

Systematically Exploring High-Performance Representations of Vector Fields Through Compile-Time Composition

<u>Stephen Nicholas Swatman^{1,2}, Ana-Lucia Varbanescu³, Andy Pimentel¹, Andreas Salzburger², Attila Krasznahorkay²</u>

Monday, April 17, 2023

¹University of Amsterdam ²CERN ³University of Twente









- Vector fields map vectors onto elements of a set
- Common case: mapping vectors onto structured grids



Credit: Wikimedia Commons user Waldyrious, CC BY-SA 4.0

- Vector fields map vectors onto elements of a set
- Common case: mapping vectors onto structured grids
- Useful for modelling flows like wind...



Credit: Wikimedia Commons user Waldyrious, CC BY-SA 4.0

- Vector fields map vectors onto elements of a set
- Common case: mapping vectors onto structured grids
- Useful for modelling flows like wind...



Credit: ATLAS Collaboration

- Vector fields map vectors onto elements of a set
- Common case: mapping vectors onto structured grids
- Useful for modelling flows like wind...
- ...or magnetic fields in high-energy physics



Credit: ATLAS Collaboration

- Vector fields map vectors onto elements of a set
- Common case: mapping vectors onto structured grids
- Useful for modelling flows like wind...
- ...or magnetic fields in high-energy physics
- Performance-relevant data structures, but which representation is best?



Credit: ATLAS Collaboration

- Large design space: performance depends heavily on the *interaction between*:
 - Application
 - Functional implementation
 - Extra-functional implementation
 - Hardware



- Example: application migrates from two-dimensional approximation to three-dimensional field:
 - Functional implementation of the field changes...
 - ...the storage size changes (extra-functional behaviour)...
 - ...and the hardware cache behaviour changes



- Our goal: allow systematic exploration of representations of vector fields
 - Define the design space
 - *Compose* representations from simple components
 - Facilitate exploration through benchmarking



Application: Access Patterns

- In our suite, *applications* are modelled by their *access patterns*, capturing:
 - \cdot Locality of access
 - Computational side-effects
 - Functional requirements
- Variants:
 - CPU and GPGPU implementations
 - For patterns based on propagation through fields: order of execution
 - Depth-first or breadth-first propagation of agents

Name	Vr.	Field type
Scan	2	$\prod_{d,d':\mathbb{N}} S^{d+d'} \to T$
Random	2	$\prod_{d:\mathbb{N}} S^d \to T$
Euler	3	$\prod_{d:\mathbb{N}} \mathbb{R}^d \to \mathbb{R}^d$
RK4	3	$\prod_{d:\mathbb{N}} \mathbb{R}^d \to \mathbb{R}^d$
Lorentz	6	$\mathbb{R}^3 \to \mathbb{R}^3$

- Implementation of field data structure remains dauntingly complex
- Many choices of implementation
 - Functional: interpolation, boundary checking, geometric transformations, etc.
 - Extra-functional: data layouts, hardware acceleration, etc.
- This behaviour can (must!) be further decomposed into *primitive* behaviour and *transformers*

- Behaviour that cannot be meaningfully deconstructed
 - Basic array access (Array, CUDAArray)
 - More complex storage like CUDA textures (CUDATEX) or opaque arrays (CUDAPITCH)
 - Constant-valued and analytical fields (CONSTANT, ANALYTICAL)

Name	Pltf.	Field type
Array	CPU	$\mathbb{N} \to T$
CudaArray	CUDA	$\mathbb{N} \to T$
CudaPitch	CUDA	$\prod_{d:\llbracket 1,3\rrbracket} \mathbb{N}^d \to T$
CudaTex	CUDA	$\prod_{d:\llbracket 1,3\rrbracket}\prod_{d':\llbracket 1,4\rrbracket}\mathbb{R}^d\to\mathbb{R}^{d'}$
Analytic	CPU	$S \rightarrow T$
Constant	Any	$S \rightarrow T$

- Additional behaviour encoded as *transformers*, built of two parts:
 - Contravariant component allows to modify input
 - Covariant component allows to modify output









- We provide a variety of transformers for common implementation details
- These can be composed as long as the types line up
- Many transformers have additional variants (e.g. WRAP)

Name	Туре

LEXIC. $\prod_{n:\mathbb{N}} (\mathbb{N}^n \to \mathbb{N}) \times (T \to T)$

MORTON $\prod_{n:\mathbb{N}} (\mathbb{N}^n \to \mathbb{N}) \times (T \to T)$

HILBERT $(\mathbb{N}^2 \to \mathbb{N}) \times (T \to T)$

Shuffle $\prod_{n:\mathbb{N}} \prod_{p:\mathfrak{S}_n} (S^n \to S^n) \times (T \to T)$

$$OOB \qquad (S \to S + 1) \times (T + 1 \to T)$$

WRAP $(S \rightarrow S) \times (T \rightarrow T)$

NEAREST $\prod_{n:\mathbb{N}} (\mathbb{R}^n \to \mathbb{N}^n) \times (T \to T)$ LINEAR $\prod_{n:\mathbb{N}} (\mathbb{R}^n \to \mathbb{N}^{2^n n} \times \mathbb{R}^n) \times (\mathbb{R}^{2^n n} \times \mathbb{R}^n \to \mathbb{R}^n)$ AFFINE $\prod_{n:\mathbb{N}} (\mathbb{R}^n \to \mathbb{R}^n) \times (T \to T)$

- · New access pattern has "free" benchmarks with hundreds of backends
- New primitive storage composes with many transformers
- New transformer is compatible with many access patterns

- Composition is usually achieved through run-time polymorphism
- Incurs overhead from dynamic function dispatch
- Manageable for most applications...
- ...but deal-breaking in HPC kernels

```
1 backend f1;
2 layer3 f2(f1);
3 layer2 f3(f2);
4 layer1 f4(f3);
5
6 vector3 v = f4.at(5.f, 2.f, -2.f);
```

- Instead of composing at *run-time*, we compose behaviour at *compile-time*
- Use of C++ templating fixes behaviour at compile-time
 - Eliminates overhead due to dispatching
 - Increases optimisation space of the compiler
- We achieve native performance *and* preserve flexibility

```
1 using storage = layer1<
2 layer2 <
3 layer3 <
4 backend <... >
5 >,
6 ...
7 >,
8 ...
9 >;
```

Compile-Time Composition

using storage = lexicographic < ulong3, ulong3 >;



1	; g++ '	12.2.0 x86_64_v1
2	imul	0x8(%rsi),%rdx
3	add	%rcx,%rdx
4	imul	0x10(%rsi),%rdx
5	add	%rdx,%r8
6	mov	0x18(%rsi),%rdx
7	lea	(%r8,%r8,2),%rax
8	lea	(%rdx,%rax,8),%rax
9	movdqu	(%rax),%xmm0
10	mov	0x10(%rax),%rax
11	movups	%xmm0,(%rdi)
12	mov	%rax,0x10(%rdi)
13	mov	%rdi,%rax
14	ret	

Compile-Time Composition



1	; g++ '	12.2.0 x86_64_v1
2	imul	0x8(%rsi),%rcx
3	add	%r8,%rcx
4	imul	0x10(%rsi),%rcx
5	lea	(%rcx,%rdx,1),%rax
6	mov	0x18(%rsi),%rdx
7	lea	(%rax,%rax,2),%rax
8	lea	(%rdx,%rax,8),%rax
9	movdqu	(%rax),%xmm0
10	mov	0x10(%rax),%rax
11	movups	%xmm0,(%rdi)
12	mov	%rax,0x10(%rdi)
13	mov	%rdi,%rax
14	ret	

Compile-Time Composition

```
using storage = shuffle <shuffle <
shuffle <shuffle <shuffle <shuffle <
shuffle <lexicographic <ulong3,
ulong3 >, perm<2, 1, 0>>, perm<0,
2, 1>>, perm<1, 2, 0>>, perm<0,
2, 1>>, perm<0, 1, 2>>, perm<2,
0, 1>>, perm<0, 2, 1>>;
```



1	; g++	12.2.0 x86_64_v1
2	imul	0x8(%rsi),%rdx
3	add	%rcx,%rdx
4	imul	0x10(%rsi),%rdx
5	add	%rdx,%r8
6	mov	0x18(%rsi),%rdx
7	lea	(%r8,%r8,2),%rax
8	lea	(%rdx,%rax,8),%rax
9	movdqu	(%rax),%xmm0
10	mov	0x10(%rax),%rax
11	movups	%xmm0,(%rdi)
12	mov	%rax,0x10(%rdi)
13	mov	%rdi,%rax
14	ret	

- \cdot We imagine a high-energy physicist
- Goal: find the best implementation of a vector field in her software
- $\cdot\,$ Select one of the access patterns that models the application
 - \cdot Not exactly the same as the real application, but sufficiently close
- Make a selection of candidate implementations
 - $\cdot\,$ Construct these from the "building blocks" discussed before
- This generates a set of benchmarks, which can be run on one or more systems
- In this case: six setups in the DAS-6¹ compute cluster!

¹https://www.cs.vu.nl/das6/

Empirical Evaluation (HEP Example): Setup

1	<pre>benchmark::register_product_bm<</pre>
2	<pre>// Seven access patterns</pre>
3	<pre>boost::mp11::mp_list<</pre>
4	Lorentz <euler>,</euler>
5	Lorentz <rungekutta4>,</rungekutta4>
6	RungeKutta4Pattern,
7	EulerPattern,
8	Random,
9	Scan>,
10	<pre>// Seven composite implementations</pre>
11	<pre>boost::mp11::mp_list<</pre>
12	FieldConstant,
13	FieldTex <texinterpolatelin>,</texinterpolatelin>
14	<pre>FieldTex<texinterpolatenn>,</texinterpolatenn></pre>
15	Field <interpolatenn, layoutstride="">,</interpolatenn,>
16	Field <interpolatenn, layoutmorton="">,</interpolatenn,>
17	Field <interpolatelin, layoutstride="">,</interpolatelin,>
18	Field <interpolatelin, layoutmorton=""></interpolatelin,>
19	>
20	<pre>// Generates a large number of benchmarks!</pre>
21	>();

Empirical Evaluation (HEP Example): Results



Empirical Evaluation (HEP Example): Results



→ Linear/Lexic
 → Linear/Morton
 → CudaTex(Linear)



23

- Our benchmark suite provides a range of components and infrastructure
- This translates (with some effort on our part) to a user-facing library
- Makes the results of benchmarks immediately *applicable*
- Eliminates ambiguity between benchmark results and manual re-implementation
- Library available under MPL-2.0 license
- Currently in use in experimental HEP software!

Artifact



- Artifact is permanently available at https://zenodo.org/record/7540593
- Provided with an optional Docker file
- Specific care taken to ensure code works without Docker
 - But using the Docker file will guarantee sane and reproducible environment

	Jensey 15, 2023 Benchmarking Softw Exploring High-Perfor of Vector Fields Thro	are for "Syste rmance Repre ugh Compile-	matically esentations Time	288 14 • vines Ladoestaads Bee more databu
	Composition"			
	Ananymous			
	This artifact contains the benchmarking code for our of Viector Fields Through Compile Tame Composition" of new results. In order to use this abstract, we recome the total process should take no more than two hour the production, and accound how days for a full reproduct	saper 'Systematically Exploring Hij It allows for the reproduction of or mend starting with Section 1 and r (most of which is non-interactive ion.	gh-Performance Representations ar results, as well as the generation eorking through it in sequence. compute time) for a quick	OpenAIRE
Image: market of the second	Note: This ZIP file does not contain a top-level directo [archive_name], zip.	ry. To unpack it into a single direct	9%,090 untip -d dir_name	Publication date:
Constrained and constrain	Preview		~	January 15, 2023
Border and a set of the set	(\$10,5381, seconds 70.0023) of size			222 12 5285, weather 7540991
Bandward and a stand and	33 Lo cella fastación consecutiva séc		<u>^</u>	License (Far files):
Bandward (Schward) (Schward) (Schward) Bandward (Schward) (Schward) (Schward) (Schward) (Schward) Bandward (Schward) (Schward) (Schward) (Schward) (Schward) Bandward (Schward)	dockerignore		30 Dyten	CP Micenia Public License 2.0
Stored 1000 Owner, Missen 1000 Owner, Missen 1000 Stored 10000	Dockerste D Makadia		734 Dytes 7.7 kB	
And Andrew State St	C REACHE and		12.8 kB	
Constructions of the second of the seco	in data			Mersions
•••••••••••••••••••••••••	 D result_27934 json 		7.4 MB	Per anorra
Control C	 D result_27915 json 		7.4 MB	
	 Dresult_27916 json 		7.4 MB	
Berlandson Berlandson Berlandson <td> Inesult_27917 (son) </td> <td></td> <td>7.4 MB</td> <td></td>	 Inesult_27917 (son) 		7.4 MB	
Demographic 1100 Demographic 1100 <td< td=""><td> D result_27979 json </td><td></td><td>7.6 MB</td><td></td></td<>	 D result_27979 json 		7.6 MB	
Barrow Barrow <td> C result_3688037 (son) </td> <td></td> <td>7.6 MB</td> <td></td>	 C result_3688037 (son) 		7.6 MB	
Binding Angelogies Binding Angelogies Binding Angelogies Binding Angelogies Statistical Statistics Binding Angelogies Binding Angelogies Binding Angelogies Statistical Statistics Binding Angelogies	Difference of the second secon		145.9 MB	
Demendance of the second secon	D brox Burdens		4.1 x8	
Biome, and particular set of the	C process cpu py		2.3 KB	
Tanana and a second and a secon	D servers operate av		2.540	
Another				Dife all revisions? You can the all versions by using the DDI
The second of the second secon	C PA PERSON NAMES OF A DESCRIPTION OF A			In Sector and All 1979. The DCO presents at vesters, and all alongs reaches in the intent over, family over,
Non No Using much West (Ling Ling	Film (co.on)		~	
Statil (marked) (b) Statil Terming (b) Centry order diverse their means of the end of th	Name	Size		Share
And Informationseement 9 Conset	10.5281_zeerods.7079829_v3.zip	4.3 MB	Prover A Download	CI AddThis
Connel Co	หม่องสระสารที่สาวที่สาวที่สองสรรณสร้าน 🧶			Cite as
Create 0 V Dispersion of the Fride Network () Universe () Sector 4				Systematically Exploring High-Performance
Brenny Classica () Chana () Classes () Classes () Losson () Incohor () Classes () Classe	Citations © O		~	Expresentations of Vector Fields Through Comple-
Cluttore to the version No citations.	Show only: C Likewiser (D) C Debaset (D) C S	offware (E) 🗆 Unknown (C)	Smith 10	Time Composition". Zenode. https://doi.org/10.5281 /perode.7540593
No citations.	CC Citations to this version		u	
No citations.				
		No citations.		
				BETOX CSL DataCity Duble Core DCAT

SW-1 Benchmarking suite:

- a Primary benchmarking code
- b Code for assembly inspection
- c Dummy field generation code
- SW-2 Plotting code.
- DATA-1 Results from our experiments
- DATA-2 Vector field data
 - a Real-world data from HEP
 - b Dummy field data



- We present a *benchmark suite* for vector fields on structured grids
- *Composition* at compile-time achieves native performance while being *extensible* and allowing *design space exploration*
- Gives rise to a *novel library*, available publicly now
- Artifact available to fully reproduce our results, and to generate new data