

# Automatic Generation of Efficient Codes from Mathematical Descriptions of Stencil Computation

Takayuki Muranushi<sup>1</sup>   Seiya Nishizawa<sup>1</sup>   Hirofumi Tomita<sup>1</sup>  
Keigo Nitadori<sup>1</sup>   Masaki Iwasawa<sup>1</sup>   Yutaka Maruyama<sup>1</sup>  
Hisashi Yashiro<sup>1</sup>   Yoshifumi Nakamura<sup>1</sup>   Hideyuki Hotta<sup>2</sup>  
Junichiro Makino<sup>3</sup>   Natsuki Hosono<sup>4</sup>   Hikaru Inoue<sup>5</sup>

<sup>1</sup>RIKEN Advanced Institute for Computational Science

<sup>2</sup>Chiba University

<sup>3</sup>Kobe University

<sup>4</sup>Kyoto University

<sup>5</sup>Fujitsu Ltd.

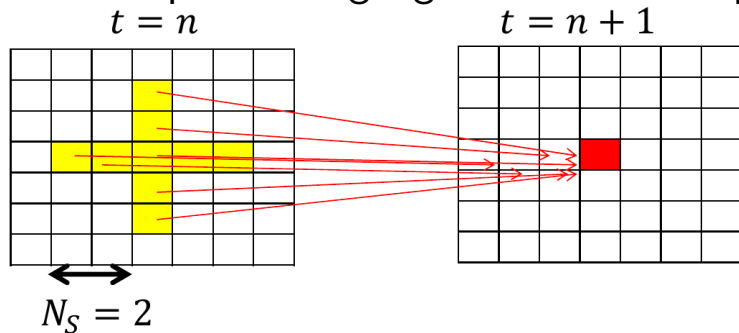
Sep 22, 2016

for FHPC 2016 workshop / ICFP'16 Nara, Japan

Programming Language

# Formura

## Domain specific language for stencil computaion





## Good news of Formura 1/2

1.184 Petaflops (11.62% of the peak)  
on 663,552 cores



# Good news of Formura 1/2

## ACM Gordon Bell Prize Finalist

Wednesday, November 16th



TIME	SESSION / PRESENTATION	PRESENTERS
11:30am - 12pm	ACM Gordon Bell Finalist: <a href="#">Simulations of Below-Ground Dynamics of Fungi: 1.184 Pflops Attained by Automated Generation and Autotuning of Temporal Blocking Codes</a>	Muranushi, Hotta, Makino, Nishizawa, Tomita, Nitadori, Iwasawa, Hosono, Maruyama, Inoue...

## Good news of Formura 2/2

$$\frac{\partial \rho}{\partial t} = - \sum_{i=1}^3 \frac{\partial}{\partial x_i} (\rho v_i)$$

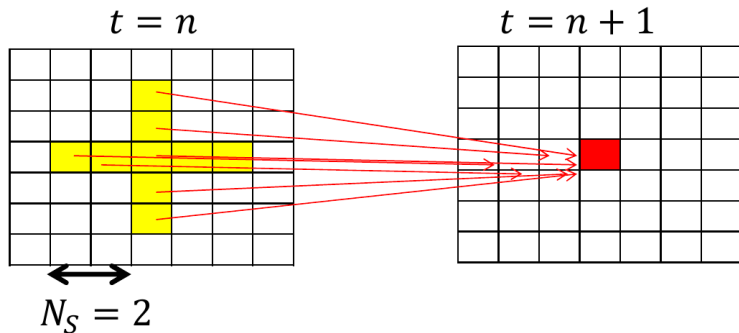
$$\text{ddt\_}\rho = - \Sigma \text{ fun}(i) \partial i (\rho * v i)$$

- is a functional programming language
- is implemented in a functional programming language (Haskell)



# Backend: How we generate efficient codes

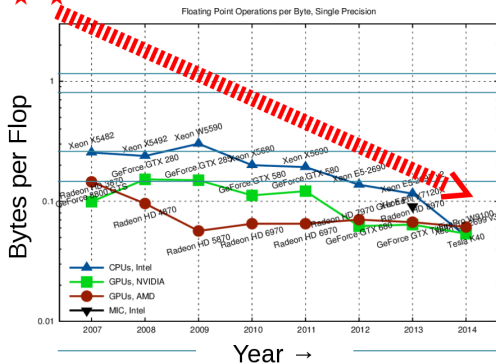
# Stencil Computation



# Byte / Flops of hardwares are decreasing

FPS-164&VAX(1976) (B/F=4)

★ NEC SX-8(2004) (B/F=4)



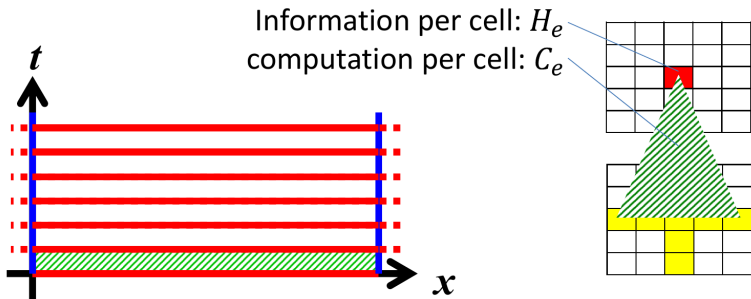
## Applications

- 2d diffusion  
B/F=1.14
- 3d diffusion  
B/F=0.8
- Seismic wave  
B/F=0.275
- 3d CIP Hydro  
B/F=0.16
- Godunov Hydro  
B/F= $5.92 \times 10^{-3}$

Data cited from: Comparison of required floating point operations per byte when using single precision in order to transition between compute-limited and memory-bandwidth-limited regimes..(c) Karl Rupp

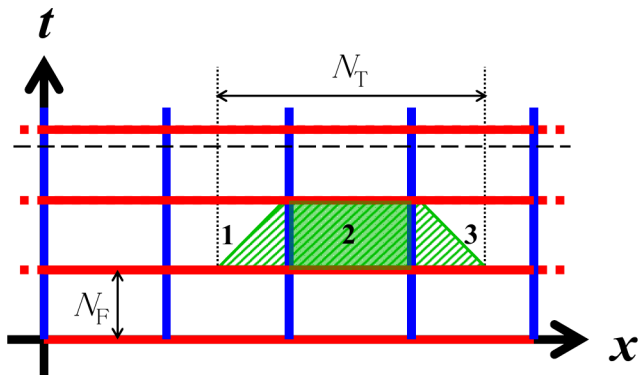
# Naive implementation of stencil computation

The optimal  $\frac{B}{F} = \frac{2H_e}{C_e}$

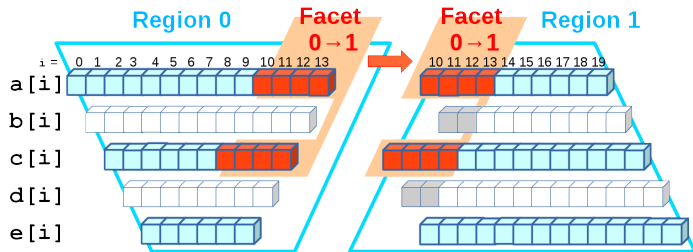


# Temporal Blocking

The optimal  $\frac{B}{F} = \frac{2H_e}{C_e} \left( \frac{1}{N_F} + \frac{2dN_s}{N_T} \right)$



## Decompose &amp; fuse array computations in space-time



```

manifest :: a[i]
           b[i] = a[i-1] + a[i] + a[i+1]
manifest :: c[i] = b[i-1] * b[i] * b[i+1]
           d[i] = c[i-1] + c[i] + c[i+1]
manifest :: e[i] = d[i-1] * d[i] * d[i+1]

```

**Octahedron comb.**

Separate the axial  $(\frac{1}{2}, \frac{1}{2}, 1)$  into four cones, and also prove the pair of cones results in regular octahedron B and four standard octahedron C.

$$S = 4N^2 \quad V = \frac{4}{3}N^3$$

$$V = \frac{4}{3}N^3 \quad V = \frac{1}{3}N^3$$

in average

**(2,2,2) tilted cube**

By similar affine transformation of cube, translation.

$$S = 4N^2 \text{ face}$$

$$V = 4N^3$$

in average of  $\frac{S}{V} = \frac{1}{N} = \frac{1}{2} = 0.5$

**(2,2,2) tilted cube**

By similar affine transformation of cube, translation.

$$S = 8(n+1)N, \quad V = 8N^3, \quad \frac{S}{V} = \text{in average edge}$$

$$\frac{S}{V} = \text{in average of } \frac{1}{n} = 0.5$$

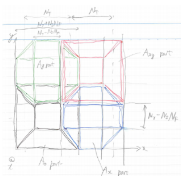
when  $\frac{S}{V} = \frac{24}{8} = 3 = 2^2 = 2^2 \times 2^0$

**正八面体 octahedron**

面積  $S = 2 \times 2 \times 2 \times 2 = 16$   
 体積  $V = 2 \times 2 \times 2 = 8$   
 $\frac{S}{V} = \frac{16}{8} = 2 = 2^1 = 2 \times 2^0$

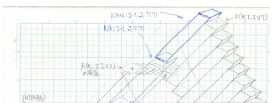
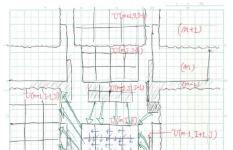
由  $\frac{S}{V} = \frac{16}{8} = 2 = 2^1 = 2 \times 2^0$  可知  $\frac{S}{V}$  的值为  $2^1$  或  $2^0$

cell	lessellation	$\frac{S}{V}$	$\frac{S}{V}$
octahedron	X	3.3019	$6^{2/3}$
$\sqrt{2}$ -tilted cube	O	3.7798	$3 \times 2^{2/3}$
$\sqrt{3}$ -tilted cube	O	4.1602	$2 \times 3^{2/3}$
octahedron	O	3.7798	$3 \times 2^{2/3}$
pseudocuboctahedron	O	4.1602	$2 \times 3^{2/3}$
octahedron comb.	X	4.4020	$\frac{4}{3} \times 6^{2/3}$
motherhip 1	X	3.5851	$(\frac{1}{2}) \times 6^{2/3}$
motherhip 2	X	3.9685	$\frac{1}{2} \times 2 \times 3^{2/3}$
stair-tetrahedron	X	3.7798	$3 \times 2^{2/3}$
non-convex	X	3.7090	$\frac{2}{3} \times (\frac{1}{2})^{2/3}$
rhombicuboctahedron	O	3.7798	$3 \times 2^{2/3}$
cube	O	6	

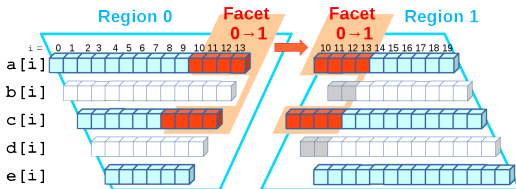


$3 \times 3 \times 3 = 27$

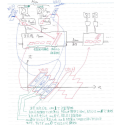
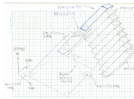
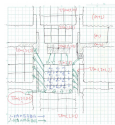
Diagram showing the decomposition of a  $3 \times 3 \times 3$  cube into smaller cubes. The decomposition is shown as a  $3 \times 3 \times 3$  cube with a  $2 \times 2 \times 2$  cube inside, and the remaining space filled with  $1 \times 1 \times 1$  cubes.



# In which language shall we code?



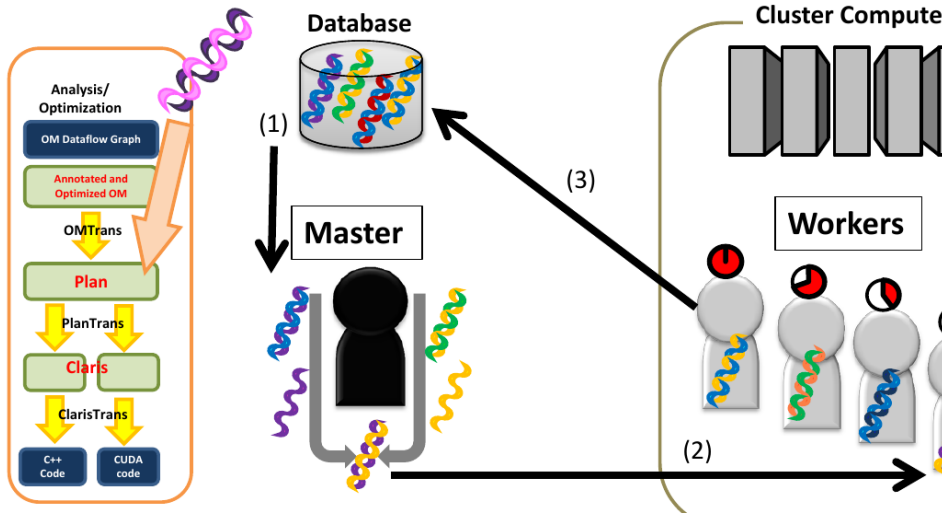
no.	intensity	no.	intensity
0	0.0000	10	0.0000
1	0.0000	11	0.0000
2	0.0000	12	0.0000
3	0.0000	13	0.0000
4	0.0000	14	0.0000
5	0.0000	15	0.0000
6	0.0000	16	0.0000
7	0.0000	17	0.0000
8	0.0000	18	0.0000
9	0.0000	19	0.0000
10	0.0000	20	0.0000
11	0.0000	21	0.0000
12	0.0000	22	0.0000
13	0.0000	23	0.0000
14	0.0000	24	0.0000
15	0.0000	25	0.0000
16	0.0000	26	0.0000
17	0.0000	27	0.0000
18	0.0000	28	0.0000
19	0.0000	29	0.0000
20	0.0000	30	0.0000
21	0.0000	31	0.0000
22	0.0000	32	0.0000
23	0.0000	33	0.0000
24	0.0000	34	0.0000
25	0.0000	35	0.0000
26	0.0000	36	0.0000
27	0.0000	37	0.0000
28	0.0000	38	0.0000
29	0.0000	39	0.0000
30	0.0000	40	0.0000
31	0.0000	41	0.0000
32	0.0000	42	0.0000
33	0.0000	43	0.0000
34	0.0000	44	0.0000
35	0.0000	45	0.0000
36	0.0000	46	0.0000
37	0.0000	47	0.0000
38	0.0000	48	0.0000
39	0.0000	49	0.0000
40	0.0000	50	0.0000
41	0.0000	51	0.0000
42	0.0000	52	0.0000
43	0.0000	53	0.0000
44	0.0000	54	0.0000
45	0.0000	55	0.0000
46	0.0000	56	0.0000
47	0.0000	57	0.0000
48	0.0000	58	0.0000
49	0.0000	59	0.0000
50	0.0000	60	0.0000
51	0.0000	61	0.0000
52	0.0000	62	0.0000
53	0.0000	63	0.0000
54	0.0000	64	0.0000
55	0.0000	65	0.0000
56	0.0000	66	0.0000
57	0.0000	67	0.0000
58	0.0000	68	0.0000
59	0.0000	69	0.0000
60	0.0000	70	0.0000
61	0.0000	71	0.0000
62	0.0000	72	0.0000
63	0.0000	73	0.0000
64	0.0000	74	0.0000
65	0.0000	75	0.0000
66	0.0000	76	0.0000
67	0.0000	77	0.0000
68	0.0000	78	0.0000
69	0.0000	79	0.0000
70	0.0000	80	0.0000
71	0.0000	81	0.0000
72	0.0000	82	0.0000
73	0.0000	83	0.0000
74	0.0000	84	0.0000
75	0.0000	85	0.0000
76	0.0000	86	0.0000
77	0.0000	87	0.0000
78	0.0000	88	0.0000
79	0.0000	89	0.0000
80	0.0000	90	0.0000
81	0.0000	91	0.0000
82	0.0000	92	0.0000
83	0.0000	93	0.0000
84	0.0000	94	0.0000
85	0.0000	95	0.0000
86	0.0000	96	0.0000
87	0.0000	97	0.0000
88	0.0000	98	0.0000
89	0.0000	99	0.0000
90	0.0000	100	0.0000

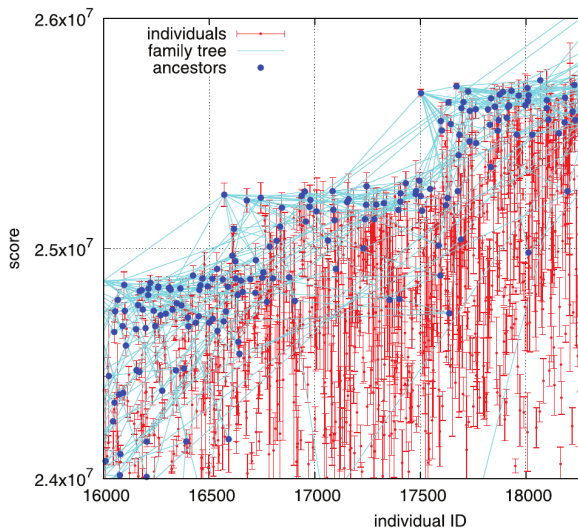
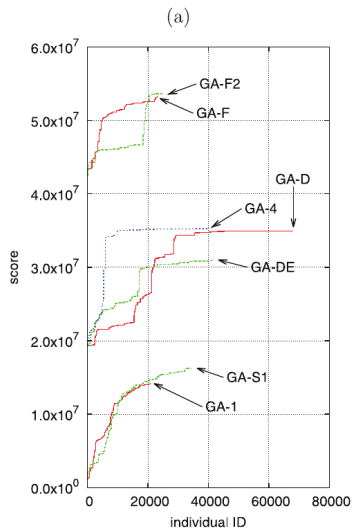




# Paraiso : a DSL embedded in Haskell (Muranushi, 2012)

among Nikola (Mainland & Morrisett, 2010), Obsidian (Svensson, 2011), Accelerate (Chakravarty et al., 2011), SPOC (Bourgoin et al., 2012), NOVA (Collins et al., 2014), and LMS series (Rompf, 2012).





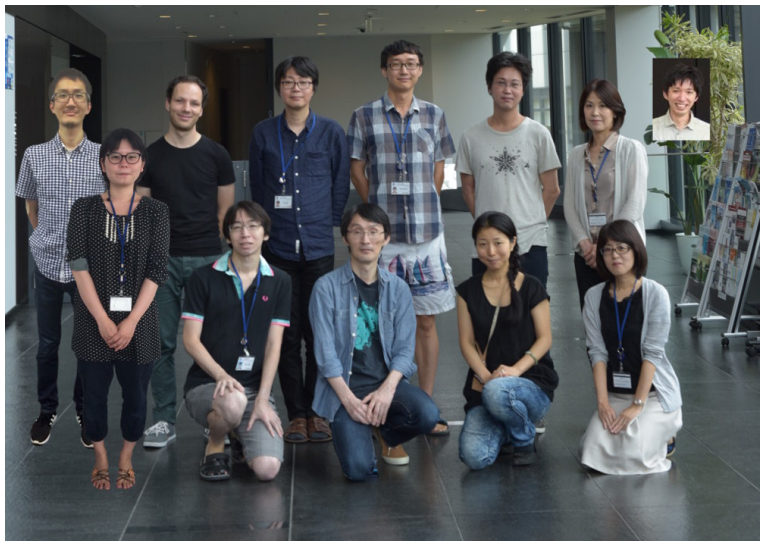
## Paraiso: a bad sell

```

proceedSingle :: Int -> BR -> Dim BR -> Hydro BR -> Hydro BR -> B (Hydro BR)
proceedSingle order dt dR cellF cells = do
  let calcWall i = do
        (lp,rp) <- interpolate order i cellF
        hllc i lp rp
      wall <- sequence $ compose calcWall
  foldl1 (.) (compose \i -> (>>= addFlux dt dR wall i))) $ return cells

```

## Our team



# Formura : a standalone DSL

## Design principle of Formura

- Simple enough
- Rich enough

## Syntax of Formura

```
# dimension declaration
dimension :: 3
# array declaration
double [] :: vx, vy, vz
# array computation
A2[i,j,k] = A[i-1] + A[i+1]
# Tuple
v = (vx, vy, vz)
# Lambda expression
tripe = fun (x) 3 * x
```



## Tuples are functions

$$\begin{array}{l} (a, b) \ 1 \quad = \ b \\ (f, (h, p, c)) \ 1 \ 2 \ = \ c \end{array}$$

## Inferred promotion to tuples and functions

$$\begin{aligned}
 x + (a, b) &= (x+a, x+b) \\
 (x, y) + (a, b) &= (x+a, y+b) \\
 (x, y) + (a, b, c) &= \perp
 \end{aligned}$$

$$\begin{aligned}
 (f + g) x &= f x + g x \\
 (f + g + 1) x &= f x + g x + 1
 \end{aligned}$$

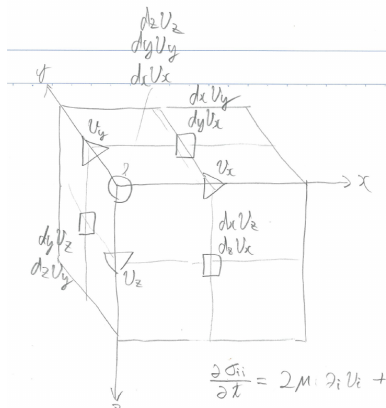
```

rk4 = fun(ddt) \
  fun(sys_0) let \
    sys_q4 = sys_0 + dt/4 * ddt(sys_0)
    sys_q3 = sys_0 + dt/3 * ddt(sys_q4)
    sys_q2 = sys_0 + dt/2 * ddt(sys_q3)
    sys_next = sys_0 + dt * ddt(sys_q2)
  in sys_next

```

# Differentiation Operators

$$\begin{aligned} \text{ddx} &= \text{fun}(a) (a[i+1/2, j, k] - a[i-1/2, j, k]) / dx \\ \text{ddy} &= \text{fun}(a) (a[i, j+1/2, k] - a[i, j-1/2, k]) / dy \\ \text{ddz} &= \text{fun}(a) (a[i, j, k+1/2] - a[i, j, k-1/2]) / dz \end{aligned}$$



## Nabla and Summation

```
 $\partial = (\text{ddx}, \text{ddy}, \text{ddz})$   
 $\Sigma = \text{fun } (e) \ e \ 0 + e \ 1 + e \ 2$ 
```

## Evaluation of formura expression

$$\Sigma \text{ fun}(i) \partial i (\rho * v i)$$

## Evaluation of formura expression

$$\Sigma \text{ fun}(i) \partial i (\rho * v i)$$

$$\Sigma = \text{fun } (e) \ e \ 0 + e \ 1 + e \ 2$$

## Evaluation of formura expression

$$\Sigma \text{ fun}(i) \partial i (\rho * v i)$$

$$\Sigma = \text{fun } (e) \ e \ 0 + e \ 1 + e \ 2$$

$$\begin{aligned} \longrightarrow & (\text{fun}(i) \partial i (\rho * v i)) \ 0 \\ & + (\text{fun}(i) \partial i (\rho * v i)) \ 1 \\ & + (\text{fun}(i) \partial i (\rho * v i)) \ 2 \end{aligned}$$

## Evaluation of formura expression

$$(\text{fun}(i) \partial i (\rho * v i)) 0$$



## Evaluation of formura expression

$$\rightarrow \partial_0 (\text{fun}(i) \partial_i (\rho * v_i))_0$$

## Evaluation of formura expression

$$\rightarrow \partial_0 (\rho * v_0)$$

$$\partial = (\text{ddx}, \text{ddy}, \text{ddz})$$

$$v = (v_x, v_y, v_z)$$

$$(a, b, c)_0 = a$$

## Evaluation of formura expression

$$\begin{aligned} & (\text{fun}(i) \partial i (\rho * v i)) 0 \\ \longrightarrow & \partial 0 (\rho * v 0) \end{aligned}$$

$$\partial = (\text{ddx}, \text{ddy}, \text{ddz})$$

$$v = (v_x, v_y, v_z)$$

$$(a, b, c) 0 = a$$

$$\longrightarrow \text{ddx} (\rho * v_x)$$

# Evaluation of formura expression

$$\text{ddx } (\rho * vx)$$

## Evaluation of formura expression

$$\text{ddx } (\rho * vx)$$

```
ddx = fun(a) (a[i+1/2,j,k] - a[i-1/2,j,k])/dx
```

## Evaluation of formura expression

$$\text{ddx } (\rho * vx)$$

$$\text{ddx} = \text{fun}(a) (a[i+1/2, j, k] - a[i-1/2, j, k]) / dx$$

$$\longrightarrow ((\rho * vx)[i+1/2, j, k] - (\rho * vx)[i-1/2, j, k]) / dx$$

## Evaluation of formura expression

$$\text{ddx } (\rho * vx)$$

$$\text{ddx} = \text{fun}(a) (a[i+1/2,j,k] - a[i-1/2,j,k])/dx$$

$$\longrightarrow ((\rho * vx)[i+1/2,j,k] - (\rho * vx)[i-1/2,j,k])/dx$$

$$\longrightarrow (\rho[i+1/2,j,k] * vx[i+1/2,j,k] - \rho[i-1/2,j,k] * vx[i-1/2,j,k])/dx$$

## Evaluation of formura expression

$$\Sigma \text{ fun}(i) \partial i (\rho * v i)$$



## Evaluation of formura expression

$$\begin{aligned} & \Sigma \text{fun}(i) \partial i (\rho * v i) \\ \longrightarrow & (\rho[i+1/2, j, k] * vx[i+1/2, j, k] - \\ & \rho[i-1/2, j, k] * vx[i-1/2, j, k]) / dx + \\ & (\rho[i, j+1/2, k] * vy[i, j+1/2, k] - \\ & \rho[i, j-1/2, k] * vy[i, j-1/2, k]) / dy + \\ & (\rho[i, j, k+1/2] * vz[i, j, k+1/2] - \\ & \rho[i, j, k-1/2] * vz[i, j, k-1/2]) / dz \end{aligned}$$

## Evaluation of formura expression

$$\sum_{i=1}^3 \frac{\partial}{\partial x_i} (\rho v_i)$$

$$\Sigma \text{ fun}(i) \partial i (\rho * v i)$$

$$\begin{aligned} \longrightarrow & (\rho[i+1/2,j,k] * vx[i+1/2,j,k] - \\ & \rho[i-1/2,j,k] * vx[i-1/2,j,k])/dx + \\ & (\rho[i,j+1/2,k] * vy[i,j+1/2,k] - \\ & \rho[i,j-1/2,k] * vy[i,j-1/2,k])/dy + \\ & (\rho[i,j,k+1/2] * vz[i,j,k+1/2] - \\ & \rho[i,j,k-1/2] * vz[i,j,k-1/2])/dz \end{aligned}$$

## More to talk about

- Modular Reifiable Matching (MRM)(Oliveira et al., 2015) + Pattern synonym solves “expression problem”
- Details of code transformation paths
- Varieties of temporal blocking methods
- How we have gave proof to certain types of temporal blocking methods

## Conclusion

### Functional programming

- is a good choice for user interface
  - weather scientists and astronomers can use it
- is crucial in implementing all the program transformations
  - achieves high performance

## Conclusion

1.184 Pflops  
Formura

# Bibliography I

- Bourgoin, M., Chailloux, E., & Lamotte, J.-L. 2012, *Parallel Processing Letters*, 22, 1240007
- Chakravarty, M. M., Keller, G., Lee, S., McDonell, T. L., & Grover, V. 2011, in *Proceedings of the sixth workshop on Declarative aspects of multicore programming*, ACM, 3–14
- Collins, A., Grewe, D., Grover, V., Lee, S., & Susnea, A. 2014, in *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, ACM, 8
- Mainland, G., & Morrisett, G. 2010 in , ACM, 67–78
- Oliveira, B. C. d. S., Mu, S.-C., & You, S.-H. 2015, in *Proceedings of the 8th ACM SIGPLAN Symposium on Haskell*, ACM, 82–93
- Rompf, T. 2012, PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
- Svensson, J. 2011, PhD thesis, Chalmers University of Technology