MAX PLANCK INSTITUTE
FOR DYNAMICS OF COMPLEX
TECHNICAL SYSTEMS
MAGDEBURG

1998 2023

CSC COMPUTATIONAL METHODS IN SYSTEMS AND CONTROL THEORY

Diagonally Addressed Matrix Nicknack

How to speed up the sparse matrix vector (SpMV) product?

Jens Saak **Jonas Schulze**

June 29, 2023

European Trilinos User Group Meeting (EuroTUG), Delft

Motivation

- Dense matrix storage: store *all* matrix entries
  $\rightsquigarrow$ infeasible for many applications

Motivation

- Dense matrix storage: store *all* matrix entries
  $\rightsquigarrow$ infeasible for many applications
- Sparse matrix storage: store only non-zeros and their locations
  $\rightsquigarrow$ requires “bookkeeping”

Motivation

- Dense matrix storage: store *all* matrix entries
  $\rightsquigarrow$ infeasible for many applications
- Sparse matrix storage: store only non-zeros and their locations
  $\rightsquigarrow$ requires “bookkeeping”
- New technique: Diagonally-Addressed (DA) storage

Uni-Precision Crowd	Multi-Precision Crowd
Reduce traffic [Byte] imposed by bookkeeping	Increase performance uplift going from e.g. **`double`** to **`float`** scalars

Motivation

- Dense matrix storage: store *all* matrix entries
  ⇝ infeasible for many applications
- Sparse matrix storage: store only non-zeros and their locations
  ⇝ requires “bookkeeping”
- New technique: Diagonally-Addressed (DA) storage

Uni-Precision Crowd	Multi-Precision Crowd
Reduce traffic [Byte] imposed by bookkeeping	Increase performance uplift going from e.g. **`double`** to **`float`** scalars

- Example: `Janna/Bump_2911` matrix has 3 million columns/rows but only “few” non-zeros
  - Dense storage: 61.7 TiB using **`double`** scalars

Motivation

- Dense matrix storage: store *all* matrix entries
  ⇝ infeasible for many applications
- Sparse matrix storage: store only non-zeros and their locations
  ⇝ requires “bookkeeping”
- New technique: Diagonally-Addressed (DA) storage

Uni-Precision Crowd	Multi-Precision Crowd
Reduce traffic [Byte] imposed by bookkeeping	Increase performance uplift going from e.g. **`double`** to **`float`** scalars

- Example: `Janna/Bump_2911` matrix has 3 million columns/rows but only “few” non-zeros
  - Dense storage: 61.7 TiB using **`double`** scalars
  - CSR storage: 1.44 GiB using **`int32_t`** indices

Motivation

- Dense matrix storage: store *all* matrix entries
  ⇝ infeasible for many applications
- Sparse matrix storage: store only non-zeros and their locations
  ⇝ requires “bookkeeping”
- New technique: Diagonally-Addressed (DA) storage

Uni-Precision Crowd	Multi-Precision Crowd
Reduce traffic [Byte] imposed by bookkeeping	Increase performance uplift going from e.g. **`double`** to **`float`** scalars

- Example: `Janna/Bump_2911` matrix has 3 million columns/rows but only “few” non-zeros
  - Dense storage: 61.7 TiB using **`double`** scalars
  - CSR storage: 1.44 GiB using **`int32_t`** indices
  - DA-CSR storage: 1.20 GiB (−16.5 %) using **`int16_t`** column indices

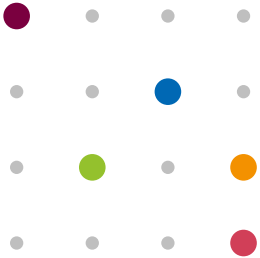
Outline

Outline

Diagonally-Addressed Storage

Example: from CSR to DA-CSR

- Dense storage (row-major):



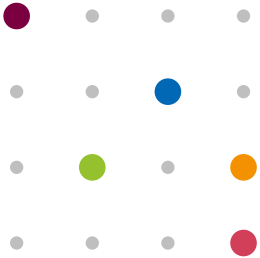
Diagonally-Addressed Storage

Example: from CSR to DA-CSR

- Dense storage (row-major):

[●, ·, ·, ·, ·, ·, ●, ·, ·, ●, ·, ●, ·, ·, ·, ●]

- Sparse storage: which and where are the non-zeros?

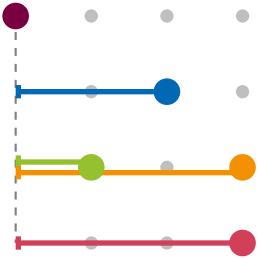


Diagonally-Addressed Storage

Example: from CSR to DA-CSR

- Dense storage (row-major):

- Sparse storage: which and where are the non-zeros?



- Example: Compressed Sparse Row (CSR) format

rows start = “at |”

column indices = [| | |]

values = [| | |]

Diagonally-Addressed Storage

Example: from CSR to DA-CSR

- Dense storage (row-major):

[●, ·, ·, ·, ·, ·, ●, ·, ·, ●, ·, ●, ·, ·, ·, ●]

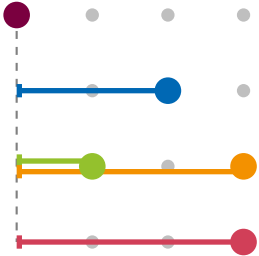
- Sparse storage: which and where are the non-zeros?

- Example: Compressed Sparse Row (CSR) format

rows start = at index [0, 1, 2, 4]

column indices = [▮ | ▬ | ▬ ▬ | ▬]

values = [● | ● | ● ● | ●]



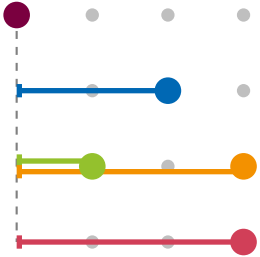
Diagonally-Addressed Storage

Example: from CSR to DA-CSR

- Dense storage (row-major):

[●, ∘, ∘, ∘, ∘, ∘, ●, ∘, ∘, ●, ∘, ●, ∘, ∘, ∘, ●]

- Sparse storage: which and where are the non-zeros?



- Example: Compressed Sparse Row (CSR) format

rows start = [0, 1, 2, 4, nnz]

column indices = [▮, ▬, ▬, ▬, ▬]

values = [●, ●, ●, ●, ●]

nnz = number of non-zeros = 5

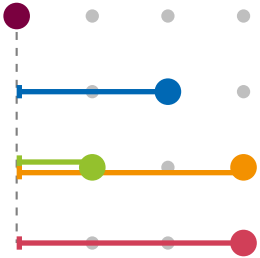
Diagonally-Addressed Storage

Example: from CSR to DA-CSR

- Dense storage (row-major):

  [●, ·, ·, ·, ·, ·, ●, ·, ·, ●, ·, ●, ·, ·, ·, ●]

- Sparse storage: which and where are the non-zeros?



- Example: Compressed Sparse Row (CSR) format

  rows start = [0, 1, 2, 4, nnz]

  column indices = [●, ▬, ▬, ▬, ▬] $\subseteq$ [0, ncols)

  values = [●, ●, ●, ●, ●]

  nnz = number of non-zeros = 5

- Idea: exploit (typically) low matrix bandwidth w, i.e. $w \ll \text{ncols}$, to use smaller data type for column indices ($w = 1$ on the right)

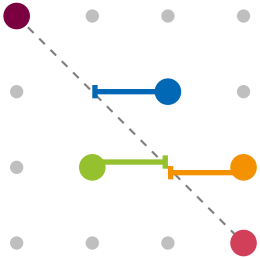
Diagonally-Addressed Storage

Example: from CSR to DA-CSR

- Dense storage (row-major):

  [●, ·, ·, ·, ·, ·, ●, ·, ·, ●, ·, ●, ·, ·, ·, ●]

- Sparse storage: which and where are the non-zeros?



- Example: Diagonally-Addressed CSR (DA-CSR) format

  rows start = [0, 1, 2, 4, nnz]

  column indices = [▮, ▬, ▬▮, ▬, ▮] $\subseteq [-w, w]$

  values = [●, ●, ●, ●, ●]

  nnz = number of non-zeros = 5

- Idea: exploit (typically) low matrix bandwidth w, i.e. $w \ll \text{ncols}$, to use smaller data type for column indices ($w = 1$ on the right)

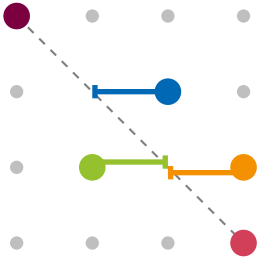
Diagonally-Addressed Storage

Example: from CSR to DA-CSR

- Dense storage (row-major):

  [◆, •, •, •, •, •, ◆, •, •, ◆, •, ◆, •, •, •, ◆]

- Sparse storage: which and where are the non-zeros?



- Example: Diagonally-Addressed CSR (DA-CSR) format

  rows start = [0, 1, 2, 4, nnz]

  column indices = [▮, ▬, ▬▮, ▬, ▮] $\subseteq [-w, w]$

  values = [◆, ◆, ◆, ◆, ◆]

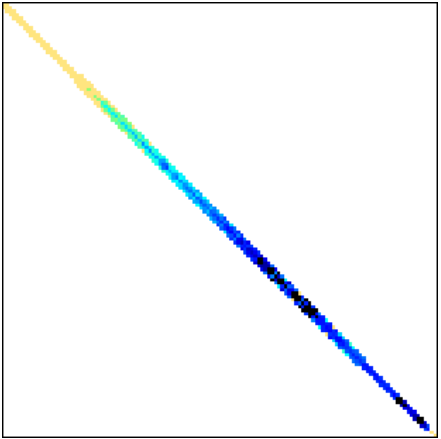
  nnz = number of non-zeros = 5

- Idea: exploit (typically) low matrix bandwidth w, i.e. $w \ll \text{ncols}$, to use smaller data type for column indices ($w = 1$ on the right)

Diagonally-Addressed Storage

Example: `Janna/Bump_2911`

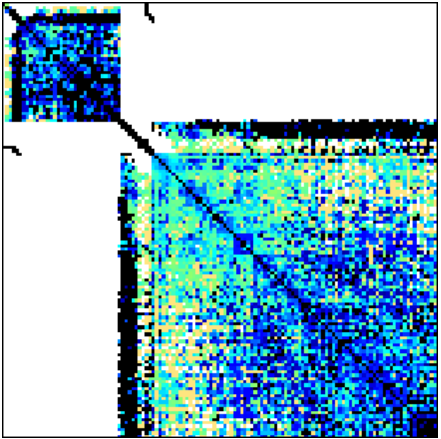
- dim $= 2\,911\,419 \gg 2^{15} = 32\,768$
- nnz $= 127\,729\,899$
- Matrix bandwidth $w = 31\,343 < 2^{15}$
- Assuming (signed) 16 bit column indices...
  - DA-CSR can address the whole matrix
  - CSR can only address left ~12 pixels



Diagonally-Addressed Storage

Example: `GHS_psdef/ldoor`

- dim $= 952\,203 \gg 2^{15} = 32\,768$
- nnz $= 46\,522\,475$
- Matrix bandwidth $w = 686\,979 \gg 2^{15}$



Diagonally-Addressed Storage

Example: `GHS_psdef/ldoor`

- dim $= 952\,203 \gg 2^{15} = 32\,768$
- nnz $= 46\,522\,475$
- Matrix bandwidth $w = 686\,979 \gg 2^{15}$
- After Reverse CutHill-McKee (RCM) permutation: $w = 9120 < 2^{15}$

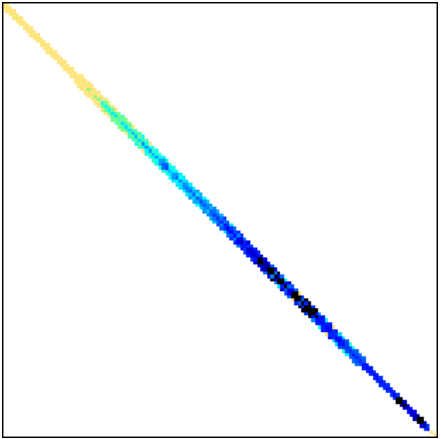


illustration symbolic

Diagonally-Addressed Storage

Relevance: not only for edge cases

The SuiteSparse Matrix Collection contains 2893 matrices in total.

- 1367 matrices fit into CSR(**`int32_t`**, **`int32_t`**, **`double`**)
  that are square, and have full structural rank
- 993 matrices (72.6 %) of which fit into CSR(**`int32_t`**, **`int16_t`**, **`double`**)
  i.e. have dimension less than $2^{15} = 32\,768$
- 1302 matrices (95.2 %) of which fit into DA-CSR(**`int32_t`**, **`int16_t`**, **`double`**)
  potentially after applying Reverse CutHill-McKee (RCM) permutation

Diagonally-Addressed Storage

Relevance: not only for edge cases

The SuiteSparse Matrix Collection contains 2893 matrices in total.

- 1367 matrices fit into CSR(`int32_t`)
  that are square, and have full structural rank
- 993 matrices (72.6 %) of which fit into CSR(`int16_t`)
  i.e. have dimension less than $2^{15} = 32\,768$
- 1302 matrices (95.2 %) of which fit into DA-CSR(`int16_t`)
  potentially after applying Reverse CutHill-McKee (RCM) permutation

Outline

Efficient CSR Baseline

Introducing Relevant Concepts

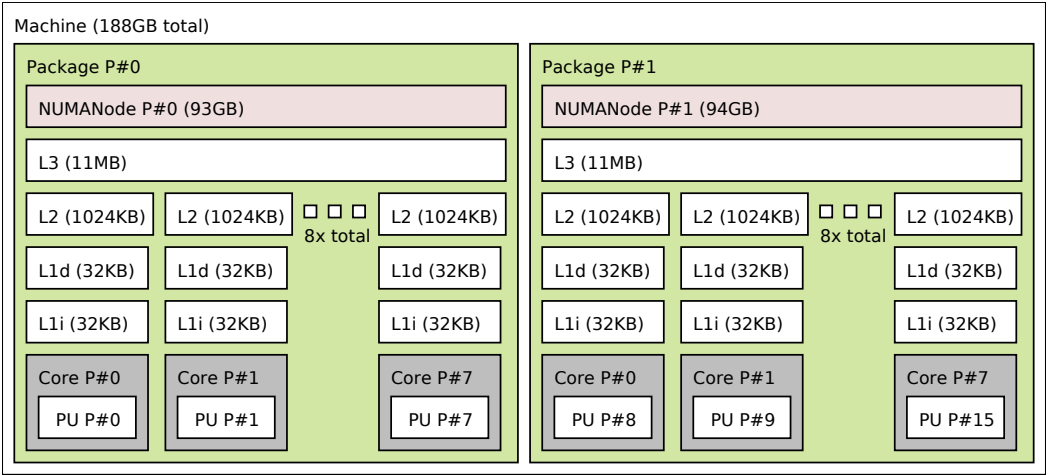
- Sparse Matrix Vector (SpMV) product: $y \leftarrow \alpha A x + \beta y$

```
accumulator = 0;
for (i = rows_start[row]; i < rows_start[row+1]; ++i) {
  accumulator += values[i] * x[column_indices[i]];
}
y[row] = α * accumulator + β * y[row];
```

- Performance [FLOP/s]: $$\frac{2\text{nnz} + 2\text{nrows}}{t}$$
- Traffic [Byte]: $$\text{sizeof}(A) + \text{sizeof}(x) + \text{sizeof}(y)$$
- Throughput [Byte/s]: $$\frac{\text{Traffic}}{t}$$

Efficient CSR Baseline

Detour: Hardware Topology (2× Intel Skylake Xeon Silver 4110)



last-level cache (LLC): L3 in this case

on-chip traffic: traffic $\leq$ sizeof(LLC)

off-chip traffic: traffic $>$ sizeof(LLC)

Efficient CSR Baseline

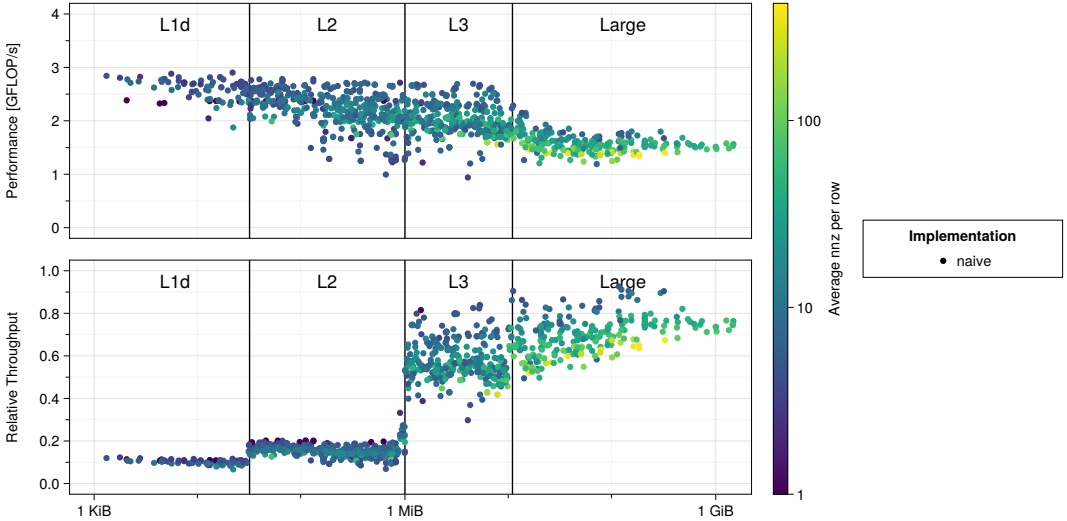
Benchmark Environment

- Hardware: ▸ Mechthild standard node
  Intel Xeon Skylake Silver 4110 (launched 2017)
  192 GB DDR4 ECC memory
- Compiler: GCC 10.3.0
  Build flags: `-O3 -DNDEBUG -mavx2 -mfma`
- CPU pinning: `taskset -c 0-$(($OMP_NUM_THREADS - 1)) ...`
- Runtime estimator: minimum of 11 epochs of 100 ms each ▸ nanobench
- Comparison against Intel Math Kernel Library 2021.1 ▸ MKL

Efficient CSR Baseline

Naive Implementation

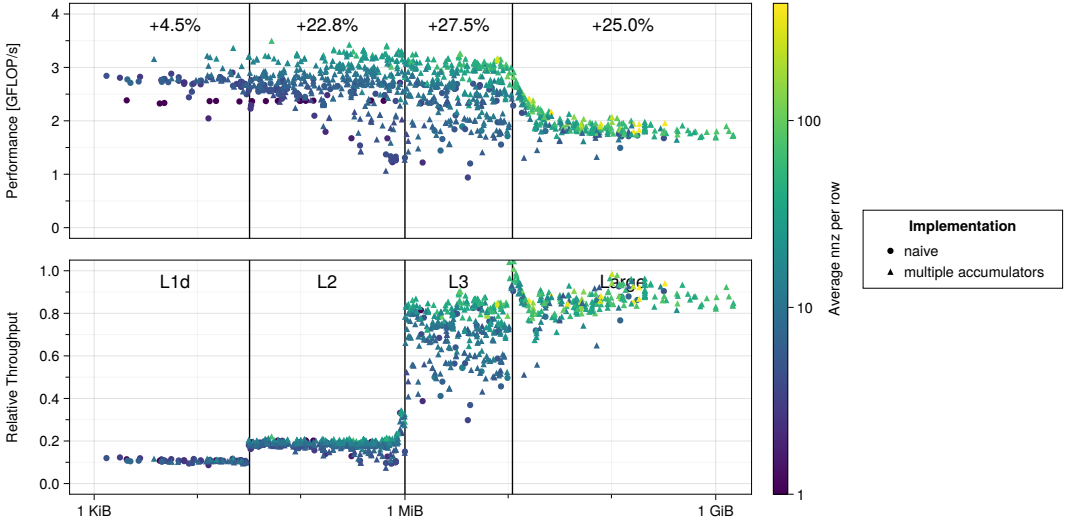
(Single-Threaded)



Efficient CSR Baseline

+ multiple accumulators

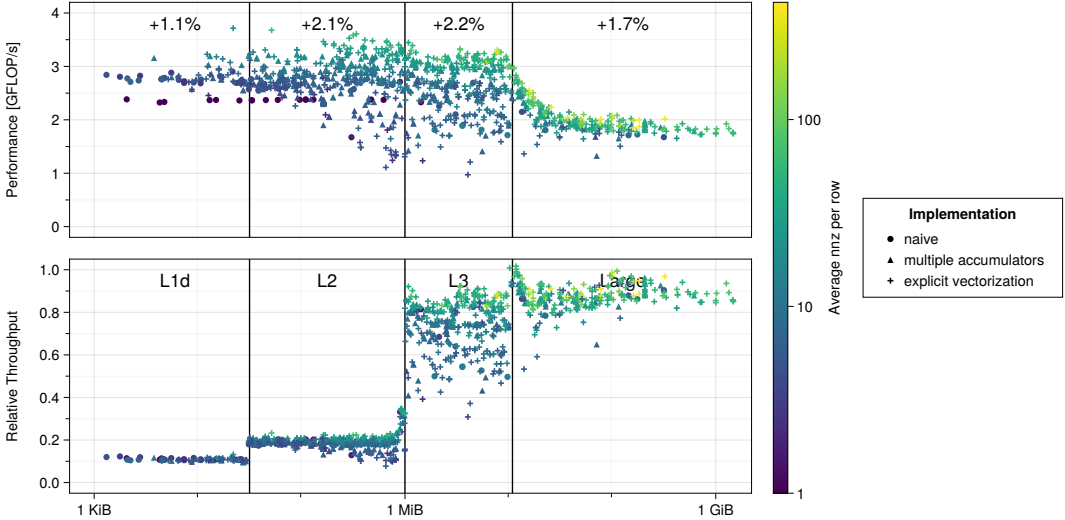
(Single-Threaded)



Efficient CSR Baseline

+ explicit vectorization

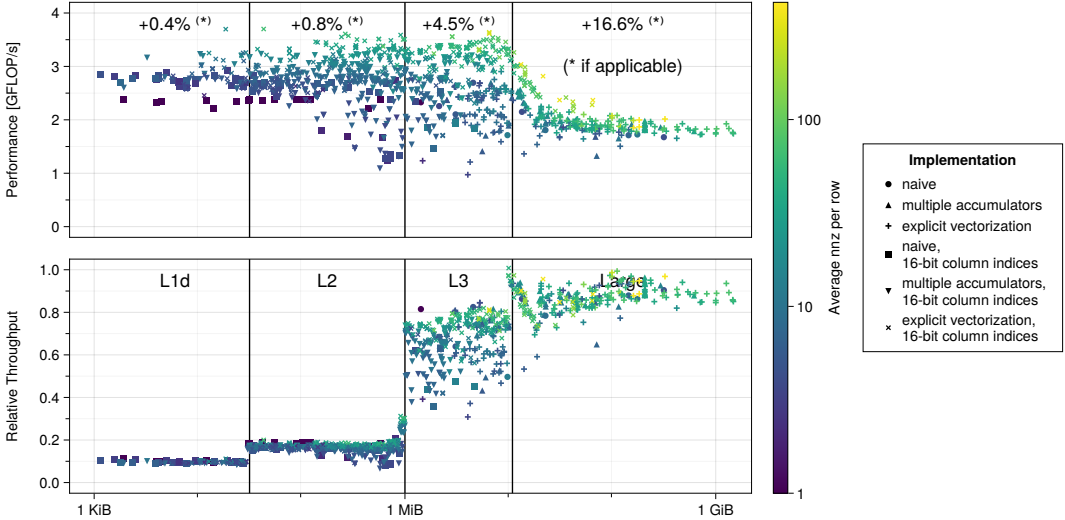
(Single-Threaded)



Efficient CSR Baseline

+ 16-bit column indices

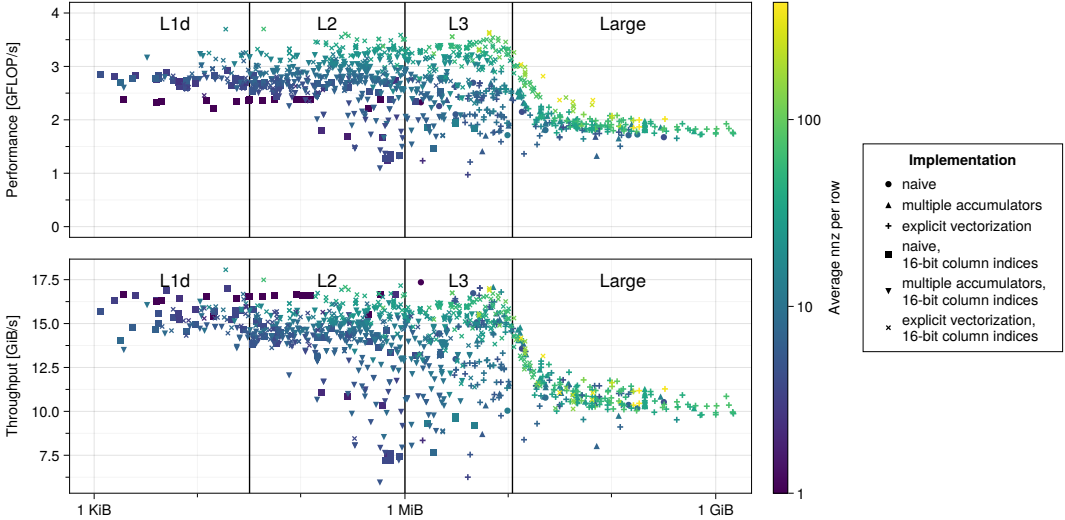
(Single-Threaded)



Efficient CSR Baseline

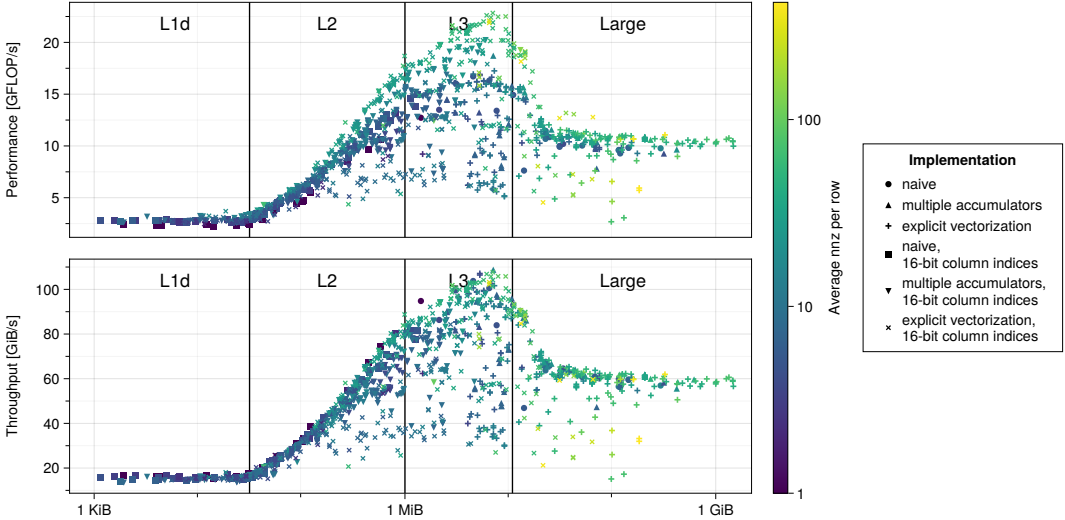
CSR(`int16_t`/`int32_t`)

(Single-Threaded)



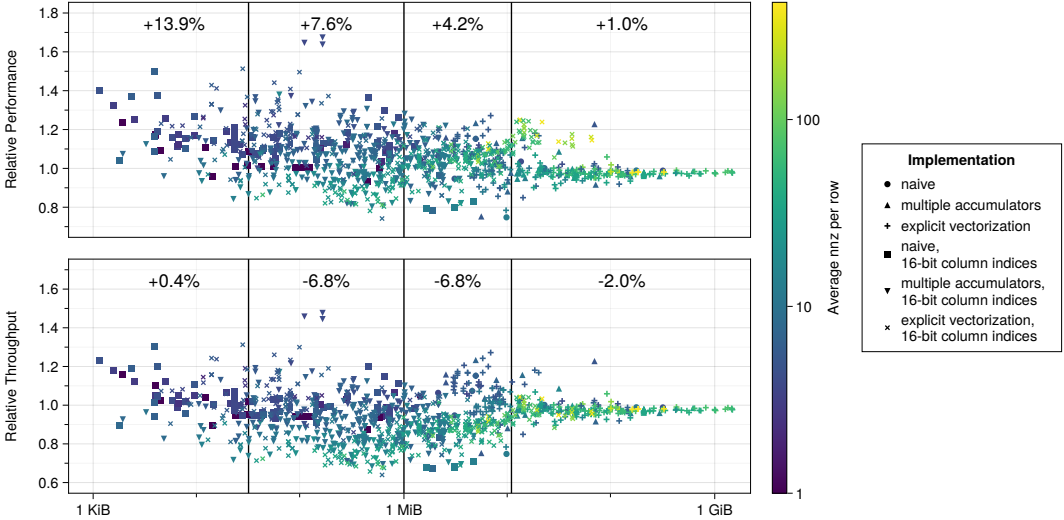
Efficient CSR Baseline

CSR(`int16_t`/`int32_t`) (Multi-Threaded)



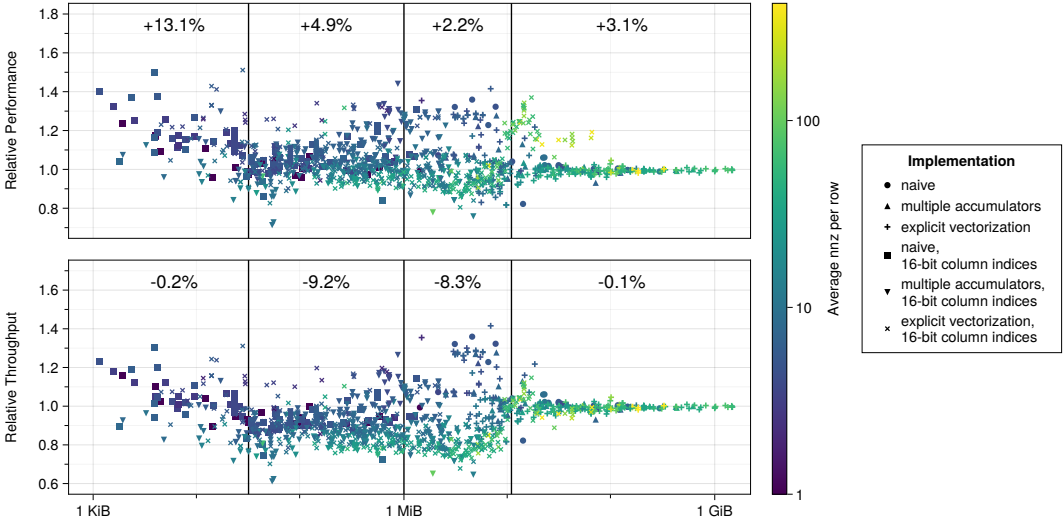
Efficient CSR Baseline

CSR(`int16_t`/`int32_t`) vs. Intel MKL CSR(`int32_t`) (Single-Threaded)



Efficient CSR Baseline

CSR(`int16_t`/`int32_t`) vs. Intel MKL CSR(`int32_t`) (Multi-Threaded)

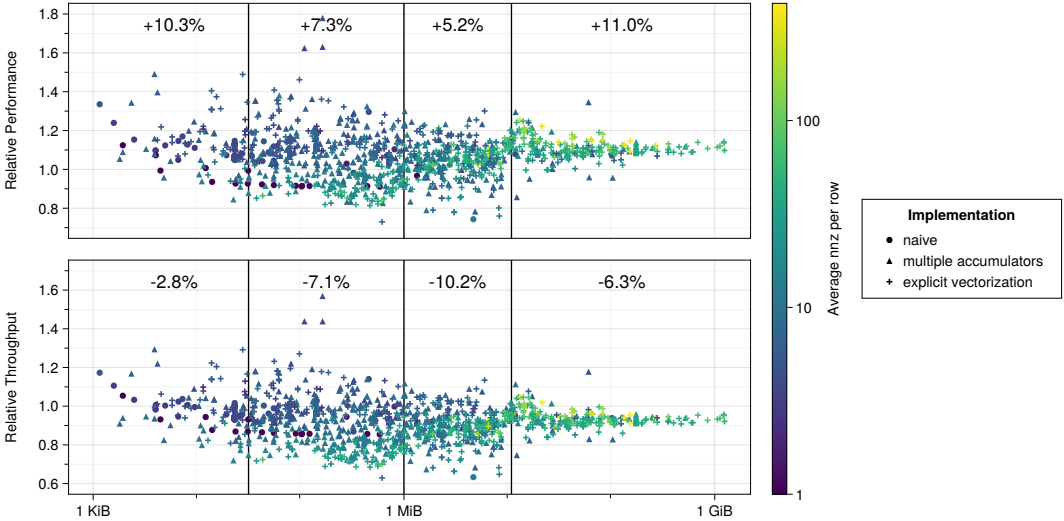


Outline

Comparison With DA-CSR

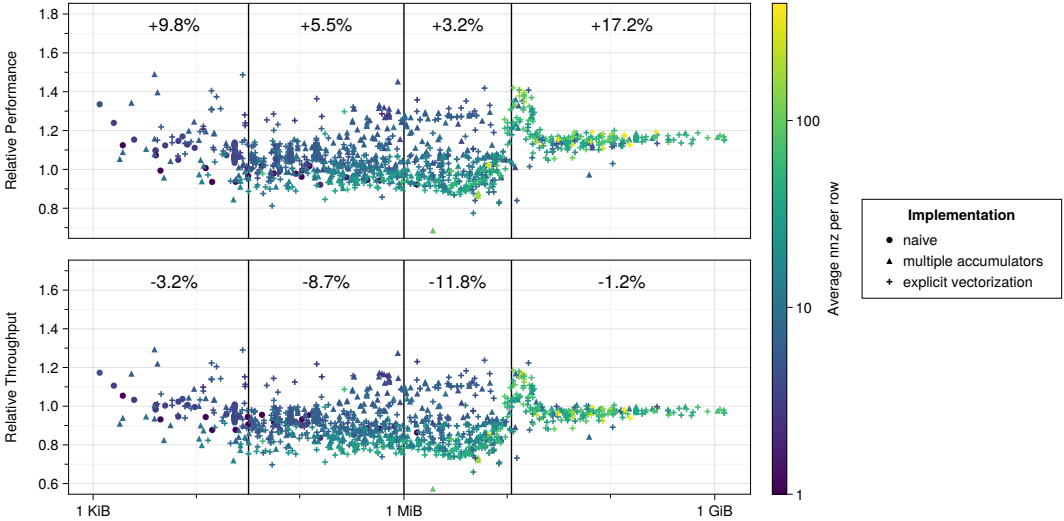
DA-CSR(`int16_t`) vs. Intel MKL CSR(`int32_t`)

(Single-Threaded)



Comparison With DA-CSR

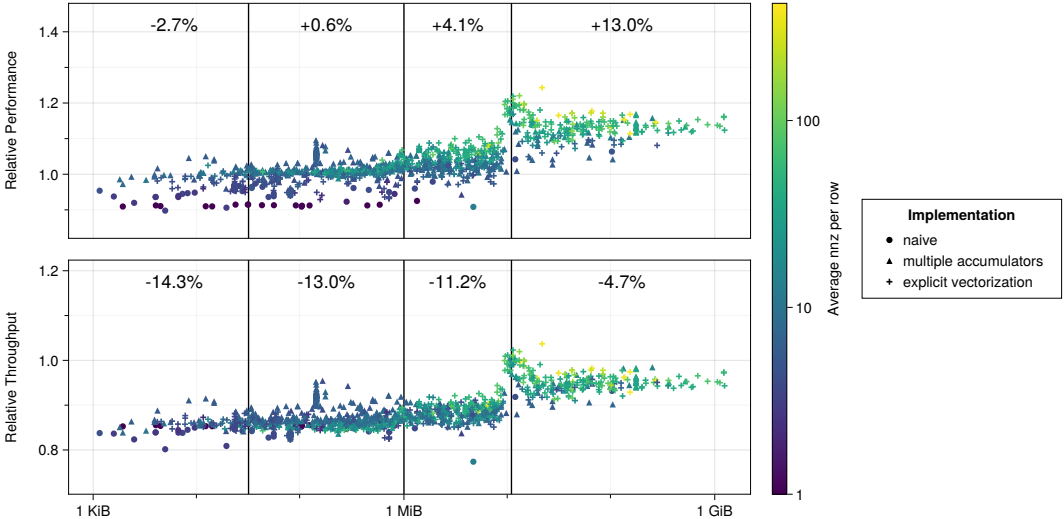
DA-CSR(`int16_t`) vs. Intel MKL CSR(`int32_t`) (Multi-Threaded)



Comparison With DA-CSR

DA-CSR(`int16_t`) vs. CSR(`int32_t`)

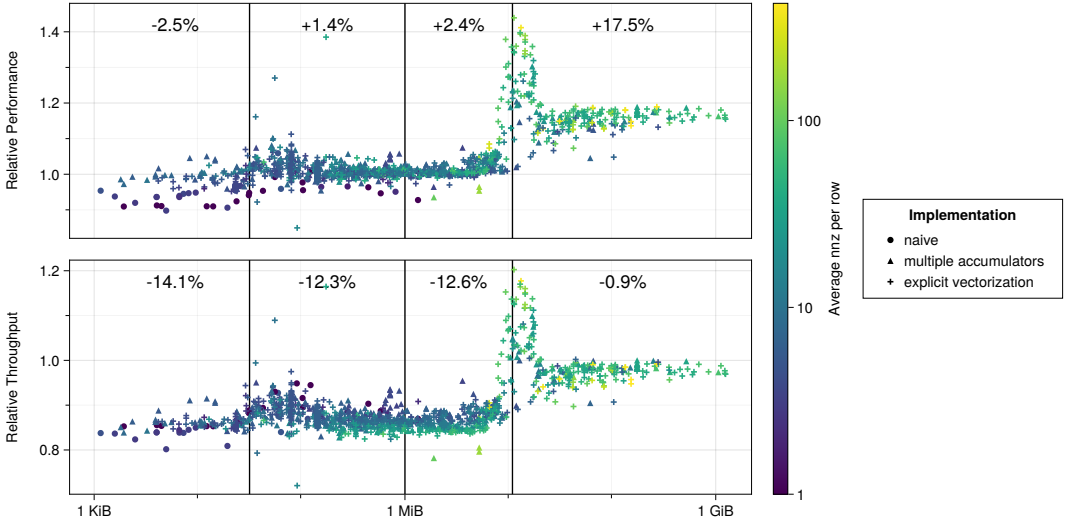
(Single-Threaded)



Comparison With DA-CSR

DA-CSR(`int16_t`) vs. CSR(`int32_t`)

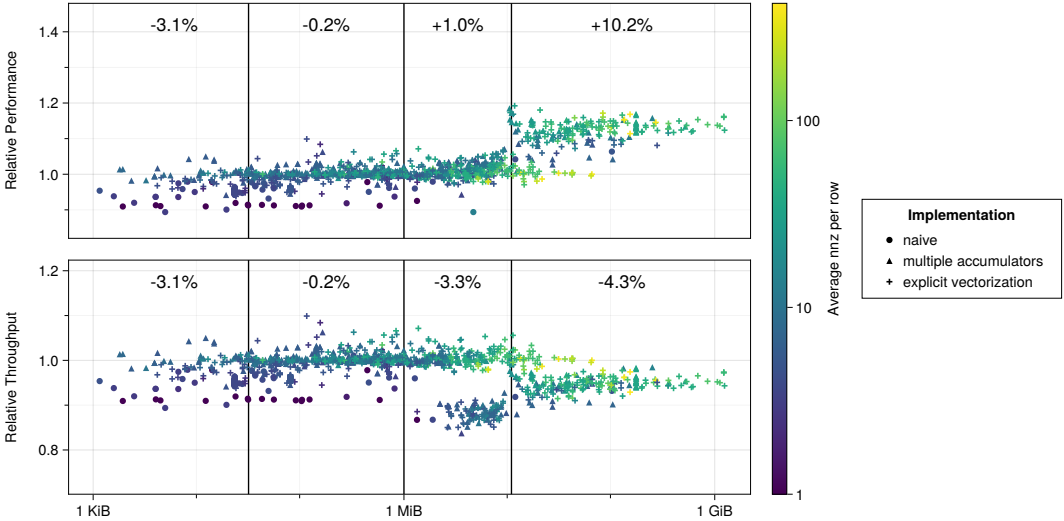
(Multi-Threaded)



Comparison With DA-CSR

DA-CSR(`int16_t`) vs. CSR(`int16_t`/`int32_t`)

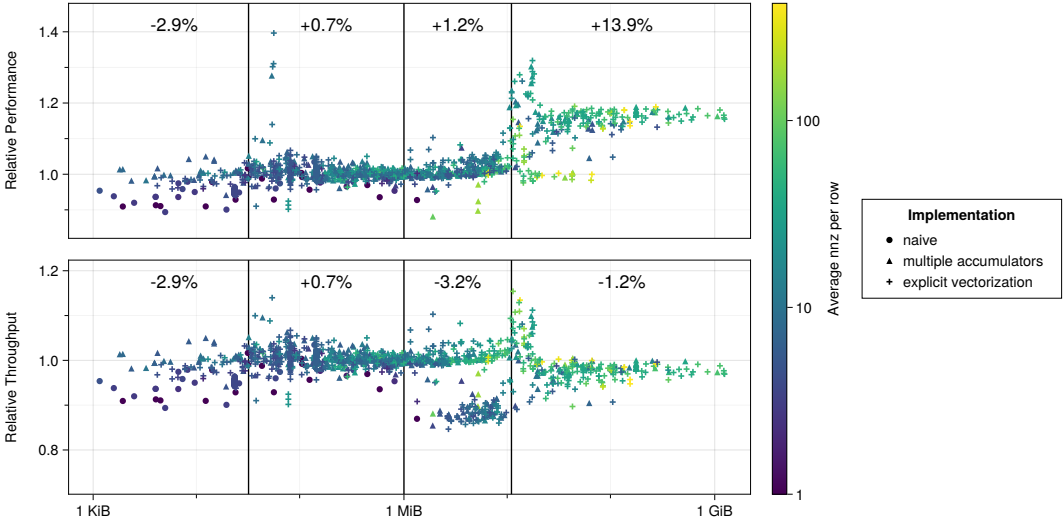
(Single-Threaded)



Comparison With DA-CSR

DA-CSR(`int16_t`) vs. CSR(`int16_t`/`int32_t`)

(Multi-Threaded)



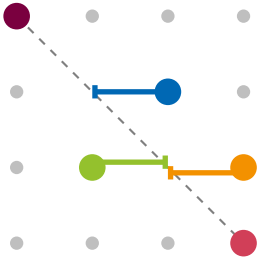
Outline

Summary

- Diagonally-Addressed (DA) storage is a technique applicable to many data types
  e.g. DA-CSR as seen today, DA-CSR$_5$, DA-SELL-C-σ, or DA-CSR as the leaf type within RSB

- Using DA-CSR(**`int16_t`**) over CSR(**`int32_t`**) yields up to 17 % more performance including vs. Intel MKL and is on par with CSR(**`int16_t`**), if the latter is applicable (which is almost what one can theoretically expect)

- Always use smallest integer types possible
  even using CSR(**`int16_t`**) over CSR(**`int32_t`**) yields up to 4 % or 17 % more performance, depending on the traffic



Outline

Appendix
Further Reading

Ahmad Abdelfattah et al, A survey of numerical linear algebra methods utilizing mixed-precision arithmetic, *The International Journal of High Performance Computing Applications* **35(4)**, 344 (2021), 10.1177/10943420211003313

Timothy A. Davis and Yifan Hu, 2011. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software* **38(1)**, Article 1 (2011), 25 pages, 10.1145/2049662.2049663

Jens Saak and Jonas Schulze, 2023. Diagonally-Addressed Matrix Nicknack: How to improve SpMV performance. *Submitted to PAMM.*
Data set: 10.5281/zenodo.7551699
Source code: 10.5281/zenodo.8104335
Preprint: 10.48550/arXiv.2307.06305

Appendix

What is the theoretical optimum in performance uplift?

Memory-bound Operations

If an operation is memory-bound, the measured throughput [GiB/s] is equal to the hardware limits. Thus, for any two measurements corresponding to the same work W [FLOP]:

$$1 = \frac{\text{traffic}_1/t_1}{\text{traffic}_2/t_2} = \frac{\text{traffic}_1}{\text{traffic}_2} \cdot \frac{W/t_1}{W/t_2} = \frac{\text{traffic}_1}{\text{traffic}_2} \cdot \frac{P_1}{P_2} \quad \Longleftrightarrow \quad \frac{P_2}{P_1} = \frac{\text{traffic}_1}{\text{traffic}_2}$$

- When replacing 32 bit indices by 16 bit ones:

$$\frac{P_{16}}{P_{32}} = \frac{\underbrace{(\text{nrows}+1)\cdot 32\,\text{bit} + \text{nnz}\cdot(32\,\text{bit}+64\,\text{bit})}_{A} + \underbrace{2\cdot\text{nrows}\cdot 64\,\text{bit}}_{x \text{ and } y}}{\underbrace{(\text{nrows}+1)\cdot 32\,\text{bit} + \text{nnz}\cdot(16\,\text{bit}+64\,\text{bit})}_{A} + \underbrace{2\cdot\text{nrows}\cdot 64\,\text{bit}}_{x \text{ and } y}} \approx \frac{32\,\text{bit}+64\,\text{bit}}{16\,\text{bit}+64\,\text{bit}} = \frac{6}{5}$$

Appendix

What is the theoretical optimum in performance uplift?

Memory-bound Operations

If an operation is memory-bound, the measured throughput [GiB/s] is equal to the hardware limits. Thus, for any two measurements corresponding to the same work W [FLOP]:

$$1 = \frac{\text{traffic}_1/t_1}{\text{traffic}_2/t_2} = \frac{\text{traffic}_1}{\text{traffic}_2} \cdot \frac{W/t_1}{W/t_2} = \frac{\text{traffic}_1}{\text{traffic}_2} \cdot \frac{P_1}{P_2} \quad \Longleftrightarrow \quad \frac{P_2}{P_1} = \frac{\text{traffic}_1}{\text{traffic}_2}$$

- When replacing 32 bit indices by 16 bit ones: up to +20 % performance [FLOP/s] (simplified)
- Example: `Janna/Bump_2911`

$$\frac{P_{16}}{P_{32}} = 1.155 \leq 1.191 = \frac{1.482\ \text{GiB}}{1.244\ \text{GiB}}$$

throughput ratio is 96.9 %

- Example: `Oberwolfach/bone010`

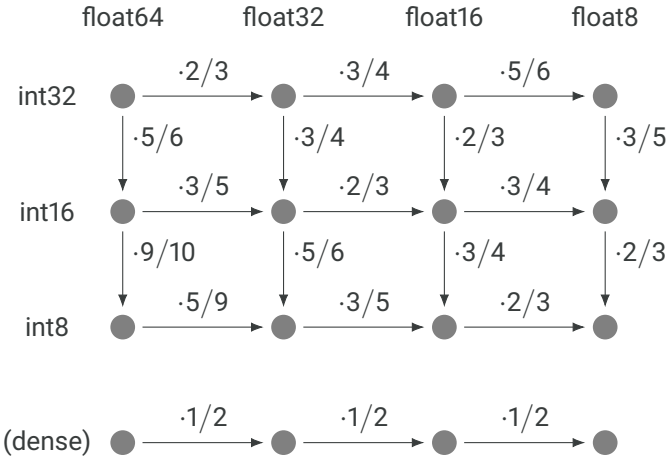
$$\frac{P_{16}}{P_{32}} = 1.184 \leq 1.195 = \frac{839.0\ \text{MiB}}{702.3\ \text{MiB}}$$

throughput ratio is 99.1 %

Appendix

What's in it for multi-precision algorithms?

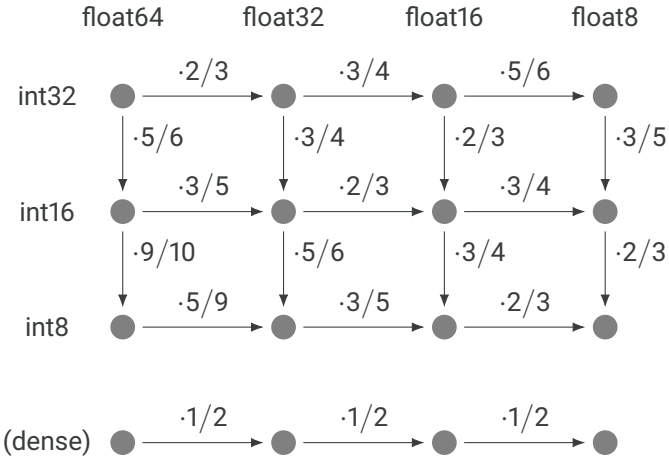
- Figure: Approximate changes in matrix-related traffic for (DA-) CSR as well as dense storage for several scalar and index types



Appendix

What's in it for multi-precision algorithms?

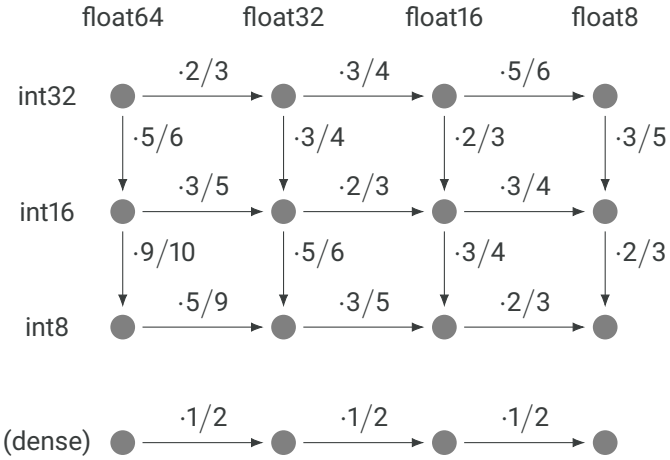
- Figure: Approximate changes in matrix-related traffic for (DA-) CSR as well as dense storage for several scalar and index types
- $\frac{P_2}{P_1} = \frac{\text{traffic}_1}{\text{traffic}_2}$ if memory-bound



Appendix

What's in it for multi-precision algorithms?

- Figure: Approximate changes in matrix-related traffic for (DA-) CSR as well as dense storage for several scalar and index types
- $\frac{P_2}{P_1} = \frac{\text{traffic}_1}{\text{traffic}_2}$ if memory-bound
- Thus: sparse storage w/ smaller integers allows for performance scaling closer to dense storage



Appendix

How to make Diagonally-Addressed storage easy to use?

We would like to bring DA storage to Trilinos, but where should we put this?

- Ideally Kokkos?
  such that projects outside Trilinos can use it, too
- Incorporate into `Tpetra::CrsMatrix`?
  e.g. via `CrsMatrix::fillComplete` or `CrsMatrix::transformToDacrs`
- Add new `Tpetra::DacrsMatrix`?
  may be too much hassle for many :-(
- Add RCM preconditioner that also transforms the matrix format?
  watch out: preconditioner must be applied only once
- ...

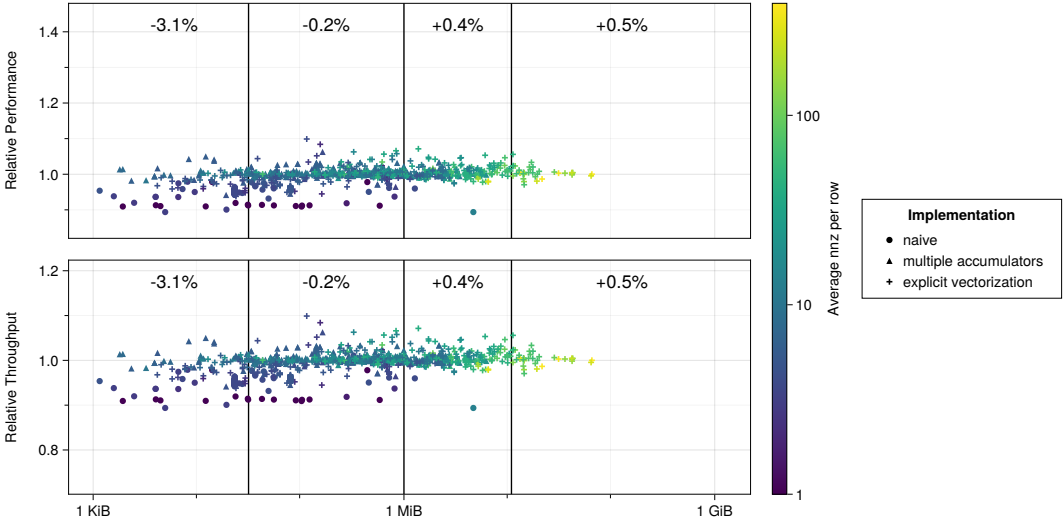
Goal: Build iterative multi-precision solver with Belos using DA storage.

We are looking for collaborations.

Appendix

DA-CSR(`int16_t`) vs. CSR(`int16_t`)

(Single-Threaded)



Appendix

DA-CSR(`int16_t`) vs. CSR(`int16_t`) (Multi-Threaded)