CLSPARSE:  
A VENDOR-OPTIMIZED OPEN-SOURCE SPARSE BLAS LIBRARY

JOSEPH L. GREATHOUSE, KENT KNOX, JAKUB POŁA*,  
KIRAN VARAGANTI, MAYANK DAGA

*UNIV. OF WROCLAW & VRATIS LTD.
Operate on matrices and vectors with many zero values

Useful in numerous domains
- Computational fluid dynamics, other engineering applications
- Computational physics, other HPC applications (e.g. HPCG)
- Graph algorithms

Requires very different optimizations than dense BLAS
- Kernels are often bandwidth-bound
- Sometimes lack parallelism

Needs different library support than traditional dense BLAS
EXAMPLES OF EXISTING LIBRARIES

- Proprietary, optimized libraries
  - Nvidia cuSPARSE
  - Intel MKL

- Open-source libraries
  - ViennaCL
  - MAGMA
  - Numerous one-off academic libraries (clSpMV, bhSPARSE, yaSpMV, etc.)
+ Often highly optimized (especially by hardware vendors) – performance matters!
  – Lots of engineers working to optimize libraries for customers

— Often work on (or optimized for) limited set of hardware
  – Nvidia cuSPARSE only works on Nvidia GPUs
  – Intel MKL optimized for Intel processors

— Can be slow to add new features from the research community
  – More than 50 GPU-based SpMV algorithms in the literature; few end up in proprietary libraries

— You can’t see or modify the code!
  – e.g. Kernel fusion shown to be performance benefit – closed-source libraries don’t allow this
  – Difficult for academic research to move forward the state of the art
OPEN-SOURCE LIBRARIES

+ You can see and modify the code!
  – Not only can you modify code to improve performance, you can advance the algorithms

+ Often closely integrated with research community
  – e.g. ViennaCL support for CSR-Adaptive and SELL-C-σ within months of their publication

+ Sometimes work across vendors (thanks to languages like OpenCL™!)
  – e.g. ViennaCL works on Nvidia GPUs, AMD CPUs & GPUs, Intel CPUs & GPUs, Intel Xeon Phi, etc.

  — Sometimes do not work across vendors
    – e.g. Caffe (DNN library) originally CUDA-only (ergo Nvidia hardware only)

  — Not always the best performance
    – Can trade off performance for portability and maintainability
    – Do not always include hardware-specific optimizations
AMD AND THE GPUOPEN INITIATIVE

Vendor-optimized open-source support for important GPU software
- [http://gpuopen.com/](http://gpuopen.com/)
- Most source code available on GitHub or Bitbucket!

Open-source Gaming Libraries
- e.g. TressFX – Hair physics
- e.g. AOFX – optimized ambient occlusion
- Many others!

Open-source Compute Libraries
- clBLAS
- clFFT
- clRNG
Open-source OpenCL™ Sparse BLAS Library for GPUs
- Source code available, mostly Apache licensed (some MIT)
- Compiles for Microsoft Windows®, Linux®, and Apple OS X

Vendor optimizations. Developed as a collaboration between:
- AMD (both product and research teams)
- Vratis Ltd. (of SpeedIT fame)

Available at https://github.com/clMathLibraries/clSPARSE
cI SPARSE: An OpenCL™ Sparse BLAS Library
CLSPARSE DESIGN CHOICES

C Library API
- Make using library in C and FORTRAN programs easier

Allow full control of OpenCL™ data structures, work with normal cl_mem buffers

Abstract internal support structures from user

Use compressed sparse row (CSR) as sparse matrix storage format
- Much existing code already uses CSR – no GPU-specific storage format
- Many complex algorithms (SpMSpM, SpTS) require CSR, so no structure swapping in clSPARSE
CLSPARSE API EXAMPLES – INITIALIZING A SPARSE MATRIX (1)

```c
// CSR matrix structure
clsparseCsrMatrix A;
// Matrix size variables
clsparseIdx_t nnz, row, col;
```
// CSR matrix structure
clsparseCsrMatrix A;

// Matrix size variables
clsparseIdx_t nnz, row, col;

// read matrix market header to get the size of the matrix
clsparseStatus fileErr = clsparseHeaderFromFile( &nnz, &row, &col, mtx_path.c_str( ) );
A.num_nonzeros = nnz; A.num_rows = row; A.num_cols = col;
CLSPARSE API EXAMPLES – INITIALIZING A SPARSE MATRIX (1)

```c
// CSR matrix structure
clSparseCsrMatrix A;

// Matrix size variables
clSparseIdx_t nnz, row, col;

// read matrix market header to get the size of the matrix
clSparseStatus fileErr = clSparseHeaderFromFile( &nnz, &row, &col, mtx_path.c_str() );
A.num_nonzeros = nnz; A.num_rows = row; A.num_cols = col;

// Allocate device memory for CSR matrix
A.values = clCreateBuffer( ctxt, CL_MEM_READ_ONLY, nnz * sizeof(float), NULL, &cl_status );
A.col_indices = clCreateBuffer( ctxt, CL_MEM_READ_ONLY, nnz * sizeof(clSparseIdx_t), NULL, &cl_status );
A.row_pointer = clCreateBuffer( ctxt, CL_MEM_READ_ONLY, (num_rows + 1) * sizeof(clSparseIdx_t), NULL, &cl_status );
```
CLSPARSE API EXAMPLES – INITIALIZING A SPARSE MATRIX (2)

// Reminder: clsparseCsrMatrix A;

// clSPARSE control object
// Control object wraps CL state (contains CL queue, events, and other library state)
clsparseCreateResult createResult = clsparseCreateControl( cmd_queue );
CLSPARSE API EXAMPLES – INITIALIZING A SPARSE MATRIX (2)

// Reminder: clsparseCsrMatrix A;

// clSPARSE control object
// Control object wraps CL state (contains CL queue, events, and other library state)
clsparseCreateResult createResult = clsparseCreateControl( cmd_queue );

// Read matrix market file with explicit zero values straight into device memory
// This initializes CSR format sparse data
err = clsparseSCsrMatrixFromFile( &A, mtx_path.c_str(), createResult.control, CL_TRUE );
CLSPARSE API EXAMPLES – INITIALIZING A SPARSE MATRIX (2)

// Reminder: clsparseCsrMatrix A;

// clSPARSE control object
// Control object wraps CL state (contains CL queue, events, and other library state)
clsparseCreateResult createResult = clsparseCreateControl(cmd_queue);

// Read matrix market file with explicit zero values straight into device memory
// This initializes CSR format sparse data
err = clsparseSCsrMatrixFromFile(&A, mtx_path.c_str(), createResult.control, CL_TRUE);

// OPTIONAL - This function allocates memory for rowBlocks structure.
// The presence of this meta data enables the use of the CSR-Adaptive algorithm
clsparseCsrMetaCreate(&A, createResult.control);
CLSPARSE API EXAMPLES – INITIALIZING VECTORS

// Allocate and set up vector
cldenseVector x;
clsparseInitVector(&x);
CLSPARSE API EXAMPLES – INITIALIZING VECTORS

// Allocate and set up vector
cldenseVector x;
clsparseInitVector(&x);

// Initialize vector in device memory
float one = 1.0f;
x.num_values = A.num_cols;

x.values = clCreateBuffer( ctxt, CL_MEM_READ_ONLY, x.num_values * sizeof(float),
    NULL, &cl_status);
cl_status = clEnqueueFillBuffer( cmd_queue, x.values, &one, sizeof(float),
    0, x.num_values * sizeof(float), 0, NULL, NULL);
CLSPARSE API EXAMPLES – INITIALIZING SCALARS

// Allocate scalar values in device memory
clsparseScalar alpha;
clsparseInitScalar(&alpha);
alpha.value = clCreateBuffer( ctxt, CL_MEM_READ_ONLY, sizeof(float), nullptr,
                             &cl_status);
// Allocate scalar values in device memory
clsparseScalar alpha;
clsparseInitScalar(&alpha);
alpha.value = clCreateBuffer( ctxt, CL_MEM_READ_ONLY, sizeof(float), nullptr, &cl_status);

// Set alpha = 1;
float* halpha = (float*) clEnqueueMapBuffer( cmd_queue, alpha.value, CL_TRUE, CL_MAP_WRITE, 0, sizeof(float), 0, NULL, NULL, &cl_status);
*halpha = 1.0f;
cl_status = clEnqueueUnmapMemObject( cmd_queue, alpha.value, halpha, 0, NULL, NULL);
CLSPARSE API EXAMPLES – PERFORMING SPMV

// Reminder:
// clsparseCsrMatrix A;
// clsparseScalar alpha, beta;
// cldenseVector x, y;
// clsparseCreateResult createResult;

// Call the SpMV algorithm to calculate y=αAx+βy
// Pure C style interface, passing pointer to structs
cl_status = clsparseScsrmv(&alpha, &A, &x, &beta, &y, createResult.control );
CLSPARSE API EXAMPLES – CG SOLVE

// Create solver control object. It keeps info about the preconditioner,
// desired relative and absolute tolerances, max # of iterations to be performed
// We use: preconditioner:diagonal, rel tol:1e-2, abs tol:1e-5, max iters: 1000
clsparseCreateSolverResult solverResult =
    clsparseCreateSolverControl( DIAGONAL, 1000, 1e-2, 1e-5 );
CLSPARSE API EXAMPLES – CG SOLVE

// Create solver control object. It keeps info about the preconditioner,
// desired relative and absolute tolerances, max # of iterations to be performed
// We use: preconditioner:diagonal, rel tol:1e-2, abs tol:1e-5, max iters: 1000
clSparseCreateSolverResult solverResult =
    clSparseCreateSolverControl( DIAGONAL, 1000, 1e-2, 1e-5 );

// OPTIONAL - Different print modes of the solver status:
// QUIET:no messages (default), NORMAL:print summary, VERBOSE:per iteration status;
clSparseSolverPrintMode( solverResult.control, VERBOSE );
CLSPARSE API EXAMPLES – CG SOLVE

// Create solver control object. It keeps info about the preconditioner,
// desired relative and absolute tolerances, max # of iterations to be performed
// We use: preconditioner:diagonal, rel tol:1e-2, abs tol:1e-5, max iters: 1000
clsparseCreateSolverResult solverResult =
    clsparseCreateSolverControl( DIAGONAL, 1000, 1e-2, 1e-5 );

// OPTIONAL - Different print modes of the solver status:
// QUIET:no messages (default), NORMAL:print summary, VERBOSE:per iteration status;
clsparseSolverPrintMode( solverResult.control, VERBOSE );

// Call into CG solve
c1_status = clsparseScsrgcg(&x, &A, &y, solverResult.control, createResult.control );
UNDERLYING ALGORITHMS FROM THE RESEARCH LITERATURE

- SpMV uses CSR-Adaptive algorithm
  - Described by AMD in research papers at SC’14 and HiPC’15
  - Requires once-per-matrix generation of some meta-data (clSparseCsrMetaCreate())
  - Falls back to slower CSR-Vector style algorithm if meta-data does not exist

- Batched CSR-Adaptive for SpM-DM multiplication

- SpMSpM uses algorithm described in Liu and Vinter at IPDPS’14 and JPDC’15
clSPARSE Performance Comparisons
BENCHMARKING CLSPARSE

AMD Test Platform

- **AMD Radeon™ Fury X**
  - Intel Core i5-4690K
  - 16 GB Dual-channel DDR3-2133
  - Ubuntu 14.04.4 LTS
  - fglrx 15.302 driver
  - AMD APP SDK 3.0

- **clSPARSE v0.11**
  - ViennaCL v1.7.1
BENCHMARKING CLSPARSE

AMD Test Platform

- AMD Radeon™ Fury X
  - Intel Core i5-4690K
  - 16 GB Dual-channel DDR3-2133
  - Ubuntu 14.04.4 LTS
  - fglrx 15.302 driver
  - AMD APP SDK 3.0

- clSPARSE v0.11
- ViennaCL v1.7.1

Nvidia Test Platform

- Nvidia GeForce GTX TITAN X
  - Intel Core i7-5960X
  - 64GB Quad-channel DDR4-2133
  - Ubuntu 14.04.4 LTS
  - Driver 352.63
  - CUDA 7.5

- clSPARSE v0.11
- cuSPARSE v7.5
COMPARISON TO PROPRIETARY VENDOR-OPTIMIZED LIBRARY

- Compare clSPARSE performance to Nvidia’s cuSPARSE library

- clSPARSE works across vendors, directly compare on identical Nvidia hardware
  - Also compare AMD GPU to all of this
clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library

SINGLE PRECISION SPMV – VENDOR OPTIMIZED

![Graph showing performance comparison between cuSPARSE-NV and clSPARSE-NV for various benchmarks, with cuSPARSE-NV in green and clSPARSE-NV in yellow.](image)
Major Algorithmic Improvements
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>cuSPARSE-NV</th>
<th>clSPARSE-NV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense8</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>Protein</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>FEM-Spheres</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>FEM-Cantilever</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>WindTunnel</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>FEM-Harbor</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>FEM-Ship</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Epidemiology</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>FEM-Accelerator</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Circuit</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Webbase</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>LP</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>circuit5M</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>eu2005</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Ga41As41H72</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>in-2004</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>mib1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>S&amp;I-Ge41H77</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>ASIC-680k</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>dc2 FullChip</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>ins2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>bone010</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>crankseg.2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>idofo</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>rajat31</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Rucci1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>bov02</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>sis</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>GeoMean</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

**All-around Performance Improvements**

**Legend**
- cuSPARSE-NV
- clSPARSE-NV
SINGLE PRECISION SPMV – VENDOR OPTIMIZED

Avg. of 4.5x faster than cuSPARSE on identical hardware

SP GFLOPS
0 10 20 30 40 50 60 70 80 90 100 110

cuSPARSE-NV  clSPARSE-NV

Dense8 Protein FEM-Spheres WindTunnel FEM-Harbor FEM-Ship Epidemiology FEM-Accelerator Circuit Webase lp circuit5M eu2005 Ga414s41H72 in-2004 mpi1 Sf41Ge41H72 ASIC680k dc2 fullChip ins2 bone10 cranlese2 idoof raju31 ruccl1 bov2 sles transient GeoMean
SINGLE PRECISION SPMV – VENDOR OPTIMIZED

- cuSPARSE-NV
- cISPARSE-NV
- cISPARSE-AMD
clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library

SINGLE PRECISION SPMV – VENDOR OPTIMIZED

AMD Hardware
20% Faster
DOUBLE PRECISION SPMV – VENDOR OPTIMIZED

AMD Hardware
87% Faster
Lack of OpenCL™
64-bit Atomics
SINGLE PRECISION SPM-SPM – VENDOR OPTIMIZED

SP GFLOPS

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

m133-b3 Circuit FEM-Accelerator cit-Patents web-Google wiki-Vote majorBasis FEM-Harbor Webbase email-Enron poison3Da marlo002 Protein WindTunnel ca-CondMat 2obbs_sphere Economics filter3D FEM-Ship hood cage12 offshore FEM-Cantilever Epidemiology roadNet-CA FEM-Spheres amazon0312 GeoMean

cuSPARSE-NV cISPARSE-NV cISPARSE-AMD

clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library
SINGLE PRECISION SPM-SPM – VENDOR OPTIMIZED

- cuSPARSE-NV
- clSPARSE-NV
- clSPARSE-AMD
SINGLE PRECISION SPM-SPM – VENDOR OPTIMIZED
clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library

Average within 20% of cuSPARSE on Nvidia Hardware
clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library

SINGLE PRECISION SPM-SPM – VENDOR OPTIMIZED

Average within 20% of cuSPARSE on Nvidia Hardware

AMD hardware within 7% of cuSPARSE on avg.
CLSPARSE IS PORTABLE ACROSS VENDORS
OPENCL™ GIVES YOU THE FREEDOM TO CHOOSE YOUR HARDWARE

AMD Radeon™ Fury X
512 GB/s Memory BW

AMD FirePro™ S9300 x2
1024 GB/s Aggregate Memory BW
COMPARISON TO OPEN-SOURCE LIBRARY

- Comparison against ViennaCL, the popular open-source linear algebra library

- Only used AMD hardware for this to ease readability
  - Both libraries work across vendors

- ViennaCL implements an older version of AMD’s CSR-Adaptive algorithm for SpMV
SINGLE PRECISION SPMV – OPEN SOURCE

SP GFLOPS

ViennaCL-AMD  cISPARSE-AMD
SINGLE PRECISION SPMV – OPEN SOURCE

Same algorithmic benefits

ViennaCL-AMD  clSPARSE-AMD
clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library

SINGLE PRECISION SPMV – OPEN SOURCE

clSPARSE 2.5x faster on average
clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library

SINGLE PRECISION SPM-SPM – OPEN SOURCE

- clSPARSE 27% faster on average
Available at:

https://github.com/clMathLibraries/clSPARSE

Contributions welcome!
For more information on the range of AMD FirePro™ S-series graphics accelerators, contact:

**Christian Seithe**  
Sr. Business Development Manager EMEA –  
AMD Professional Graphics  
AMD GMBH, Einsteinring 24, D-85609 Dornach b. München, GERMANY  
Email: christian.seithe@amd.com  
Mobile Office: +49 (0) 89 45053 255  
Mobile Phone: +49 (0) 172 999 77 41

**Donal Harford**  
Business Development Manager, UK/Ireland/Nordics –  
AMD Professional Graphics Division  
Email: donal.harford@amd.com  
Mobile: +353 87 442 62 62

**Joshue “Josh” Saenz**  
Sales, AMD Professional Graphics  
7171 Southwest Parkway, Austin, TX 78735 USA  
Email: Joshue.Saenz@amd.com  
Office: +(1) 512-602-0256  
Mobile: +(1) 512-201-3065
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2016 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD Radeon, AMD FirePro, AMD Catalyst and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. OpenCL is a trademark of Apple, Inc. used by permission by Khronos. Microsoft is a registered trademark of Microsoft Corporation. Windows is a registered trademark of Microsoft Corporation. Linux is a registered trademark of Linus Torvalds. Other names are for informational purposes only and may be trademarks of their respective owners.
DOUBLE PRECISION SPMV – OPEN SOURCE

DP GFLOPS

- ViennaCL-AMD
- clSPARSE-AMD

54 | clSPARSE: A Vendor-Optimized Open-Source Sparse BLAS Library