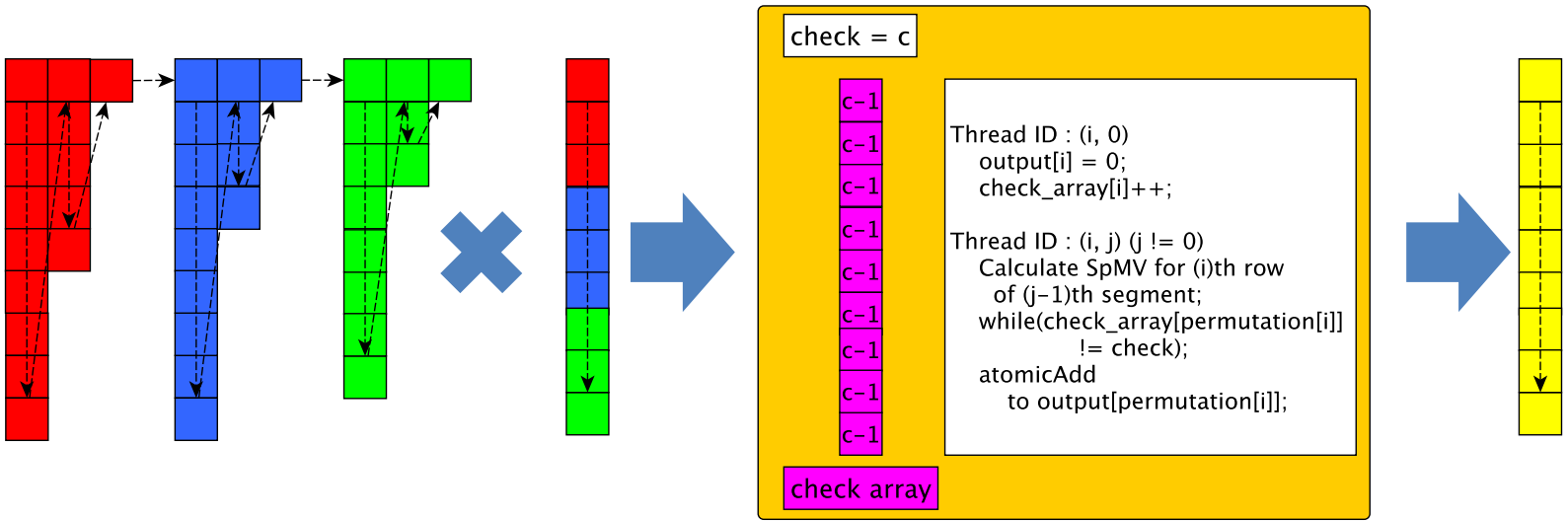
Segmented formats
SpMV Execution

- Two ways of SpMV computation
  - 2 phases computation : Reduce random write
    - 1st phase : Computing SpMV for each sub-matrix and sub-vector, and storing the result into the memory
    - 2nd phase : Accumulation of the intermediate vectors



Segmented formats
SpMV Execution

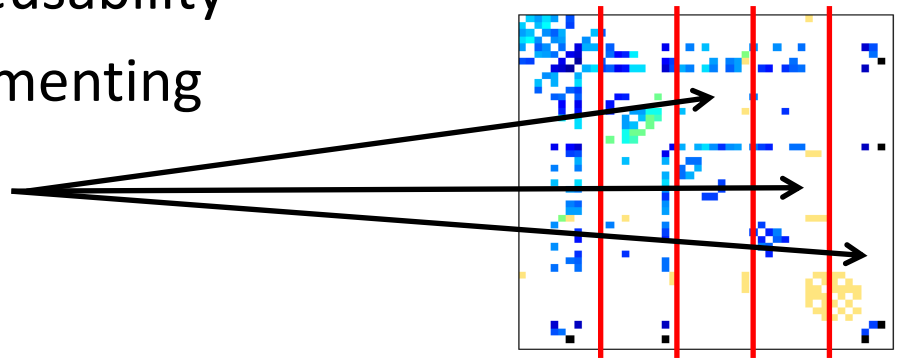
- Two ways of SpMV computation
  - 1 phase computation using atomic operation
  - Prepare additional threads to initialize output vector



Segmented Formats disadvantages

- Increase memory access cost
  - Additional memory access (2 phase SpMV computation)
  - Atomic operation is expensive
- Generate the segments having few non-zero elements
  - Improvement of reusability

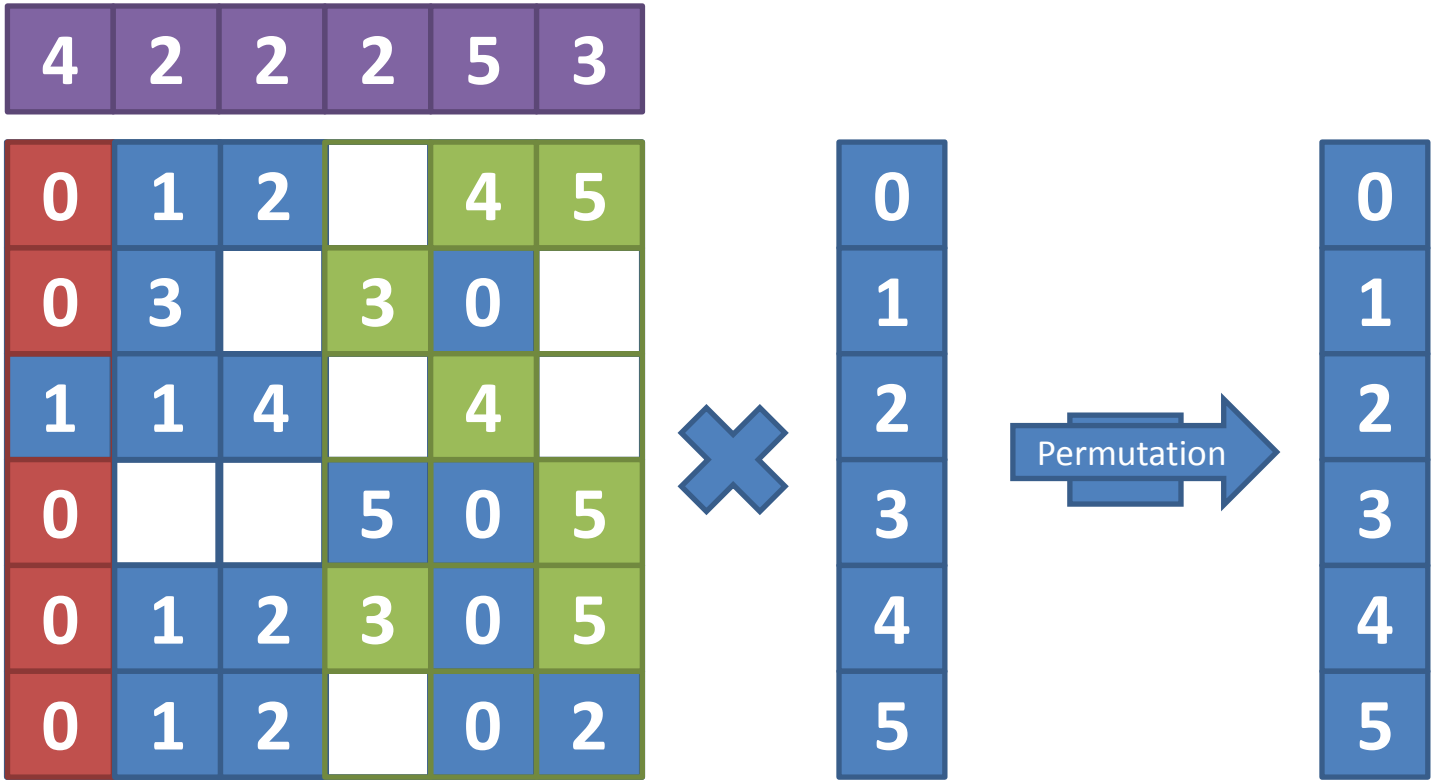
    < Overhead of segmenting
  - => Low efficiency



Non-Uniformly Segmented Formats (NUS Formats)

- Mixing the multi level segmentation size
  - Large segmentation width for the low density area
  - => Reduce the number of segments
- Sorting by the number of non-zero elements per column
  - Set the high density column to left side and high reusability vector elements to the top

Converting NUS Format



Matrix index : column index

Vector index : original row index

- Count # of non-zero elements per column
- Sorting
- Reordering
- Update col index
- Converting to Segmented CSR
- Converting sub-matrix to JDS

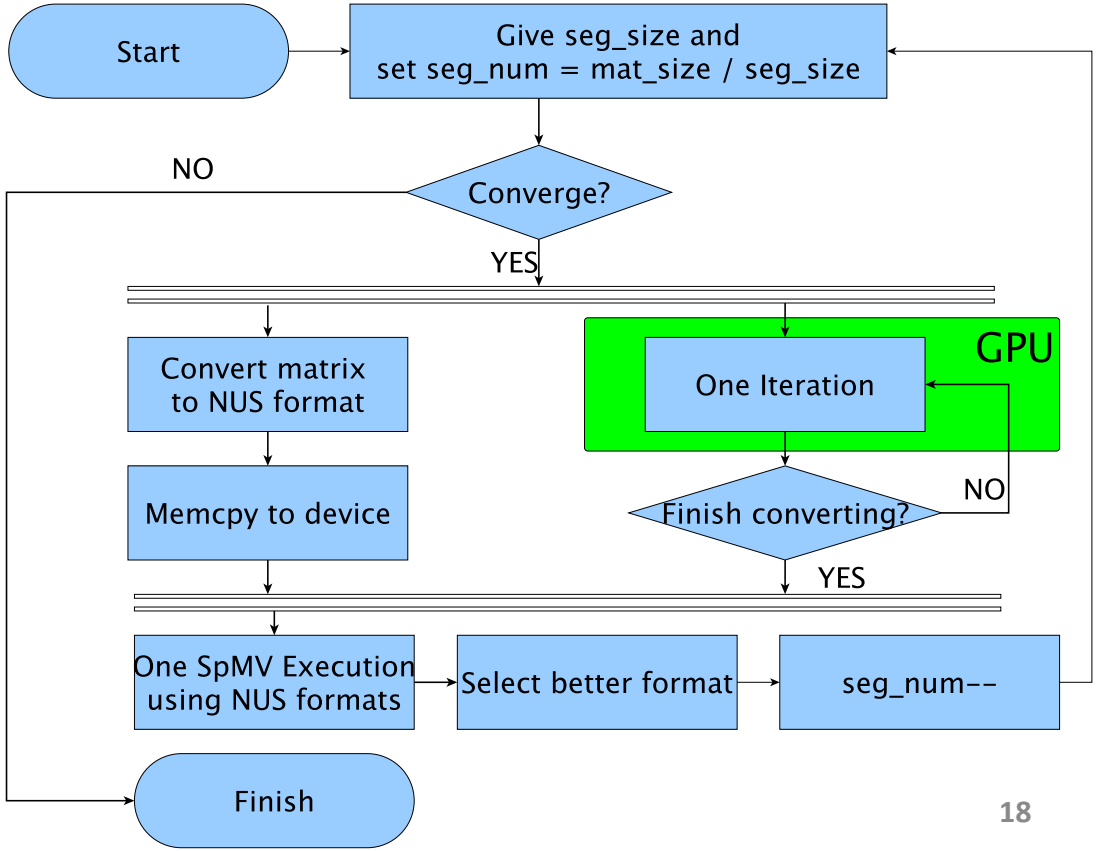
Auto Parameter tuning mechanism for Conjugate Gradient method

- Difficulty of setting parameter
  - NUS formats have 2D parameter space
    - Number of segments (seg_num)
    - Size of segment (seg_size)
- Detection for best parameter in iterative method
  - Time of converting matrix to NUS format

    <<< Duration time until converging

Auto Parameter tuning mechanism for Conjugate Gradient method

- Parallelizing by OMP section
  - CPU : Converting matrix
  - GPU : Executing iteration
- Parameter
  - Giving seg_size
  - Changing # of segments



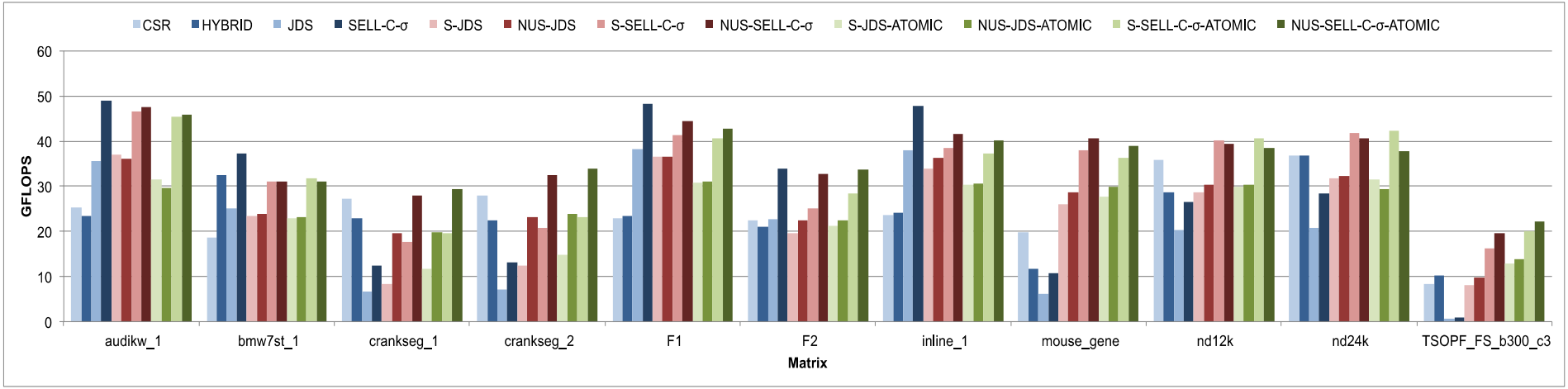
PERFORMANCE EVALUATION

Experiment Environment

- TSUBAME-KFC
  - CPU : Intel Xeon E5-2620 v2 2.10GHz x 2
  - GPU : NVIDIA Tesla K20X x 4
    - Single precision peak performance : 3.95 [TFLOPS]
    - Bandwidth : 250 [GB / sec]
    - Memory size : 6 [GB]
    - L2 cache : 1.5 [MB]
    - Read-only cache : 12 * 4 [KB / SMX]
- CUDA 5.5
- cuSPARSE
  - Provided by NVIDIA
  - CSR format
  - HYBRID format

Performance Evaluation
SpMV (Florida data sets)

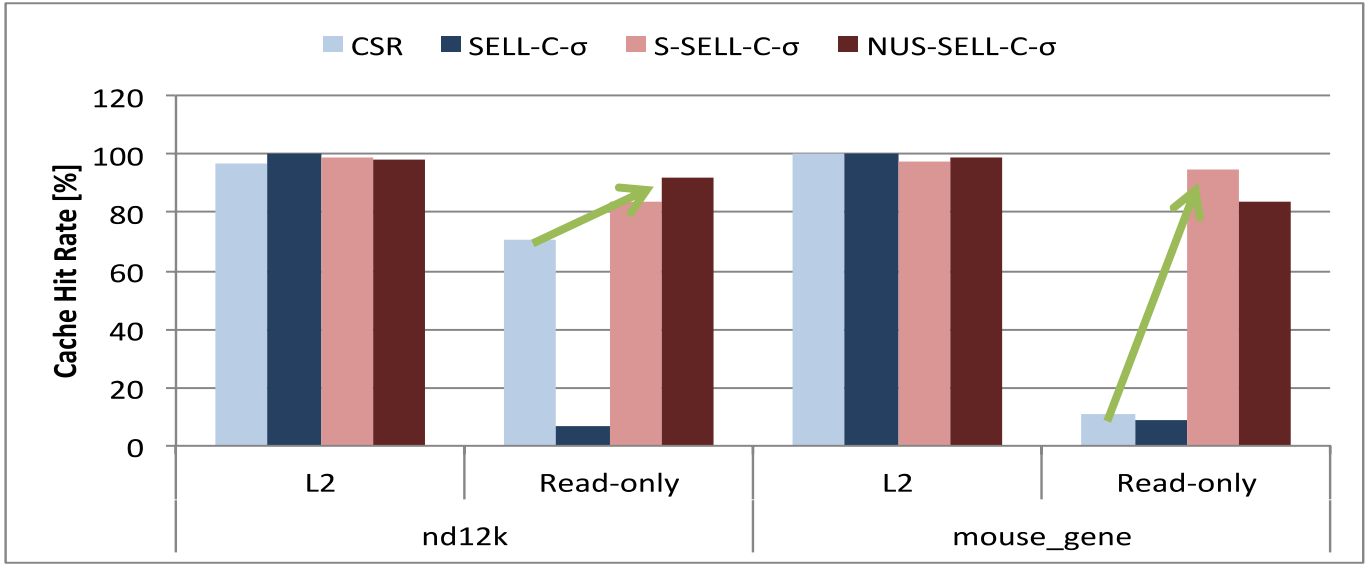
- Our formats show
  - speedup of x0.86 ~ x2.13
  - stable performance



Blue : Existing formats, **Red : Proposal (2 phases ver.)**, **Green : Proposal (Atomic ver.)**

Performance Evaluation
Cache Hit Rate of SpMV

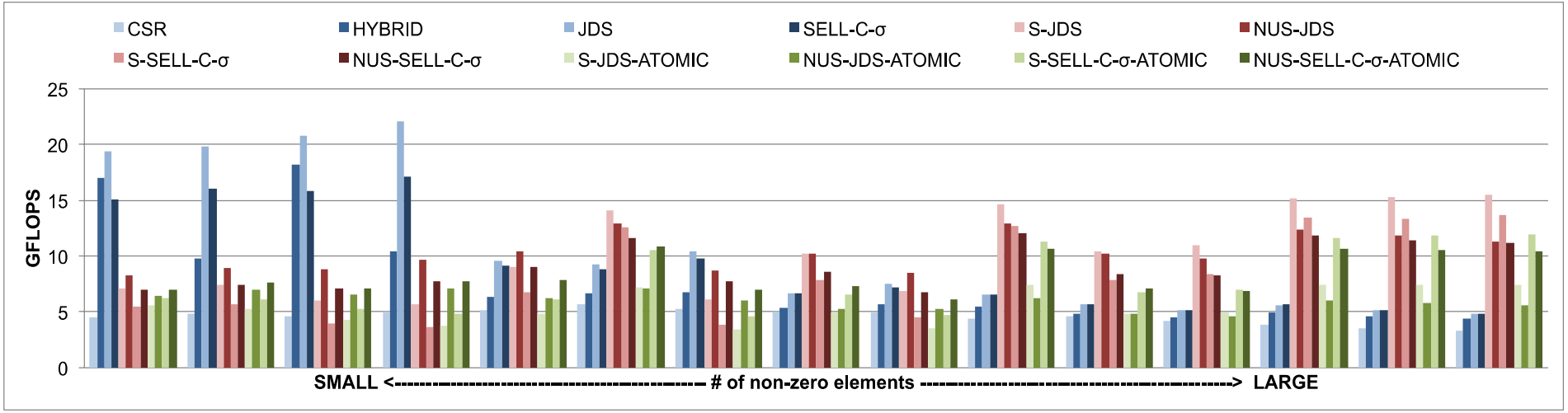
- Segment size suits to read-only cache
  - Improvement of cache hit rate from non-segmented formats



Performance Evaluation
SpMV (Randomly generated matrix)

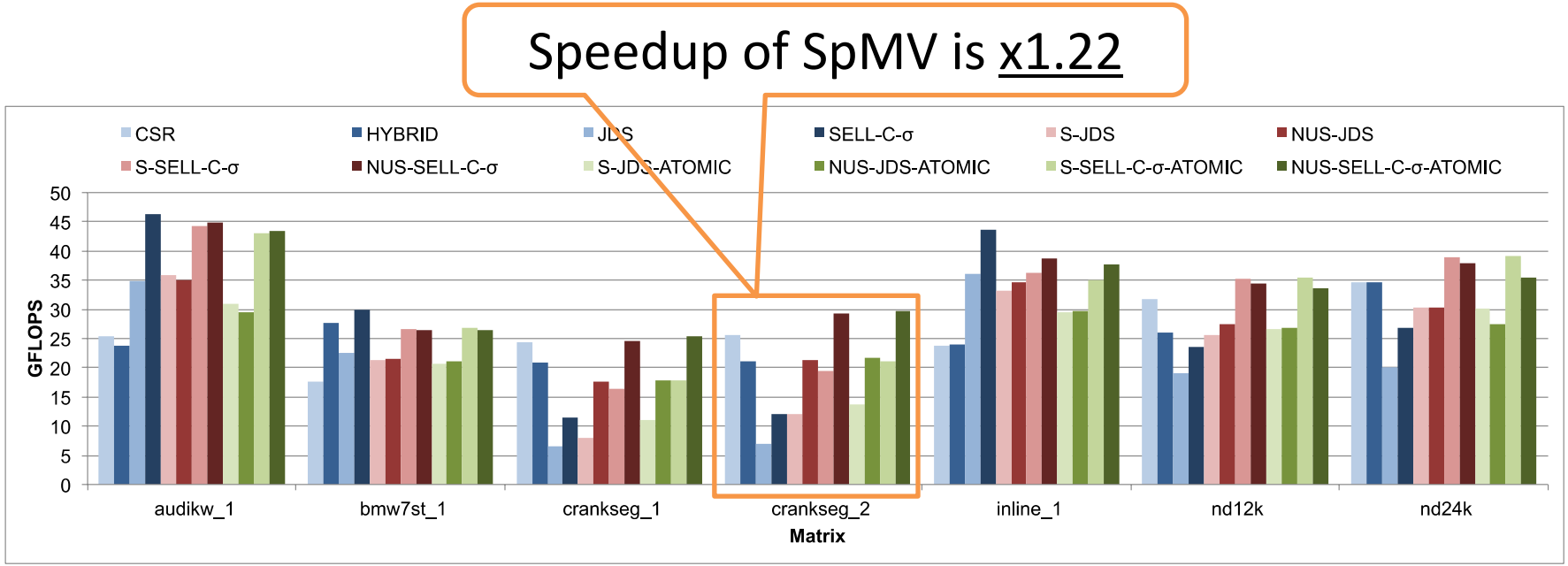
- Investig
  - Nur
  - Nor

Speedup of up to x3.2 and our formats are stable to matrix properties



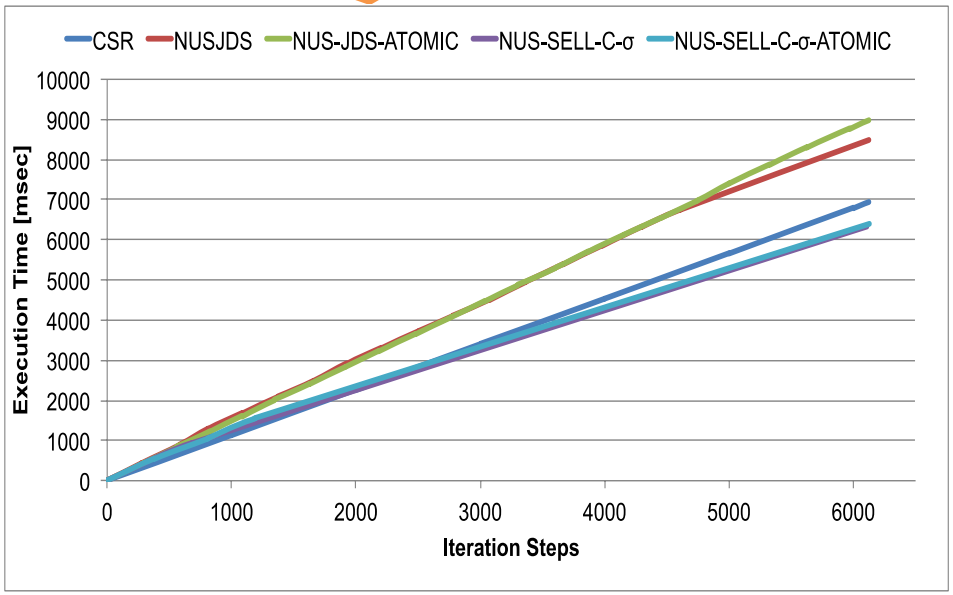
Performance Evaluation
Conjugate Gradient method

- CG computation for positive definite matrices
  - Similar speedup to SpMV ; Up to x1.15

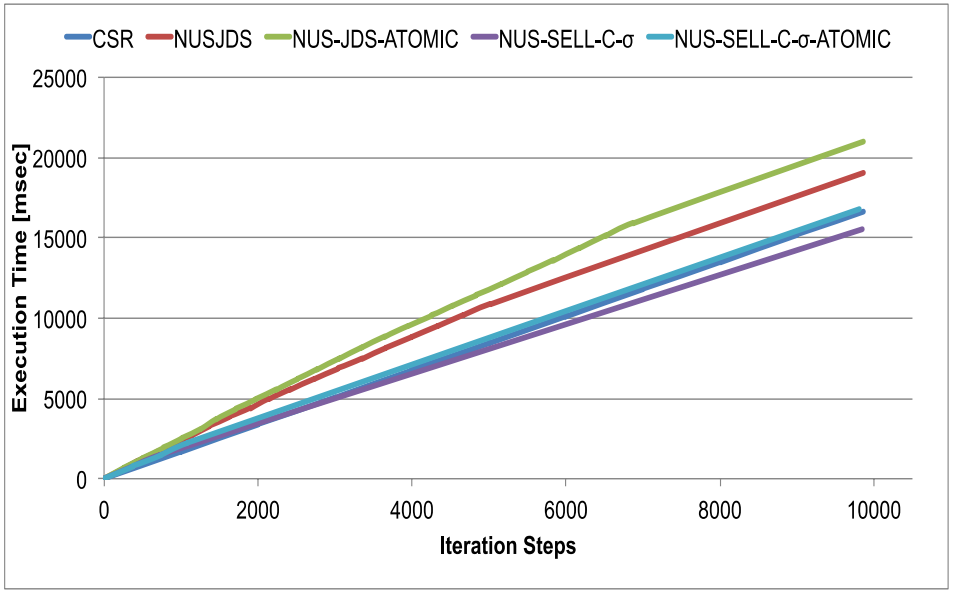


Performance Evaluation
Auto Parameter Tuning CG method

Speedup is x1.09



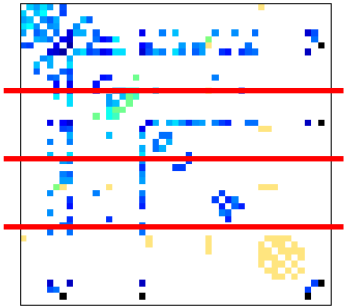
crankseg_2



nd24k

Performance Evaluation
Multi-node CG method

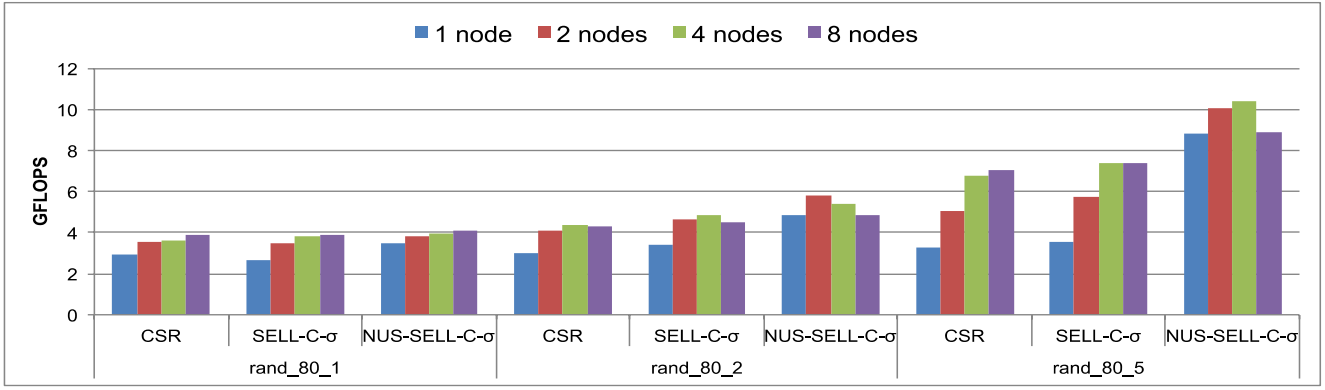
- Strong scaling
  - One GPU for each node
    - Communication between nodes by MPI
    - Send / receive the vector and MPI_Reduce each residual to each iteration
  - Assign row block to each node
    - Each row block has fewer non-zero elements
    - => Cause performance degradation



- Generate larger random matrices; row size is 8M

Performance Evaluation
Multi-node CG method

- NUS-SELL-C-σ shows superiority to CSR and SELL-C-σ
  - Speedup of up to x1.68
  - In lower density matrix, data transfer time between nodes takes relatively longer
    - Performance difference between formats is not noticeable



Features of matrices

- Family of Segmented formats works well for the matrix such that
  - Input vector access is more random
    - Improving the cache hit rate using Segmented formats
  - Matrix has many non-zero elements
    - Achieve high cache reusability
  - Matrix has large variance of the number of non-zero elements per row
    - Reduce idle threads from JDS or SELL-C-σ

Conclusion

- S-Formats and NUS-Formats improve the cache locality and SpMV performance
  - NUS formats achieved speedups of up to
    - X2.1 for real datasets and x3.2 for synthetic matrix in SpMV
    - X1.15 for real datasets in CG

E-mail : nagasaka.y.aa@m.titech.ac.jp