NEW PARALLEL IN-PLACE UPDATE ALGORITHM FOR BETTER MEMORY USAGE IN 3D LATTICE BOLTZMANN METHOD

Minh Quan Ho*, Christian Obrecht† and Bernard Tourancheau*

* CNRS, LIG UMR 5217, Grenoble Alps University, F-38058 Grenoble, France.
e-mail: {minh-quan.ho, bernard.tourancheau}@univ-grenoble-alpes.fr

† Univ Lyon, CNRS, INSA-Lyon, Université Claude Bernard Lyon 1,
CETHIL UMR5008, F-69621 Villeurbanne, France.
e-mail: christian.obrecht@insa-lyon.fr

Key words: 3D Lattice Boltzmann method (LBM), In-place update, GPU, KNL, OpenMP, OpenCL

Abstract. The spatio-temporal dependencies of the LBM lead developers, for the sake of code simplicity, to allocate two instances of the computational array. The main drawback of this technique, which is usually known as two-lattice, is a doubled memory consumption. Different approaches, known as one-lattice algorithms, were introduced to reduce the memory footprint by working on only one lattice array. The fundamental challenge of such algorithm is that memory accesses (read and write) must be performed carefully on the same lattice buffer to enforce spatio-temporal dependencies. However, elaborate LBM boundary conditions raise difficulties when combined with existing one-lattice algorithms. In this paper, we present two new algorithms belonging to the one-lattice class simpler to implement and as efficient than other existing approaches. We also aim at demonstrating the interest of these new algorithms on future CPU-based many-core architectures with sophisticated memory hierarchies.

1 TWO-WALL PROPAGATION ALGORITHM

The main idea of two-wall is, before a lattice node is independently updated with its new state, its old state is copied out to a temporary buffer so that neighboring nodes can still access to these data to enforce spatio-temporal dependency. These buffers are presented as two 2D buffers, from which comes the name two-wall (Fig. 1).

Implementing complex boundary conditions with two-wall is trivial by replacing any access to z − 1 and z cells by past_wall and current_wall, respectively. Data locality is also fostered, as memory accesses from different cores have more chance to share and hit in cache (2D walls). In return, there are two performance challenges of the two-wall algorithm. The first is the data movement of 4q\(^1\) per node per time step versus 2q or 3q of two-lattice and others algorithms. Secondly, updating a wall requires two global

\(^1\)DdQq LBM stencil
barsriers, one after the copy operation (read-after-write) and one between subsequent walls, resulting a total of $2L_z$ barrier count for a grid $G$ of $L_z$ walls for each timestep.

2 THREE-WALL PROPAGATION ALGORITHM

We also introduce three-wall, an extended version of two-wall which breaks down the read-after-write dependency within each wall and reduces the number of barriers from $2L_z$ to $L_z$. This is possible by introducing a third wall buffer, called future_wall. The future_wall is used to store the pre-collision state of the wall $z+1$, during the update of the wall $z$. At the next iteration, there will be no need to perform the copy of the wall of interest, as this has been done before, though there is still need to copy the next wall to keep the process continuing. Copying the wall $z+1$ is independent from the computation of the wall $z$ and thus can be performed in parallel.

3 CONCLUSIONS

The two proposed algorithms are implemented in OpenMP for the shared-memory systems and OpenCL for the heterogeneous context. Our algorithms do not deliver the highest performance on CPU and GPU compared to other algorithms, but offer up to 1.5 times higher memory-efficiency (lattice-updates-per-second (LUPS) per byte) than the state-of-the-art two-lattice algorithm. Especially, on a high-core-count shared-memory architecture like KNL, three-wall gives concurrent performance and memory efficiency compared to AA-pattern [1], one of the most efficient propagation algorithms in literature. Employing three-wall on future many-core architectures which embed more local memory and private cache as well as the non-temporal streaming store instruction or DMA engines could lead to further performance improvements.

REFERENCES