Performance of ODROID-MC1 for Scientific Flow Problems

Andreas Lintermann[a,b,*], Dirk Pleiter[c], Wolfgang Schröder[a,b]

[a]*Institute of Aerodynamics and Chair of Fluids Mechanics, RWTH Aachen University, Wüllnerstr. 5a, 52062 Aachen, Germany*
[b]*Jülich Aachen Research Alliance, High Performance Computing (JARA-HPC), RWTH Aachen University, Seffenter Weg 23, 52074 Aachen, Germany*
[c]*Jülich Supercomputing Centre, Forschungszentrum Jülich, Wilhelm-Johnen-Straße, 52425 Jülich, Germany*

Received 18 September 2018, Revised 21 November 2018, Accepted 27 December 2018, Available online 4 January 2019.
https://doi.org/10.1016/j.future.2018.12.059

Abstract

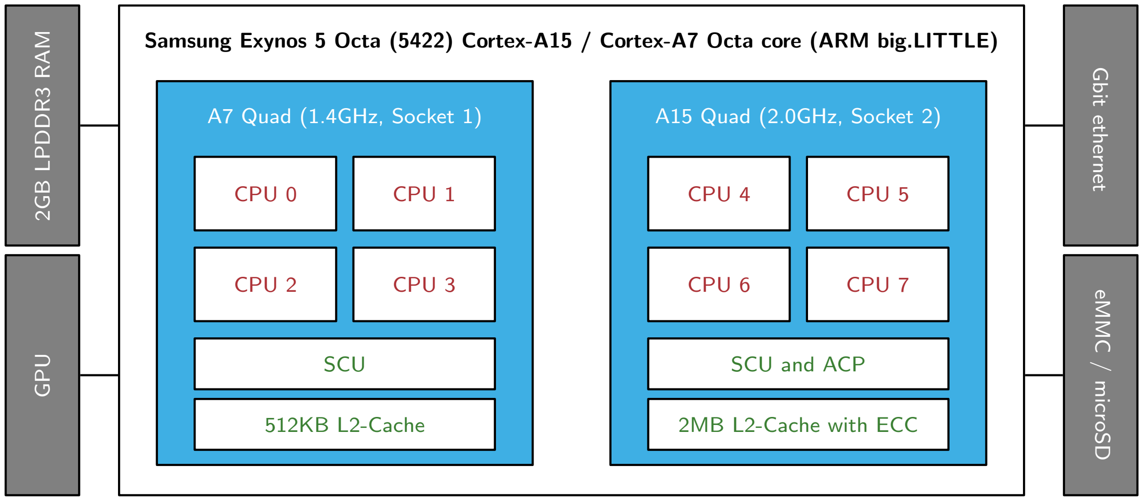
In late 2017, Hardkernel released the ODROID-MC1 cluster system, which is based on the ODROID-XU4 single-board computer. The cluster consists of four nodes, each equipped with a Samsung Exynos 5 Octa (5422) CPU. The system promises high computational power under low energy consumption. In this paper, the applicability of such a systems to scientific problems is investigated. Therefore, flow computations using a lattice-Boltzmann method are employed to evaluate the single core, single node, and multi-node performance and scalability of the cluster. The lattice-Boltzmann code is part of a larger simulation framework and scales well across several high-performance computers. Performance measurement results are juxtaposed to those obtained on high-performance computers and show that the ODROID-MC1 can indeed compete with high-class server CPUs. Energy measurements corroborate the ODROID's energy efficiency. Its drawbacks result from the limited amount of available memory, the corresponding memory bandwidth, and the low-performing Cortex A7 cores of the big.LITTLE architecture. The applicability to scientific applications is shown by a three-dimensional simulation of the flow in a slot burner configuration.

Keywords: *Single-board computer, Odroid, Lattice-Boltzmann method, High-performance computing, Performance analysis, Power consumption*

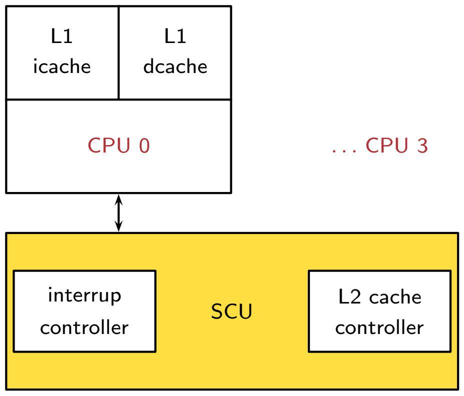
1. Introduction

Single-board computers (SBCs) have become popular in the maker community. Especially their low price to compute power ratio and their low power consumption make them attractive for, e.g., home automation, gaming, or media server applications. A wide range of SBCs is available on the market. Depending on the targeted application and the end user's flavor, SBCs are shipped with a variety of CPUs, GPUs, memory, and interfacing devices such as USB, ethernet, WiFi, and HDMI adapters, or GPIO pins for measuring and controlling. Most SBCs have in common that they are equipped with low-power ARM-based CPUs. Recent CPUs can be subdivided into 32- and 64-bit systems. ARMv7

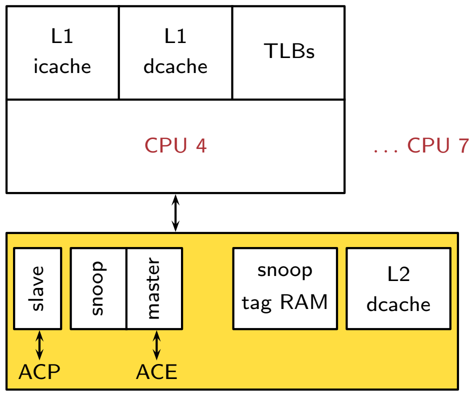
[*]Corresponding author. Tel.: +49 241 80 90419 / fax: +49 241 80 92257.
Email address: A.Lintermann@aia.rwth-aachen.de (Andreas Lintermann)



(a) The ODROID-XU4 consists of 2 sockets equipped with four-core ARM Cortex-A7 and ARM Cortex-A15 CPUs.



(b) Sketch of the ARM Cortex-A7 CPU.



(c) Sketch of the ARM Cortex-A15 CPU.

Figure 1: Sketch of the hardware layout of the ODROID-XU4.

CPUs [1] are 32-bit systems and are frequently found on SBCs in their ARM-Cortex-A implementation such as Cortex-A$\{5, 7, 8, 9, 15, 17\}$ system on a chip (SoC). In contrast, ARMv8 CPUs [2] such as ARM-Cortex-A$\{53, 57, 73, 75\}$ support 64-bit. The amount of memory, which is either of DDR2, DDR3, or LPDDR3 type, is usually small, i.e., in the $512 MByte$ to $4 GByte$ range. It is often the amount of memory, which determines the price of the whole system. SBCs with larger memory usually base on `x86` architectures. On-board GPUs stem from ARM, e.g., Mali-T628, Mali-400, Mali-450MP, Mali-450MP2, or Mali-450MP4 are often found. Ethernet adapters usually feature $10/100 Mbit$ or $GBit$ ethernet and storage is either wired on-board or can be attached via eMMC modules, SD, or microSD cards. The availability of different operating systems such as Linux (Debian, Android, Ubuntu, Raspbian, and so forth) or Microsoft Windows makes SBCs easy to configure, program, and use.

The most prominent SBCs are probably the Raspberry Pi-$\{1, 2, 3\}$ [3], its variants Orange Pi, Banana Pi, and the Cubiboards [4], to name just a few. The evolution of these SBCs bases on changes of the SoC and the CPU, which includes an increase of the CPU clocking, the number of available cores, and the amount of available memory. In 2015, Hardkernel released the ODROID-XU4 system [5], which is equipped with Samsung Exynos 5 Octa (5422) Cortex-A15 [6] $2GHz$ and Cortex-A7 [7] $1.4GHz$ Octa core CPUs, i.e., it is powered by the ARM big.LITTLE technology with two sockets and features heterogeneous multi-processing (HMP). Both CPUs feature a snoop control unit (SCU) for memory access. The A15 additionally features an accelerator coherency port (ACP), translation look-aside buffers (TLBs), and AXI coherency extensions (ACE). The GPU of the ODROID-XU4 is a Mali-T628 MP6, which supports OpenGL ES 3.1/2.0/1.1 and OpenCL 1.2 full profile. The SBC has $2GByte$ LPDDR3 RAM PoP stacked and a eMMC5.0 HS400 flash storage socket, two USB 3.0 ports, a USB 2.0 port, a *GBit* ethernet adapter, and an HDMI 1.4a display port. It is powered by 5V/4A input. A sketch of the hardware architecture is shown in Fig. 1a. Figures 1b and 1c show the layout of the Cortex-A7 and Cortex-A15. Hardkernel claims to have created a SBC, which outruns the latest Raspberry Pi 3 model by CPU/RAM performance. Recently, Hardkernel released the stackable cluster solution ODROID-MC1, which consists of four slimmed ODROID-XU4, actively cooled by a single fan. To evaluate the ODROID-MC1 for scientific flow problems, a hybrid MPI/OpenMP simulation framework based on a lattice-Boltzmann method (LBM) is employed. The LBM is natively used for large-scale multi-physics engineering applications on high-performance computer (HPC) systems, e.g., for the simulation of the flow in the human respiratory system [8–15]. It is an explicit method operating on unstructured data which is a representative of a whole class of simulation codes, e.g., for all unstructured (flow) simulation codes that are rather memory-bound than compute-bound. Note that the majority of HPC flow simulation codes have, due to these limitations peak performances, which are in the range of 1-5% of the actual peak performance of a CPU. It is of high interest to understand the capabilities and limitations of SBC systems such as the ODROID-MC1 with respect to hardware and compute performance, scalability, memory limitations, network performance, energy consumption, and price. The present manuscript investigates these aspects and comparatively juxtaposes the results to state-of-the-art HPC systems installed at German HPC centers.
In the following, the numerical methods are presented in Sec. 2. Subsequently, Sec. 3 discusses the employed hardware and software stack, before performance and power consumption results are presented in Sec. 4 Finally, the results are summarized and a conclusion is drawn in Sec. 5, and an outlook is given in Sec. 6.

2. Numerical methods

Since the LBM is employed to evaluate the performance of the ODROID-MC1, a brief introduction into the grid generation and the LBM is given in the following Secs. 2.1 and 2.2.

2.1. Grid generation

Computational meshes are generated by a massively parallel grid generator [16], which is suited for the construction of large-scale hierarchical Cartesian octree-meshes on $\mathcal{O}(10^5)$ computational cores. The mesh generation is subdivided into a serial and parallel stage. In the serial stage, first each participating process reads the geometry from disk and stores the triangles for fast cell/triangle overlap- and inside/outside-detections in an alternating digital tree (ADT) [17]. Subsequently, an initial cube surrounding the geometry is continuously refined. The refinement constitutes an octree, from which cells outside the geometry are removed at every refinement level. The mesh is refined towards an initial base level l_α and levels $l < l_\alpha$ are deleted. A Hilbert curve [18] is placed across the remaining cells and used to decompose the mesh for further parallel refinement. In the parallel stage, each process continues to subdivide the remaining cells it is responsible for towards a coarse uniformly refined computational mesh on level $l_\beta > l_\alpha$. Then, boundary refinement is introduced using cross-process, distance-based, and recursive propagation algorithms. This leads to meshes on level $l_\gamma > l_\beta$ in which cells continuously become finer in the vicinity of walls. The refinement is constrained by a maximum cell-level distance of 1 that is allowed between neighboring cells in the mesh. Boundary-refined meshes enable to highly resolve free and boundary-attached shear layers and hence improve the overall accuracy of simulations featuring high-gradient regions and of wall-shear stress computations. To avoid load-imbalance during meshing a dynamic load-balancing algorithm is capable of efficiently redistributing the work load. In principle, the cells on level l_α are finally employed for the mesh decomposition in the simulation. The mesh is written to disk using the either the parallel methods from HDF5 [19] or Parallel NetCDF [20]. For more details, the interested reader is referred to [16].

2.2. Lattice-Boltzmann method

The simulations employ an LBM, which is part of a larger simulation framework. The LBM has proven to be an efficient method for the computation of low-Mach and low- to moderate-Reynolds number flows in complex geometries [8–15]. Easy second-order accurate boundary condition implementations and its straightforward parallelizability excels this method for the computation of complex flows in intricate geometries on HPC systems. The code is hybrid MPI and OpenMP parallelized, makes use of the same I/O methods as the grid generator, and has been validated in [21, 22].
To solve for fluid flows, the Boltzmann equation with the right-hand side BhatnagarGrossKrook (BGK) collision operator is discretized to yield the lattice-BGK equation

$$f_i\left(\mathbf{x} + \xi_i \Delta t, t + \Delta t\right) \quad = \quad f_i\left(\mathbf{x}, t\right) + \omega \Delta t \cdot \left(f_i^{eq}\left(\mathbf{x}, t\right) - f_i\left(\mathbf{x}, t\right)\right), \qquad i = 0, \ldots, 18 \tag{1}$$

in D3Q19 discretization space [23]. Eq. 1 is solved for the particle probability distribution functions f_i (PPDFs) with $\mathbf{x}$ representing the spatial location, t the time, Δt the time increment, and ξ_i is the discrete particle velocity in direction i. The equation describes the relaxation toward the discrete Maxwell equilibrium distribution function f_i^{eq}. The speed of relaxation is given by

$$\omega \Delta t = \frac{c_s^2 \Delta t}{\nu + \frac{1}{2} c_s^2 \Delta t}, \tag{2}$$

with the speed of sound $c_s = \sqrt{1/3}$ and the kinematic viscosity ν. The collision operator, which is the right hand side of Eq. 1, describes in a statistical sense the collision process of particles in a finite fluid volume, while the left hand side of this equation describes the transport mechanisms from one fluid volume to neighboring fluid volumes. The conservative macroscopic variables are obtained from the moments of the PPDFs. For algorithmic reasons the collision step is separated from the propagation step. Mesh refinement is realized by the method of Dupuis and Chopard [24]. In this method, the transfer of the conservative variables and the PPDFs across different hierarchy levels in the octree are tri-linearly interpolated and converted by an adaption factor that depends on the time steps of the involved mesh levels.
Solid walls employ the interpolated, second-order accurate no-slip boundary condition by Bouzidi et al. [25]. Various von Neumann and Dirichlet boundary conditions are implemented for in- and outflows [13].

	ODROID-MC1	JURECA	JUQUEEN	HAZEL HEN
bandwidth $[GByte/s]$	14.9	68.0	42.6	68.0

Table 1: Memory bandwidth of CPUs of the systems investigated. The values for the ODROID-MC1 are taken from [26]. Note that both JURECA and HAZEL HEN are equipped with the same Intel Xeon E5-2680 v3 Haswell CPUs, hence having the same memory bandwidth.

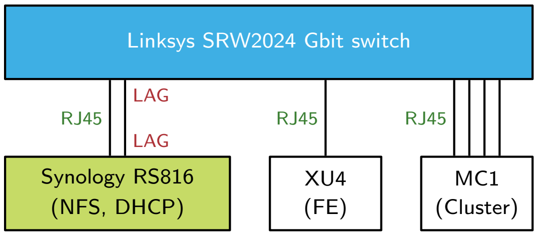


Figure 2: Hardware setup for performance measurements of the ODROID-MC1.

3. Hardware and software stack

In Sec. 4, performance measurements are performed on four systems, i.e., on an ODROID-MC1 cluster and on the JURECA [27] and JUQUEEN [28] supercomputers located at Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich, and the HAZEL HEN system located at High-Performance Computing Center Stuttgart (HLRS). Therefore, the employed hardware and for the ODROID the software stack will be presented in the following. Since memory bandwidth plays a crucial role in the performance, Tab. 1 compares the according values for each system presented below.

3.1. ODROID-MC1

The cluster front end is an ODROID-XU4 equipped with a 16*GByte* eMMC 5.0 module. The ODROID cluster is a single four-node ODROID-MC1 with headless slimmed ODROID-XU4 SBCs. Each node is equipped with a MicroSDHC SDC8*GByte* card from Kingston. The ODROID SBCs are powered by a Meanwell RD-125B power supply. Shared file systems are mounted via autofs/NFS from a Synology RS816 server, which also functions as a DHCP server. All units are interconnected via a Linksys SRW2024 24-port *Gbit* switch. Employed RJ45 cables are at least of CAT.6e type. On the Synology server, the available two ethernet ports are bonded for link aggregation (LAG). On the switch, two ports that connect to the file server are also configured as LAG ports. Fig. 2 gives an overview of the hardware setup.
All ODROID SBCs have the latest Ubuntu release 16.04.4 with kernel 4.9.27-35 armv7l installed. For code compilation the gnu compiler collection 7.2.0 (GCC) is used. Code parallelization employs the mpich-3.2.1 library. Shared memory parallelization makes use of the GCC-shipped OpenMP 4.5 features. The LBM code uses PnetCDF 1.9.0 [20] for parallel I/O and FFTW 3.3.7 [29] for some flow field initialization. To schedule jobs on the ODROID-MC1, slurm-17.11.5 with munge-0.5.13 for authentication and pmix-2.1.0 is employed. For more information on the compilation options employed for the libraries and for the simulation framework, and on the Slurm configuration, the interested reader is referred to Appendix A, Appendix B, and Appendix E.

3.2. JURECA supercomputer

The JURECA supercomputer consists of 1,872 compute nodes, each equipped with a dual-socket system consisting of two Intel Xeon E5-2680 v3 Haswell CPUs. The CPUs are clocked at $2.5GHz$ and have 12 cores each. That is, the whole system consists of 44,928 cores. 1,605 compute nodes are equipped with $128GByte$, 128 with $256GByte$, and 64 with $512GByte$ DDR4 memory clocked at $2,133MHz$. 75 of the compute nodes are furthermore equipped with two NVIDIA K80 GPUs each. Additionally, the JURECA has a booster module with 1,640 compute nodes with one Intel Xeon Phi 7250-F Knights Landing CPUs (KNL) per node. Each KNL has 68 cores clocked at $1.4GHz$ and is equipped with $96GByte$ memory plus $16GByte$ MCDRAM high-bandwidth memory. Altogether, the booster module has 111,520 CPU cores. The overall CPU, GPU, and KNL peak performances of JURECA are 1.8, 0.44, and 5 Petaflop. The JURECA uses also uses Slurm for job scheduling and a Mellanox EDR InfiniBand high-speed network with non-blocking fat tree topology for communication. JURECA is attached to a storage system with a bandwidth of about $150GByte/s$. For code compilation the gnu compiler suite 7.3.0 is used. Further details on the compile options are given in Appendix C.

3.3. JUQUEEN supercomputer

The JUQUEEN is an IBM BlueGene/Q system and consists of 28,672 nodes containing IBM PowerPC A2 CPUs at $1.6GHz$, 16 cores, and $16GByte$ of RAM per node. The overall peak performance is $5.9PFlop/s$. Due to its 4-way SMT hardware threaded floating point units it is capable of running a maximum number of 4 OpenMP threads per core. The JUQUEEN system uses the IBM LoadLeveler as job scheduler and has a 5D Torus network with a bandwidth of $2GByte/s$ per link and direction. On JUQUEEN the Clang compiler 6.0 is used with the options given in Appendix D.

3.4. HAZEL HEN supercomputer

The CRAY HAZEL HEN system consists of 7,712 dual socket nodes containing each two Intel Xeon E5-2680 v3 Haswell CPUs, each with 12 cores clocked at $2.5GHz$. The system has a peak performance of $7.4PFlop/s$ for 185,088 cores. The nodes contain $128GByte$ of RAM. Parallel I/O is implemented via a Lustre File System (LFS), see [30]. Further details on the compile options are given in Appendix C.

4. Results

The performance of the ODROID-MC1 is in the following analyzed from a memory (Sec. 4.1), compute performance (Sec. 4.2), and power consumption (Sec. 4.3) point of view using the grid generator and LBM as introduced in Sec. 2 and the hardware setup outlined in Sec. 3. While for the measurements a canonical simulation case is employed, Sec. 4.4 presents some results for a realistic simulation. The performance findings are juxtaposed to results obtained on the JURECA and HAZEL HEN supercomputers. For the performance analyses, mainly strong scalability [31, 32] results and mega lattice site updates per second $MLUPs$ [33] are considered.

4.1. Memory consumption

First, the memory consumption for the grid generation described in Sec. 2.1 is investigated. Therefore, the massively parallel grid generator is started using $1, \dots, 4$ nodes with a single MPI rank per node. The number of cells is continuously increased until the upper memory limit is reached. Tab. 2 shows the amount of cells that can be generated on the ODROID-MC1. Having $2GByte$ LPDDR3

	1 rank	2 ranks	3 ranks	4 ranks
no. cells (grid) [10^6]	9.5	19.0	28.5	38.0
no. cells (sim) [10^6]	2.3	4.6	6.9	9.2
LPDDR3 [$GByte$]	2.0	4.0	6.0	8.0

Table 2: Memory consumption for grid generation (grid) and LBM simulation (sim). Furthermore, the total amount of available memory using an increasing number of nodes is shown.

RAM available per node, the total amount of cells for a single node is approximately $9.5 \cdot 10^6$, which amounts to roughly $38 \cdot 10^6$ cells using all four nodes of the ODROID-MC1. Considering the memory footprint of a simulation, it is obvious that a simulation run requires way more memory than a grid generation. The present simulation framework under application of the D3Q19 discretization model of the LBM allows to have a maximum number of cells of $2.3 \cdot 10^6$ per node, leading to a total problem size of $9.2 \cdot 10^6$ that can be simulated on the cluster (see Tab. 2). It should be noted that the memory footprint of the LBM is due to the 32-bit nature of the SBC smaller than on 64-bit systems.

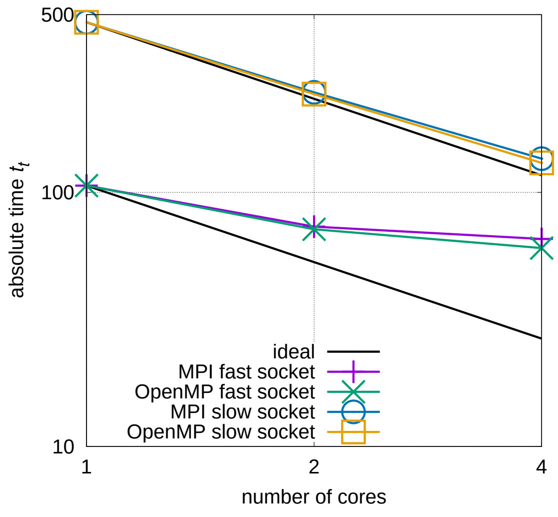
4.2. Compute performance

To evaluate the performance of the ODROID-MC1, different run time measurements are performed with `MPI_Wtime()` functions. First, the single node performance is investigated in Sec. 4.2.1. Subsequently, Sec. 4.2.2 discusses the inter-node performance of the ODROID-MC1, before in Sec. 4.2.3 the performance is analyzed for the complete system and a comparison to the performance on HPC systems is performed in Sec. 4.2.4. For all simulation cases, a cubic domain with periodic boundaries in all Cartesian directions serves as a benchmark case. Three mesh sizes are considered. The first mesh consist of $\mathcal{C}_1 = 2.05 \cdot 10^6$ cells and has levels $l_\alpha = 6$ and $l_\beta = 7$. The second mesh consist of $\mathcal{C}_2 = 8.89 \cdot 10^6$ cells and has levels $l_\alpha = 6$ and $l_\beta = 8$. The third mesh consist of $\mathcal{C}_3 = 1.225 \cdot 10^9$ cells and has levels $l_\alpha = 8$ and $l_\beta = 10$. The total run time t_t, excluding the pre-processing and I/O, is subdivided into the time for the collision step t_c, the time for the propagation step t_p, compiling the communication buffer and distributing incoming data to the cells t_b, and the communication time t_m. Simulations employ the D3Q19 discretization scheme and are run for 100 LBM iterations.

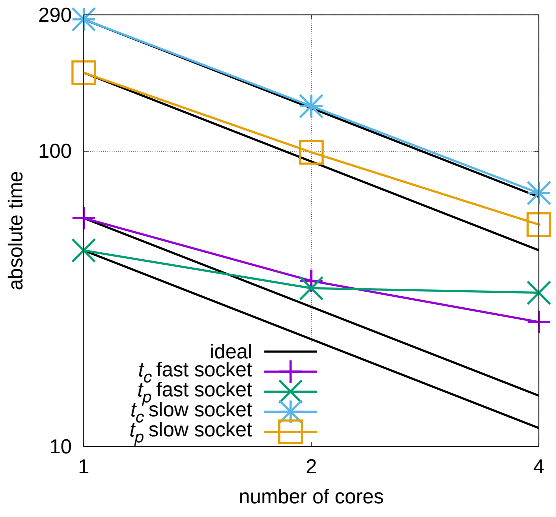
4.2.1. Single node performance

First, the performance on a single node is tested using either the fast or the slow socket of the SBC, i.e., either the big or the LITTLE part of the ARM big.LITTLE technology is used. For each of the sockets, pure MPI (M) and pure OpenMP (O) measurements are performed, i.e., for the fast and slow sockets S^+ and S^- the MPI/OpenMP tuples are $r_M^\pm \in \{(1,1),(2,1),(4,1)\}$. Note that the first entry of these tuples corresponds to the total number of MPI ranks, while the second entry represents the number of OpenMP threads per MPI rank. For the OpenMP measurements it is $r_O^\pm \in \{(1,1),(1,2),(1,4)\}$ and for scheduling[1] `OMP_SCHEDULE=static` is used. Note that tests using `guided` instead of `static` is not faster for these cases. For job submission, slurm is employed and jobs are pinned to $S^\pm$ with the batch command `srun` and the CPU-binding masks Y^+ =`0xf0` and Y^- =`0x0f` (for a sample job script, the interested reader is referred to Appendix E). Table 3

[1]OpenMP Loop Scheduling `https://software.intel.com/en-us/articles/openmp-loop-scheduling`



(a) Strong scalability of the pure MPI and OpenMP runs on each the fast and the slow socket. The number of cores is shown over the absolute complete run time.



(b) Detailed strong scaling plots for the collision (time t_c) and propagation (time t_p) steps of the OpenMP LBM cases in Fig. 3a.

Figure 3: Strong scalability of the LBM using pure MPI and pure OpenMP executions on a single ODROID-MC1 node using either the fast or the slow socket. The LBM is run for 100 iterations.

socket	type	$r^{\pm}_{\{M,O\}}$	t_t $[s]$	t_c $[s]$	t_p $[s]$	t_m $[s]$	t_b $[s]$	speedup	par. eff. [%]
S^+	M	$(1,1)$	106.29	59.39	46.13	0.04	0.73	1.00	100.00
		$(2,1)$	73.32	36.66	35.15	0.35	1.17	1.45	72.48
		$(4,1)$	65.54	27.66	34.90	1.35	1.64	1.62	40.54
	O	$(1,1)$	106.29	59.39	46.13	0.04	0.73	1.00	100.00
		$(1,2)$	71.56	36.39	34.37	0.04	0.75	1.49	74.26
		$(1,4)$	60.36	26.39	33.17	0.04	0.75	1.76	44.03
S^-	M	$(1,1)$	466.80	280.18	184.72	0.23	1.68	1.00	100.00
		$(2,1)$	247.43	143.47	100.89	0.82	2.24	1.89	94.33
		$(4,1)$	135.70	74.71	58.16	0.83	2.00	3.44	86.00
	O	$(1,1)$	466.80	280.18	184.72	0.23	1.68	1.00	100.00
		$(1,2)$	243.46	142.31	99.24	0.22	1.69	1.92	95.87
		$(1,4)$	130.60	72.15	56.47	0.30	1.68	3.57	89.36

Table 3: Absolute run times of 100 iteration of the LBM on a single node on either the fast socket S^+ or the slow socket S^-. The absolute run time t_t is subdivided into the time for the collision step t_c, the time for the propagation step t_p, compiling the communication buffer and distributing incoming data to the cells t_b, and the communication time t_m.

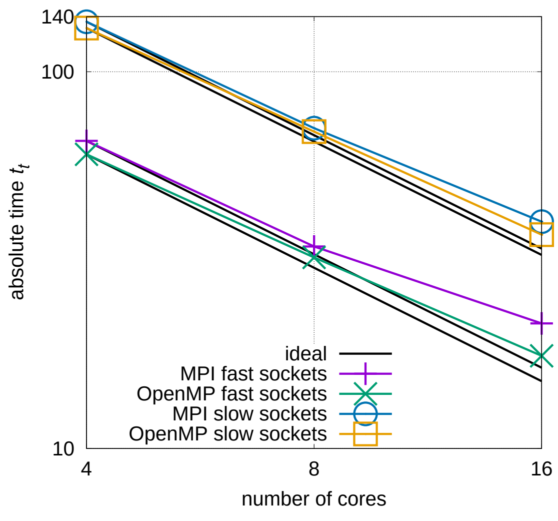
and Fig. 3a show the result of the strong scaling tests. Obviously, both the pure MPI and the pure OpenMP execution scale well across the slow socket, i.e., their parallel efficiency, which is defined by the ratio of the expected scaling value under optimal scaling conditions and the achieved scaling value in percent, is at 86.0% and 89.36% on 4 cores. In contrast, the scalability on the fast socket is not optimal. The OpenMP case scales with 44.03% parallel efficiency on 4 cores slightly better than the MPI case with 40.54%. The timings t_c and t_p of the collision and propagation are responsible for this behavior. Fig. 3b exemplarily shows these timings for the OpenMP runs on both sockets. While the parallel efficiency of the collision computation slightly increases from 2 to 4 cores, the propagation is on 4 cores almost as expensive as on 2 cores. Since the LBM is operating on an unstructured grid, memory access, especially in the propagation step, becomes quasi-random. In more detail, in the propagation step local PPDFs are accessed in a succeeding manner, need, however, to be distributed to quasi-random locations in memory. It is suspected that this leads to page faults, which requires to reload data from memory more frequently. The memory channels, which are at $14.9 GByte/s$ [26], are hence overloaded, rendering the propagation step bandwidth-bound. Note that a detailed study via a roof-line model [34] is not possible due to missing tools on ODROID-MC1. The results, however, underline that the LBM resides on the memory-bound side of the roof-line graph, which is a typical behavior for codes with unstructured memory access patterns. On the slow socket the collision operation scales perfectly. Again the propagation is responsible for the drop of the total parallel efficiency. However, since the slow socket is roughly 4.4 times slower than the fast socket on 1 core and 2.1 times slower on 4 cores, expensive memory operations are hidden behind expensive computational operations. Looking at Tab. 3, the impact of the buffer and communication times t_b and t_m are negligible. For OpenMP runs t_b and t_m stay almost constant for varying numbers of OpenMP threads. Furthermore, the buffer time t_b slightly increases from 1 to 2 ranks and slightly decreases again from 2 to 4 ranks. Note that periodicity is realized via MPI communication and, hence, it is $t_m > 0$ for all tuples $r_{\{M,O\}}$, i.e., even for all single rank cases the buffer is filled and information is exchanged with the same rank via MPI.

4.2.2. Inter-node performance

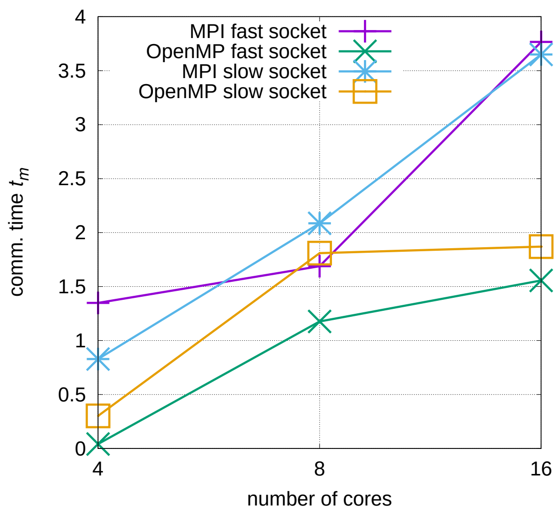
Next, the inter-node performance is investigated using either only S^+ or S^-. Therefore, configuration triplets (nodes / MPI ranks per node / OpenMP threads per MPI rank) $h^{\pm}_{\{M,O\}} \in \{(1,4,1), \ldots, (4,4,1), (1,1,4), \ldots, (4,1,4)\}$ are tested. Figure 4 and Tab. 4 show the results of the experiments. From Fig. 4a it is obvious that the OpenMP scalability across the whole ODROID-MC1 using S^+ and S^- is with parallel efficiencies of 85.63% and 88.36% good. Also the pure MPI run on S^- scales well across the system. Looking, however, at Fig. 4b, a linear increase in communication time is visible for the MPI version on S^-, which is only hidden by the slow computation and compensated by the almost perfect bisection of the times t_c and t_p under increasing node numbers. That is, it is expected that for larger node numbers the scaling becomes worse. Among all scaling plots of Fig. 4a, the MPI runs on the S^+ scale worst. This can be explained by the strong increase of communication time from 8 to 16 cores shown in Fig. 4b. Unlike on S^-, the fast computation, which also shows an almost perfect bisection of t_c and t_p, cannot compensate this effect in the complete scaling graph. Considering the OpenMP runs the communication times experience a jump from 1 node to 2 nodes, which is due to the additional inter-node communication overhead and the already small initial communication times on a single node (also compare Tab. 3). Interesting is the change in t_m from 2 to 4 nodes. While t_m on S^- stays almost constant, t_m on S^+ slightly increases.

4.2.3. Performance of the complete cluster

To evaluate the performance of the whole cluster, scaling tests are performed on all four nodes of the ODROID-MC1 using all cores of the nodes. Therefore, different parallelization strategies are employed. First, the configuration triplets $\hbar^{\pm}_{\{M,O\}} \in \{(1,2,4), \ldots, (4,2,4), (1,1,8), \ldots, (4,1,8)\}$ are used. For the



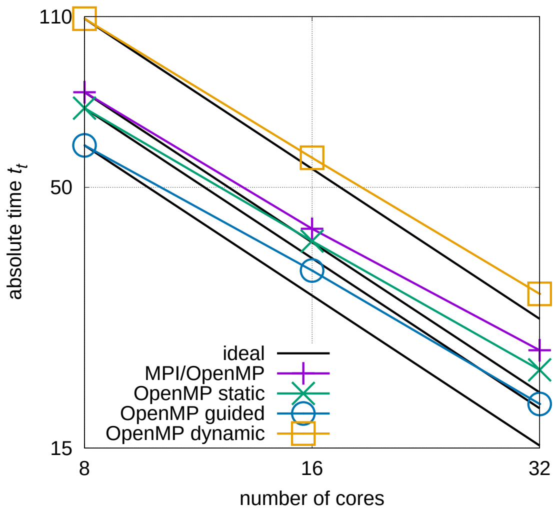
(a) Strong scalability of the pure MPI and OpenMP runs across the whole cluster using either the fast or the slow socket. The number of cores is shown over the absolute complete run time.



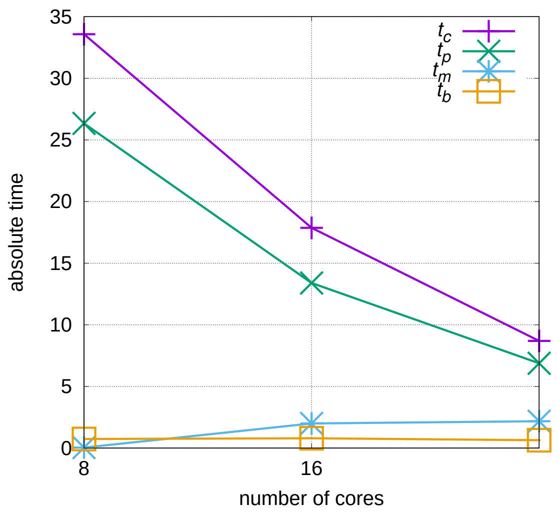
(b) Communication time t_m of the runs shown in Fig. 4a.

Figure 4: Strong scalability of the LBM using pure MPI and pure OpenMP executions across the ODROID-MC1 using either the fast or the slow socket. The LBM is run for 100 iterations.

triplets with 8 OpenMP threads per node the time difference between the different OpenMP scheduling options `OMP_SCHEDULE`={`static, guided, dynamic`} is measured (types $M/O^{st,gu,dy}$ in Tab. 5). Figure 5a and Tab. 5 show the results of the measurements. Configuration $\hbar^{\pm}_{\{M,O\}} = (x, 2, 4), x \in \{1, 2, 4\}$ suffers from the distribution of equally sized chunks of cells on S^+ and S^-, i.e, although non-blocking communication is used, sockets S^+ need to wait for sockets S^- to finish their work. Considering the absolute total run times this case is in its execution even slower than the cases $h^{\pm}_{\{M,O\}} = (x, 1, 4)$. Instead of using 2 MPI ranks with 4 OpenMP threads on each node, starting 8 OpenMP threads per node slightly enhances the performance. Among the parallelization types $M/O^{st,gu,dy}$, the `guided` scheduling outperforms the `static` and `dynamic` scheduling, reaches, however, not the performance of the so far best computing configuration $h^{\pm}_{\{M,O\}} = (x, 1, 4)$. The `static` case distributes the loop iterations of the collision and propagation equally on the available cores and hence their run times are dictated by S^-. In contrast, `dynamic` scheduling allows to use the internal work queue to give a chunk-sized block of loop iterations to each thread, the corresponding costly overhead, however, renders this method the most expensive. Using `guided` the chunk-size per thread continuously decreases and allows for better load-balancing. In Fig. 5b, the run times of the individual parts of the LBM are shown for case $h^{\pm}_{M/O^{gu}} = (x, 1, 8)$. While the communication time t_m increases slightly and the time for setting up the buffer t_b stays almost constant, the collision and propagation times t_c and t_p are almost bisected for each doubling of the number of cores (see also Tab. 5).
In addition to using a static decomposition of the computational domain on the fast and slow cores, simulations are run with performance-weighted distributions of the number of cells on the fast and slow cores with configuration $\hbar^{\pm}_{\{M,O\}} = (4, 2, 4)$ with `OMP_SCHEDULE=guided`, i.e., a distribution $\mathcal{D}$ of



(a) Strong scalability of mixed MPI/OpenMP runs with different OpenMP scheduling across the whole cluster using both the fast and the slow sockets. The number of cores is shown over the absolute complete run time.



(b) Run times of the individual parts of the LBM for the OpenMP guided case shown in Fig. 5a.

Figure 5: Strong scalability of the LBM using mixed MPI/OpenMP executions across the whole ODROID-MC1. The LBM is run for 100 iterations.

the number of total cells $\mathcal{C} = \mathcal{C}_1$ on Θ compute nodes of

$$\mathcal{D} = \begin{cases} \left\lceil \frac{\mathcal{C}}{\Theta} \cdot \frac{1}{1+\kappa} \right\rceil, & \text{rank } \%2 = 0 \ (S^-) \\ \left\lceil \frac{\mathcal{C}}{\Theta} \cdot \left(1 - \frac{1}{1+\kappa}\right) \right\rceil, & \text{rank } \%2 = 1 \ (S^+) \end{cases} \tag{3}$$

is used. The performance factor $\kappa_{c,1}$ between the fast and slow cores is varied as $\kappa_{c,1} \in [1.0, 13.0]$ with a coarse $\delta\kappa_{c,1} = 0.1$ across the whole interval and a fine $\delta\kappa_{f,1} = 0.01$ in the interval $[2.0, 3.0]$. The corresponding results are shown in Fig. 6a. The optimum is reached at $\kappa_{min,1} = 2.46$, requiring only $t_{t,1} = 19.44s$ for the complete execution. As the times for the collision and propagation computation $t_{c,1}$ and $t_{p,1}$ continuously decrease, the communication time continuously increases after a slight drop for small $\kappa_{c,1} < 1.4$. The strong increase of $t_{m,1}$ is also the reason for the increase of $t_{t,1}$ for $\kappa_{c,1} > \kappa_{min,1}$. Although the computation is faster than $\hbar^{\pm}_{\{M,O\}} = (4, 2, 4)$ using `static` scheduling, it cannot compete with $h^+_O = (4, 1, 4)$ ($t_{min,1} = 17.62s$). That is, even a perfect performance-weighted distribution of the computational work does not allow to increase the speed of the computation in this case. It has to be noted, however, that execution times may vary. This is, e.g., visible when comparing the inset of Fig. 6a with $\delta\kappa_{f,1}$ to the results of $\delta\kappa_{c,1}$. That is, the same execution times are not exactly matched for the same values of κ.

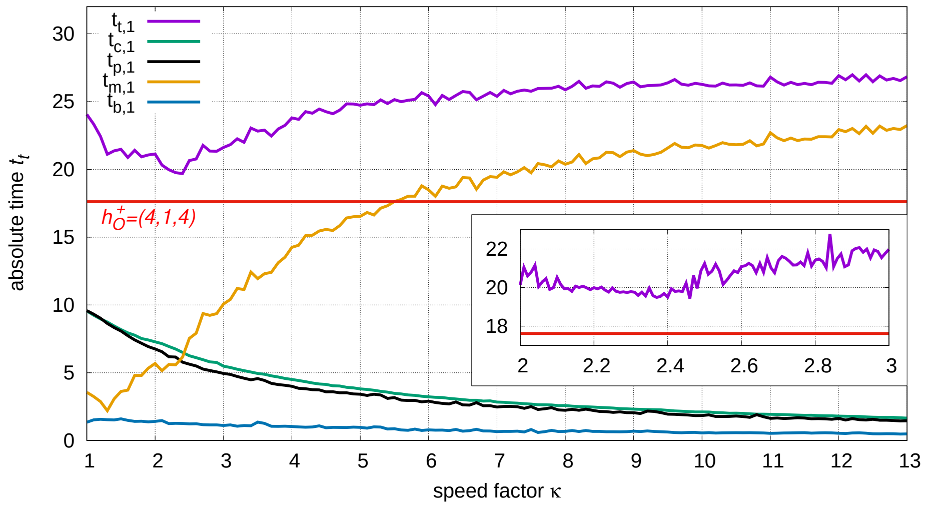
To furthermore check if the execution for larger cases using the weighted approach is also slower than the standard approach, the bigger mesh with $\mathcal{C}_2$ cells is employed and $\kappa_{c,2}$ and $\kappa_{f,2}$ are varied again in the intervals $[1.0, 13.0]$ and $[2.0, 3.0]$. The results for the measurements are shown in Fig. 6b and show a similar behavior as for $\mathcal{C}_1$. That is, while the times $t_{p,2}$ and $t_{c,2}$ continuously decrease, the

socket	type	$\hbar^{\pm}_{\{M,O\}}$	t_t $[s]$	t_c $[s]$	t_p $[s]$	t_m $[s]$	t_b $[s]$	speedup	par. eff. [%]
S^+	M	$(1,4,1)$	65.54	27.66	34.90	1.35	1.64	1.0	100.00
		$(2,4,1)$	34.37	13.79	17.85	1.69	1.04	1.91	95.35
		$(4,4,1)$	21.48	7.57	9.40	3.77	0.74	3.05	76.27
	O	$(1,1,4)$	60.36	26.39	33.17	0.04	0.75	1.0	100.0
		$(2,1,4)$	32.14	13.38	16.79	1.18	0.80	1.88	93.89
		$(4,1,4)$	17.62	6.84	8.58	1.56	0.64	3.43	85.63
S^-	M	$(1,4,1)$	135.70	74.71	58.16	0.83	2.00	1.0	100.00
		$(2,4,1)$	70.82	38.02	29.36	2.08	1.36	1.92	95.80
		$(4,4,1)$	40.00	19.97	15.36	3.65	1.01	3.39	84.83
	O	$(1,1,4)$	130.60	72.15	56.47	0.30	1.68	1.0	100.00
		$(2,1,4)$	69.35	36.82	28.55	1.81	2.17	1.88	94.16
		$(4,1,4)$	36.95	18.72	14.55	1.87	1.81	3.53	88.36

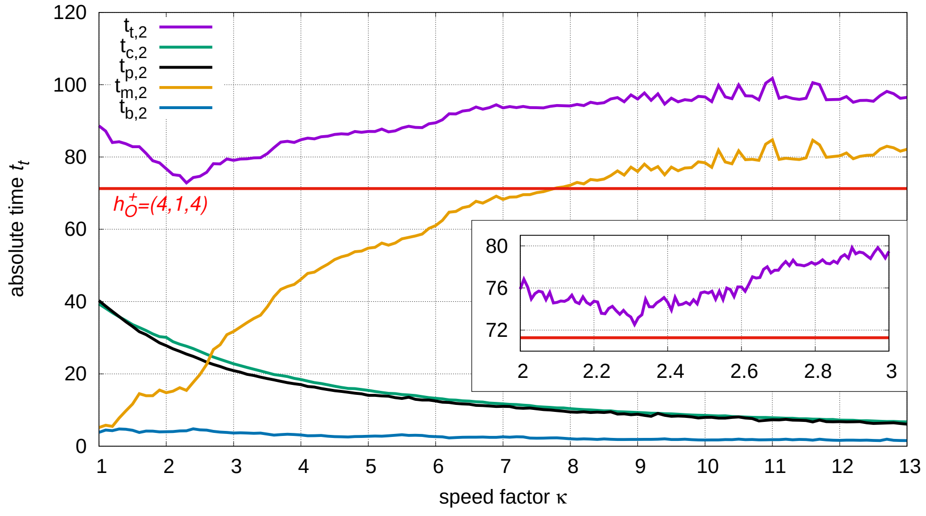
Table 4: Absolute run times of 100 iteration of the LBM on the full ODROID-MC1 using either the fast socket S^+ or the slow socket S^-. The absolute run time t_t is subdivided into the time for the collision step t_c, the time for the propagation step t_p, compiling the communication buffer and distributing incoming data to the cells t_b, and the communication time t_m.

socket	type	$\hbar^{\pm}_{\{M,O\}}$	t_t $[s]$	t_c $[s]$	t_p $[s]$	t_m $[s]$	t_b $[s]$	speedup	par. eff. [%]
$S^{\pm}$	M/O^{st}	$(1,2,4)$	77.58	37.17	37.47	0.53	2.41	1.0	100.00
		$(2,2,4)$	41.30	18.86	18.87	1.57	2.00	1.88	93.93
		$(4,2,4)$	23.57	10.09	9.74	2.35	1.38	3.29	82.29
	M/O^{st}	$(1,1,8)$	72.09	38.27	32.76	0.09	0.97	1.0	100.0
		$(2,1,8)$	39.02	19.45	16.33	1.64	1.59	1.85	92.39
		$(4,1,8)$	21.52	9.98	8.44	1.93	1.16	3.35	83.73
	M/O^{gu}	$(1,1,8)$	60.70	33.57	26.34	0.04	0.74	1.0	100.0
		$(2,1,8)$	34.06	17.87	13.39	2.00	0.80	1.78	89.11
		$(4,1,8)$	18.39	8.69	6.88	2.18	0.64	3.30	82.52
	M/O^{dy}	$(1,1,8)$	108.97	49.78	58.42	0.04	0.74	1.0	100.0
		$(2,1,8)$	57.32	25.16	29.87	1.34	0.95	1.90	95.05
		$(4,1,8)$	30.58	13.20	15.05	1.68	0.64	3.56	89.09

Table 5: Absolute run times of 100 iteration of the LBM on the full ODROID-MC1 using both the fast and slow sockets $S^{\pm}$. The absolute run time t_t is subdivided into the time for the collision step t_c, the time for the propagation step t_p, compiling the communication buffer and distributing incoming data to the cells t_b, and the communication time t_m.



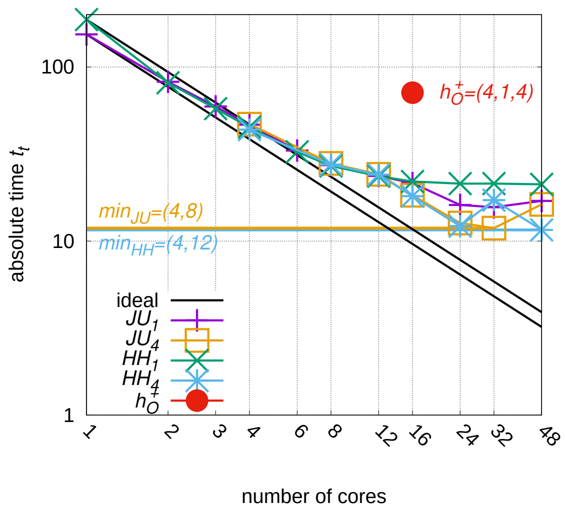
(a) Time measurements for a mesh size of $\mathcal{C}_1 = 2.05 \cdot 10^6$.



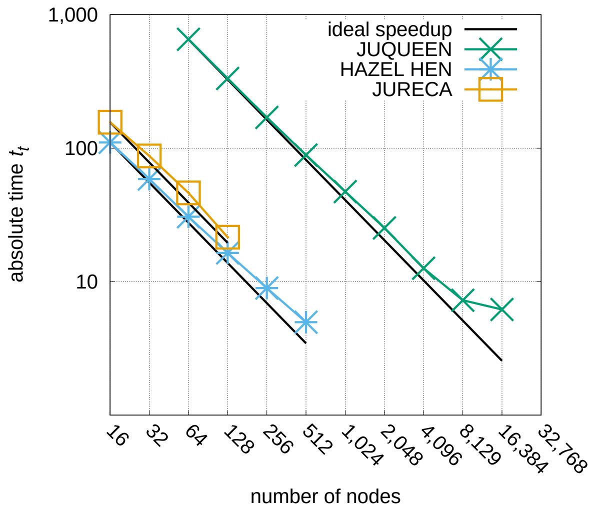
(b) Time measurements for a mesh size of $\mathcal{C}_2 = 8.89 \cdot 10^6$.

Figure 6: Change of the total t_t and fractional times $t_{c,\{1,2\}}$, $t_{p,\{1,2\}}$, $t_{m,\{1,2\}}$, and $t_{b,,\{1,2\}}$ using an increasing speed factor $\kappa_{c,\{1,2\}}$ between $S^{\pm}$ with $\delta\kappa_{c,\{1,2\}} = 0.1$ on the complete ODROID-MC1 for 100 LBM iterations of the D3Q19 algorithm. The red lines shows the minimum execution time obtained for $h_O^+ = (4, 1, 4)$ and the insets show results of a second execution in the interval $\kappa_{c,\{1,2\}} \in [2.0, 3.0]$ with $\delta\kappa_{f,\{1,2\}} = 0.01$.

time for $t_{m,2}$ continuously increases, rendering again the runs with $\hbar^{\pm}_{\{M,O\}} = (4, 2, 4)$ a slightly more efficient approach. However, in contrast to $\mathcal{C}_1$, the complete time $t_{min,2} = 71.27s$ of $\hbar^{\pm}_{\{M,O\}} = (4, 2, 4)$ is almost matched at $\kappa_{min,2} = 2.31$ with $t_{t,2} = 72.53s$. Furthermore, it should be noted that the ratio $\mathcal{C}_2/\mathcal{C}_1 = 4.34$ is smaller than the ratio $t_{min,2}/t_{min,1} = 4.05$ proving a good weak scaling of the problem.



(a) Strong scaling of 100 iteration of the LBM on a single JURECA (JU) and HAZEL HEN (HH) node using the mesh with $\mathcal{C}_2 = 8.89 \cdot 10^6$ cells. Furthermore, the single result at $h_O^+ = (4, 1, 4)$ for the ODROID-MC1 is displayed. For the HPC systems the run times are shown for a single MPI rank and for four MPI ranks (indices 1 and 4).



(b) Strong scalability of the LBM on the JURECA, HAZEL HEN, and JUQUEEN systems using a large production run mesh consisting of $\mathcal{C}_3 = 1.225 \cdot 10^9$ cells.

Figure 7: Performance comparison of the ODROID-MC1 and the JURECA and HAZEL HEN supercomputers. Results for a strong scalability analysis are shown for a large simulation case.

4.2.4. Comparison to the performance on an HPC system

The configuration $h_O^+ = (4, 1, 4)$ shows the best scaling as well the best run time behavior on the ODROID-MC1. Therefore, the corresponding result is comparatively juxtaposed to runs on the JURECA and HAZEL HEN supercomputers for mesh $\mathcal{C}_2$. For the computations on the HPC systems a single node is employed and strong scalability and run times are analyzed. Figure. 7a and Tab. 6 show the corresponding results in absolute run times. Note that due to the memory limitations of the ODROID-MC1, the minimum number of nodes that can be employed for this mesh is four, which is why only a single data point is shown for $h_O^+ = (4, 1, 4)$ in Fig. 7a. Furthermore, Tab. 7 shows the parallel performance given in $MLUPs$ for selected configurations. On JURECA and HAZEL HEN two scaling experiments are run, i.e., each with a single MPI rank per node and increasing numbers of OpenMP threads and with four MPI ranks per node and increasing number of OpenMP threads. Latter runs contain the best performing runs with the minimal run times. JU and HH denote the runs on JURECA and HAZEL HEN, respectively, the indices 1 and 4 the number of MPI ranks. From the results it is obvious that using a single MPI rank on both HPC systems brings not the best time to solution. Case HH_1 shows a superlinear scaling behavior for a small number of OpenMP threads and a good scaling is obtained up to 8 cores. For a larger number of OpenMP threads the scalability becomes worse and the run times for higher core counts stay almost constant. The single core performance for HH_1 is with a difference of 33.23 s to JU_1 worse than on JU_1, i.e., JURECA is 1.22 times faster. Similar to HH_1, a good scaling behavior is visible for JU_1 up to 8 cores, crossing the NUMA domain from 12 to 16 cores brings, however, a strong drop in performance. The fast runs JU_4 and HH_4 show a similar performance

MC-1 h_O^+	t_t $[s]$	JU_1	t_t $[s]$	JU_4	t_t $[s]$	HH_1	t_t $[s]$	HH_4	t_t $[s]$
$(4,1,4)$	71.27	$(1,1)$	154.09	$(4,1)$	46.98	$(1,1)$	187.32	$(4,1)$	43.70
		$(1,2)$	82.31	$(4,2)$	27.94	$(1,2)$	81.20	$(4,2)$	27.51
		$(1,3)$	59.35	$(4,3)$	24.19	$(1,3)$	57.68	$(4,3)$	24.07
		$(1,4)$	46.67	$(4,4)$	18.34	$(1,4)$	45.13	$(4,4)$	18.12
		$(1,6)$	33.12	$(4,6)$	12.70	$(1,6)$	32.57	$(4,6)$	12.19
		$(1,8)$	27.32	$(4,8)$	11.88	$(1,8)$	27.02	$(4,8)$	17.20
		$(1,12)$	23.70	$(4,12)$	16.24	$(1,12)$	23.61	$(4,12)$	11.59
		$(1,16)$	21.52			$(1,16)$	22.01		
		$(1,24)$	16.12			$(1,24)$	21.39		
		$(1,32)$	15.66			$(1,32)$	21.42		
		$(1,48)$	17.02			$(1,48)$	21.18		

Table 6: Absolute run times of 100 iteration of the LBM on the ODROID-MC1 using only the fast cores and a single JURECA (JU) and HAZEL HEN (HH) node. For the HPC systems the run times are shown for a single MPI rank $(1, x)$ and for four MPI ranks $(4, x)$ (indices 1 and 4). Run times in red indicate the fastest computations on the individual systems.

MC-1 h_O^+	$MLUPs$	JU_4	$MLUPs$	HH_4	$MLUPs$
$(4,1,4)$	12.47	$(4,4)$	48.47	$(4,4)$	49.06
		$(4,8)$	74.83	$(4,12)$	76.70

Table 7: Mega lattice updates per second $MLUPs$ of 100 iteration of the LBM on the ODROID-MC1 using only the fast cores and a single JURECA (JU) and HAZEL HEN (HH) node for selected configurations.

behavior from $(4,1)$ to $(4,6)$ with the HAZEL HEN being slightly faster. While for HH_4 a massive increase of the run time is visible for 32 cores, i.e., for combination $(4,8)$, case JU_4 continuous to scale resulting in the lowest run time at $min_{JU} = (4,8)$ with $74.83 MLUPs$. For case HH_4 the overall best performance among all run times is achieved at $min_{HH} = (4,12)$ with $76.70 MLUPs$. Comparing now the performance of the ODROID-MC1 to the results on the HPC systems, it is obvious that despite only $Gbit$ ethernet is used between the ODROID-MC1 nodes, the performance of the system is on the same order as on the HPC systems. Considering furthermore the fastest run at $h_O^+ = (4,1,4)$ with $12.47 MLUPs$, the runs at $min_{HH} = (4,8)$ and $min_{HH} = (4,12)$ are by a factor of 6.04 and 6.15 faster. This is, however, only true by considering the fastest computations. A comparison of the run times using altogether 16 cores, i.e., $h_O^+ = (4,1,4)$, $JU_4 = (4,4)$, and $HH_4 = (4,4)$ reveals that the code is on the ODROID-MC1 only of 3.89 times (JURECA) and 3.93 times (HAZEL HEN) slower. The ODROID-XU4 has a dual channel memory bandwidth of $14.9 GByte/s$ [26], i.e., the memory bandwidth for the ODROID-MC1 sums up to $56 GByte/s$. In contrast, the memory bandwidth on the Intel Xeon E5-2680 v3 Haswell CPUs is at $68 GByte/s$. For the dual socket system this adds up to $136 GByte/s$, which is a factor of 2.43 faster as the on the ODROID-MC1.
It should be noted that the cases considered here are relatively small compared to real production runs on HPC systems, which leads to the previously discussed scalability limits of the LBM, even on

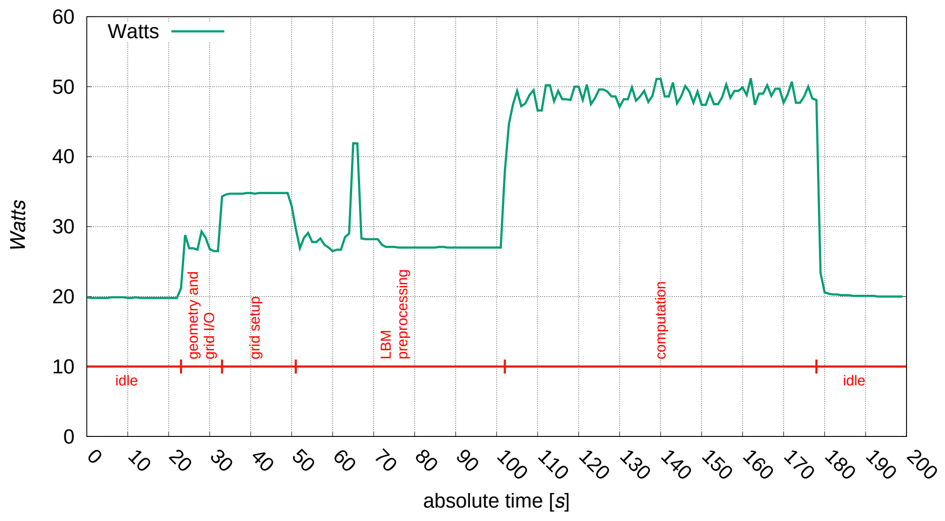


Figure 8: Power consumption of a single computation of $h_O^+(4,1,4)$ on mesh $\mathcal{C}_2$ over time.

HPC systems. To show, however, that the LBM indeed scales across a large number of core counts, Fig. 7b presents strong scaling results for mesh $\mathcal{C}_3$ with $1.1225 \cdot 10^9$ cells, which corresponds to an average production run simulation. The experiments are run on the systems JURECA, HAZEL HEN, and additionally on JUQUEEN. Unlike the previous scaling graphs, Fig. 7b shows the run times over the node counts. Obviously, the LBM shows a very good strong scaling behavior on all three systems. That is, on JURECA an almost linear behavior is visible up to 128 nodes. On HAZEL HEN the code scales well up to 512 nodes and on the massively parallel system JUQUEEN a good scalability up to $8,192$ nodes with a slight decrease in parallel efficiency up to $16,384$ nodes is visible. The rather high absolute run times on JUQUEEN compared to the Intel-based systems are probably due to serial memory accesses and larger cache line sizes on IBM BlueGene/Q. Furthermore, non-optimal compilation could be a cause for the performance loss. That is, on JUQUEEN the IBM XL compiler suite, which would produce highly-optimized machine specific code, cannot be used due to non-existing `C++11` features of the compilers. They are, however, required by the simulation code. Instead, the Clang 6.0 compiler, which supports `C++11`, see Appendix D, is used.

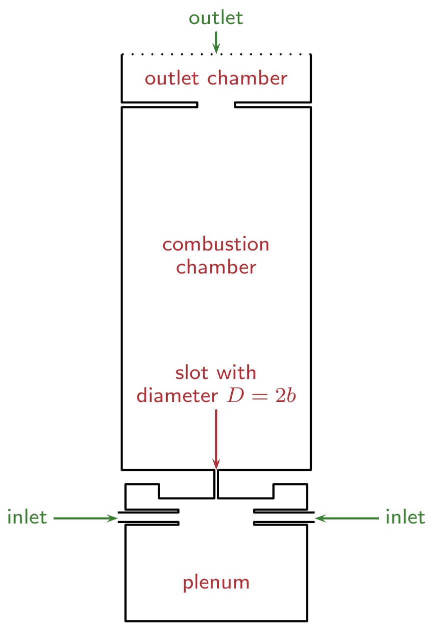
4.3. Power consumption

The power consumption of the ODROID-MC1 is measured by running simulation $h_O^+ = (4,1,4)$ on mesh $\mathcal{C}_2$, cf. Sec. 4.2.4, and by taking the $Watts$ with a Conrad Electronic Voltcraft Energy-Logger 4000, which is installed between the power outlet and the Meanwell RD-125B power supply, powering the whole ODROID-MC1. This logger allows to log power consumption in the sub-$Watt$ range over time with an accuracy of $0.2W$. It stores the corresponding data on SD-card with an interval of 1 minute. Since this accuracy is not sufficient, the data is read from the display of the Energy Logger 4000, which is updated each second. The power consumption of the idling ODROID cluster is measured at $e_{min} = 23.1W$. Fig. 8 shows the power consumption over time, i.e., starting from e_{min} and having an increased consumption over the course of the computation. For better understanding, the time line includes a description of the different periods of the computation. It should be noted that changing

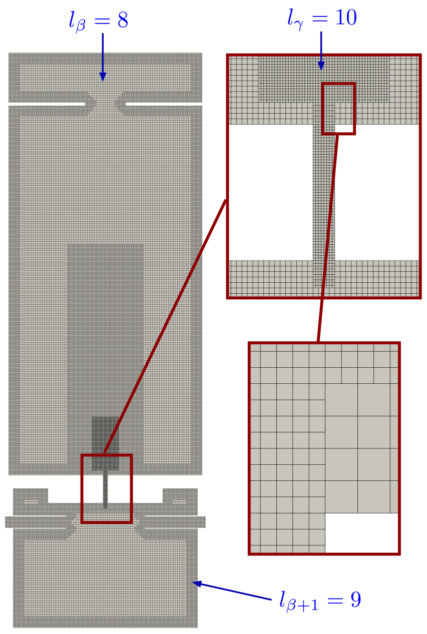
system	power [W]	energy/sim. [Wh]	Wh/MLU [Wh]	W factor	energy factor
ODROID-MC1	54.5	1.079	$1.214 \cdot 10^{-3}$	1.00	1.00
JURECA	160	0.815	$0.917 \cdot 10^{-3}$	0.34	0.76
HAZEL HEN	160	0.805	$0.906 \cdot 10^{-3}$	0.34	0.75
JURECA	300	1.528	$1.719 \cdot 10^{-3}$	0.18	1.42
HAZEL HEN	300	1.510	$1.700 \cdot 10^{-3}$	0.18	1.40

Table 8: Comparison of the power consumptions between the different systems. For the ODROID-MC1 the power consumption is measured. Two estimates are given for the power consumption of the Intel-based systems ($160W$ and $300W$). For each, the energy for the whole simulation, the energy requirement per MLU, and $Watt$ factors as well as energy factors are given. The $Watt$ factors are given by the ratio of the $Watts$ of the ODROID-MC1 simulation and the HPC system simulation. The energy factors are obtained by the ratio of the energy results of the ODROID-MC1 and the compared HPC system.

the number of iterations does not change the initial steps of the simulations (geometry and grid I/O, mesh setup, and LBM preprocessing) but only the computational part. The maximum consumption $e_{max} = 54.5W$ is reached in the computation section, which is fully OpenMP parallelized and employs all four cores. Using these values and considering the preprocessing time of a simulation small compared to the computational part, the energy result is roughly at $1.079Wh$ for this simulation. Considering furthermore the $MLUPs$ from Tab. 7 the energy per mega lattice update MLU is at $1.214 \cdot 10^{-3}Wh$. Measuring the power consumption of HPC systems is complicated. HPC centers usually have no hardware installed to detect the power consumption of single jobs, which is why the following analysis is based on estimated values. The thermal design power (TDP) of the Intel Xeon E5-2680 v3 Haswell CPU installed in JURECA and HAZEL HEN is at $120W$ TDP (Intel Specs). That is, by looking solely at the TDP of two CPUs using only 8 cores, a rough estimate of 160 W and a total energy result of $0.815Wh$ on JURECA and $0.805Wh$ on HAZEL HEN is expected for the computation. Using these estimates the $Watts$ of the Intel CPUs are a factor of 2.94 more than on the ODROID-MC1, the consumed energy is, however, by factors of 0.76 and 0.75 smaller on the JURECA and HAZEL HEN. To be more precise, the energy results per MLU are at $0.917 \cdot 10^{-3}Wh$ and $0.906 \cdot 10^{-3}Wh$ on the HPC systems. In contrast, considering the power consumption of HAZEL HEN, which is at 3.2 MW^2, and a node count of $7,712$, the power consumption per node equates to $414.94W$. Using again only 16 of the 24 cores approximately results in $276.63W$, which is based on the assumption that a core requires $\approx 17.29W$. This, however, distributes the remaining power consumption of the node over the cores, i.e., it is necessary to add missing $Watts$ for the mainboard and the peripherals. Taking additionally the 8 idling CPUs into account, the power consumption can be estimated at $\approx 300W$. This estimate delivers energy results of $1.528Wh$ on JURECA and $1.51Wh$ on HAZEL HEN. The according energy results per MLU equate to $1.719 \cdot 10^{-3}Wh$ and $1.7 \cdot 10^{-3}Wh$. Based on these estimates, it is obvious that the power consumption of the Intel-based systems are with a factor of 5.5 higher than on the ODROID-MC1. Also the consumed energy is with factors 1.42 and 1.4 for JURECA and HAZEL HEN higher than on the ODROID-MC1. These results are summarized in Tab. 8.



(a) Slot burner configuration, which consists of two inlets leading into the plenum and a slot leading into the main combustion chamber. The chamber is connected to the outlet chamber. The REYNOLDS number Re is based on the shorter diameter length of the slot $D = 2b$.



(b) Computational mesh of the slot burner configuration consisting of $\mathcal{C}_3 = 7.9 \cdot 10^6$ cells and levels $l_\alpha = 7$, $l_\beta = 8$, and $l_\gamma = 10$. Visible are the levels $l = 8, \ldots, 10$.

Figure 9: Setup and computational mesh for the simulation of the flow in a three-dimensional slot burner configuration.

4.4. Simulation of the flow in a slot burner

To show that the ODROID-MC1 can indeed be used for scientific applications, a three-dimensional simulation is run on the cluster using configuration $h_O^+ = (4, 1, 4)$ for a slot burner case without combustion. Figure 9a shows the schematic setup of the simulation. A mass flux is prescribed at the two inlets leading into the plenum using a Dirichlet boundary condition for the velocity. The density is extrapolated with a von Neumann condition at the inlets. A second-order accurate interpolated bounce-back no-slip condition is employed at the wall [25]. The slot connects the plenum and the combustion chamber, which merges into the outlet chamber. At the outlet, a Dirichlet condition for the density and a von Neumann condition for the velocity is employed. The slot has a width of D and a length of $7.4D$. The slot half-width is $b = D/2$. The REYNOLDS number $Re = v_b \cdot D/\nu$ is based on the bulk velocity in the slot v_b, the slot width D, and the viscosity of air ν, and is set to $Re = 1,750$. The computational mesh is shown in Fig. 9b and consists of $\mathcal{C}_3 = 7.9 \cdot 10^6$ cells. The base level is given by $l_\alpha = 7$. The mesh is uniformly refined to $l_\beta = 8$. The wall is refined up to level $l_{\beta+1} = 9$ using the boundary refinement method described in Sec. 2.1. Additionally, the slot and the slot outlet region is refined to $l_\gamma = 10$. The level increase is visible in the magnification insets of Fig. 9b. To reach

[2]HAZEL HEN Specs `https://www.hlrs.de/systems/cray-xc40-hazel-hen/`

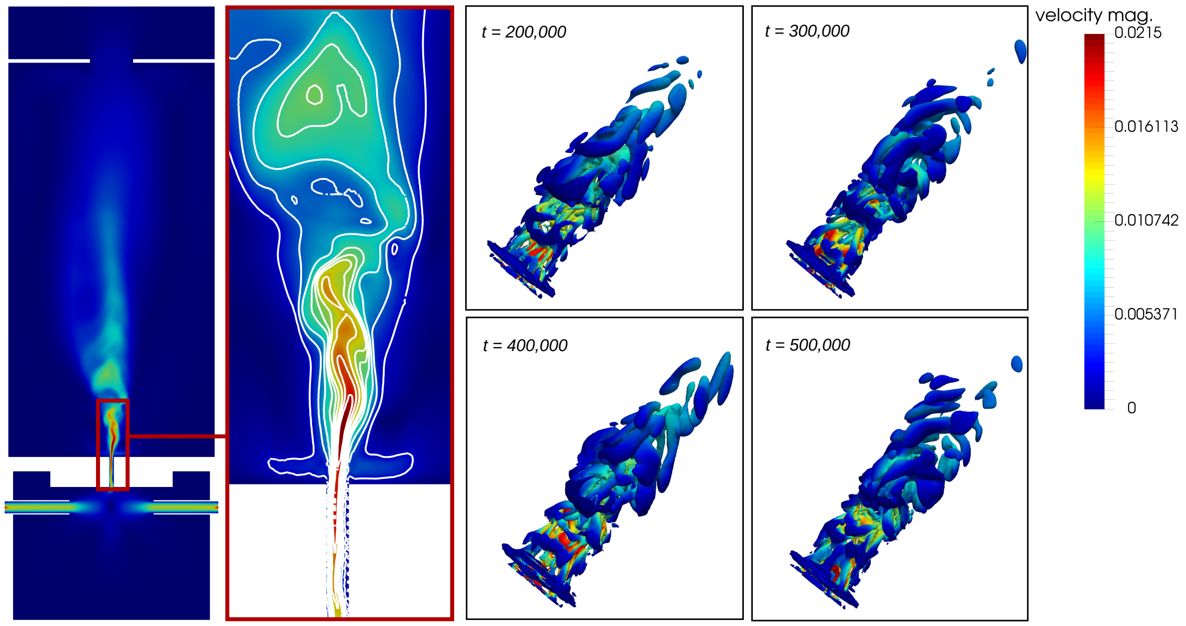
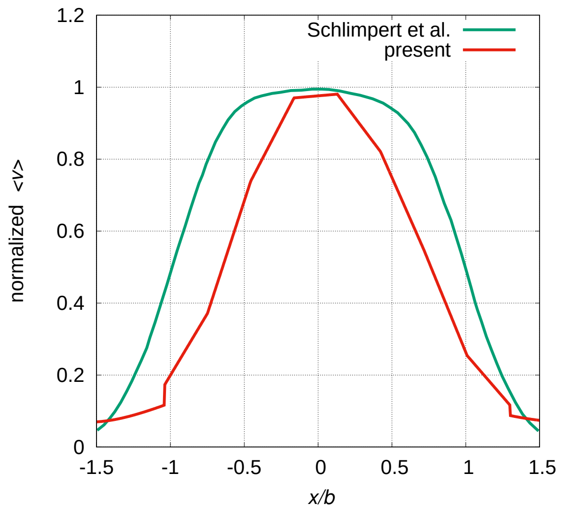
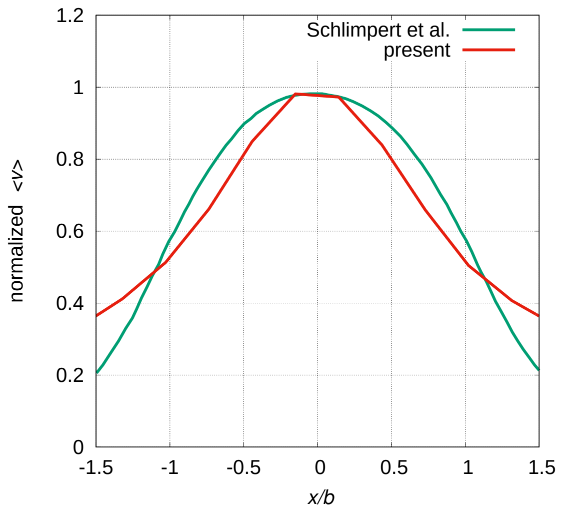


Figure 10: Simulation results of a slot burner configuration. The cross-sections and the vortical structures, which are visualized by the Δ-criterion, are colored by the velocity magnitude. The cross-sections on the left are snapshots at $t = 500,000$.



(a) Profile at $z = D = 2b$.



(b) Profile at $z = 2D = 4b$.

Figure 11: Comparison of the profiles of the temporally averaged velocity magnitude $< |v| >$ to the results from [35] ($Re_S = 7,000$) at the two positions $z = D$ and $z = 2D$ downstream of the slot. The velocity is normalized by the temporally averaged maximum slot velocity $< v_{max} >$.

a quasi-steady state the simulation is advanced for $t = 500,000$ LBM iterations. The corresponding residuals of the density and velocity components are monitored to guarantee an asymptotic behavior. Figure 10 shows the corresponding results of the simulation. On the left side a cross-section though center of the geometry at time step $t = 500,000$ is shown. The cross-section is colored by the velocity magnitude in LBM units. The inset shows the region where the jet from the slot enters the combustion chamber. The contours correspond to intervals of the velocity magnitude. The images on the right show the change of the vortical structures in the jet over LBM iterations $t \in \{200,000, \ldots, 500,000\}$. It is obvious that due to the strong shear layer between fluid at rest and the jet fluid the flow features unsteady fluctuations. The vortical structures are visualized by the Δ-criterion, which is determined by

$$\Delta = \left(\frac{Q}{3}\right)^3 + \left[\frac{det\left(\nabla \otimes \mathbf{v}\right)}{2}\right]^2 > 0, \tag{4}$$

with the velocity vector $\mathbf{v}$ and the Q-criterion

$$Q = \frac{1}{2}\left(|\Omega|^2 - |S|^2\right) > 0 \tag{5}$$

and the vorticity tensor Ω and the strain tensor S.
It should be noted that from the Reynolds number the viscosity and hence the relaxation factor is calculated by Eq. 2, i.e., by

$$\nu = \Delta t c_s^2 \left(\frac{1}{\omega \Delta t} - \frac{1}{2}\right) = \frac{v_b \cdot D}{Re}. \tag{6}$$

That is, $0 < \omega\Delta t < 2$ must be ensured to keep the scheme stable. This can be achieved by keeping the Reynolds number low or by increasing the resolution. Since latter is not possible due to the memory limitations of the ODROID-MC1, the Reynolds number is $Re = Re_S/4$, where $Re_S = 7,000$ is the Reynolds number from Schlimpert et al. [35]. Fig. 11 furthermore compares the profiles of the temporally averaged velocity magnitude $< |v| >$ at $z = D$ and $z = 2D$ downstream of the slot to those of [35]. The results are obtained by averaging the solution over 500, 000 LBM iterations starting at $t = 500,000$ and normalizing them by the maximum temporally averaged slot velocity $< v_{max} >$. From both Fig. 11a and 11b it is obvious that the resolution at level l_γ, which resolves the slot diameter by $D = 6 \cdot \Delta x$, is not sufficient to reconstruct the full velocity profile. The maximum normalized velocity for both cases is, however, matched well. It is clear that the velocity profile of the present solution is due to the smaller Reynolds number thinner, i.e., the flow is in the laminar regime. Outside the jet core the velocity is slightly overpredicted. Such a behavior for coarse solutions is also found in [35]. Furthermore, comparing the findings at $z = D$ and $z = 2D$ shows the fine solution at $Re_S = 7,000$ as well as the coarse solution at $Re_S = 1,750$ to feature a more flat velocity profile at $z = 2D$.

5. Summary and conclusion

An LBM simulation code that is usually employed for large-scale flow simulations on HPC machines has been used to measure the performance of an ODROID-MC1 cluster consisting of four ODROID-XU4 nodes. The memory limitation of the cluster allowed to generate a maximum of $38 \cdot 10^6$ cells with a massively parallel grid generator, a computation is, however, only possible on a maximum of $9.2 \cdot 10^6$ cells. This means, that from a memory point of view a single ODROID-MC1 is limited to the simulation of small cases. An extension by further nodes, e.g., by another ODROID-MC1 or additional nodes (ODROID-MC1 Solo) can break this limitation.

A single core performance analysis revealed the fast cores to be roughly 4.4 times faster than the slow cores. Considering the whole node, the fast cores are still a factor of 2.1 faster for both pure MPI or combined MPI/OpenMP measurements. The decrease of the performance difference was due to band-width limitations, especially in the memory-intensive propagation step of the LBM. A change of the OpenMP scheduling for these cases did not show a significant change in the run times. Inter-node performance measurements using either the fast or slow cores revealed using a single MPI rank per node and four OpenMP threads to deliver the smallest time to solution. Amongst all measurements, this configuration showed to have the smallest communication times, hence resulting in the best scaling performance. Using all cores of the cluster and distributing with different OpenMP scheduling schemes showed the `guided` option to distribute the loops the most efficiently due to the best load-balancing withing each MPI rank. Assigning the same amount of computational cells to the fast and slow cores lead to idling fast cores that had to wait for the slow cores. From these findings it was evident that inclusion of the slow cores in the computation does not make any sense, i.e., using only the fast cores lead to the best results. Changing the cell distribution on the fast and slow cores according to the performance difference between the CPUs did not change this fact. That is, despite the additional slow cores represent additional computing power, the computation is limited by the memory bandwidth making an inclusion of the slow cores pointless. Varying cell distributions were tested with two different configurations with a different total amount of computational cells. A comparison of the performance of the ODROID-MC1 to state-of-the-art supercomputers, such as the JURECA and the HAZEL HEN system showed that by using the same amount of cores, the ODROID-MC1 is only ≈ 3.9 slower than the HPC systems. This is due to the slower memory bandwidth and the lower CPU clocking. Looking at the potential of a full JURECA or HAZEL HEN node, a single node leads to a ≈ 6.1 times faster computation, i.e., by using Intel's hyper-threading technology on all 24 available cores compared to the 16 fast cores of the ODROID-MC1. The capabilities for large-scale computations of the code have been shown by high scalabilities across the HPC systems JURECA, JUQUEEN, and HAZEL HEN.
The power consumption has been determined by measuring the $Watts$ for a sample computation on the ODROID-MC1. The findings showed the ODROID system to have a much lower power footprint than Intel-based HPC systems. In more detail, the system consumes $54.5W$ under full load using only the fast cores. This is by a factor of 5.5 below the power consumption of Intel-based nodes assuming a consumption of $300W$. For this case the $Flop/s$ per $Watt$ ratio is better on the ODROID-MC1 than on the HPC systems using the same amount of cores on both the HPC systems and the ODROID-MC1. Considering, however, only a TDP of $160W$ the HPC systems are slightly better than the ODROID-MC1.
It has to be mentioned that the ODROID-MC1 is much more competitive than an HPC node. The current price (status as of Nov. 2018) of an ODROID-MC1, excluding SD-cards, cables, and power supplies is at \$US220. In comparison, the price of a JURECA node, excluding any quantity discounts that are usually given to HPC centers, is at approximately \$US9,000 - \$US10,000.
Finally, the applicability to engineering applications has been shown by running a three-dimensional simulation of the flow in a slot burner configuration.

To summarize, the ODROID-MC1 is a promising system for the simulation of scientific flow problems. Its drawbacks result from the limited amount of available memory, the corresponding memory bandwidth, and the low-performing Cortex-A7 cores. Performance-wise it can compete with single nodes of HPC systems, considering small simulation cases. Its scalability still has to be tested for larger node counts, it is, however, expected that the *Gbit* connectivity is not sufficient to allow for high scalability, and hence for simulation cases that run on current HPC systems. It will certainly be unable to compete with the high memory-bandwidths, high-performance network connectivity, and hyper-threading technologies that are key to HPC systems. It is, however, fair to state that for localized cluster solutions the ODROID-MC1 is definitely a prospective procurement option. This certainly depends on the application. It should be noted that since the LBM investigated in this study operates on unstructured data and uses explicit time stepping, the results can be considered typical for a whole class of simulation codes that are rather memory- than compute-bound. Codes for the simulation of flow problems belong to this class.

6. Outlook

The investigations revealed the propagation step in conjunction with the limited bandwidth being the limiting factors in the performance of the ODROID-MC1. To get a deeper insight, the impact of the memory layout and access patterns on the performance will be analyzed. A detailed inspection of cache line misses and prospective performance gain using prefetching mechanisms will be performed. Since only a single ODROID-MC1 was tested, only constrained information was collected on the scalability of the system. Therefore, further nodes will be added to the cluster to investigate how the inter-node performance (strong and weak scalability) develops for larger node counts and for larger problem sizes. Also taking the step from 32-bit based systems to 64-bit systems suited for HPC, such as the ARMv8 Cavium ThunderX2[3], makes sense. That is, similar comparisons such as performed in this manuscript could help to validate the suitability of ARMv8 systems for scientific computing. Since the present study concentrates on memory-bound computational methods, it furthermore makes sense to extend the investigations to more compute-bound applications in the future.

Acknowledgments

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (GCS)[4], the Jülich Aachen Research Alliance, High-Performance Computing (JARA-HPC)[5] Vergabegremium, and the Partnership for Advanced Supercomputing in Europe (PRACE)[6] for funding this project by providing computing time on the supercomputers JUQUEEN and JURECA at Jülich Supercomputing Centre (JSC), and the HAZEL HEN systems at the High-Performance Computing Center Stuttgart (HLRS) through the John von Neumann Institute for Computing (NIC), the JARA-HPC partition, and PRACE partitions.

Conflict of interest declaration

The authors declare that there is no conflict of interest.

Supplementary materials

Underlying research materials can be obtained by contacting the authors of this manuscript.

[3]Cavium ThunderX2 `https://www.cavium.com/product-thunderx2-arm-processors.html`
[4]GCS `http://www.gauss-centre.eu`
[5]JARA-HPC `https://www.jara.org/hpc`
[6]PRACE `http://www.prace-ri.eu`

Appendix A. Library compilation options

The following libraries have been compiled to a shared location on the Synology NFS server.

The mpich-3.2.1 library is configured by:

```
./configure --enable-mpi-cxx --prefix=MPI_PREFIX --disable-fortran
```

The fftw-3.3.7 library is configured by:

```
./configure --prefix=FFTW_PREFIX --enable-mpi MPICC=MPI_PREFIX/bin/mpicc
```

The parallel-netcdf-1.9.0 library is configured by:

```
./configure --prefix=PNETCDF_PREFIX --with-mpi=MPI_PREFIX
```

The munge-0.5.13 library is configured by:

```
./configure --prefix=MUNGE_PREFIX
```

The pmix-2.1.0 library is configured by:

```
./configure --prefix=PMIX_PREFIX
```

Appendix B. Simulation framwork compilation options on ODROID-MC1

The LBM simulation framework uses GCC 7.2.0 with the following compiler options:

```
-Wall -Wextra -std=c++11 -pedantic -Wshadow -Wfloat-equal -Wfloat-equal
-Wdisabled-optimization -Wformat=2 -Winvalid-pch -Winit-self -Wmissing-include-dirs
-Wredundant-decls -Wpacked -Wpointer-arith -Wnostack-protector -Wstrict-aliasing=3
-Wswitch-default -Wwrite-strings -Wlogical-op -Wno-array-bounds
-fdiagnostics-color -Wlogical-op -Wshift-overflow=2 -Wnull-dereference
-Wunused-const-variable=1 -O3 -fstrict-aliasing -fno-rtti -fno-exceptions
-fomit-frame-pointer -Wno-conversion -Wno-unused-result -Wno-implicit-fallthrough
-Wno-psabi -Wno-cast-align -Wnostack-protector -Wno-maybe-uninitialized
-DCOMPILER_ATTRIBUTES -DUSE_RESTRICT -DNDEBUG
```

Appendix C. Simulation framwork compilation options on JURECA and HAZEL HEN

The LBM simulation framework uses GCC 7.3.0 with the following compiler options on top of those given in Appendix B:

```
-Wcast-align -Wconversion -Wunused-result -Wimplicit-fallthrough -Wstack-protector
```

Appendix D. Simulation framwork compilation options on JUQUEEN

The LBM simulation framework uses Clang 6.0 with the following compiler options:

```
-march=native -mtune=native -std=c++11 -stdlib=libc++ -Wall -Wextra
-pedantic -Wshadow -Wfloat-equal -Wcast-align -Wfloat-equal -Wdisabled-optimization
```

```
-Wformat=2 -Winvalid-pch -Winit-self -Wmissing-include-dirs -Wredundant-decls
-Wpacked -Wpointer-arith -Wstack-protector -Wswitch-default -Wwrite-strings
-Wno-type-safety -Werror -Wunused -Wno-infinite-recursion -fcolor-diagnostics
-Wno-inconsistent-missing-override -Wno-undefined-var-template
```

Appendix E. Slurm configuration

The slurm-17.11.5 scheduler is compiled with the following options:

```
./configure --prefix=SLURM_PREFIX --sysconfdir=/etc/slurm --with-munge=MUNGE_PREFIX
--with-pmix=PMIX_PREFIX
```

The configuration file `/etc/slurm/slurm.conf` contains the options shown in Tab. E.9

A sample hybrid MPI/OpenMP job script using the fast cores of the ODROID-MC1 may look as follows:

```
#!/bin/bash -x
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --output=mpi-out.%j
#SBATCH --error=mpi-err.%j
#SBATCH --time=00:20:00
#SBATCH --partition=batch
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun --cpu-bind=verbose,mask_cpu:0xf0 --mpi=pmi2 ./lbm
```

General options	
ControlMachine	fe
AuthType	auth/munge
CryptoType	crypto/munge
MpiDefault	none
ProctrackType	proctrack/pgid
ReturnToService	1
SlurmctldPidFile	/var/run/slurm/slurmctld.pid
SlurmdPidFile	/var/run/slurm/slurmd.pid
SlurmdSpoolDir	/var/spool/slurmd
SlurmUser	slurm
StateSaveLocation	/var/spool/slurmctld
SwitchType	switch/none
TaskPlugin	task/affinity
TaskPluginParam	sched

Scheduling options	
FastSchedule	1
SchedulerType	sched/backfill
SelectType	select/cons_res
SelectTypeParameters	CR_Core

Logging / accounting options	
AccountingStorageType	accounting_storage/none
ClusterName	odroid
JobAcctGatherType	jobacct_gather/none
SlurmctldDebug	verbose
SlurmctldLogFile	/var/log/slurmctld.log
SlurmdDebug	verbose
SlurmdLogFile	/var/log/slurmd.log

Compute node options

```
NodeName=cl[1-4] CPUs=8 RealMemory=1994 State=UNKNOWN
PartitionName=batch Nodes=cl[1-4] OverSubscribe=EXCLUSIVE \
                    Default=YES MaxTime=INFINITE State=UP
```

Table E.9: Slurm run time options.

References

[1] ARM Architecture Reference Manual, ARMv7-A and ARMv7-R Edition, ARM Ltd., 2014.

[2] ARMv8-A Reference Manual, issue b.a Edition, ARM Ltd., 2017.

[3] G. Halfacree, E. Upton, Raspberry Pi User Guide, 1st Edition, Wiley Publishing, 2012.

[4] A20 User Manual, rev. 1.1 Edition, Allwinner Technology Co., Ltd., 2013.

[5] R. Roy, V. Bommakanti, ODROID-XU4 Users Guide, rev. 20170310 Edition, Hard Kernel, Ltd., 2015.

[6] Cortex-A15 Technical Reference Manual, rev. r0p3 Edition, ARM Ltd., 2011.

[7] Cortex-A7 MPCore Technical Reference Manual, rev. r2p0 Edition, ARM Ltd., 2012.

[8] G. Eitel, R. K. Freitas, A. Lintermann, M. Meinke, W. Schröder, Numerical Simulation of Nasal Cavity Flow Based on a Lattice-Boltzmann Method, in: A. Dillmann, G. Heller, M. Klaas, H.-P. Kreplin, W. Nitsche, W. Schröder (Eds.), New Results in Numerical and Experimental Fluid Mechanics VII, Vol. 112 of Notes on Numerical Fluid Mechanics and Multidisciplinary Design, Springer Berlin / Heidelberg, 2010, pp. 513–520.

[9] G. Eitel, T. Soodt, W. Schröder, Investigation of Pulsatile flow in the Upper Human Airways, International Journal of Design & Nature and Ecodynamics 5 (4) (2010) 335–353. `doi:10.2495/DNE-V5-N4-335-353`.

[10] A. Lintermann, G. Eitel-Amor, M. Meinke, W. Schröder, Lattice-Boltzmann Solutions with Local Grid Refinement for Nasal Cavity Flows, in: New Results in Numerical and Experimental Fluid Mechanics VIII, Springer, 2013, pp. 583–590. `doi:10.1007/978-3-642-35680-3_69`.

[11] A. Lintermann, M. Meinke, W. Schröder, Investigations of Nasal Cavity Flows based on a Lattice-Boltzmann Method, in: M. Resch, X. Wang, W. Bez, E. Focht, H. Kobayashi, S. Roller (Eds.), High Performance Computing on Vector Systems 2011, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 143–158. `doi:10.1007/978-3-642-22244-3`.

[12] A. Lintermann, M. Meinke, W. Schröder, Investigations of the Inspiration and Heating Capability of the Human Nasal Cavity Based on a Lattice-Boltzmann Method, in: Proceedings of the ECCOMAS Thematic International Conference on Simulation and Modeling of Biological Flows (SIMBIO 2011), Brussels, Belgium, 2011.

[13] A. Lintermann, M. Meinke, W. Schröder, Fluid mechanics based classification of the respiratory efficiency of several nasal cavities, Computers in Biology and Medicine 43 (11) (2013) 1833–1852. `doi:10.1016/j.compbiomed.2013.09.003`.

[14] A. Lintermann, W. Schröder, A Hierarchical Numerical Journey Through the Nasal Cavity: from Nose-Like Models to Real Anatomies, Flow, Turbulence and Combustion`doi:10.1007/s10494-017-9876-0`.

[15] A. Lintermann, W. Schröder, Simulation of aerosol particle deposition in the upper human tracheobronchial tract, European Journal of Mechanics - B/Fluids 63 (2017) 73–89. `doi:10.1016/j.euromechflu.2017.01.008`.

[16] A. Lintermann, S. Schlimpert, J. Grimmen, C. Günther, M. Meinke, W. Schröder, Massively parallel grid generation on HPC systems, Computer Methods in Applied Mechanics and Engineering 277 (2014) 131–153. doi:10.1016/j.cma.2014.04.009.

[17] J. Bonet, J. Peraire, An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems, International Journal for Numerical Methods in Engineering 31 (1) (1991) 1–17. doi:10.1002/nme.1620310102.

[18] H. Sagan, Space-Filling Curves, 1st Edition, Universitext, Springer New York, New York, NY, 1994. doi:10.1007/978-1-4612-0871-6.

[19] M. Folk, E. Pourmal, Balancing performance and preservation lessons learned with HDF5, in: Proceedings of the 2010 Roadmap for Digital Preservation Interoperability Framework Workshop on - US-DPIF '10, 2010, pp. 1–8. doi:10.1145/2039274.2039285.

[20] J. Li, M. Zingale, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, Parallel netCDF: A High-Performance Scientific I/O Interface, in: Proceedings of the 2003 ACM/IEEE conference on Supercomputing - SC '03, ACM Press, New York, New York, USA, 2003, p. 39. doi:10.1145/1048935.1050189.

[21] R. K. Freitas, A. Henze, M. Meinke, W. Schröder, Analysis of Lattice-Boltzmann methods for internal flows, Computers & Fluids 47 (1) (2011) 115–121. doi:10.1016/j.compfluid.2011.02.019.

[22] G. Eitel-Amor, M. Meinke, W. Schröder, A lattice-Boltzmann method with hierarchically refined meshes, Computers & Fluids 75 (2013) 127–139. doi:10.1016/j.compfluid.2013.01.013.

[23] Y. H. Qian, Simulating thermohydrodynamics with lattice BGK models, Journal of Scientific Computing 8 (3) (1993) 231–242. doi:10.1007/BF01060932.

[24] A. Dupuis, B. Chopard, Theory and applications of an alternative lattice Boltzmann grid refinement algorithm, Physical Review E 67 (6) (2003) 1–7. doi:10.1103/PhysRevE.67.066707.

[25] M. Bouzidi, M. Firdaouss, P. Lallemand, Momentum transfer of a Boltzmann-lattice fluid with boundaries, Physics of Fluids 13 (11) (2001) 3452–3459. doi:10.1063/1.1399290.

[26] V. Nikl, M. Hradecky, J. Keleceni, J. Jaros, The Investigation of the ARMv7 and Intel Haswell Architectures Suitability for Performance and Energy-Aware Computing, in: ISC 2017: High Performance Computing, Lecture Notes in Computer Science book series (LNCS, volume 10266), 2017, pp. 377–393. doi:10.1007/978-3-319-58667-0_20.

[27] D. Krause, P. Thörnig, JURECA: General-purpose supercomputer at Jülich Supercomputing Centre, Journal of large-scale research facilities JLSRF 2 (2016) A62. doi:10.17815/jlsrf-2-121.

[28] M. Stephan, J. Docter, JUQUEEN: IBM Blue Gene/Q Supercomputer System at the Jülich Supercomputing Centre, Journal of large-scale research facilities JLSRF 1 (2015) A1. doi:10.17815/jlsrf-1-18.

[29] M. Frigo, S. Johnson, The Design and Implementation of FFTW3, Proceedings of the IEEE 93 (2) (2005) 216–231. doi:10.1109/JPROC.2004.840301.

[30] W. Yu, J. Vetter, R. S. Canon, S. Jiang, Exploiting Lustre File Joining for Effective Collective IO, in: Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07), IEEE, 2007, pp. 267–274. doi:10.1109/CCGRID.2007.51.

[31] M. Vázquez, G. Houzeaux, S. Koric, A. Artigues, J. Aguado-Sierra, R. Arís, D. Mira, H. Calmet, F. Cucchietti, H. Owen, A. Taha, E. D. Burness, J. M. Cela, M. Valero, Alya: Multiphysics Engineering Simulation Towards Exascale, Journal of Computational Science (2016) 6–27doi:10.1016/j.jocs.2015.12.007.

[32] C. Moulinec, J. C. Uribe, J. Gotts, B. Xu, D. R. Emerson, Sleeve leakage gas impact on fuel assembly temperature distribution, International Journal of Computational Fluid Dynamics 30 (6) (2016) 419–424. doi:10.1080/10618562.2016.1218481.

[33] G. Wellein, T. Zeiser, G. Hager, S. Donath, On the single processor performance of simple lattice Boltzmann kernels, Computers & Fluids 35 (8-9) (2006) 910–919. doi:10.1016/j.compfluid.2005.02.008.

[34] S. Williams, A. Waterman, D. Patterson, Roofline, Communications of the ACM 52 (4) (2009) 65. doi:10.1145/1498765.1498785.

[35] S. Schlimpert, A. Feldhusen, J. H. Grimmen, B. Roidl, M. Meinke, W. Schröder, Hydrodynamic instability and shear layer effects in turbulent premixed combustion, Physics of Fluids 28 (1) (2016) 017104. doi:10.1063/1.4940161.