Performance Optimization of D3Q19 Lattice Boltzmann Kernels on Intel® KNL

Ivan Girotto[*‡§], Sebastiano Fabio Schifano[†], Enrico Calore[†], Gianluca Di Staso[‡] and Federico Toschi[‡]

[*] The Abdus Salam, International Centre for Theoretical Physics
[†] University of Ferrara and INFN Ferrara
[‡] Eindhoven University of Technology
[§] University of Modena and Reggio Emilia
E-mail: igirotto@ictp.it, schifano@fe.infn.it, enrico.calore@fe.infn.it, g.di.staso@tue.nl, f.toschi@tue.nl

***Abstract*—This work discusses and assesses the impact of fundamental code optimization steps performed to maximize computing performances and memory throughput on Intel® Knights Landing (KNL) processor for Lattice Boltzmann (LB) applications. The benefits of using different memory data layouts is presented in regards to the most computationally intensive kernels of such applications, reporting performance results measured for the LBE3D code developed at the Applied Physics Department of the Eindhoven University of Technology, and run on a single KNL node for a common flow simulation case. We finally analyze and discuss the impact of different memory layouts on energy efficiency.**

***Index Terms*—LBE3D; KNL; Optimization; Energy Efficiency; Data Memory Layout; Vectorization; Performance Analysis.**

I. Introduction

The combination of multi-threaded programming and vectorization, combined with efficient use of different levels of memory hierarchy, are still considered to be the most relevant solution to achieve high computing performances on latest generations x86-64 processors. However, the majority of legacy scientific codes are not yet capable to exploit all these features, and optimized memory data layouts and access patterns, together with efficient use of a large number of threads and data vectorization, are necessary to obtain high computing performances.

The Lattice Boltzmann Method [1] (LBM) is widely used in computational fluid-dynamics to describe behavior of fluid flows, and nowadays is commonly applied in several science and engineering fields to accurately model single and multiphase flows, also using irregular boundary conditions. Furthermore, applications based on LBM are also employed to perform large scale simulations to study the dynamics and the behavior of fluid and gases, requiring a huge amount of computational resources. However, while these applications are renown to deliver good scaling performances on distributed systems enabling simulation of physical phenomena at high-resolution, it is not easy to achieve high computing efficiency even at level of single node. In fact, most of applications based on LBM are not engineered and optimized for modern CPUs based on multi-core architecture and using large vector unit, where data organisation of application domain plays a key role for enabling high computing efficiency.

In this work, we focus on the LBE3D code based on D3Q19 LBM model, and assess the impact on computing performances of several code optimization steps to maximize both the number of flops and the memory throughput on modern Intel® based many-core systems. In particular, we use several data layouts to store the data domain of the application, with the aim to find out a single memory layout that fits the computing requirements of several kernel routines of the code.

Performances in term of both computing and energy consumption of LBM applications have been studied in several works for different architectures [2]–[4]. Here, we follow a similar approach with the main difference that the analysis reported refers to a real case application for a 3-dimensional lattice.

The remainder of this paper is organized as follow: in Section II, we introduce the main features of the KNL processor, in Section III, we briefly describe the LBE3D code and the data layouts aimed to improve performances of LBM based applications, in Section IV, we present the results obtained measuring the LBE3D code performances on the KNL processors while in Section V, a short analysis on energy efficiency is reported.

II. The KNL processor

The results presented in the following sections are all obtained on a 64-cores Intel ® Xeon Phi™ CPU 7230 processor, commonly referred as KNL, running at 1.30 GHz and delivering a theoretical peak performance of about 3 TFlop/s in double precision. The KNL processor is equipped with 6 Double Data Rate fourth-generation (DDR4) channels, supporting 98 GB of synchronous Dynamic Random-Access Memory (DRAM) with a peak raw bandwidth of 115.2 GB/s and four high-speed memory banks based on the Multi-Channel DRAM (MCDRAM) that provides 16 GB of memory, capable to deliver an aggregate bandwidth of more than 450 GB/s. In this work, we only consider the Quadrant cluster configuration in which the 64-cores available are divided in four quadrants, each directly connected to one MCDRAM bank. MCDRAM on a KNL can be configured at boot time in Flat, Cache or Hybrid mode. The Flat mode defines the whole MCDRAM as addressable memory allowing explicit data allocation, whereas Cache mode uses the MCDRAM as a last-level cache. In this work, we used Intel® library for Message Passing Interface (MPI) to compile the LBE3D application. Vectorization is enabled at compile level using the -xMIC-AVX512 Intel® compiler option. Multi-thread version of LBE3D is implemented

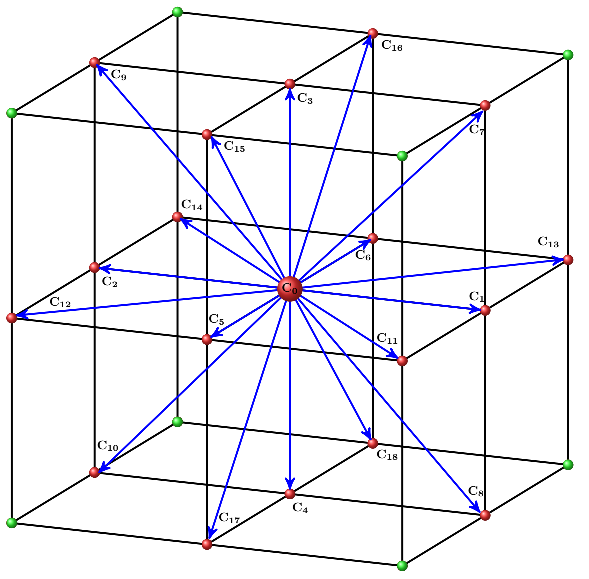

Figure 1. Schematic representation of the D3Q19 lattice employed in this work.

using OpenMP and enabled at compile time. Threads affinity at run time is obtained with ”KMP_AFFINITY=compact” and ”I_MPI_PIN_DOMAIN=socket” environment variables. Memory allocation for all results related to the KNL configured in Flat mode is made by using the `numactl -m 1 mpirun ./lbe3d` command.

III. LBE3D AND DATA LAYOUTS

The LBM is based on the synthetic dynamics of populations arranged at the edges of a discrete lattice. It is discrete in time, space and momenta, offering a large amount of easily identifiable parallelism while making it an ideal tool for investigating performances of modern systems for high-performance computing [5]–[7]. At each time step, populations are first moved from lattice-site to lattice-site applying the propagate operator, and then are modified through a collisional operator changing their values according to the local equilibrium condition. The computing pattern for the collision (`collide`) and the consequent propagation (`propagate`) within the lattice grid are renown main bottlenecks for the LB class of applications.

With this work, we transferred previous experiences on code optimization for LB class of applications implementing new data layouts on the LBE3D code, a LBM based application featuring a standard single relaxation time with the Bhatnagar-Gross-Krook (BGK) collision operator [8], which builds on top of a generic compile/profiling library (ftmake), currently maintained at the Eindhoven University of Technology, The Netherlands. More in particular, the LBE3D code implements a numerical scheme based on the LBM [9] and it has been used to perform simulations under a broad range of flow and fluid conditions [10]–[13].

Here, we focus on the D3Q19 LB stencil, a 3-dimensional model with a set of 19 population elements corresponding to (pseudo-)particles moving one lattice point away along all 19 possible directions. A schematic representation of the stencil pattern is shown in Figure 1, where the arrows represent the direction along which populations sitting at a site node are allowed to stream. All presented performance measurements are referred to a simple channel flow set-up as shown in Figure 2. A fluid between two parallel solid walls is put into motion by a homogeneous body force, and no-slip boundary conditions are applied at these walls, while periodic boundary conditions are applied along the two other directions.

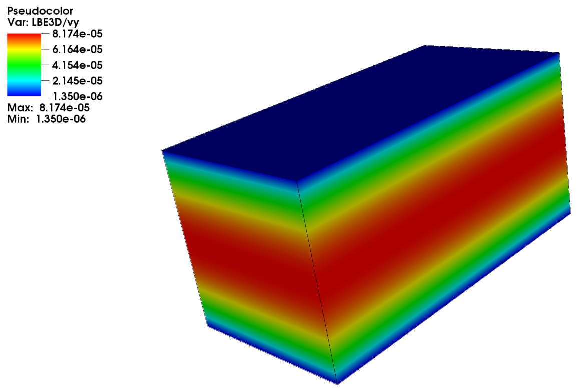

Figure 2. Velocity field along the forcing direction. Snapshot taken once the flow field reached the final steady state.

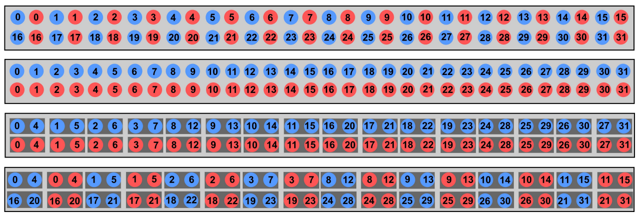

Figure 3. Top to bottom, AoS, SoA, CSoA and CAoSoA data memory layouts for a 4 x 8 lattice with two populations (red and blue) per site.

Many stencil based applications, including LBM, are commonly implemented using either Array of Structures (AoS) or Structure of Arrays (SoA) data layouts. In the AoS approach, originally used in the LBE3D code too, population elements of each lattice site are stored contiguously in memory. Therefore, the AoS structure is constructed as a 3-dimensional lattice where each element is composed by N population values (NPOP). On the other hand, LB based applications adopting the SoA schema allocate the 3-dimensional lattice as a collection of NPOP arrays, each storing for every site a single element population value. However, none of those two data layouts have been demonstrated to be ideal for LB based applications since, while the AoS delivers better performances for the `collide` kernel, it lacks in memory bandwidth if compared with SoA when considering the `propagate` kernel.

Alternate data layouts aim to improve the overall performances of LB based applications, and in Figure 3 we show a graphical representation for a sample lattice of 4 x 8 using two populations per site (memory addresses increase left-to-right top-to-bottom). In summary, beside the AoS and SoA layout, we have two new layouts called CSoA and CAoSoA. The CSoA layout is an extension of the SoA where VL lattice-site data at distance L/VL (L dimension of major order store) are clustered in consecutive elements for each population array, with VL equal to the number of double precision elements that can be stored in the vector register available on the given architecture. This layout keeps the data properly aligned in memory and allows to vectorize the steps of `propagate` kernel. The CAoSoA structure is a mix between CSoA and AoS, and allows to exploit the benefit of the VL clusterization of lattice sites element as introduced by the CSoA schema but with the benefit of higher locality in regards to the populations. For this reason, this layout may deliver better overall performances for the `collide` kernel. In Figure 3, each dark grey-box is a cluster with VL=2 for both the CSoA and the CAoSoA data layouts.

IV. Analysis of Results

In this Section, we measure the impact of the different memory layouts in terms of computing while an analysis in term of energy consumption is provided in the following paragraph. We first focus and analyze performance for the `propagate` and `collide` kernels, and then we assess the impact on the whole LBE3D code.

As mentioned earlier, LBM is characterised by a phase of propagate, where populations move from lattice-site to lattice-site, describing the flows momentum. Kernels implementing this phase are, in particular, memory bounded because the movement is practically implemented as the copy of a single data (double precision floating point number) from one location to another location in memory. Therefore, a key aspect to achieve maximum speed is to describe this operation with a memory access pattern that can exploit the maximum memory bandwidth. Clustered data layouts allow to vectorize such operation, whether data are aligned in memory, as the compiler replaces the scalar operation of copy with a copy operation on registers capable of multiple elements of the same kind (vector registers). In Figure 4, we report the measured performances of memory bandwidth obtained by the `propagate` kernel on a KNL node configured in Flat mode. The CSoA version allows to achieve almost 350 GB/s memory with a significant improvement in performance if compared with the canonical AoS or SoA approaches. We can confirm that also for the cases of 3-dimensional lattices the CSoA version allows for the `propagate` kernel to achieve the highest value of memory bandwidth if compared with other analysed data layouts. It is relevant to underline how in most cases the memory bandwidth saturates at 64 threads (a single hardware thread per core).

The measured memory bandwidth drops if considering larger lattices with a data domain unable to fit in the 16 GB/s of the MCDRAM. In Figure 5, data are obtained with the KNL configured in Cache mode and the real peak performance achieved by the CSoA data layout is of about 80 GB/s, with a factor of 4x reduction if compared with the memory bandwidth obtained when data fit into the MCDRAM. Moreover, the intensive use of the DRAM memory squeezes the performance gap among the various data layouts such that CSoA becomes only 10% better of the SoA and comparable with the CAoSoA versions.

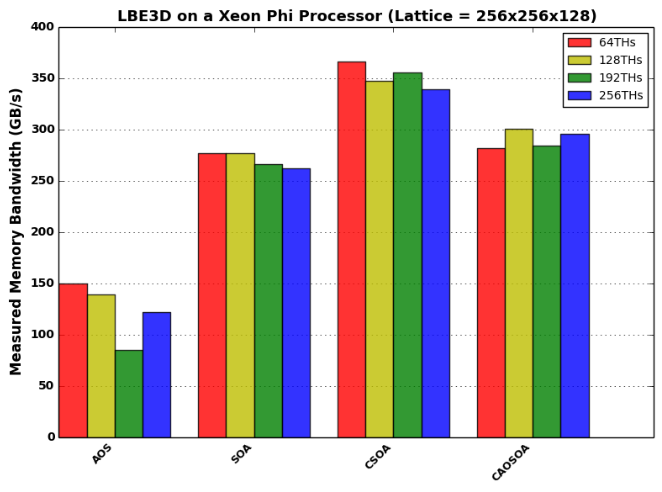

Figure 4. Measured memory bandwidth for the `propagate` kernel using different data layouts: AoS, SoA, CSoA and CAoSA. KNL is configured in flat mode.

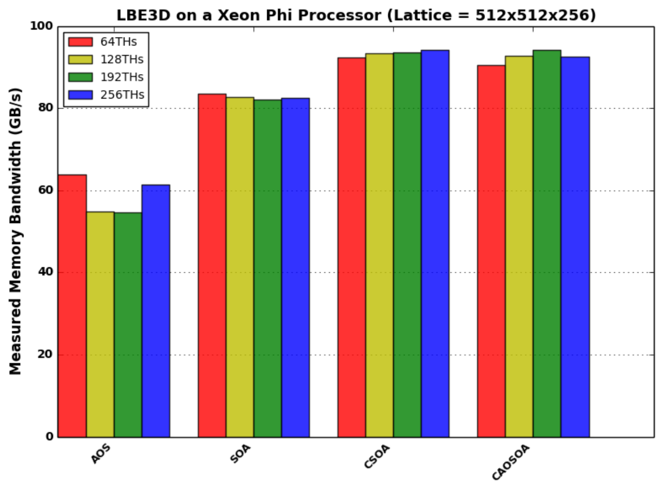

Figure 5. Measured memory bandwidth for the `propagate` kernel using different data layouts: AoS, SoA, CSoA and CAoSA. KNL is configured in cache mode.

The other important phase of LBM is the collision phase. It is implemented as the `collide` kernel which is generally considered compute bounded because all discretised quantities such as forces, velocities and densities are per site computed with high data locality. For most of those quantities the lattice elements are read from the input lattice and written to the output lattice contiguously, but for the computation of the density, and velocity, which requires accessing all population elements for each site. While this is a cache friendly operation for the AoS scheme, due to data locality (all populations elements are stored contiguously), it is not for data layouts where per-site population are scattered in memory, such as SoA and CSoA. However, as per the CSoA version the speedup achieved by vectorized operations on clustered data helps to overrun this problem. The mixed schema of the CAoSoA is expected to take advantage of accessing contiguous population elements while working on clustered data.

In Figure 6, the measured value of peak performance is

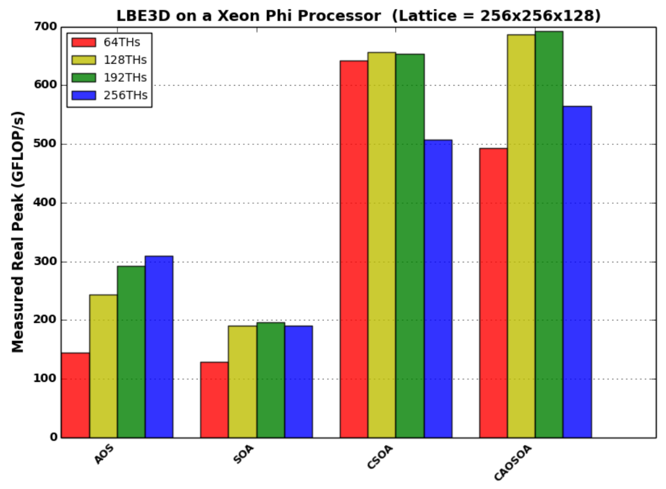

Figure 6. Measured real peak of performances for the `collide` kernel using different data layouts: AoS, SoA, CSoA and CAoSA. The KNL is configured in Flat mode

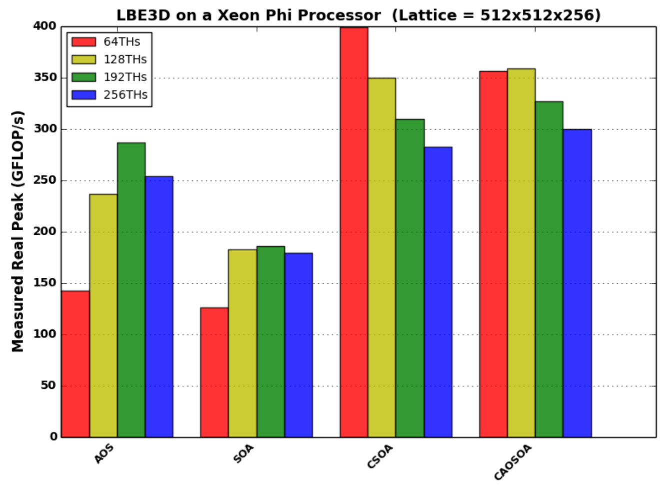

Figure 7. Measured real peak of performances for the `collide` kernel using different data layouts: AoS, SoA, CSoA and CAoSA. The KNL is configured in Cache mode

reported. In flat mode for a medium size grid of 256x256x128 we measure a peak performance of about 700 GFLOP/s corresponding to around the 25% of the nominal peak: approximately half of the value reported on the Nov 2018 list of the TOP500 for KNL based architectures running the High Performance Computing Linpack Benchmark. CSoA and CAoSoA are confirmed to outperform canonical AoS and SoA data layouts by a factor between 2x to 3x. However, when increasing the lattice size while switching to the Cache mode configuration, the performance gap between the various data layouts drops significantly also in the case of the `collide` kernel. As we report in Figure 7, when intensively using DRAM the real performance peak is reduced by a average factor 2x if compared with the same kernel running on KNL in Flat mode, and the gaps between the different data layouts drops between 1.5x to 2.5x.

In the following we present the impact of the different data layouts in regards to the LBE3D application. We concentrate on performances of the main LB loop (Figure 8), computationally the most significant part of the LBE3D application. Indeed, initialization and finalization phases, as well as the time spent on I/O operations are disregarded because becoming irrelevant at the increasing of the number of time steps (in production). In the particular case of the the I/O, it remains irrelevant as long as the frequency of I/O operations is kept low in regards to the number of LB loops (user driven).

Other than the `propagate` and the `collide` kernels also boundaries update is considered as well as the update of the sites near to the walls (see par. 4). We also include the analysis of performances for the fused version of the code. As illustrated in Figure 8 by fused we mean a version of the LBE3D were the `propagate` and the `collide` kernels are nested within the same loop that parses all the populations of the 3-dimensional lattice. In the case of 3-dimensional lattices the number of elements per dimension is limited because considering regular lattices the memory requirement grows exponentially at the increasing of the number of elements per dimension. For instance a lattice dimensions of 512^3 requires 20GB of memory which goes much beyond the memory available on the KNL's MCDRAM. At the same time, reducing the number of elements per dimension includes the risk of unbalance at the increasing of the number of threads while the vectorization benefits of irregular grids larger on the most inner dimension. To reduce the problem of threads imbalance we also introduced the OpenMP `collapse` clause distributing threads workload among the two outermost dimensions of nested loops over a 3d-lattice for all significant sections of the LBE3D code. However, this only minimally reduced the effect of imbalance when increasing the number of threads.

```
1: for all time step do
2:     < Set boundary conditions >
3:     for all lattice site do
4:         < Move >
5:     for all lattice site do
6:         < Collide_Fused >
7:     end for
8: end for
```

```
1: for all time step do
2:     < Set boundary conditions >
3:     for all lattice site do
4:         < Move_Collide_Fused>
6:     end for
7: end for
```

Figure 8. On the left, a pseudo-code description of the main loop of LB based application. On the right, a representation of the fused version of the two main kernels.

In Figure 9, performance results for the LBE3D case are reported. It is evident the benefit of using the clustered versions CSoA or CAoSoA if compared with more canonical AoS or SoA data layout. We have measured this impact to be a factor of 2x to 3x depending by the lattice dimension with the KNL configured in Flat mode. Achieved vectorization by the fused version of LBE3D is shown in Table I where we report the Vector Processing Unit (VPU) activity as the measured ratio of the two Vtune's counters UOPS_RETIRED.SCALAR_SIMD and UOPS_RETIRED.PACKED_SIMD. The final value is given by the ratio between the number of vector operations the core performed (PACKAED_SIMD) to the sum of all operations (SCALAR_SIMD and PACKED_SIMD) as properly described in [14].

TABLE I. VPU USAGE MEASURED BY THE INTEL® VTUNE FOR FUSED LBE3D

data layouts	*Vector VPU Intensity*
AoS	20%
SoA	20%
CSoA	100%
CAoSoA	100%

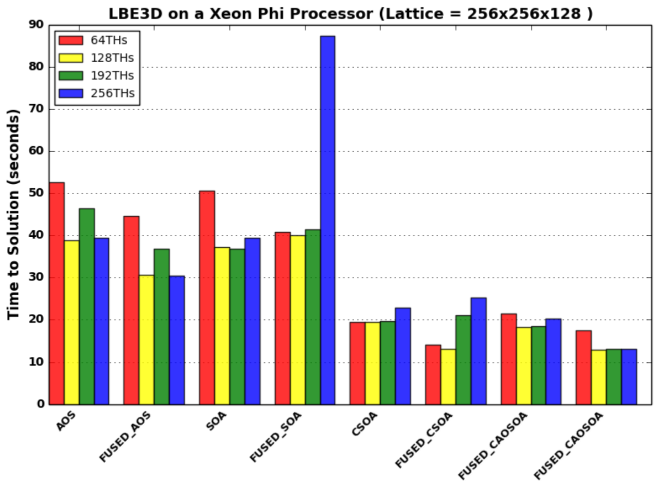

Figure 9. Time to solution for 1000 time steps of the LB main loop for the LBE3D application. Performances are reported across the multiple data layout presented, at the increasing of the number of hardware threads active per core.

Increasing the number of threads beyond 2 hardware threads per core does not provide any performance improvement, actually Figure 9 reports that with the KNL configured in Flat mode the best results are achieved using 2 hardware threads per core among all versions. In general, beyond the 2 hardware threads per core there are some fluctuations within the 20% of the total time of the main LBE3D loop (see Figure 8) but there is an unexpected degradation of performances, almost by a factor 2x, for both the fused SoA and CSoA versions at 256 threads. By deeper analyzing this phenomena with Vtune we saw that the peak is associated to a strong increase of misses at Level-2 (L2) of the translation lookaside buffer (TLB), measured monitoring the counter MEM_OUPS_RETIRED.DTLB_MISS_LOADS (see [14] for better details). At the same time, a deeper profiling has shown that for this particular cases the most time consuming function becomes the routine kmp_flag_64::wait, from the limiomp5.so library which includes the Intel® implementation of OpenMP. Despite we still cannot explain this high number of TLB misses for this particular case, we can state how this configuration drastically slows down due to a problem of threads unbalancing that coincides with a peak of the number of L2 TLB misses.

The benefit of the fused version is generally 1.5x with respect to the canonical version which uses two separate kernels for `propagate` and `collide`. In Figure 10, we report the profiling breakdown for 2 hardware threads per core (128 threads in total). The profiling chart shows the impact of the various data layouts for the `propagate` and `collide` kernel in regards to the whole LB main loop. The impact of the fused version of the kernel is also well in evidence.

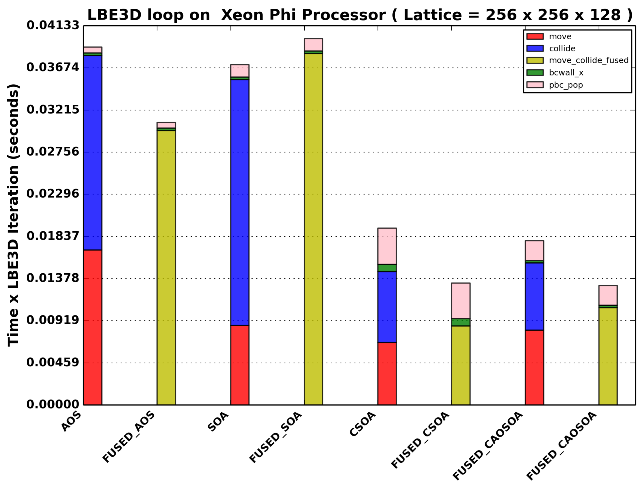

Figure 10. Profiling breakdown for a single time step of the LB main loop for the LBE3D application. Data are reported using the OpenMP version of the LBE3D application using 2 hardware thread per core.

V. Energy Efficiency

We now consider energy efficiency for the LBE3D across the multiple data layouts presented. We use data from the Running Average Power Limit (RAPL) register counters available in the KNL read through the custom library developed in [15]. In Figure 11, we show the measured values of energy consumption (Joule) for the LBE3D application respectively for the processor and the off-chip DRAM memory.

The DRAM energy consumption is lower as the KNL is configured in Flat mode and during the simulation data are all stored into the MCDRAM. Indeed, energy consumption for the DRAM memory only registers the value in the state of idle while for the MCDRAM is accounted within the CPU (on cheap memory).

What is relevant to notice is that despite the CSoA and CAoSoA data layouts are expected to stress the CPU system more than the AoS and SoA (higher utilization of the VPU), we can assume the absorbed power remains approximately constant when considering different data layouts. Indeed, it is evident how the energy consumption remains mostly proportional to the time to solution such that Figures 11 and 9 are comparable and showing a similar trend: usage of the CSoA data layout brings a factor from 2x to 3x advantage both in term of time to solution, and energy to solution.

VI. Conclusion

In this contribution, we have presented the impact on computing performances and energy consumption of different data layouts for the case of the LBE3D code.

Best improvements are given by the newly introduced data layouts (CSoA and CAoSoA) on the KNL, when the data domain fits the MCDRAM memory capability of 16 GB. In this case, the LBE3D application shows best performances setting 2 hardware threads per core (128 threads in total), while exhibits a problem of load unbalancing when increasing the number of hardware threads per core beyond 2. Considering the whole main LB loop in terms of time to solution, the

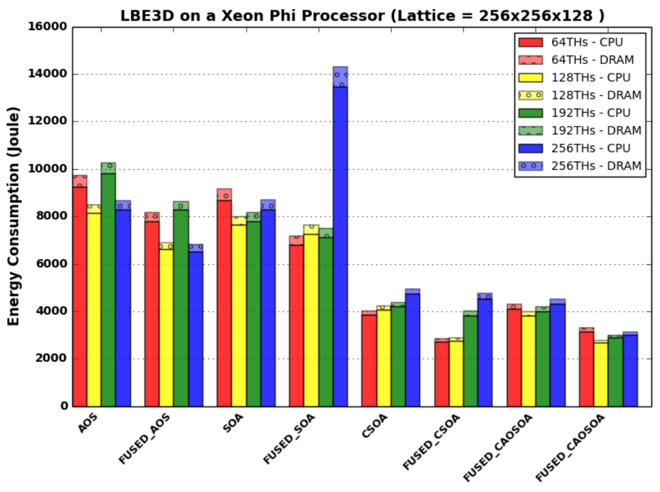

Figure 11. Energy consumption profiling of the LBE3D application.

CSoA and the CAoSoA data layouts are comparable offering an overall better performance corresponding to approximately 2-3x faster compared to the other layout schemes: AoS and SoA. In particular, the CSoA shows a peak performance of about 700 GFlop/s for the `collide` kernel, and 350 GB/s in terms of memory bandwidth for the `propagate` kernel. Larger lattices that do not fit the MCDRAM requires access to the DRAM memory which in Cache mode provide an average time to solution lower by approximately a factor 2x. This bottleneck put at the same level the performances exhibit by all version of the code.

Analysis on the LBE3D in regards to energy efficiency shows that the both CSoA and CAoSoA data layouts are more efficient because delivering faster time to solution, although a slightly higher average power drain is measured due to a more intense utilization of the on-chip system.

The optimization steps presented here for the case of the LBE3D are quite general and can also be applied to other LB production codes. Moreover, we expect that similar performance improvements can also be achieved on other kind of processor based on large vector registers. In fact, the optimizations shown are mainly targeted to make efficient use of VPU capable to perform high number of operations per clock cycle.

In future works, we will keep on investigating how the proposed data layouts perform on current and next generation of computer architectures for high-performance computing, including accelerators based platforms.

Acknowledgment

We would like to thank CINECA, INFN and The University of Ferrara for access to their HPC systems.

References

[1] S. Succi, *The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond*. Clarendon Press, 2001, ISBN: 978-0-19-850398-9.

[2] E. Calore, N. Demo, S. F. Schifano, and R. Tripiccione, "Experience on Vectorizing Lattice Boltzmann Kernels for Multi- and Many-Core Architectures," in *Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers, Part I*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 53–62. doi: 10.1007/978-3-319-32149-3_6

[3] E. Calore, A. Gabbana, S. F. Schifano, and R. Tripiccione, "Early experience on using knights landing processors for lattice boltzmann applications," in *Parallel Processing and Applied Mathematics: 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017*, ser. Lecture Notes in Computer Science, vol. 1077, 2018, pp. 1–12. doi: 10.1007/978-3-319-78024-5_45

[4] E. Calore, A. Gabbana, J. Kraus, S. F. Schifano, and R. Tripiccione, "Performance and portability of accelerated lattice Boltzmann applications with OpenACC," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 12, pp. 3485–3502, 2016. doi: 10.1002/cpe.3862

[5] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick, "Optimization of a Lattice Boltzmann computation on stateof-the-art multicore platforms," *Journal of Parallel and Distributed Computing*, vol. 69, pp. 762–777, 2009. doi: 10.1016/j.jpdc.2009.04.002

[6] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. A. Yelick, "Lattice Boltzmann simulation optimization on leading multicore platforms," *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–14, 2008. doi: 10.1109/IPDPS.2008.4536295

[7] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, and E. Kaxiras, "A flexible high-performance lattice Boltzmann gpu code for the simulations of fluid flows in complex geometries," *Concurrency Computat.: Pract. Exper.*, pp. 22: 1–14, 2009. doi: 10.1002/cpe.1466

[8] P. L. Bhatnagar, E. P. Gross, and M. Krook, "A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems," *Phys. Rev.*, vol. 94, pp. 511–525, 1954. doi: 10.1103/PhysRev.94.511

[9] R. Benzi, S. Succi, and M. Vergassola, "The lattice Boltzmann equation: theory and applications," *Physics Reports*, vol. 222, no. 3, pp. 145 – 197, 1992. doi: 10.1016/0370-1573(92)90090-M

[10] P. Perlekar, R. Benzi, H. J. H. Clercx, D. R. Nelson, and F. Toschi, "Spinodal decomposition in homogeneous and isotropic turbulence," *Phys. Rev. Lett.*, vol. 112, p. 014502, Jan 2014. doi: 10.1103/PhysRevLett.112.014502

[11] A. Scagliarini, H. Einarsson, A. Gylfason, and F. Toschi, "Law of the wall in an unstably stratified turbulent channel flow," *Journal of Fluid Mechanics*, vol. 781, p. R5, 2015, doi: 10.1017/jfm.2015.498.

[12] G. Di Staso, S. Srivastava, E. Arlemark, H. Clercx, and F. Toschi, "Hybrid lattice Boltzmann-direct simulation Monte Carlo approach for flows in three-dimensional geometries," *Computers & Fluids*, 2018. doi: 10.1016/j.compfluid.2018.03.043

[13] A. Gupta, H. Clercx, and F. Toschi, "Simulation of finite-size particles in turbulent flows using the lattice Boltzmann method," *Communications in Computational Physics*, vol. 23, no. 3, pp. 665–684, 2018. doi: 10.4208/cicp.OA-2016-0268

[14] J. Jeffers, J. Reinders, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming*. Morgan Kaufmann, Jun. 2016, ISBN: 978-0-12-809194-4.

[15] E. Calore, A. Gabbana, S. F. Schifano, and R. Tripiccione, "Evaluation of dvfs techniques on modern hpc processors and accelerators for energy-aware applications," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, pp. 1–19, 2017. doi: 10.1002/cpe.4143