clSpMV: A Cross-Platform OpenCL SpMV Framework on GPUs

Bor-Yiing Su, subrian@eecs.berkeley.edu
Kurt Keutzer, keutzer@eecs.berkeley.edu

Parallel Computing Lab,
University of California, Berkeley
Outline

- Motivation
- The Cocktail Sparse Matrix Format
- The clSpMV Framework
- Experimental Results
- Conclusion
Many iterative methods are composed of a BLAS2 operation with BLAS1 updates

- BLAS2 operation dominates the execution time

Many matrices are sparse in natural

- We need to optimize the SpMV operation

**Algorithm:** Conjugate Gradient  
**Input:**  
- $A$ (Symmetric Matrix)  
- $b$ (Vector)  
- $x_0$ (Initial Solution)  
**Output:**  
- $x$ (Final Solution)  

1. \( r_0 \leftarrow b - Ax_0 \);  
2. \( p_0 \leftarrow r_0 \);  
3. **for** $k \leftarrow 0, 1, \ldots, \text{until convergence}$  
   a. \( v_k \leftarrow Ap_k \);  
   b. \( \alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T v_k} \);  
   c. \( x_{k+1} \leftarrow x_k + \alpha_k p_k \);  
   d. \( r_{k+1} \leftarrow r_k - \alpha_k v_k \);  
   e. **Test bounds for convergence**;  
   f. \( \beta_k \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \);  
   g. \( p_{k+1} \leftarrow r_{k+1} + \beta_k p_k \);  
4. **end for**  
5. Return $x_{k+1}$.

**Algorithm:** Lanczos  
**Input:**  
- $A$ (Symmetric Matrix)  
- $v$ (Initial Vector)  
**Output:**  
- $\Theta$ (Ritz Values)  
- $X$ (Ritz Vectors)  

1. Start with $r \leftarrow v$;  
2. \( \beta_0 \leftarrow \|r\|_2 \);  
3. **for** $j \leftarrow 1, 2, \ldots, \text{until convergence}$  
   a. \( v_j \leftarrow r / \beta_{j-1} \);  
   b. \( \alpha_j \leftarrow v_j^T r \);  
   c. \( r \leftarrow r - v_{j-1} \beta_{j-1} \);  
   d. **Reorthogonalize if necessary**;  
   e. \( \beta_j \leftarrow \|r\|_2 \);  
   f. **Compute Ritz values** $T_j = SS^T$;  
   g. **Test bounds for convergence**;  
4. **end for**  
5. **Compute Ritz vectors** $X \leftarrow V_j S$;
Optimizing the SpMV Computation

- Challenges of SpMV
  - Low arithmetic intensity (memory bounded)
  - Irregular memory access

- Minimizing memory footprint
  - Proposing new sparse matrix formats

- Saturating memory bandwidth
  - Optimizing the memory access pattern on the memory system

- Block matrix
- Symmetric
- Diagonal

Intel Xeon E5345 (Clovertown)
NVIDIA G80
Outline

- Motivation
- The Cocktail Sparse Matrix Format
- The clSpMV Framework
- Experimental Results
- Conclusion
Pros and Cons of Matrix Formats

- Every sparse matrix format has its own pros and cons
- Most of the matrix formats fall into three categories

<table>
<thead>
<tr>
<th>Matrix Format Category</th>
<th>Example Sparse Matrix</th>
<th>Included Matrix Formats</th>
<th>Pros</th>
<th>Cons</th>
<th>Suggested Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagonal</td>
<td></td>
<td>BDIA DIA</td>
<td>• Implicit column indices for diagonals • Aligned memory access pattern</td>
<td>• Need zero fillings on sparse diagonals</td>
<td>• Matrices that are mainly dense diagonals</td>
</tr>
<tr>
<td>Blocked</td>
<td></td>
<td>SBELL BELL BCSR</td>
<td>• Implicit column indices for blocks • Can reuse the multiplied vector</td>
<td>• Need zero fillings on sparse blocks</td>
<td>• Matrices that are mainly dense blocks</td>
</tr>
<tr>
<td>Flat</td>
<td></td>
<td>SELL ELL CSR COO</td>
<td>• No zero fillings • Need explicit column indices • Unaligned memory access</td>
<td></td>
<td>• Irregular matrices</td>
</tr>
</tbody>
</table>
### Pros and Cons of Diagonal-Based Formats

- **DIA**: Diagonal format
- **BDIA**: Banded DIA format

<table>
<thead>
<tr>
<th>Matrix Format</th>
<th>Example Sparse Matrix</th>
<th>Pros</th>
<th>Cons</th>
<th>Suggested Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIA</td>
<td><img src="image1" alt="DIA Matrix Example" /></td>
<td>• More flexible on the width of the diagonals</td>
<td>• Cannot use shared memory to cache the vector</td>
<td>• Matrices with arbitrary dense diagonals</td>
</tr>
<tr>
<td>BDIA</td>
<td><img src="image2" alt="BDIA Matrix Example" /></td>
<td>• Can use shared memory to cache the vector</td>
<td>• Need extra storage to store the pointers to each band</td>
<td>• Matrices with dense bands</td>
</tr>
</tbody>
</table>
## Pros and Cons of Flat Formats

<table>
<thead>
<tr>
<th>Matrix Format</th>
<th>Example Matrix Storage</th>
<th>Pros</th>
<th>Cons</th>
<th>Suggested Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ELL</strong></td>
<td><img src="ell.png" alt="Example Matrix" /></td>
<td>• Aligned memory access</td>
<td>• Need zero paddings</td>
<td>• Matrices with similar # of non-zero per row</td>
</tr>
<tr>
<td><strong>SELL</strong></td>
<td><img src="sell.png" alt="Example Matrix" /></td>
<td>• Aligned memory access • Fewer zero paddings</td>
<td>• Still need zero paddings • Additional pointers to slices</td>
<td>• Matrices with similar # of non-zero per slice</td>
</tr>
<tr>
<td><strong>CSR</strong></td>
<td><img src="csr.png" alt="Example Matrix" /></td>
<td>• No zero paddings</td>
<td>• Unaligned memory access • Bad load balance</td>
<td>• Matrices with moderate irregular # of non-zero per row</td>
</tr>
<tr>
<td><strong>COO</strong></td>
<td><img src="coo.png" alt="Example Matrix" /></td>
<td>• No zero paddings • Good load balance</td>
<td>• Explicit row indices</td>
<td>• Matrices with highly irregular # of non-zero per row</td>
</tr>
</tbody>
</table>
### Pros and Cons of Blocked Formats

- **BELL: Blocked ELL**
- **SBELL: Sliced blocked ELL**
- **BCSR: Blocked CSR**

<table>
<thead>
<tr>
<th>Matrix Format</th>
<th>Example Matrix Storage</th>
<th>Pros</th>
<th>Cons</th>
<th>Suggested Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BELL</td>
<td></td>
<td>• Aligned memory access</td>
<td>• Need zero paddings</td>
<td>• Matrices with similar # of blocks per blocked row</td>
</tr>
</tbody>
</table>
| SBELL         |                         | • Aligned memory access  
                 |                         | • Fewer zero paddings | • Matrices with similar # of blocks per slice |
| BCSR          |                         | • No zero paddings      | • Unaligned memory access  
                 |                         | • Bad load balance      | • Matrices with irregular # of blocks per blocked row |
The Cocktail Format

- Our premise: Every specialized region on a matrix deserves its own specialized representation
- The Cocktail Format: A combination of many different sparse matrix formats
  - A specialized submatrix is represented by a specialized format
  - Trivial case: Only one format is selected to represent the matrix
  - Complicated case: a matrix is partitioned into many submatrices, each represented by a different format
The Cocktail Matrix Partitioning Problem

- Challenges in matrix partitioning
  - The partition is matrix dependent
  - The partition is platform dependent
  - The partition is implementation dependent

- The Cocktail Matrix Partitioning (CMP) problem
  - Input: matrix $A$, $k$ formats supported by the Cocktail Format, $f_1, f_2, \ldots, f_k$, $k$ sets of implementations $P_1$ to $P_k$ for formats $f_1$ to $f_k$
  - Let $t(A_i, f_i, L_i)$ be the execution time of a SpMV kernel using format $f_i$ and implementation $L_i$ on submatrix $A_i$
  - Output: submatrices $A_1$ to $A_k$, implementations $L_1$ to $L_k$

$$\begin{align*}
\min & \quad \sum_{i=1}^{k} t(A_i, f_i, L_i) \\
\text{s.t.} & \quad \sum_{i=1}^{k} A_i = A \\
& \quad L_i \in P_i \quad \forall 1 \leq i \leq k
\end{align*}$$
Outline

- Motivation
- The Cocktail Sparse Matrix Format
- The clSpMV Framework
- Experimental Results
- Conclusion
Overall Structure of clSpMV

- Offline benchmarking
  - Used to estimate the $t(A_i, f_i, L_i)$ values
- Online decision making
  - Partition the input matrix according to the offline benchmarking profiles
Offline Benchmarking

- One-time cost
- For every implementation of every format supported by clSpMV, sample the execution time on different sparse matrices
  - Sample on the matrix dimension and # non-zeros per row
  - Use interpolation to estimate \( t(A_i, f_i, L_i) \) values in the online decision making stage
  - The estimation accuracy can be further improved by getting more sample points (e.g. variations of # non-zeros per row)
Online Decision Making

- Analyze the input matrix
- Extract specialized regions that should be represented by specialized formats
- Use offline benchmarking profile to choose the best implementation for the underlying hardware platform
- Use a decision tree to guide the procedure of analysis and extraction
- Decide the priority of the matrix categories
  - Based on the highest estimated performance each category can achieve
Converting between formats is expensive

Follow a three-step strategy
  - Feature collection: Collecting features that are able to differentiate performance of different formats in the same category
  - Evaluation: Estimating the performance of different partitioning scenarios, find the best scenario
  - Extraction: Extracting submatrices based on the best scenario
Decision Tree: Extract Diagonals

- Feature collection
  - Compute the number of non-zeros per diagonal
- Evaluation
  - Evaluate the estimated performance of each tree branch, and make decision
- Extraction
  - Extract diagonals or bands based on the evaluation decision
Extracting Diagonals: Evaluation

- Definition of dense diagonals
  - $g_d$: maximum GFLOPS achievable by the diagonal category at the current matrix settings
  - $g_f$: maximum GFLOPS achievable by the flat category at the current matrix settings
  - $n_d$: the dimension of a diagonal
  - $e_d$: # of non-zeros in a diagonal
  - A diagonal is considered dense if $e_d > n_d g_f / g_d$

- Decision tree branches
  - Extract DIA: Representing all dense diagonals with DIA
  - Extract BDIA: Representing all dense diagonals with BDIA
  - Extract DIA and BDIA: Representing thick bands with BDIA, and thin bands with DIA
Decision Tree: Extract Blocks

- Feature collection
  - Compute the number of dense/sparse blocks per row
- Evaluation
  - Evaluate the estimated performance of each tree branch, and make decision
- Extraction
  - Extract blocks based on the evaluation decision
Extracting Blocks: Evaluation

- Definition of dense blocks
  - \( g_b \): maximum GFLOPS achievable by the blocked category at the current matrix settings
  - \( g_f \): maximum GFLOPS achievable by the flat category at the current matrix settings
  - \( n_b \): the size of a block
  - \( e_b \): # of non-zeros in a block
  - A block is considered dense if \( e_b > n_b g_f / g_b \)

- Decision tree branches
  - Extract SBELL: Representing all dense blocks/all non-zeros with SBELL
  - Extract BELL: Representing all dense blocks/all non-zeros with BELL
  - Extract BCSR: Representing all dense blocks/all non-zeros with BCSR
  - Extract None: Do not extract any dense blocks
We should extract regular # of non-zeros per row using ELL or SELL, then use CSR or COO to represent the remaining irregular non-zeros.

Feature collection
- Compute the number of non-zeros per row

Evaluation
- Evaluate the estimated performance of each tree branch, and make decision

Extraction
- Extract ELL or SELL parts based on the evaluation decision
Extracting ELL or SELL: Evaluation

- Decision tree branches
  - Extract ELL
    - \( w \): ELL width
    - \( z(w) \): zero paddings with width \( w \)
    - \( e(w) \): # of non-zeros covered with width \( w \)
    - \( r(w) \): # of remaining non-zeros not covered with width \( w \)
    - \( g_{ELL} \): achievable performance of ELL
    - \( m_c \): maximum achievable GFLOPS with CSR or COO formats
    - \( c \): # of columns of the matrix
    - Solve the following problem:
      \[
      \min \frac{(z(w) + e(w))}{g_{ELL}} + \frac{r(w)}{m_c} \]
      (the estimated execution time)
      s. t. \( w \leq c \)
      \( w \) is an integer
  - Extract SELL: Similar to ELL, but consider each slice separately
  - Extract None: Do not extract ELL or SELL portions
Decision Tree: Extract CSR or COO

- Feature collection
  - Compute the load balancing problem of the CSR format
- Evaluation
  - Evaluate the estimated performance of each tree branch, and make decision
- Extraction
  - Representing the remaining matrix with CSR or COO format based on the evaluation decision
Extracting CSR or COO: Evaluation

- Decision tree branches (CSR vs. COO)
  - $u$: # of work groups created in CSR
  - $n$: # of non-zeros
  - $nnz(i)$: # of non-zeros computed by work group $i$
  - $g_{CSR}$: achievable performance of CSR
  - $g_{COO}$: achievable performance of COO
  - Select CSR if the following criterion is met; select COO if the criterion is not met

\[
\frac{u \times \max_{1 \leq i \leq u} nnz(i)}{g_{CSR}} < \frac{n}{g_{COO}}
\]
Overhead of the Online Decision Making Stage

- Analysis and extraction cost
  - Diagonal analysis: 2 SpMV
  - Block analysis: 20 SpMV per block size
  - Flat analysis: 4 SpMV

- Block analysis dominates the online decision making stage

- Possible fixes
  - Let user to provide clues on the block dimension, and the uniformity of the number of dense blocks per row
    - Skip the entire analysis procedure, just do extraction
      - Might reduce the cost to 1-2 SpMV
  - Instead of analyzing the entire matrix, sample it
    - OSKI by Vuduc et al. achieves good performance based on this approach\(^1\)
  - Parallelize the analysis procedure
    - All the features are basically histogram accumulation, very likely to get 10-30x speedups

---

Outline

- Motivation
- The Cocktail Sparse Matrix Format
- The clSpMV Framework
- Experimental Results
- Conclusion
Experiment Setup

- The benchmarking sparse matrices
  - 14 matrices from William et al.’s 2007 SC paper\(^1\)
    - Most of them are regular, only one format is enough
  - 6 matrices from the University of Florida Sparse Matrix Collection
    - Choose irregular matrices

- clSpMV statistics
  - 9 sparse matrix formats
  - 107 kernels

- Experiment platform and comparison
  - Nvidia GTX 480
    - Compare to the Hybrid format from Nvidia’s 2009 SC paper\(^2\)
    - Compare to the best format from Nvidia’s 2009 SC paper\(^2\)
    - Compare to the best single format including Nvidia’s implementation and our implementation
  - AMD Radeon 6970
    - Compare to the best single format

---

Offline Benchmarking on Nvidia GTX 480
clSpMV Performance on Nvidia GTX 480: Regular Matrices

- Performance on 11 regular matrices
  - Only one format is chosen by clSpMV to represent these matrices
  - 114% better than the Nvidia Hybrid format
  - 48% better than the best Nvidia format
  - 0.5% worse than the best single format
### cIspMV Format Selection on Regular Matrices (GTX 480)

<table>
<thead>
<tr>
<th>Name</th>
<th>Spyplot</th>
<th>Dimension</th>
<th>Nonzeros (nnz/row)</th>
<th>Best Single Format</th>
<th>cIspMV Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td><img src="image0.png" alt="image" /></td>
<td>2kx2k</td>
<td>4M (2k)</td>
<td>BCSR</td>
<td>BCSR</td>
</tr>
<tr>
<td>Protein</td>
<td><img src="image1.png" alt="image" /></td>
<td>36kx36k</td>
<td>4.3M (119)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Spheres</td>
<td><img src="image2.png" alt="image" /></td>
<td>83kx83k</td>
<td>6M (72)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Cantilever</td>
<td><img src="image3.png" alt="image" /></td>
<td>62kx62k</td>
<td>4M (65)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Wind</td>
<td><img src="image4.png" alt="image" /></td>
<td>218kx218k</td>
<td>11.6M (53)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Harbor</td>
<td><img src="image5.png" alt="image" /></td>
<td>47kx47k</td>
<td>2.37M (50)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>QCD</td>
<td><img src="image6.png" alt="image" /></td>
<td>49kx49k</td>
<td>1.9M (39)</td>
<td><strong>SELL</strong></td>
<td><strong>ELL</strong></td>
</tr>
<tr>
<td>Ship</td>
<td><img src="image7.png" alt="image" /></td>
<td>141kx141k</td>
<td>3.98M (28)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Epidemiology</td>
<td><img src="image8.png" alt="image" /></td>
<td>526kx526k</td>
<td>2.1M (4)</td>
<td><strong>SELL</strong></td>
<td><strong>ELL</strong></td>
</tr>
<tr>
<td>Accelerator</td>
<td><img src="image9.png" alt="image" /></td>
<td>121kx121k</td>
<td>2.62M (22)</td>
<td>SBELL</td>
<td><strong>SELL</strong></td>
</tr>
<tr>
<td>LP</td>
<td><img src="image10.png" alt="image" /></td>
<td>4kx1.1M</td>
<td>11.3M (2825)</td>
<td>BCSR</td>
<td>BCSR</td>
</tr>
</tbody>
</table>
The performance on 9 irregular matrices
- clSpMV decides to partition the matrix into many submatrices
- 46% better than the Nvidia Hybrid format
- 29% better than the best Nvidia format
- 38% better than the best single format
# cISpMV Format Selection on Irregular Matrices (GTX 480)

<table>
<thead>
<tr>
<th>Name</th>
<th>Spyplot</th>
<th>Dimension</th>
<th>Nonzeros (nnz/row)</th>
<th>Best Single Format</th>
<th>cISpMV Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Economics</td>
<td><img src="image1" alt="Spyplot" /></td>
<td>207kx207k</td>
<td>1.27M (6)</td>
<td>SELL</td>
<td>ELL(81%) COO(19%)</td>
</tr>
<tr>
<td>Circuit</td>
<td><img src="image2" alt="Spyplot" /></td>
<td>171kx171k</td>
<td>959k (6)</td>
<td>SELL</td>
<td>ELL(84%) COO(16%)</td>
</tr>
<tr>
<td>Webbase</td>
<td><img src="image3" alt="Spyplot" /></td>
<td>1Mx1M</td>
<td>3.1M (3)</td>
<td>COO</td>
<td>ELL(64%) COO(36%)</td>
</tr>
<tr>
<td>Circuit5M</td>
<td><img src="image4" alt="Spyplot" /></td>
<td>5.56Mx5.56M</td>
<td>59.5M (11)</td>
<td>COO</td>
<td>DIA(9%) SELL(73%) COO(18%)</td>
</tr>
<tr>
<td>Eu-2005</td>
<td><img src="image5" alt="Spyplot" /></td>
<td>863Kx863K</td>
<td>19M (22)</td>
<td>SBELL</td>
<td>SELL(85%) COO(15%)</td>
</tr>
<tr>
<td>Ga41As41H72</td>
<td><img src="image6" alt="Spyplot" /></td>
<td>268kx268k</td>
<td>18M (67)</td>
<td>CSR</td>
<td>BDIA(18%) ELL(32%) CSR(50%)</td>
</tr>
<tr>
<td>in-2004</td>
<td><img src="image7" alt="Spyplot" /></td>
<td>1.38Mx1.38M</td>
<td>17M (12)</td>
<td>SBELL</td>
<td>SELL(79%) COO(21%)</td>
</tr>
<tr>
<td>mip1</td>
<td><img src="image8" alt="Spyplot" /></td>
<td>66Kx66K</td>
<td>10M (152)</td>
<td>CSR</td>
<td>SBELL(80%) SELL(17%) COO(3%)</td>
</tr>
<tr>
<td>Si41Ge41H72</td>
<td><img src="image9" alt="Spyplot" /></td>
<td>186kx186k</td>
<td>15M (81)</td>
<td>CSR</td>
<td>BDIA(15%) ELL(27%) CSR(58%)</td>
</tr>
</tbody>
</table>
Offline Benchmarking on AMD Radeon 6970
The performance on 9 regular matrices
- Only one format is chosen by clSpMV to represent these matrices
- 2% worse than the best single format
<table>
<thead>
<tr>
<th>Name</th>
<th>Spyplot</th>
<th>Dimension</th>
<th>Nonzeros (nnz/row)</th>
<th>Best Single Format</th>
<th>clSpMV Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td></td>
<td>2kx2k</td>
<td>4M (2k)</td>
<td>BCSR</td>
<td>BCSR</td>
</tr>
<tr>
<td>Spheres</td>
<td></td>
<td>83kx83k</td>
<td>6M (72)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Wind</td>
<td></td>
<td>218kx218k</td>
<td>11.6M (53)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Harbor</td>
<td></td>
<td>47kx47k</td>
<td>2.37M (50)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>QCD</td>
<td></td>
<td>49kx49k</td>
<td>1.9M (39)</td>
<td>SELL</td>
<td>BELL</td>
</tr>
<tr>
<td>Ship</td>
<td></td>
<td>141kx141k</td>
<td>3.98M (28)</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Epidemiology</td>
<td></td>
<td>526kx526k</td>
<td>2.1M (4)</td>
<td>ELL</td>
<td>ELL</td>
</tr>
<tr>
<td>Accelerator</td>
<td></td>
<td>121kx121k</td>
<td>2.62M (22)</td>
<td>SELL</td>
<td>SELL</td>
</tr>
<tr>
<td>LP</td>
<td></td>
<td>4kx1.1M</td>
<td>11.3M (2825)</td>
<td>BCSR</td>
<td>BCSR</td>
</tr>
</tbody>
</table>
The performance on 11 irregular matrices
- clSpMV decides to partition the matrix into many submatrices
  - On Nvidia 480, 9 matrices are considered regular
    - The huge gap between BDIA and other formats drives clSpMV to extract more BDIA regions on matrices
- 80% better than the best single format
## clSpMV Format Selection on Irregular Matrices (Radeion 6970)

<table>
<thead>
<tr>
<th>Name</th>
<th>Spyplot</th>
<th>Dimension</th>
<th>Nonzeros (nnz/row)</th>
<th>Best Single Format</th>
<th>clSpMV Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protein</td>
<td><img src="image1" alt="Image" /></td>
<td>36kx36k</td>
<td>4.3M (119)</td>
<td>SBELL</td>
<td>BDIA(43%)SBELL(57%)</td>
</tr>
<tr>
<td>Cantilever</td>
<td><img src="image2" alt="Image" /></td>
<td>62kx62k</td>
<td>4M (65)</td>
<td>DIA</td>
<td>BDIA(90%)ELL(10%)</td>
</tr>
<tr>
<td>Economics</td>
<td><img src="image3" alt="Image" /></td>
<td>207kx207k</td>
<td>1.27M (6)</td>
<td>SELL</td>
<td>ELL(81%)COO(19%)</td>
</tr>
<tr>
<td>Circuit</td>
<td><img src="image4" alt="Image" /></td>
<td>171kx171k</td>
<td>959k (6)</td>
<td>COO</td>
<td>ELL(84%)COO(16%)</td>
</tr>
<tr>
<td>Webbase</td>
<td><img src="image5" alt="Image" /></td>
<td>1Mx1M</td>
<td>3.1M (3)</td>
<td>COO</td>
<td>ELL(64%)COO(36%)</td>
</tr>
<tr>
<td>Circuit5M</td>
<td><img src="image6" alt="Image" /></td>
<td>5.56Mx5.56M</td>
<td>59.5M (11)</td>
<td>COO</td>
<td>DIA(9%)SELL(73%)COO(18%)</td>
</tr>
<tr>
<td>Eu-2005</td>
<td><img src="image7" alt="Image" /></td>
<td>863Kx863K</td>
<td>19M (22)</td>
<td>COO</td>
<td>SELL(85%)COO(15%)</td>
</tr>
<tr>
<td>Ga41As41H72</td>
<td><img src="image8" alt="Image" /></td>
<td>268kx268k</td>
<td>18M (67)</td>
<td>CSR</td>
<td>BDIA(18%)ELL(32%)CSR(50%)</td>
</tr>
<tr>
<td>in-2004</td>
<td><img src="image9" alt="Image" /></td>
<td>1.38Mx1.38M</td>
<td>17M (12)</td>
<td>COO</td>
<td>SELL(79%)COO(21%)</td>
</tr>
<tr>
<td>mip1</td>
<td><img src="image10" alt="Image" /></td>
<td>66Kx66K</td>
<td>10M (152)</td>
<td>BCSR</td>
<td>SBELL(80%)SELL(17%)COO(3%)</td>
</tr>
<tr>
<td>Si41Ge41H72</td>
<td><img src="image11" alt="Image" /></td>
<td>186kx186k</td>
<td>15M (81)</td>
<td>SBELL</td>
<td>BDIA(15%)ELL(27%)CSR(58%)</td>
</tr>
</tbody>
</table>
## clSpMV Format Selection on Different Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>clSpMV on GTX 480</th>
<th>clSpMV on Radeon 6970</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>BCSR</td>
<td>BCSR</td>
</tr>
<tr>
<td>Protein</td>
<td>SBELL</td>
<td>BDIA(43%) SBELL(57%)</td>
</tr>
<tr>
<td>Spheres</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Cantilevel</td>
<td>SBELL</td>
<td>BDIA(90%) ELL(10%)</td>
</tr>
<tr>
<td>Wind</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Harbor</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>QCD</td>
<td>ELL</td>
<td>BELL</td>
</tr>
<tr>
<td>Ship</td>
<td>SBELL</td>
<td>SBELL</td>
</tr>
<tr>
<td>Economics</td>
<td>ELL(81%) COO(19%)</td>
<td>ELL(88%) COO(12%)</td>
</tr>
<tr>
<td>Epidemiology</td>
<td>ELL</td>
<td>ELL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>clSpMV on GTX 480</th>
<th>clSpMV on Radeon 6970</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerator</td>
<td>SELL</td>
<td>SELL</td>
</tr>
<tr>
<td>Circuit</td>
<td>ELL(84%) COO(16%)</td>
<td>ELL(88%) COO(12%)</td>
</tr>
<tr>
<td>Webbase</td>
<td>ELL(64%) COO(36%)</td>
<td>ELL(70%) COO(30%)</td>
</tr>
<tr>
<td>LP</td>
<td>BCSR</td>
<td>BCSR</td>
</tr>
<tr>
<td>Circuit5M</td>
<td>DIA(9%) SELL(73%) COO(18%)</td>
<td>SELL(82%) COO(18%)</td>
</tr>
<tr>
<td>Eu-2005</td>
<td>SELL(85%) COO(15%)</td>
<td>ELL(83%) COO(17%)</td>
</tr>
<tr>
<td>Ga41As41H72</td>
<td>BDIA(18%) ELL(32%) CSR(50%)</td>
<td>BDIA(18%) ELL(32%) CSR(50%)</td>
</tr>
<tr>
<td>in-2004</td>
<td>SELL(79%) COO(21%)</td>
<td>SBELL(28%) ELL(53%) COO(19%)</td>
</tr>
<tr>
<td>mip1</td>
<td>SBELL(80%) SELL(17%) COO(3%)</td>
<td>BDIA(20%) SBELL(62%) SELL(14%) COO(4%)</td>
</tr>
<tr>
<td>Si41Ge41H72</td>
<td>BDIA(15%) ELL(27%) CSR(58%)</td>
<td>BDIA(15%) SBELL(85%)</td>
</tr>
</tbody>
</table>
Outline

- Motivation
- The Cocktail Sparse Matrix Format
- The clSpMV Framework
- Experimental Results
- Conclusion
Conclusion

- We proposed a new format for sparse matrices: the Cocktail Format that is a composition of many matrix formats
- We developed the clSpMV framework that can automatically tune the representation and implementation of SpMV on an input matrix
  - On regular matrices, it chooses one out of 9 formats and achieves similar performance compared with the best out of the 9 formats
  - On irregular matrices, it partitions the matrix into many submatrices, represents them using the Cocktail Format, and achieves significant speedups
- The general ideas behind the Cocktail Format and the clSpMV framework are applicable to all kinds of parallel platforms
  - We can expand the framework by plugging in implementations optimized for other platforms
- Code is available at
  - http://www.eecs.berkeley.edu/~subrian/clSpMV.html
Thank You