

LIBXS Samples

Foepl Fingerprint

This sample generates paper-oriented experiments for the LIBXS Foepl fingerprint API. It focuses on the fingerprint itself, leaving Rosetta as the higher-level algebra/type-discovery showcase.

What It Produces

Set FPRINT_OUTDIR to collect CSV artifacts:

```
mkdir -p results/fprint
FPRINT_OUTDIR=results/fprint ./fprint.x
```

The sample writes:

File	Purpose
convergence.csv	Same functions sampled at increasing resolution and compared to a high-resolution reference.
sensitivity.csv	Perturbations with matched element RMS but different structural character.
geometry.csv	Foepl boundary area, centroid, moments, and shape fingerprints.
hierarchy.csv	2-D smooth, creased, diagonal-creased, and noisy fields in hierarchical and per-axis modes.
compression.csv	Newton truncation order versus reconstruction error.
collision.csv	Pseudometric collision test: pairs with shared L2 norms but different signed means.
timing.csv	Wall-clock cost per element at various input sizes.
convergence_rate.csv	Log-log convergence rate (slope and R-squared) for smooth functions.

Build

```
cd samples/fprint
make GNU=1
```

Use From The Paper

The Foepl paper driver `papers/foepl/run.sh` builds this sample when needed and stores artifacts under `papers/foepl/results/fprint` by default.

Batched GEMM

Exercises the LIBXS batched GEMM API (`libxs_gemm.h`) in several flavors: strided, pointer-array, indexed, and grouped. All programs are multithreaded via OpenMP, report wall-clock time and GFLOPS/s, and optionally validate results via `libxs_matdiff`.

Building

```
cd samples/gemm
make GNU=1
```

LIBXS must be built first from the repository root.

Programs

gemm_strided Strided batch DGEMM: all matrices packed contiguously with constant stride. Uses `libxs_gemm_index` with `index_stride=0`.

```
./gemm_strided.x [M [N [K [batchsize [nrepeat [beta [pad]]]]]]]
```

Defaults: M=N=K=23, batchsize=30000, nrepeat=3, beta=1.0, pad=0.

gemm_batch Pointer-array batch DGEMM: matrices accessed through pointer arrays. Supports a duplicate C-matrix mode to exercise lock-forward sync.

```
./gemm_batch.x [M [N [K [batchsize [nrepeat [dup [beta [pad]]]]]]]]
```

Defaults: M=N=K=23, batchsize=30000, nrepeat=3, dup=0, beta=1.0, pad=0.

dup	Mode	Description
0	None	Each C-matrix is unique (LIBXS_GEMM_FLAG_NOLOCK)
1	Sorted	Half-unique C-pointers, duplicates consecutive
2	Shuffled	Half-unique C-pointers, randomly shuffled

gemm_index Index-array batch DGEMM: matrices in contiguous buffers, addressed through explicit element-offset index arrays (ia, ib, ic).

```
./gemm_index.x [M [N [K [batchsize [nrepeat [beta [pad]]]]]]]
```

Defaults: M=N=K=23, batchsize=30000, nrepeat=3, beta=1.0, pad=0.

gemm_indexf Fortran variant of `gemm_index`. Demonstrates the LIBXS Fortran module interface with one-based index arrays and C_LOC/C_SIZEOF interop. Requires a Fortran compiler.

```
./gemm_indexf.x [M [N [K [batchsize [nrepeat]]]]]
```

Defaults: M=N=K=23, batchsize=30000, nrepeat=3.

gemm_groups Grouped batch DGEMM: multiple groups of different matrix shapes dispatched in sequence, each with its own `libxs_gemm_config_t`. Per-group dimensions grow by 4 starting from `base_m`.

```
./gemm_groups.x [ngroups [batch_per_group [nrepeat [base_m [beta [pad]]]]]]
```

Defaults: ngroups=2 (max 4), batch_per_group=30000, nrepeat=3, base_m=8, beta=1.0, pad=0.

Validation

Set the CHECK environment variable to validate results:

```
CHECK=1 ./gemm_strided.x
CHECK=1 ./gemm_batch.x 23 23 23 1000 1 2
CHECK=1 ./gemm_index.x
CHECK=1 ./gemm_groups.x
```

When padding is enabled (pad>0), the check also verifies that leading-dimension padding in C-matrices has not been overwritten.

Runtime Controls

Set LIBXS_GEMM_PRINT=0 to print a compact GEMM registry summary at termination. The first line reports registry size, capacity, memory, and the selected LIBXS_GEMM_BACKEND policy; the second line reports a histogram by datatype and backend.

Memory Comparison Benchmark

Benchmarks `libxs_diff`, `libxs_memcmp`, and the C standard library's `memcmp` for both large contiguous comparisons and many small fixed-size comparisons. A Fortran variant compares `libxs_diff` against Fortran array intrinsics.

Building

```
cd samples/memory
make GNU=1
```

Produces memcmp.x (C) and, if a Fortran compiler is found, memcmpf.x and matcpyf.x.

Usage (C)

```
./memcmp.x [elsize [stride [nelems [niters]]]]
```

Argument	Default	Description
elsize	INSIZE/MAXSIZE (compile, 64)	Element comparison size in bytes
stride	max(MAXSIZE, elsize)	Distance between successive elements
nelems	2 GB / stride	Number of elements
niters	5	Repetitions (arithmetic average)

Environment Variables

Variable	Default	Description
STRIDED	0	0: one-call when stride==elsize, else loop; 1: seq loop; >=2: random
CHECK	0	Non-zero: validation pass (inject diffs, cross-check implementations)
MISMATCH	0	0: equal buffers; 1: first byte; 2: middle; 3: last; >=4: random
OFFSET	0	Shift buffer b off its natural alignment by this many bytes

Compile-Time Knobs

Macro	Default	Description
MAXSIZE	64	Stride floor (elements spaced >= this far)
INSIZE	MAXSIZE	Default element size when argument is omitted

```
## 32-byte elements, sequential strided access, 10 repetitions
./memcmp.x 32 0 0 10

## 8-byte elements, random traversal, 3 repetitions
STRIDED=2 ./memcmp.x 8 0 0 3

## contiguous (single-call) comparison, ~2 GB
./memcmp.x 64 64 0 5

## mismatch at last byte, buffer b misaligned by 3 bytes
MISMATCH=3 OFFSET=3 ./memcmp.x 32 0 0 5
```

Environment variables and compile-time macros apply only to memcmp.x.

Usage (Fortran -- memcmpf)

```
./memcmpf.x [nelements [nrepeat]]
```

Element type is hard-coded as INTEGER(4). Compares ~2 GB by default. Reports per-iteration times (ms) and average throughput (MB/s).

Usage (Fortran -- matcopyf)

```
./matcopyf.x [m [n [ldi [ldo [nrepeat [nmb]]]]]]
```

Argument	Default	Description
m	4096	First matrix dimension
n	m	Second matrix dimension
ldi	m (enforced m)	Leading dimension of input
ldo	ldi	Leading dimension of output
nrepeat	2	Number of repetitions
nmb	2048	Memory budget in MB

Benchmarks `libxs_matcopy` and `libxs_matcopy_task` for matrix copy and zeroing. When the matrix is square and `ldi==ldo`, also benchmarks `libxs_otrans` and `libxs_otrans_task` for out-of-place transpose.

What Is Measured

Three comparison implementations are benchmarked:

- `libxs_diff` -- SIMD-dispatched short-buffer comparison (SSE/AVX2/AVX-512). Limited to `elsize < 256` bytes, per-element.
- `libxs_memcmp` -- SIMD-dispatched `memcmp` replacement. Single call for contiguous buffers, or per-element for strided access.
- `stdlib memcmp` -- C library implementation under the same pattern.

Each kernel gets freshly initialized data before its timed section. An untimed warm-up pass runs before the first measured iteration. The summary reports min / median / max across iterations plus peak MB/s. Reported MB/s counts both buffers (`2 * nbytes / time`).

Ozaki-Scheme Low-Precision GEMM

Intercepts `DGEMM`, `SGEMM`, `ZGEMM`, and `CGEMM` at link time, replacing them with low-precision Ozaki schemes (mantissa slicing or CRT). Real GEMM calls go through the selected scheme; complex GEMM (`ZGEMM/CGEMM`) is mapped to real GEMM internally.

Build

```
cd samples/ozaki
make GNU=1 -j $(nproc)
```

LIBXS must be built first from the repository root. When a sibling `libxstream` directory is detected, the optional OpenCL/GPU path is compiled in (see `OZAKI_OCL` below).

Link-Time Interception

Two link-time variants are built per precision:

- `dgemm-blas.x` / `sgemm-blas.x` -- dynamically linked against BLAS
- `dgemm-wrap.x` / `sgemm-wrap.x` -- linked with `--wrap=` (GNU ld)
- `zgemm-wrap.x` / `cgemm-wrap.x` -- complex GEMM wrappers

Static (`libwrap.a`) and shared (`libwrap.so`) wrapper libraries are built by default. The shared library can intercept any application:

```
LD_PRELOAD=/path/to/libwrap.so ./application
```

The static library requires explicit `--wrap` flags:

```
-Wl,--wrap=dgemm_ -Wl,--wrap=sgemm_ -Wl,--wrap=zgemm_ -Wl,--wrap=cgemm_
```

Test scripts: `test-wrap.sh` exercises all variants, `test-check.sh` validates both schemes, `test-mhd.sh` tests MHD file-based input.

Test Driver

```
dgemm-wrap.x [A.mhd|M [B.mhd|N [K [TA [TB [ALPHA [BETA [LDA [LDB [LDC]]]]]]]]]]
```

Defaults: `M=N=K=257`, `TA=TB=0`, `ALPHA=BETA=1.0`. The first argument can be 0 followed by MHD filenames to load A and B matrices from files. The `zgemm-wrap.x` and `cgemm-wrap.x` drivers call ZGEMM and CGEMM.

Environment Variables

Scheme Selection

Variable	Default	Description
OZAKI	2	0=bypass (BLAS), 1=mantissa slicing, 2=CRT, 3=adaptive
OZAKI_COMPLEX	(auto)	Complex dispatch: 0=BLAS, 1=CPU, 2=GPU+fallback. Auto: 2 if on
OZAKI_N	(auto)	Slices (Sch.1: fp64=8, fp32=4) or primes (Sch.2: fp64=16, fp32=9)

OZAKI=3 (adaptive) starts with Scheme 1 on the first GPU call to learn the effective cutoff from preprocessing occupancy. Subsequent calls compare the Scheme-1 pair count (at the cached cutoff) against the Scheme-2 prime count and pick the cheaper path. On CPU, adaptive falls back to Scheme 2.

Accuracy

Variable	Default	Description
OZAKI_FLAGS	3	Sch.1 bitmask: 1=Triangular, 2=Symmetrize, 0=full S^2 square
OZAKI_TRIM	0	Levels to trim (0=exact). ~7 bits/level (Sch.1), ~4 bits (Sch.2)
OZAKI_I8	0	Sch.2: signed i8 residues (moduli $\leq$ 128) instead of u8
OZAKI_GROUPS	0	Sch.2: K-grouping (0/1=off). Consecutive K panels, one reconstr.
OZAKI_MAXK	32768	Max K per preprocessing pass (0=full K in one pass)
OZAKI_THRESHOLD	12	Intensity threshold. Bypass when $flops/(bytes*thr)<1$. 0=always

GPU Path (requires LIBXSTREAM)

Variable	Default	Description
OZAKI_OCL	0	Enable OpenCL/GPU path (0=off, 1=on)
OZAKI_TM	(auto)	GPU output tile height (multiple of 8)
OZAKI_TN	(auto)	GPU output tile width (multiple of 16)

GPU-specific kernel tuning variables (OZAKI_RTM, OZAKI_RTN, OZAKI_WG, OZAKI_SG, OZAKI_KU, OZAKI_RC, OZAKI_PB, OZAKI_HIER, OZAKI_PREFETCH, OZAKI_SCALAR_ACC, OZAKI_DEVPOOL, OZAKI_CACHE) are documented in the LIBXSTREAM Ozaki README.

Monitoring and Diagnostics

Variable	Default	Description
OZAKI_VERBOSE	0	0=silent, 1=stats at exit, N=print every Nth GEMM call
OZAKI_STAT	0	Track C-matrix (0), A-representation (1), or B-representation (2)
OZAKI_DUMP	0	Dump A/B as MHD files at the given call-count (0=off)
OZAKI_EPS	inf	Dump A/B when epsilon error exceeds threshold
OZAKI_RSQ	0	Dump A/B when RSQ drops below threshold (updated after dump)
OZAKI_EXIT	1	Exit on accuracy violation after dump. 0=continue
OZAKI_PROFILE	0	Profile: 0=off, 1=all, 2=kernel, 3=preprocess A, 4=preprocess B

Benchmark

Variable	Default	Description
CHECK	0	Validate vs BLAS: 0=off, negative=auto-threshold, positive=custom
NREPEAT	3	Number of GEMM calls (first call is warmup when >1)
EVIL	0	Adversarial exponent-span test (see below)
OZAKI_DECAY	0	Decay diagnostic + K-permutation (0-4, see below)

Adversarial Input (EVIL) The EVIL variable initializes A and B with controlled exponent structure for stress-testing accuracy and adaptive slice reduction. The magnitude sets the exponent span in bits; the sign selects the distribution:

EVIL=N (N>0) Per-column. Column j of A is scaled by $2^{(N*j/(ncols-1))}$, column j of B by the inverse. Product A*B is well-conditioned. Uniform exponents within each column.

EVIL=-N (N>0) Per-element. Each element gets a pseudorandom exponent in [0,N] via coprime shuffle, with opposite sign for B. Every row of A spans the full exponent range -- worst case for row-wise alignment and adaptive cutoff.

EVIL=0 Default shuffle mode (no exponent structure).

The per-column mode (EVIL>0) matches the NVIDIA emulation grading test (diagonal scaling with D and D⁻¹). The per-element mode (EVIL<0) is adversarial for the adaptive slice-pair reduction: it forces all slices to be populated in every row.

Decay Diagnostic and K-Permutation (OZAKI_DECAY) Controls K-dimension row reordering for smoothness and the forward-difference decay diagnostic. Values map to libxs_sort_t:

Value	Behavior
0	Off (default). No permutation, no diagnostic.
1	Identity permutation. Measure decay on original ordering.
2	Sort B rows by L1-norm, apply same K-permutation to A.
3	Sort B rows by mean value, apply same K-permutation to A.
4	Greedy nearest-neighbor chain on B rows (O(K ² *N)).

Values 2-4 compute a K-permutation from B (via `libxs_sort_smooth`) before Ozaki-1 slicing. Both A and B are sliced using the permuted K-order. Since $C = A*B = (A*P^T)*(P*B)$ for any permutation matrix P, the result is mathematically identical -- only the int8 digit structure along K changes.

When nonzero, reports the forward-difference decay of int8 slice buffers along K, M, and N axes (first K-group only, single- threaded). Uses `libxs_fprint` per-axis mode internally. Output goes to `stderr`:

OZ1[MxNxK] Delta-K: d1=... d2=... ... OZ1[MxNxK] Delta-M: d1=... d2=... ... OZ1[MxNxK] Delta-N: d1=... d2=...
...

Decaying values indicate exploitable smoothness; growing values (~2x per order) indicate unstructured data where data-independent schemes (Ozaki-1, Ozaki-2) are well-matched.

Profiling

The `OZAKI_PROFILE` variable enables per-GEMM timing collected into a histogram reported at program exit:

OZAKI PROF: 850 DP-GFLOPS/s (17.0 INT8-TOPS/s, 20x)

Profile modes select which phase is measured:

Mode	CPU	GPU
1/negative	All phases (preprocess+dot)	All profiled kernels
2	Kernel only (dot products)	Dotprod kernel only
3	Preprocessing (A+B)	Preprocess A kernel
4	Preprocessing (A+B)	Preprocess B kernel

On the CPU, modes 3 and 4 are equivalent (A and B preprocessing is interleaved across OpenMP threads).

Example

```
./dgemm-wrap.x 256 # CRT scheme (default)
OZAKI=1 ./dgemm-wrap.x 256 # mantissa slicing
OZAKI_TRIM=4 OZAKI=1 ./dgemm-wrap.x 256 # drop 4 least significant diagonals
OZAKI=2 OZAKI_GROUPS=4 ./dgemm-wrap.x 4096 # CRT with K-grouping
OZAKI=3 ./dgemm-wrap.x 4096 # adaptive scheme selection
EVIL=512 ./dgemm-wrap.x 1024 # accuracy grading (wide exponent span)
EVIL=1 OZAKI_PROFILE=1 ./dgemm-wrap.x 1024 # narrow span (shows pair savings)
EVIL=-52 ./dgemm-wrap.x 1024 # per-element span (worst for cutoff)
OZAKI_DECAY=1 OZAKI=1 ./dgemm-wrap.x 256 # decay diagnostic (no reorder)
OZAKI_DECAY=2 OZAKI=1 ./dgemm-wrap.x 256 # sort by norm + decay diagnostic
OZAKI_DECAY=4 OZAKI=1 ./dgemm-wrap.x 256 # greedy NN + decay diagnostic
```

Predict Samples

Seven executables demonstrating fingerprint-guided prediction:

- **predict_params** -- Parameter prediction from structured CSV (GPU kernel tuning, configuration databases).
- **predict_sunspots** -- Timeseries forecasting via sliding-window kNN (monthly sunspot numbers, 1749-present).
- **predict_earthquakes** -- Spatial prediction of earthquake magnitude from location and depth (USGS catalog).
- **predict_discharge** -- River discharge forecasting via sliding-window kNN with day-of-year seasonality (USGS NWIS daily streamflow).
- **predict_soi** -- Southern Oscillation Index prediction from anti-correlated Tahiti/Darwin sea level pressure using SPREAD decomposition (sum/diff modes).
- **predict_stock** -- Paired-stock timeseries prediction from CSV using SPREAD decomposition on two correlated price series.
- **predict_crystal** -- Crystal system classification from composition features (AFLOW ICSD, 7 classes, 60K entries).

Build

```
make
```

Or from the LIBXS root:

```
make GNU=1 samples/predict
```

predict_params

Train a prediction model from a CSV file and save it for later use. Reports validation quality on a held-out subset.

```
./predict_params.x [fraction] [auto|cat|compress[Q]|interp|rf|hknn] [-N] <csvfile> [modelfile] [confidence-prefix]
```

```
fraction    Validation split 0..1 for quality report (default: 0.8).
auto        Auto-detect mode per output (default).
cat         Force categorical (kNN) for all outputs.
compress    Drop redundant entries (Q: threshold, default 0.9).
interp      Force interpolation for all outputs.
rf          Random Forest classification.
hknn        Hierarchical kNN (Fisher-guided partition).
-N          Max polynomial order (default: 0 = auto).
csvfile     Delimited text file.
modelfile   Output path for the binary model.
confidence-prefix  Optional prefix for confidence map MHD files.
```

```
./predict_params.x ../../samples/smm/params/tune_multiply_PVC.csv
./predict_params.x 0.8 hknn tune_multiply_PVC.csv model.bin
```

predict_sunspots

Timeseries forecasting using sliding-window nearest-neighbor prediction.

```
./predict_sunspots.x <csvfile> [train_fraction] [compress[Q]] [hknn|rf]
```

```
./predict_sunspots.x predict_sunspots.csv 0.8
```

Data Source Monthly mean total sunspot number from SILSO (World Data Center, Royal Observatory of Belgium). Semicolon-delimited: year, month, decimal__year, sunspot__number.

predict_earthquakes

Predict earthquake magnitude from geographic location and depth.

```
./predict_earthquakes.x <usgs_csv> [train_fraction] [compress[Q]] [hknn|rf]
```

```
./predict_earthquakes.x predict_earthquakes.csv
```

Data Source USGS Earthquake Hazards Program (public domain). Comma-delimited: time, latitude, longitude, depth, mag, ...

predict_discharge

River discharge forecasting with day-of-year seasonality and log-transform on outputs for heavy-tailed data.

```
./predict_discharge.x <discharge_tsv> [train_fraction] [compress[Q]] [hknn|rf]
```

```
./predict_discharge.x predict_discharge.tsv
```

Data Source USGS National Water Information System (public domain). Colorado River at Lees Ferry, site 09380000. Tab-delimited RDB format.

predict_soi

Southern Oscillation Index prediction from anti-correlated sea level pressure at Tahiti and Darwin using SPREAD decomposition.

```
./predict_soi.x <tahiti_file> <darwin_file> [train_fraction] [compress[Q]] [hknn|rf]
```

```
./predict_soi.x predict_soi_tahiti.dat predict_soi_darwin.dat
```

Data Source NOAA Climate Prediction Center (public domain). Fixed-width monthly sea level pressure (mb above 1000 mb).

predict_stock

Multi-stock timeseries prediction with auto-differencing and PCA/SPREAD decomposition.

```
./predict_stock.x <csv_file> [columns] [train_fraction] [compress[Q]] [hknn|rf]
```

columns Comma-separated 0-based column indices (default: 1,2).

```
./predict_stock.x stocks.csv 1,2,3
```

predict_crystal

Crystal system prediction (7-class classification) from chemical composition features.

```
./predict_crystal.x <crystal_csv> [train_fraction] [order] [nclusters] [compress[Q]] [fisher|hknn|setdiff|rf|none]

fisher      Fisher discriminant feature weighting.
hknn        Hierarchical kNN (Gini-guided partition).
setdiff     Setdiff feature selection.
rf          Random Forest classification (default).
none        Raw kNN without feature processing.

./predict_crystal.x predict_crystal.csv
```

Data Source AFLOW ICSD catalog (free for academic use). 60,386 entries with Magpie-style composition features (37 features). Crystal systems: triclinic(1), monoclinic(2), orthorhombic(3), tetragonal(4), trigonal(5), hexagonal(6), cubic(7).

Registry Microbenchmark

Benchmarks the performance of the LIBXS registry (key-value store) dispatch path under different access patterns and concurrency levels.

Programs

```
./registry.x [total [nrepeat [nthreads]]]
```

Argument	Default	Description
total	10000	Number of unique keys to register (min 2)
nrepeat	10	Repeat iterations for lookup phases
nthreads	max available	Number of OpenMP threads

Measurements:

- Duration to register (insert) all keys into the registry.
- Cold lookup with shuffled access pattern (defeats thread-local cache).
- Cached lookup with sequential/repeating pattern (hits thread-local cache).
- Multi-threaded parallel reads across all threads.
- Contended parallel writes (each thread registers its own key range).
- Mixed read/write: one writer thread while remaining threads read.

The multi-threaded benchmarks require at least 2 threads and are skipped when running single-threaded.

./registryf.x

Fortran variant with hardcoded parameters (10000 keys, 10 repeats). Measures registration, cold lookup, and cached lookup.

Scaling Behavior

Read-only accesses stay roughly constant in per-op duration due to the thread-local cache. Write accesses are serialized and duration scales with the number of threads.

Rosetta

Demonstrates hierarchical type discovery on opaque binary data. A table of harmonic-series values is encoded as a flat byte blob with no metadata. The analysis rediscovers the element type, record stride, and per-field structure -- recovering mathematical content from anonymous bytes.

What It Shows

The sample proceeds through the levels described in the algebra paper (Section "Hierarchical Type Discovery"):

Step	Tool used	What it finds
Flat probe	libxs_fprint (UNKNOWN)	inconclusive (expected)
Stride sweep	libxs_fprint per-column	f64, stride=6
Shuffle test	libxs_shuffle + fprint	order matters (3-4x)
Field analysis	libxs_fprint per-field	decay rates per column
Sort test	libxs_sort_smooth (GREEDY)	already optimally ordered
Verification	libxs_setdiff_min	0 unmatched, tol=0

Key observations from the output:

- The flat 1-D probe fails because interleaved fields of different scales ($1/n$ mixed with n mixed with $\ln(n)$) look like noise when read sequentially. This motivates the stride sweep.
- The stride sweep requires ALL columns at a candidate width to have decay < 1 before accepting it, which eliminates false positives from partial correlations at wrong strides.
- Shuffle stability confirms that the record order carries genuine structure (decay increases $\sim 4x$ after coprime permutation).
- Per-field analysis reveals:
 - [0] decay=0 -- perfect ramp (sequential index)
 - [1] decay ~ 0.63 -- $1/n$ (decaying but not ultra-smooth)
 - [2] decay ~ 0.20 -- H_n (partial sums, very smooth)
 - [5] decay ~ 0.29 -- converges to Euler-Mascheroni gamma
- GREEDY sort confirms the data is already optimally ordered (the natural 1.64 sequence is the smoothest permutation).

The Data

For $n = 1, 2, \dots, 64$ the table stores six f64 fields per record:

[0]	n	integer index
[1]	1/n	harmonic term
[2]	H_n	partial sum of the harmonic series
[3]	ln(n)	natural logarithm
[4]	H_n - ln(n)	difference (converges to gamma from above)
[5]	H_n - ln(n) - 1/(2n)	better gamma estimate

Total: 64 records x 6 fields = 384 doubles = 3072 bytes.

Build

```
cd samples/rosetta
make GNU=1
```

Usage

```
./rosetta.x
```

No arguments. The sample is self-contained: it generates the data, encodes it as a flat blob, runs the full analysis, and prints the results.

To collect machine-readable artifacts, set ROSETTA_OUTDIR to an existing directory:

```
mkdir -p results/rosetta
ROSETTA_OUTDIR=results/rosetta ./rosetta.x
```

This writes summary.csv, rosetta_original.mhd, rosetta_shuffled.mhd, rosetta_fields.mhd, and rosetta_fields_shuffled.mhd. If LIBXS_MHD_PNG=1 is also set, the MHD writer emits PNG files with the same basenames. The fields images are normalized per field, making the ordered and shuffled record traces easy to compare visually. The artifact mode is optional and leaves the interactive sample output unchanged.

Example Output

```
ROSETTA: Recovering structure from opaque bytes
Ground truth (hidden from the analysis):
  64 records x 6 fields = 384 doubles (3072 bytes)
  Data: harmonic series H_n and derived quantities.
  Field [5] converges to Euler-Mascheroni gamma.
Level 0: Flat 1-D probe (all bytes as one sequence)
  Input: 3072 opaque bytes
  No type has decay < 1 -- flat stream is not smooth.
  This is expected: interleaved fields of different
  scales look like noise when read sequentially.
Stride sweep: discovering record layout
  Best type:    f64
  Best stride:  6 elements (48 bytes per record)
  Records:     64
  Avg decay:    0.288244
Shuffle stability (record-level)
  Avg decay (original): 0.288244
  Avg decay (shuffled): 1.103839
  Ratio:        3.8x
-> Record order carries structure.
Per-field decay analysis
[0] n          decay=0.000000
[1] 1/n        decay=0.631512
[2] H_n        decay=0.203226
```

```
[3] ln(n)          decay=0.259621
[4] H_n-ln(n)      decay=0.342853
[5] gamma_est      decay=0.292251 (last=0.5771953203, gamma=0.5772156649)
Smoothest: [0] n (perfect ramp, decay=0)
-> This reveals a sequential index column.
Most interesting: [5] gamma_est -- converges to a constant.
GREEDY sort test (64 rows x 6 cols)
Data is already optimally ordered for row smoothness.
Verification: setdiff(original, blob)
Unmatched: 0, tolerance: 0.00e+00
-> Byte-perfect recovery.
```

Why This Is Interesting

From 3072 anonymous bytes the framework discovers:

- 1. The element type (f64) -- not i32, not f32, not i64.
- 2. The record structure (6-field records of 48 bytes each).
- 3. A sequential index column (field [0], decay = 0).
- 4. A column converging to Euler-Mascheroni gamma (field [5]).
- 5. That the natural ordering is already optimal (no resorting).

No metadata, no format knowledge, no human guidance. The hierarchical composition -- stride sweep with all-columns-must-pass filtering, shuffle stability, per-field decay ranking -- is what makes this possible. Any single tool alone would either fail (flat probe) or produce ambiguous results (stride sweep without the all-columns requirement accepts wrong strides).

Memory Allocation (Microbenchmark)

Benchmarks pooled scratch memory allocation (libxs_malloc / libxs_free) against the standard C library malloc / free. The pool-based allocator recycles memory after an initial warm-up, avoiding repeated system calls in steady state. A Fortran variant compares libxs_malloc against Fortran ALLOCATE / DEALLOCATE.

Building

```
cd samples/scratch
make GNU=1
```

Produces scratch.x (C) and, if a Fortran compiler is found, scratchf.x.

Usage (C)

```
./scratch.x [ncycles [max_nactive [nthreads]]]
```

Argument	Default	Description
ncycles	100	Number of allocation/deallocation cycles
max_nactive	4	Max concurrent live allocations per cycle (max 24)
nthreads	1	OpenMP threads (clamped to omp_get_max_threads)

Environment Variables

Variable	Default	Description
CHECK	0	Non-zero: memset each allocation (validates writeable)

Compile-Time Knobs

Macro	Default	Description
MAX_MALLOC_MB	100	Upper bound on individual allocation size in MB
MAX_MALLOC_N	24	Array size for random-number table and alloc slots

default: 100 cycles, up to 4 active allocations, 1 thread
./scratch.x

heavy load: 500 cycles, up to 12 active allocations, 4 threads
./scratch.x 500 12 4

validate pages are writable
CHECK=1 ./scratch.x 43 8 4

Usage (Fortran)

./scratchf.x [mbytes [nrepeat]]

Argument	Default	Description
mbytes	100	Allocation size in MB
nrepeat	20	Number of repetitions

Single-threaded benchmark comparing libxs_malloc / libxs_free against Fortran ALLOCATE / DEALLOCATE. Reports per-iteration times (ms) and an average speedup ratio.

What Is Measured

Each cycle draws a random number of allocations (1 to max_nactive) with random sizes (1 to MAX_MALLOC_MB MB each). The cycle allocates all buffers, optionally touches them (CHECK), then frees them.

Both paths use the same randomized sequence. An untimed warm-up cycle runs before the measured loops. Reported metrics:

- calls/s (kHz) -- allocation+free throughput for each allocator
- Scratch size -- high-water mark of the pool (MB)
- Malloc size -- peak aggregate allocation per cycle (MB)
- Scratch Speedup -- ratio of pool throughput to stdlib throughput
- Fair -- size-adjusted speedup: (malloc_size / scratch_size) * speedup

Setdiff

Runs small deterministic experiments for `libxs_setdiff` and `libxs_setdiff_min`. The sample writes CSV files that are useful for paper figures and for quick local proxy runs.

Build

```
cd samples/setdiff
make GNU=1
```

Usage

```
./setdiff.x [outdir [sizes [reps [land_n [land_steps]]]]]
```

Positional	Default	Description
outdir	results/setdiff	Directory for CSV output
sizes	128 1024 8192 65536	Vector sizes for tolerance and scaling runs
reps	10	Repetitions per scaling size
land_n	32	Tolerance-landscape vector length
land_steps	40	Number of tolerance grid steps

Output

- `summary.csv` -- order-independence and duplicate-consumption cases
- `landscape.csv` -- sampled tolerance landscape
- `tolerance.csv` -- `libxs_setdiff_min` compared with bisection
- `scaling.csv` -- fixed-tolerance and automatic-tolerance timings
- `complex.csv` -- complex-valued validation case

The sample uses a fixed pseudo-random seed and writes into a deterministic output directory by default.

Shuffle

Benchmarks three shuffling strategies and compares their throughput:

Label	Method
RNG-shuffle	Fisher-Yates via <code>libxs_rng_u32</code> (reference baseline)
DS1-shuffle	<code>libxs_shuffle</code> -- deterministic in-place (coprime stride)
DS2-shuffle	<code>libxs_shuffle2</code> -- deterministic out-of-place (src to dst)

Each iteration works on an array of unsigned integers initialized to 0..n-1. On the final iteration the shuffled result can optionally be written as an MHD image file or analyzed for randomness quality.

Build

```
cd samples/shuffle
make GNU=1
```

Usage

```
./shuffle.x [nelems [elemsize [niters [repeat]]]]
```

Positional	Default	Description
nelems	64 MB / elemsize	Number of elements
elemsize	4	Element size in bytes (1, 2, 4, or 8)
niters	1	Shuffle passes per timed measurement
repeat	3	Timed iterations (first is warmup)

Environment Variables

Variable	Default	Description
RANDOM	0	Non-zero: count inversions and report randomness (rand=%)
SPLIT	1	Partitioning depth for the STATS metrics
STATS	0	Non-zero: print partition imbalance (imb) and distance (dst)

```
## Default run (64 MB of 4-byte elements, 3 repeats)
./shuffle.x
```

```
## 1M elements of 8 bytes, 4 shuffle passes, 5 repeats
./shuffle.x 1000000 8 4 5
```

```
## Enable randomness quality metric
RANDOM=1 ./shuffle.x 10000 4 1 3
```

```
## Enable partition statistics with split depth 2
STATS=1 SPLIT=2 ./shuffle.x
```

What Is Measured

Reported bandwidth is $2 * \text{data-size} / \text{time}$ (in MB/s). The factor of two accounts for reading and writing each element during the shuffle. The first iteration is excluded from the average.

Optional quality metrics:

- `rand` -- Enabled by `RANDOM=1`. Inversions are counted via merge-sort and compared against the expected count for a random permutation $(n*(n-1)/4)$ to give a percentage.
- `dst` -- Manhattan distance of the element sum from the expected uniform value, split into hierarchical partitions (SPLIT).
- `imb` -- Partition imbalance, measuring how unevenly element sums are distributed across sub-partitions (SPLIT).

MHD Image Output When `RANDOM` is not set and the element size is 1, 2, 4, or 8 bytes, the final shuffled array is written as an MHD image per method (`shuffle_rng.mhd`, `shuffle_ds1.mhd`, `shuffle_ds2.mhd`). The images are shaped close to square ($\text{isqrt}(n) \times n/\text{isqrt}(n)$) and converted to unsigned 8-bit via modulus.

Compile-Time Knobs

Variable	Default	Description
OMP	0	Enable OpenMP (not used by this sample)
SYM	1	Include debug symbols (-g)
BUBBLE_SORT	undefined	Define (-DBUBBLE_SORT) for $O(n^2)$ inversion counter

Stratify

Samples for folding higher-dimensional data into lower-dimensional layouts using libxs space-filling-curve stratification. The folder is intended to hold related data-preparation and framework-adaptor examples that share the same mapping primitive.

The current dense 3D sample folds a regular voxel grid into a 2D sheet. It is intentionally a data-preparation example: the mapping can be computed once for a fixed detector or simulation grid, then reused by a Python, PyTorch, TensorFlow, or file-based pipeline.

Sample names include the subject after the folder prefix, following the style used by other multi-sample folders such as `samples/predict`: regular dense 3D-grid tools are named `stratify_dense3d.py` and `stratify_dense3d_metrics.py`. Future medical-imaging or detector-specific adapters can use the same pattern, for example `stratify_medmnist3d.py` or `stratify_mhd.c` if a dependency-light C reader is useful.

Python is the preferred language for dataset adapters because formats such as HDF5, NPZ, NIfTI, and framework dataloaders are easiest to keep optional there. C samples remain appropriate for small dependency-free demonstrations, performance checks, or formats already supported in C, such as MHD volume data.

The transform is not a stack of slices. Each source voxel coordinate is encoded as a 3D Hilbert or Morton key and decoded as a 2D coordinate:

3D coordinate -> 3D curve rank -> inverse 2D curve coordinate

This preserves the deterministic curve order while giving downstream code a 2D tensor layout that can be consumed by optimized 2D convolution kernels.

The samples distinguish the curve order from the 2D frame. The default `compact` frame streams voxels in curve-rank order into a dense near-square sheet, avoiding unused cells when an exact factor pair is available. The `canonical` frame decodes the finite 3D curve rank through the corresponding 2D curve, preserving the canonical destination curve coordinate at the cost of possible empty cells.

Usage

Build libxs first so that lib/libxs.so exists:

```
cd ../../
make GNU=1 PEDANTIC=2
cd samples/stratify
make
```

Optional HDF5 and MedMNIST3D adapters need numpy and h5py. In an isolated environment:

```
python3 -m venv .venv
. .venv/bin/activate
python -m pip install -r requirements.txt
```

The default target runs the dense 3D sample. It can also be invoked explicitly:

```
make dense3d
make dense3d FRAME=canonical
```

The older make volume spelling remains as a compatibility alias.

Direct invocation:

```
python3 stratify_dense3d.py --shape 8 16 16 --curve hilbert \
--frame compact --out sheet.pgm
python3 stratify_dense3d.py --shape 8 16 16 --curve morton \
--frame canonical --map-csv map.csv
```

HDF5 input is optional and requires h5py and numpy. The input dataset may be a single D,H,W volume, a batched N,D,H,W dataset, a channelled N,C,D,H,W or N,D,H,W,C dataset, or a flattened N,V dataset when --hdf5-reshape D H W is supplied. The default auto layout recognizes the 3DGAN Caffe sample layout N,C,D,H,W:

```
python3 stratify_dense3d.py --hdf5 /tmp/3Dgan-risk-audit/caffe/train.h5 \
--hdf5-dataset ECAL --hdf5-event 0 --hdf5-channel 0 \
--curve hilbert --frame compact --out sheet.pgm --out-hdf5 sheet.h5
```

The Makefile exposes the same path without making the default target depend on external data:

```
make hdf5 HDF5_FILE=/tmp/3Dgan-risk-audit/caffe/train.h5 FRAME=compact
```

For flattened HDF5 files, pass the dataset name, layout, and target 3D shape:

```
make hdf5 HDF5_FILE=dataset_2_1.hdf5 HDF5_DATASET=showers \
HDF5_LAYOUT=flat HDF5_RESHAPE="45 16 9"
```

Public calorimeter HDF5 datasets with a documented download path are available from the CaloChallenge project. The challenge HDF5 files contain incident_energies and flattened showers datasets, so use --hdf5-dataset showers and --hdf5-reshape D H W with the geometry stated for the selected dataset. Useful starting points are:

Source	HDF5 format
CaloChallenge homepage	Dataset overview, geometry, and evaluation format.
Dataset 1 DOI	ATLAS photons/pions, flattened showers with dataset-
Dataset 2 DOI	Electrons, reshape showers to 45 16 9.
Dataset 3 DOI	Higher-granularity electrons, reshape showers to 45 50 1
Legacy dataset 1 photons, dataset 1 pions, dataset 2, dataset 3	Submitted model outputs and samples for reproducing

Arguments:

Option	Description
--shape D H W	Source volume shape. Default: 8 16 16.
--curve	hilbert or morton. Default: hilbert.
--frame	compact or canonical. Default: compact.
--libxs	Explicit path to the libxs shared library.
--hdf5	Read the source volume from an HDF5 file.

Option	Description
<code>--hdf5-dataset</code>	Dataset to read from the HDF5 file. Default: <code>ECAL</code> .
<code>--hdf5-layout</code>	<code>auto</code> , <code>dhw</code> , <code>ndhw</code> , <code>ncdhw</code> , <code>ndhwc</code> , or <code>flat</code> .
<code>--hdf5-reshape D H W</code>	Reshape selected flat HDF5 data to a 3D volume.
<code>--hdf5-event</code>	Event index for batched HDF5 layouts. Default: <code>0</code> .
<code>--hdf5-channel</code>	Channel index for channelled HDF5 layouts. Default: <code>0</code> .
<code>--out</code>	Write an 8-bit PGM image of the stratified sheet.
<code>--map-csv</code>	Write <code>src,z,y,x,dst,v,u</code> mapping rows.
<code>--out-hdf5</code>	Write <code>sheet</code> and <code>map</code> datasets to an HDF5 file.

The script prints source shape, resulting sheet shape, density, mapping time, and deposition sums before and after stratification.

MedMNIST3D

The `stratify_medmnist3d.py` sample reads one standardized MedMNIST3D volume from an `.npz` file and applies the same 3D-to-2D stratification primitive. It is a dataset adapter rather than a training script, which keeps the dependency surface small: only `numpy` is required to parse the MedMNIST file. The sample does not require `PyTorch` or the `medmnist` Python package unless you use those tools separately to download the data.

The official MedMNIST distribution is available from the project page and Zenodo: MedMNIST and Zenodo DOI. The 3D subsets are `organmnist3d`, `nodulemnist3d`, `adrenalmnist3d`, `fracturemnist3d`, `vesselmnist3d`, and `synapsemnist3d`.

Direct invocation with an explicit NPZ file:

```
python3 stratify_medmnist3d.py --npz ~/.medmnist/organmnist3d.npz \
  --split train --index 0 --curve hilbert --frame compact \
  --out stratified_medmnist3d.pgm \
  --map-csv stratified_medmnist3d.csv \
  --label-csv stratified_medmnist3d_label.csv
```

The Makefile exposes the same path without making the default target depend on external data:

```
make medmnist3d MEDMNIST3D_NPZ=~/.medmnist/organmnist3d.npz FRAME=compact
```

The same NPZ input can be used with the metrics script to report invariants, locality distortion, and LIBXS Foepl fingerprint distances for a selected volume:

```
make medmnist3d-metrics MEDMNIST3D_NPZ=~/.medmnist/organmnist3d.npz \
  MEDMNIST3D_SPLIT=test MEDMNIST3D_INDEX=0 FRAME=canonical
```

If `MEDMNIST3D_NPZ` is omitted, the script looks for a dataset under `MEDMNIST3D_ROOT` using `MEDMNIST3D_FLAG` and `MEDMNIST3D_SIZE`:

```
make medmnist3d MEDMNIST3D_ROOT=~/.medmnist MEDMNIST3D_FLAG=nodulemnist3d
```

This gives us a lightweight benchmark bridge: native MedMNIST3D models can use the original `N,D,H,W` arrays, while 2D models can consume the generated stratified sheet and keep the label from `stratified_medmnist3d_label.csv`.

Metrics

The `stratify_dense3d_metrics.py` script reports invariants and locality distortion for the same synthetic and HDF5 inputs:

```
make metrics
python3 stratify_dense3d_metrics.py --hdf5 /tmp/3Dgan-risk-audit/caffe/train.h5 \
  --hdf5-dataset ECAL --curve hilbert
python3 stratify_dense3d_metrics.py --hdf5 dataset_2_1.hdf5 --hdf5-dataset showers \
  --hdf5-layout flat --hdf5-reshape 45 16 9 --curve morton
```

The invariant metrics check that stratification is a lossless layout transform: the total energy, reconstructed voxel values, and per-axis energy profiles match after applying the inverse map. The distortion metrics measure what changes for a convolutional model: how far source 3D grid neighbors move apart in the 2D sheet, and how often adjacent sheet cells correspond to adjacent source voxels.

The script also reports LIBXS Foeppl fingerprint metrics. These are compact multi-order L2 descriptors of value and finite-difference structure. The `fprint.source_reconstructed.diff` value should be zero for a correct lossless round trip, while `fprint.source_sheet.diff` summarizes how different the stratified 2D sheet looks as a structured field. This is useful as an additional quality marker in a paper, but it should be interpreted as a representation roughness/shape descriptor, not as a detector-physics fidelity score.

These metrics do not prove physics fidelity. They expose the representation tradeoff: scalar voxel values are preserved exactly, but the neighborhood graph seen by a 2D convolution is different from the original 3D grid.

Geometry

The current sample treats the source as a regular `D,H,W` index grid. This is enough for dense 3DGAN-style arrays and for CaloChallenge datasets after their flattened `showers` arrays are reshaped with the documented dimensions.

Supporting other detector geometries should be data-driven rather than encoded as a list of known experiments. A general geometry adapter needs one of the following descriptions:

Geometry input	Role
Integer logical coordinates per voxel, for example <code>z,alpha,r</code> or <code>z,y,x</code>	Direct input to Hilbert/Morton stratification.
Floating physical coordinates per voxel	Quantized or ranked into integer coordinates before
Optional adjacency edges between voxels	Used by metrics to measure physical-neighbor disto
Optional voxel weights or cell volumes	Used by downstream physics metrics when bins ha

With such a coordinate or adjacency table, the stratification primitive does not need hard-coded knowledge of a detector. The values remain losslessly permuted; the geometry file defines which neighborhoods and distances should be considered meaningful when judging whether the 2D layout is a good substitute for native 3D convolutions.

Framework Use

For a fixed geometry, the CSV mapping or the in-memory index arrays can be cached and reused. A framework integration does not need a custom operator at first: use the mapping during dataset preparation, or use native gather and scatter operations in the data loader. The training and inference hot path can then use ordinary 2D convolution models.

The intended comparison is:

```
3D volume -> 3D convolution baseline
3D volume -> naive 2D slicing or flattening -> 2D convolution control
3D volume -> Hilbert/Morton stratified sheet -> 2D convolution candidate
```

This separates the amortized layout cost from the model throughput and quality measurements.

Related Samples

Additional self-contained integrations can live in this folder when they reuse the same stratification primitive. Framework-specific adapters should first be kept as external patches or scripts until their data, dependency versions, and runtime setup are reproducible. For example, a 3DGAN adapter should only move into this directory once it can clone the reference model, prepare stratified 2D calorimeter sheets from documented HDF5 showers, and compare them against the original 3D convolutional baseline without private paths or fragile legacy packages.

Synchronization Primitives

Micro-benchmark for the lock implementations provided by LIBXS (`libxs_sync.h`). Measures single-thread latency (uncontended acquire/release) and multi-thread throughput (mixed read/write workload) for every compiled lock kind:

Lock kind	Description
LIBXS_LOCK_DEFAULT	Compile-time default (typically atomic)
LIBXS_LOCK_SPINLOCK	OS-native or CAS-based spin lock (if available)
LIBXS_LOCK_MUTEX	OS-native mutex / <code>pthread_mutex_t</code> (if available)
LIBXS_LOCK_RWLOCK	Reader/writer lock / <code>pthread_rwlock_t</code>

The default lock is always benchmarked. The remaining kinds are conditionally compiled depending on platform support.

Build

```
cd samples/sync
make GNU=1
```

OpenMP is enabled by default (`OMP=1`) for multi-threaded tests.

Run

```
./sync.x [nthreads] [wratio%] [work_r] [work_w] [nlat] [ntpt]
```

Argument	Default	Description
nthreads	all available	Number of OpenMP threads
wratio%	5	Percentage of write operations (0-100)
work_r	100	Simulated work inside read-critical section (cy)
work_w	10 * work_r	Simulated work inside write-critical section
nlat	2000000	Iterations for latency measurement
ntpt	10000	Iterations per thread for throughput measurement

```
./sync.x 4 5 100 1000
```

```
LIBXS: default lock-kind "atomic" (Other)
```

```
Latency and throughput of "atomic" (default) for nthreads=4 wratio=5% ...
ro-latency: 11 ns (call/s 91 MHz, 33 cycles)
rw-latency: 11 ns (call/s 90 MHz, 33 cycles)
throughput: 0 us (call/s 9128 kHz, 328 cycles)
```

Measurement Details

- RO-latency: uncontended read-lock acquire/release pairs (4x unrolled), reported as nanoseconds per operation and TSC cycles.
- RW-latency: uncontended write-lock acquire/release pairs (4x unrolled).
- Throughput: all threads run a mixed read/write workload governed by `wratio%`. Simulated work inside the critical section is subtracted so only synchronization overhead is reported.

SYRK / SYR2K Sample

Demonstrates symmetric rank-k and rank-2k updates using the LIBXS dispatch-and-call model. The sample validates correctness against a plain Fortran reference and reports performance.

Operations

SYRK: $C := \alpha * A * A^T + \beta * C$ (lower triangle) SYR2K: $C := \alpha * (A * B^T + B * A^T) + \beta * C$ (upper triangle)

Only the specified triangle of C is written; the other triangle is left untouched.

Build

```
make
```

Requires a Fortran compiler (gfortran, ifort, or ifx). The Makefile picks up the compiler from the top-level Makefile.inc. MKL or BLAS linkage is enabled (BLAS=1) so that dispatch can use JIT-compiled kernels when available.

Run

```
./syrk.x [N [K [nrepeat [direct]]]]
```

Arguments (all optional, positional):

N	Matrix dimension of C (N x N).	Default: 64
K	Inner dimension (columns of A).	Default: N
nrepeat	Number of timed repetitions.	Default: 100
direct	Interface selection.	Default: 0
	0 = config (dispatch + call separately)	
	1 = direct (one-shot generic)	

Example Output

```
syrk(F): N=64 K=64 nrepeat=100

--- libxs_syrk (lower) ---
  max error (lower): 0.00000E+00

--- libxs_syr2k (upper) ---
  max error (upper): 0.00000E+00

--- SYRK performance ---
  time:      0.002 s (100 calls)
  perf:      28.4 GFLOPS/s
```

Notes

- The dispatch step (`libxs_syrk_dispatch` / `libxs_syr2k_dispatch`) returns a pointer to a registry-owned config. This pointer remains valid until `libxs_finalize` or the registry is destroyed. There is no need to release it manually.
- Internally, SYRK/SYR2K decompose into GEMM tiles on the diagonal and off-diagonal blocks. The dispatched GEMM kernel (MKL JIT, LIBXSMM, or fallback BLAS) handles the inner loop.
- Scratch memory for the temporary full-panel product is managed via a thread-local buffer that grows on demand and is freed at finalization.