Intro to Mir

tinyurl.com/muc16mir

Sebastian Wilzbach
seb@wilzbach
seb.wilzba.ch

@swilzbach

@wilzbach

Overview

- BLAS
- SIMD & @fastmath
- Mir
- Ndslice
- GLAS
- Hands on

BLAS (Basic Linear Algebra Subprograms)

- 1979: Fortran (reference)
- Well-known implementations
  - Intel Math Kernel Library (MKL)
  - ATLAS
  - OpenBLAS
  - Eigen

- In BLAS: (s|d|c|z)-operation, e.g. daxpy

BLAS

- Vector addition
- Scalar multiplication
- Dot products
- Linear combinations
- Matrix multiplication

BLAS kernels - Level 1 O(n)

- Vector operations
- `axpy` (αx + y)
- `dot` (x * y)
- `dotc` (x^T * y)
- `rot`: Givens rotation
- `nrm2`: Euclidean norm, `sqnrm2`: square of Euclidean norm
- `asum`: sum of absolute values
- `amax`: max abs. value, `imax`: index of max abs. value

http://www.netlib.org/lapack/explore-html/d9/d0e/group__level1.html

BLAS - Level 2 O(n²)

`gemv`: Generalized matrix-vector multiplication: α * A * x + β * y

`All:`

`gbmv, ger, bmv, spmv, spr, spr2, symv, syr, syr2, tbmv, tbsv, tpmv, tpsv, trsv`

http://www.netlib.org/lapack/explore-html/dd/d15/group__level2.html

BLAS - Level 3 $O(n^3)$

- General matrix multiplication (gemm) - $\alpha * A * B + \beta * C$

All:

`symm, syr2k, syrk, trmm, trsm`

http://www.netlib.org/lapack/explore-html/d1/d54/group__double__blas__level3.html

SIMD (Single instruction, multiple data)

- Streaming SIMD Extensions (SSE)
  - 128 bit registers: XMM0 - XMM7
  - SSE2 - SSE4
- AVX (Streaming SIMD Extensions)
  - 128 → 256
  - 3-operand instructions
  - AVX512
- https://dlang.org/spec/simd.html

@fastmath

- LDC >= 1.1.0-beta2
- https://github.com/ldc-developers/ldc/pull/1472
- Allows vector-optimizations
- Allows fused multiply-add (FMA)
- Applied per function
- Extensively used in Mir

Looking under the hood

Online:

- d.godbolt.org (GDC, LDC)
- ldc.acomirei.ru
- asm.dlang.org

Offline

- LDC: `--output-ll` (IR), `--output-s`
- DMD: `obj2asm` (included)

Mir

- Restart of “DLangScience”
- Started end of 2015
- Main developer: Ilya Yaroshenko (aka @9il)
- Contains
  - GLAS
  - Ndslice (dev version)
  - Sparse tensors
  - Random
  - Combinatorics, Sum
  - LDA

mir-cpuid

- In D: core.cpuid
- Cache sizes (L1, L2, L3)
- Threads / Cores per cache
- Page size
- AVX, SSE, …
- ...
- cpuid.unified

http://docs.cpuid.dlang.io/latest/index.html

ndslice

- Std.experimental.ndslice
- Merged in 2016-01-02 (2.070), #3397
- Multidimensional view
- Code coverage > 98%
- Stable in 1 or 2 releases

A slice

One dimensional array:

```
auto arr = new double[6]; // [0, 0, 0, 0, 0, 0]
```

View:

```
auto matrix = arr.sliced(2, 3);  // [0, 0, 0]
                                 // [0, 0, 0]
```

Internal representation

- `Slice!(N, T*)`
  - `size_t[N] lengths` — elements of the dim.
  - `sizediff_t[N] strides` — jump to next element in dim.
  - `T* ptr` — raw array/range

Internal representation: example

```
auto arr = 24.iota.sliced(2, 3, 4)
```

Lengths	`[2, 3, 4]`
Strides	`[12, 4, 1]`
Ptr	`&a[0]`

```
arr[1, 2, 3] = &a[0] + 12 * 1 + 4 * 2 + 1 * 3] = a[23]

arr[0, 2, 3] = &a[0] + 12 * 0 + 4 * 2 + 1 * 3] = a[11]
```

Internal representation: example

```
arr.transposed!(0, 2, 1)
```

Lengths	`[2, 4, 3]`
Strides	`[12, 1, 4]`
Ptr	`&a[0]`

```
arr[1, 2, 1] = &a[0] + 12 * 1 + 1 * 2 + 4 * 1] = a[18]

arr[1, 0, 1] = &a[0] + 12 * 1 + 1 * 0 + 4 * 1] = a[16]
```

Internal representation: example

```
arr.reversed!1
```

Lengths `[2, 3, 4]`
Strides `[12, -4, 1]`
Ptr `&a[8]` `strides[n] * (lengths[n] - 1)`

```
arr[1, 2, 1] = &a[8] + 12 * 1 - 4 * 2 + 1 * 1] = a[13]

arr[1, 0, 1] = &a[8] + 12 * 1 - 4 * 0 + 1 * 1] = a[21]
```

Row/column styles

```
arr = 6.iota.sliced(2, 3)
   [0, 1, 2]
   [3, 4, 5]
```

- Row major arr[0, 1] = 1
- Column major arr(0, 1) = 3

Selection

```
auto arr = 9.iota.sliced(3, 3)

          [0, 1, 2]
          [3, 4, 5]
          [6, 7, 8]

 -  arr.diagonal       [0, 4, 5]
 -  arr[0..2, 0..2]    [0, 1]
                       [3, 4]
```

Ndslice with allocations

- slice, makeSlice
- ndarray, makeNdarray

```
6.iota.sliced(2, 3).slice
```

```
import std.experimental.allocator           : dispose;
import std.experimental.allocator.mallocator : Mallocator;

alias Allocator = Mallocator.instance;
auto tup = makeSlice!int(Allocator, 2, 3);

assert(tup.array.length         == 6);
assert(tup.slice.elementsCount  == 6);
Allocator.dispose(tup.array);
```

Pack & unpack

- Only a view
- No calculation/overhead

	.shape	typeof()
`arr.shape`	`[2, 3]`	`Slice!(2LU, Result)`
`arr.pack!1`	`[2]`	`Slice!(1LU, Slice!(2LU, Result))`
`arr.slice.unpack`	`[2, 3]`	`Slice!(2LU, int*)`

ndslice.algorithm

- Partially merged (#4652)

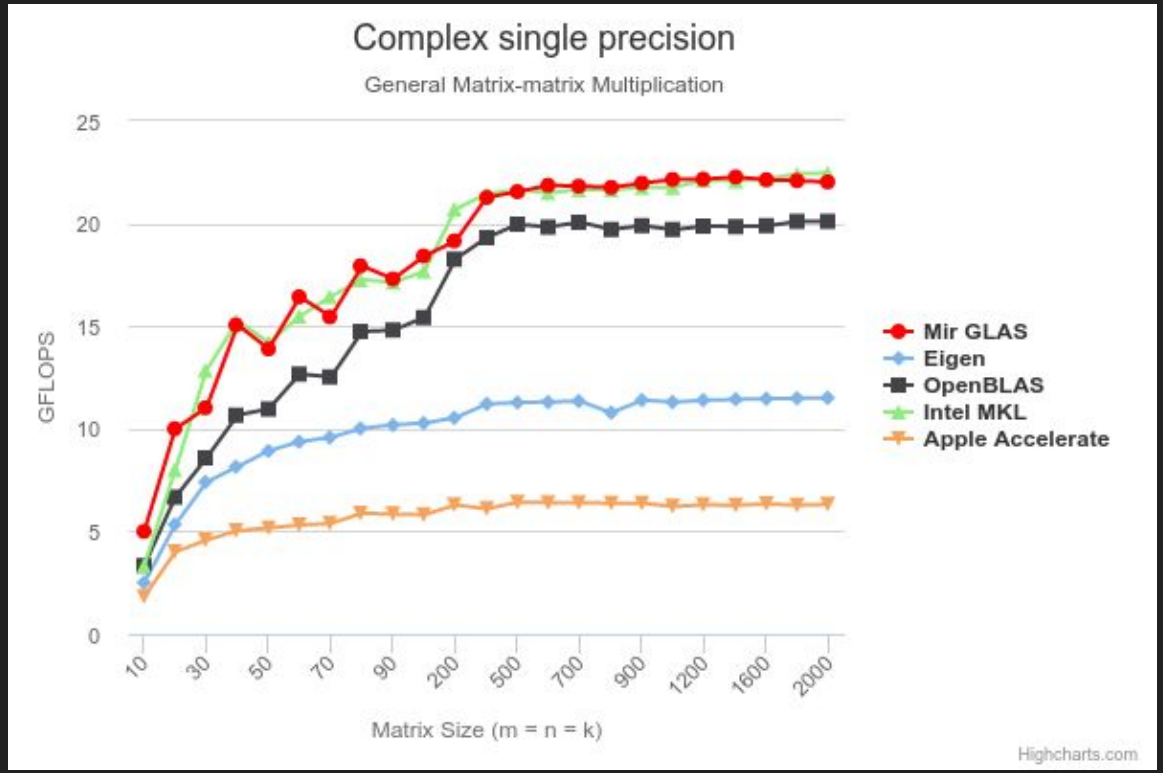
```
arr = 6.iota.sliced(2, 3)
      [0, 1, 2]
      [3, 4, 5]

arr.mapSlice!`a + 1`[1, 1]              // 5
arr.pack!1.mapSlice!`a.sum`             // [3, 12]
arr.transposed.pack!1.mapSlice!`a.sum`    // [3, 5, 7]
arr.pack!1.evertPack.mapSlice!`a.sum`     // [3, 5, 7]
```

GLAS (Generic Linear Algebra Subprograms)

- No assembly
- Completely in D
- Single, generic kernel
- Can be run without D runtime! (since last week)
- WIP
- Zero-cost transposition with `transposed`

Numerical age



https://github.com/libmir/mir/tree/master/benchmarks/glas

Floating point math

- FP math isn’t distributive
- CT math uses double (in DMD)
- Use “%a” to print the hexadecimal string

https://github.com/libmir/mir/wiki/Floating-point-issues

FP precision is an illusion

- Precision:
  https://gist.github.com/wilzbach/3d27d06b55821aa9795deb15d4d47679
  https://gist.github.com/wilzbach/ed5033e5be08b09dc181972a4f56fb7c
- 32 vs. 64:
  https://gist.github.com/wilzbach/afbc2fffd770bfa98c6d353904df687a
  https://gist.github.com/wilzbach/f2e5fc48e76c92fef7eb4f8b55b80e4b
- std.math != C API
  https://gist.github.com/wilzbach/ec7cc9464e4e8f97689637f23238d510
- Don’t trust real on Windows (issue 16344)

Optimizations tricks

- Avoid function calls
- Avoid allocation
- Pack your data → (“Bitpacking like a mad man”, Amaury Sechet)
- Use const / immutable
- For fast compile-time
  - Avoid template bloat
  - Avoid ctRegex

Optimization tricks

- Never use `^^` (prefer LLVM internals)
- `sgn` → `copysign`
- Instead of `a / b`
  - `enum one_div_two = 0.5`
  - `a * one_div_two`
- Prefer switch over if
- Switch with default: one jump less
- `goto` is your friend

Wanna learn more?

blog.mir.dlang.io

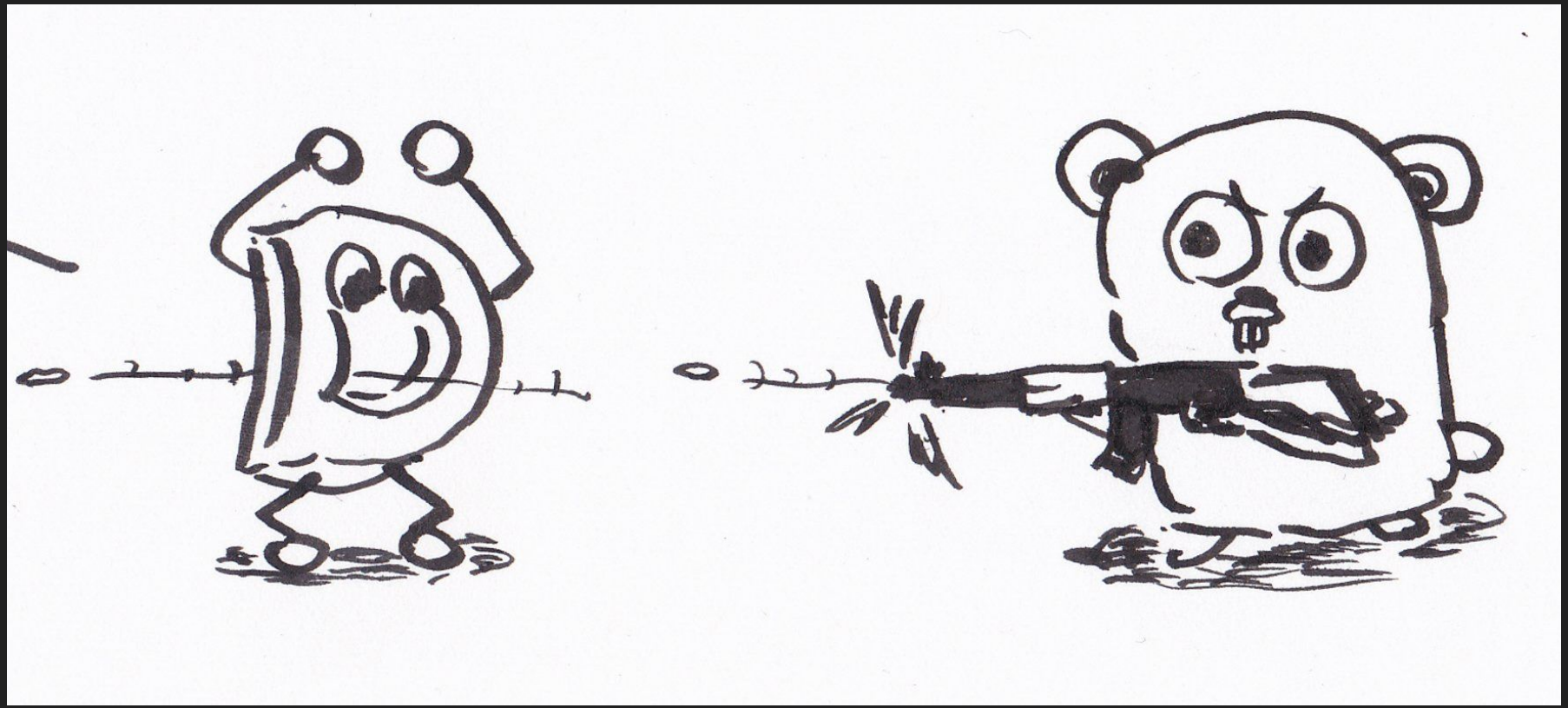
docs.mir.dlang.io

johanengelen.github.io

 @libmir

 @libmir

Let’s start



How far do you get with ndslice?

github.com/rougier/numpy-100