Exceptional service in the national interest

Sandia National Laboratories

Sparse Matrix-Matrix Multiplication for Modern Manycore Architectures

Mehmet Deveci, Erik Boman,

Siva Rajamanickam

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

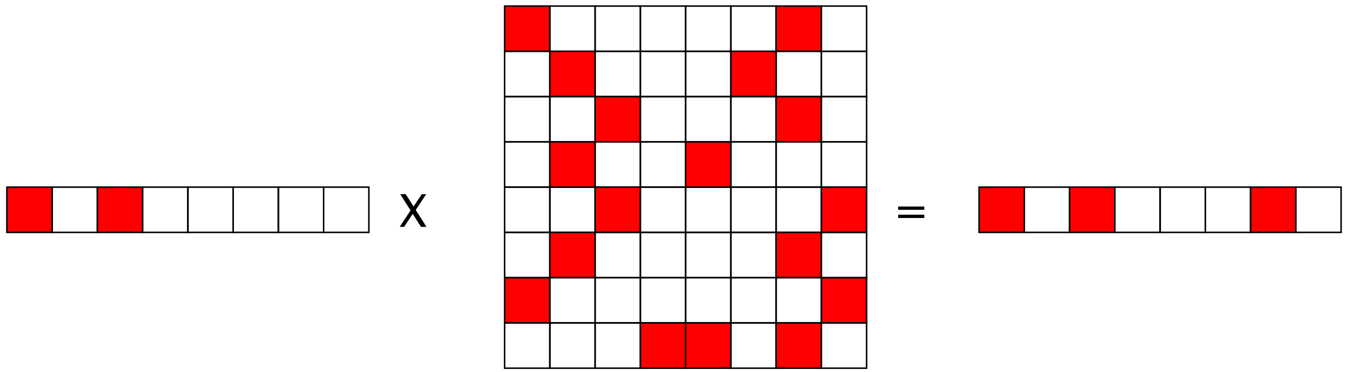
Problem

Sandia National Laboratories

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

- SPGEMM: fundamental block for
  - Algebraic multigrid
  - Various graph analytics problems: clustering, betweenness centrality…
- Extra irregularity: nnz of C is unknown beforehand

Background



- Distributed algorithms:
  - 1D Trilinos
  - 2D Combinatorial Blas [Buluç 12],
  - 3D [Azad 15]
  - Hypergraph-based: [Akbudak 14], [Ballard 16]
- Most of the shared memory algorithms bases on 1D-Gustavson algorithm [Gustavson 78]

Background

- Multi-threaded algorithms:
  - Dense Accumulator (with B column partitions) [Patway 15]
  - Sparse Heap accumulators: ViennaCL, CommBlass
  - Sparse accumulators: MKL
- GPUs:
  - CUSP [Dalton 15]: 3D - outer product (O(FLOPS) memory)
  - Hierarchical: cuSPARSE, bhSparse [Liu 14]
- Aim: Portable methods for GPUs and massively-threaded architectures using Kokkos
  - C++ templated library
  - Abstracting execution, memory spaces, and data layouts
  - Contact: Carter Edwards hcedwar@sandia.gov

Portable SPGEMM Method

- 2-phase, symbolic (calculate #nnz), then numeric (actual flops)
  - Over allocation is expensive or dynamic increase are not suitable on GPUs. Estimations [Cohen 98] are still not an upperbound.
  - It is common in scientific computing where multiplication is repeated for different numeric values with same symbolic structure
- Speedup symbolic with compression:
  - Symbolic phase performs unions on rows, which consists of binary relations
  - Compress the rows of B: O(nnz(B)) using 2 integers.
    - Column Set Index (CSI): represents column set index
    - Column Set (CS): the bits represent the existence of a column
  - Symbolic complexity: O(FLOPS) -> on average ~O(avgdeg(A)x nnz(B))

6	7	8	9	10	33	34	35	36	37

CSI
0
2

CS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

KokkosKernels (KK) - SPGEMM

- Each team works on a bunch of rows of C (or A)
  - Team: Thread block (GPU)

    group of hyper-threads in a core (CPU)
- Each worker in team works on consecutive rows of C
  - Worker: Warp (GPUs), hyperthread (CPU)
  - More coalesced access on GPUs,
  - better L1-cache usage on CPUs.
- Each vectorlane in a worker works on a different multiplications within a row:
  - Vectorlane: Threads in a Warp (GPUs), vector units (CPU)

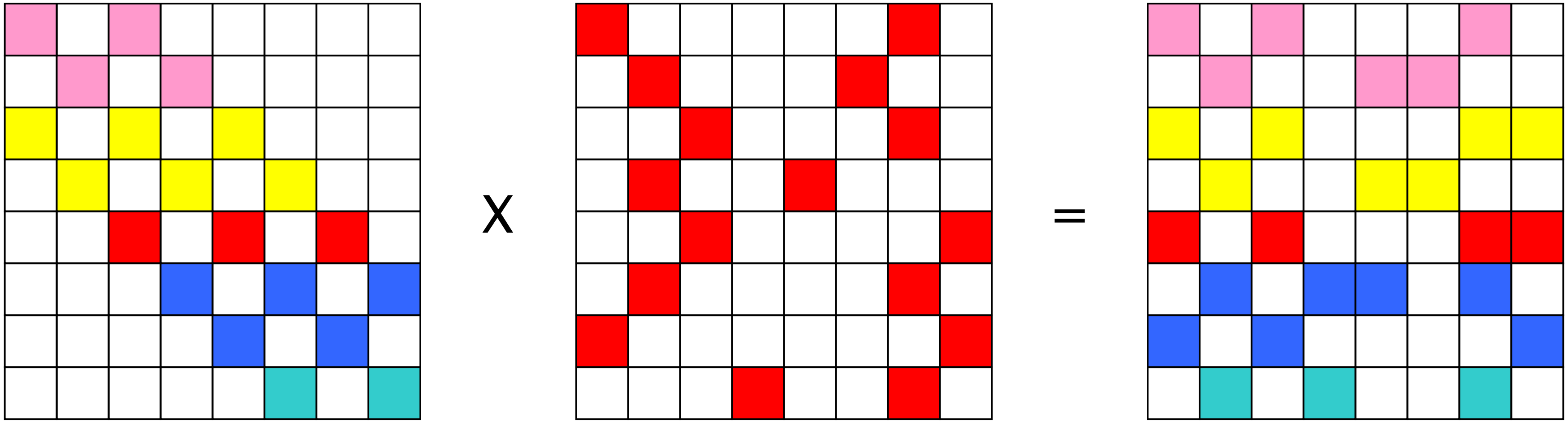
KK - SPGEMM

- Implemented 4 methods
  - KKMEM: Memory efficient
    - Uses sparse hashmap accumulators and memory pools
  - KKSPEED:
    - Dense accumulators on CPU
  - KKMCR
    - Graph coloring variant - 1
  - KKMCW
    - Graph coloring variant - 2

KKMEM

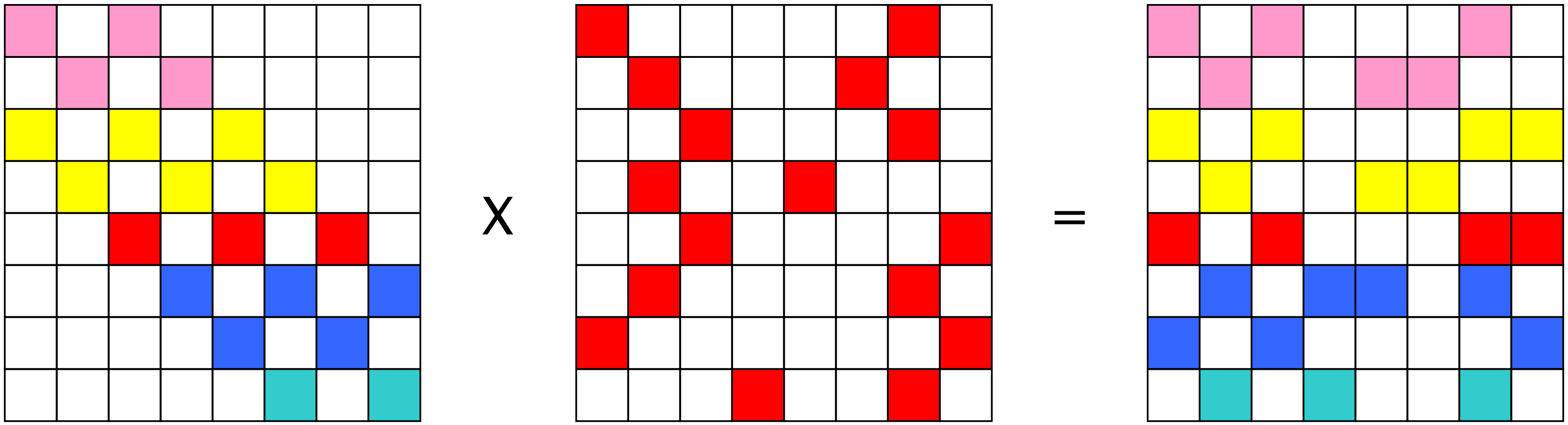
- Hierarchical 1D Gustavson Algorithm
  - Features to make it thread scalable
- 2 level Hashmap Accumulator:
  - 1st level uses scratch space:
    - GPUs shared memory
    - Small memory that will fit in L1 cache on CPUs
  - 2nd level goes to global memory
- Memory Pool:
  - Only some of the workers need 2nd level hash map.
  - Request memory from memory pool.

Distance-2 Graph Coloring



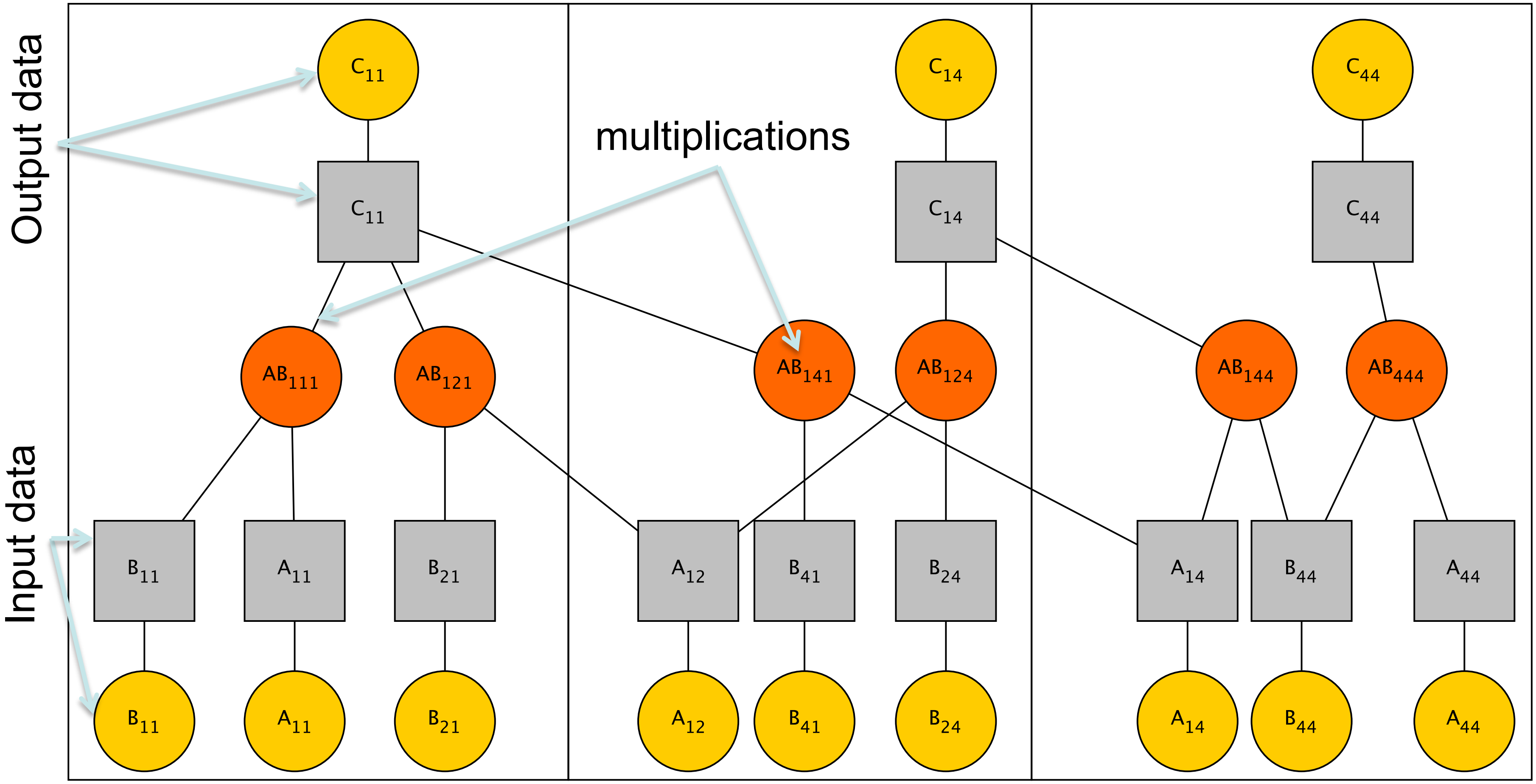
- Distance-2 coloring on the structure of C in symbolic phase
  - Dense accumulator per color
  - Coloring on C is more restrictive coloring on A
    - It is also distance-2 coloring on A
      - The rows of A do not share any column (!)
    - No reuse of rows of B

Distance-2 Graph Coloring



- Distance-2 coloring on the structure of C in symbolic phase
  - Dense accumulator per color
  - Coloring on C is more restrictive coloring on A
    - No reuse of rows of B
- Improve by using multiple colors at a time=nnz(C) / numcols(C)
  - MCR: Permute rows within multicolors – better reads
  - MCW: Permute rows within single colors – better writes

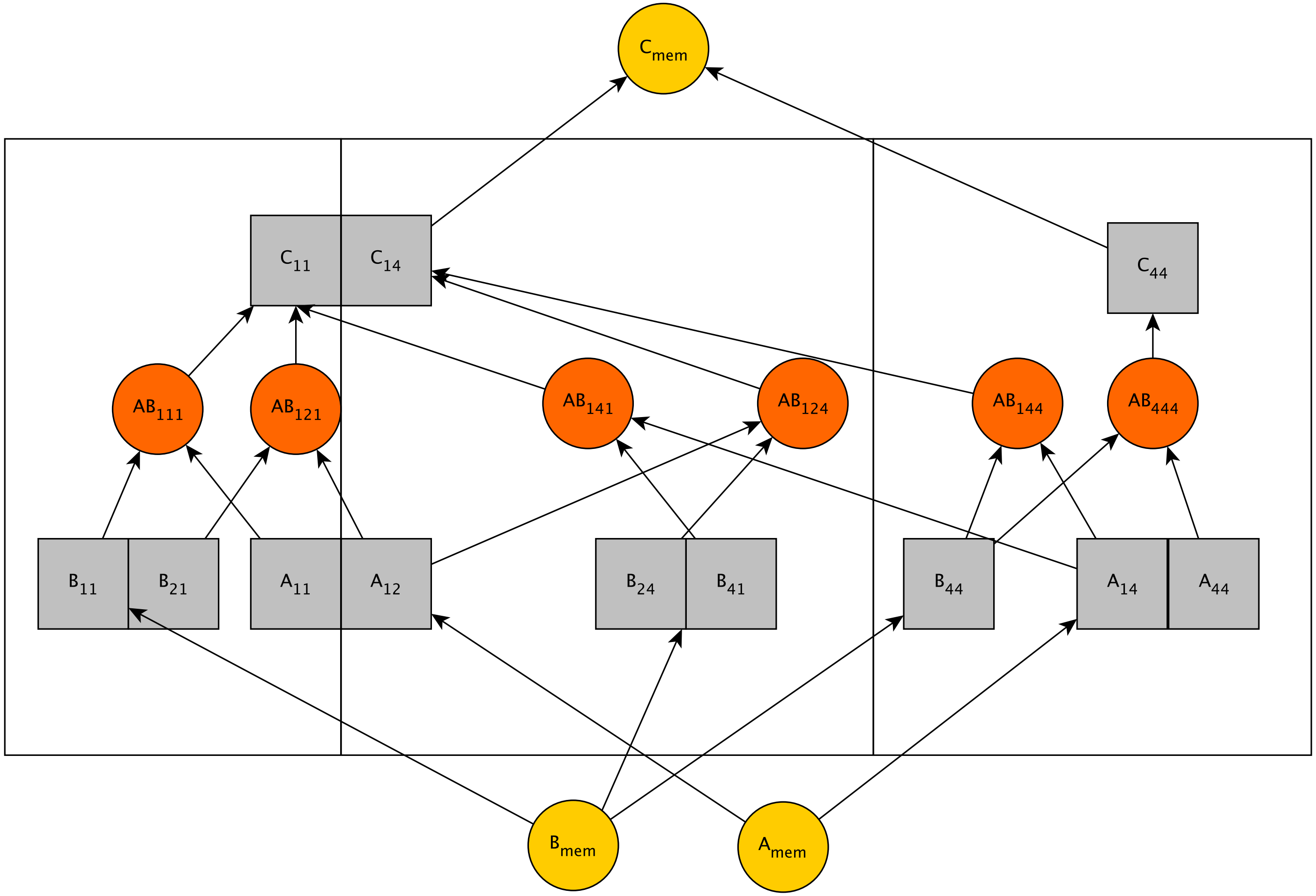
Hypergraph Model [Ballard 15]



- $W_{computation}$= 1 for red vertices, 0 for yellow
- W_{memory} = 0 for red vertices, 1 for yellow

SHMEM Directed HG Model

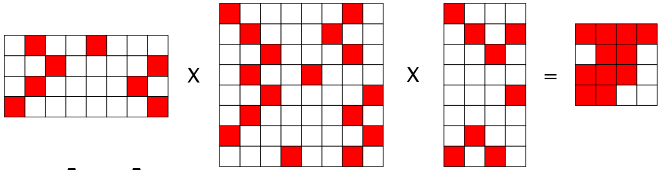
Sandia National Laboratories



- No owners of the data, data lies in the memory (part k+1)
  - There are no messages exchanged between parts
  - Instead incoming/outgoing arrows correspond reads/writes
- Merge nets for data that lives in the same cache line, or range of coalesced accesses
- We use the model to evaluate the read/write of algorithms

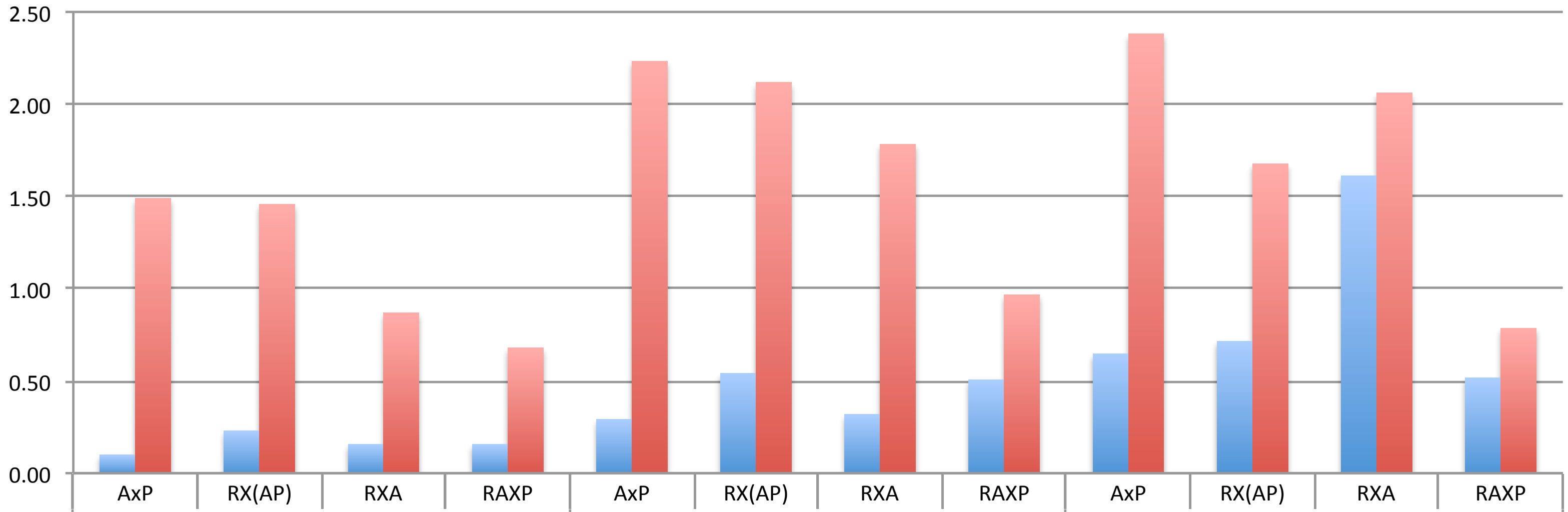
Experiments

- Experiments on matrices
  - Laplace3D (15M, 109M), Brick (15M, 418M) and Empire (2M, 303M)(Internal Sandia App.)
    - Multiplications for multigrid solver in the form
      - $A_{coarse} = R_{restriction} \times A_{fine} \times P_{prolongation}$
      - RxA, RAxP, AxP RxAP



  - Some matrices used in the literature for AxA
- Bowman and Hansen Clusters
  - Bowman: Intel KNL
    - 68 cores, 1.40 GHz, 4 hyper-threads per core.
    - 16 Gb HBW MCDRAM (476.2 GB/s), 96 GB DDR4 (84.3 GB/s)
  - Hansen: NVIDIA Tesla K80
    - CC 3.7 and 11.25 GB memory

GPU Gflops for RxAxP

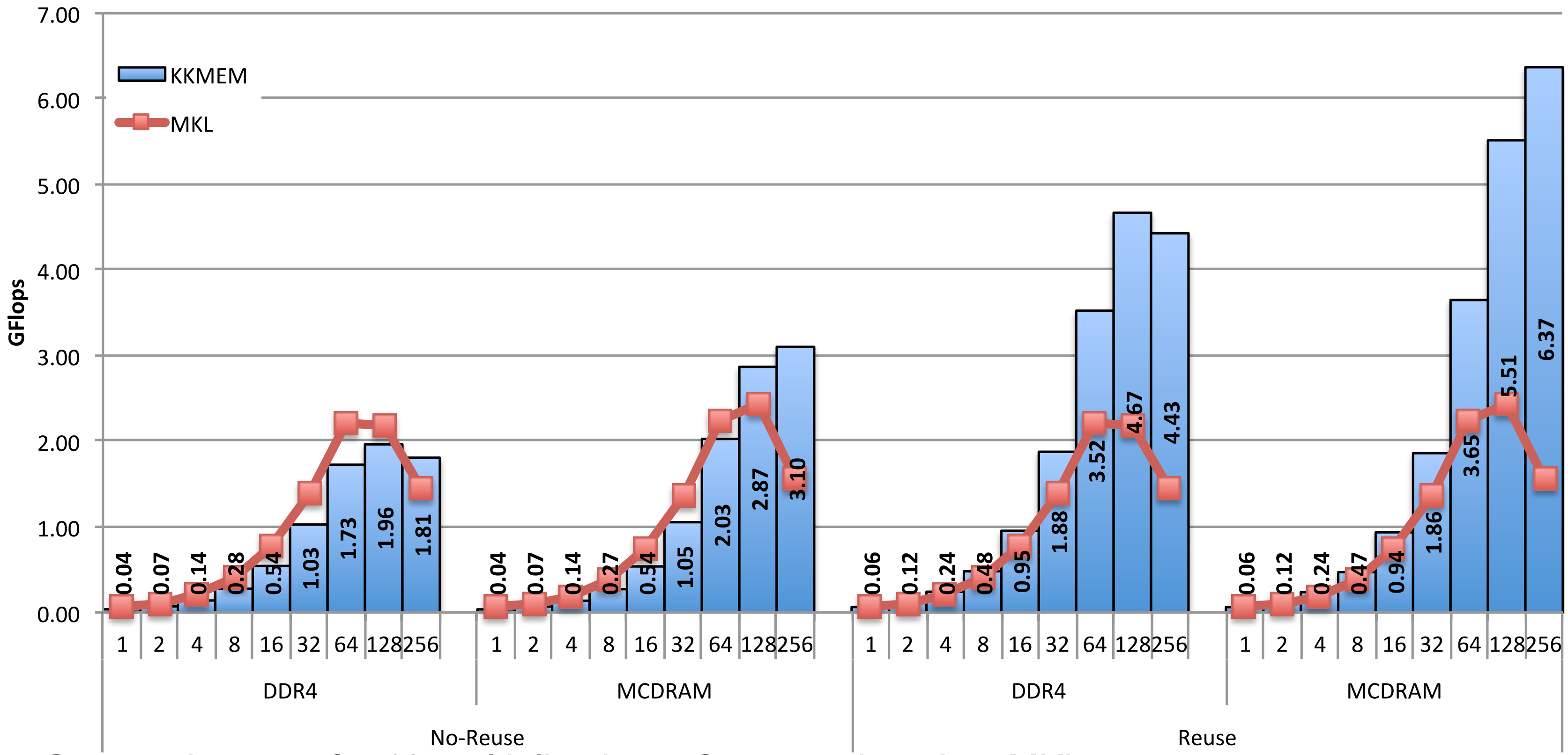


	AxP	RX(AP)	RXA	RAXP	AxP	RX(AP)	RXA	RAXP	AxP	RX(AP)	RXA	RAXP
	Laplace				Brick				Empire			
CUSPARSE	0.10	0.23	0.16	0.16	0.29	0.54	0.32	0.51	0.65	0.71	1.61	0.52
KKMEM	1.49	1.46	0.87	0.68	2.23	2.12	1.78	0.97	2.38	1.68	2.06	0.79

Higher is better

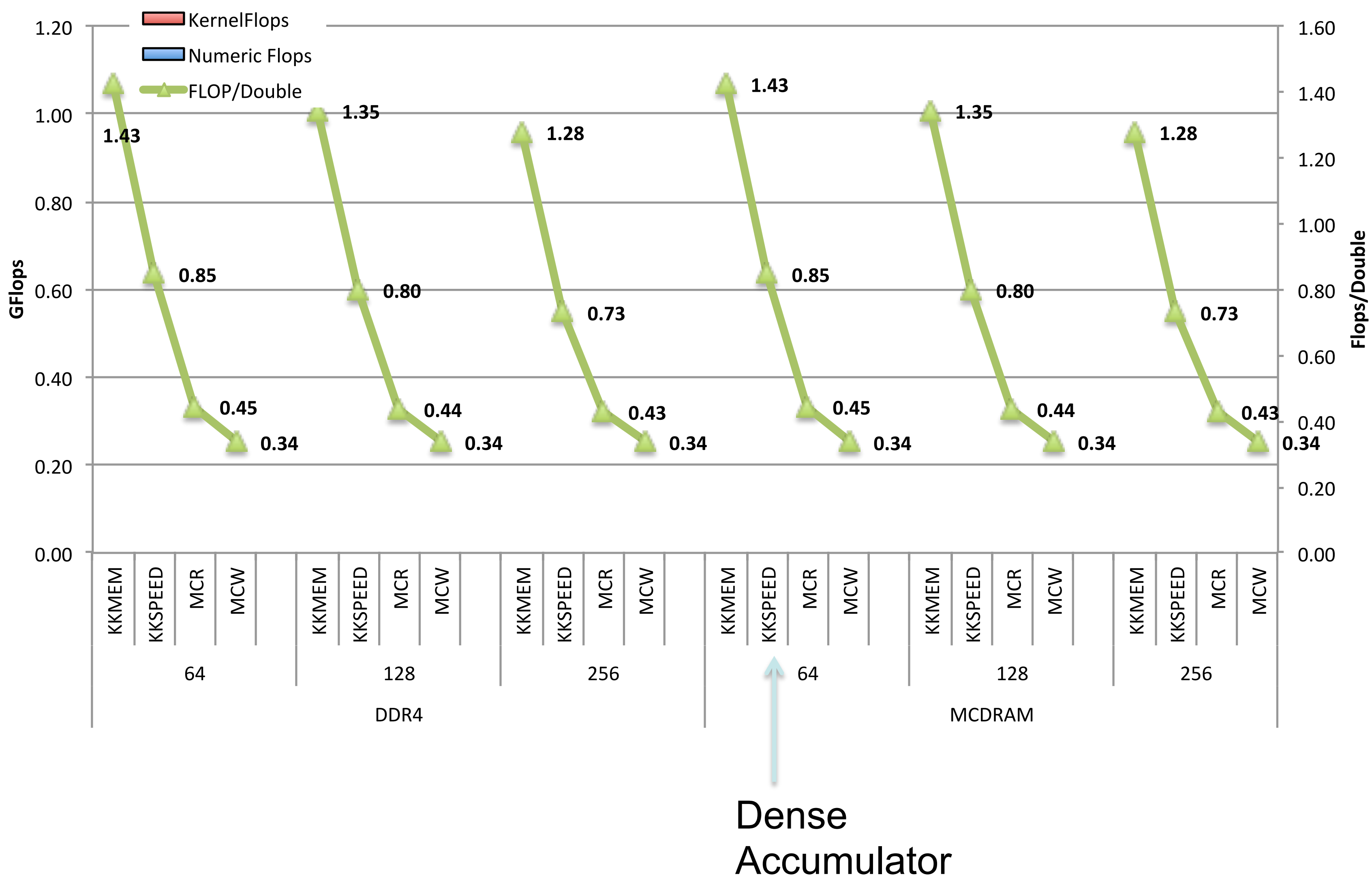
- CUSP runs out of memory
- Speedups range from 1.28 to 14.83. Average 3.90

KNL Experiments

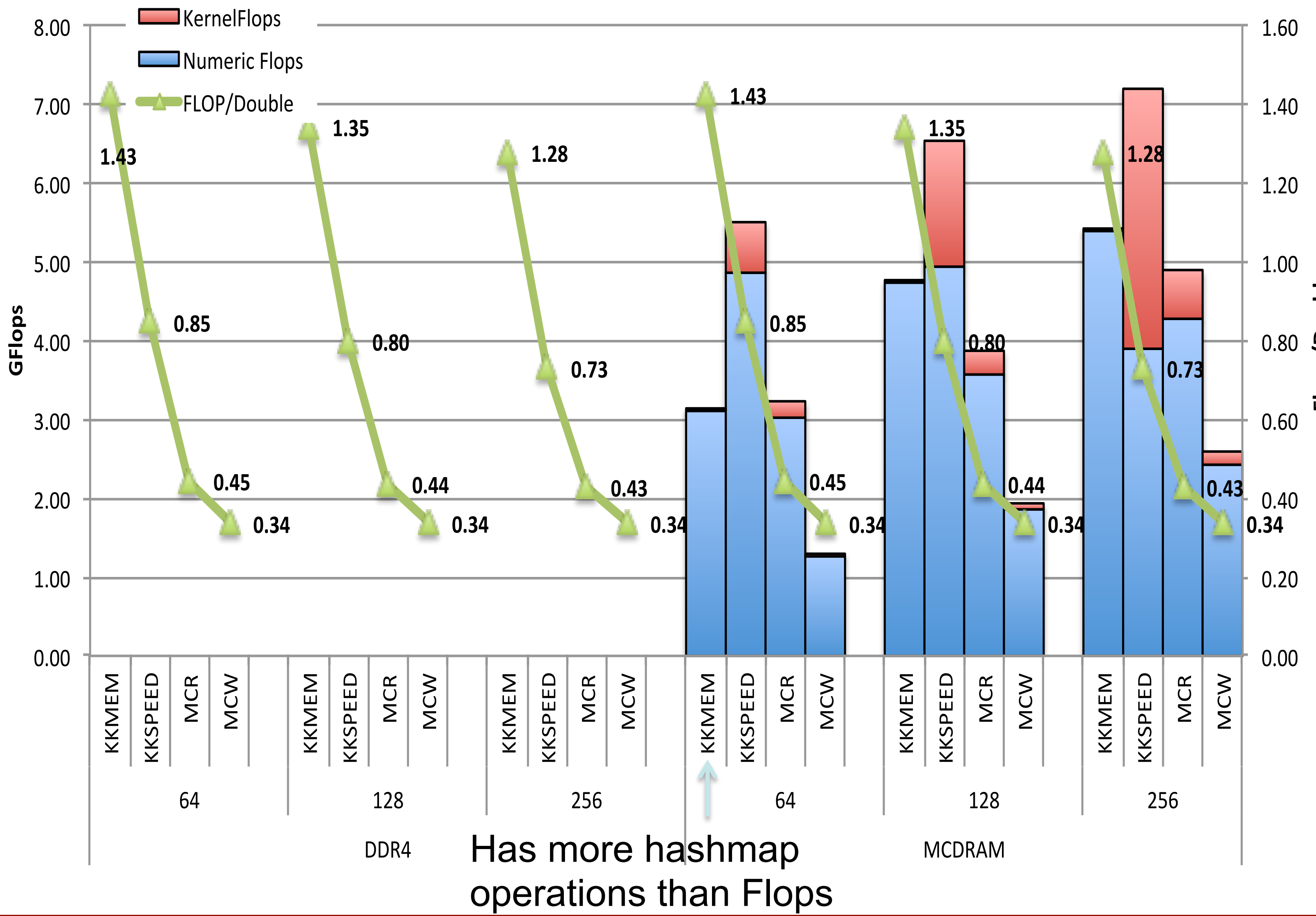


- Geometric mean for 13 multiplications. Compared against MKL.
  - First MKL run takes 4-5x times more than the next ones. First one is excluded.
- Overall: almost linear scaling up to 64 cores.
  - MKL is slightly faster up to 64 cores – no performance diff for MCDRAM and DDR4 (!).
    - KKMEM is 1.17 times faster on 128 threads MCDRAM,
    - MKL does not scale on 256 threads
  - If reuse 2.12 - 2.25 on 1-128 threads (3.05, 4.08 on 256 threads) times faster.
  - The difference between reuse vs no-reuse is high.
  - Compression reduces the size 7-20 % for RxAxP, while it can reduce 87% for UFL matrices

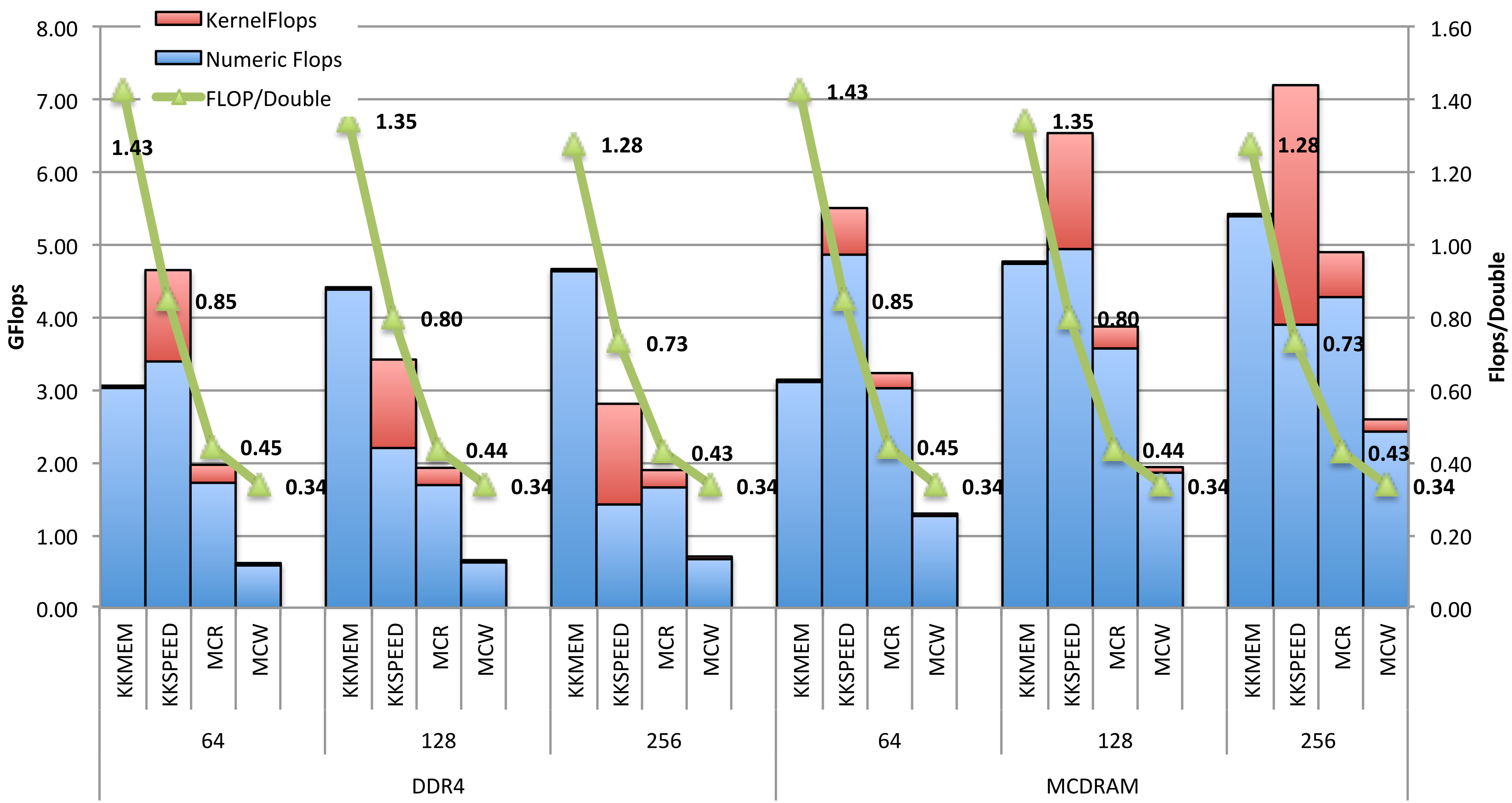
Flop per Double Laplace AxP



Laplace AxP MCDRAM



Laplace AxP DDR4



Conclusions & Future Work

- Portable SPGEMM method with decent performance on various new architectures
- Hypergraph model to study the effect of read/writes to the overall performance
- Ongoing:
  - Analyzing flop per read and flop per write and experiment with MCDRAM and DDR4.
- Future:
  - Fast packing of columns of B for better compression
  - Fast reordering of rows of A to use better locality

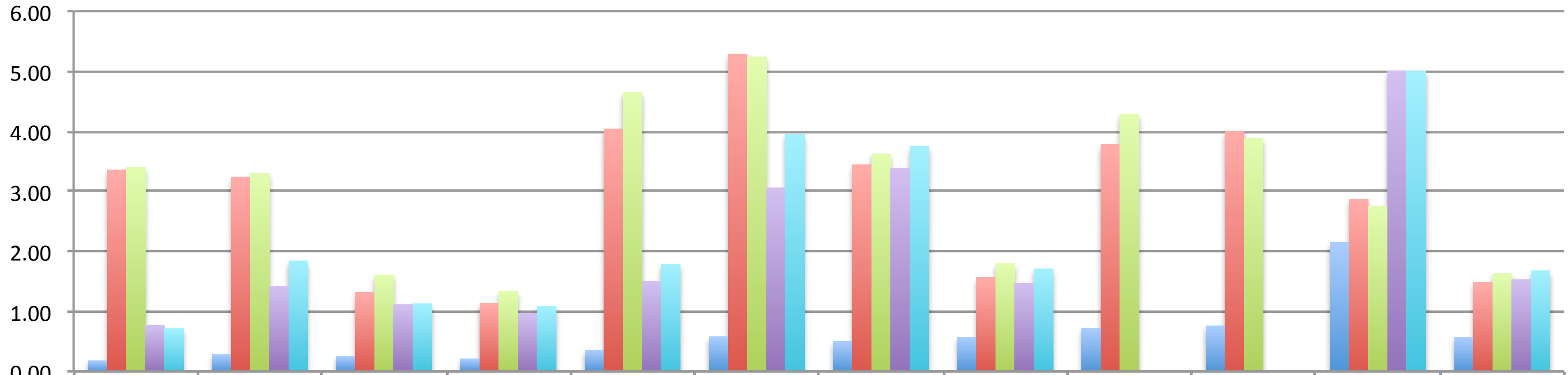
For more information

- KokkosKernels:
  - Download through Trilinos: http://trilinos.org
  - Public git repository: http://github.com/trilinos
- For more information:
  - mndevec@sandia.gov
- Thanks to:
  - NNSA ASC program
  - DOE ASCR SciDAC FASTMath Institute
  - ATDM

References

- F. G. Gustavson, “Two fast algorithms for sparse matrices: Multiplication and permuted transposition," ACM Transactions on Mathematical Software (TOMS), vol. 4, no. 3, pp. 250{269, 1978.
- Buluç, Aydin, and John R. Gilbert. "Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments." *SIAM Journal on Scientific Computing* 34.4 (2012): C170-C191.
- Azad, Ariful, et al. "Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication." *arXiv preprint arXiv:1510.00844* (2015).
- Akbudak, Kadir, and Cevdet Aykanat. "Simultaneous Input and Output Matrix Partitioning for Outer-Product--Parallel Sparse Matrix-Matrix Multiplication." *SIAM Journal on Scientific Computing* 36.5 (2014): C568-C590.
- Ballard, Grey, et al. "Brief announcement: Hypergraph partitioning for parallel sparse matrix-matrix multiplication." *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*. ACM, 2015.
- Patwary, Md Mostofa Ali, et al. "Parallel efficient sparse matrix-matrix multiplication on multicore platforms." *International Conference on High Performance Computing*. Springer International Publishing, 2015.
- Liu, Weifeng, and Brian Vinter. "An efficient GPU general sparse matrix-matrix multiplication for irregular data." *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014.
- Dalton, Steven, Luke Olson, and Nathan Bell. "Optimizing sparse matrix—matrix multiplication for the gpu." *ACM Transactions on Mathematical Software (TOMS)* 41.4 (2015): 25.

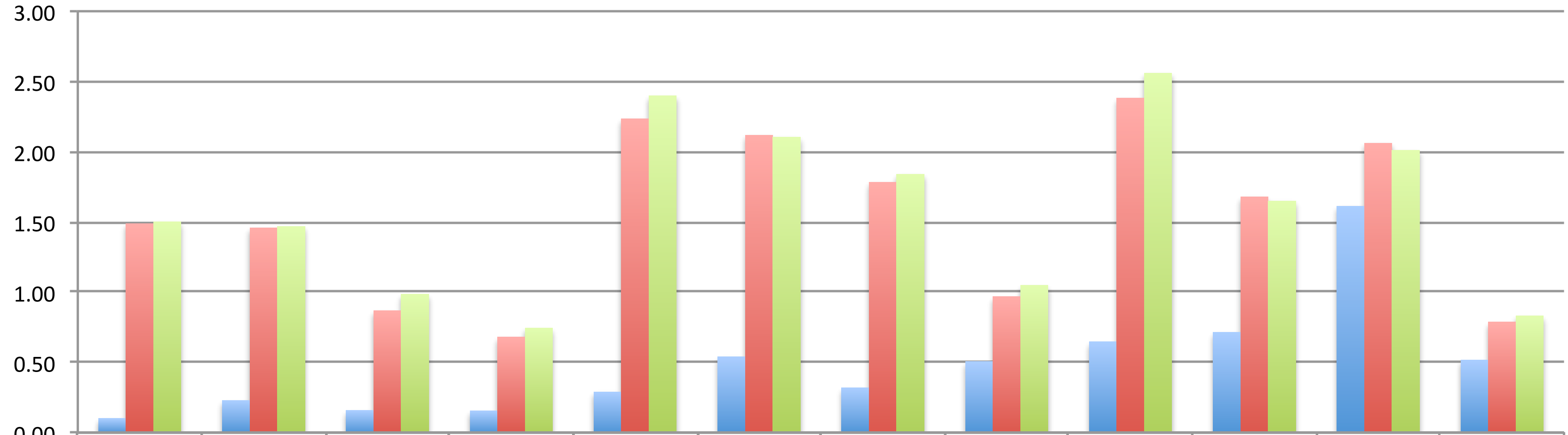
GPU RxAxP Numeric Flops



	Laplace				Brick				Empire			
	AxP	RX(AP)	RXA	RAXP	AxP	RX(AP)	RXA	RAXP	AxP	RX(AP)	RXA	RAXP
CUSPARSE	0.18	0.28	0.25	0.21	0.36	0.58	0.50	0.57	0.72	0.76	2.15	0.57
KKMEM	3.36	3.24	1.32	1.14	4.04	5.29	3.45	1.57	3.78	4.00	2.86	1.48
KKSPEED	3.41	3.30	1.60	1.33	4.65	5.24	3.62	1.79	4.28	3.89	2.75	1.64
KKMCR	0.77	1.42	1.11	0.97	1.49	3.06	3.39	1.47			4.99	1.53
KKMCW	0.71	1.84	1.13	1.09	1.79	3.95	3.75	1.71			5.00	1.68

- Coloring based ones does much less operations.
  - But accesses to B (second matrix) suffer from non-coalesced
  - Still performance is comparable or better when second matrix has dense rows.
  - Or when KKMEM also suffers from noncoalesced B accesses

GPU Gflops for RxAxP



	Laplace AxP	Laplace RX(AP)	Laplace RXA	Laplace RAXP	Brick AxP	Brick RX(AP)	Brick RXA	Brick RAXP	Empire AxP	Empire RX(AP)	Empire RXA	Empire RAXP
CUSPARSE	0.10	0.23	0.16	0.16	0.29	0.54	0.32	0.51	0.65	0.71	1.61	0.52
KKMEM	1.49	1.46	0.87	0.68	2.23	2.12	1.78	0.97	2.38	1.68	2.06	0.79
KKSPEED	1.50	1.47	0.99	0.75	2.40	2.10	1.84	1.05	2.56	1.65	2.01	0.83

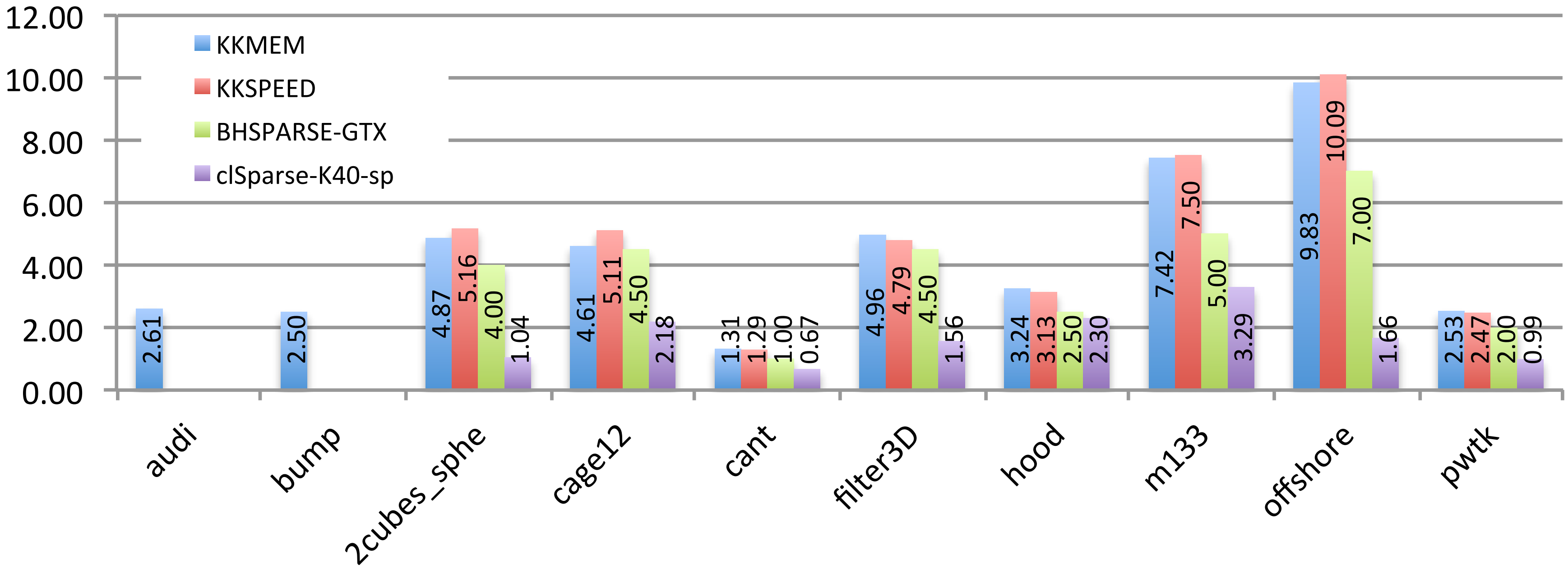
Higher is better

- CUSP runs out of memory
- Speedups range from 1.28 (1.25) to 14.83 (14.93). Average 3.90 (4.06)
- Cons:

  KKMEM – cost to get memory through uniform pool

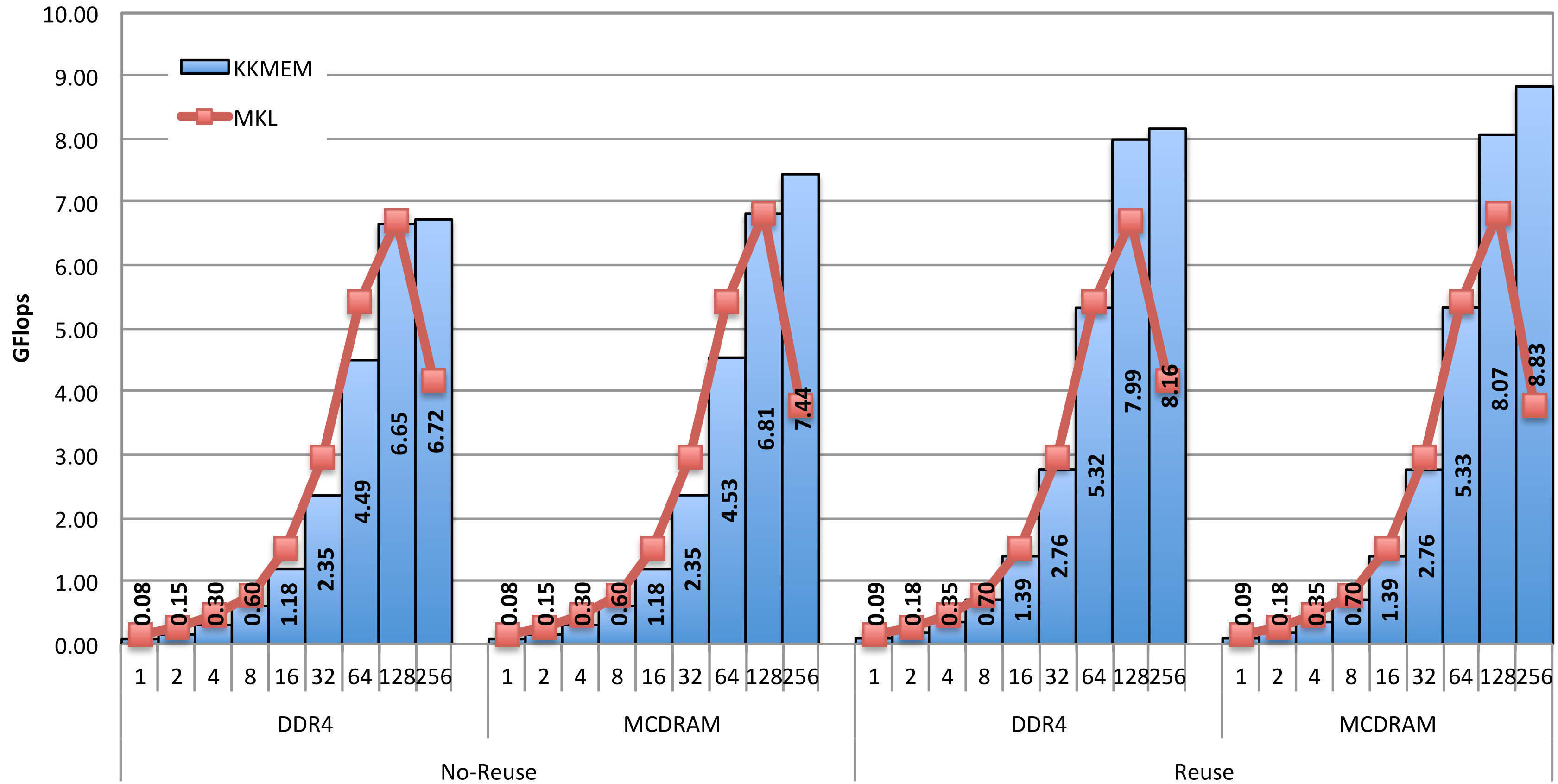
  KKSPEED – hash operations are done through ‘%’ instead of &.

GPU AxA Speedup w.r.t cuSPARSE



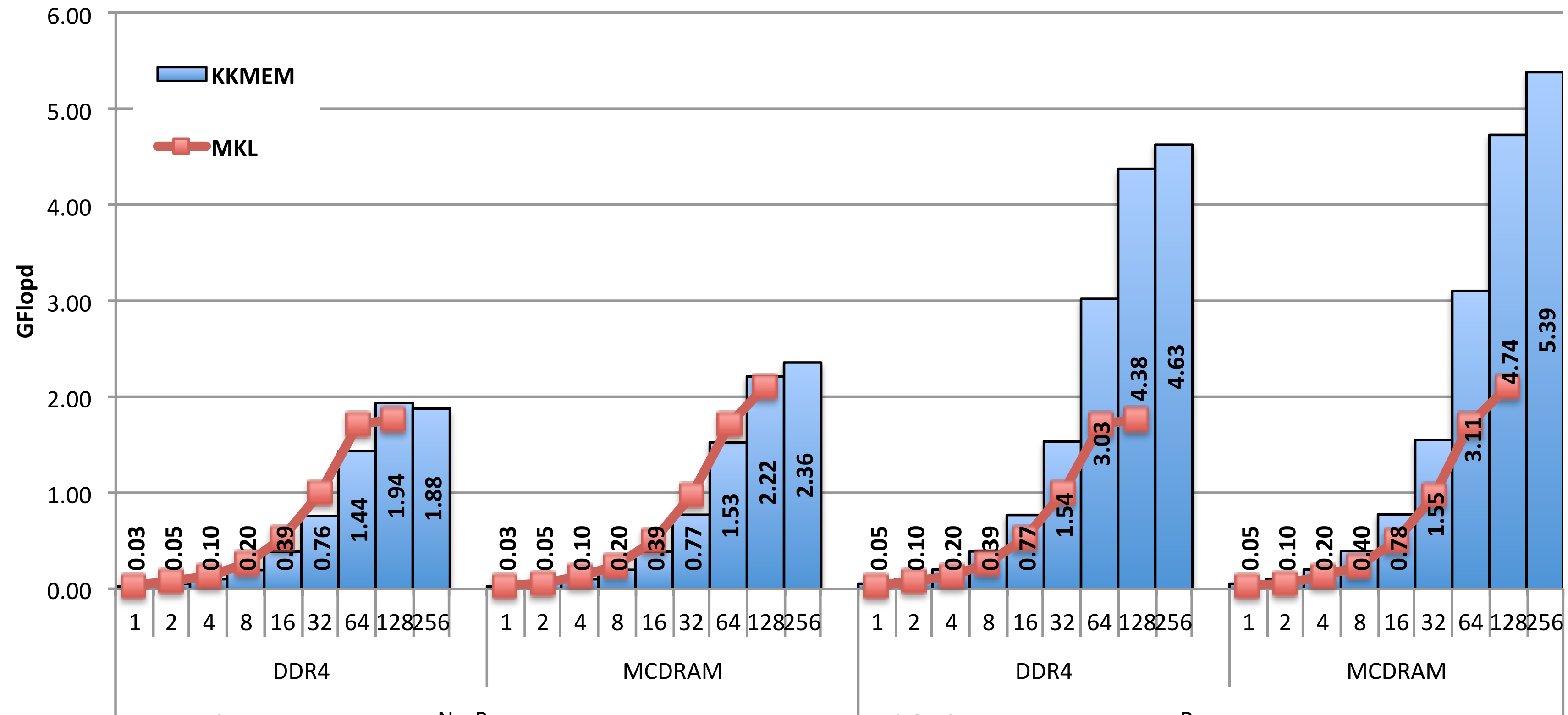
- Overall KKMEM speedup: 3.76
  - KKSPEED - 4.19 (4.14 for KKMEM)
- Audi has a very irregular row distribution. Output 7586MB
  - Pool requires -> 952 MB symbolic and 308MB numeric
- Bump – Output 6410MB: pool: 280MB and 87 MB

KNL Audi AxA



- MKL is faster upto 64 cores. Similar performance on 128, and MKL does not scale on 256 threads.
- With reuse upto 1.95 to 2.33 (1.20 on 128) speedups.
- Compression is successful here. Symbolic is 85% faster than numeric.

KNL Laplace AxP



- MKL is faster upto 64 cores. KKMEM is 10% faster on 128 threads
  - MKL does not finish in 1000 seconds on 256 threads.
- With reuse upto 2.48 speedups.
- Compression is not successful here (7% reduction).
  - Symbolic has same time with numeric, sometimes even more expensive
- Need: Reorder/Pack of columns to improve compression. (SPMV cache locality)

KKMEM FLOP/Double vs GFLOPS