ACCELERATING MATRIX PROCESSING WITH GPUs

Nicholas Malaya, Shuai Che, Joseph Greathouse,
Rene van Oostrum, and Michael Schulte
AMD Research

ACCELERATING MATRIX PROCESSING WITH GPUS

MOTIVATION

- Matrix operations are ubiquitous
  - Critical in HPC, machine learning, 3D rendering, gaming, signal processing, and more
- Serial performance no longer doubling every ~2 years
  - Parallel solutions are needed
  - Emerging GPUs provide tremendous compute capabilities
- Important matrix processing tasks include
  - **SpMV**: Sparse Matrix-Vector Multiply
  - **SpTS**: Sparse Triangle Solve
  - **Graph Processing:** Spare-Matrix Operations
  - **GEMM**: General Matrix-Matrix Multiply
- Representative of a range of challenges
  - Solutions also apply to manycore CPUs

ACCELERATING MATRIX PROCESSING WITH GPUS

REPRESENTATIVE PROBLEMS

- **SpMV**: Memory-bound problem with divergence
- **SpTS**: Heavily-researched sparse BLAS routine
- **Graph Processing:** Difficult to find general solutions
- **GEMM**: Compute-bound problem with new challenges
- Precision and accuracy requirements may vary greatly
- Optimizing data movement and memory accesses can be very important

Problem Type	Challenges
SpMV	lack of parallelism, memory divergence
SpTS	fine-grained parallelism, frequent synchronization
Graph Processing	building block optimizations, algorithm construction
GEMM	block matrix decomposition, numerical stability

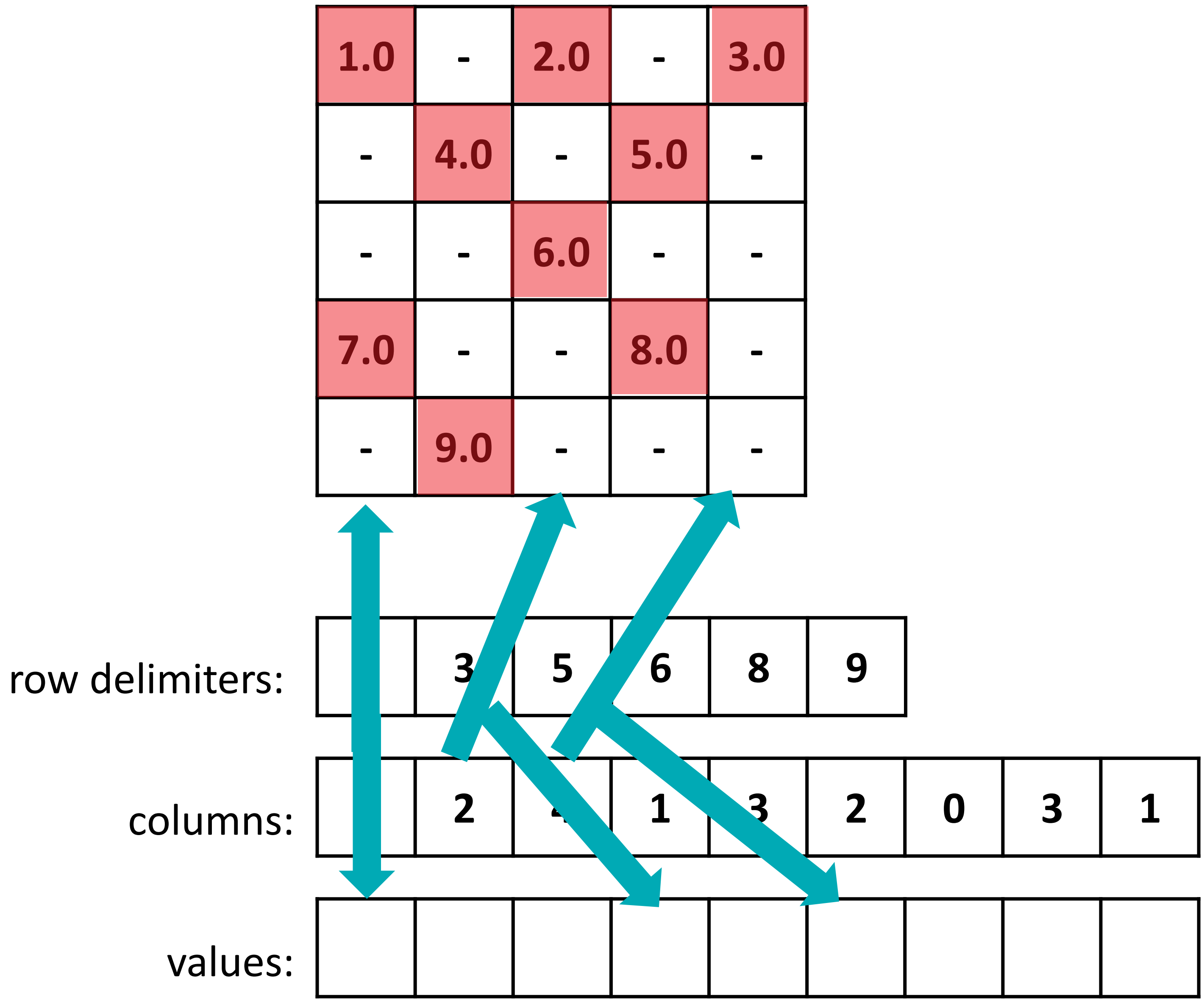
SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)

$$\begin{bmatrix} 1.0 & - & 2.0 & - & 3.0 \\ - & 4.0 & - & 5.0 & - \\ - & - & 6.0 & - & - \\ 7.0 & - & - & 8.0 & - \\ - & 9.0 & - & - & - \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

$$1*1 + 2*3 + 3*5$$

- SpMV applications include iterative solvers, machine learning, and graph analytics
- SpMV is memory-bound with performance dominated by how the sparse matrix is stored in memory
- Compressed sparse row (CSR) is the most common storage format
  - Compresses most matrices well and runs efficiently on CPUs

COMPRESSED SPARSE ROW (CSR)

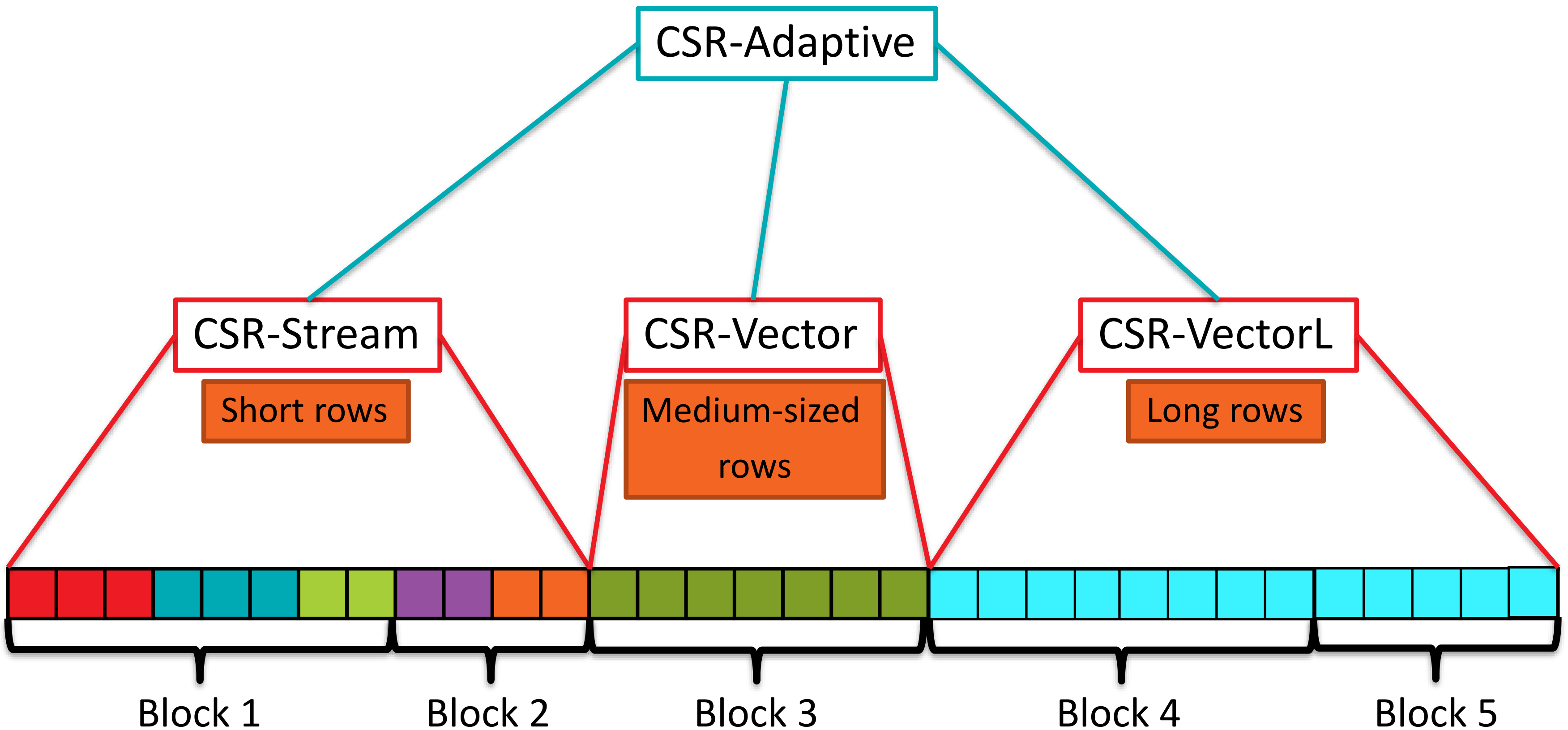


CSR-ADAPTIVE

- Efficient technique for SpMV with CSR format on GPUs
  - Measures the number of non-zero values in each row, when matrix first created
  - Groups together rows with roughly the same number of non-zero values
  - Determines the number of rows each SIMD unit operates on based on the number of non-zero values in each row
  - Loads contiguous rows into on-chip scratch-pad storage without causing memory divergence
- Solves the problems of memory divergence and lack of parallelism
  - Achieves up to 95% efficiency for many input matrices
  - Average of 28% faster than previous fastest technique
- Available as part of AMD's clSPARSE library

CSR-ADAPTIVE

A **complete** SpMV solution



"Efficient Sparse Matrix-Vector Multiplication on GPUs using the CSR Storage Format", J. Greathouse and M. Daga. SC 2014.

SPMV FUTURE RESEARCH

- Sparse matrix operations are becoming increasingly important in machine learning
  - Small weights and inputs are set to zero to reduce the number of computations
  - Some problems can leverage much lower precision, but methods are needed to determine how much precision is acceptable
- Some input matrices have much worse performance because their vector inputs do not cache well
  - Very large rows require a large number of vector accesses, which can displace useful data in the caches and cause memory conflicts
  - Cache bypassing of large rows may improve performance
- New systems have closely coupled CPUs and GPUs
  - Interesting to investigate which calculations should occur on the CPU and which should occur on the GPU

SPTS: SPARSE TRIANGLE SOLVE

START WITH THE DENSE CASE

- **SpTS**: Solve $Ax = b$, where A is lower triangular
  - Used in direct solves, iterative methods, least squares, etc.
- Consider a dense 3x3 lower triangular matrix:

$$\begin{pmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- Solution for each row is:

$$x_1 = b_1 / a_{11}$$
$$x_2 = (b_2 - a_{21} x_1) / a_{22}$$
$$x_3 = (b_3 - a_{31} x_1 - a_{32} x_2) / a_{33}$$

- Every row must be solved in series
  - Contention and dependencies

SPTS: SPARSE TRIANGLE SOLVE

IMPACT OF SPARSITY

- For, $x_2 = (b_2 - a_{21}x_1)/a_{22}$
  - If a_{21}= 0, first two rows can be solved in parallel
  - However, data dependencies not known *a-priori*
- Essential challenges of **SpTS**:
  - Determine data dependencies between rows
  - Lack of parallelism for some problems due to dependencies
- Existing solutions:
  - Level sets
    - Requires analysis to determine data dependencies between rows
    - Enables load balancing and fast traversal of the solution
  - Graph Coloring
    - Requires simpler analysis phase and row re-ordering
    - Row re-ordering can perturb the problem solution

SPTS: SPARSE TRIANGLE SOLVE

AVOIDING THE SPARSITY ANALYSIS AND FUTURE RESEARCH

- One solution [*Liu et al.*]
  - Transpose from CSR format to Compressed Sparse Column (CSC)
  - No longer requires pre-processing for data-dependencies
  - Transpose is expensive, but faster than full analysis
  - Requires additional memory
- Efficient implementations for SpTS on GPUs and manycore CPUs remains an important open research area
  - May be able to apply solutions that are similar to those used for SpMV
  - Determining accuracy and precision needed based on problem to be solved

"A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves", W. Liu, A. Li, J. Hogg, I. S. Duff, and B. Vinter, Euro-Par, 2016.

GRAPH PROCESSING WITH BELRED

OVERVIEW AND AN EXAMPLE

- Graph algorithms are widely used in diverse application domains
  - Business analytics, social network, life sciences, healthcare, infrastructure planning, engineering simulations, and more
- BelRed uses sparse-matrix routines to perform graph applications on GPUs
  - Includes a set of key sparse-matrix and vector routines
  - Similar to GraphBLAS, but optimized for GPUs
  - Initial implementation with OpenCL and SNACK
  - Clean abstraction and various underlying optimizations

BelRed: Constructing GPGPU Graph Applications with Software Building Blocks," S. Che, B. M. Beckmann, and S. K. Reinhardt," HPEC, 2014

BELRED

LIBRARY ROUTINES AND GRAPH ALGORITHMS

- Implemented linear-algebra routines

Functions	Description
$\vec{u} = SpMV(M, \vec{v})$	sparse-matrix vector multiplication
$\vec{u} = SpMinDotPlus(M, \vec{v})$	the $min.+$ operation
$\vec{u} = SegReduc\_Op(M)$	segmented reduction. $Op : +, \&, min\ldots$
$U = SpGeMM(M, N)$	sparse-matrix and sparse-matrix multiply
$U = vOuterSum(\vec{v}, \vec{w})$	the outer-sum operation
$\vec{u} = vElemWise\_Op(\vec{v}, \vec{w})$	vector elem. wise. $Op : +, \&, min, .$
$U = SpElemWise\_Op(\vec{v}, \vec{w})$	sparse matrix elem. wise. $Op : +, \&, min, .$

- BelRed implements important graph algorithms using sparse linear algebra operations
  - PageRank (SpMV)
  - Graph coloring (SegReduc)
  - Maximal independent set (vElementWise, SpMV)
  - K-truss (SpMV, SpGeMM, SpElemWise)
  - …

“Programming GPGPU graph applications with linear algebra building blocks,” S. Che, B. M. Beckmann, and S. K. Reinhardt, IJPP 2016.

BELRED

FUTURE RESEARCH

- Optimizations for the Radeon Open Compute Platform (ROCm)
  - Optimize for lower-precision arithmetic when appropriate
  - Leverage very efficient on-chip memories to improve performance
- Additional optimizations can build on previous work
  - Greathouse and Daga (SC'14) for SpMV, Liu and Vinter (IPDPS'14) for SpGEMM
  - Classify matrix regions into different bins (e.g., rows with different sizes), and launch different optimized GPU kernels to process different bins
- Multi-GPU implementations with efficient static and dynamic work partitioning across GPUs
- Some graph applications have interesting dependencies across different sparse-matrix routines
  - Provides opportunities for more parallelism and asynchronous execution

ACCELERATING MATRIX-MATRIX MULTIPLICATION (GEMM)

OVERVIEW AND AN EXAMPLE

- Product of two dense matrices: $\mathrm{C} = AB$
  - Common operation in scientific computing and machine learning
  - Computationally expensive and compute-bound
  - Precision and accuracy requirements may vary greatly
- Consider 2x2 matrix multiply,

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

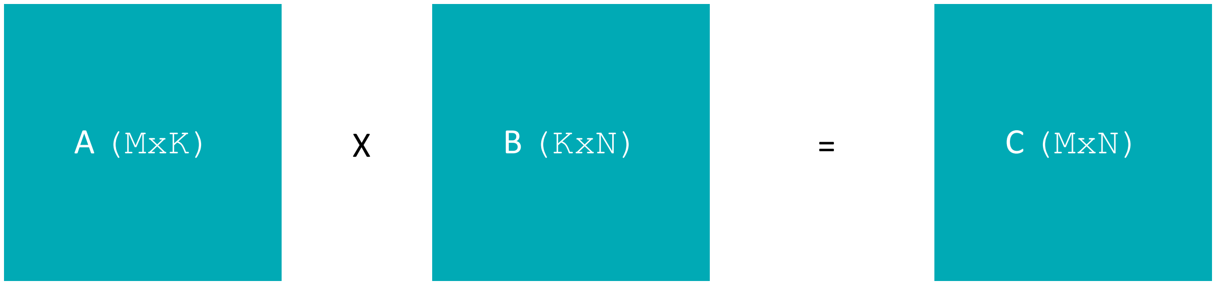
- Written out,

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} * b_{11} + a_{12} * b_{21} & a_{11} * b_{12} + a_{12} * b_{22} \\ a_{21} * b_{11} + a_{21} * b_{21} & a_{21} * b_{12} + a_{22} * b_{22} \end{pmatrix}$$

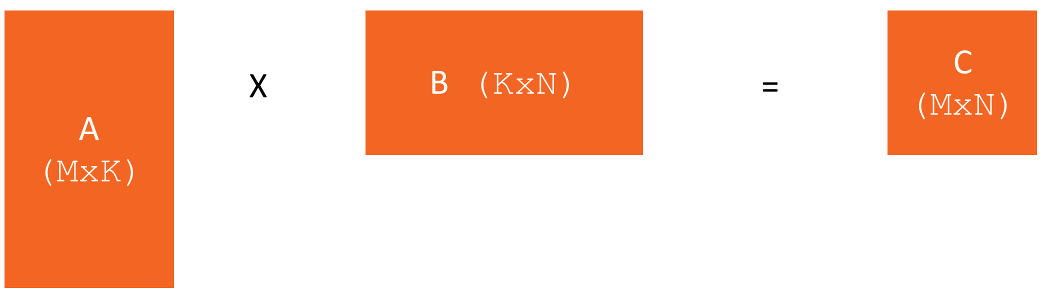
- This requires a total of 8 multiplies and 4 additions
- Matrix-matrix multiply scales asymptotically as $O(n^3)$

GEMM MACHINE LEARNING

- With HPC, matrices are often square or close to square



- With machine learning, matrix dimensions can vary greatly based on problem being solved and layer in the network



- Optimized GEMM routines available in AMD's MIOpen library for a wide range of matrix sizes

TECHNIQUES FOR FAST MATRIX-MATRIX MULTIPLICATION

- Strassen-Winograd Matrix Multiplication
  - Recursive approach that reduces the number of multiplies while increasing the number of addative operations
  - Reduces complexity from $O(n^3)$ to $O(n^{2.807})$
  - Can increase numerical error and need for communication
- Several other techniques exist for speeding up matrix-matrix multiplication, but they may not work as well in practice
- Important future research includes:
  - Algorithms for non-square matrices of various sizes
  - Optimizing low-precision GEMM
  - Optimizing SpGEMM performance on GPUs and manycore CPUs

"Accelerating Strassen-Winograd's Matrix Multiplication Algorithm on GPUs," W. Lai, H. Arafat, V. Elango, and P. Sadayappan, HiPC, 2013.

CONCLUSIONS

SOLUTIONS TO ACCELERATING MATRIX PROCESSING

- Better algorithms
  - E.g., designed to expose more parallelism
- Careful mapping of algorithms to hardware
  - New instructions and specialized hardware for fast matrix computations
- Fitting problem into scratchpad memory
  - Often requires direct programmer management
- Match precision to application and problem requirements
  - Scientific computing: High precision
  - Machine learning training: Low precision
  - Machine learning inference: Very low precision
- Libraries that capture and provide users the above

DISCLAIMER & ATTRIBUTION

ATTRIBUTION