ParCFD'2020
32^{nd} International Conference on Parallel Computational Fluid Dynamics
May-11-13 2020, Nice FRANCE

IMPLICIT PROPAGATION OF DIRECTLY ADDRESSED GRIDS IN LBM – PARCFD'2021

Adrian Kummerländer[*], **Mathias J. Krause**[*]

[*] Lattice Boltzmann Research Group
Institute of Applied and Numerical Mathematics 2
Karlsruhe Institute of Technology
Englerstr. 2, 76131 Karlsruhe, Germany
e-mail: adrian.kummerlaender@student.kit.edu
web page: https://www.lbrg.kit.edu/

Key words: Lattice Boltzmann Methods, Propagation patterns, Directly addressed grids, SIMD, GPU, Benchmark results

Abstract. Lattice Boltzmann methods are well suited to highly parallel computational fluid dynamics simulations due to their separability into a perfectly parallel collision step and a propagation step that only communicates within a local neighborhood. The implementation of the propagation step bounds both the maximum possible bandwidth-limited performance and places restrictions on the memory layout and usage of vector instructions. This contribution continues the work on implicit propagation pattern started by the A-A pattern [1] and its SSS formulation [2] to introduce a revert- and paddingless Periodic Shift (PS) pattern. Extensive benchmark results for SSS and PS on Intel and AMD CPUs including a Intel Xeon Phi processor using AVX2 and AVX-512 as well as GPUs using CUDA are provided.

1 IMPLICIT PROPAGATION

Implicit propagation is based on manipulating the SFC used for mapping spatial cell locations to directly addressed memory location. Such manipulation depends on a neighborhood property and usage of a Structure-of-Arrays memory layout for population data.

Definition 1.1 (Location invariance of neigborhood distance)**.** Let $x, y \in C$ be a pair of locations in cuboid C. The one-dimensional distance w.r.t. SFC $m_c : C \to \mathbb{N}_0$ is given by

$$\delta : C \times C \to \mathbb{Z},\ (x, y) \mapsto m_c(x) - m_c(y).$$

This distance is called *location invariant* iff

$$\forall \xi \in \mathbb{Z}^d\ \forall x, y \in \{x \in C | x + \xi \in C\} : \delta(x, x + \xi) = \delta(y, y + \xi).$$

This invariance of the one-dimensional or *in-memory* distance between all well-defined spatial neighbor locations is the essential property employed for implicit propagation. Curves for which this requirement holds enable streaming of all populations along their respective discrete velocity directions by translation of the starting point.

Definition 1.2 (Implicit Propagation)**.** Let C be a cuboid with memory bijection m_c fullfilling Definition 1.1, $\xi \in \mathbb{Z}^d$ a discrete velocity and t the current time. The memory access function

$$p_t : \mathbb{Z} \to \mathbb{R}$$

returns the current population values for all $x \in m_c(C)$ at time t and dummy values for $x \in \mathbb{Z} \setminus m_c(C)$. Propagation of a population at $x \in C$ in timestep t to $x + \xi \in C$ at time $t + 1$ is equivalent to

$$p_{t+1}(m_c(x + \xi)) = p_t(m_c(x)).$$

Due to invariance of the neighborhood distance the memory access function p_{t+1} can be defined as

$$p_{t+1} : x \mapsto p_t(\tilde{m}_c(x)) \text{ where } \tilde{m}_c : x \mapsto m_c(x) + \delta(x, x + \xi)$$

while being equivalent to propagation along ξ for all $x \in \{x \in C | x + \xi \in C\}$. Note that p_{t+1} is simply a *shifted* view of the original memory function p_t. The propagation is thus performed implicitly.

It can be shown that the Sweep SFC is the only discrete SFC that fullfills the location invariance of neighborhood distances. Shift-Swap-Streaming [2] (SSS) and Periodic Shift (PS) differ in the way that the indexing function is shifted resulting in different memory access patterns and performance results as well as isotropy properties. Existing patterns such as A-A [1] and A-B can also be formulated in this context.

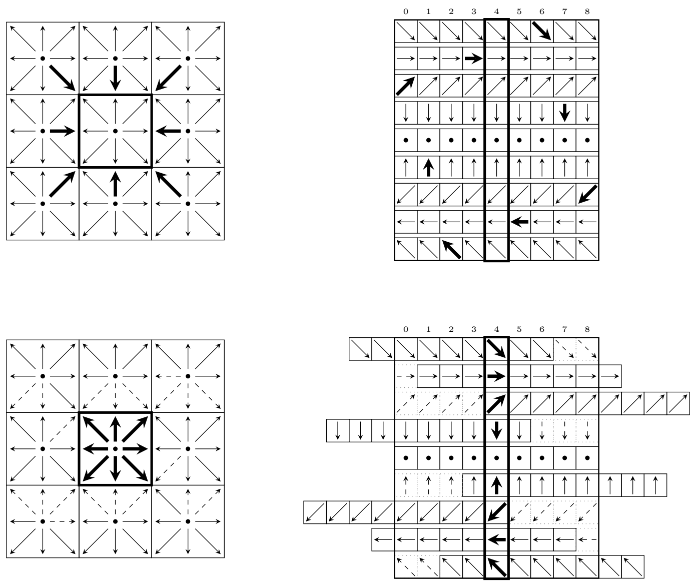

Figure 1: Propagation without data transfer by control structure update in PS on a D2Q9 lattice

2 PERIODIC SHIFT

The novel Periodic Shift pattern eliminates the need for padding and reverted stores present in SSS by viewing the individual population arrays as cyclic. The streaming step then consists only of rotating the arrays by their propagation distances. In this context the implementation challenge is to perform rotation of such cyclic arrays as efficiently as possible. Approaches using pre-computation of pointers to eliminate modulo operations and page table modifications to utilize virtual address translation (enabling the usage of SIMD intrinsics) are discussed and evaluated.

```
PeriodicShift($D, f_{\text{old}}$)
// Rotate the cyclic arrays along the distances given by discrete velocities
for $iPop \in \{1, \dots, D.q\}$
    $f_{\text{new}}[iPop] = \text{ROTATE}\,(f_{\text{old}}[iPop], \text{ShiftOffset}(D.c[iPop]))$
return $f_{\text{new}}$
```

3 BENCHMARKS

Detailed single-node CPU and GPU performance evaluations are performed using benchmark implementations based on SIMD intrinsics resp. CUDA and expression-level code optimization (CSE). All performance results are related to memory bandwidth measurements and achieve saturation up to 99%. Scalability benchmarks comparing to SWAP [3] use the OpenLB [4] framework which employs PS starting from version 1.4.

REFERENCES

[1] P. Bailey, J. Myre, S. Walsh, D. Lilja, M. Saar, Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors, in: 2009 International Conference on Parallel Processing, IEEE, Vienna, 2009, pp. 550–557. doi:10.1109/ICPP.2009.38.

[2] M. Mohrhard, G. Thäter, J. Bludau, B. Horvat, M. J. Krause, Auto-vectorization friendly parallel lattice Boltzmann streaming scheme for direct addressing, Computers & Fluids 181 (2019) 1–7. doi:10.1016/j.compfluid.2019.01.001.

[3] K. Mattila, J. Hyväluoma, T. Rossi, M. Aspnäs, J. Westerholm, An efficient swap algorithm for the lattice Boltzmann method, Computer Physics Communications 176 (3) (2007) 200–210. doi:10.1016/j.cpc.2006.09.005.

[4] M. J. Krause, A. Kummerländer, S. J. Avis, H. Kusumaatmaja, D. Dapelo, F. Klemens, M. Gaedtke, N. Hafen, A. Mink, R. Trunk, J. E. Marquardt, M.-L. Maier, M. Haussmann, S. Simonis, Openlb—open source lattice boltzmann code, Computers & Mathematics with Applications 81 (2021) 258 – 288. doi:https://doi.org/10.1016/j.camwa.2020.04.033.

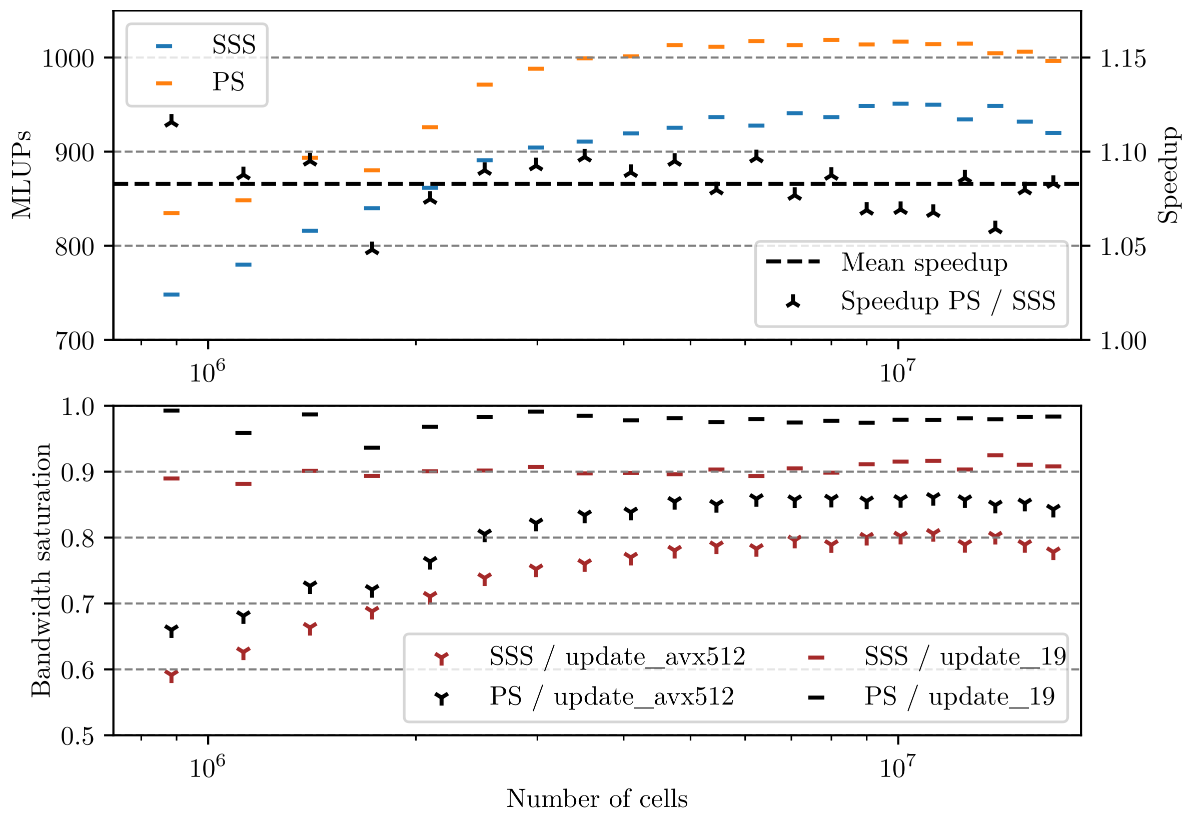

Figure 2: Performance for SSS and PS on Intel Xeon Phi using AVX-512 and OpenMP

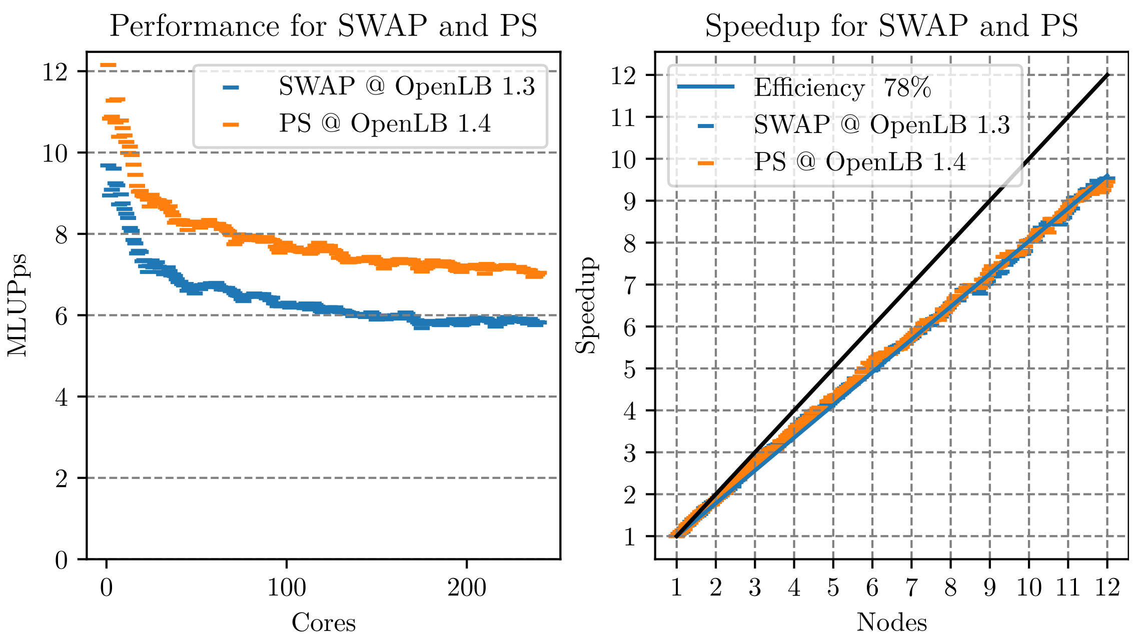

Figure 3: Speedup and efficiency for SWAP and PS using OpenLB and OpenMPI