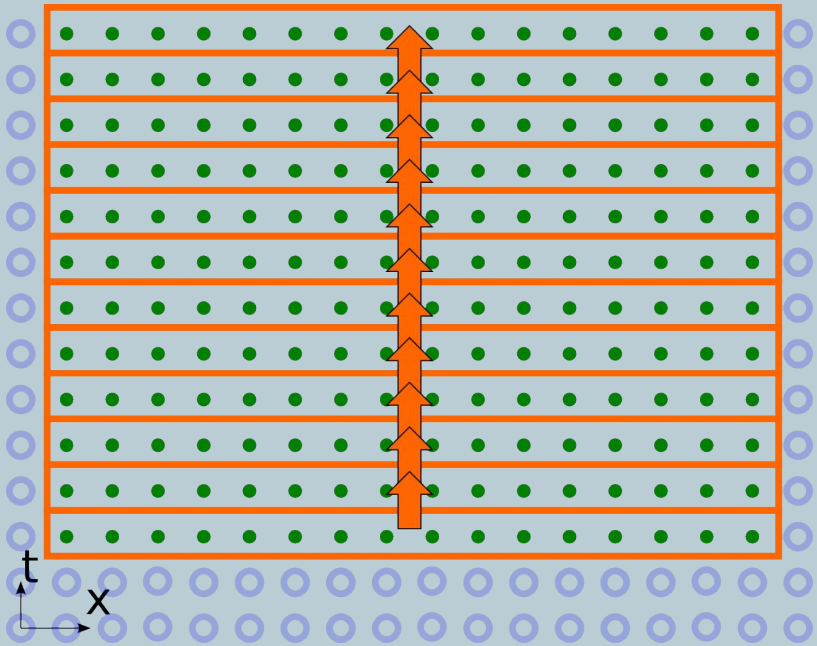
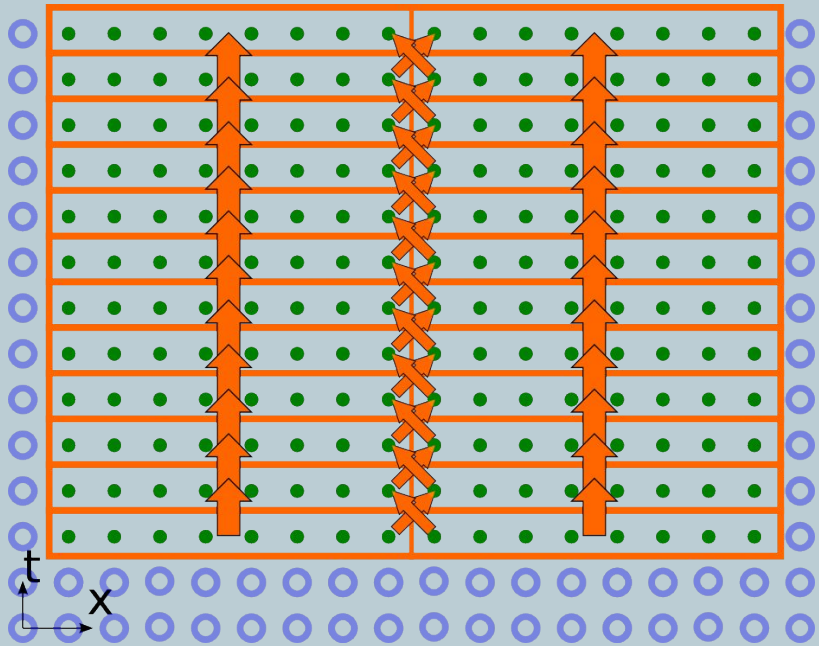
LRnLA Algorithm ConeFold with non-Local Vectorization for Lattice Boltzmann Method Implementation

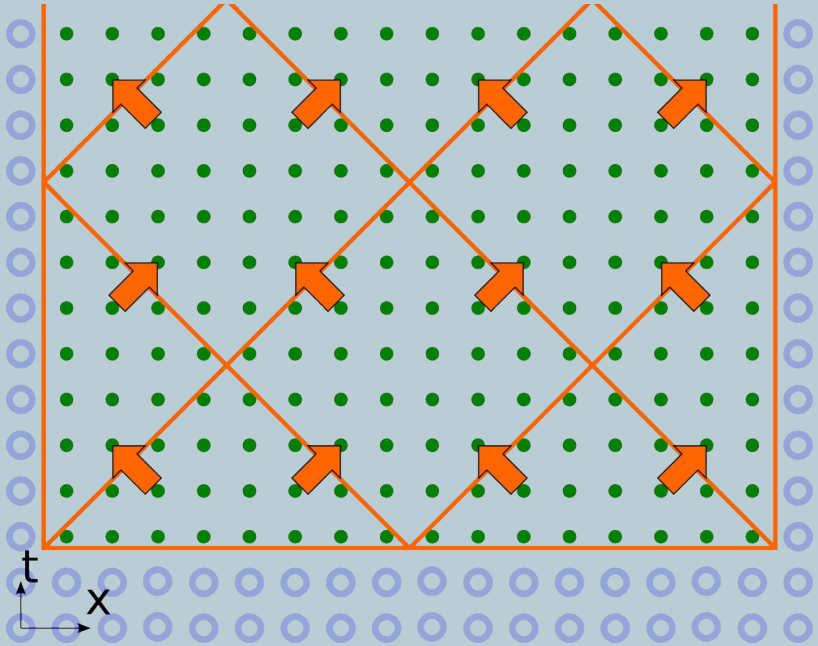
Anastasia Perepelkina,
Vadim Levchenko
Keldysh Institute of Applied Mathematics RAS, Moscow, Russia
mogmi@narod.ru, lev@keldysh.ru

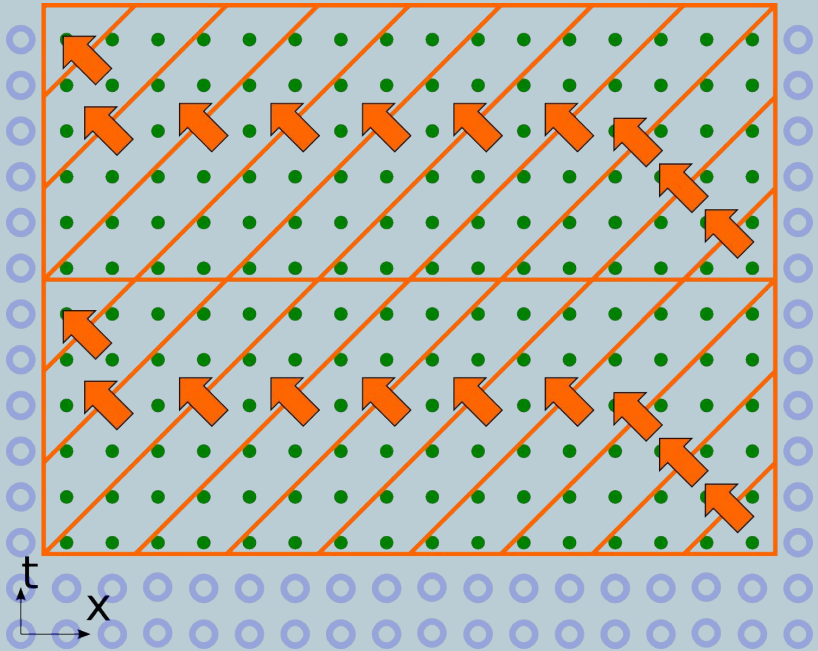
Algorithm as a Decomposition of Dependency Graph



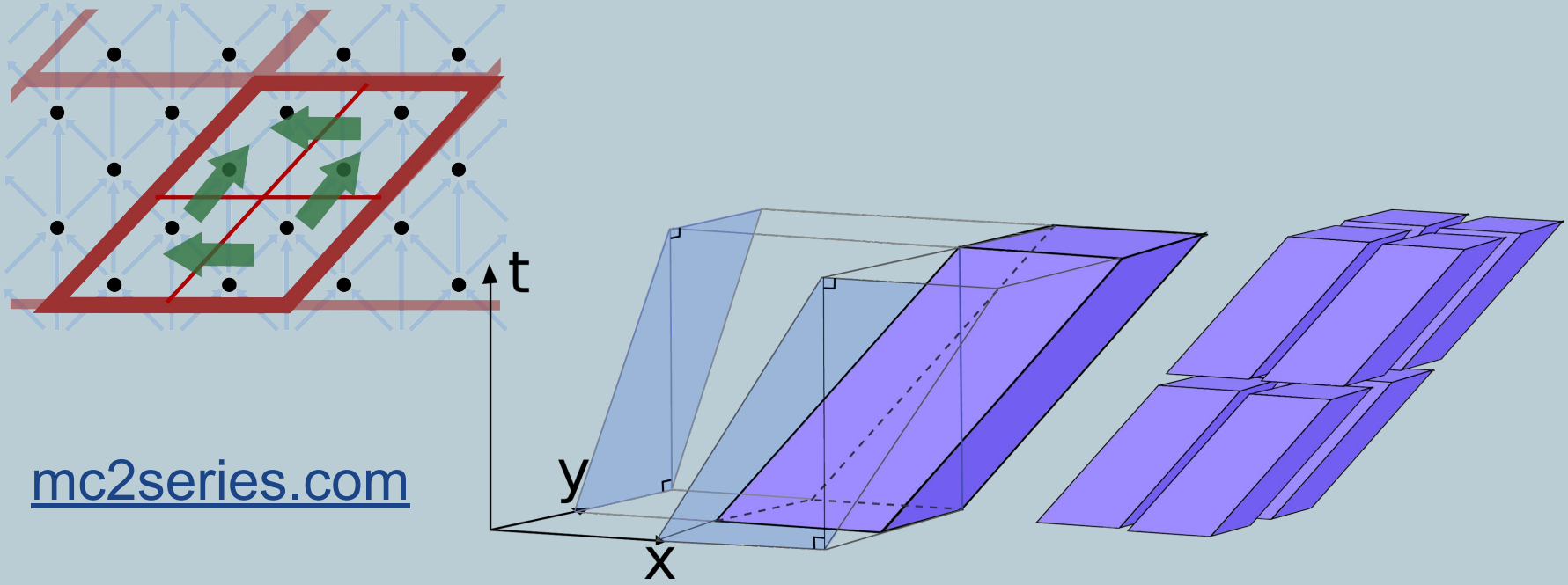


Algorithm as a Decomposition of Dependency Graph





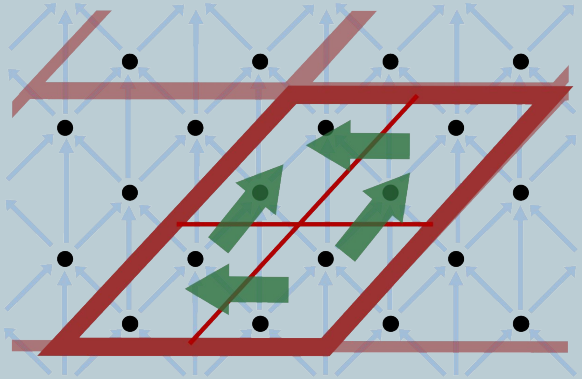
Locally Recursive non-Locally Asynchronous (LRnLA) ***ConeFold*** Algorithm

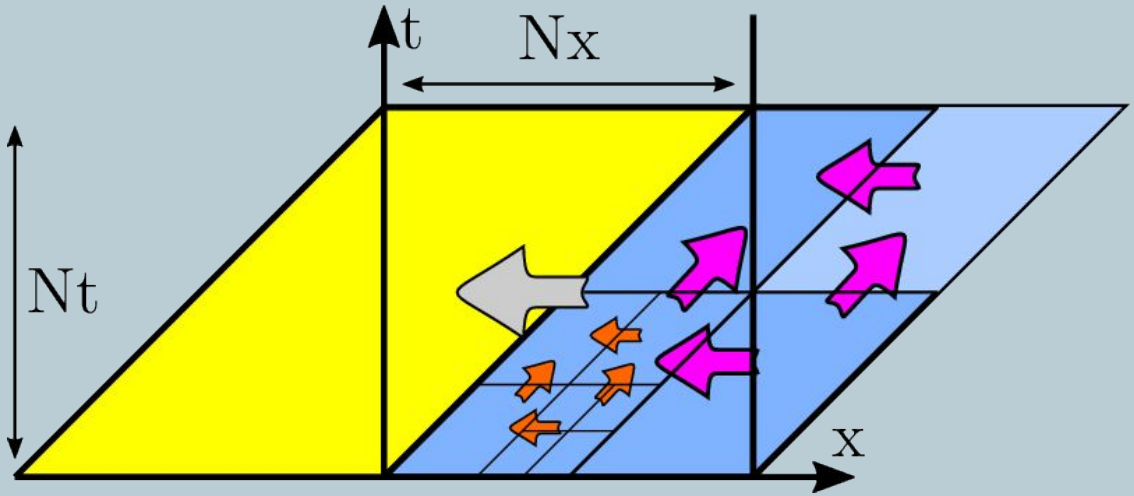


mc2series.com

Locally Recursive non-Locally Asynchronous (LRnLA)

ConeFold Algorithm



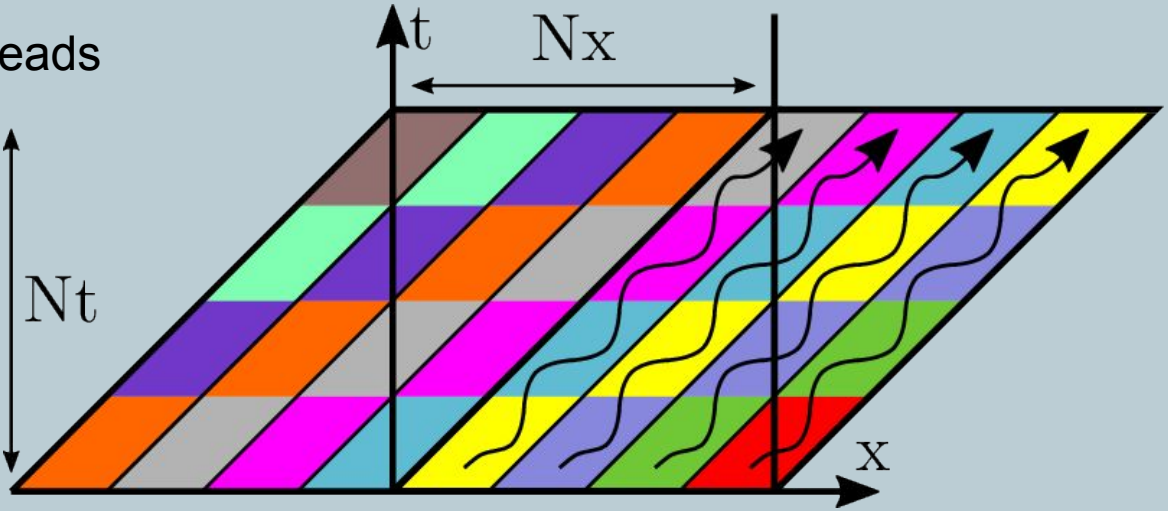


mc2series.com

Locally Recursive non-Locally Asynchronous (LRnLA) ***ConeFold*** Algorithm

Parallelism in space-time

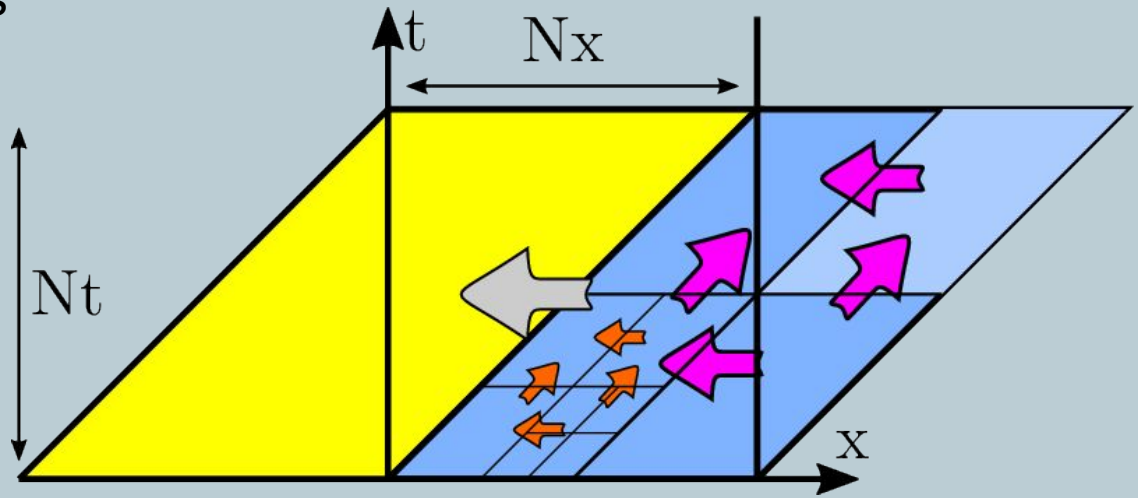
Implemented with POSIX threads



ConeFold Algorithm

The current topic:

- Implement the Lattice Boltzmann method with ConeFold
- Make periodic boundaries
- Use vectorization



mc2series.com

Lattice Boltzmann Method

$$\frac{\partial f}{\partial t} + \vec{v}\frac{\partial f}{\partial \vec{x}} = \Omega(f, f')$$

Discrete Distribution Function

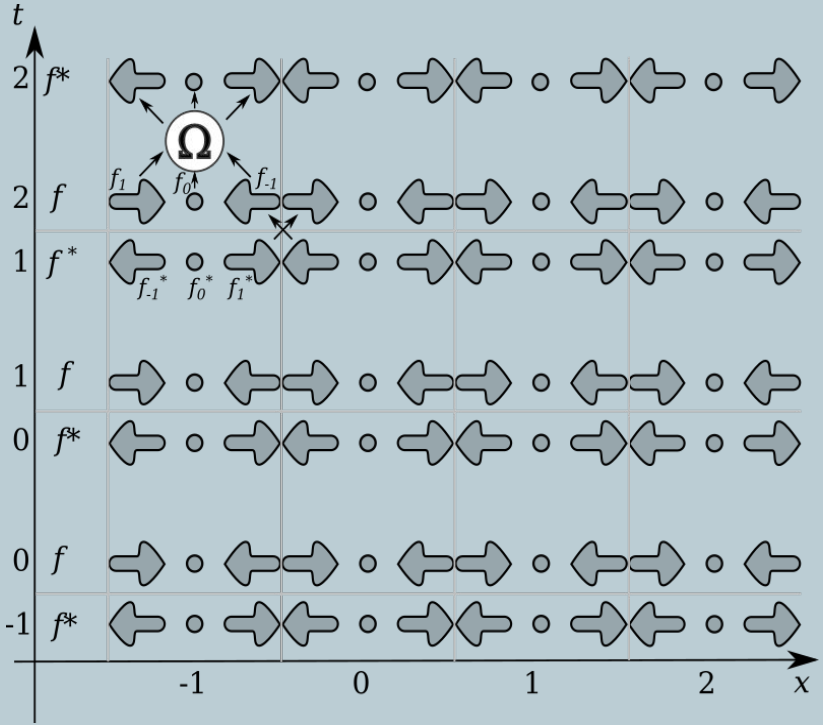
$$f(\vec{x}, \vec{v}, t) \rightarrow f_{ijk}(\vec{x}, t);\ i,\ j,\ k = -1, 0, 1;$$

Collision

$$f^*_{ijk}(\vec{x}, t) = \Omega(f_{i'j'k'}(\vec{x}, t));$$

Streaming

$$f_{ijk}(\vec{x} + \vec{c}\Delta t, t + \Delta t) = f^*_{ijk}(\vec{x}, t),\ \vec{c} = i, j, k$$



Lattice Boltzmann Method

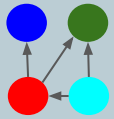
$$\frac{\partial f}{\partial t} + \vec{v}\frac{\partial f}{\partial \vec{x}} = \Omega(f, f')$$

Discrete Distribution Function

$$f(\vec{x}, \vec{v}, t) \rightarrow f_{ijk}(\vec{x}, t);\ i,\ j,\ k = -1, 0, 1;$$

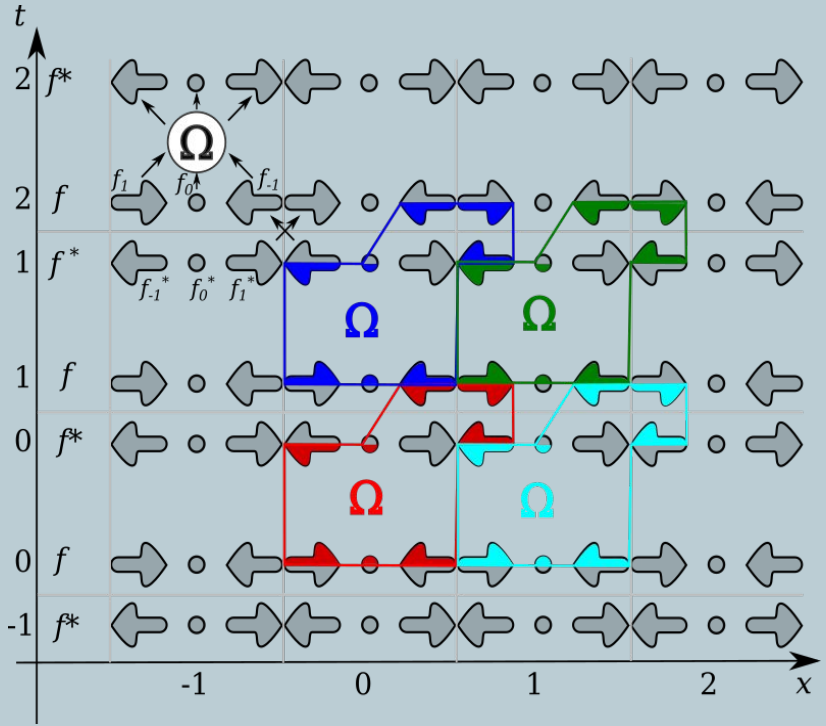
Collision

$$f^*_{ijk}(\vec{x}, t) = \Omega(f_{i'j'k'}(\vec{x}, t));$$



Streaming

$$f_{ijk}(\vec{x} + \vec{c}\Delta t, t + \Delta t) = f^*_{ijk}(\vec{x}, t),\ \vec{c} = i, j, k$$



LBM with ConeFold

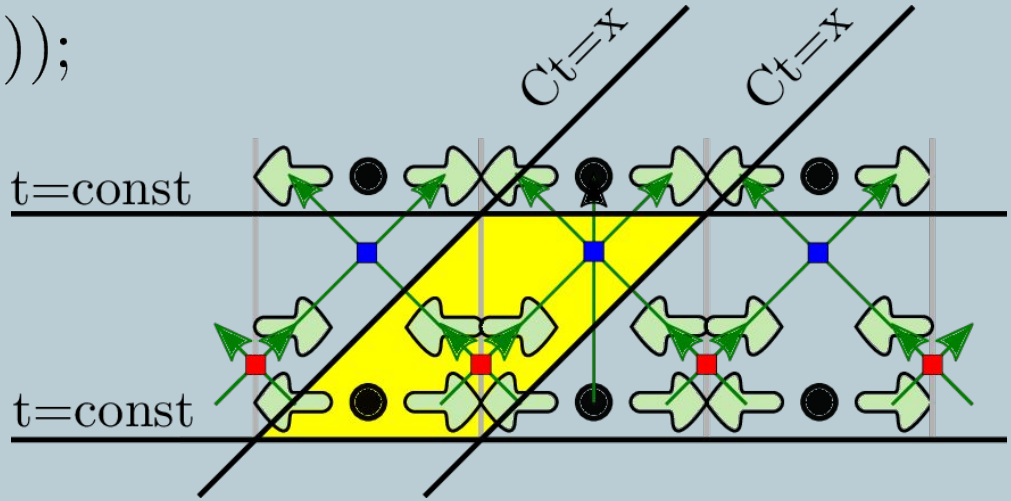
Swap between 2 cells

$$f_{i,j,k} \leftrightarrows f_{-i,j,k}$$

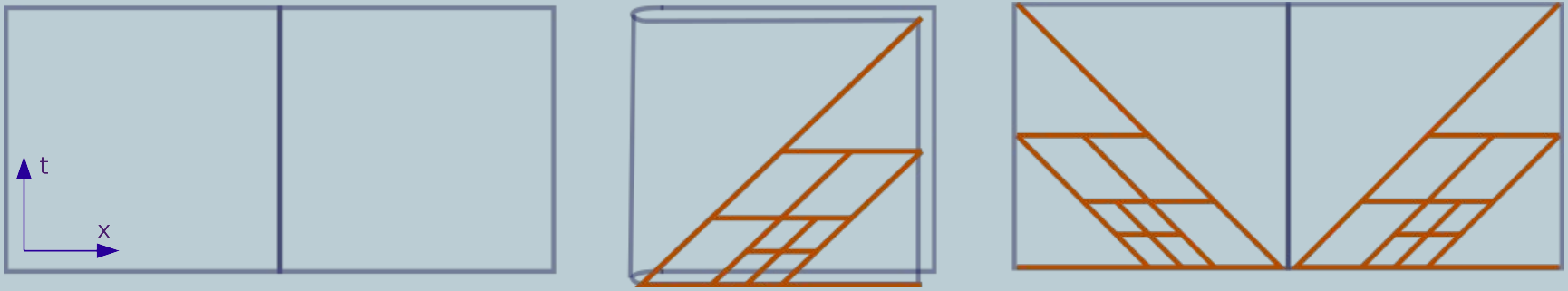
Collision in one cell

$$f^*_{ijk}(\vec{x},t) = \Omega(f_{i'j'k'}(\vec{x},t));$$

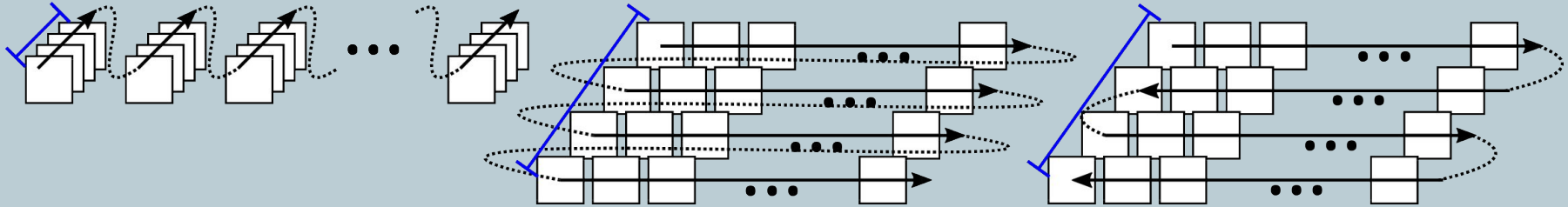
$$f_{i,j,k} \leftrightarrows f_{-i,j,k}$$



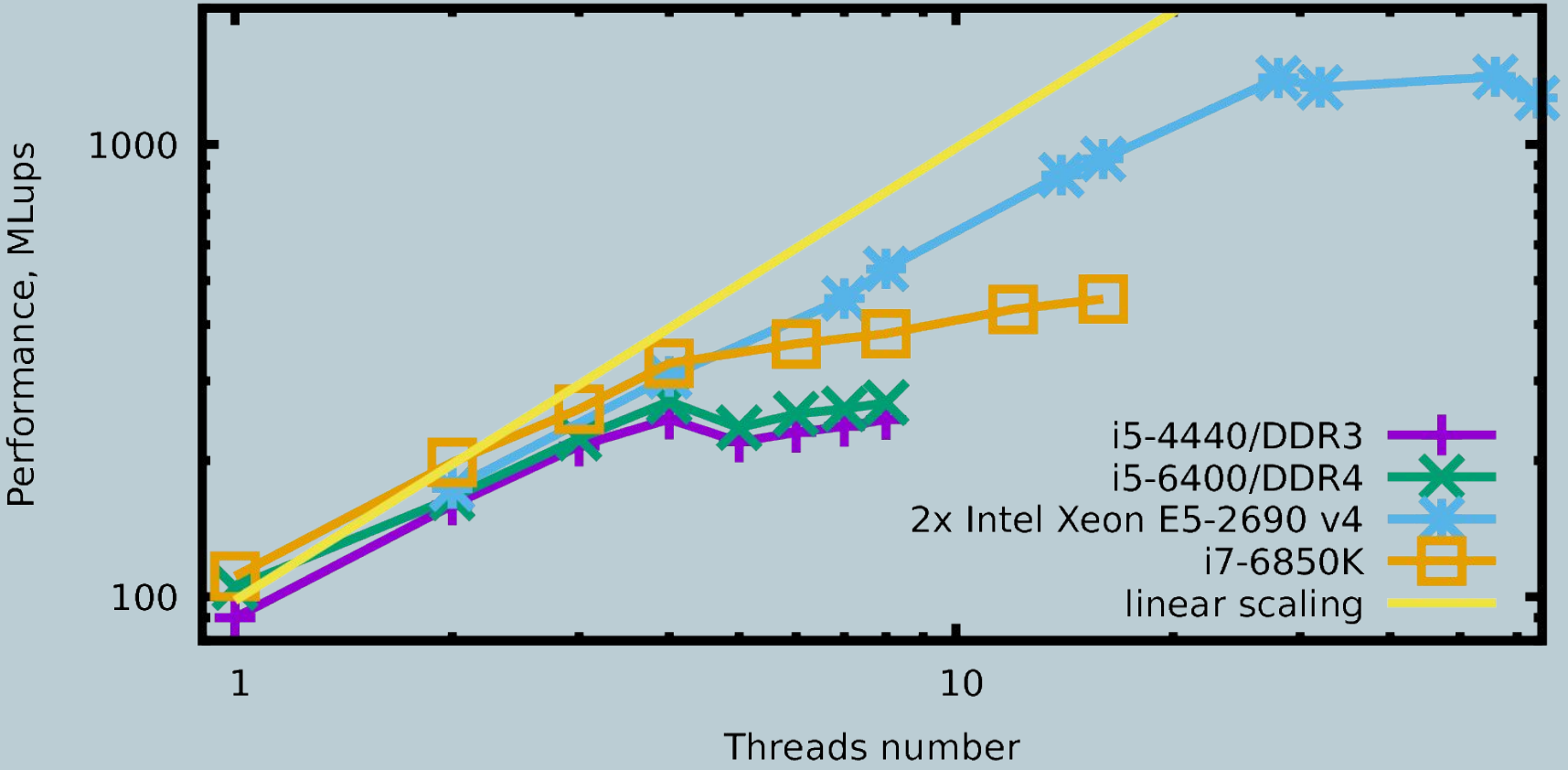
Non-local Vectorization



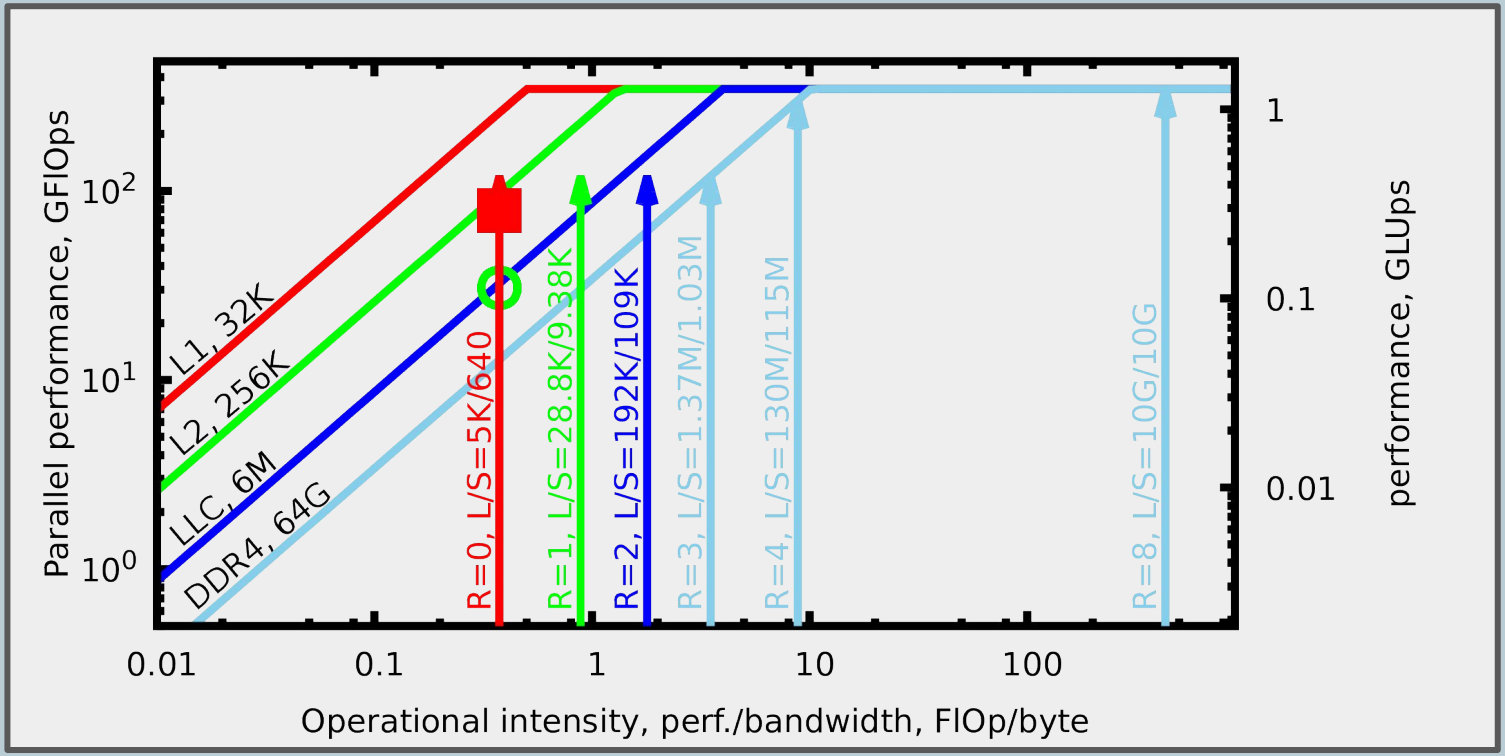
Local **Non-local** **Non-local mirrored**



Performance Results



Performance Results



Summary

The Locally Recursive non-Locally Asynchronous (LRnLA) algorithm ConeFold was implemented for the Lattice Boltzmann method on CPU.

A new non-local mirrored vectorization was used.

We have achieved a ~ 0.3 GLUps performance on a 4 core CPU for the D3Q19 Lattice Boltzmann method by taking an advanced time-space decomposition approach.

On one node of the K60 cluster (Intel Xeon E5-2690 v4, 8ch 256GB DDR4 RAM), we have achieved the performance up to 1.41 GLUps.

Conclusion

The algorithmic optimization allows to achieve significant performance boost for the LBM method.

The periodic boundary condition is possible for the time skewing approaches and comes with the efficient use of the vectorization mechanism available on CPU

LRnLA algorithms have been easily adapted for the LBM method and the target hardware.

Markus Wittmann, Erlangen

Name		D	K	Simulations-geometrie	Prozesssor	B_W	$\mathcal{S}$[a]	Performance e	Performance s[b]	Kommentare
HemeLB	[75]	D3Q15	BGK	Aterien	AMD Opteron 6267	31,0	1,3	18,0	23,4	
OpenLB	[60]	D3Q19	BGK	Atmungsprozess	Intel Xeon X3550	8,6	4,7	2,4	11,3	
Musubi	[89]	D3Q19	BGK	Spacer[d]	AMD Opteron 6276	31,0	1,3	45,8	59,5	AoS-Daten-Layout, OS-Pull
Walberla	[72]	D3Q19	TRT	Nischenströmung[e]	Intel Xeon E5-2680	40,0	1,0	90,0	90,0	SoA-Daten-Layout, AVX, OS-Pull+NTS+Split-Loops[f]
Obrecht	[157]	D3Q19	–[c]	Nischenströmung[e]	NVidia Kepler GK110	231,0	0,17	687,5	116,9	
Habich	[76]	D3Q19	BGK	Nischenströmung[e]	Intel Xeon E5-2680	40,0	1,0	98,0	98,0	AVX, OS-Pull+NTS+Split-Loops[f]
Habich-Tmp	[76]	D3Q19	BGK	Nischenströmung[e]	Intel Xeon E5-2680	40,0	1,0	107,0	107,0	AVX, Temporal-Blocking
Habich-GPU	[77]	D3Q19	BGK	Nischenströmung[e]	NVidia Tesla C2070	120,0	0,33	290,0	95,7	ECC deaktiviert
Palabos	[161]	D3Q19	BGK	Nischenströmung[e]	Intel Xeon X5570	24,0	1,7	33,5	57,0	
Pochoir	[196]	–[c]	–[c]	–[c]	Intel Xeon X5650	20,0	2,0	28,5	57,0	Cache-Oblivios-Algorithmus, Temporal-Blocking
Nguyen	[156]	D3Q19	–[c]	–[c]	Intel Core-i7 960/ 965 Extreme Edition	24,0	1,7	85,0	144,5	Temporal-Blocking

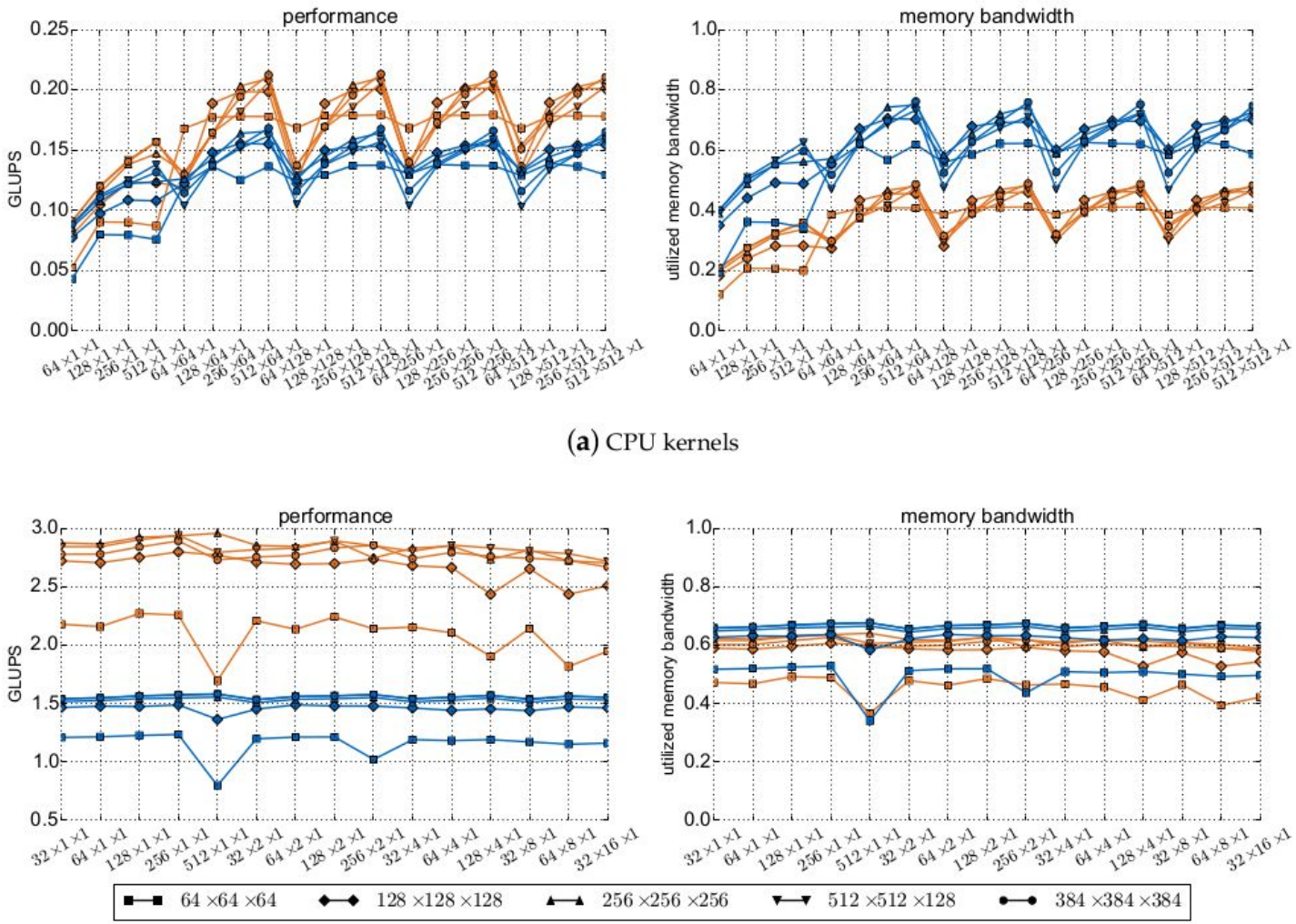
Tabelle 8.4: Übersicht über Lattice-Boltzmann-basierte Löser, deren Performance mit der des ILBDCs verglichen wird. Aufgeführt sind dabei die Parameter, wie etwa Diskretisierungsmodell (D) oder Kollisionsmodell (K), unter der die in der Literaturreferenz angegeben Performance (e) erreicht wurde. Die erreichte Performance wird auf die Performance eines Sockels des SandyBridge-Systems (Intel Xeon E5-2680) aus Kapitel 4.4.2 skaliert (s). Dies geschieht mit Hilfe des Skalierungsfaktors $\mathcal{S}$, der das Verhältnis der Speicherbandbreite des SandyBridge-Systems (ca. 40 GB/s) zu der des ursprünglichen Messsystems (B_W) angibt.

Piz Daint

Xeon
E5-2690v3 x 12

Tesla P100

Computation
2017, 5, 48;



(a) CPU kernels

(b) GPU kernels