Alleviating memory-bandwidth limitations for scalability and energy efficiency

Lessons learned from the optimization of SpMxV

Georgios Goumas
goumas@cslab.ece.ntua.gr
Computing Systems Laboratory
National Technical University of Athens

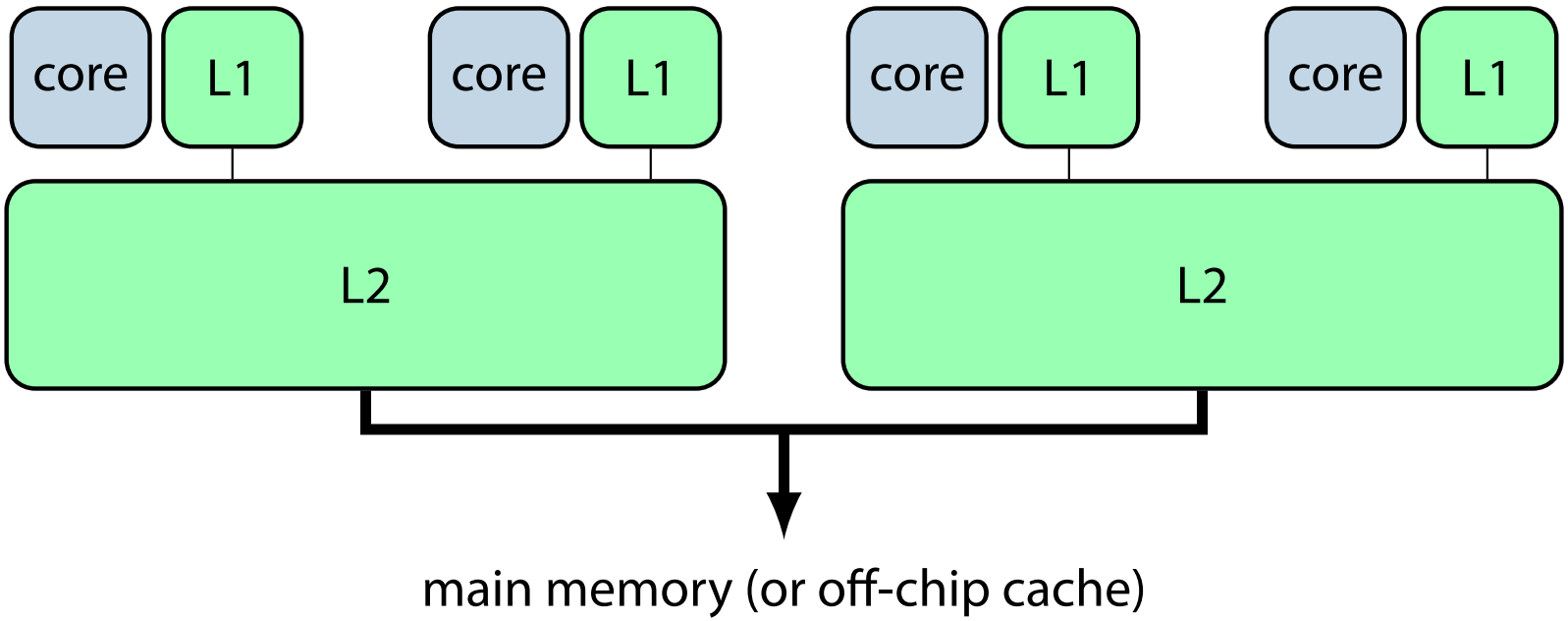
Oct 3, 2013
Parallel Processing for Energy Efficiency

Outline

1. Compression as an approach to scale up memory-bound applications
2. Sparse Matrices and SpMxV
3. CSX: A new storage format for sparse matrices
4. Conclusions – Areas of future research – Discussion

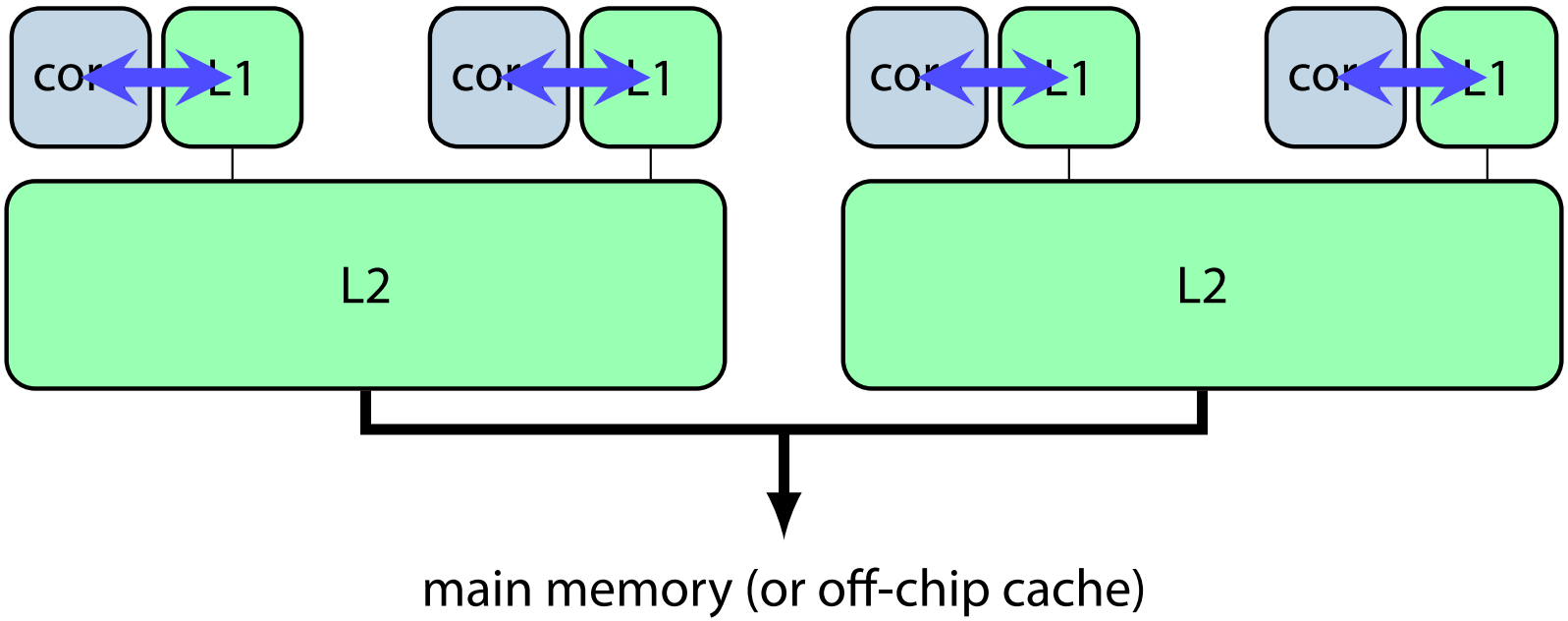
Outline

Application classes
(based on their performance on shared memory systems)



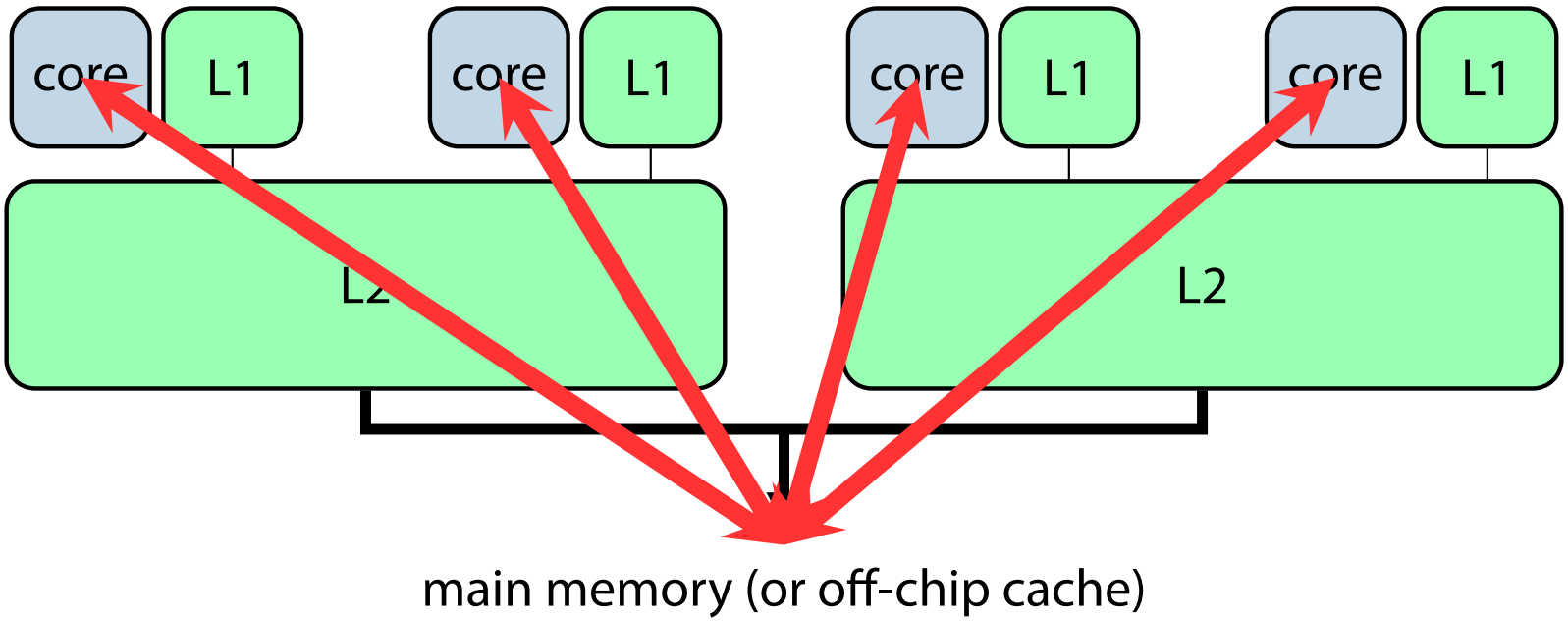
Application classes
(based on their performance on shared memory systems)

- ✔ Good scalability
  - ✔ temporal locality
  - ✔ no synchronization
  - ✔ load balance



Application classes
(based on their performance on shared memory systems)

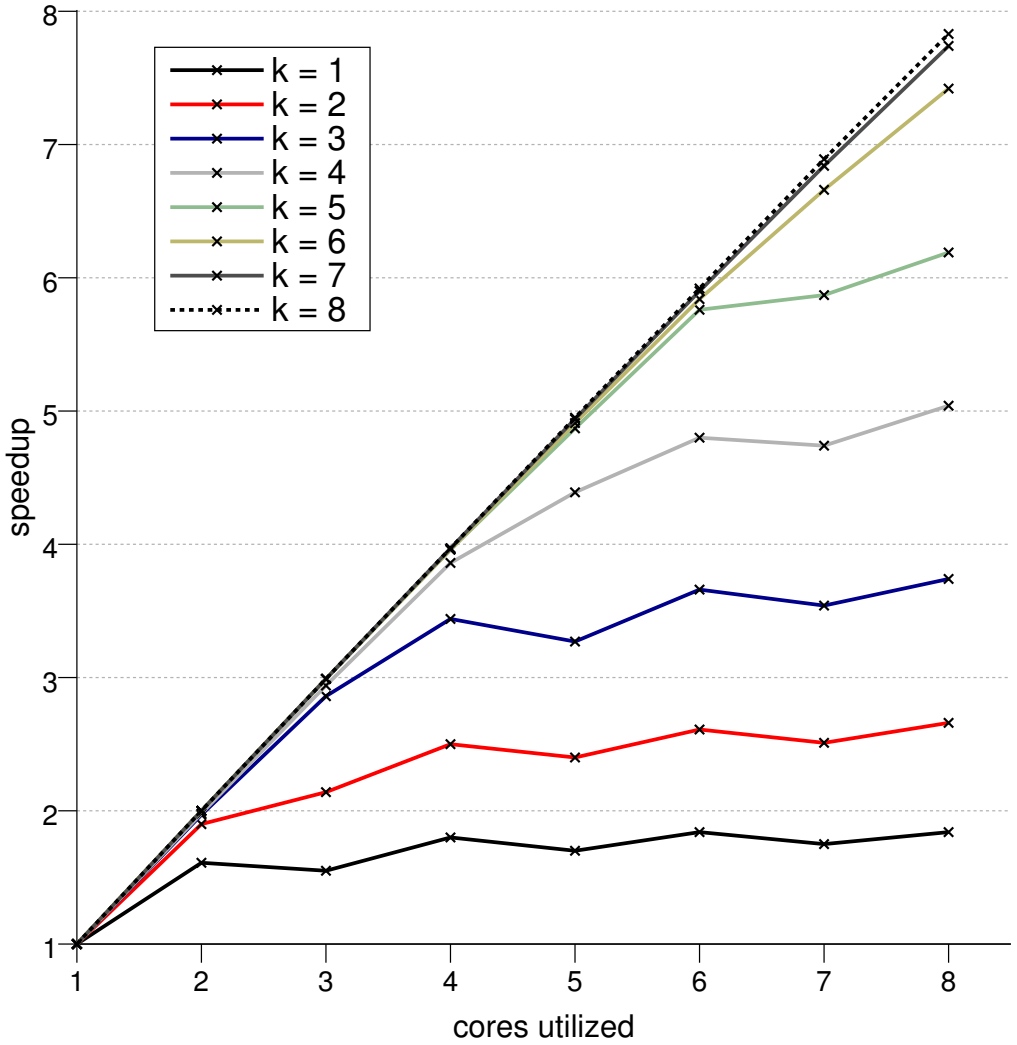
- ✗ Applications with intensive memory accesses
  - ✗ (very) poor temporal locality
  - ✗ high memory-to-computation ratio
  - ✗ limited scalability due to contention on memory



Applications with intensive memory accesses
(Example: memcomp benchmark)

memcomp

- 1 LOAD
- k ADDs
- FP (double)
- unrolled

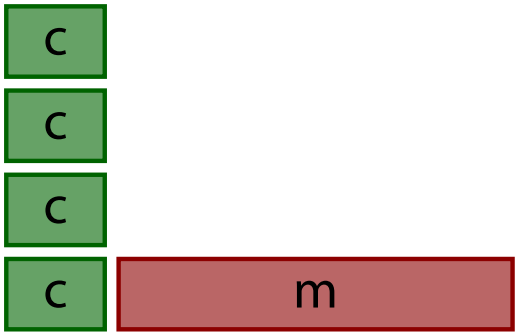


Improving performance using compression

exchange memory cycles for CPU cycles

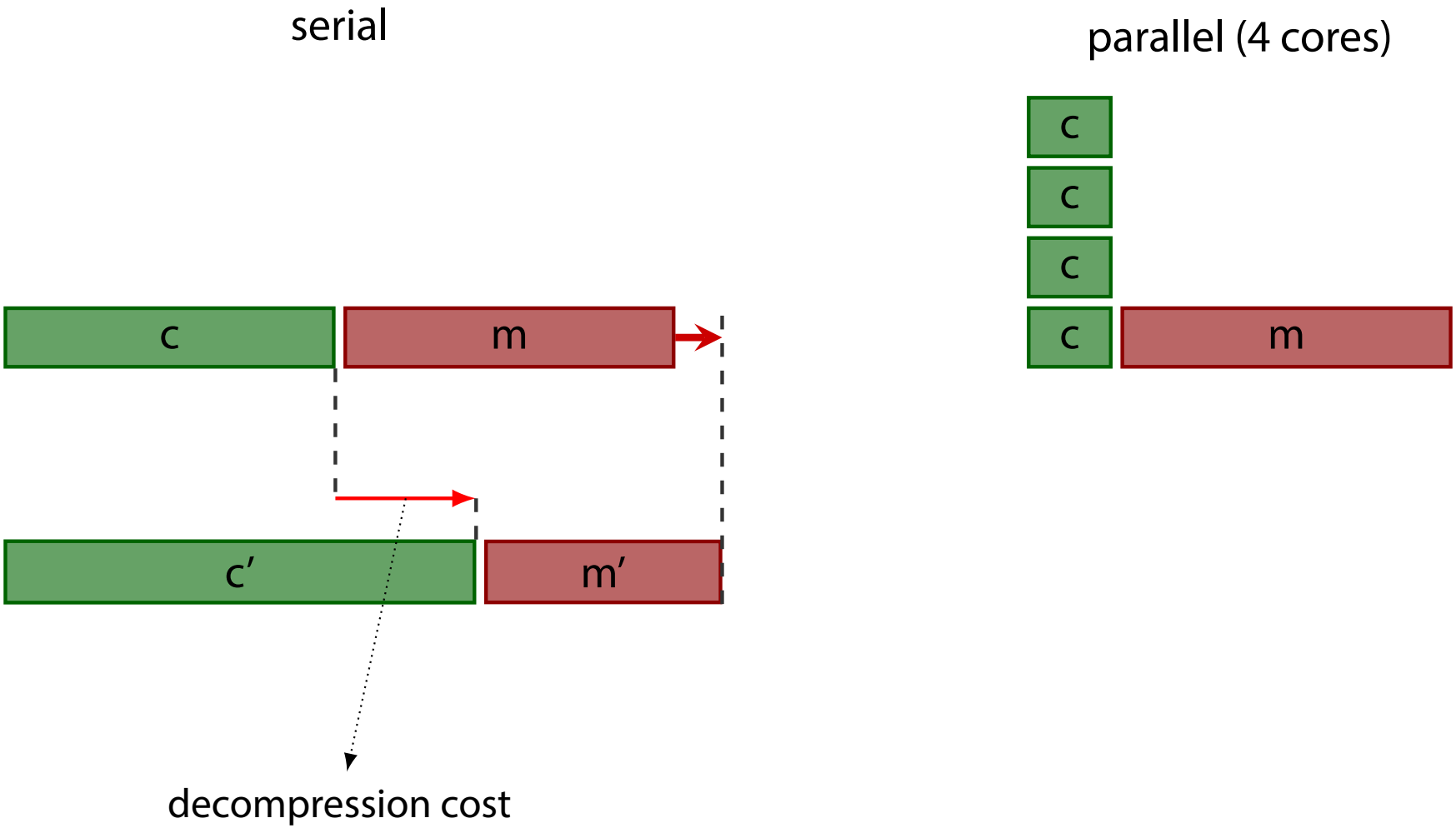
serial

parallel (4 cores)



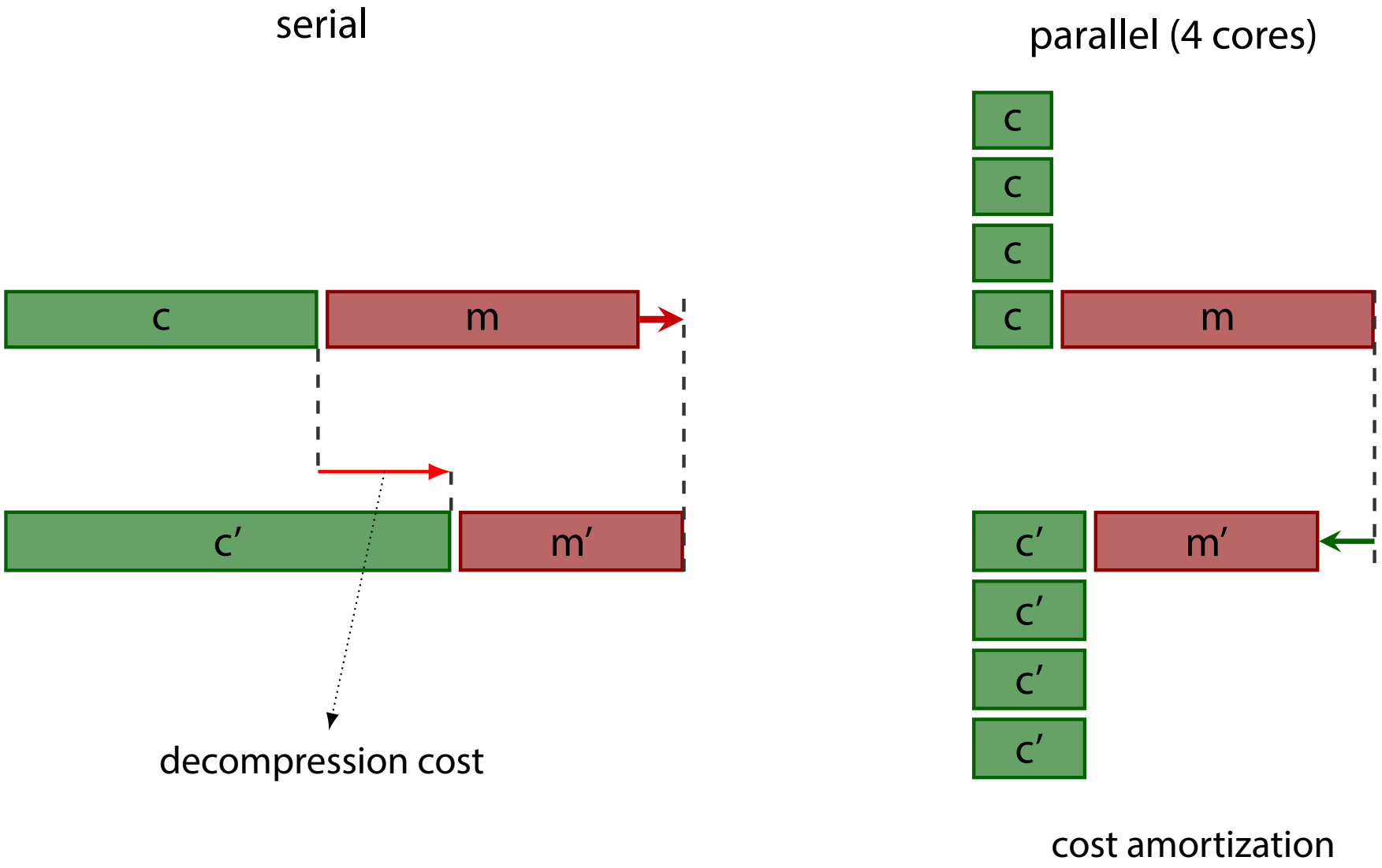
Improving performance using compression

exchange memory cycles for CPU cycles



Improving performance using compression

exchange memory cycles for CPU cycles

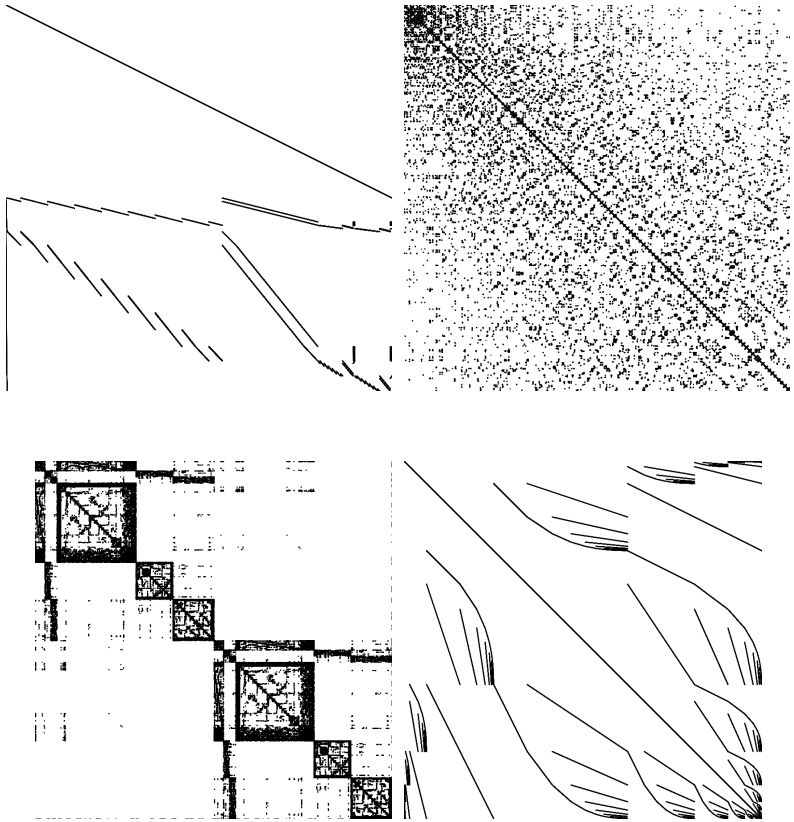


Outline

1. Compression as an approach to scale up memory-bound applications
2. Sparse Matrices and SpMxV
   - Storage formats
   - Sparse-matrix vector multiplication: SpMxV
   - SpMxV performance
3. CSX: A new storage format for sparse matrices
4. Conclusions – Areas of future research – Discussion

Sparse Matrices

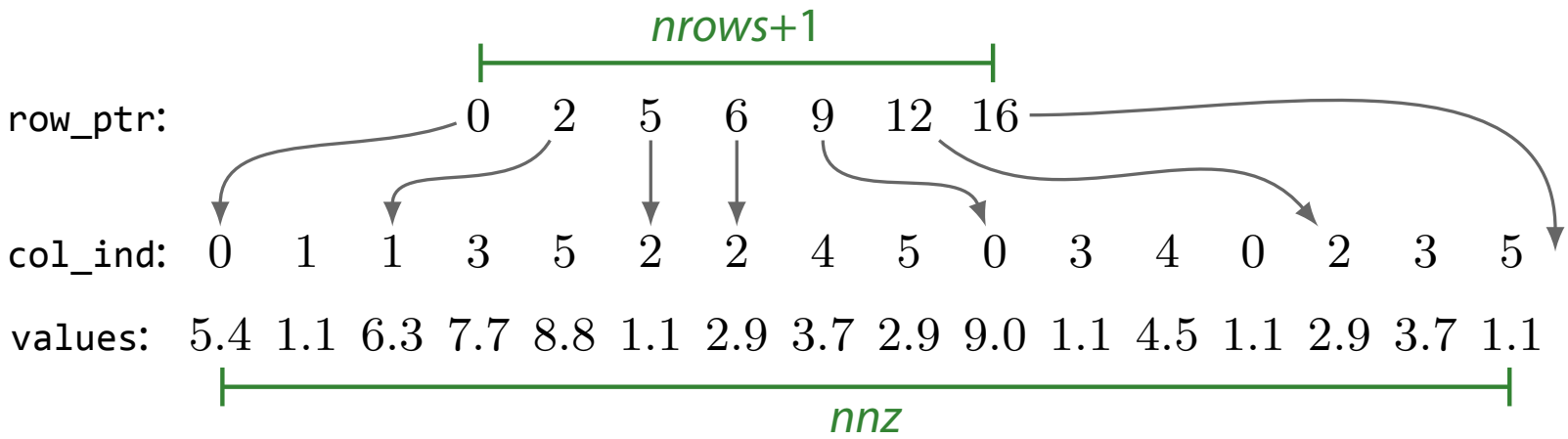
- Dominated by zeroes
- Applications:
  - PDEs
  - Graphs
  - Linear Programming
- Efficient Representation (space and computation)
  - non-zero values (value data)
  - structure (index data)



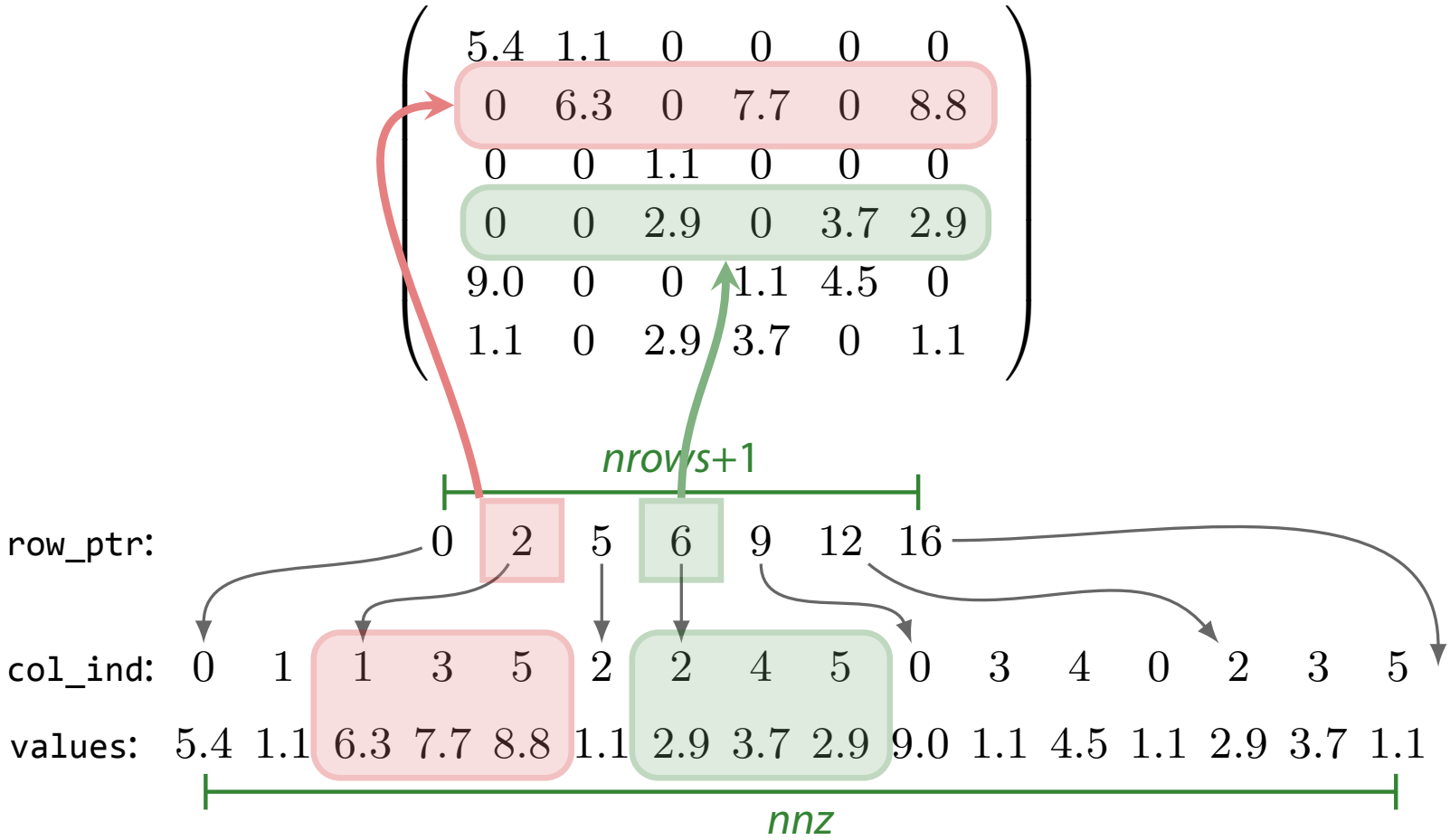
- Sparse storage formats
  - COO: Basic
  - CSR: most common, base-line
  - BCSR: state of the art

CSR (Compressed Sparse Row)

$$\begin{pmatrix} 5.4 & 1.1 & 0 & 0 & 0 & 0 \\ 0 & 6.3 & 0 & 7.7 & 0 & 8.8 \\ 0 & 0 & 1.1 & 0 & 0 & 0 \\ 0 & 0 & 2.9 & 0 & 3.7 & 2.9 \\ 9.0 & 0 & 0 & 1.1 & 4.5 & 0 \\ 1.1 & 0 & 2.9 & 3.7 & 0 & 1.1 \end{pmatrix}$$



CSR (Compressed Sparse Row)



SpMxV (sparse-matrix vector multiplication)

$$y = A \cdot x, \quad A \text{ is sparse}$$

- Important computational kernel
  - Solving PDEs (GMRES, CG) for CFD, economic modeling
  - Graphs (PageRank)
  - abundant amount of research work

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_1 = \sum a_{1i} \cdot x_i \\ y_2 = \sum a_{2i} \cdot x_i \\ y_3 = \sum a_{3i} \cdot x_i \\ y_4 = \sum a_{4i} \cdot x_i \end{pmatrix}$$

SpMxV (sparse-matrix vector multiplication)

$$y = A \cdot x, \quad A \text{ is sparse}$$

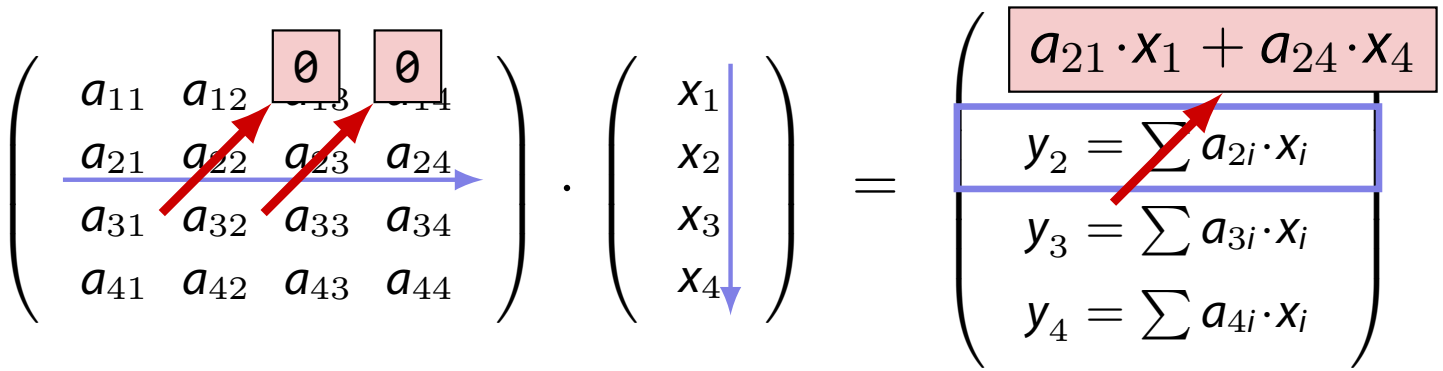
- Important computational kernel
  - Solving PDEs (GMRES, CG) for CFD, economic modeling
  - Graphs (PageRank)
  - abundant amount of research work

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_1 = \sum a_{1i} \cdot x_i \\ y_2 = \sum a_{2i} \cdot x_i \\ y_3 = \sum a_{3i} \cdot x_i \\ y_4 = \sum a_{4i} \cdot x_i \end{pmatrix}$$

SpMxV (sparse-matrix vector multiplication)

$$y = A \cdot x, \quad A \text{ is sparse}$$

- Important computational kernel
  - Solving PDEs (GMRES, CG) for CFD, economic modeling
  - Graphs (PageRank)
  - abundant amount of research work



CSR SpMxV

```
for (i=0; i < N; i++)
   for (j=row_ptr[i]; j < row_ptr[i+1]; j++)
      y[i] += values[j] * x[col_ind[j]];
```

row_ptr:				0	2	5	6	9	12	16						
col_ind:	0	1	1	3	5	2	2	4	5	0	3	4	0	2	3	5

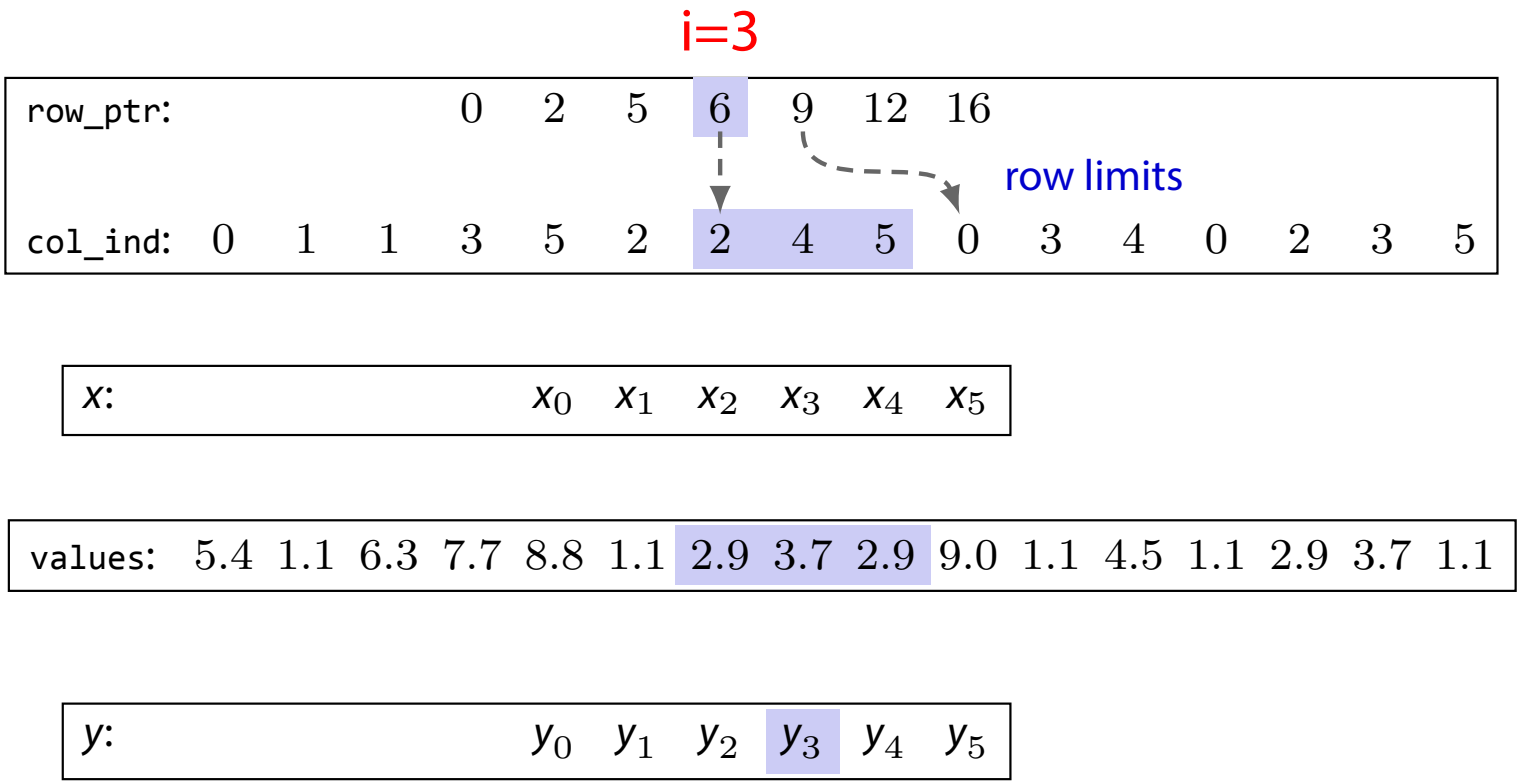
x:				x_0	x_1	x_2	x_3	x_4	x_5

values:	5.4	1.1	6.3	7.7	8.8	1.1	2.9	3.7	2.9	9.0	1.1	4.5	1.1	2.9	3.7	1.1

y:				y_0	y_1	y_2	y_3	y_4	y_5

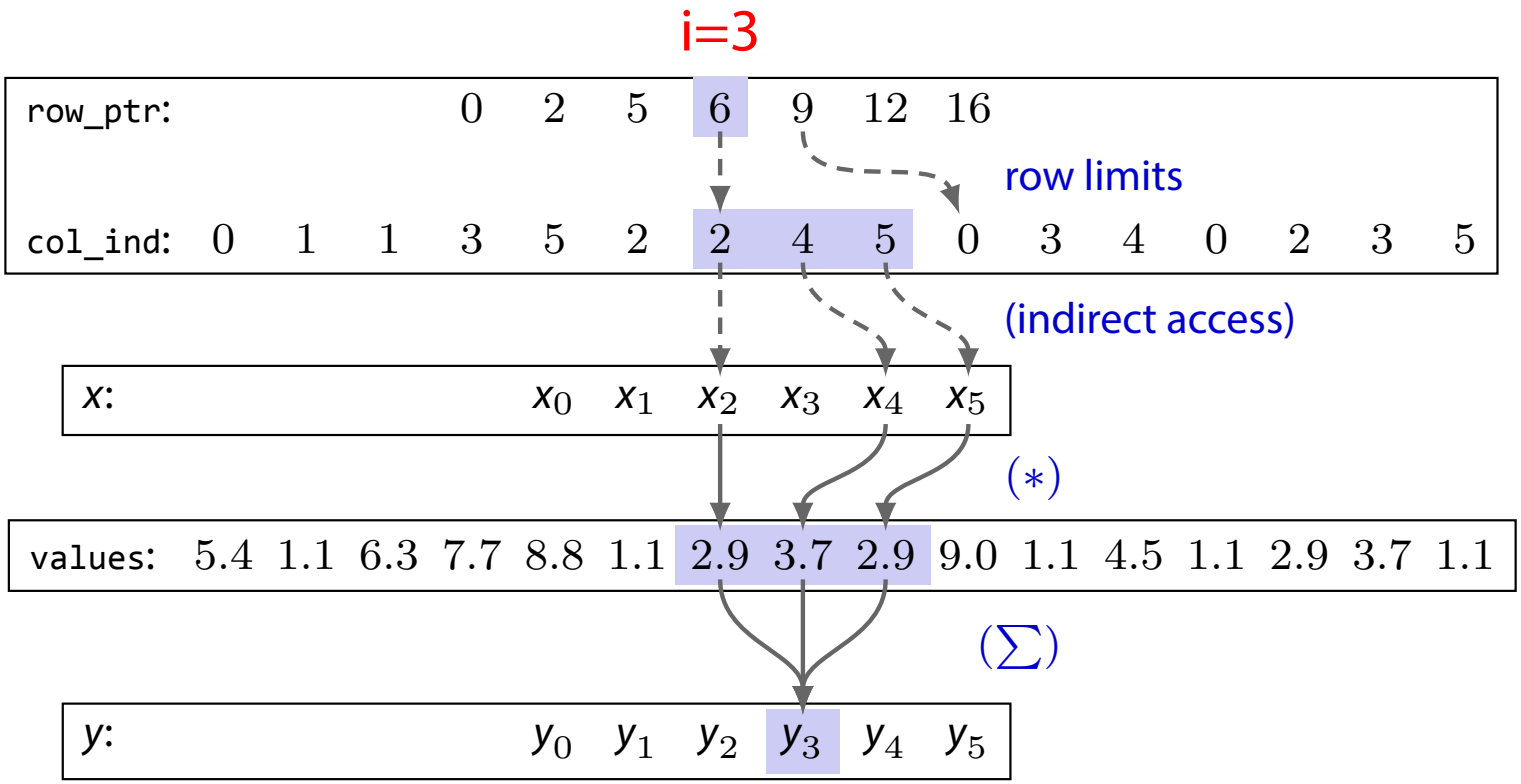
CSR SpMxV

```
for (i=0; i < N; i++)
   for (j=row_ptr[i]; j < row_ptr[i+1]; j++)
      y[i] += values[j] * x[col_ind[j]];
```



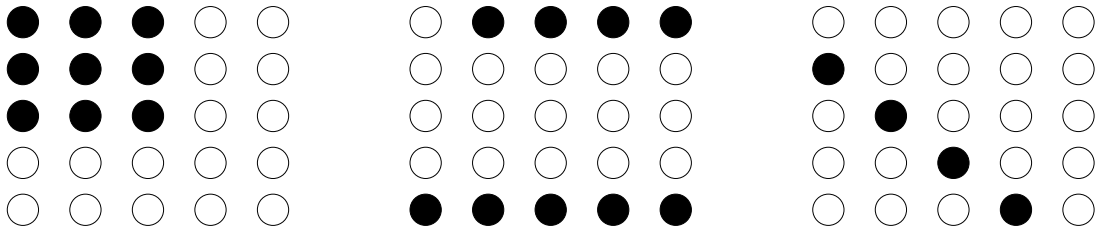
CSR SpMxV

```
for (i=0; i < N; i++)
   for (j=row_ptr[i]; j < row_ptr[i+1]; j++)
      y[i] += values[j] * x[col_ind[j]];
```



Traditional SpMxV optimization methods

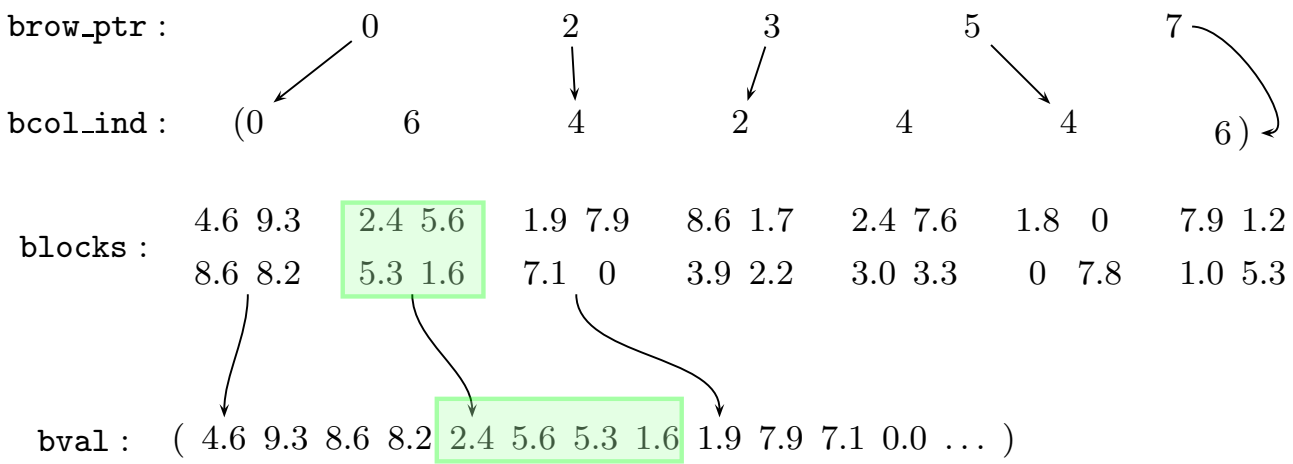
- traditional goal: optimizing computation
- specialized sparse storage formats
  (exploitation of “regularities”)
- examples (regularity $\leftrightarrow$ format):
  - 2D blocks of constant size $\leftrightarrow$ BCSR [Im and Yelick ’01]
  - 1D blocks of variable size $\leftrightarrow$ [Pinar and Heath ’99]
  - Diagonals $\leftrightarrow$ DIAG



Traditional SpMxV optimization: BCSR
[Im and Yelick '01]

- CSR extension: $r \times c$ blocks instead of elements $\Rightarrow$ per-block index information
- optimize computation (register blocking) $\Rightarrow$ specialized SpMxV versions for $r \times c$

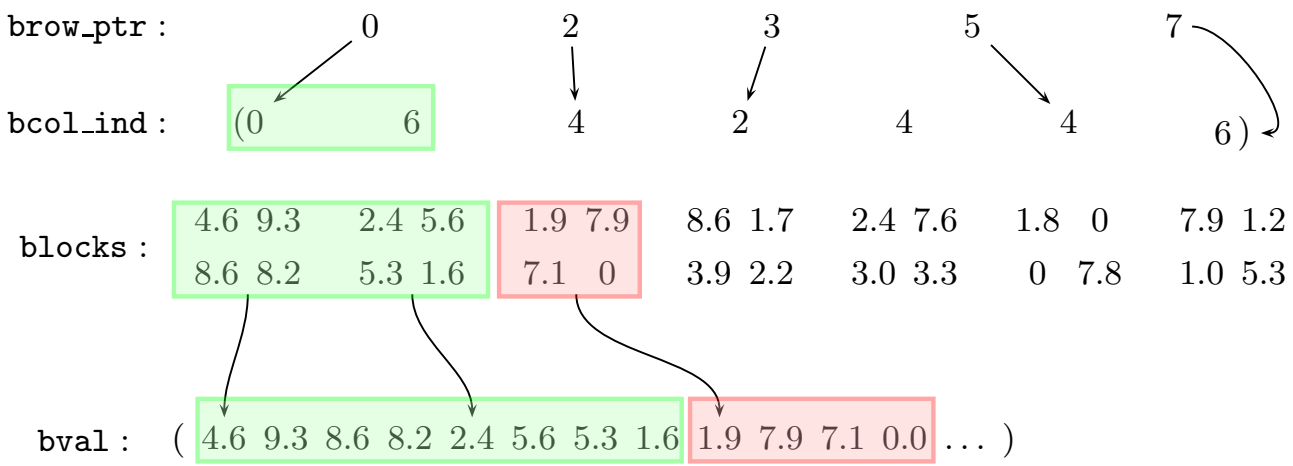
$$A = \left(\begin{array}{cc|cc|cc|cc} 4.6 & 9.3 & 0 & 0 & 0 & 0 & 2.4 & 5.6 \\ 8.6 & 8.2 & 0 & 0 & 0 & 0 & 5.3 & 1.6 \\ \hline 0 & 0 & 0 & 0 & 1.9 & 7.9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7.1 & 0 & 0 & 0 \\ \hline 0 & 0 & 8.6 & 1.7 & 2.4 & 7.6 & 0 & 0 \\ 0 & 0 & 3.9 & 2.2 & 3.0 & 3.3 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1.8 & 0 & 7.9 & 1.2 \\ 0 & 0 & 0 & 0 & 0 & 7.8 & 1.0 & 5.3 \end{array}\right)$$



Traditional SpMxV optimization: BCSR
[Im and Yelick '01]

- CSR extension: $r \times c$ blocks instead of elements $\Rightarrow$ per-block index information
- optimize computation (register blocking) $\Rightarrow$ specialized SpMxV versions for $r \times c$
- padding may be required

$$A = \left(\begin{array}{cccccccc}
4.6 & 9.3 & 0 & 0 & 0 & 0 & 2.4 & 5.6 \\
8.6 & 8.2 & 0 & 0 & 0 & 0 & 5.3 & 1.6 \\
\hline
0 & 0 & 0 & 0 & 1.9 & 7.9 & 0 & 0 \\
0 & 0 & 0 & 0 & 7.1 & 0 & 0 & 0 \\
\hline
0 & 0 & 8.6 & 1.7 & 2.4 & 7.6 & 0 & 0 \\
0 & 0 & 3.9 & 2.2 & 3.0 & 3.3 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1.8 & 0 & 7.9 & 1.2 \\
0 & 0 & 0 & 0 & 0 & 7.8 & 1.0 & 5.3
\end{array}\right)$$



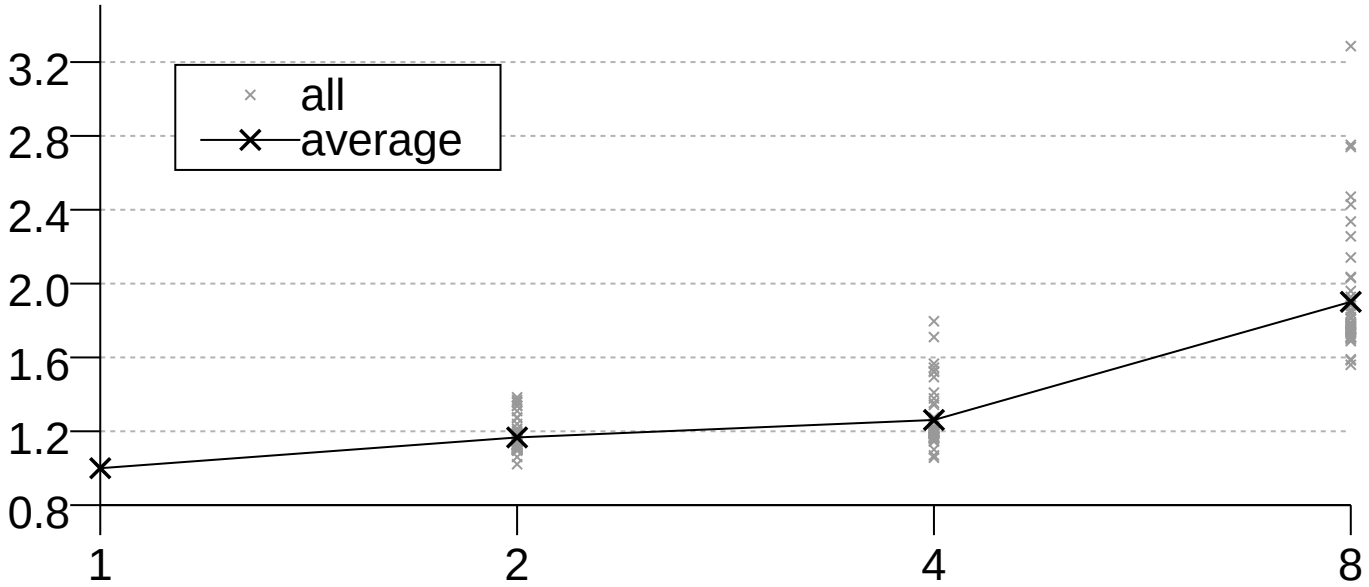
SpMxV performance
(CSR)

- related work $\rightarrow$ several performance issues
- performance evaluation in 100 matrices [Goumas et. al. '09]

[1] for matrices larger than cache

SpMxV performance
(CSR)

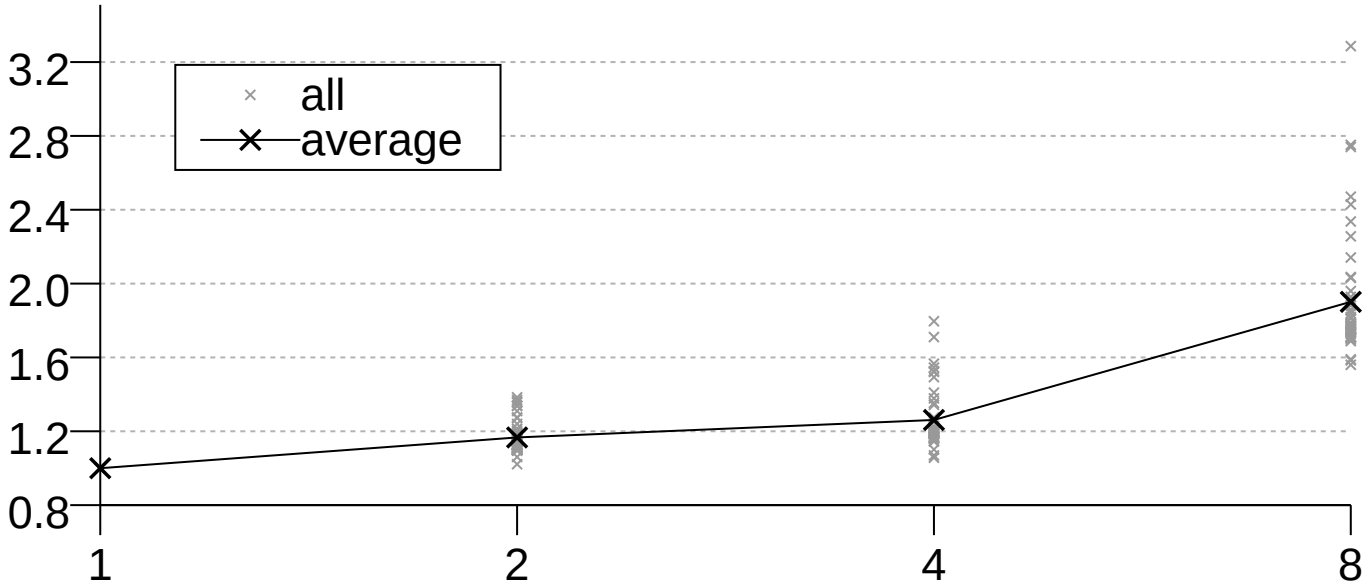
- related work $\rightarrow$ several performance issues
- performance evaluation in 100 matrices [Goumas et. al. '09]
- memory bandwidth is the bottleneck [1]
  (optimization attempts to improve computations are misguided)



[1] for matrices larger than cache

SpMxV performance
(CSR)

- related work $\rightarrow$ several performance issues
- performance evaluation in 100 matrices [Goumas et. al. '09]
- memory bandwidth is the bottleneck [1]
  (optimization attempts to improve computations are misguided)



- **compression** for improving SpMxV performance
  (reduce working set)

[1] for matrices larger than cache

CSR SpMxV working set

($nnz \gg N$)

s_{idx} : column index size

s_{val} : value size

nnz : non-zero values

ws : working set size

$$ws = \overbrace{nnz \cdot s_{idx}}^{\text{index data}} + \overbrace{nnz \cdot s_{val}}^{\text{value data}}$$

$$\left.\begin{aligned} s_{idx} &= 32\ bit \\ s_{val} &= 64\ bit \end{aligned}\right\} \Longrightarrow$$

Outline

Index compression

Initial remarks:

- SpMxV is a memory-bound kernel
- data compression can be a viable approach
- index data seem a good target for compression (include a lot of redundancy)

Index compression

Initial remarks:

- SpMxV is a memory-bound kernel
- data compression can be a viable approach
- index data seem a good target for compression (include a lot of redundancy)

Specialized storage formats

- indirectly may lead to index compression
- typically exploit regularities: e.g., 2D blocks, diagonals, etc.
- e.g., BSCR one column index per block
  - original goal: register blocking
  - **but** may lead to data increase due to padding

Index compression

Initial remarks:

- SpMxV is a memory-bound kernel
- data compression can be a viable approach
- index data seem a good target for compression (include a lot of redundancy)

Specialized storage formats

- indirectly may lead to index compression
- typically exploit regularities: e.g., 2D blocks, diagonals, etc.
- e.g., BSCR one column index per block
  - original goal: register blocking
  - **but** may lead to data increase due to padding

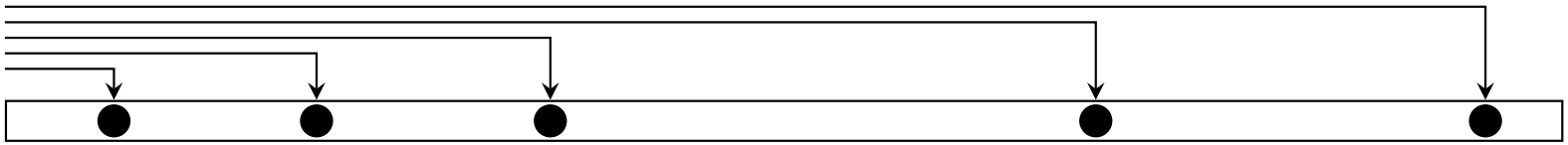
Storage formats that explicitly target index compression

- delta encoding (DCSR [Willcock and Lumsdaine '06], CSR-DU)
- CSX (generalization of CSR-DU)

First step towards index compression: Delta Encoding

applied in each matrix row

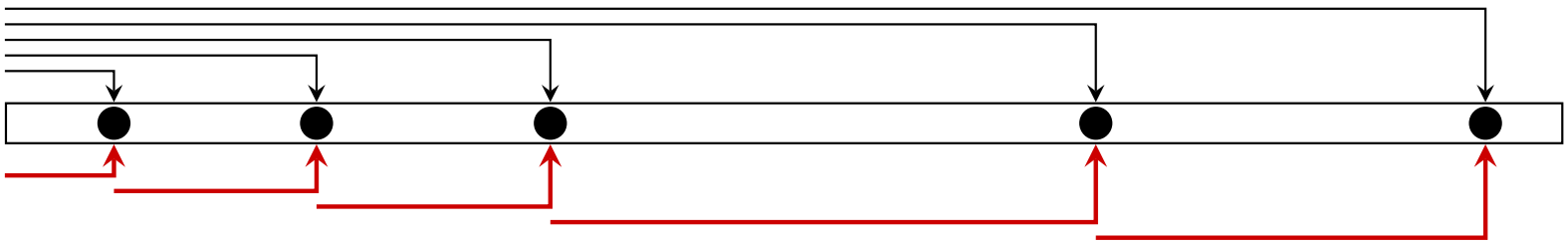
- index data $\rightarrow$ column indices
- Delta encoding for column indices ([Willcock and Lumsdaine '06])



First step towards index compression: Delta Encoding

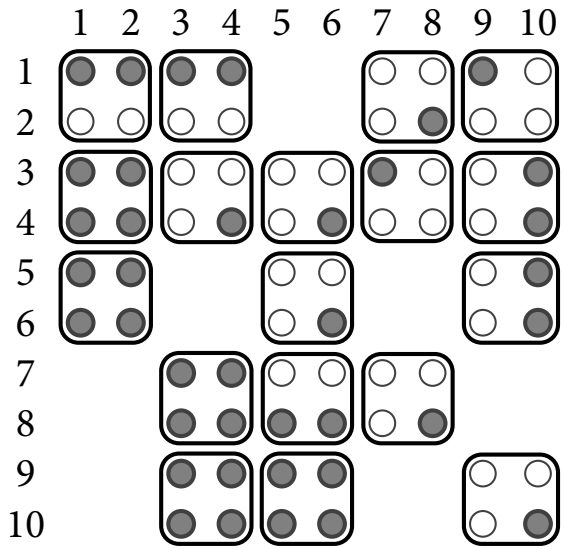
applied in each matrix row

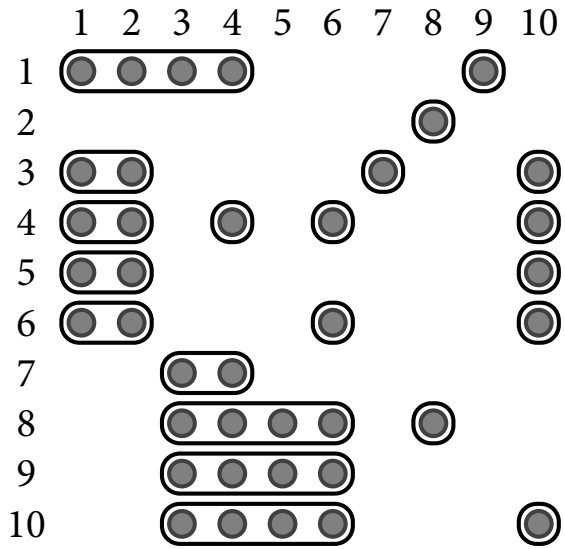
- index data $\rightarrow$ column indices
- Delta encoding for column indices ([Willcock and Lumsdaine '06])
- store delta distance from previous index, not absolute value
- instead of ci_i, store:
  $\delta_i = ci_i - ci_{i-1} \Rightarrow \delta_i \leq ci_i \Rightarrow$ (potentially) less space per index



regularities and sparse storage formats

- BCSR, VBL [Pinar and Heath '99], DIAG





- multiple regularities $\leftrightarrow$ *composite formats* [Agarwal et. al '92]
  multiple sub-matrices — each in different format
  $A \cdot x = (A_0 + A_1) \cdot x = A_0 \cdot x + A_1 \cdot x$

CSX: Compressed Sparse eXtended

In a nutshell

Objectives

- target memory-bandwidth limitations of SpMxV (implied large matrices)
- adapt to matrix structure and architecture
- generate efficient code

Approach

- apply aggressive index compression
- exploit a wide set of matrix “regularities”
- employ code generation to produce efficient code tailored per matrix
- 3 preprocessing phases:
  - detection of regularities
  - matrix encoding
  - code generation
- drastically reduce preprocessing times

CSX substructures
(regularities supported by CSX)

- **Horizontal**

x	x	x	x	x

  (e.g: col. indices: 1,2,3,4,5)

  sequential elements

  $(y, x+i) \rightarrow \quad (y,x) \quad (y,x+1) \quad (y,x+2) \quad \ldots$

CSX substructures
(regularities supported by CSX)

- **Horizontal (delta run-length-encoding — drle)**

x	x	x	x	x

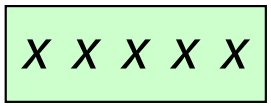
  (e.g: col. indices: 2,4,6,8,10)

  ~~sequential~~ elements with a constant difference δ

  $$(y, x + i \cdot \delta) \rightarrow \quad (y, x) \quad (y, x + \delta) \quad (y, x + 2 \cdot \delta) \ldots$$

CSX substructures
(regularities supported by CSX)

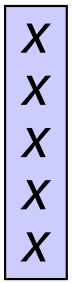
- **Horizontal (delta run-length-encoding — drle)**



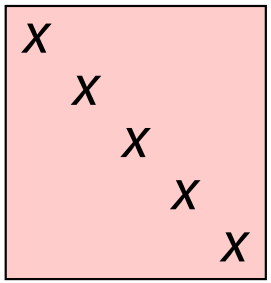
~~sequential~~ elements with a constant difference δ

$$(y, x + i \cdot \delta) \rightarrow \quad (y, x) \quad (y, x + \delta) \quad (y, x + 2 \cdot \delta) \ldots$$

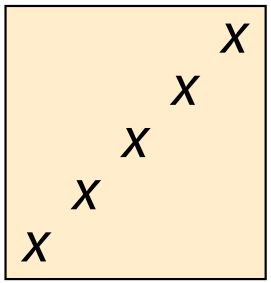
- **Other 1D directions (Vertical, Diagonal, Anti-Diagonal)**



$(y + i \cdot \delta, x)$



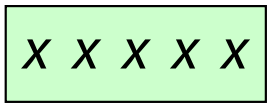
$(y + i \cdot \delta, x + i \cdot \delta)$



$(y - i \cdot \delta, x + i \cdot \delta)$

CSX substructures
(regularities supported by CSX)

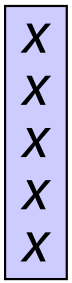
- **Horizontal (delta run-length-encoding — drle)**



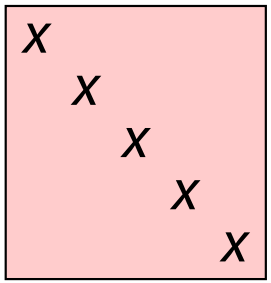
~~sequential~~ elements with a constant difference δ

$$(y, x + i \cdot \delta) \rightarrow \quad (y, x) \quad (y, x + \delta) \quad (y, x + 2 \cdot \delta) \ldots$$

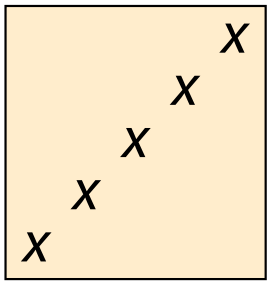
- **Other 1D directions (Vertical, Diagonal, Anti-Diagonal)**



$(y + i \cdot \delta, x)$

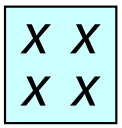


$(y + i \cdot \delta, x + i \cdot \delta)$



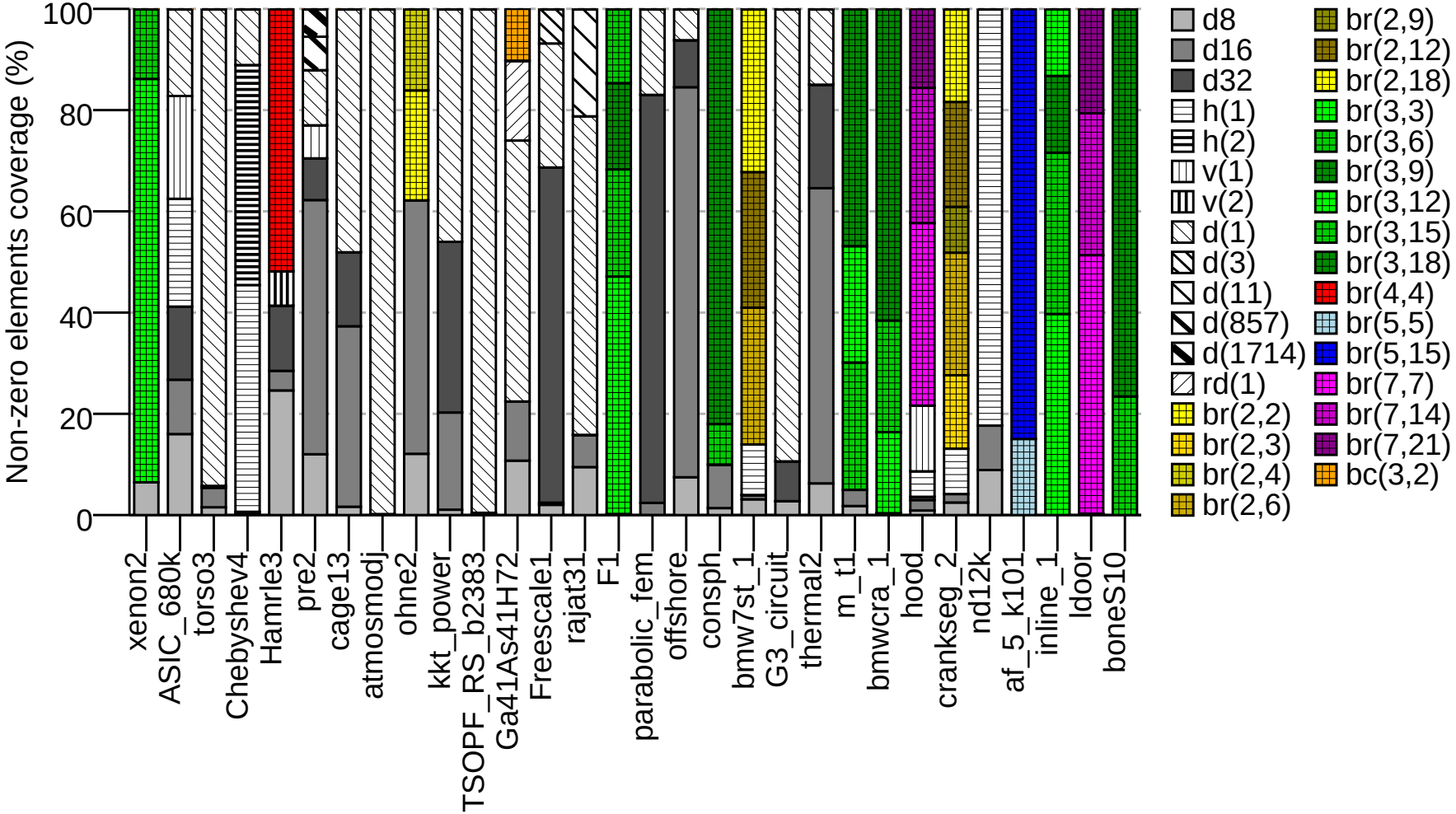
$(y - i \cdot \delta, x + i \cdot \delta)$

- **2D blocks**

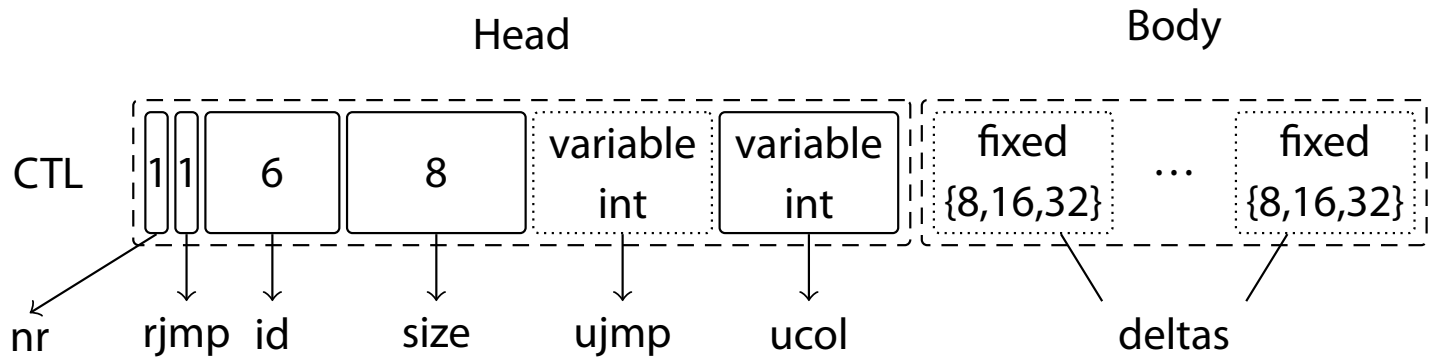


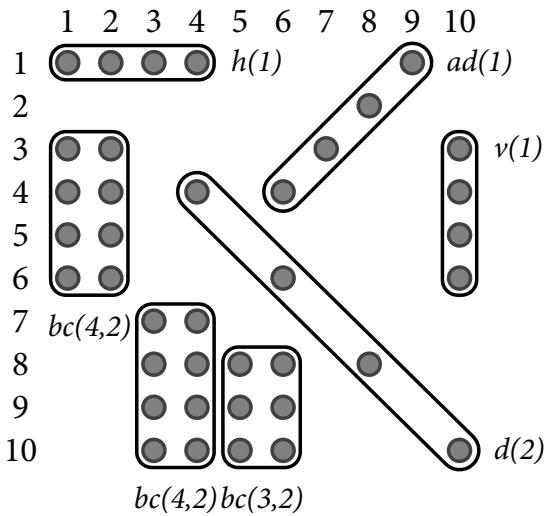
$(x + i) \times (y + j)$ (double nested loop)

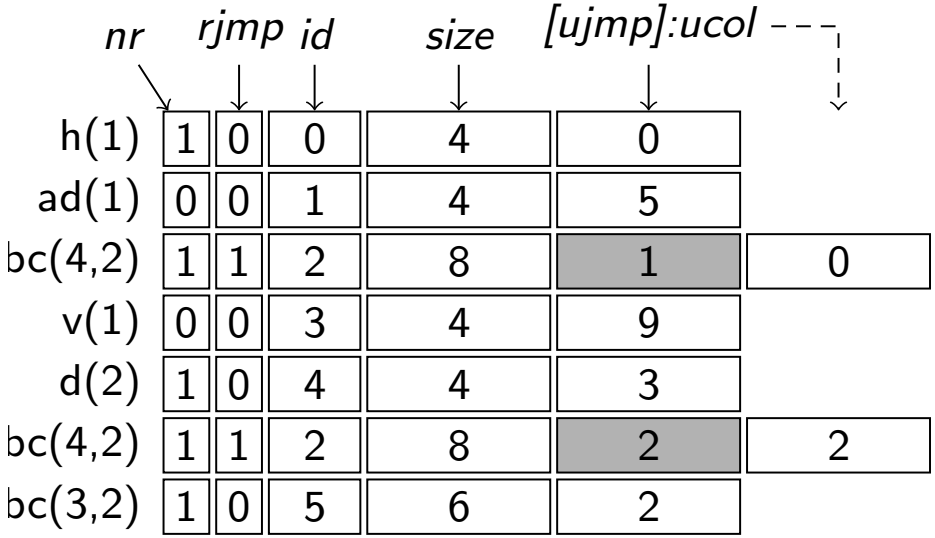
CSX substructures on matrices



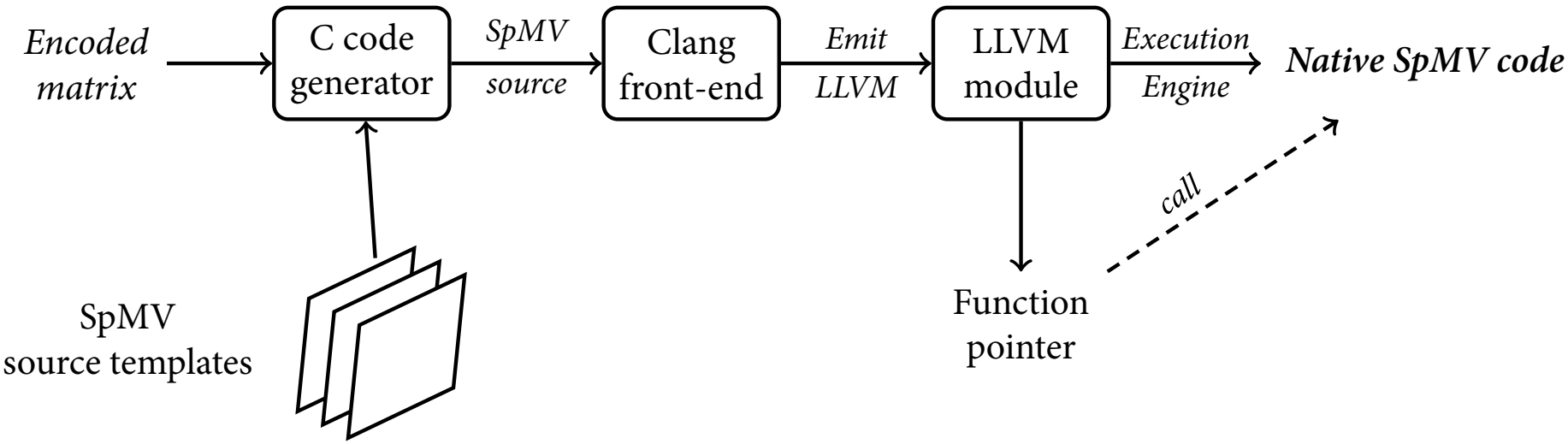
CSX Encoding







CSX Code generation



- top-level SpMxV template
  - big case statement based on substructure
- code for each substructure in the matrix

CSX preprocessing cost

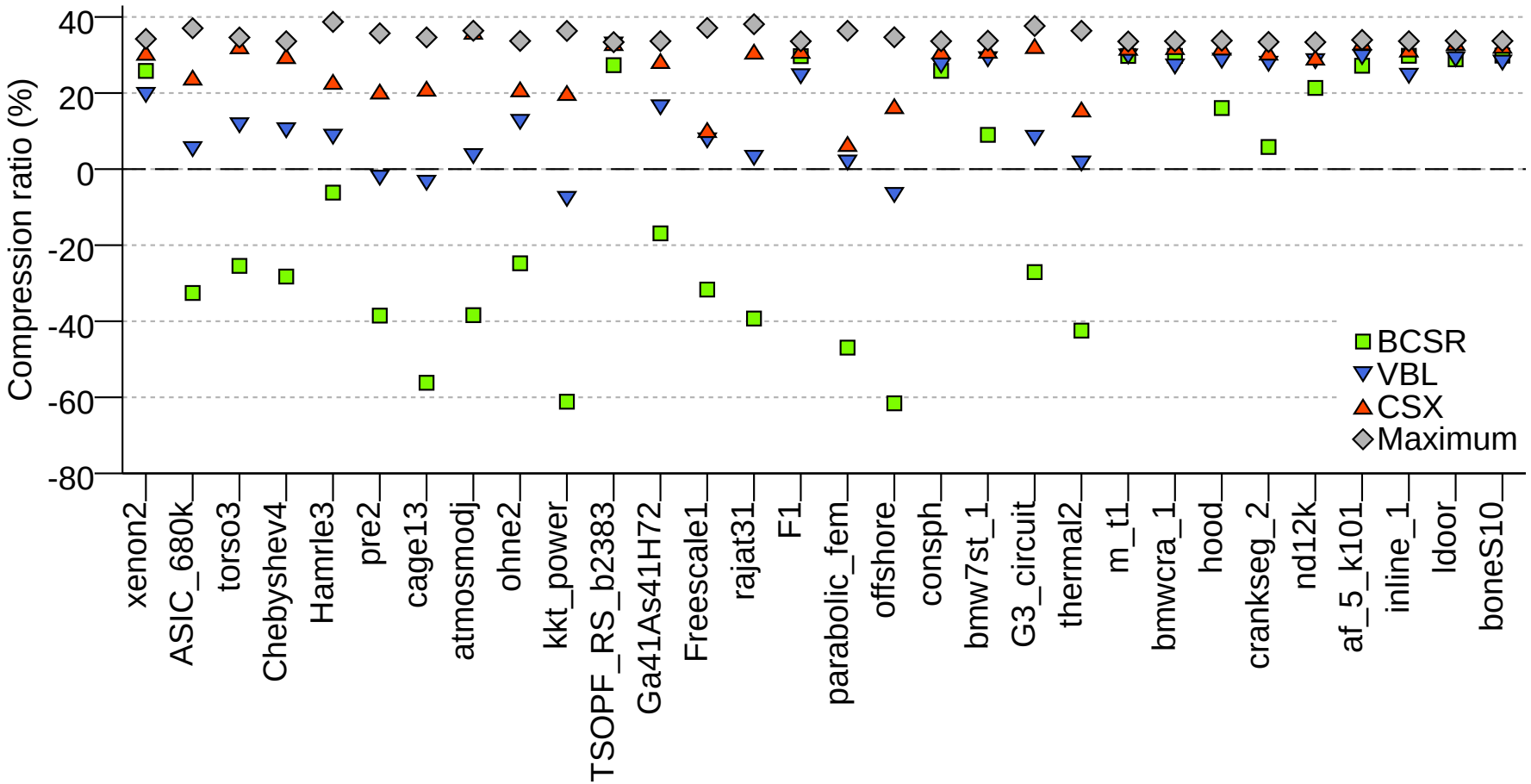
➜ what about preprocessing (compression) cost?

- depends on the application
- frequently, the matrix is used across numerous SpMxV runs
  - ➙ sufficient repetitions → overhead will be amortized
- methods to reduce preprocessing cost:
  - reduce the number of substructures scanned
  - sample the matrix for substructures
  - parallelize preprocessing

Experimental preliminaries

- matrix suite:
  - 30 matrices
  - University of Florida sparse matrix collection [Davis and Hu, 2011]
  - real world applications, large variety of applications
  - including problems without an underlying 2D/3D geometry
  - do not fit into aggregate cache
- compare against:
  - CSR
  - BCSR (we always select the best performing block)
  - VBL (1D variable length blocks)
- double (64-bit) floating point values

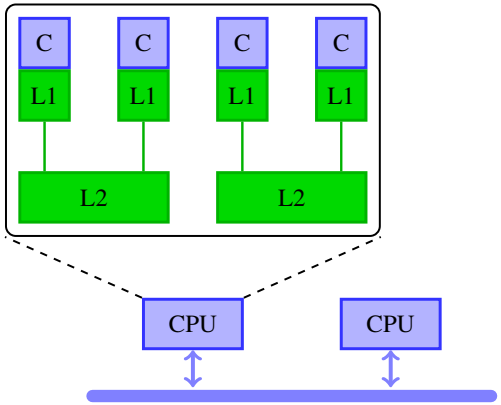
CSX Compression ratio



- *maximum*: only consider values
- CSX always the best option
- CSX never has negative compression

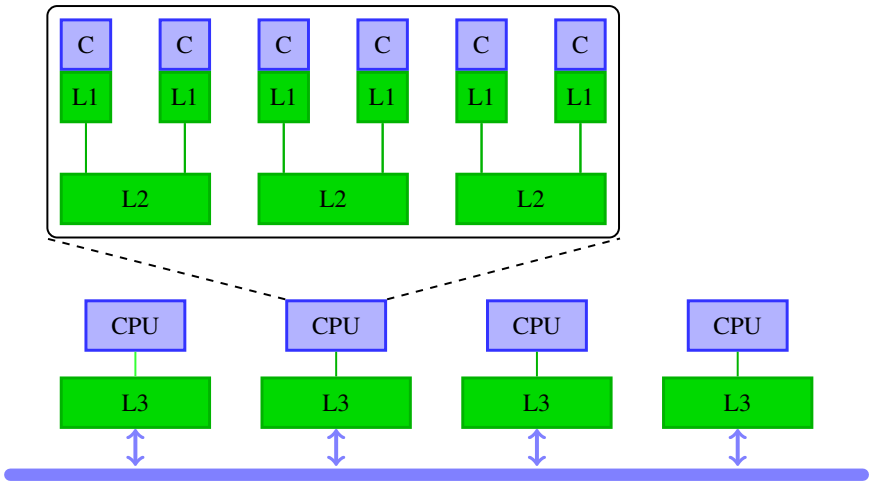
CSX: Evaluation on SMP

Harpertown



- $2 \times 4 = 8$ cores
- L2: 6MiB, per 2 cores

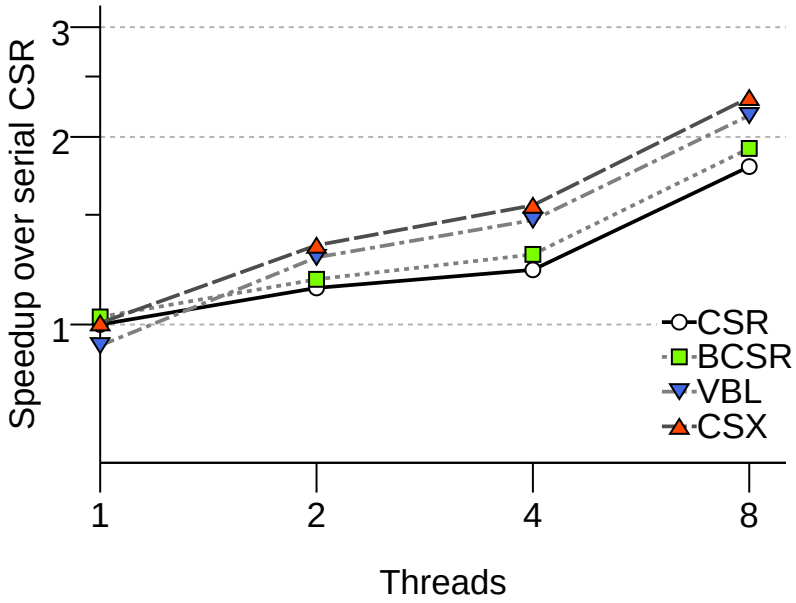
Dunnington



- $4 \times 6 = 24$ cores
- L2: 3MiB, per 2 cores
- L3: 16MiB

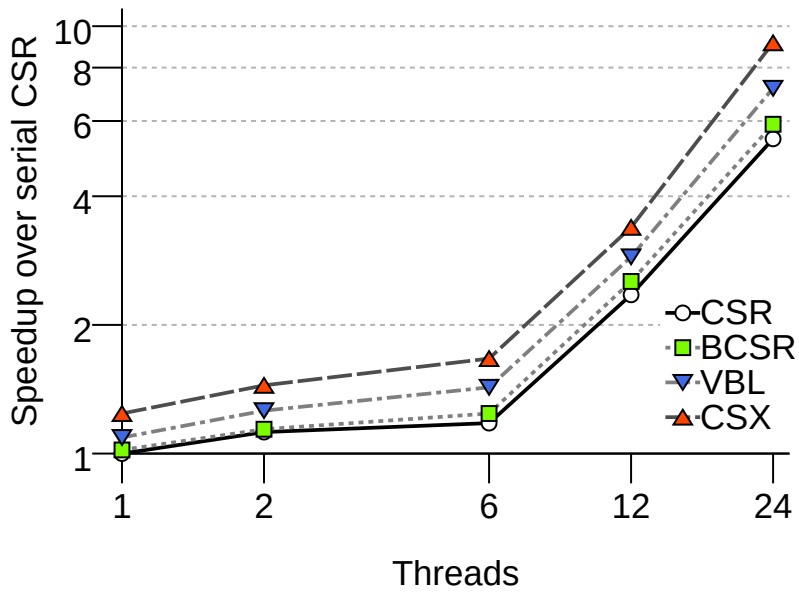
CSX: SMP: Average speedup over serial CSR
(*share-all* core filling policy)

Harpertown



- improvement over MT CSR for 8 threads:
  - CSX: 26.4%
  - VBL: 18.5%
  - BCSR: 4.1%

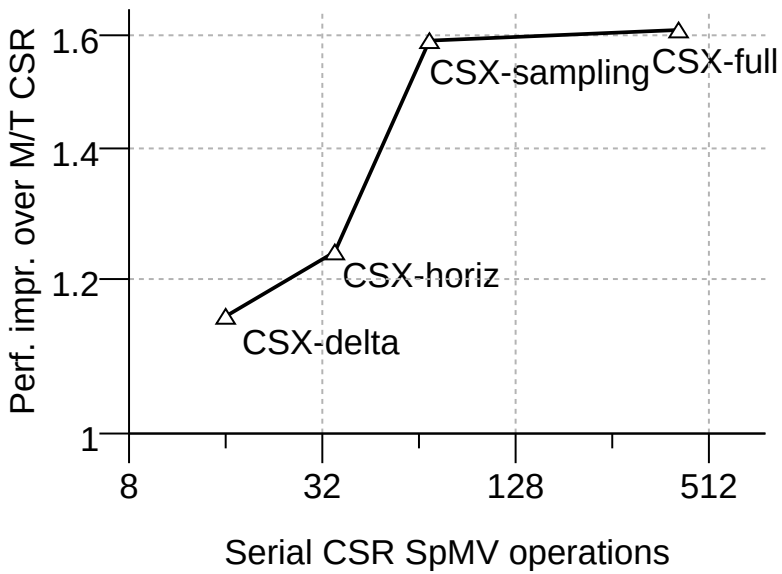
Dunnington



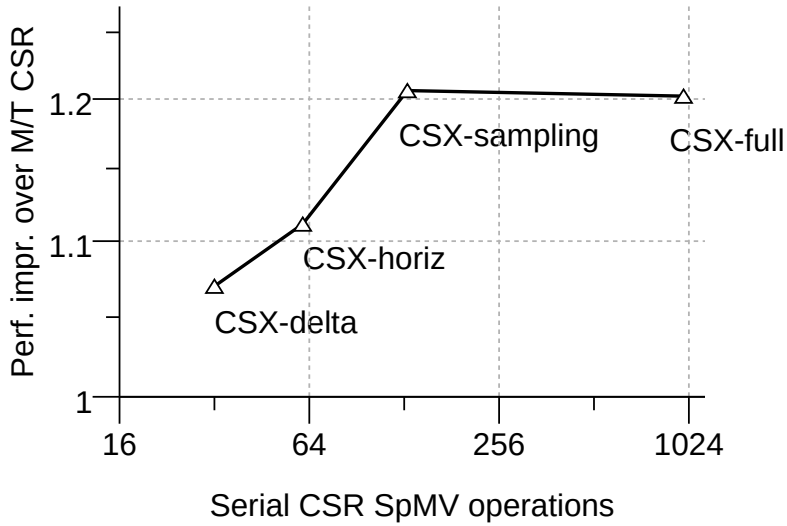
- improvement over MT CSR for 24 threads:
  - CSX: 61%
  - VBL: 28.8%
  - BCSR: 6.3%

CSX: Preprocessing cost

Dunnington



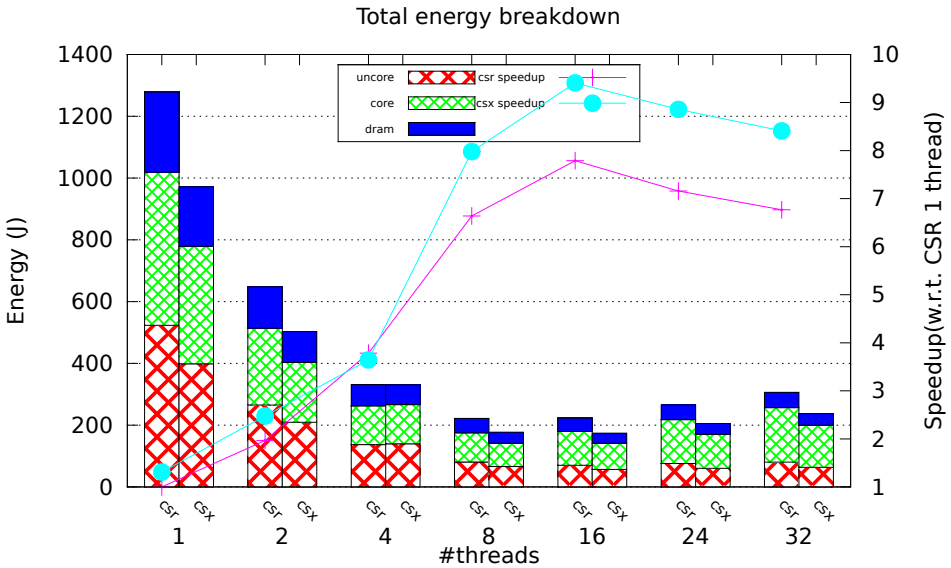
Gainestown (NUMA)



Energy and power measurements

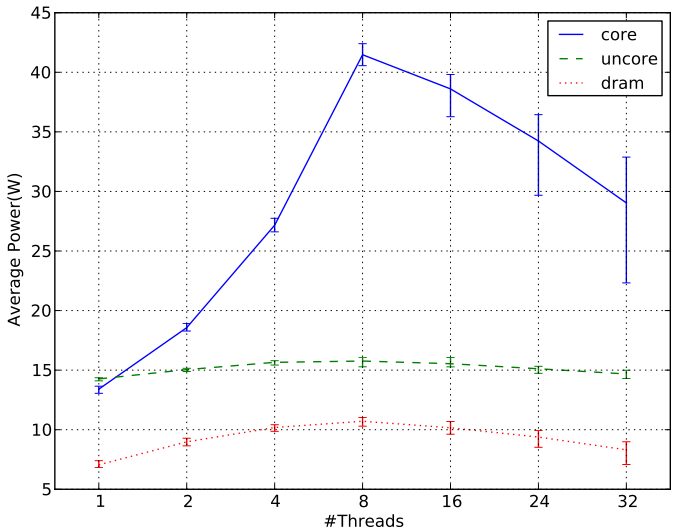
Total energy (idle cores included)

matrix af_5_k101



Power

dense matrix



More info on CSX (papers)

CSX details:

K. Kourtis, V. Karakasis, G. Goumas, and N. Koziris. **"CSX: an extended compression format for spmv on shared memory systems,"** *16th ACM symposium on Principles and practice of parallel programming (PPoPP '11). ACM, New York, NY, USA, 247-256.*

V. Karakasis, T. Gkountouvas, K. Kourtis, G. Goumas, N. Koziris, **"An Extended Compression Format for the Optimization of Sparse Matrix-Vector Multiplication,"** *IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 10, pp. 1930-1940, Oct., 2013.*

CSX for symmetric matrices:

T. Gkountouvas, V. Karakasis, K. Kourtis, G. Goumas, and N. Koziris. **"Improving the performance of the symmetric sparse matrix-vector multiplication in multicore".** *In 27th IEEE International Parallel & Distributed Processing Symposium (IPDPS'13), Boston, MA, USA, 2013.*

CSX integrated with Elmer multiphysics simulation software

V. Karakasis, G. Goumas, K. Nikas, N. Koziris, J. Ruokolainen, and P. Råback. **"Using State-of-the-Art Sparse Matrix Optimizations for Accelerating the Performance of Multiphysics Simulations"**. *In PARA 2012: Workshop on State-of-the-Art in Scientific and Parallel Computing, Helsinki, Finland, 2012. Springer.*

Value compression for SpMxV

K. Kourtis, G. Goumas and N. Koziris, **"Exploiting Compression Opportunities to Improve SpMxV Performance on Shared Memory Systems,"** *ACM Transactions on Architecture and Code Optimization (TACO), Vol 7, No 3, December 2011.*

The energy profile of CSX and CSR

J. C. Meyer, V. Karakasis, J. Cebrián, L. Natvig, D. Siakavaras, and K. Nikas. **"Energy-efficient sparse matrix autotuning with CSX – A trade-off study"**. *In Ninth Workshop on High-Performance, Power-Aware Computing (HPPAC'13), IPDPS'13, Boston, MA, USA, 2013.*

More info on CSX

- download code:

  `http://www.cslab.ece.ntua.gr/csx/`

  `https://github.com/cslab-ntua/csx`

- current status:
  - working on the release of an API and library
  - support tools (disk representation, file format converters)

Outline

Conclusions

Compression can improve SpMxV performance

- CSX applies aggressive index data compression to optimize SpMxV
- supports arbitrary regularities
- tunable preprocessing cost
  - yet, preprocessing can be a concern
- outperforms baseline and state-of-the-art alternatives

Areas of current and future research
and (hopefully) opportunities for collaboration

relevant to CSX, SpMxV and compression

- apply compression to the floating-point values of the matrix (recall: these consume 2/3 of the data!)
- generalize to other applications
- investigate opportunities for hardware support (scalability, space, energy)

Areas of current and future research
and (hopefully) opportunities for collaboration

relevant to CSX, SpMxV and compression

- apply compression to the floating-point values of the matrix (recall: these consume 2/3 of the data!)
- generalize to other applications
- investigate opportunities for hardware support (scalability, space, energy)

contention-aware scheduling

- time and space scheduling of resource-hungry applications (for homogeneous and heterogeneous CMPs)
- performance prediction

Areas of current and future research
and (hopefully) opportunities for collaboration

relevant to CSX, SpMxV and compression

- apply compression to the floating-point values of the matrix (recall: these consume 2/3 of the data!)
- generalize to other applications
- investigate opportunities for hardware support (scalability, space, energy)

contention-aware scheduling

- time and space scheduling of resource-hungry applications (for homogeneous and heterogeneous CMPs)
- performance prediction

energy-aware computing

- power and energy-aware algorithms and techniques
- predict execution behavior based on power consumption snapshots

EOF

Thank you!
Questions?