Multi-level Blocking Optimization for Fast Sparse Matrix Vector Multiplication on GPUs

Yusuke Nagasaka, Akira Nukada, Satoshi Matsuoka
Tokyo Institute of Technology

Introduction and Motivation

Many scientific problems involve computation on large sparse matrices. These matrices are stored in special sparse formats to reduce memory usage; various such formats have been proposed for specific architectures and matrix types. Sparse matrix vector multiplication (SpMV) is the most time consuming part in many solvers and the particular choice of sparse matrix format can largely affect the performance of SpMV. Performance can be largely affected by the total amount of memory access including random memory access to input vector. To reduce memory access we devise new sparse matrix format which requires less memory usage.

Proposal

Adaptive Multi-level Blocking (AMB) format

1st Level blocking : **Segment the matrix along columns**

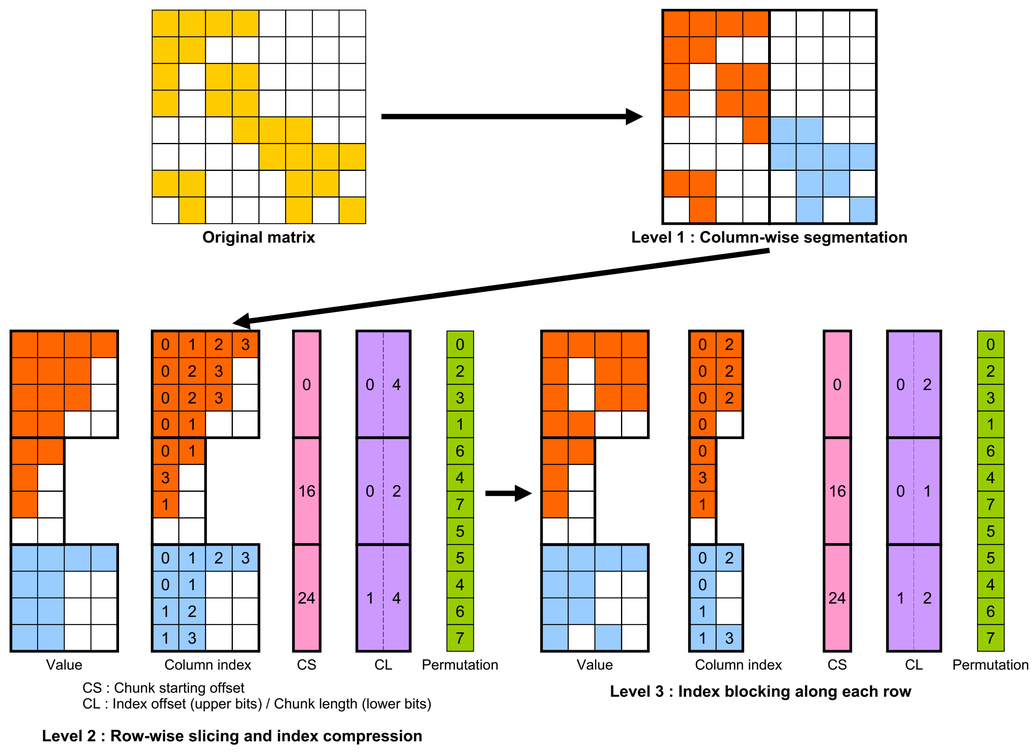
- Locality of the memory access to input vector is improved
- Segmentation width is limited to 64k columns

2nd Level blocking : Reducing memory usage with **row-wise slice**

- Each sub-matrix is converted to SELL-C-σ [1]
- Any Chunk which does not have non-zero elements is removed and column index is represented in 16-bit integer (*unsigned short*) to reduce memory usage.
- Offset of column index is stored to upper bits of cl (chunk length) array.
- In SpMV computation, the output vector is initialized and the result of each row is added using *atomicAdd* (in case of single precision)

3rd Level blocking : The column index is compressed by **blocking along each row**

- Block size : 1, 2, 3, 4, 5, 6, 7, 8
- This not only reduces memory access, but also reduces divergence in iteration by unrolling
- Requires more zero filled for value array



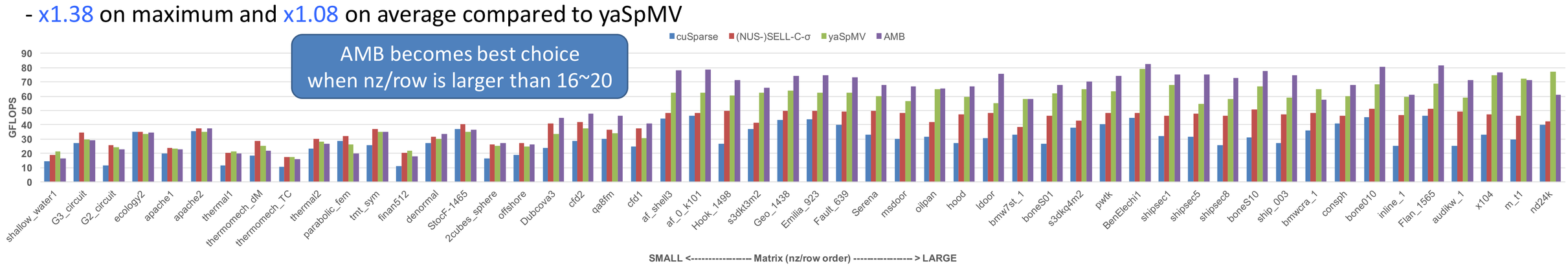
Performance Evaluation

- NVIDIA Tesla K20X GPU (ECC off) on TSUBAME KFC
- Matrix data is taken from the University of Florida Sparse Matrix Collection. 62 selected matrices are positive definite and row size is no less than 64k.
- Compared to cuSparse (CSR, HYBRID, BSR), SELL-C-σ, Segmented-SELL-C-σ, NUS-SELL-C-σ and yaSpMV [2] (BCCOO, BCCOO+)
- All SpMV computation is in single precision

FLOPS Performance

Speedup of AMB format is x1.38 on maximum and x1.06 on average compared to existing sparse format and library

- x2.83 on maximum and x1.75 on average compared to cuSparse
- x1.38 on maximum and x1.08 on average compared to yaSpMV

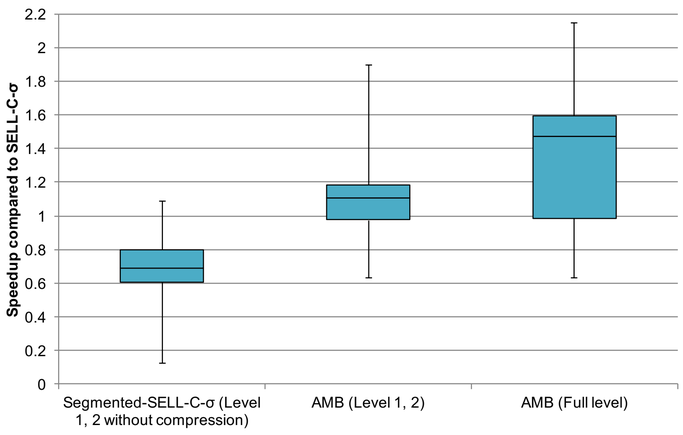


Memory Traffic

The traffic of GPU's global memory in SpMV computation is evaluated by using nvprof. For most of the matrices, the format which requires minimum traffic shows best SpMV performance.

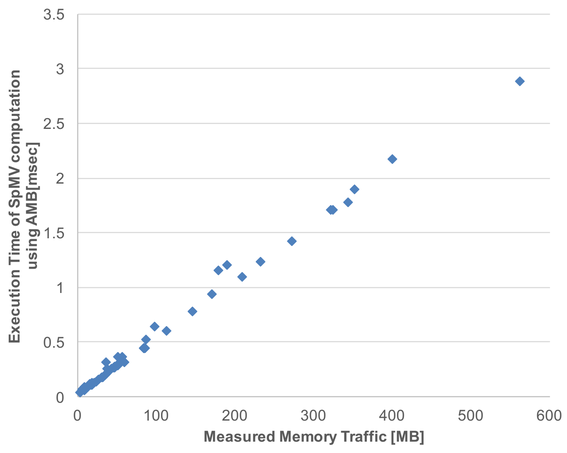
Performance Contribution

Speedups compared to SELL-C-σ show performance gains are the result of column index compression by using 16-bit integer and blocking.



Memory Traffic and Execution time

Basically execution time depends on memory traffic.
Other reasons such as load imbalance causes performance down for some matrices.



Conclusion and Future work

We proposed the novel sparse matrix formats which were designed for reducing the memory access in SpMV computation. AMB format achieves x1.38 on maximum and x1.06 on average performance gain compared to the existing fast SpMV library and other sparse matrix formats.
For future work, we will devise auto-parameter tuning mechanism for AMB format.

Reference

[1] Kreutzer et al., "A Unified Sparse Matrix Data Format for Modern Processors with Wide SIMD Units", CoRR, 2013
[2] Schengen Yan et al. "yaSpMV : Yet Another SpMV Framework on GPUs", PPoPP'14