# **KIWAKU, CONTAINERS WELL MADE**

GENERATIVE PROGRAMMING FOR EFFICIENT HPC DATA STRUCTURES

April 8, 2022 REPRISES

Sylvain JOUBE

Joël FALCOU, Hadrien GRASLAND, David CHAMONT.







# Context

#### nD Arrays : a pervasive tool

- Main data structure in **numerical simulations**
- Expected to be **efficient**
- Expected to be **easy to use** yet **expressive**

### Problem already addressed many times :



# Context

#### nD Arrays : a pervasive tool

- Main data structure in **numerical simulations**
- Expected to be **efficient**
- Expected to be **easy to use** yet **expressive**

#### Problem already addressed many times :

Libraries	C++ Standard ?
<pre>std::vector, std::valarray, std::mdspan</pre>	<ul> <li>Image: A set of the set of the</li></ul>
EIGEN, Armadillo, xtensor, Blaze, Kokkos,	×

What if the real issue was a software design problem?

# nD Array Design Space

## A small sampling

- Owning or non-owning array ?
- Are my dimensions runtime or compile-time ? or both ?
- Storage order : C, FORTRAN, arbitrary, ... ?
- Indexes start at 0 ? 1 ? -3 ? any user value ?
- Memory allocation : Allocator-based ? Which allocator model ? Stack or Heap ?

### One implementation to rule them all ?

- Monolithic implementation leads to unmaintainable code
- Arbitrary restrictions on API is not a solution

#### **OUR POSITION**

Apply a different design strategy to the problem : generative programming

# Generative Programming [Czarnecki et al. : 2002]



# **Generative Programming for nD arrays**

# **Advantages of Generative programming**

- Reduce the number of implementations from  $|X| \cdot |Y| \cdot |Z|$  to |X| + |Y| + |Z|
- Generative programming in C++ is based on **template metaprogramming** and **code generation**

A-	B Save/Load + Add new ▼ 12 Vim <sup>P</sup> CppInsights ★ Quick-bench C++	-	Intel -	G++ 1	10	-	-O3 -DNDEBUG	-
1 #	include <kwk kwk.hpp=""></kwk>		A- 1	🏚 Outp	put 👻	Filter	👻 📃 Libraries (1) 🕂 Add new 👻 🖌 Add tool 👻	
2 #	include <vector></vector>		85		Lea	ra	dx, Irdx+rdx*41	
3 #	include <iostream></iostream>		86		mova	DS XM	mm0. XMMWORD PTR .LC3[rip]	
4			87		mov	ra	ax. rdi	
5 t	emplate <typename view=""></typename>		88		sal	rd	dx. 4	
6 V	roid process(View& v)		89		add	rd	dx. rdi	
7 {		_	90	.L12:				
8	<pre>kwk::for_each( [](auto&amp; c, auto i0, auto i1, auto i2) { c(i0,i1,i2) /= 10.f; }</pre>		91		movu	ps xm	mm1, XMMWORD PTR [rax]	
9	, v		92		add	ra	ax, 80	
10	);		93		div	s···xm	xmm1, xmm0	
11 }			94		movi	ps XM	(MMWORD PTR -80[rax], xmm1	
12			95		movu	ps xm	mm1, XMMWORD PTR -64[rax]	
13 i	nt main()		96		div	s xm	xmm1, xmm0	
14 {			97		movi	ps XM	(MMWORD PTR -64[rax], xmm1	
15	using namespace kwk::literals;		98		movu	ps xm	mm1, XMMWORD PTR -48[rax]	
16			99		div	s xm	xmm1, xmm0	
17	int n;	1	00		movi	ps XM	(MMWORD PTR -48[rax], xmm1	
18	std::cin >> n;	1	01		movu	DS XM	mm1, XMMWORD PTR -32[rax]	
19	<pre>std::vector<float> d(20*n);</float></pre>	1	02		div	s xm	xmm1, xmm0	
20	for(int i=0; auto& e : d) e = i++;	1	03		movi	ps XM	(MMWORD PTR -32[rax], xmm1	
21		1	04		movu	ps xm	mm1, XMMWORD PTR -16[rax]	
22	<pre>auto v = kwk::view{kwk::size = kwk::of_size(4_c, 5_c, n), kwk::source = d};</pre>	1	05		div	s xm	xmm1, xmm0	
23	process(v);	1	06		movi	DS XM	MMWORD PTR -16[rax], xmm1	
24 }		1	.07		cmp	rd	dx, rax	
		1	.08		jne	. L	L12	
		1	.09		test	rd	di, rdi	
		1	10		ine	. L	L8	
		1	11	.L10:				Edit on Compiler Explorer

5/21

# **Existing solutions**

# **Overview**

#### What do users want?

- Small and non-ambiguous interface
- High discoverability
- Clear and concise documentation

#### What do users deal with?

- Some APIs suffer from feature creep
- Some APIs require strong programming skills
- Some (most) APIs are sometimes counterintuitive and verbose

## What is missing?

- User friendly for non computer experts
- Keep API scope reasonable

# Painful example : std::vector

#### Higher dimensions vector declaration

// Access to fields
my\_vect[i0][i1][i2][i3][i4] = some\_float;

- For a  $d_1 imes d_2 imes \ldots imes d_n$  array,  $1 + d_1 imes d_2 imes \ldots imes d_{n-1}$  allocations occurs
- Data access induces spurious cache misses
- Memory is not contiguous, leading to sub-par performance

# Painful example : std::mdspan

#### **Extent definition**

```
1 std::extents<4,3> fixed_4x3; // Compile-time known extent
2 std::extents<std::dynamic_extent,std::dynamic_extent> dynamic(4,3); // Runtime known extent
3
4 std::extents< std::dynamic_extent,std::dynamic_extent,std::dynamic_extent,std::dynamic_extent
5 , std::dynamic_extent,std::dynamic_extent,std::dynamic_extent,std::dynamic_extent // Ughhhh
6 > dyn8d;
```

## **Type interface**

# **Painful example : Expression template libraries**

#### **API inconsistencies and traps**

```
auto unexpected_lazy_evaluation(matrix const& A, matrix const& B)
  auto C = A * B; // The type of C is not matrix
  return C;
}
template<typename T> auto dangerous_reference_to_temporary(T const& t)
  matrix m = t + t:
  return m / t; // returns a reference to a temporary
}
template<typename T> auto silent_performance_bug(T const& t)
  T that = t * t; // may or may not be of type T
  return that;
```

# Kiwaku

# Kiwaku design - Overview

#### Scope reduction limits feature creep

- Monolithic designs are bound to fail
- Kiwaku design focuses on a small set of closely relates features...

#### **Kiwaku's features**

- Owning and non-owning containers (kwk::view, kwk::table) with proper allocators
- Slicing of containers (soon)
- Algorithmic skeletons for order n containers

#### Kiwaku's non-features

- No linear algebra
- No expression templates
- No computation

## kwk**::**view

- Never owns any memory
- Wraps existing memory in a nD-array like interface
- Is designed to be as small as possible in 'resting position'

# kwk::table

- Owns memory allocated via an allocator or on the stack
- Wraps said memory in a nD-array like interface
- Is designed to be as small as possible in 'resting position'

## Two sides of the same coin

- A kwk::table is a kwk::view over the memory it owns
- Efficient kwk::view leads to efficient kwk::table

# Kiwaku design

### **Supported options**

- Kiwaku options are defined as orthogonal policies
- can be used as positional arguments in constructors
- options are exploited both at compile-time and run-time

Options	Values
kwk::source	pointers, static arrays, contiguous ranges
kwk::size	<pre>kwk::of_size(dn), integers are 1D-shapes</pre>
kwk::base_index	<pre>kwk::index<i>, kwk::indexes<i0,i1,></i0,i1,></i></pre>
kwk::label	any string-like values
kwk::allocator	any types modeling kwk::concepts::allocator

# Simple view construction

A - ■ Save/Load + Add new V Vim & CppInsights & Quick-bench C++ -	Intel - G++ 10 -O3 -DNDEBUG -std=c++20 -									
1 // simple_view.cpp	A ▼ ♦ Output ▼ Filter ▼ El Libraries (1) + Add new ▼ ✓ Add tool ▼									
2 #include <kwk kwk.hpp=""></kwk>	46 <b>call</b> std::basic ostream <char, std::char="" traits<char=""> &gt;&amp;</char,>									
3 #include <iostream></iostream>	47 movapd xmm0, XMMWORD PTR .LC4[rip]									
4	48 movaps XMMWORD PTR [rsp], xmm0									
F int moin()	49 movapd xmm0, XMMWORD PTR .LC5[rip]									
	50 movaps XMMWORD PTR 16[rsp], xmm0									
6 {	51 movapd xmm0, XMMWORD PTR .LC6[rip]									
7 using namespace kwk;	52 movaps XMMWORD PTR 32[rsp], xmm0									
8	53 .L2:									
9 double data[6] = $\{1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7\}$	54 movsd xmm0, QWORD PTR [rbx]									
	55 mov rdi, rbp									
	56 add rbx, 8									
<pre>11 auto vd = view{ source = data };</pre>	57 <b>call</b> std::basic_ostream <char, std::char_traits<char=""> &gt;&amp;</char,>									
<pre>12 std::cout &lt;&lt; vd.shape() &lt;&lt; "\n";</pre>	58 mov edx, 1									
<pre>13 std::cout &lt;&lt; vd.rank() &lt;&lt; "\n":</pre>	59 lea rsi, .LC1[rip]									
14	60 mov rdl, rax									
$\frac{14}{45} = \frac{1}{5} + \frac{1}{5} = \frac{1}{5} + $	61 Call std::basic_ostream <char, std::char_traits<char=""> &gt;&amp;</char,>									
$15  \text{for}(\text{std}::\text{size}_t \ 1 = 0; \ 1 < \text{vd}.\text{size}();++1)$										
16 vd(i) *= 10.;	CA MOV MAY OWODD DTD EXEMPT									
17	A - □ Wrap lines									
18 for(auto e : data)	ASM generation compiler returned: 0									
19 std::cout << e << " ";	Execution build compiler returned: 0									
20	Program returned: 0									
	12 23 34 43 30 07									

# **Rank 3 view from standard range**

A۰	B Save/Load + Add new ▼ V Vim <sup>P</sup> CppInsights ★ Quick-bench C++ ▼	Int	tel - G++ :	10	✓ O3 -DNDEBUG -std=c++20	-
1	<pre>#include <kwk kwk.hpp=""></kwk></pre>	A٠	🔹 🗘 Outp	out 🔻	▼ Filter ▼ 📃 Libraries (1) + Add new ▼ 🖌 Add tool ▼	
2	<pre>#include <vector></vector></pre>	1	.1.00	ə:		
3	<pre>#include <iostream></iostream></pre>	2			.string "["	
4		3		1:	ioci ing [	
5	<pre>int main()</pre>	4		1	string " "	
6	{	5	i .LC	2:	i o ci i ing	
7	using namespace kwk;	6		1	.string " ]"	
8				- ×		
9	<pre>std::vector<double> data(18);</double></pre>	Α-	🗖 🗌 Wraj	p lines		
10		Exe	cution	bui	ld compiler returned: 0	
11	<pre>auto vd = view{ source = data</pre>	Pro	gram r	etur	ned: 0	
12	, of_size(2,3,3)	I I	2 3 3	1		
13	, label = "base_freq"	2	хзх	3		
14	};	2	~ • ~			
15		1-	11 110	101	100 101 100 011 010 001 000 001 000 011 010 001 000	221 222
16	<pre>std::cout &lt;&lt; vd.shape() &lt;&lt; "\n";</pre>	11	11 112	121	122 131 132 211 212 221 222 231 232 311 312 321 322	331 332
17	std::cout << dim<0>(vd) << " x "					
18	<< dim<1>(vd) << " x "	ba	ase_fr	eq:		
19	<< dim<2>(vd) << "\n";		[ 111	112	]	
20	<pre>std::cout &lt;&lt; vd.rank() &lt;&lt; "\n";</pre>		[ 121	122	]	
21			[ 131	132	]	
22	for(std::size_t i2 = 0; i2 < dim<2>(vd);++i2)		-		-	
23	<pre>for(std::size_t i1 = 0; i1 &lt; dim&lt;1&gt;(vd);++i1)</pre>		۲ <u>2</u> 11	212	1	
24	for(std::size_t i0 = 0; i0 < dim<0>(vd);++i0)		[ 221	222	1	
25	vd(10, 11, 12) = (10+1) + 10*(11+1) + 100*(12+1);		[ 221	222	]	
26			[ 231	232	1	
27	for(auto e : data)					
28	<pre>std::cout &lt;&lt; e &lt;&lt; " ";</pre>		[ 311	312	]	
29	<pre>std::cout &lt;&lt; "\n\n";</pre>		[ 321	322	]	
30	and the second state of the second state the second state of the s		[ 331	332	]	
31	sta::cout << va << "\n";					Edit on Compiler Explorer

# **Code generation comparison with standard C**

A۰	B +- v 🔑 🖈	C++	A	▼ Left	t: Ir	ntel - G++ 10 -O3 -	DNDEBUG <del>y</del>	Assembly	•	Right:	ntel - G++ 10 -O3 -I	DNDEBUG <del>y</del>	Assembly	•
1	<pre>#include <kwk kwk.hpp=""></kwk></pre>			1-f	_kwl	k(float*, i	nt):			1+f	<pre>[_c(float*,</pre>	int):		
2				2		endbr64				2	endb	r64		
3	template <typename auto="" os="" t,=""></typename>			3		movsx	rsi, es	si		3	movs	x rsi,	esi	
4	<pre>void g( kwk::view<t,os> v)</t,os></pre>			4		test	rsi, rs	si		4	test	rsi,	rsi	
5	{			5		jle	.L1			5	jle	.L1		
6	<pre>for(std::ptrdiff_t i=0;i<v.size();++i)< pre=""></v.size();++i)<></pre>			6		lea	rax, -1	l[rsi]		6	lea	rax,	-1[rsi]	
7	v(i) *= 2.354f;			7		cmp	rax, 2			7	cmp	rax,	2	
8	}			8		jbe	.L6			8	jbe	.L6		
9	2			9		mov	rdx, rs	si		9	mov	rdx,	rsi	
10	<pre>void f_kwk(float* ptr, int n)</pre>			10		movaps	xmm1, X	(MMWORD PTR .	LC0[ri	10	mova	ps xmm1	, XMMWORD P	TR .LC0[ri
11	{			11		mov	rax, rd	li		11	mov	rax,	rdi	
12	<pre>q( kwk::view{kwk::source = ptr, kwk::s</pre>	ize = n});		12		shr	rdx, 2			12	shr	rdx,	2	
13	}	27.		13		sal	rdx, 4			13	sal	rdx,	4	
	5			14		add	rdx, rd	li		14	add	rdx,	rdi	
				15 .	L4:					15 .	L4:			
		0		16		movups	xmm0, X	(MMWORD PTR [	rax]	16	movu	ps xmm@	, XMMWORD P	TR [rax]
A.		C++ •		17		add	rax, 16	5		17	add	rax,	16	
1	<pre>#include <cstddef></cstddef></pre>			18		mulps	xmm0, x	(mm1		18	mulp	s xmm@	, xmm1	
2				19		movups	XMMWORD	) PTR <b>-16[</b> rax	:], xmm	1 <b>1</b> 9	movu	ps XMMW	ORD PTR -16	[rax], xmm
3	<pre>void f_c(float* ptr, int n)</pre>			20		cmp	rax, rd	ix		20	cmp	rax,	rdx	
4	{			21		jne	.L4			21	jne	.L4		
5	<pre>for(std::ptrdiff_t i=0;i<n;++i)< pre=""></n;++i)<></pre>			22		mov	rax, rs	si		22	mov	rax,	rsi	
6	ptr[i] *= 2.354f;			23		and	rax, -4	t –		23	and	rax,	- 4	
7	}			24		test	sil, 3			24	test	sil,	3	
				25		je	.L9			25	je	.L9		
				26.	L3:					26 .	L3:			
				27		lea	rdx, 0[	[0+rax*4]		27	lea	rdx,	0[0+rax*4]	
				28		movss	xmm0, D	WORD PTR .LC	:1[rip]	28	movs	s xmm@	, DWORD PTR	.LC1[rip]
				29		lea	rcx, [r	ˈdi+rdx]		29	lea	rcx,	[rdi+rdx]	
				30		movss	xmm1, D	WORD PTR [rc	:x]	30	movs	s xmm1	, DWORD PTR	[rcx]
				31		mulss	xmm1, x	¢mm⊙		31	muls	s xmm1	, ×mm0	
				32		movss	DWORD P	TR [rcx], xm	ım1	32	movs	S DWOR	D PTR [rc됐]	on Compiler Explorer

# Conclusion

# Conclusion

### **Current work**

- Kiwaku on Github
- Allocators : adaptation of Alexandrescu's model
- Slicers : complete the implementation

## Future of my PhD

- Support for tiling and non-trivial walkthrough (Morton, ...)
- Support for data locality in distributed computing scenarios
- Support for heterogenous platforms (GPU, FPGA, ...)
- Seamless integration with SYCL

# Thanks for your attention !