Package: delarr (via r-universe)

June 30, 2026

Type Package

Title Lazy Delayed Arrays with Fused Execution

Version 0.1.0

Description Provides a lightweight delayed array abstraction for lazy, fused evaluation of multi-dimensional numeric data. Compared with Bioconductor's 'DelayedArray', it offers a small S3 API with tidy-friendly verbs, chunked materialisation, and optional backends for HDF5, memory mapping, and shared-memory parallelism. Expression trees are optimised before streaming evaluation, and results can be written directly to disk without fully materialising arrays in memory.

URL https://bbuchsbaum.github.io/delarr/, https://github.com/bbuchsbaum/delarr

BugReports https://github.com/bbuchsbaum/delarr/issues

License MIT + file LICENSE

Depends R (>= 4.1)

Imports rlang

Suggests hdf5r, mmap, matrixStats, shard, testthat (>= 3.1.0), knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

Language en-US

Config/testthat/edition 3

RoxygenNote 7.3.3

NeedsCompilation yes

Author Bradley Buchsbaum [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-1108-4866>)

Maintainer Bradley Buchsbaum <brad.buchsbaum@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-06-30 18:40:02 UTC

RemoteUrl https://github.com/cran/delarr

RemoteRef HEAD

RemoteSha 434d8616669193fe6be284186f51a24750fc0dc7

Contents

`[.delarr` *Subset a delayed array*

Description

Performs array-style slicing lazily, capturing the indices in the DAG. For 2D arrays, standard `x[i, j]` syntax works. For N-d arrays, provide one index expression per dimension: `x[i, j, k, ...]`.

Usage

```
## S3 method for class 'delarr'
x[..., drop = FALSE]
```

Arguments

`x`	A `delarr`.
`...`	Index expressions, one per dimension. Missing indices select all.
`drop`	Logical indicating whether to drop dimensions (ignored lazily).

Value

A `delarr` containing the slice operation.

`as.matrix.delarr` *Materialise a delayed matrix as a base matrix*

Description

Materialise a delayed matrix as a base matrix

Usage

```
## S3 method for class 'delarr'
as.matrix(x, ...)
```

Arguments

`x`	A `delarr`.
`...`	Passed to `collect()`.

Value

A base matrix containing the realised data.

`block_apply` *Apply a function to streamed matrix blocks*

Description

Evaluates a `delarr` slice-by-slice, materialising manageable chunks for further processing without realising the full matrix.

Usage

```
block_apply(
  x,
  margin = c("cols", "rows"),
  size = 16384L,
  fn,
  parallel = FALSE,
  workers = NULL
)
```

Arguments

`x`	A `delarr` object.
`margin`	Dimension along which to chunk (`"cols"` or `"rows"`).
`size`	Approximate chunk size.
`fn`	Function applied to each materialised chunk.
`parallel`	Logical; process chunks in parallel when possible.
`workers`	Number of worker processes for parallel execution.

Value

A list of results returned by `fn`.

Examples

```
mat <- matrix(1:20, nrow = 4, ncol = 5)
darr <- delarr(mat)

# Apply function to column chunks
col_maxes <- block_apply(darr, margin = "cols", size = 2L, fn = function(block) {
  apply(block, 2, max)
})
unlist(col_maxes)

# Apply function to row chunks
row_means <- block_apply(darr, margin = "rows", size = 2L, fn = function(block) {
  rowMeans(block)
})
```

```
unlist(row_means)
```

`collect` *Materialise a delayed matrix*

Description

Streams column chunks from the backing seed, applying deferred operations and optional reductions on the fly. By default the result is returned as a base matrix or vector; alternatively, supply a writer via `into` to stream the output elsewhere (e.g., `hdf5_writer()`).

Usage

```
collect(
  x,
  into = NULL,
  chunk_size = NULL,
  chunk_margin = c("cols", "rows"),
  target_bytes = NULL,
  parallel = FALSE,
  workers = NULL,
  optimize = TRUE
)
```

Arguments

`x`	A `delarr` object.
`into`	Optional writer or callback used to receive streamed chunks.
`chunk_size`	Optional chunk size along `chunk_margin`.
`chunk_margin`	Chunking axis for non-reduction collection.
`target_bytes`	Optional memory budget (bytes) used to adapt chunk size.
`parallel`	Logical; attempt parallel chunk execution when safe.
`workers`	Number of worker processes when `parallel = TRUE`.
`optimize`	Logical; run lightweight DAG optimizations before evaluation.

Value

A realised matrix/vector, or `NULL` invisibly when writing to `into`.

Examples

```
# Basic materialization
mat <- matrix(1:12, nrow = 3, ncol = 4)
darr <- delarr(mat)
collect(darr)

# Collect after lazy operations
result <- darr |>
  d_map(~ .x^2) |>
  collect()
result
```

`collect_shard` *Parallel collect using shard's shared-memory workers*

Description

Evaluates a `delarr` pipeline in parallel using `shard::shard_map()`. This gives proper multi-process parallelism with shared-memory I/O, including parallel reductions.

Usage

```
collect_shard(x, workers = NULL, chunk_size = NULL, optimize = TRUE)
```

Arguments

`x`	A `delarr` object.
`workers`	Number of worker processes. Defaults to `parallel::detectCores() - 1`.
`chunk_size`	Column chunk size for sharding. If `NULL`, uses the seed's `chunk_hint` or a sensible default.
`optimize`	Logical; run DAG optimizations before evaluation.

Details

Pipelines that require full-matrix evaluation (row-wise center/scale/zscore/ detrend), paired RHS delarrs (`d_map2` with two delarrs), or generic (user-supplied) reductions automatically fall back to sequential `collect()`.

Value

A materialised matrix or vector.

Examples

```
if (requireNamespace("shard", quietly = TRUE)) {
  old_conn <- getAllConnections()
  mat <- matrix(rnorm(100), 10, 10)
  darr <- delarr_shard(mat)
  result <- collect_shard(darr |> d_map(~ .x^2), workers = 2)
  all.equal(result, mat^2)
  new_conn <- setdiff(getAllConnections(), old_conn)
  for (con in new_conn) try(close(getConnection(con)), silent = TRUE)
}
```

`colMeans2` *Column means for delayed matrices*

Description

Generic counterpart to `matrixStats::colMeans2()`. Methods are provided for `delarr` objects, but packages can extend the generic for their own delayed types.

Usage

```
colMeans2(x, ...)
```

Arguments

`x`	An object for which row means should be computed.
`...`	Additional arguments passed to methods.

Value

Typically a numeric vector of column means.

Examples

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
darr <- delarr(mat)

# Compute column means lazily
colMeans2(darr)

# Compare with base R
colMeans(mat)
```

colMeans2.delarr *Column means for a delayed matrix*

Description

Computes column means lazily via `d_reduce()`; acts as a drop-in replacement for `matrixStats::colMeans2()`.

Usage

```
## S3 method for class 'delarr'
colMeans2(x, ..., na.rm = FALSE)
```

Arguments

`x`	A `delarr` object.
`...`	Unused.
`na.rm`	Logical; remove missing values before averaging.

Value

A numeric vector of column means.

d_aperm *Permute dimensions of a delayed array*

Description

Permute dimensions of a delayed array

Usage

```
d_aperm(x, perm = rev(seq_along(dim(x))), chunk_size = NULL)
```

Arguments

`x`	A `delarr`.
`perm`	A permutation of `seq_along(dim(x))`.
`chunk_size`	Optional chunk size used for internal pulls.

Value

A permuted `delarr`.

d_center *Center a delayed matrix along rows or columns*

Description

Center a delayed matrix along rows or columns

Usage

```
d_center(x, dim = c("rows", "cols"), axis = NULL, na.rm = FALSE)
```

Arguments

x	A delarr.
dim	Dimension along which to subtract the mean.
axis	Integer axis for N-d arrays (alternative to dim).
na.rm	Logical; remove missing values when computing the centre.

Value

A delarr with a deferred centering operation.

Examples

```
mat <- matrix(c(1, 2, 3, 10, 20, 30), nrow = 2, ncol = 3)
darr <- delarr(mat)

# Center rows (subtract row means)
centered_rows <- darr |> d_center(dim = "rows") |> collect()
centered_rows
rowMeans(centered_rows)  # Should be ~0

# Center columns (subtract column means)
centered_cols <- darr |> d_center(dim = "cols") |> collect()
colMeans(centered_cols)  # Should be ~0
```

d_detrend *Detrend a delayed matrix*

Description

Removes a polynomial trend of the specified degree along the chosen dimension.

Usage

```
d_detrend(x, dim = c("rows", "cols"), axis = NULL, degree = 1L)
```

Arguments

x	A delarr.
dim	Dimension along which to fit the trend.
axis	Integer axis for N-d arrays (alternative to dim).
degree	Polynomial degree (default 1).

Value

A delarr with the detrend operation queued.

Examples

```
# Create matrix with linear trend in rows
mat <- matrix(1:12 + rep(1:4, each = 3), nrow = 3, ncol = 4)
darr <- delarr(mat)

# Remove linear trend along rows
detrended <- darr |> d_detrend(dim = "rows", degree = 1L) |> collect()
detrended

# Remove quadratic trend
quad_detrend <- darr |> d_detrend(dim = "rows", degree = 2L) |> collect()
quad_detrend
```

---

d_map *Apply an elementwise transformation lazily*

Description

Apply an elementwise transformation lazily

Usage

```
d_map(x, f)
```

Arguments

x	A delarr.
f	A function or formula suitable for rlang::as_function().

Value

A delarr representing the transformation.

Examples

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
darr <- delarr(mat)

# Apply elementwise transformation with formula
squared <- darr |> d_map(~ .x^2) |> collect()
squared

# Apply with function
logged <- darr |> d_map(log1p) |> collect()
logged
```

---

d_map2 *Apply a binary elementwise transformation lazily*

Description

Apply a binary elementwise transformation lazily

Usage

```
d_map2(x, y, f)
```

Arguments

x	A delarr.
y	Another delarr, matrix, or numeric vector/scalar.
f	A function or formula combining two arguments.

Value

A delarr representing the fused binary operation.

Examples

```
mat1 <- matrix(1:12, nrow = 3, ncol = 4)
mat2 <- matrix(12:1, nrow = 3, ncol = 4)
darr1 <- delarr(mat1)
darr2 <- delarr(mat2)

# Binary operation between two delayed matrices
added <- d_map2(darr1, darr2, ~ .x + .y) |> collect()
added

# Binary operation with scalar
scaled <- d_map2(darr1, 10, ~ .x * .y) |> collect()
scaled
```

d_matmul *Delayed matrix multiplication*

Description

Delayed matrix multiplication

Usage

```
d_matmul(x, y, chunk_size = NULL)
```

Arguments

x	A delarr or base matrix.
y	A delarr or base matrix.
chunk_size	Optional chunk size used during block pulls.

Value

A delarr representing %*%.

d_reduce *Reduce along a dimension lazily*

Description

For 2D arrays use dim = "rows" or "cols". For N-d arrays you can also supply a numeric axis indicating which dimension to collapse.

Usage

```
d_reduce(x, f = base::sum, dim = c("rows", "cols"), axis = NULL, na.rm = FALSE)
```

Arguments

x	A delarr.
f	A reduction function (defaults to sum).
dim	Dimension to reduce: "rows" (keep rows, collapse cols) or "cols" (keep cols, collapse rows).
axis	Integer axis to collapse (alternative to dim for N-d arrays). Takes precedence over dim when both are supplied.
na.rm	Logical; remove missing values while reducing.

Value

A `delarr` capturing the reduction.

Examples

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
darr <- delarr(mat)

row_sums <- darr |> d_reduce(sum, dim = "rows") |> collect()
row_sums

col_means <- darr |> d_reduce(mean, dim = "cols") |> collect()
col_means
```

---

d_reduce_many *Run multiple reductions and collect results*

Description

Run multiple reductions and collect results

Usage

```
d_reduce_many(
  x,
  fns,
  dim = c("rows", "cols"),
  na.rm = FALSE,
  chunk_size = NULL,
  simplify = TRUE
)
```

Arguments

`x`	A `delarr`.
`fns`	A named list of reduction functions.
`dim`	Reduction dimension (`"rows"` or `"cols"`).
`na.rm`	Logical; remove missing values in each reducer.
`chunk_size`	Optional chunk size passed to `collect()`.
`simplify`	Logical; combine equal-length outputs into a matrix.

Value

A named list (or matrix when `simplify = TRUE`) of reductions.

d_scale *Scale a delayed matrix along rows or columns*

Description

Scale a delayed matrix along rows or columns

Usage

```
d_scale(
  x,
  dim = c("rows", "cols"),
  axis = NULL,
  center = TRUE,
  scale = TRUE,
  na.rm = FALSE
)
```

Arguments

x	A delarr.
dim	Dimension to scale.
axis	Integer axis for N-d arrays (alternative to dim).
center	Logical; subtract the mean before scaling.
scale	Logical; divide by the standard deviation.
na.rm	Logical; remove missing values when computing statistics.

Value

A delarr with a deferred scaling operation.

Examples

```
mat <- matrix(c(1, 2, 3, 10, 20, 30), nrow = 2, ncol = 3)
darr <- delarr(mat)

# Scale rows (center and divide by SD)
scaled <- darr |> d_scale(dim = "rows") |> collect()
scaled

# Scale without centering
scaled_only <- darr |> d_scale(dim = "rows", center = FALSE) |> collect()
scaled_only
```

d_transpose *Transpose a delayed matrix*

Description

Transpose a delayed matrix

Usage

```
d_transpose(x, chunk_size = NULL)
```

Arguments

x	A delarr.
chunk_size	Optional chunk size used for internal pulls.

Value

A transposed delarr.

d_where *Apply a boolean mask to a delayed matrix*

Description

Elements failing the predicate are replaced with fill at materialisation time.

Usage

```
d_where(x, predicate, fill = 0)
```

Arguments

x	A delarr.
predicate	A function or formula returning a logical matrix.
fill	Replacement value for elements where the predicate is FALSE.

Value

A delarr including the mask.

Examples

```
mat <- matrix(c(-1, 2, -3, 4, -5, 6), nrow = 2, ncol = 3)
darr <- delarr(mat)

# Replace negative values with 0
masked <- darr |> d_where(~ .x >= 0, fill = 0) |> collect()
masked

# Replace values below threshold with NA
filtered <- darr |> d_where(~ .x > 1, fill = NA) |> collect()
filtered
```

---

d_zscore *Z-score a delayed matrix*

Description

Equivalent to centering and scaling with unit variance.

Usage

```
d_zscore(x, dim = c("rows", "cols"), axis = NULL, na.rm = FALSE)
```

Arguments

x	A delarr.
dim	Dimension over which to compute the z-score.
axis	Integer axis for N-d arrays (alternative to dim).
na.rm	Logical; remove missing values when computing statistics.

Value

A delarr with the z-score applied lazily.

Examples

```
mat <- matrix(c(1, 2, 3, 10, 20, 30), nrow = 2, ncol = 3)
darr <- delarr(mat)

# Z-score normalize rows
zscored <- darr |> d_zscore(dim = "rows") |> collect()
zscored

# Row means should be ~0, row SDs should be ~1
rowMeans(zscored)
```

delarr *Create a delayed matrix*

Description

Wraps an existing matrix or `delarr_seed` in the lightweight delayed pipeline. Matrix inputs are wrapped in a seed that simply slices the source object, while `delarr` inputs are returned unchanged.

Usage

```
delarr(x, ...)
```

Arguments

`x`	A base matrix or a `delarr_seed` to wrap.
`...`	Future extensions; currently ignored.

Value

A `delarr` object representing the delayed matrix.

Examples

```
# Create a delayed matrix from a regular matrix
mat <- matrix(1:12, nrow = 3, ncol = 4)
darr <- delarr(mat)
darr

# Operations are queued lazily
result <- darr * 2
result

# Materialize with collect()
collect(result)
```

delarr_backend *Wrap a custom backend as a delayed matrix*

Description

Provides a convenience helper that turns a user-supplied slice function into a ready-to-use `delarr` object.

Usage

```
delarr_backend(
  nrow,
  ncol,
  pull,
  chunk_hint = NULL,
  dimnames = NULL,
  begin = NULL,
  end = NULL
)
```

Arguments

nrow, ncol	Dimensions of the logical matrix.
pull	Function of rows and cols returning a base matrix slice.
chunk_hint	Optional preferred chunking metadata.
dimnames	Optional dimnames to expose lazily.
begin	Optional function invoked before streaming.
end	Optional function invoked after streaming.

Value

A delarr backed by the provided pull function.

Examples

```
# Create a custom backend from a pull function
data <- matrix(1:20, nrow = 4, ncol = 5)

darr <- delarr_backend(
  nrow = 4,
  ncol = 5,
  pull = function(rows = NULL, cols = NULL) {
    rows <- rows %||% seq_len(4)
    cols <- cols %||% seq_len(5)
    data[rows, cols, drop = FALSE]
  }
)
darr

# Use like any delarr
result <- darr |> d_map(~ .x * 2) |> collect()
result
```

`delarr_hdf5` *Create a delayed array sourced from an HDF5 dataset*

Description

Uses `hdf5r` to lazily read slices from disk on demand.

Usage

```
delarr_hdf5(path, dataset)
```

Arguments

`path`	Path to the HDF5 file.
`dataset`	Name of the dataset within the file.

Value

A `delarr` that streams data from the HDF5 dataset.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
  # Create a temporary HDF5 file
  tf <- tempfile(fileext = ".h5")
  data <- matrix(1:20, nrow = 4, ncol = 5)

  # Write test data
  f <- hdf5r::H5File$new(tf, mode = "w")
  f$create_dataset("X", robj = data)
  f$close_all()

  # Load as delayed array
  darr <- delarr_hdf5(tf, "X")
  darr

  # Apply operations and collect
  result <- darr |> d_map(~ .x * 2) |> collect()
  result

  # Clean up
  unlink(tf)
}
```

delarr_mem *Create a delayed matrix from an in-memory matrix*

Description

Create a delayed matrix from an in-memory matrix

Usage

```
delarr_mem(x)
```

Arguments

x	A numeric or logical matrix, or an array with at least 2 dimensions.

Value

A delarr referencing the original object.

Examples

```
# Wrap an in-memory matrix
mat <- matrix(1:12, nrow = 3, ncol = 4)
darr <- delarr_mem(mat)
darr

# Apply operations lazily
result <- darr |> d_center(dim = "rows") |> collect()
result
```

delarr_mmap *Create a delayed matrix from a memory-mapped file*

Description

Uses the mmap package to lazily read slices from a binary file on demand. The file must contain raw numeric data in column-major order (R's default).

Usage

```
delarr_mmap(path, nrow, ncol, mode = NULL)
```

Arguments

path	Path to the binary file containing matrix data.
nrow	Number of rows in the matrix.
ncol	Number of columns in the matrix.
mode	mmap mode object specifying data type. Default is double().

Value

A `delarr` that streams data from the memory-mapped file.

Note

This backend supports 2D matrices only. For N-d arrays, use `delarr_hdf5()` or wrap an in-memory array with `delarr()`.

Examples

```
if (requireNamespace("mmap", quietly = TRUE)) {
  # Create a binary file with matrix data
  mat <- matrix(1:20, nrow = 4, ncol = 5)
  tf <- tempfile()
  writeBin(as.double(mat), tf)

  # Load as delayed array
  darr <- delarr_mmap(tf, nrow = 4, ncol = 5)
  darr

  # Apply operations and collect
  result <- darr |> d_map(~ .x * 2) |> collect()
  result

  # Clean up
  unlink(tf)
}
```

`delarr_seed` *Construct a seed backend for* `delarr`

Description

Seeds encapsulate storage access for delayed matrices. They define matrix dimensions and a `pull()` function that returns materialised slices.

Usage

```
delarr_seed(
  nrow,
  ncol,
  pull,
  chunk_hint = NULL,
  dimnames = NULL,
  begin = NULL,
  end = NULL
)
```

Arguments

nrow, ncol	Number of rows and columns.
pull	A function accepting rows and cols indices and returning a base matrix slice.
chunk_hint	Optional list describing preferred chunk sizes (e.g. list(cols = 4096L)).
dimnames	Optional list of dimnames to expose lazily.
begin	Optional function invoked before streaming begins.
end	Optional function invoked after streaming completes.

Value

An object of class delarr_seed.

Examples

```
# Create a custom seed with a pull function
data <- matrix(1:12, nrow = 3, ncol = 4)

seed <- delarr_seed(
  nrow = 3,
  ncol = 4,
  pull = function(rows = NULL, cols = NULL) {
    rows <- rows %||% seq_len(3)
    cols <- cols %||% seq_len(4)
    data[rows, cols, drop = FALSE]
  }
)
seed

# Wrap in delarr() to use with lazy operations
darr <- delarr(seed)
result <- darr |> d_map(~ .x * 2) |> collect()
result
```

---

delarr_seed_nd *Construct an N-dimensional seed backend for* delarr

---

Description

Creates a seed for arrays with 2 or more dimensions. The pull function receives a list of per-dimension index vectors and returns the corresponding sub-array.

Usage

```
delarr_seed_nd(
  dims,
  pull,
  chunk_hint = NULL,
```

```
  dimnames = NULL,
  begin = NULL,
  end = NULL
)
```

Arguments

dims	Integer vector of dimension extents (length >= 2).
pull	A function accepting a single argument indices — a named or positional list of integer vectors (one per dimension, or NULL for "all") — and returning an array with the requested sub-dimensions.
chunk_hint	Optional list describing preferred chunk sizes.
dimnames	Optional list of dimnames (one element per dimension).
begin	Optional function invoked before streaming begins.
end	Optional function invoked after streaming completes.

Value

An object of class delarr_seed.

Examples

```
arr <- array(seq_len(24), dim = c(3, 4, 2))
seed <- delarr_seed_nd(
  dims = c(3, 4, 2),
  pull = function(indices) {
   idx <- lapply(seq_along(dim(arr)), function(k) indices[[k]] %||% seq_len(dim(arr)[k]))
    do.call("[", c(list(arr), idx, list(drop = FALSE)))
  }
)
dim(seed)
```

---

delarr_shard *Create a delayed array backed by shared memory*

Description

Wraps a numeric matrix or array into shard's shared memory, returning a delarr. The shared ALTREP vector is stored on the seed so that collect_shard() can reuse it without re-sharing (zero-copy).

Usage

```
delarr_shard(x, backing = "auto")
```

Arguments

x	A numeric matrix or array.
backing	Backing type passed to shard::share().

Value

A delarr backed by shared memory.

Examples

```
if (requireNamespace("shard", quietly = TRUE)) {
  mat <- matrix(rnorm(20), 4, 5)
  darr <- delarr_shard(mat)
  collect(darr)
}
```

---

dim.delarr *Dimensions of a delayed array*

Description

Computes the realised dimensions after taking queued slice and reduce operations into account.

Usage

```
## S3 method for class 'delarr'
dim(x)
```

Arguments

x	A delarr.

Value

An integer vector of dimension extents.

dim.delarr_seed *Dimensions for a* delarr_seed

Description

Dimensions for a delarr_seed

Usage

```
## S3 method for class 'delarr_seed'
dim(x)
```

Arguments

x	A delarr_seed.

Value

An integer vector of dimension extents.

dimnames.delarr *Dimension names for a delayed array*

Description

Dimension names for a delayed array

Usage

```
## S3 method for class 'delarr'
dimnames(x)
```

Arguments

x	A delarr.

Value

A list of per-dimension names or NULL placeholders.

explain *Explain a delayed execution plan*

Description

Explain a delayed execution plan

Usage

```
explain(
  x,
  chunk_size = NULL,
  chunk_margin = c("cols", "rows"),
  target_bytes = NULL,
  optimize = TRUE
)
```

Arguments

x	A delarr.
chunk_size	Optional chunk size hint.
chunk_margin	Chunking axis for non-reduction materialization.
target_bytes	Optional memory budget used for adaptive chunking.
optimize	Logical; whether to explain the optimized DAG.

Value

An object of class delarr_explain.

hdf5_writer *HDF5 writer for streaming* collect()

Description

Creates or extends an HDF5 dataset so that collect(x, into = writer) can stream column blocks directly to disk without materialising the full matrix in memory.

Usage

```
hdf5_writer(path, dataset, ncol, chunk = c(128L, 4096L), compression = 4L)
```

Arguments

path	Path to the HDF5 file. The file is created if it does not exist.
dataset	Name of the dataset to create or update.
ncol	Total number of columns that will be written. The writer uses this to size the target dataset up-front.
chunk	Integer vector of length two giving the chunk size (rows, cols) for the target dataset (optional).
compression	Gzip compression level (0-9). Use 0 for no compression, higher values for better compression at cost of speed. Default is 4. Use NULL to disable compression entirely.

Value

A writer object with $write() and $finalize() methods understood by collect().

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
  # Create source data in a temp HDF5 file
  tf_in <- tempfile(fileext = ".h5")
  data <- matrix(1:20, nrow = 4, ncol = 5)
  f <- hdf5r::H5File$new(tf_in, mode = "w")
  f$create_dataset("X", robj = data)
  f$close_all()

  # Load, transform, and stream to output file
  darr <- delarr_hdf5(tf_in, "X")
  transformed <- darr |> d_center(dim = "cols")

  tf_out <- tempfile(fileext = ".h5")
  writer <- hdf5_writer(tf_out, "result", ncol = ncol(transformed), compression = 4L)
  collect(transformed, into = writer)

  # Verify output
  g <- hdf5r::H5File$new(tf_out, mode = "r")
  result <- g[["result"]]$read()
  g$close_all()
  result

  # Clean up
  unlink(c(tf_in, tf_out))
}
```

Ops.delarr *Arithmetic and comparison operators for* delarr

Description

Supports elementwise operations between delayed matrices or between a delayed matrix and scalars/matrices.

Usage

```
## S3 method for class 'delarr'
Ops(e1, e2)
```

Arguments

e1, e2	Operands supplied by the R math group generics.

Value

A delarr representing the fused operation.

---

optimize_delarr *Optimize a delayed pipeline*

Description

Applies lightweight algebraic simplifications to reduce unnecessary work during collect().

Usage

```
optimize_delarr(x)
```

Arguments

x	A delarr object.

Value

A delarr with an optimized operation list.

print.delarr *Pretty-print a delayed matrix*

Description

Pretty-print a delayed matrix

Usage

```
## S3 method for class 'delarr'
print(x, ...)
```

Arguments

x	A delarr.
...	Unused.

Value

The original object, invisibly.

profile_collect *Profile* collect() *runtime*

Description

Profile collect() runtime

Usage

```
profile_collect(x, reps = 3L, ...)
```

Arguments

x	A delarr.
reps	Number of repetitions.
...	Additional arguments forwarded to collect().

Value

An object of class delarr_profile.

read_hdf5 *Read a matrix from an HDF5 file*

Description

Simple convenience function to read a matrix from an HDF5 dataset. For lazy/streaming access, use `delarr_hdf5()` instead.

Usage

```
read_hdf5(path, dataset)
```

Arguments

`path`	Path to the HDF5 file.
`dataset`	Name of the dataset to read.

Value

The matrix stored in the dataset.

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
  # Write and read back
  mat <- matrix(1:20, nrow = 4, ncol = 5)
  tf <- tempfile(fileext = ".h5")
  write_hdf5(mat, tf, "X")
  read_hdf5(tf, "X")

  # Clean up
  unlink(tf)
}
```

rowMeans2 *Row means for delayed matrices*

Description

Generic counterpart to `matrixStats::rowMeans2()`. Methods are provided for `delarr` objects, but packages can extend the generic for their own delayed types.

Usage

```
rowMeans2(x, ...)
```

Arguments

x	An object for which row means should be computed.
...	Additional arguments passed to methods.

Value

Typically a numeric vector of row means.

Examples

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
darr <- delarr(mat)

# Compute row means lazily
rowMeans2(darr)

# Compare with base R
rowMeans(mat)
```

---

rowMeans2.delarr *Row means for a delayed matrix*

Description

Computes row means lazily via `d_reduce()`; acts as a drop-in replacement for `matrixStats::rowMeans2()`.

Usage

```
## S3 method for class 'delarr'
rowMeans2(x, ..., na.rm = FALSE)
```

Arguments

x	A `delarr` object.
...	Unused.
na.rm	Logical; remove missing values before averaging.

Value

A numeric vector of row means.

shard_writer *Shared-memory writer for streaming* collect()

Description

Creates a writer object backed by shard::buffer() conforming to delarr's writer protocol. Pass to collect(x, into = shard_writer(...)) to stream results into shared memory.

Usage

```
shard_writer(nrow, ncol, backing = "auto")
```

Arguments

nrow, ncol	Dimensions of the output matrix.
backing	Backing type passed to shard::buffer().

Value

A writer list with $write(), $finalize(), $result(), and $close() methods.

Note

This writer supports 2D matrices only. N-d array collection does not currently support writer-style into targets.

Examples

```
if (requireNamespace("shard", quietly = TRUE)) {
  mat <- matrix(rnorm(20), 4, 5)
  darr <- delarr(mat)
  w <- shard_writer(4, 5)
  collect(darr |> d_map(~ .x * 2), into = w)
  w$result()
  w$close()
}
```

write_hdf5 *Write a matrix to an HDF5 file*

Description

Simple convenience function to write a matrix to an HDF5 dataset. For streaming writes during collect(), use hdf5_writer() instead.

Usage

```
write_hdf5(x, path, dataset, compression = 4L)
```

Arguments

x	A matrix to write.
path	Path to the HDF5 file. Created if it doesn't exist.
dataset	Name of the dataset to create.
compression	Gzip compression level (0-9), or NULL for no compression.

Value

The path to the HDF5 file (invisibly).

Examples

```
if (requireNamespace("hdf5r", quietly = TRUE)) {
  # Write a matrix to HDF5
  mat <- matrix(1:20, nrow = 4, ncol = 5)
  tf <- tempfile(fileext = ".h5")
  write_hdf5(mat, tf, "X")

  # Read it back as a delarr
  darr <- delarr_hdf5(tf, "X")
  collect(darr)

  # Clean up
  unlink(tf)
}
```

Index