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

Usage of Sparse Matrix Vector Multiplication

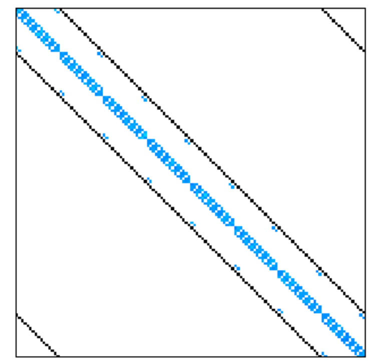
- Many iterative methods are composed of a BLAS2 operation with BLAS1 updates
  - BLAS2 operation dominates the execution time
- Many matrices are sparse in natural
  - We need to optimize the SpMV operation

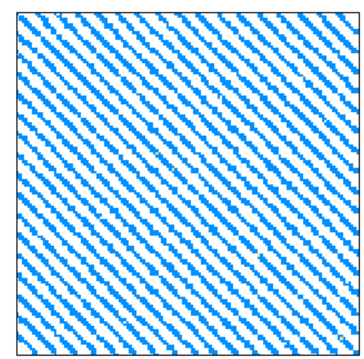
Algorithm: Conjugate Gradient
Input: A (Symmetric Matrix)
b (Vector)
x_0 (Initial Solution)
Output: x (Final Solution)

$r_0 \leftarrow b - Ax_0$;
$p_0 \leftarrow r_0$;
for $k \leftarrow 0, 1, \ldots,$ **until** convergence
  $v_k \leftarrow Ap_k$;
  $\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T v_k}$;
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$;
  $r_{k+1} \leftarrow r_k - \alpha_k v_k$;
  Test bounds for convergence ;
  $\beta_k \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$;
  $p_{k+1} \leftarrow r_{k+1} + \beta_k p_k$;
end for
Return x_{k+1} ;

Algorithm: Lanczos
Input: A (Symmetric Matrix)
v (Initial Vector)
Output: Θ (Ritz Values)
X (Ritz Vectors)

Start with $r \leftarrow v$;
$\beta_0 \leftarrow \|r\|_2$;
for $j \leftarrow 1, 2, \ldots,$ **until** convergence
  $v_j \leftarrow r/\beta_{j-1}$;
  $r \leftarrow Av_j$;
  $r \leftarrow r - v_{j-1}\beta_{j-1}$;
  $\alpha_j \leftarrow v_j^T r$;
  $r \leftarrow r - v_j \alpha_j$;
  Reorthogonalize if necessary ;
  $\beta_j \leftarrow \|r\|_2$;
  Compute Ritz values $T_j = S\Theta S$;
  Test bounds for convergence ;
end for
Compute Ritz vectors $X \leftarrow V_j S$;



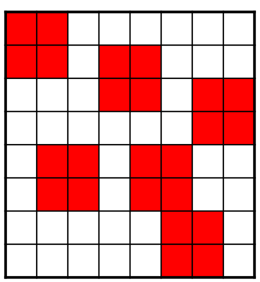


Optimizing the SpMV Computation

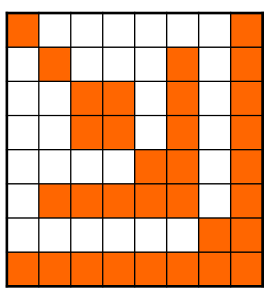
- Challenges of SpMV
  - Low arithmetic intensity (memory bounded)
  - Irregular memory access

- Minimizing memory footprint
  - Proposing new sparse matrix formats

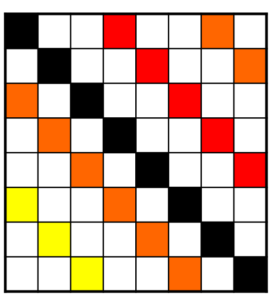
- Saturating memory bandwidth
  - Optimizing the memory access pattern on the memory system



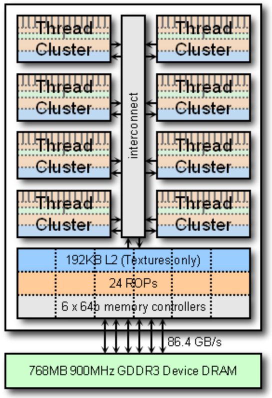
Block matrix



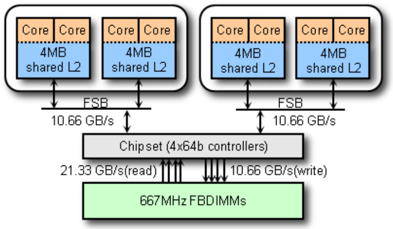
Symmetric



Diagonal



NVIDIA G80



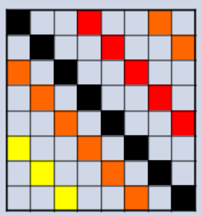
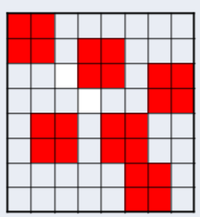
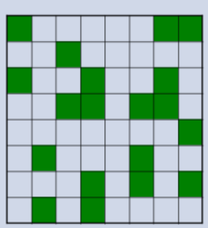
Intel Xeon E5345 (Clovertown)

Outline

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

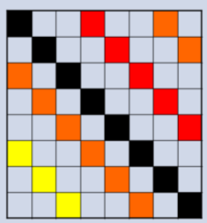
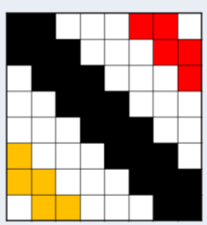
Pros and Cons of Matrix Formats

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

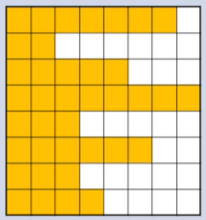
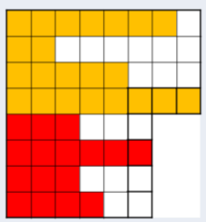
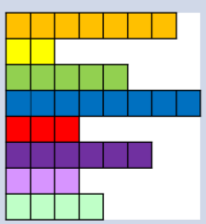
Matrix Format Category	Example Sparse Matrix	Included Matrix Formats	Pros	Cons	Suggested Usage
Diagonal		BDIA DIA	• Implicit column indices for diagonals • Aligned memory access pattern	• Need zero fillings on sparse diagonals	• Matrices that are mainly dense diagonals
Blocked		SBELL BELL BCSR	• Implicit column indices for blocks • Can reuse the multiplied vector	• Need zero fillings on sparse blocks	• Matrices that are mainly dense blocks
Flat		SELL ELL CSR COO	•No zero fillings	• Need explicit column indices • Unaligned memory access	• Irregular matrices

Pros and Cons of Diagonal-Based Formats

- DIA: Diagonal format
- BDIA: Banded DIA format

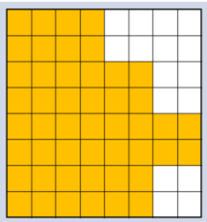
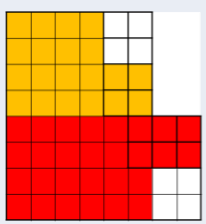
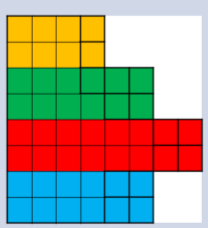
Matrix Format	Example Sparse Matrix	Pros	Cons	Suggested Usage
DIA		• More flexible on the width of the diagonals	• Cannot use shared memory to cache the vector	• Matrices with arbitrary dense diagonals
BDIA		• Can use shared memory to cache the vector	• Need extra storage to store the pointers to each band	• Matrices with dense bands

Pros and Cons of Flat Formats

Matrix Format	Example Matrix Storage	Pros	Cons	Suggested Usage
ELL		• Aligned memory access	• Need zero paddings	• Matrices with similar # of non-zero per row
SELL		• Aligned memory access • Fewer zero paddings	• Still need zero paddings • Additional pointers to slices	• Matrices with similar # of non-zero per slice
CSR		• No zero paddings	• Unaligned memory access • Bad load balance	• Matrices with moderate irregular # of non-zero per row
COO		• No zero paddings • Good load balance	• Explicit row indices	• Matrices with highly irregular # of non-zero per row

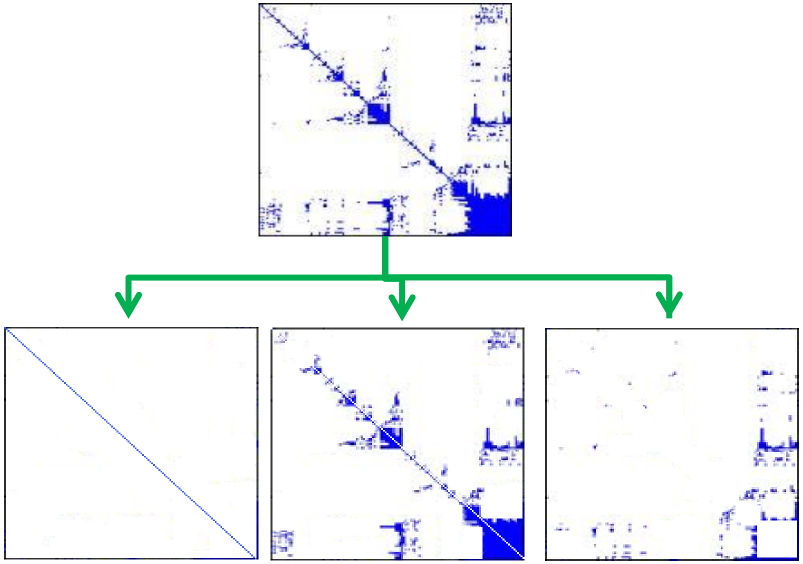
Pros and Cons of Blocked Formats

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

Matrix Format	Example Matrix Storage	Pros	Cons	Suggested Usage
BELL		• Aligned memory access	• Need zero paddings	• Matrices with similar # of blocks per blocked row
SBELL		• Aligned memory access • Fewer zero paddings	• Still need zero paddings • Additional pointers to slices	• 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, ..., 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

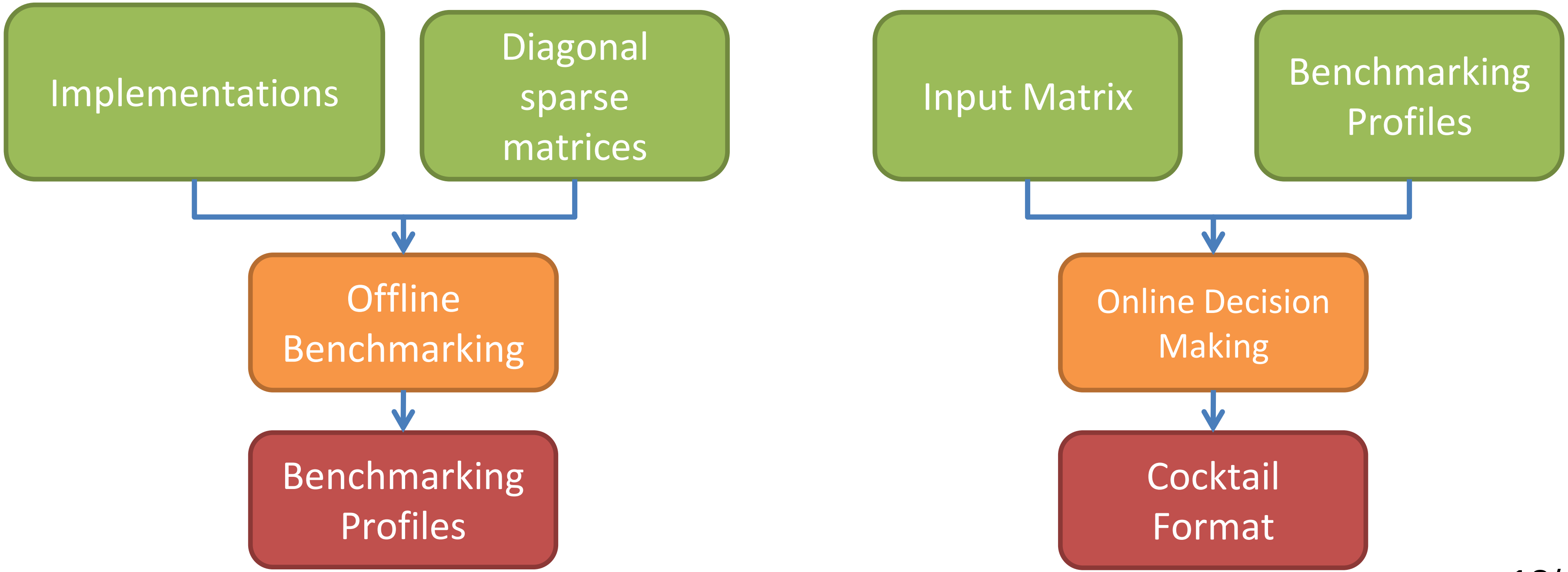
$$\begin{array}{ll} \min & \sum_{i=1}^{k} t(A_i, f_i, L_i) \\ s.t. & \sum_{i=1}^{k} A_i = A \\ & L_i \in P_i \quad \forall 1 \leq i \leq k \end{array}$$

Outline

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

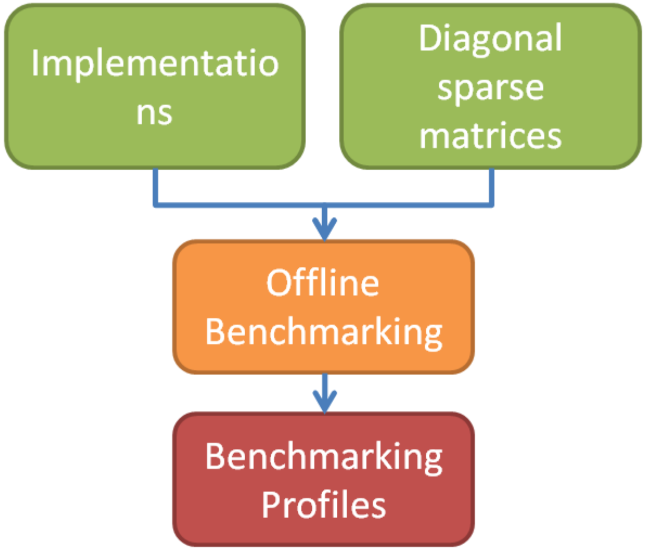
Overall Structure of clSpMV

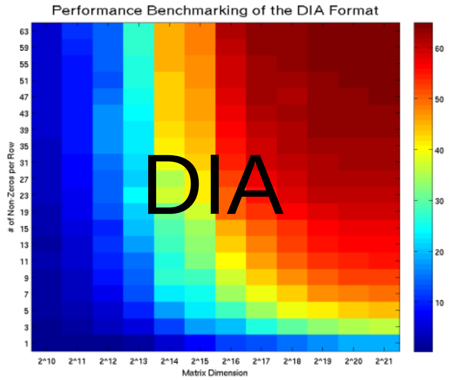
- Offline benchmarking
  - Used to estimate the $t(A_i, f_i, L_i)$ values
- Online decision making
  - Partition the input matrix according to the offline benchmarking profiles



Offline Benchmarking

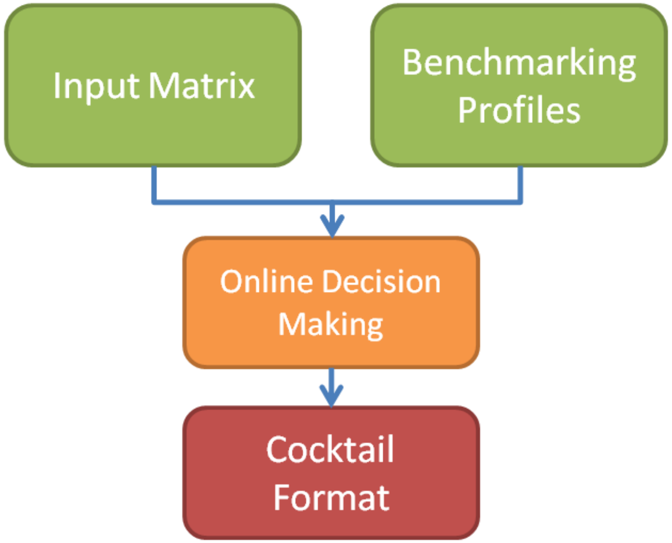
- One-time cost
- For every implementation of every format supported by clSpMV, sample the execution time on different sparse matrices
  - Sample on the matrix dimension and # non-zeros per row
  - Use interpolation to estimate $t(A_i, f_i, L_i)$ values in the online decision making stage
  - The estimation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





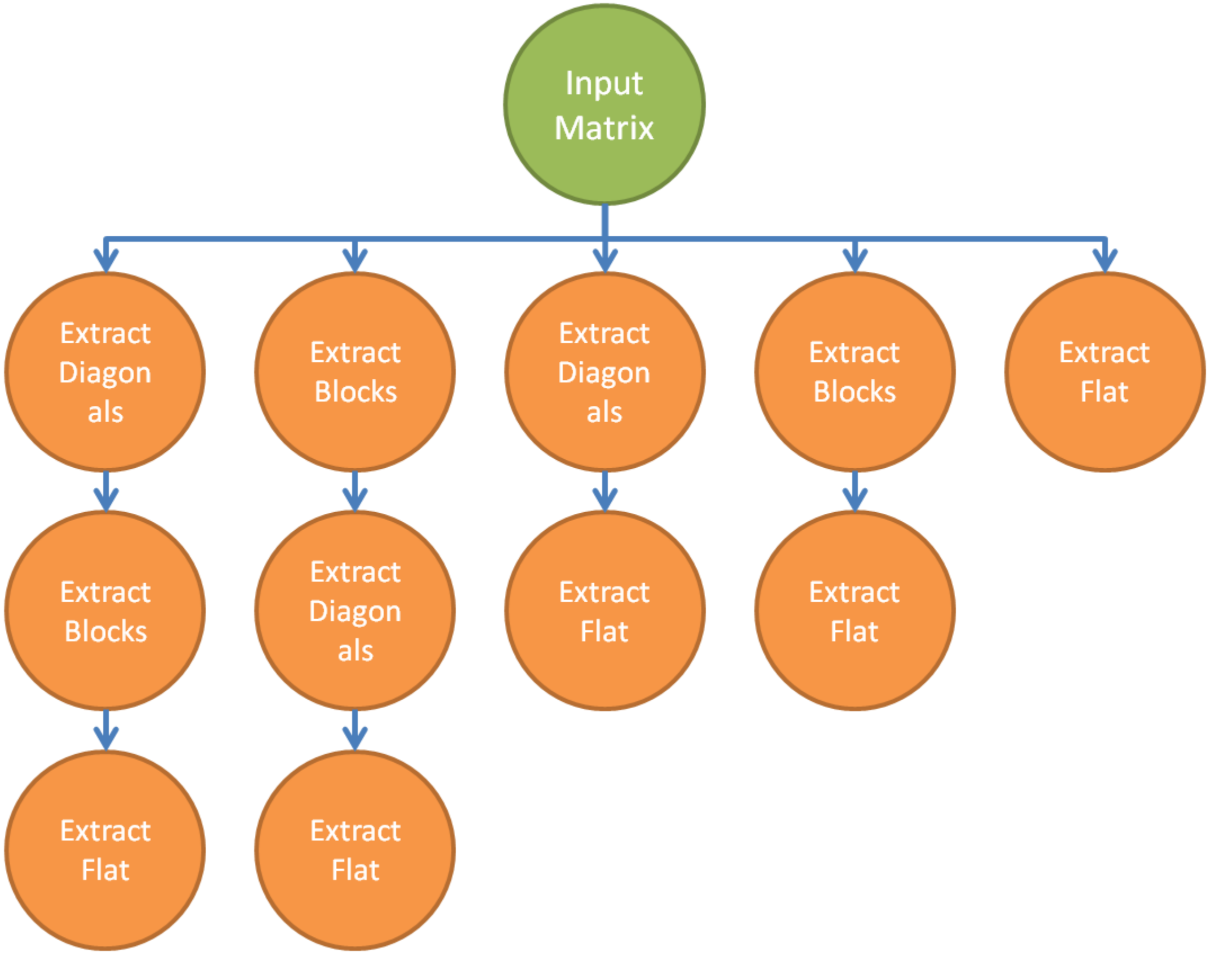
Online Decision Making

- 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



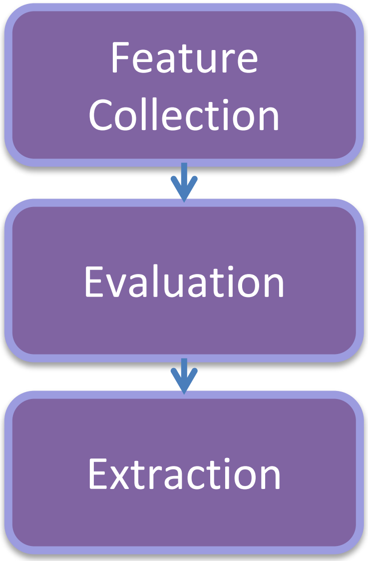
Decision Tree: Topmost Level

- Decide the priority of the matrix categories
  - Based on the highest estimated performance each category can achieve



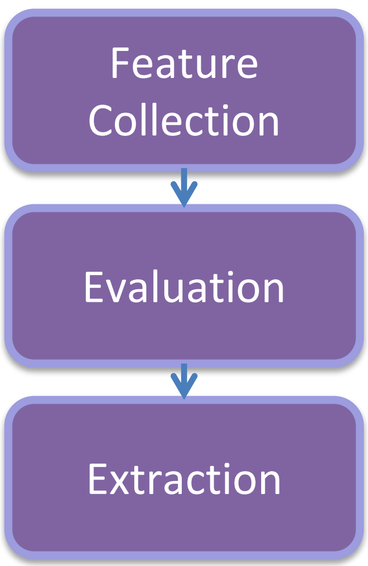
Extracting Submatrices from a Format Category

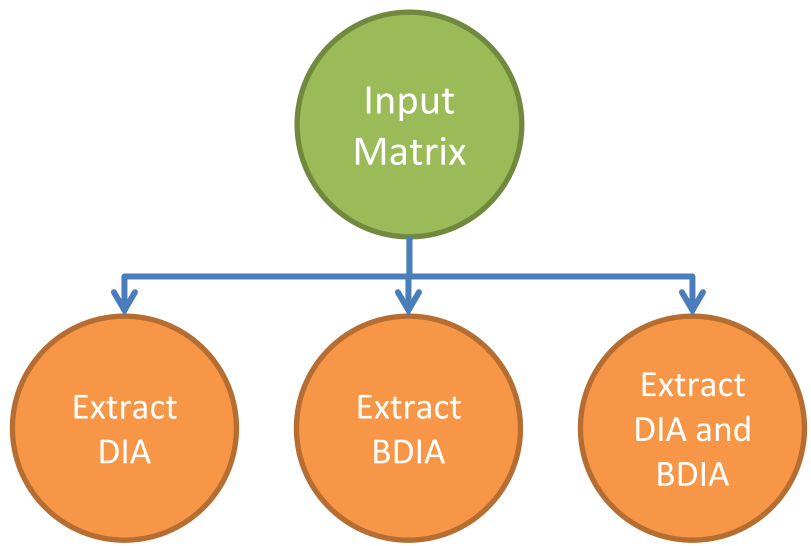
- Converting between formats is expensive
- Follow a three-step strategy
  - Feature collection: Collecting features that are able to differentiate performance of different formats in the same category
  - Evaluation: Estimating the performance of different partitioning scenarios, find the best scenario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



Decision Tree: Extract Diagonals

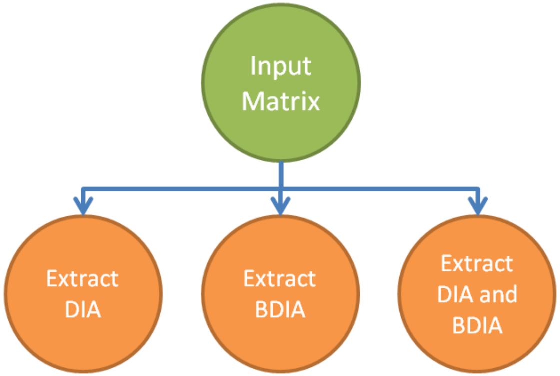
- Feature collection
  - Compute the number of non-zeros per diagonal
- Evaluation
  - Evaluate the estimated performance of each tree branch, and make decision
- Extraction
  - Extract diagonals or bands based on the evaluation decision





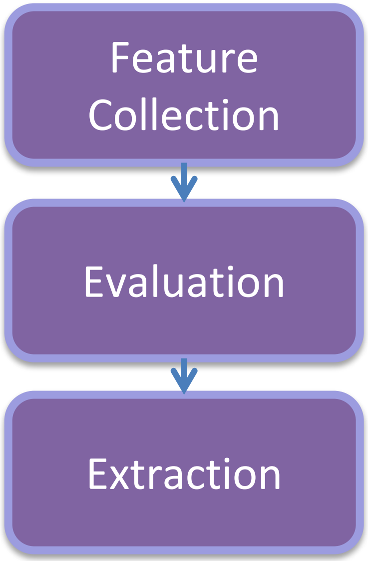
Extracting Diagonals: Evaluation

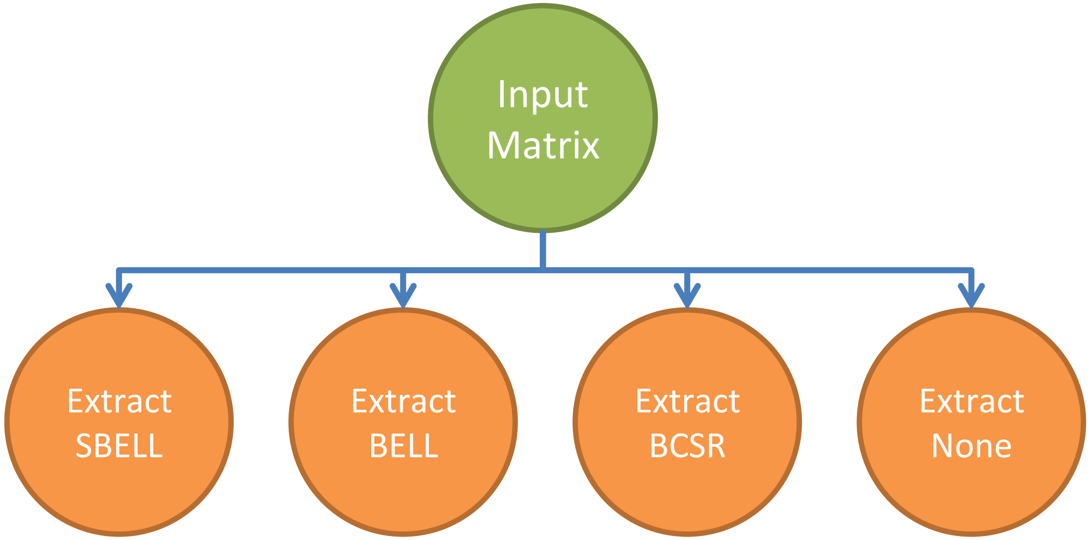
- Definition of dense diagonals
  - g_d: maximum GFLOPS achievable by the diagonal category at the current matrix settings
  - g_f: maximum GFLOPS achievable by the flat category at the current matrix settings
  - n_d: the dimension of a diagonal
  - e_d: # of non-zeros in a diagonal
  - A diagonal is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

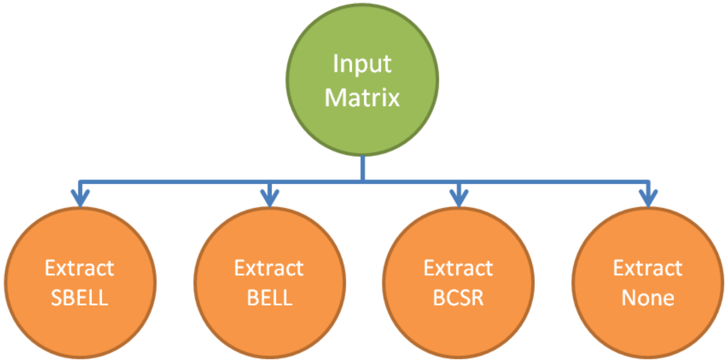
- Feature collection
  - Compute the number of dense/sparse blocks per row
- Evaluation
  - Evaluate the estimated performance of each tree branch, and make decision
- Extraction
  - Extract blocks based on the evaluation decision





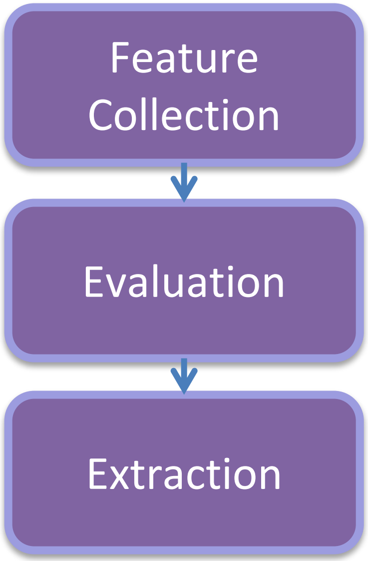
Extracting Blocks: Evaluation

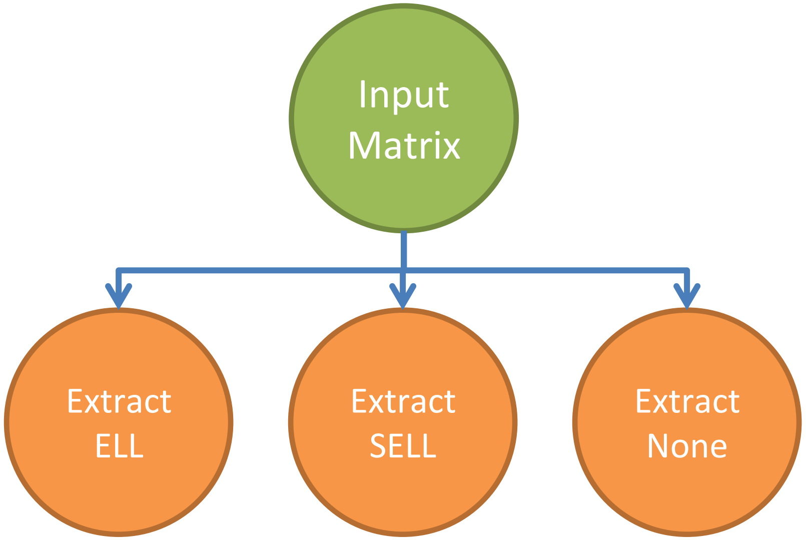
- Definition of dense blocks
  - g_b: maximum GFLOPS achievable by the blocked category at the current matrix settings
  - g_f: maximum GFLOPS achievable by the flat category at the current matrix settings
  - n_b: the size of a block
  - e_b: # of non-zeros in a block
  - A block is considered dense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



Decision Tree: Extract ELL or SELL

- 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





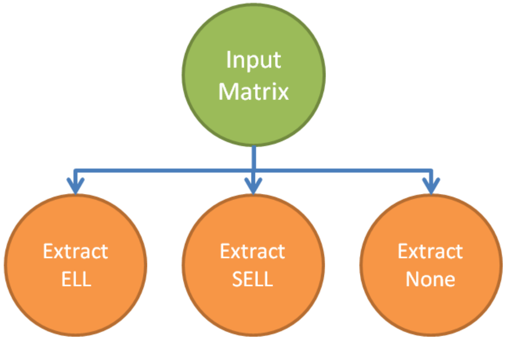
Extracting ELL or SELL: Evaluation

- 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 achivable GFLOPS with CSR or COO formats
    - c : # of columns of the matrix
    - Solve the following problem:

    min $(z(w)+e(w))/g_{ELL} + r(w)/m_c$ (the estimated execution time)

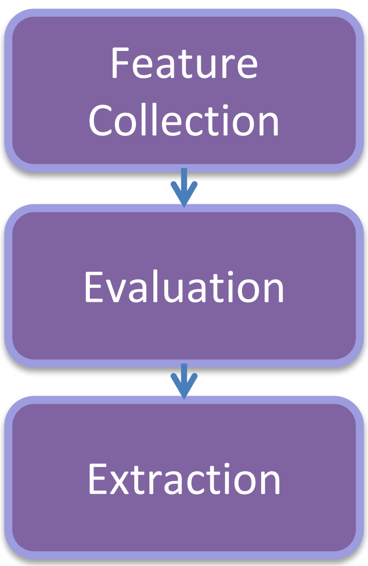
    s. t. $w <= c$

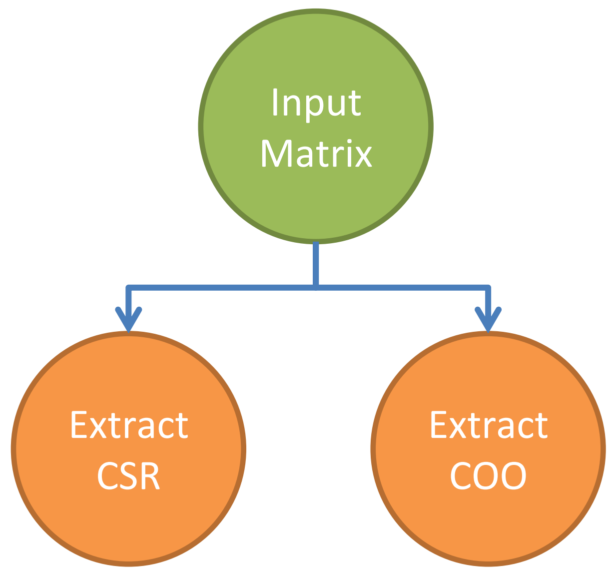
    w is an integer
  - Extract SELL: Similar to ELL, but consider each slice separately
  - Extract None: Do not extract ELL or SELL portions



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

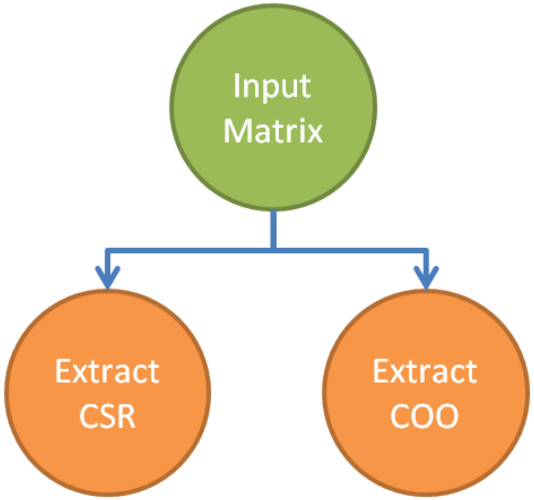




Extracting CSR or COO: Evaluation

- 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

$$\frac{u \times \max\limits_{1 \le i \le u} nnz(i)}{g_{CSR}} < \frac{n}{g_{COO}}$$



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

1. R. Vuduc, J. W. Demmel, and K. A. Yelick. Oski: A library of automatically tuned sparse matrix kernels. In Proceedings of SciDAC 2005, Journal of Physics: Conference Series, June 2005.

Outline

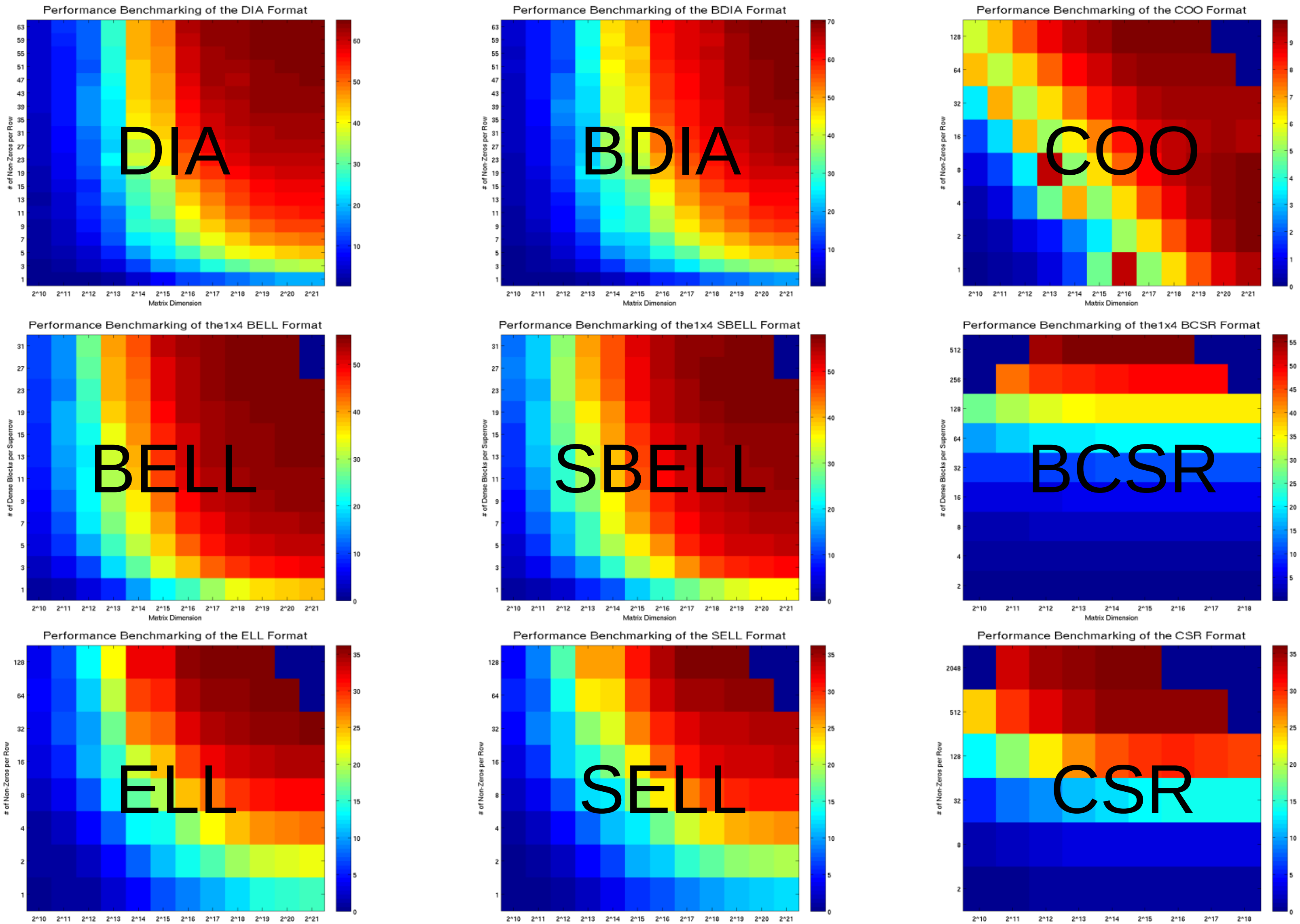
- Motivation
- The Cocktail Sparse Matrix Format
- The clSpMV Framework
- Experimental Results
- Conclusion

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

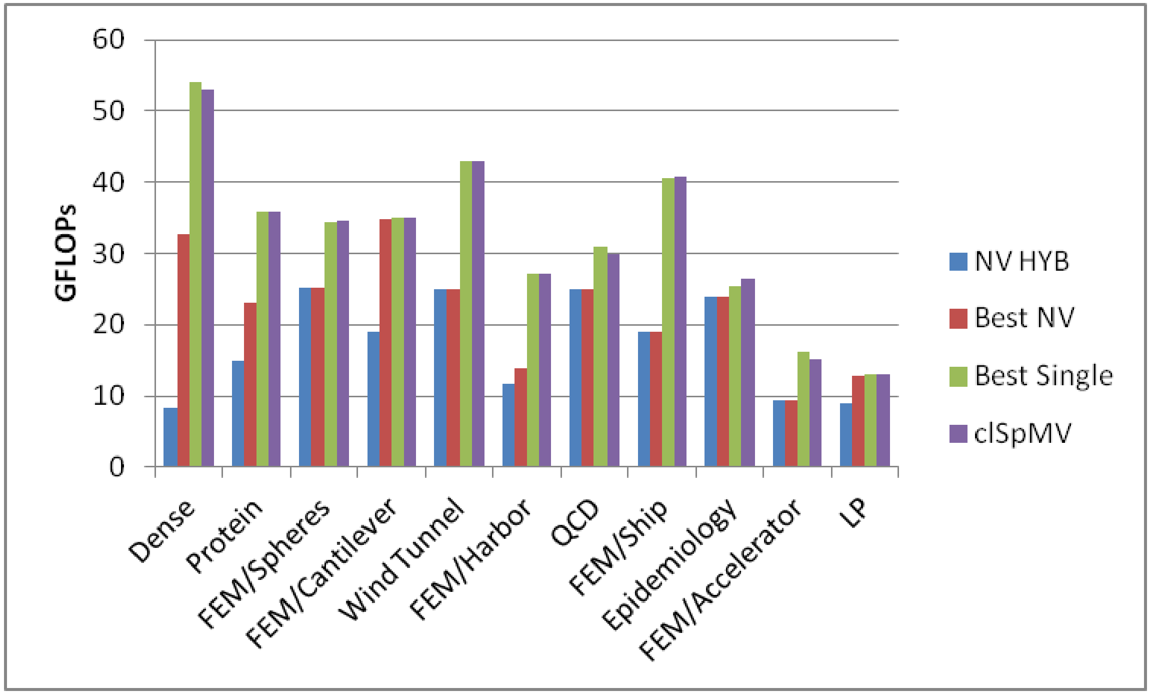
1. S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In Proceedings of the ACM/IEEE conference on Supercomputing, pages 38:1–38:12, New York, USA, 2007.
2. N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pages 18:1–18:11, New York, USA, 2009.

Offline Benchmarking on Nvidia GTX 480

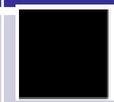
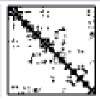
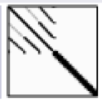
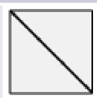
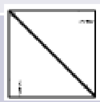
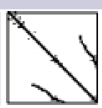
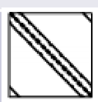
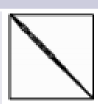
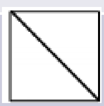
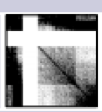


clSpMV Performance on Nvidia GTX 480: Regular Matrices

- 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

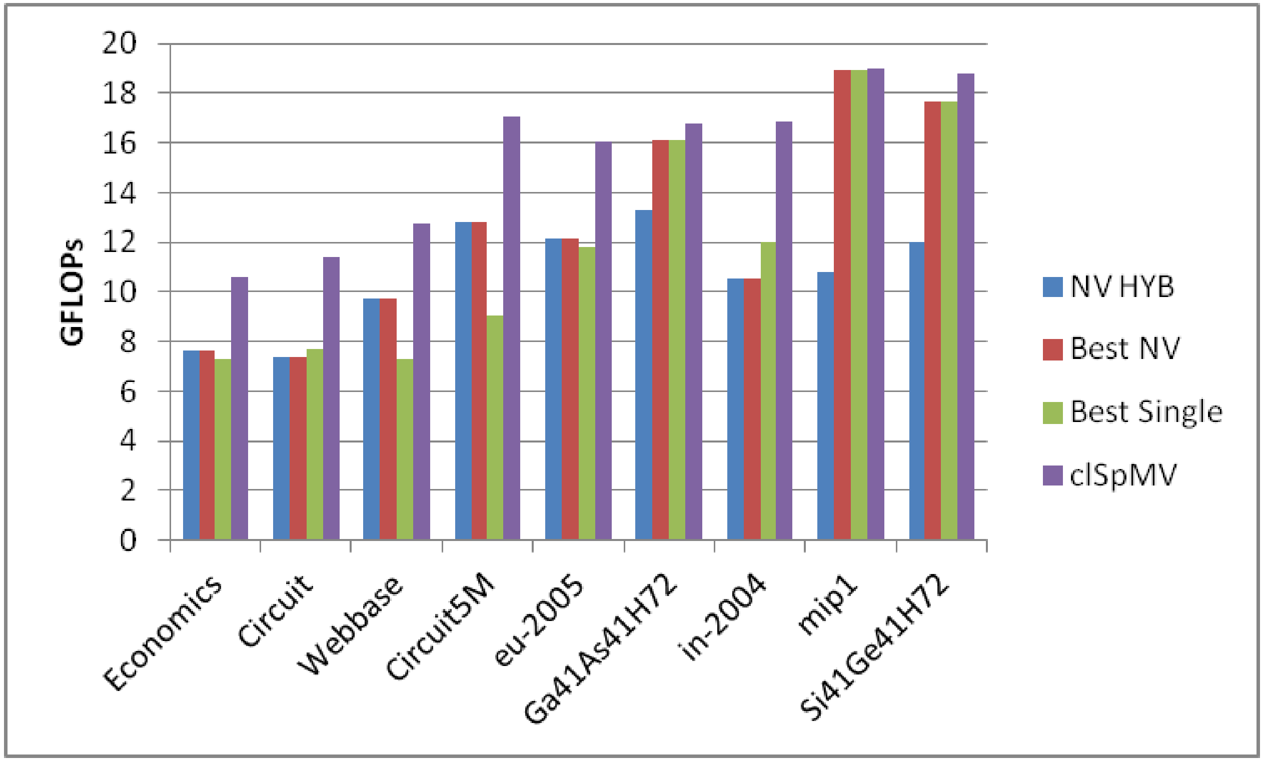


clSpMV Format Selection on Regular Matrices (GTX 480)

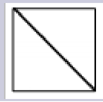
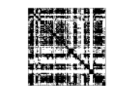
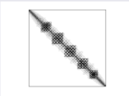
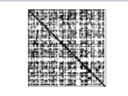
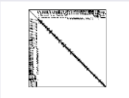
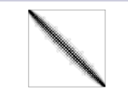
Name	Spyplot	Dimension	Nonzeros (nnz/row)	Best Single Format	clSpMV Format
Dense		2kx2k	4M (2k)	BCSR	BCSR
Protein		36kx36k	4.3M (119)	SBELL	SBELL
Spheres		83kx83k	6M (72)	SBELL	SBELL
Cantilever		62kx62k	4M (65)	SBELL	SBELL
Wind		218kx218k	11.6M (53)	SBELL	SBELL
Harbor		47kx47k	2.37M (50)	SBELL	SBELL
QCD		49kx49k	1.9M (39)	SELL	ELL
Ship		141kx141k	3.98M (28)	SBELL	SBELL
Epidemiology		526kx526k	2.1M (4)	SELL	ELL
Accelerator		121kx121k	2.62M (22)	SBELL	SELL
LP		4kx1.1M	11.3M (2825)	BCSR	BCSR

clSpMV Performance on Nvidia GTX 480: Irregular Matrices

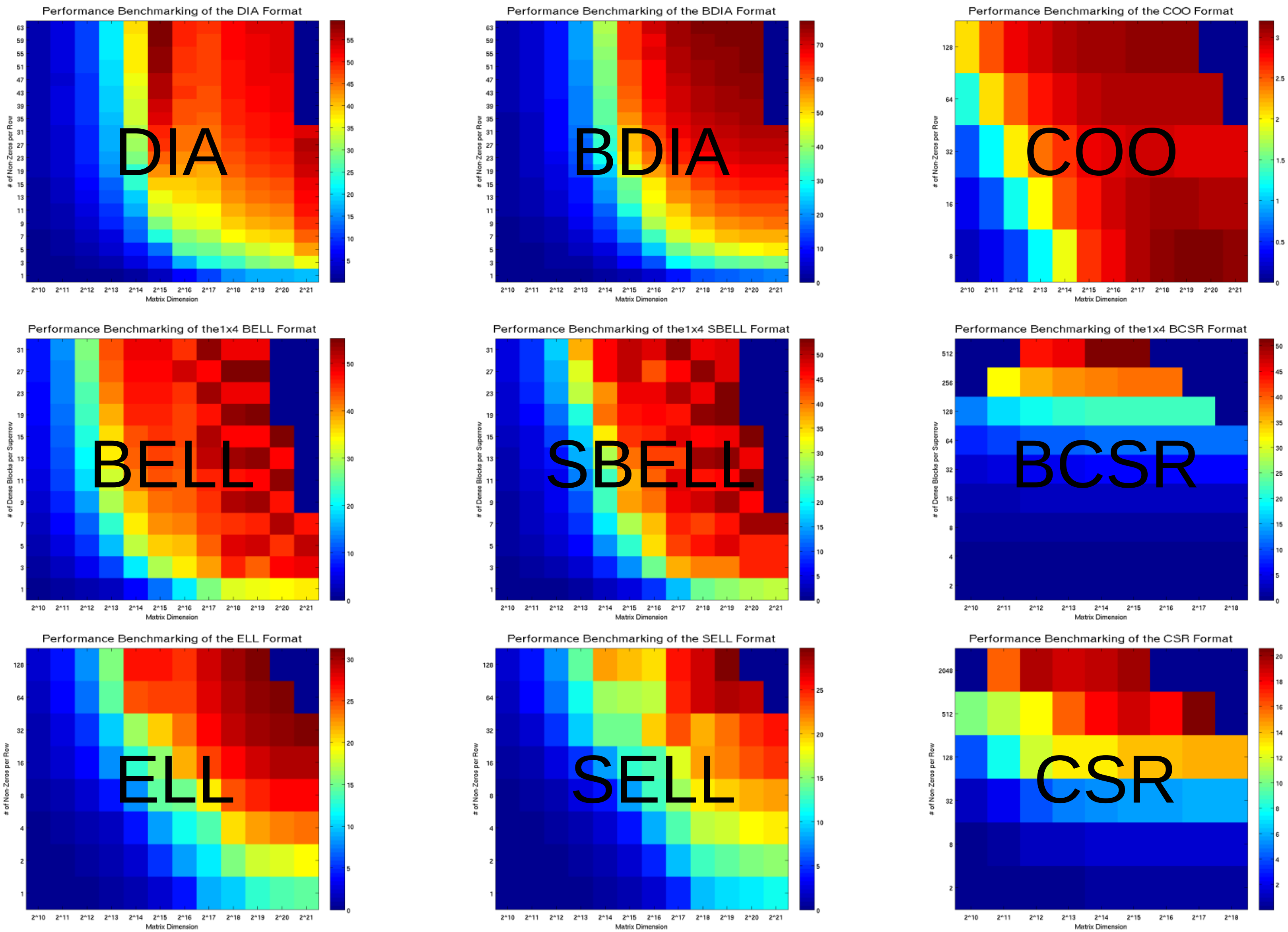
- The performance on 9 irregular matrices
  - clSpMV decides to partition the matrix into many submatrices
  - 46% better than the Nvidia Hybrid format
  - 29% better than the best Nvidia format
  - 38% better than the best single format



clSpMV Format Selection on Irregular Matrices (GTX 480)

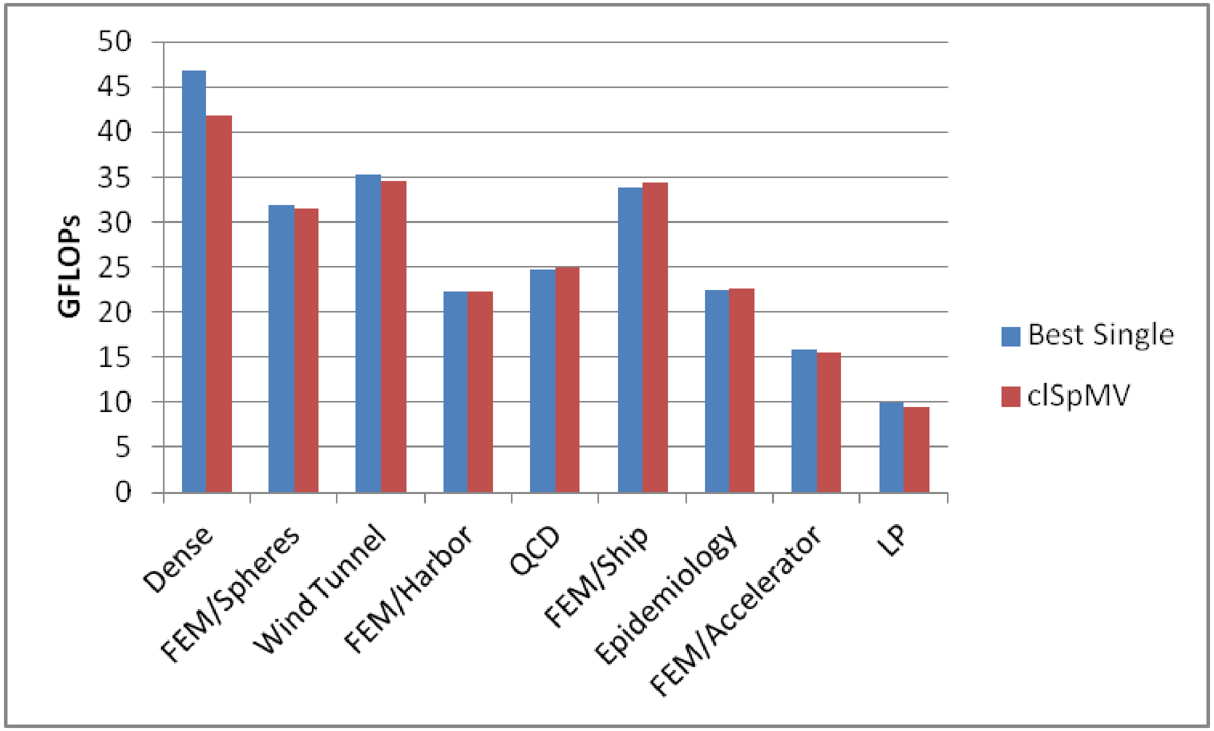
Name	Spyplot	Dimension	Nonzeros (nnz/row)	Best Single Format	clSpMV Format
Economics		207kx207k	1.27M (6)	SELL	ELL(81%) COO(19%)
Circuit		171kx171k	959k (6)	SELL	ELL(84%) COO(16%)
Webbase		1Mx1M	3.1M (3)	COO	ELL(64%) COO(36%)
Circuit5M		5.56Mx5.56M	59.5M (11)	COO	DIA(9%)SELL(73%)COO(18%)
Eu-2005		863Kx863K	19M (22)	SBELL	SELL(85%) COO(15%)
Ga41As41H72		268kx268k	18M (67)	CSR	BDIA(18%)ELL(32%)CSR(50%)
in-2004		1.38Mx1.38M	17M (12)	SBELL	SELL(79%) COO(21%)
mip1		66Kx66K	10M (152)	CSR	SBELL(80%)SELL(17%)COO(3%)
Si41Ge41H72		186kx186k	15M (81)	CSR	BDIA(15%)ELL(27%)CSR(58%)

Offline Benchmarking on AMD Radeon 6970

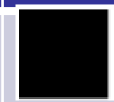
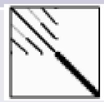
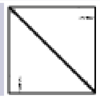
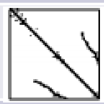
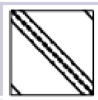
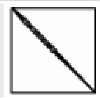
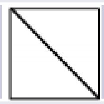
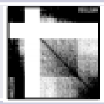


clSpMV Performance on AMD Radeon 6970: Regular Matrices

- The performance on 9 regular matrices
  - Only one format is chosen by clSpMV to represent these matrices
  - 2% worse than the best single format

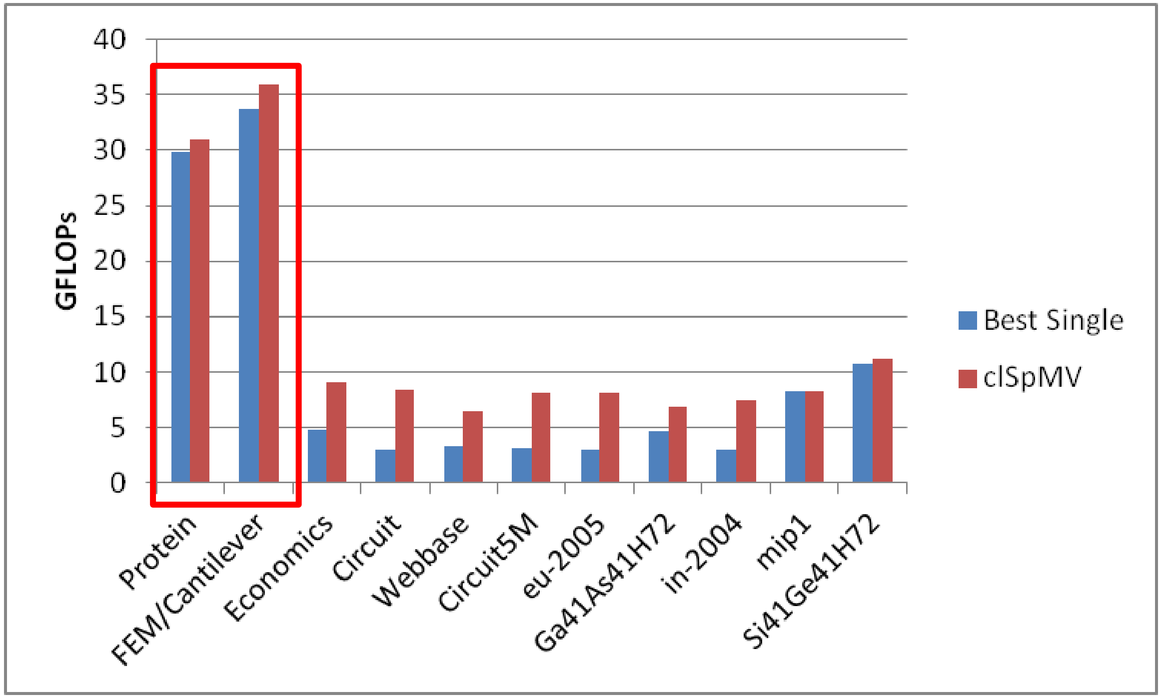


clSpMV Format Selection on Regular Matrices (Radeon 6970)

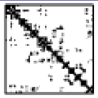
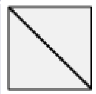
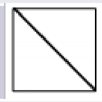
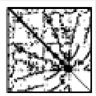
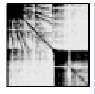
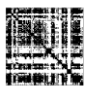
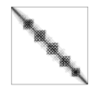
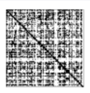
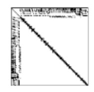
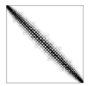
Name	Spyplot	Dimension	Nonzeros (nnz/row)	Best Single Format	clSpMV Format
Dense		2kx2k	4M (2k)	BCSR	BCSR
Spheres		83kx83k	6M (72)	SBELL	SBELL
Wind		218kx218k	11.6M (53)	SBELL	SBELL
Harbor		47kx47k	2.37M (50)	SBELL	SBELL
QCD		49kx49k	1.9M (39)	SELL	BELL
Ship		141kx141k	3.98M (28)	SBELL	SBELL
Epidemiology		526kx526k	2.1M (4)	ELL	ELL
Accelerator		121kx121k	2.62M (22)	SELL	SELL
LP		4kx1.1M	11.3M (2825)	BCSR	BCSR

clSpMV Performance on AMD Radeon 6970: Irregular Matrices

- 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)

Name	Spyplot	Dimension	Nonzeros (nnz/row)	Best Single Format	clSpMV Format
Protein		36kx36k	4.3M (119)	SBELL	BDIA(43%)SBELL(57%)
Cantilever		62kx62k	4M (65)	DIA	BDIA(90%) ELL(10%)
Economics		207kx207k	1.27M (6)	SELL	ELL(81%) COO(19%)
Circuit		171kx171k	959k (6)	COO	ELL(84%) COO(16%)
Webbase		1Mx1M	3.1M (3)	COO	ELL(64%) COO(36%)
Circuit5M		5.56Mx5.56M	59.5M (11)	COO	DIA(9%)SELL(73%)COO(18%)
Eu-2005		863Kx863K	19M (22)	COO	SELL(85%) COO(15%)
Ga41As41H72		268kx268k	18M (67)	CSR	BDIA(18%)ELL(32%) CSR(50%)
in-2004		1.38Mx1.38M	17M (12)	COO	SELL(79%) COO(21%)
mip1		66Kx66K	10M (152)	BCSR	SBELL(80%)SELL(17%)COO(3%)
Si41Ge41H72		186kx186k	15M (81)	SBELL	BDIA(15%)ELL(27%) CSR(58%)

clSpMV Format Selection on Different Platforms

Name	clSpMV on GTX 480	clSpMV on Radeon 6970
Dense	BCSR	BCSR
Protein	SBELL	BDIA(43%) SBELL(57%)
Spheres	SBELL	SBELL
Cantilevel	SBELL	BDIA(90%) ELL(10%)
Wind	SBELL	SBELL
Harbor	SBELL	SBELL
QCD	ELL	BELL
Ship	SBELL	SBELL
Economics	ELL(81%) COO(19%)	ELL(88%) COO(12%)
Epidemiology	ELL	ELL

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

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