SCALABLE

Project Title	SCAlable LAttice Boltzmann Leaps to Exascale
Project Acronym	SCALABLE
Grant Agreement No.	956000
Start Date of Project	01.01.2021
Duration of Project	36 Months
Project Website	www.scalable-hpc.eu

Application performance, accuracy and energy efficiency

D2.3

Work Package	WP 2: Systematic assessment of LBM covers and their extreme scale performance
Lead Author	Lubomir Riha, Gabriel Staffelbach(IT4I)
Contributing Authors	Radim Vavrik (IT4I), Ondrej Vysocky (IT4I), Lubomir Riha (IT4I), Joeffrey Legaux (CERFACS), Markus Holzer (CERFACS), Gabriel Staffelbach (CERFACS)
Reviewed By	Jayesh Badwaik (FZJ)
Due Date	31.8.2022
Date	31.8.2022
Version	1.0

Dissemination Level

☒ PU: Public

☐ PP: Restricted to other programme participants (including the Commission)

☐ RE: Restricted to a group specified by the consortium (including the Commission)

☐ CO: Confidential, only for members of the consortium (including the Commission)

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Deliverable Information

Deliverable	Application performance, accuracy and energy efficiency
Deliverable Type	
Deliverable Title	D2.3
Keywords	benchmark, performance assessment, energy efficiency assessment, READEX
Dissemination Level	Public
Work Package	WP 2: Systematic assessment of LBM covers and their extreme scale performance
Lead Partner	IT4I
Lead Author	Lubomir Riha, Gabriel Staffelbach
Contributing Authors	Radim Vavrik (IT4I), Ondrej Vysocky (IT4I), Lubomir Riha (IT4I), Joeffrey Legaux (CERFACS), Markus Holzer (CERFACS), Gabriel Staffelbach (CERFACS)
Reviewed By	Jayesh Badwaik (FZJ)
Due Date	31.8.2022
Planned Date	
Version	1.0
Final Version Date	31.8.2022

Disclaimer:

The opinions of the authors expressed in this document do not necessarily reflect the official opinion of the SCALABLE partners nor of the European Commission.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Contents

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

List of Figures

List of Tables

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Part 1
Introduction

This document contains the updated benchmark information at M18 when compared to deliverable D2.1 and D2.2. Due to release delays, LABS/ProLB version 2.6 is used for the performance evaluation. For waLBerla version 6.1 is used. Due to the delayed release of version 2.8 of LABS/ProLB, it was not possible to include comparative performance analysis. It will be done in the next document.

Part 2
Updated benchmark suite

The SCALABLE use cases are described in D2.1.

However the initial mesh requirements were suited for smaller core counts, since the maximum number of usable cores at the time peaked at 590 to 1120s. Execution time in LABS/ProLB rendered them unusable in practice for larger core counts due to some inefficient communication schemes whose execution time increased exponentially with the number of cores involved.
While those limitations were lifted, when going for large scalability studies targeting up to 10k cores, those meshes proved rather ineffective as each core would have a very poor balance between the number of internal nodes (used for its local computations) and the number of external nodes (used to communicate values with the relevant neighbouring cores).

With this in mind, the meshes used for the four use cases : COVO, Turbulent Channel, Lagoon and S2A were scaled up to be able to run more efficiently on up to 10k cores of the Karolina system at IT4I.

1 Benchmarks changes

1.1 Lagoon

The original Lagoon mesh consists of 64 465 328 fluid nodes. We modified the min_mesh_size parameter by dividing it by 2, which in turn produced a mesh of 276 999 093 fluid nodes, thereby increase the number of nodes by a factor of 4.

1.2 S2A

The original S2A mesh consists of 62 621 959 fluid nodes. We modified the min_mesh_size parameter value from 0.0025 to 0.0015, which produced a mesh of 276 999 093 fluid nodes, thereby increase the number of nodes by a factor of 4,42.

1.3 TurbulentChannel

The original TurbulentChannel mesh consists of 92 221 800 fluid nodes. We modified the primarySpaceUnit parameter from 0.0005 to 0.00025, and adjusted accordingly the numberOfPrimaryCells parameter from [628, 400, 150] to [1256, 800, 300], leading to a mesh containing 730 615 200 fluid nodes, thereby increase the number of nodes by a factor of 7,92.

1.4 COVO

The original COVO mesh consists of 160 000 fluid nodes. We modified the primarySpaceUnit parameter from 0.005 to 0.0001, and adjusted accordingly the numberOfPrimaryCells parameter from [400, 400, 1] to [20000, 20000, 1], leading to a mesh containing 400 million fluid nodes, thereby increase the number of nodes by a factor of 2500. As the mesh size was dramatically increased, in order to avoid extremely long execution time

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

we also reduced the number of iterations by changing the numberOfSmallTimeSteps parameter from 97 590 down to 1000.

The updated uses cases are available in the repository for the project at IT4I. In order to keep track of the previous benchmarks and their updated versions we have created a new git branch *ProLB_high_nodes_counts* that contains the large scale benchmarks for LABS/ProLB and Walberla. The repository address is the following : https://code.it4i.cz/eurohpc-scalable/showcases/-/commits/ProLB_high_nodes_counts

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Part 3
Theory of Performance Analysis for LBM

This section aims at explaining the attainable performance we can expect for the codes in ideal circumstances.

Optimal program performance can not be achieved without deep knowledge of the algorithm and the hardware it is executed. For this, it is essential to understand the relevant bottlenecks of code execution. Analytic performance modeling provides a useful way of understanding these bottlenecks. While many more and less complex performance models exist, it is crucial to understand that a more complex model is not necessarily better. Overall the idea is to get an upper performance bound for a specific algorithm executed on specific hardware. If a reasonable upper bound can be determined with a simple model, it is sufficient, or in other words, if complex phenomena like cache hierarchy, pipeline latency, or cache miss play no significant role, it is not essential to represent these phenomena in a performance model [6, 13].

In the case of the lattice Boltzmann method, it has been shown that the code is highly memory bound [10, 1, 7]. The roofline model can determine a good upper-performance limit for memory-bound codes[13]. Usually, the first step of the roofline model would be to determine the so-called limiting factor of a code. However, we will not do this but instead directly assume the main memory bandwidth is the limiting factor, as shown in many publications before [7].

To calculate the maximum achievable performance $P_{\max}$ of a main memory-bound code the following equation can be used:

$$P_{\max} = \frac{b_s}{n_b}, \tag{1}$$

where b_s is the main memory bandwidth and n_b is the number of bytes transferred by the algorithm. In the case of this deliverable, all measurements are done with a D3Q19 stencil. Thus in each cell, 19 PDF values are read from the main memory, and 19 PDF values are stored. For double precision computation this results in $38 * 8B = 304B$ per cell.

In the next step, we will show the single node CPU performance and the single GPU performance on the Karolina supercomputer. Table 6 of deliverable 6.2 shows that the single node bandwidth on Karolina is 410 GB/s. With this information, we can directly calculate the expected performance of an LBM code if this bandwidth could be reached:

$$P_{\max} = \frac{410\,\mathrm{GB/s}}{304\,\mathrm{B}} = 1349\,\mathrm{MLups}. \tag{2}$$

Note that the result is obtained in MLups (mega lattice updates per second), a common way to report performance in the context of LBM. From the calculation above, however, it should be clear that this is just a way to report the effectively used bandwidth. Thus bandwidth numbers can be directly transferred to MLups and vice versa with the information of the memory traffic per cell, which is usually determined by the lattice stencil. Note also that high bandwidth utilization does not necessarily indicate optimal program performance. Still, unnecessary memory transfers could happen. This is mainly tackled by calculating the effective bandwidth as done, for example, in [7].

So far, we have done the calculation based on the bandwidth that the hardware vendor provides. In practice, this bandwidth can hardly be ever achieved. For this reason, the widely used STREAM-copy benchmark by Dr. John McCalpin[1] [2] should be employed on the Karolina supercomputer. In figure 1 the STREAM-copy performance from 1 to 128 cores is shown. The maximum bandwidth that can be achieved is 346 GB/s at 16 cores (two cores per NUMA domain) resulting in 1138 MLups. It is important to note that the STREAM-copy benchmark only copies one array element in each loop iteration from a source array to a destination array. While this comes very close to an LBM implementation, there are two major differences. First of all, LB algorithms are usually executed on a 3D domain, and second, in each cell, Q (in our case 19) values are streamed from a source to a destination vector. These differences can already cause major performance problems if not treated correctly. Thus, in addition to the STREAM-copy benchmark that functions as the baseline, we employ a second bandwidth benchmark called D3Q19 stream-copy. This benchmark is an LBM kernel without any computation and streaming pattern. Thus only a 3D copy kernel where 19 values are copied in each inner loop iteration from a source to a destination vector. As shown in figure 1 the performance of the D3Q19 stream-copy is very similar to the STREAM-copy kernel. However, it must be emphasized that this result was impossible to obtain without loop-splitting [1, 12].

[1] https://www.cs.virginia.edu/stream/FTP/Code/stream.c
[2] https://www.tacc.utexas.edu/about/directory/john-mccalpin

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

So far, we have not done any computations, and no LBM streaming pattern has been used. Thus the performance of two more benchmarks should be shown. These are an LBM kernel using the SRT and the Cumulant collision model with the Esoteric Twist streaming pattern[4, 5, 9, 14]. The kernels are chosen for the following reason. The SRT collision model is one of the simplest models, while the cumulant collision model is one of the most complex and compute-intensive models. As pictured in Figure 1 both kernels saturated the memory bandwidth from 16 cores on, achieving the same performance and about 900 MLups per Node. This is consistent with the literature results [8]. Thus performance-wise, the collision model shows no difference. The performance measurements of the D3Q19 stream-copy, SRT and the Cumulant compute kernels are done using generated compute kernels with *lbmpy* [1] and combining them with the waLBerla [2] framework. In all cases, the single node parallelisation is done using OpenMP and the domain is chosen with $1024 \times 1024 \times 512$ grid points.

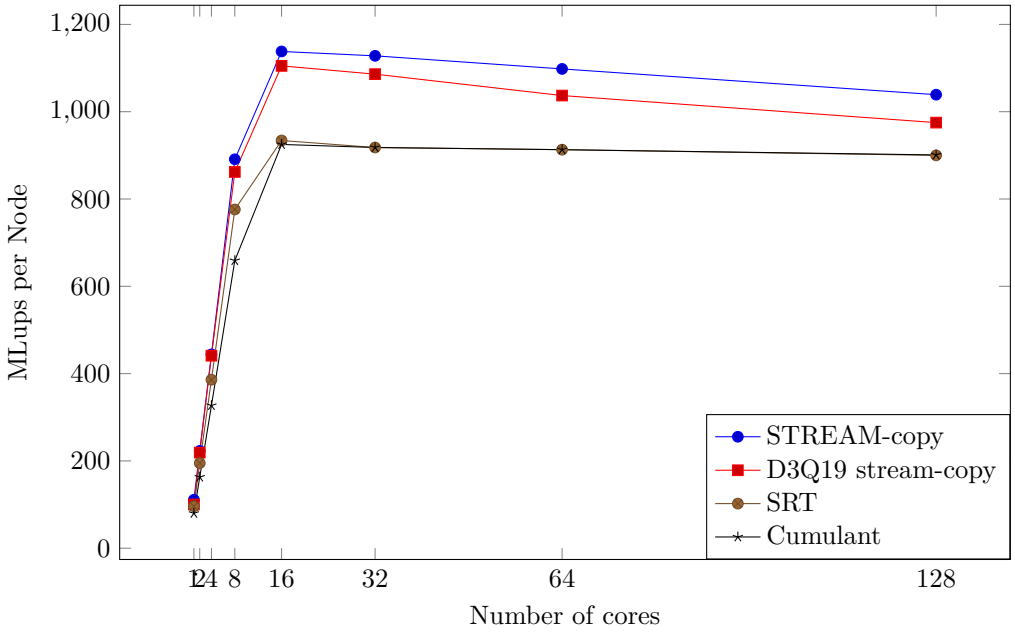


Figure 1: Single node performance benchmark on the Karolina super computer

This provides an idea of the ideal attainable performance on CPU. The next step is to compare the single node performance to the single GPU performance. On the Karolina supercomputer, NVIDIA A100 GPUs are used. Thus the A100 GPU is used for the benchmark. This GPU is expected to deliver up to 1555 GB/s of High Memory Bandwith (HBM2). This results in:

$$P_{\max} = \frac{1555\,\mathrm{GB/s}}{304\,\mathrm{B}} = 5115\,\mathrm{MLups}. \tag{3}$$

Thus, we can expect a performance increase of about 3.79 when comparing a complete CPU node with a single GPU on Karolina. As a next step, this theoretical analysis should be tested by the same benchmarks as shown in Figure 1. For the D3Q19 stream-copy and the SRT and cumulant kernel, we achieve about 4500 MLups. Once again, the performance is completely independent of the collision model and highly dependent on the main memory bandwitdth. Comparing the maximum single node performance of the SRT with the single GPU performance results in an increase of about 4.81. The performance measurements of the GPU version of the D3Q19 stream-copy, SRT and the Cumulant compute kernel are done using generated compute kernels with *lbmpy* [1] and combining them with the waLBerla [2] framework. In all cases the domain is chosen with $448 \times 448 \times 448$ grid points.

These theoretical and hardware benchmark thus provide a framework to analyze the results on the real use cases.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

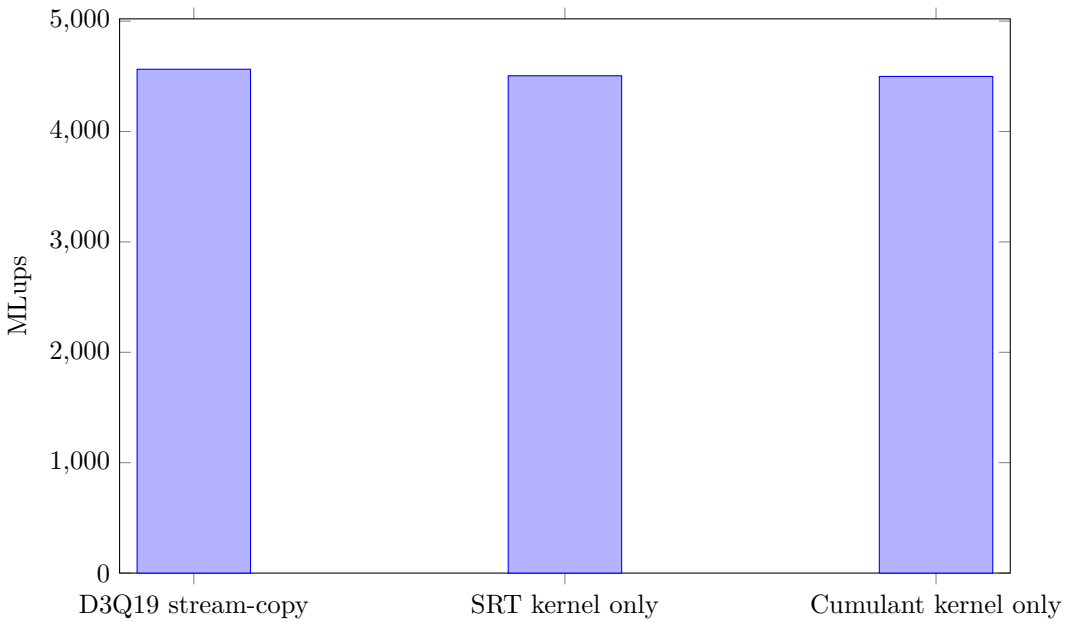


Figure 2: Single GPU performance benchmark on the Karolina super computer (NVIDIA A100 GPU)

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Part 4
Performance, accuracy and efficiency assessment

2 Scalability and Performance Evaluation

2.1 Strong scaling of Walberla and LABS/ProLB up to 10 240 cores

This section provides an updated view of the performance of the codes using up to 10k cores on the Karolina system. It updates results from D2.2 and will be followed in the next sections by updated profiling analysis. Following a consortium meeting, we have also opted to use MLups to represent the performance contrary to the first deliverable.

2.1.1 Lagoon case using LABS/ProLB

The Lagoon case performance has been measured starting from 8 Karolina nodes (1024 cores) up to 80 nodes (10240 cores), using the MLups metric. It increases by a factor of 5,33 while using 10 times more cores, thus providing a 53% scaling efficiency.
Figure 3 illustrates performance on Karolina.

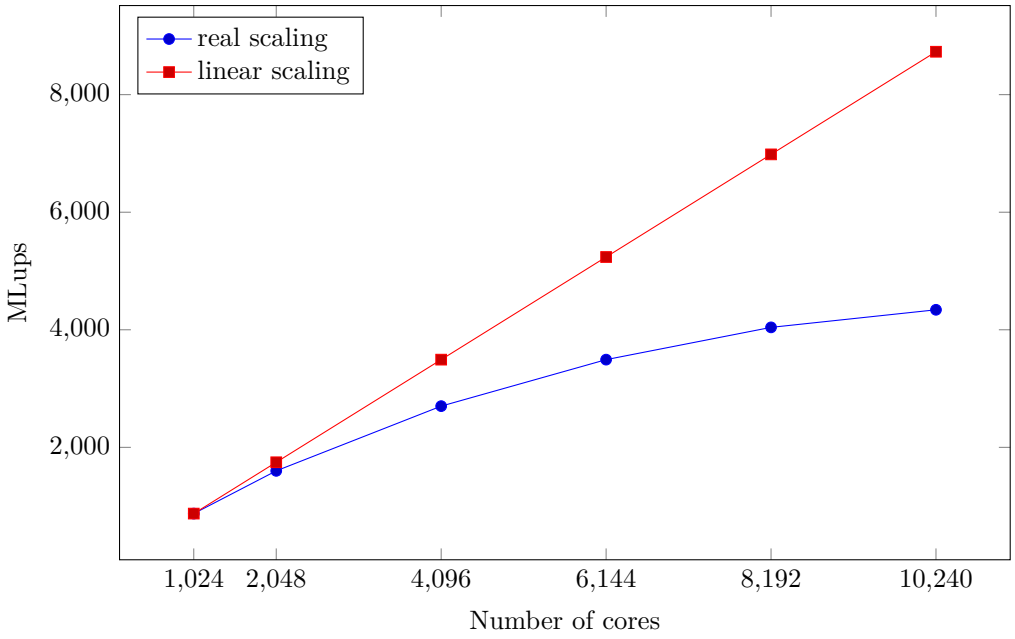


Figure 3: Lagoon performance on LABS/ProLB (MLups)

2.1.2 S2A case using LABS/ProLB

The S2A case performance has been measured starting from 8 Karolina nodes (1024 cores) up to 80 nodes (10240 cores), using the MLups metric. It shows an increase by a factor of 5.25 while utilising 10 times more cores, thus providing an efficiency in scaling of 52 % at that scale. The case could not be run with more than 5000 cores on LABS/ProLB 2.6 due to a bug in the code. It was instead run using a development branch based on LABS/ProLB 2.8 that corrects that bug.
Figure 4 illustrates performance on Karolina.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

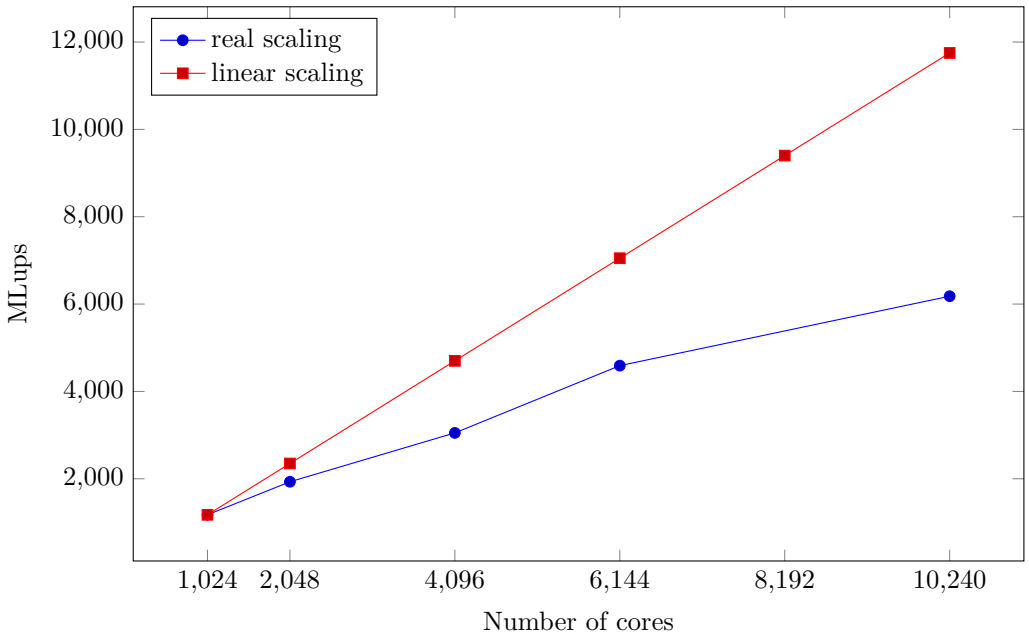


Figure 4: S2A performance on LABS/ProLB (MLups)

2.1.3 TurbulentChannel case using LABS/ProLB

The TurbulentChannel case performance has been measured starting from 16 Karolina nodes (2048 cores) up to 80 nodes (10240 cores), using the MLups metric. It showed an increase by 3.47 while using 5 times more cores, thus providing a scalability of 69 %.
Figure 5 illustrates performance on Karolina.

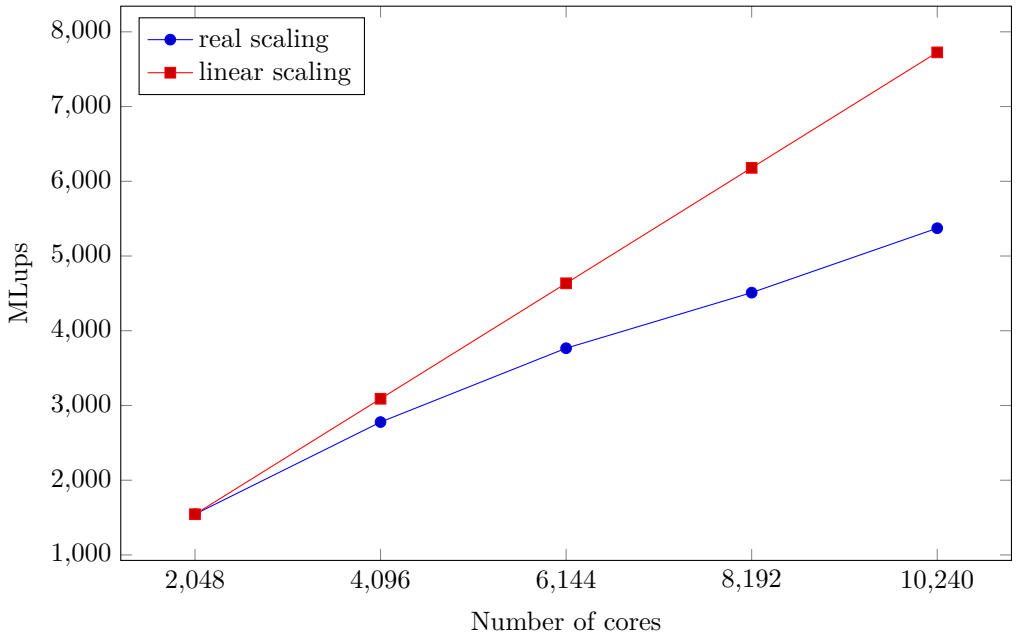


Figure 5: TurbulentChannel performance on LABS/ProLB (MLups)

2.1.4 COVO case using LABS/ProLB

The TurbulentChannel case performance has been measured starting from 16 Karolina nodes (2048 cores) up to 80 nodes (10240 cores), using the MLups metric. It shows an increase by a factor of 6.63 while utilising 10 times more cores, thus providing an efficiency in scaling of 66 % at that scale.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Figure 6 illustrates performance on Karolina.

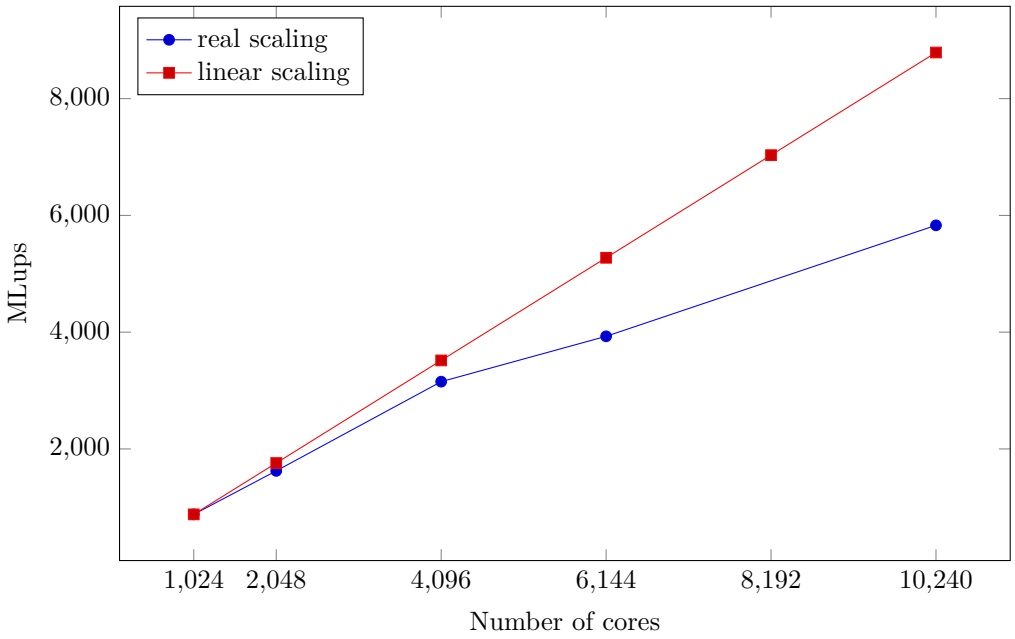


Figure 6: COVO performance on LABS/ProLB (MLups)

2.1.5 Lagoon case using waLBerla on CPU

Again, all use cases in this deliverable are performance using strong scaling. The initial domain contains 282 330 416 lattice Cells. Due to the usage of inplace-streaming, this results in memory consumption of about:

$$B_{\text{mem}} = \frac{282\,330\,416 \cdot 23 \cdot 8\,\text{B}}{1024 \cdot 1024 \cdot 1024} = 48.3\,\text{GiB}. \tag{4}$$

Note here that the memory consumption per cell of 23 values could be reduced to 19 if the density and the velocity were not stored explicitly for the sake of simplicity. Due to the low memory consumption of only about 48.3 GiB, we start the strong scaling benchmark on one compute node. On a single compute node, we achieve a performance of 640 MLups which comes close to the theoretical investigation from section 3. However, in contrast to the microbenchmarks in section 3, boundary conditions and communication come on top in the Lagoon application. We see that these do not generate a large overhead. Furthermore, on 128 nodes (16 384 cores) we achieve a performance of 58 982 MLups which indicates a parallel efficiency of about 72 %. Detailed scaling results are shown in figure 7

2.1.6 S2A case using waLBerla on CPU

As introduced in deliverable D2.1, the S2A test case is executed in a strong scaling scenario on the Karolina supercomputer. The initial domain contains 282 330 416 lattice Cells resulting in a memory consumption of about 48.3 GiB (see equation (4))
On a single compute node, we achieve a performance of 652 MLups which comes close to the theoretical investigation from section 3. Again boundary conditions and communicatio do not generate a large overhead. Furthermore, on 128 nodes (16 384 cores) we achieve a performance of 49 152 MLups which indicates a parallel efficiency of about 59 %. The scaling results are shown in figure 8.

2.1.7 TurbulentChannel case using waLBerla on CPU

As introduced in deliverable D2.1, the TurbulentChannel test case is executed in a strong scaling scenario on the Karolina supercomputer. The initial domain contains 729 000 000 lattice Cells. Using equation (4) this results in about 124.9 GiB. We start the strong scaling benchmark on one compute node. On a single compute node, we achieve a performance of 793 MLups which comes close to the theoretical investigation from section 3. Again boundary conditions and communicatio do not generate a large overhead and the performance results are better than in the Lagoon and the S2A benchmark because the domain is larger and does not contain an

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

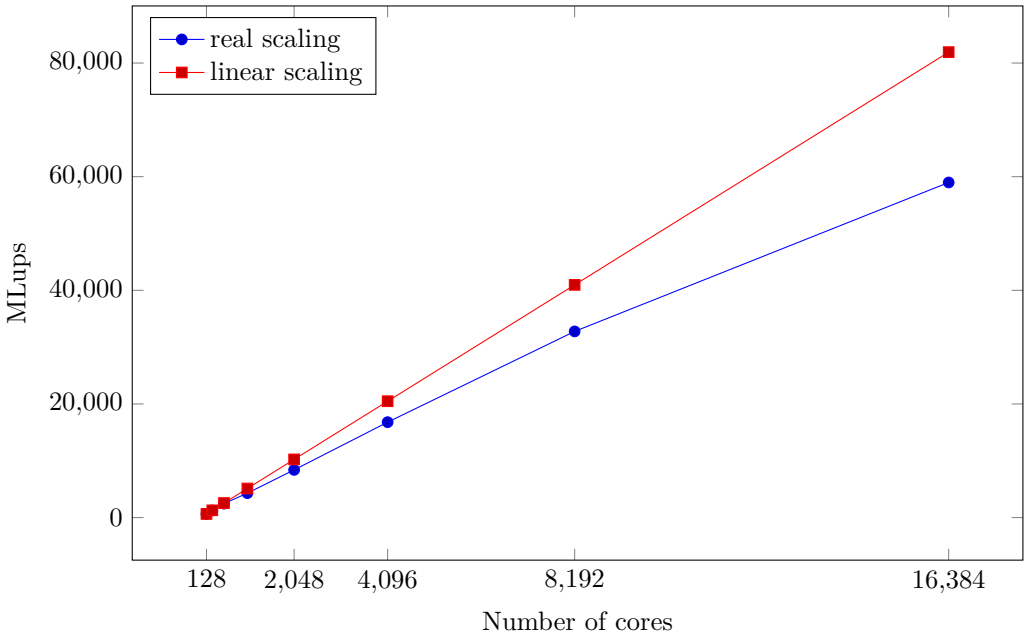


Figure 7: Lagoon benchmark on Karolina supercomputer with waLBerla (MLups)

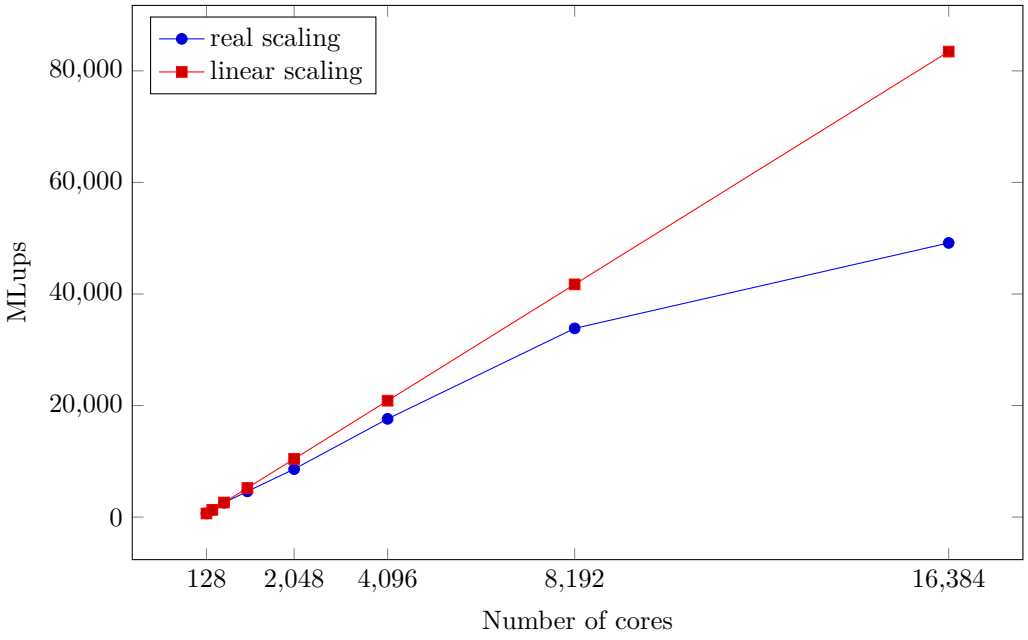


Figure 8: S2A benchmark on Karolina supercomputer with waLBerla (MLups)

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

obstacle. On 128 nodes (16 384) we achieve a performance of 73 728 MLups which indicates a parallel efficiency of about 73 %. The scaling results are shown in figure 9.

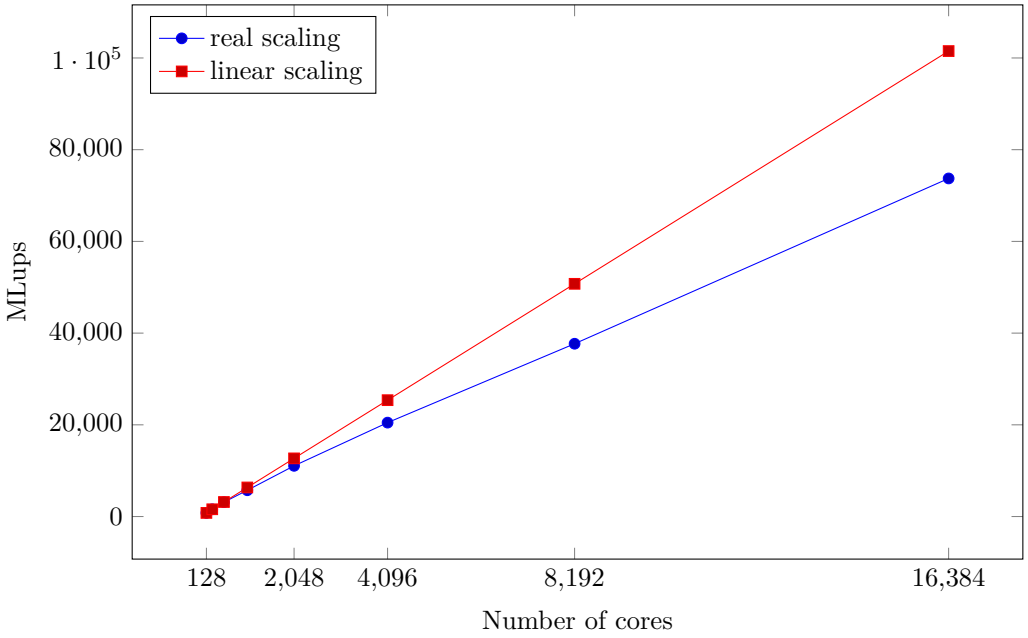


Figure 9: TurbulentChannel benchmark on Karolina supercomputer with waLBerla (MLups

2.2 Memory usage of LABS/ProLB

LABS/ProLB has an internal mechanism that measures and reports the actual memory consumption on every process. On Linux platforms, this is achieved by taking regular samples of the memory information inside the /proc/self/statm file, which is a system file automatically generated by the system for every running process. We will report the amount of memory from the thread with the maximum usage among all.

We mainly focus on the Solver step as it is the part of the application that actually conducts the LBM computations. Other steps manage input, output, pre and post-proc, distribution and balancing of the fluid nodes overs processors etc… As those steps are numerous, we only mention those that consume the most and the least memory.
Table 1 illustrates the range of memory usage for the three test cases Lagoon, S2A and COVO which all have similar profiles. Table 2 illustrates the memory usage for the TurbulentChannel test case which starts at 2048 cores since it cannot be run using less nodes due to the memory requirements that exceed the amount present on the nodes.

The general tendency is that using more nodes decreases memory usage per process, however this decrease is far from linear as using 10 times more processes barely reduces in half the memory footprint.

Table 1: Memory usage of LABS/ProLB on Karolina - Lagoon, S2A and COVO cases

Core number	Solver	Other steps (min)	Other steps (max)
1024	1861-1864	559-592	1809-1949
4096	752-1229	353-358	1176-1602
10240	653-1149	336-371	942-1314

2.3 waLBerla performances and scalability on GPUs

In this section the performance of the TurbulentChannel, Lagoon and S2A test case on NVIDIA A100 GPUs is be evaluated. This has be done on the JUWELS Booster (JSC) and the Karolina supercomputer (IT4I).

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Table 2: Memory usage of LABS/ProLB on Karolina - TurbulentChannel case

Core number	Solver	Other steps (min)	Other steps (max)
2048	1987	510	1827
4096	1307	512	1603
10240	1127	437	1300

2.3.1 TurbulentChannel case using waLBerla - Weak scaling on JUWELS Booster

As a first benchmark we show the TurbulentChannel in a weak scaling scenario. Due to the high parallel performance of GPU architectures large problem chunks must be executed all at once. This was shown for example by Latt et al. [8]. It was shown that at least 2 097 152 cells per NVIDIA RTX 2080 GPU are necessary to saturate the memory bandwidth of the hardware. With this in mind we execute two weak scaling benchmark on the JUWELS Booster supercomputer with 2 097 152 cells per A100 GPU and 16 777 216 cells per GPU respectively. The results are shown in figure 10. The smaller problem size of 2 097 152 cells per GPU is not enough to saturate the bandwidth of the NVIDIA A100 GPU and thus only 2668 MLups can be achieved on a single node. This is only about 59 % when compared to the results of Figure 2. In addition the parallel efficiency on up to 1024 GPUs is only at about 71 %.

Increasing the problem size to 16 777 216 cells per GPU on the other hand results in a single GPU performance of 3568 MLups and thus almost 80 % of the results shown in Figure 2. In addition a parallel efficiency of 99.6 % can be achieved. This shows that the communication overhead can be completely hidden behind the computation provided the problem is large enough to have big compute sections. This also suggests that efficient computing on GPUs will require larger use cases than their CPU counterparts.

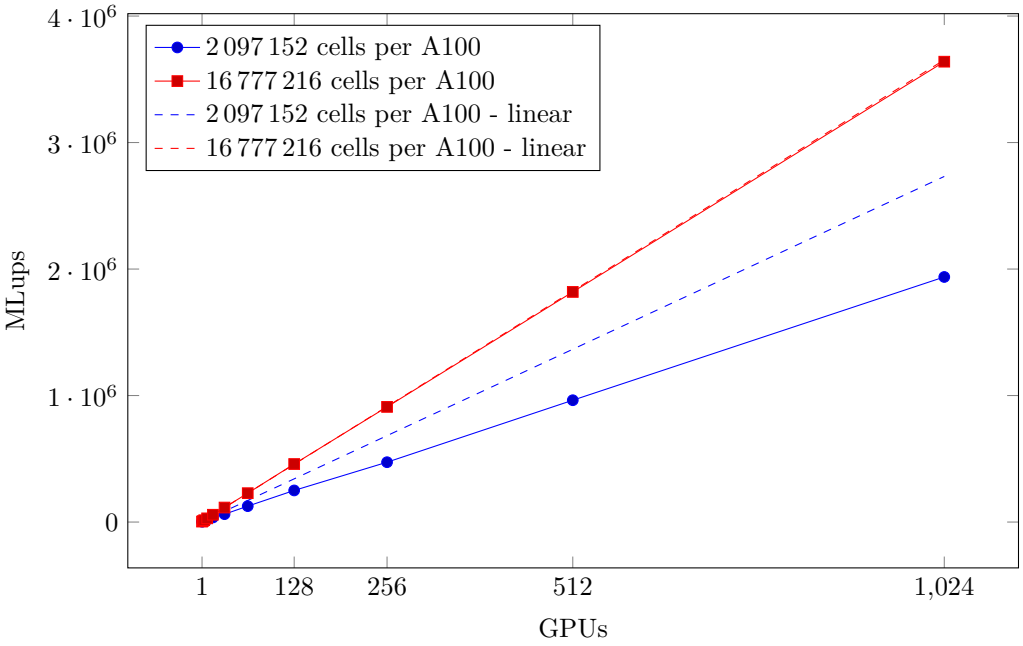


Figure 10: TurbulentChannel weak scaling benchmark on the JUWELS Booster supercomputer with waLBerla and different problem size per NVIDIA A100 GPU. Dashed lines show linear scaling that is calculated from the single GPU performance as basis. With 16 777 216 cells per GPU not only significanlty higher single GPU performance can be achieved but also almost perfect scaling.

2.3.2 Use case waLBerla Weak scaling on JUWELS Booster

With the results from section 2.3.1 we now execute a weak scaling benchmark run for all test cases with a problem size of 16 777 216 cells per A100. The results are shown in Figure 11. With all test cases consistent performance and scalability can be achieved.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

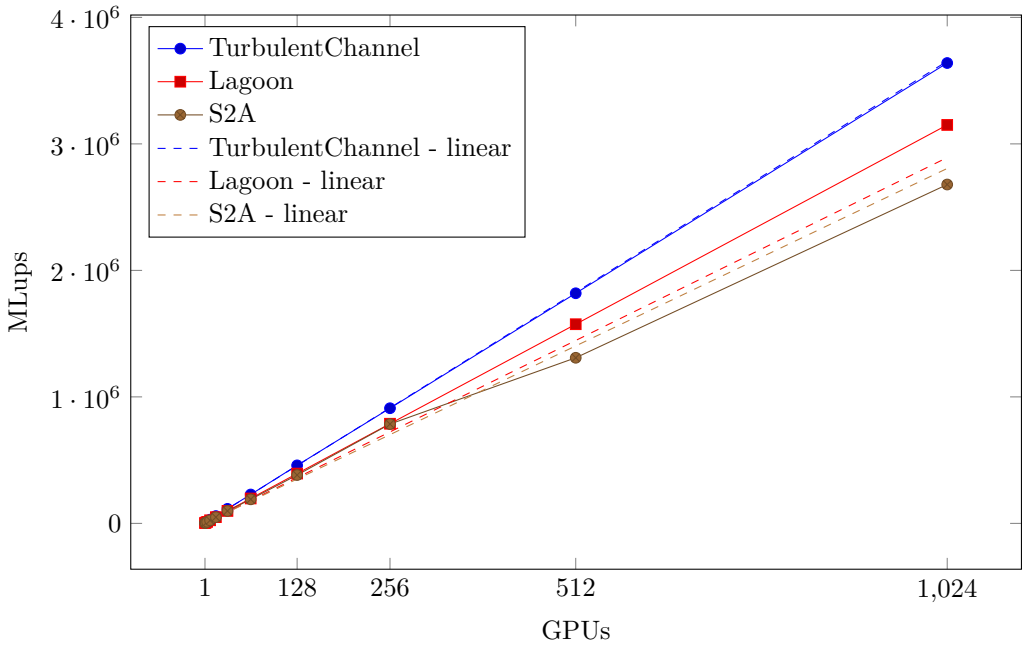


Figure 11: Weak scaling benchmark on the JUWELS Booster supercomputer with waLBerla. Dashed lines show linear scaling that is calculated from the single GPU performance respectively.

2.3.3 waLBerla Strong scaling JUWELS Booster with larger problem size

In this section, the problem size of the uses cases has been modified again to be able to scale up to 1024 GPUs.

In this section a strong scaling benchmark of all test cases is shown. In order to keep a reasonable problem size of at least 2 097 152 cells per A100 at 1024 GPUs we start at 16 GPUs (4 nodes) and fill these 16 GPUs completely. This results in a total problem size of $2048 \times 1024 \times 1024$ cells and thus a total memory consumption of about 368 GiB (see equation (4)):

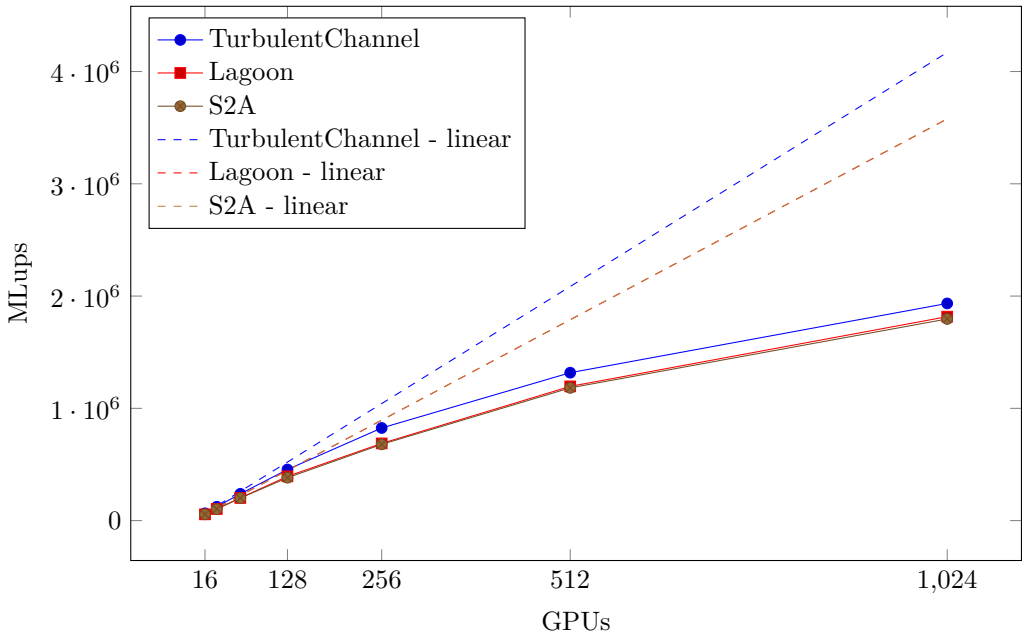


Figure 12: Strong scaling benchmark on the JUWELS Booster supercomputer with waLBerla and larger problem size. Dashed lines show linear scaling that is calculated from the single GPU performance respectively

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

2.3.4 waLBerla Strong scaling JUWELS Booster

This section investigates the strong scaling behavior of the three test cases on JUWELS Booster. The domain size of the TurbulentChannel scenario is the largest, with about 729 000 000 fluid cells. This equals a total memory consumption of about 124 GiB. Thus the problem size fits on four A100 GPUs. For comparing the CPU and the GPU version, we compare a complete CPU node with two sockets to a single GPU. Thus we show the scaling behavior to 128 GPUs. This equals 32 nodes on the JUWELS booster supercomputer. The scaling behavior of the three benchmark cases can be seen in figure 13 and table 3. The scaling efficiency between the CPU and the GPU runs is very similar. For the GPU version, we achieve about four times better performance. These results are consistent with the theoretical investigation of section 3.

Table 3: Comparison Strong Scaling on 128 Nodes between JUWELS Booster (GPU) and Karolina (CPU)

	JUWELS Booster			Karolina CPU		
	TurbulentChannel	Lagoon	S2A	TurbulentChannel	Lagoon	S2A
MLups	368 623	209 512	210 357	73 728	58 982	49 152
Efficiency	73 %	52 %	53 %	73 %	72 %	59 %
Speedup	5	3.5	4.3	-	-	-

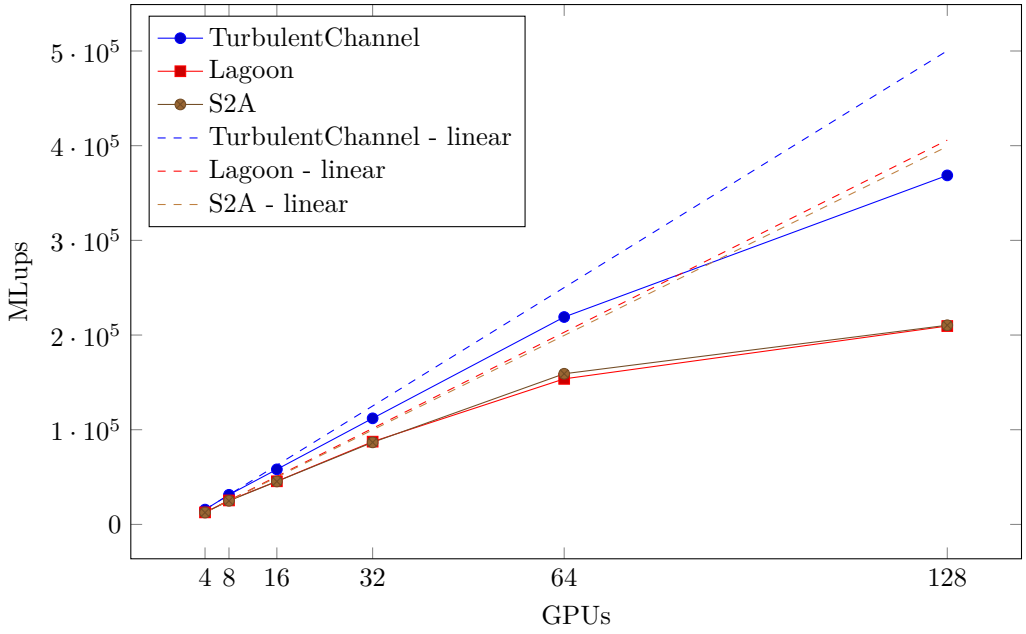


Figure 13: Strong scaling benchmark on the JUWELS Booster supercomputer with waLBerla. Dashed lines show linear scaling that is calculated from the single GPU performance respectively

2.3.5 waLBerla Strong scaling on Karolina GPU partition

We execute the same benchmark as in section 2.3.4 on the Karolina supercomputer. The results are not as regular for the Karolina supercomputer as indicated by Figure 14. As shown in Table 4 we get inconsistent results and no scaling results.

This behavior is caused by the fact that GPU Direct is not yet properly configured and functional on the Karolina GPU partition. (IT4I and HPE support team are working on this issue.) However, based on this observation we can state that the GPU Direct is an essential technology for GPU accelerated version of waLBerla and in general it is very important technology for GPU accelerated LB codes.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Table 4: Comparison Strong Scaling on 128 Nodes between Karolina (GPU) and Karolina (CPU)

	Karolina GPU			Karolina CPU		
	TurbulentChannel	Lagoon	S2A	TurbulentChannel	Lagoon	S2A
MLups	54 601	42 519	75 330	73 728	58 982	49 152
Efficiency	11 %	15 %	20 %	73 %	72 %	59 %
Speedup	0.74	0.72	1.53	-	-	-

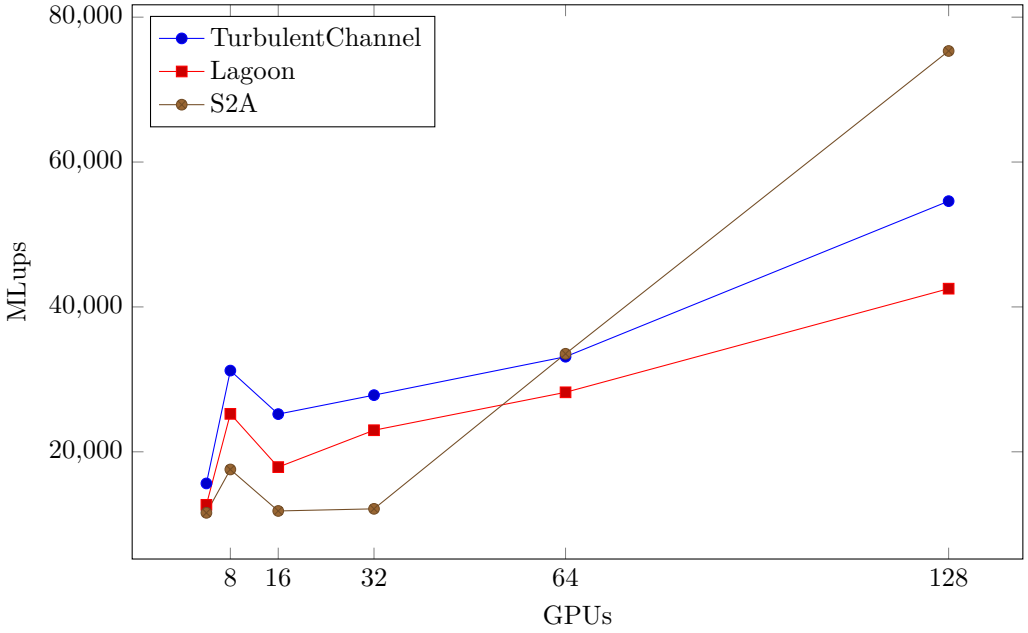


Figure 14: Strong scaling Karolina waLBerla (MLups)

3 Energy efficiency evaluation

In the D2.2 we have analysed the LaBS and CPU version of waLBerla using static tuning (single hardware configuration during the whole application run). In this deliverable we present dynamic tuning of these codes, and moreover static tuning of the GPU version of the waLBerla. In case of the dynamic tuning inside the analysed applications several regions are being identified which cover most of the runtime. In dynamic tuning method we find for each region its own optimal hardware configuration and during the production runs we apply these settings while a region is being executed. This runtime tuning is performed by a runtime system that runs in background of the application. We use MERIC runtime system tool-suite which is developed at IT4Innovations. The tool-suite provides the significant regions identification, resources consumption measurement per region and finally the hardware parameters tuning.

The CPU codes were analysed on the Barbora cluster, which accommodates two Intel Xeon Cascade Lake 6240 per node, that allow to tune CPU core frequency in the range 1.0–3.9 GHz (2.6 GHz nominal frequency, 3.3 GHz peak turbo frequency when all cores are active) and uncore frequency 1.2–2.4 GHz. The essential property of Babora cluster is that its compute nodes are equipped by Atos HDEEM power monitoring system, that provides high-resolution energy measurement of the compute nodes.

The GPU waLBerla runtime and energy consumption were analysed on a single accelerated node of the Karolina cluster – 2x AMD EPYC 7763, 8x Nvidia A100-SMX4. On such hardware we can tune both GPUs' streaming multiprocessors frequency and CPU core frequencies (on AMD Zen3 processors we cannot tune the uncore frequency). Energy consumption has been measured using corresponding performance counters of the GPU (provided by Nvidia Management Library) and CPU (AMD RAPL Package and core power domains).

Definition of the energy-efficiency is usually expressed as "performance per watt" (PPW) unit [3]. In HPC applications we usually measure performance in FLOPs, in case of LBM codes performance is measured in MLups, so we have evaluated the energy-efficiency of the waLBerla and LABS/ProLB as MLups/W. In case of the instrumented CPU codes the energy-efficiency is being evaluated from energy consumption of the solver measured by HDEEM, while the GPU application has not been instrumented, so the MLups/W is evaluated from the energy consumption of the whole application run. Moreover the NVML+RAPL energy measurement method is way less precise then the HDEEM. All the presented energy consumption measure-

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

ments are in Joules[3] as usual for energy-efficient HPC studies.

3.1 Dynamic tuning of the LABS/ProLB (CPU)

LaBS behavior has been analysed using the original version of the S2A and Lagoon (defined in the D2.2) on 4 fully occupied nodes of the Barbora cluster, which has 18 cores on each of two Intel Cascade Lake sockets.

Inside the application 80 regions were identified, combining both automatic static binary instrumentation and manual instrumentation, while most of the runtime is covered by only 9 of them. In contrast to static analysis where each use case has a different optimal hardware configuration, results for both S2A and Lagoon analyses showed similar behavior of the individual regions resulting in the same optimal configuration of each region. Therefore we applied the same optimal region configurations for all tuned LABS/ProLB executions. Example of CPU frequencies scaling on various application regions in the Figure 15 and Figure 16.

Unfortunately we were not able to split the main solver loop into smaller regions, due to following limitations – (1) minimal region runtime of 100 ms to have reliable energy measurement, (2) the same number of region execution by each MPI process per node required by MERIC due to per node energy measurement and per socket uncore frequency tuning. To improve the energy-efficiency of the solver we plan to specify static uncore frequency for the solver, while dynamically tune core frequency individually be each core. Such solution requires modification of the MERIC runtime system, so we plan to present results of such optimisation in the upcoming deliverable D2.4.

From LABS/ProLB's behavior analysis we have identified two configurations for each region. One set of configurations applies strict rule of no runtime extension for any of the regions (2 % of runtime difference considered as a variability in executions), while the second set pushes the energy savings to the maximum possible. Finally, we have compared these dynamically tuned executions with the default and static tuning (universal CPU configuration identified in the D2.2), which is presented in the Table 5 and Table 6. Over 10 % of energy was possible to save without extending the application runtime, while the maximum savings reached 19.5 % of energy with 4.1 % runtime trade-off.

High static savings corresponds with the fact, that most of the application runtime is spend in the solver, which highly influence the static configuration. On the other hand the initialization phase has significantly different behavior, different hardware requirements, which implicates sub-optimal hardware configuration for both of these phases (2.6 GHz core, 2.2 GHz uncore). This problem is solved by the dynamic tuning (3.1 GHz core, 2.0 GHz uncore for the solver for both dynamic configurations), which leads to non-negligible energy savings without runtime extension.

In comparison to the results presented in the D2.2 we have identified single optimal configuration valid for both evaluated usecases, which brings energy savings without extending the application runtime.

Table 5: LABS/ProLB Lagoon comparison of the static and dynamic tuning and their respective impact on energy consumption and performance (runtime).

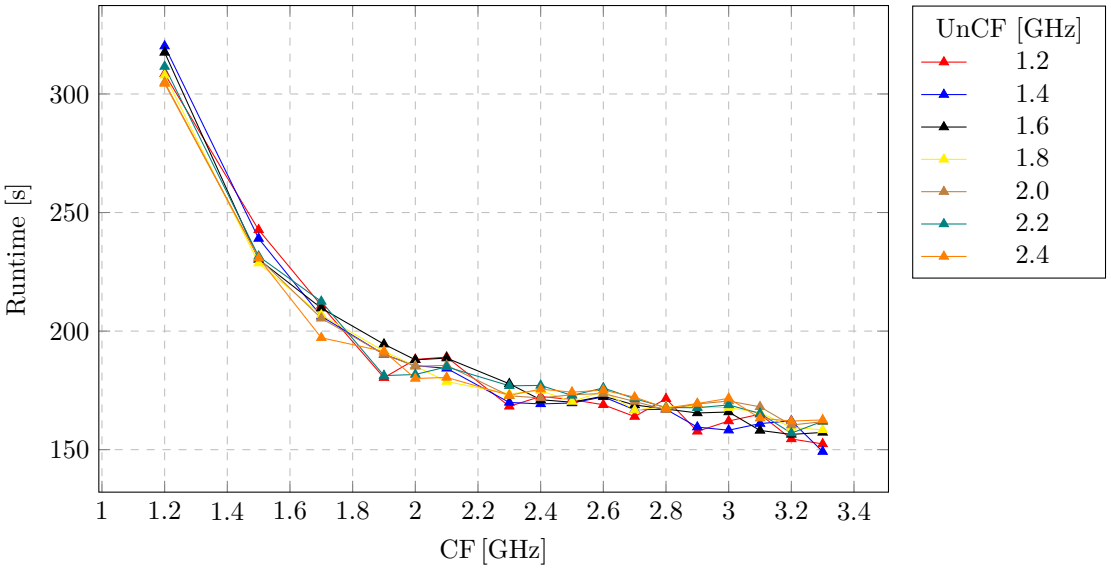
	Default	Static tuning	Dynamic tuning constant runtime	Dynamic tuning
Runtime [s]	1797.9	1942.73	1807.13	1871.14
Energy consumption [kJ]	3102.3	1942.73	2726.7	2496.71
Solver energy-efficiency [MLups/W]	0.054	0.059	0.056	0.056
Runtime extension [%]	-	8.1	0.5	4.1
Energy savings [%]	-	15.1	12.1	19.5

Table 6: LABS/ProLB S2A comparison of the static and dynamic tuning and their respective impact on energy consumption and performance (runtime).

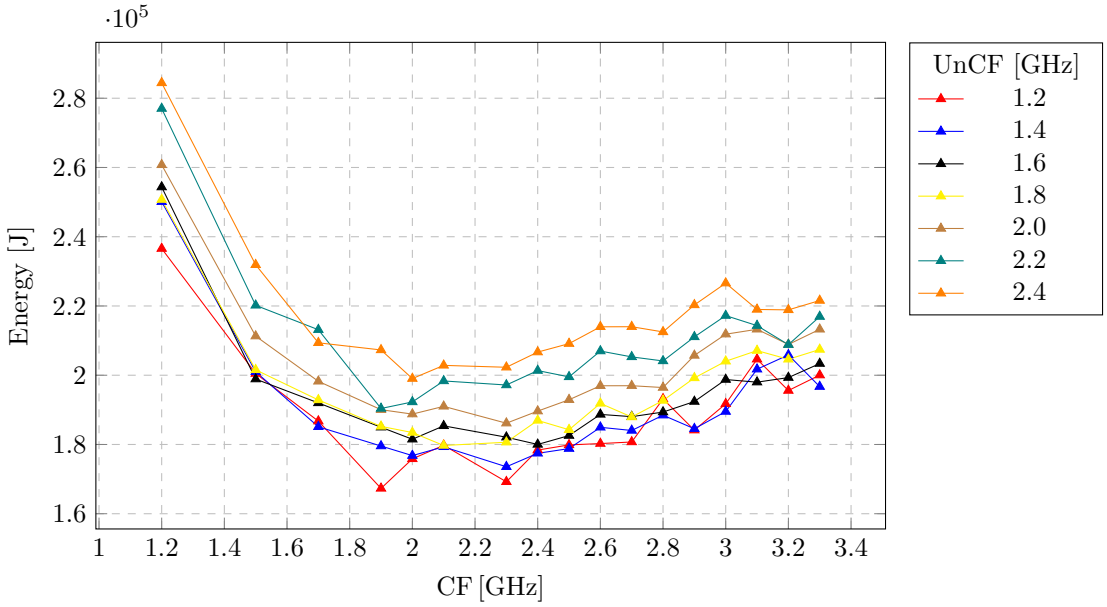
	Default	Static tuning	Dynamic tuning constant runtime	Dynamic tuning
Runtime [s]	1055.2	1160.5	1053.2	1175.9
Energy consumption [kJ]	1786.8	1542.6	1601.7	1505.8
Solver energy-efficiency [MLups/W]	0.052	0.058	0.055	0.055
Runtime extension [%]	-	10.0	−0.2	11.44
Energy savings [%]	-	13.7	10.4	15.7

[3] 1 Wh equals 3600 J.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.



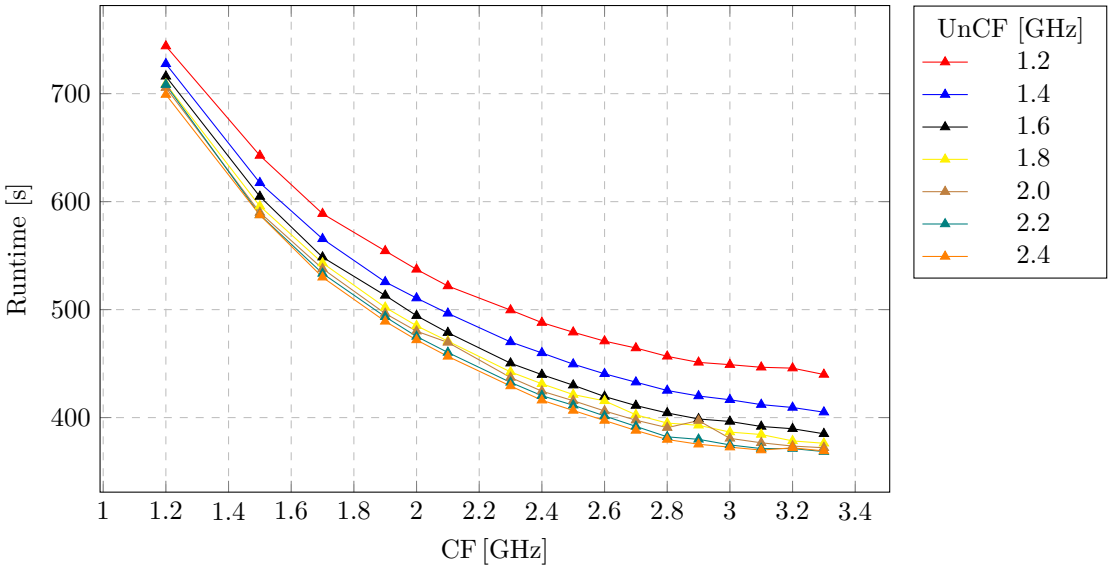
(a) Average runtime.



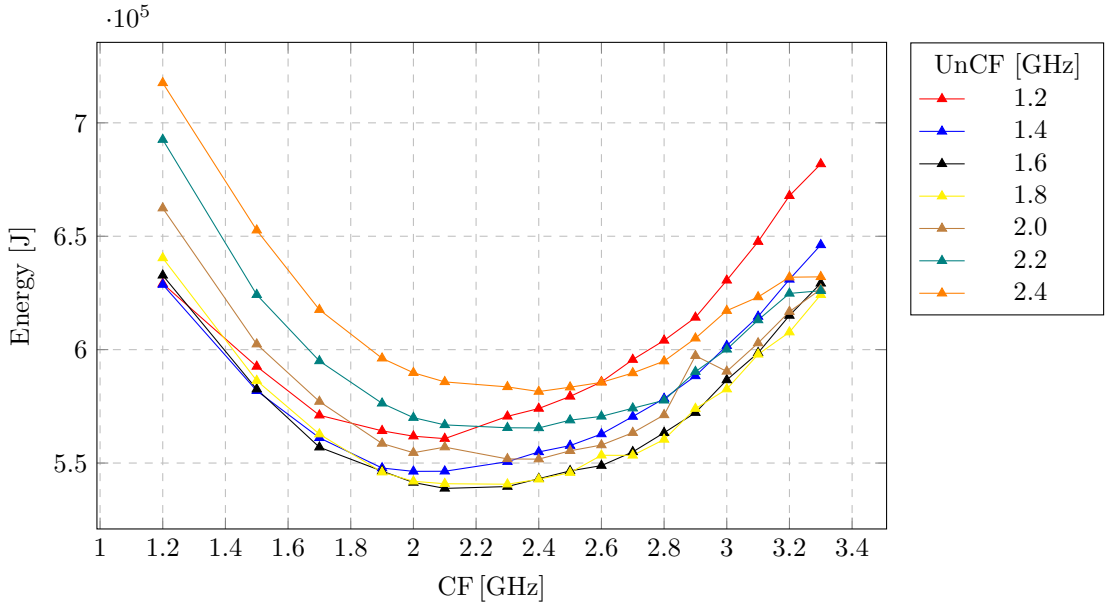
(b) Overall energy.

Figure 15: sequenceFluidLinks region resources consumption when scaling CPU core (CF) and uncore (UnCF) frequencies.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.



(a) Average runtime.



(b) Overall energy.

Figure 16: LABS/ProLB MainSolverLoop region resources consumption when scaling CPU core (CF) and uncore (UnCF) frequencies.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

3.2 Dynamic tuning of waLBerla (CPU)

Same as LABS/ProLB, also waLBerla has been analysed using dynamic tuning, which requires instrumentation of the code. Due to exceptions usage in the waLBerla code, it was not possible to use fully automatic binary instrumentation, because the execution then resulted in runtime error of uncaught exceptions. We have instrumented the code manually, which results in less fine-grain instrumentation than possible. Same as in case of the LABS/ProLB waLBerla solver consists of a single region. From the power consumption timeline presented in the D2.2 it is visible, that the solver should be split at least into two regions. This is our goal for the upcoming D2.4. Despite that the instrumentation is not optimal, we were able to cover 99 % of the application runtime, by regions of mostly unified behavior.

waLBerla also has been evaluated on 4 fully occupied nodes of the Barbora cluster in the default hardware configuration, static configuration of the CPU core and uncore frequencies which was identified as universal optimal static configuration in the D2.2 (2.6 GHz core, 1.8 GHz uncore), and in two different dynamic configurations, specifying hardware configuration of each instrumented region. In the first place, we have identified configurations of the regions, that reduces energy consumption of the regions, but not extend the runtime (solver configuration: 3.0 GHz core, 2.4 GHz uncore). The other one dynamic tuning was focused on maximum energy savings measured by HDEEM, without runtime extension limitation (solver configuration: 1.5 GHz core, 1.6 GHz uncore). Resources consumption of the solver when scaling the CPU frequencies shows Figure 17. These dynamic configurations of the waLBerla regions were applied to both Lagoon and S2A usecases without any modification. waLBerla behavior in these four settings presented in Table 7 and Table 8 for each usecase separately.

These tables present not only energy savings 7.1 % to 19.1 % of the whole application but also energy-efficiency of the solver itself. It is not possible to compare the presented energy-efficiency of the CPU and GPU code, since each version has been measured in a different way (different power monitoring system, non-instrumented GPU version), but it is possible to compare CPU versions of the LABS/ProLB and waLBerla, where waLBerla shows over 10 times higher energy-efficiency for the Lagoon, and nearly 20 times higher efficiency for the S2A usecase.

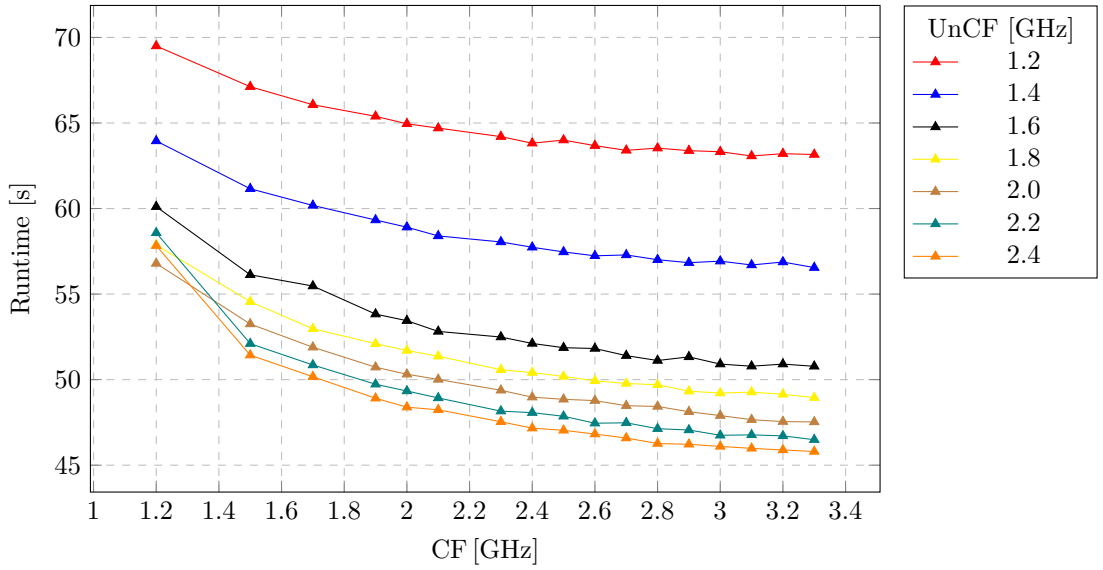
Table 7: waLBerla Lagoon comparison of the static and dynamic tuning and their respective impact on energy consumption and performance (runtime).

	Default	Static tuning	Dynamic tuning constant runtime	Dynamic tuning
Runtime [s]	66.86	70.75	66.64	77.69
Energy consumption [kJ]	99.04	90.88	91.24	80.17
Solver energy-efficiency [MLups/W]	0.548	0.667	0.595	0.734
Runtime extension [%]	-	5.82	−0.4	16.2
Energy savings [%]	-	9.2	7.9	19.1

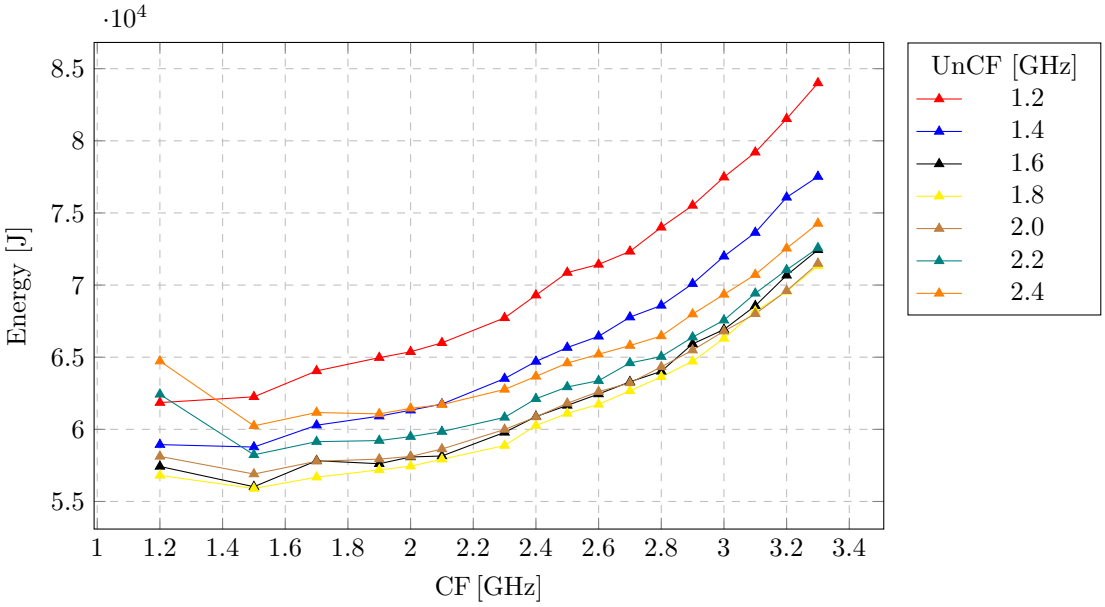
Table 8: waLBerla S2A comparison of the static and dynamic tuning and their respective impact on energy consumption and performance (runtime).

	Default	Static tuning	Dynamic tuning constant runtime	Dynamic tuning
Runtime [s]	91.26	97.67	91.21	111.36
Energy consumption [kJ]	148.53	130.92	136.25	121.71
Solver energy-efficiency [MLups/W]	0.772	0.925	0.814	0.965
Runtime extension [%]	-	7.02	−0.6	22.0
Energy savings [%]	-	11.9	8.3	18.1

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.



(a) Average runtime.



(b) Overall energy.

Figure 17: waLBerla solver region resources consumption when scaling CPU core (CF) and uncore (UnCF) frequencies.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

3.3 Energy efficiency evaluation of GPU accelerated waLBerla

In this case we have executed waLBerla Lagoon and S2A usecases on a single Karolina accelerated node – 2x AMD EPYC 7763, 8x Nvidia A100-SMX4. One MPI process per GPU is used, 4 on each CPU. To improve energy efficiency of the application we have specified the frequency of the GPUs' streaming multiprocessors (SMs), which is an analogy to the CPU core frequency. For this purpose we use Nvidia Management Library (NVML) which provides function *nvmlDeviceSetApplicationsClocks()* that can set a specific clock speed of a target GPU for both (i) its memory and (ii) its SMs'. However, the A100-SMX4 uses HBM2 memory, which is not possible to tune, in contrast to GDDR memory. Therefore on data-center grade GPUs, like A100, only SMs' frequency can be tuned. We perform static tuning, which means that a constant single frequency is used during the application execution.

By default the GPU runs uses the maximum turbo frequency of 1.410 GHz (if not forced to reduce the frequency by power consumption exceeding power limit or by thermal throttling) when it is under a load and switches to nominal frequency of the GPU (1.095 GHz) when copying the data to/from the GPU memory.

Figure 18 and Figure 19 shows application runtime and energy consumption of the GPUs (NVML) plus CPUs (AMD RAPL Package + Core power domains). The power consumption of the remaining active components of the server (mainboard, cooling fans, NICs, etc.) which remains static is not measured. For this fat GPU accelerated server it is approximately 600 W.

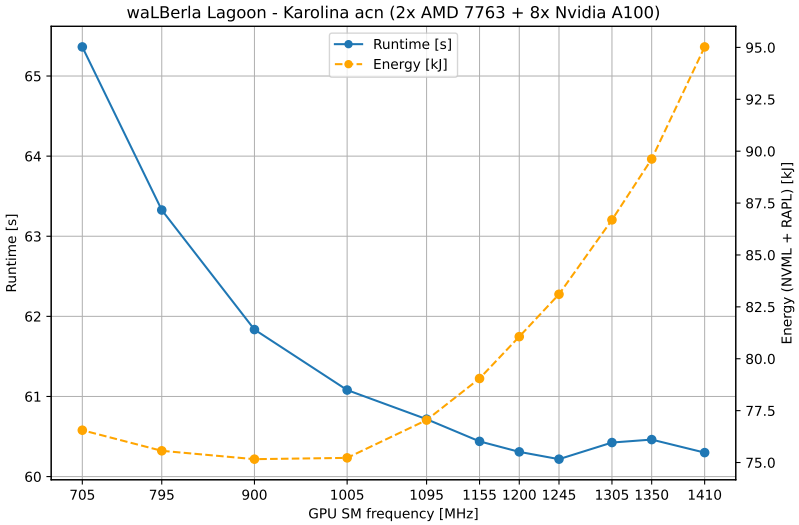


Figure 18: Energy consumption and simulation runtime of the Lagoon usecase when scaling GPUs' SMs frequency.

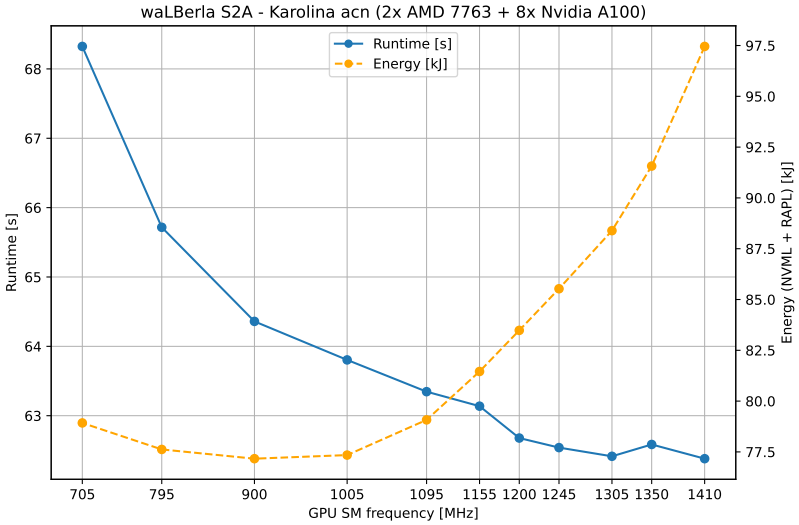


Figure 19: Energy consumption and runtime of the S2A usecase when scaling GPUs' SMs frequency.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

In contrast to static tuning of waLBerla CPU version presented in the D2.2, both evaluated usecases behaved similar, which is clearly visible from the Figure 20 presenting energy-efficiency of waLBerla in various GPU frequencies. Be aware, that because of the missing instrumentation, the energy consumption measurement was not done for the solver, but for the whole application run. We may compare the results between each other, but not with the results presented for CPU waLBerla and LABS/ProLB analyses. Moreover the CPU codes were analysed using HDEEM, providing complex power monitoring of the compute node, which is not case of the performance counters of both NVML and RAPL.

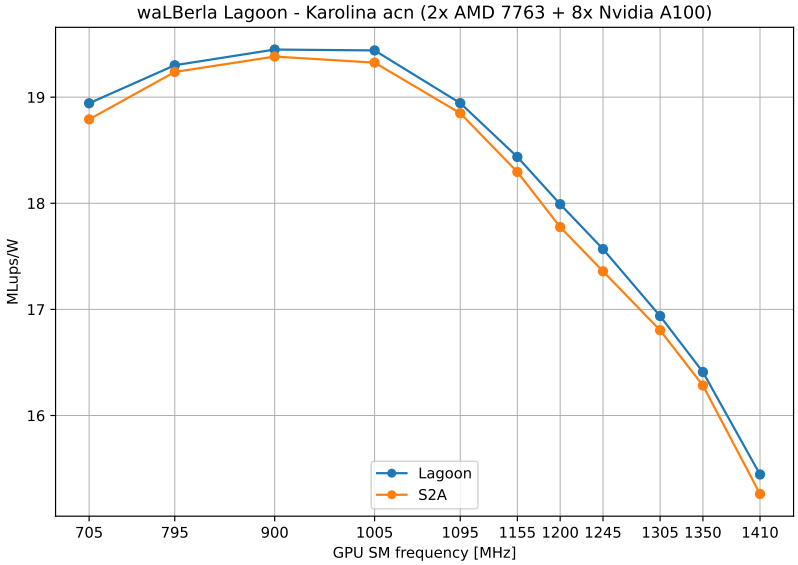


Figure 20: Energy-efficiency of the whole application run of waLBerla when scaling GPUs' SMs frequency.

Based on the presented measurements we have identified that the optimal configuration of the A100 SMs' frequency is 1.005 GHz, in which the application reaches almost the best energy-efficiency, reduces energy consumption about 20 % (Lagoon 19.8 %, S2A 20.6 %), while the runtime is extended about slightly over 2 % only. Table 9 compares the resources consumption in the default and the optimal 1.005 GHz SM's frequency.

Table 9: waLBerla GPU energy savings when scaling SM frequency.

	Lagoon		S2A	
	Default	Static tuning	Default	Static tuning
Runtime [s]	60.3	61.1	62.4	63.8
Energy consumption [kJ]	95.0	75.2	97.5	77.3
Energy-efficiency [MLups/W]	15.4	19.4	15.3	19.3
Runtime extension [%]	-	2.2	-	2.3
Energy savings [%]	-	19.8	-	20.6

For the optimal SM frequency we have evaluated also CPU core frequency scaling, expecting that the maximum boost frequency is not necessary to reach maximum performance that is delivered by GPUs. AMD EPYC 7763 nominal frequency is 2.45 GHz, while the CPU can run up to 3.525 GHz boost frequency.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

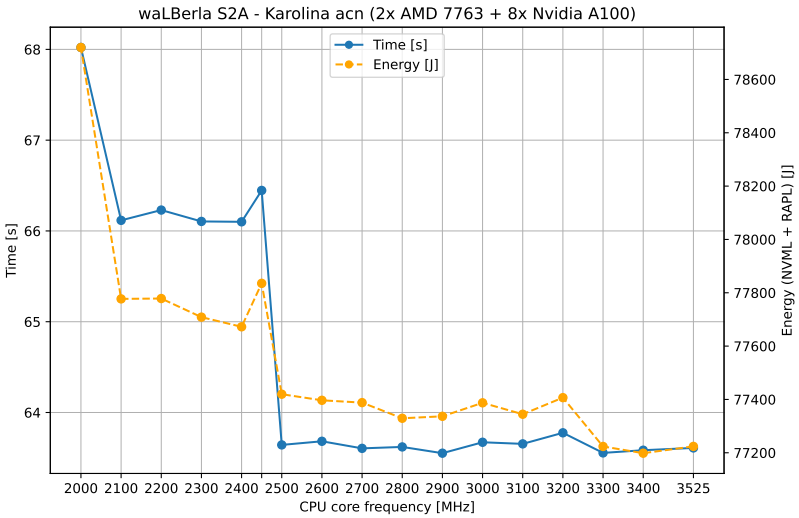


Figure 21: Runtime and energy consumption of the waLBerla S2A usecase using the optimal (1.005 GHz) GPU frequency when CPU core frequency scaling.

Contrary to Intel processors in Barbora, the CPU core frequency of AMD EPYC CPUs in Karolina cannot be locked, but they can be limited from the top. The Figure 21 presents runtime and energy consumption (NVML + RAPL Package and core power domains) for various CPU core frequency limits (still using the GPUs). Decreasing the CPU core frequency does not bring any energy savings. The most significant explanation we reckon, is that only 4 cores out of the 64 of each CPU are under a load, and the remaining cores are idle. We have used the default affinity, scatter threads by NUMA domains (no more than one active thread per Core Complex (CCX)), and no threads pinning. If the active threads would be located closer to each other, the impact of the frequency scaling could be more visible. For the purpose of this analysis it seems only GPU frequency matters when looking at energy efficiency.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

4 POP and DRAM bandwidth analysis

This section follows on from Section 4 of D2.2, where the application structure identification and initial performance assessment of all the test cases were conducted. Also the POP performance model was introduced in Section 3 of D2.2 and the presented results including the POP performance metrics will be based on that description of the model. In the following subsections, we focused only on the Lagoon test case as it is a representative complex simulation example. The analysed application was LABS/ProLB version 2.6, in particular, the iterative phase of the code – the solver. It is currently a pure MPI parallel code.

All the performance data for this section were collected on the CPU partition of Karolina cluster at IT4I, i.e using compute nodes equipped with 2x AMD EPYC 7H12 64-core 2.6 GHz processors and 256 GB DDR4 3200 MT/s memory that are interconnected through the 100 Gb/s InfiniBand HDR100 network.

The DRAM bandwidth analysis was performed by implementing a manual instrumentation in the LABS/ProLB code that accesses the AMD Zen2 DFPMC (Data Fabric Performance Monitoring Counters) hardware counters via MSR-SAFE kernel module [11]. This approach was used as an alternative to the Likwid (likwid-perftr) tool, which does not fully support the NPS4 socket configuration on Karolina cluster, or PAPI tool, which does not provide access to AMD Zen2 uncore performance counters.

The POP-based metrics were obtained using the BSC performance tools Extrae, Paraver, and Basic Analysis.

Please note that the presented results includes small performance degradation caused by the overhead of the particular data collecting approach.

4.1 LABS/ProLB: Analysis on the original use case

The need for the large core-count simulations up to 10k cores required a transition from Barbora cluster to larger Karolina cluster. This brought the need to re-evaluate performance of the original-size test cases on the new Karolina cluster. The two sets of measurement can be used for a direct comparison of the two machines with respect to the LABS/ProLB performance and scalability.

The first test was held in order to evaluate the single-node scaling and to determine the optimal number of MPI processes to be used for the optimal utilisation of the node memory bandwidth. Actually, the test had to be executed on 2 nodes because the minimal memory requirements of the test case exceeds the memory available on a single node. The utilisation of the maximum number of DRAM channels was ensured by a proper process pinning using Likwid (likwid-pin) tool in the manner of spreading adjacent processes across the sockets with the highest possible number of free cores between each other.

The original Lagoon test case does not scale perfectly on per-node basis as shown in Figure 22, but since one node is the smallest resource unit one can allocate, it is still most efficient to use the full node, i.e. 128 processes (active cores) per node. The use of 64 processes only degrades the overall performance by ~24% for that case, but this might potentially change with higher node count.

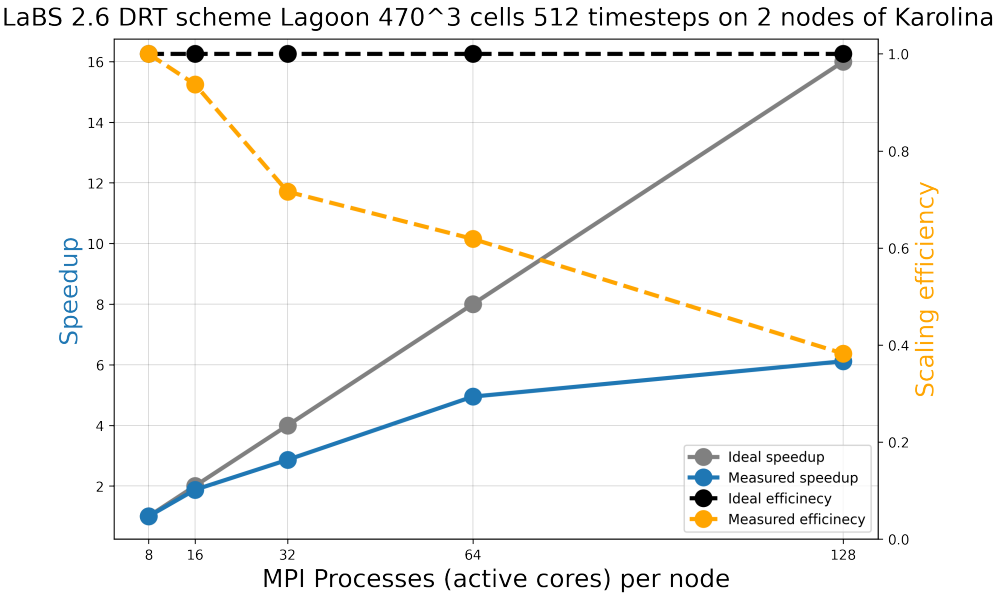


Figure 22: LABS/ProLB original Lagoon scaling on 2 nodes.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Figure 23 shows that the performance in MLups perfectly correlate with the memory bandwidth, which confirms the theoretical assumptions in Section 3. Despite the correlation, the observed bandwidth utilisation is rather far from the stream benchmark bandwidth values – it is only 22 % of the stream values for 16 processes per node (2 processes per NUMA domain) and 75 % for 128 processes per node.

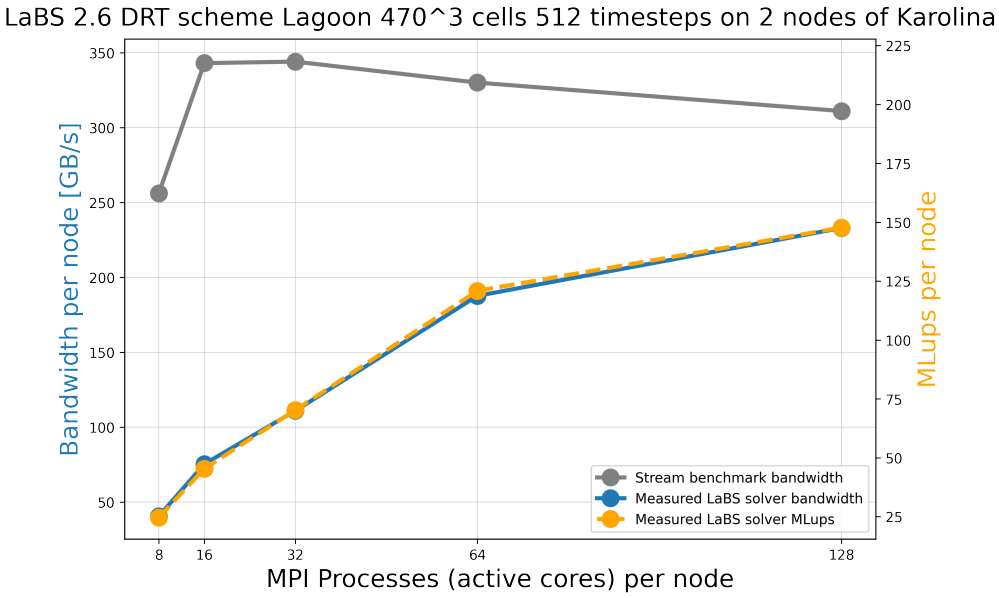


Figure 23: LABS/ProLB original Lagoon DRAM bandwidth vs MLups on 2 nodes.

We brought the Barbora measurements from D2.2 to Figure 24 to illustrates the difference in speedup and its efficiency using 2 - 32 full nodes, i.e. 36 processes per node in case of Barbora cluster and 128 processes per node in Karolina cluster. More than 8 nodes can be considered inefficient in case of Karolina for the test case as the efficiency drops below 80 %. Scaling efficiency on Barbora degrades more slowly meaning the hardware is easier to utilise with scale. The difference is given mostly by the almost 3.6x higher number of cores per node on Karolina and therefore a significantly higher pressure is on the MPI parallel implementation.

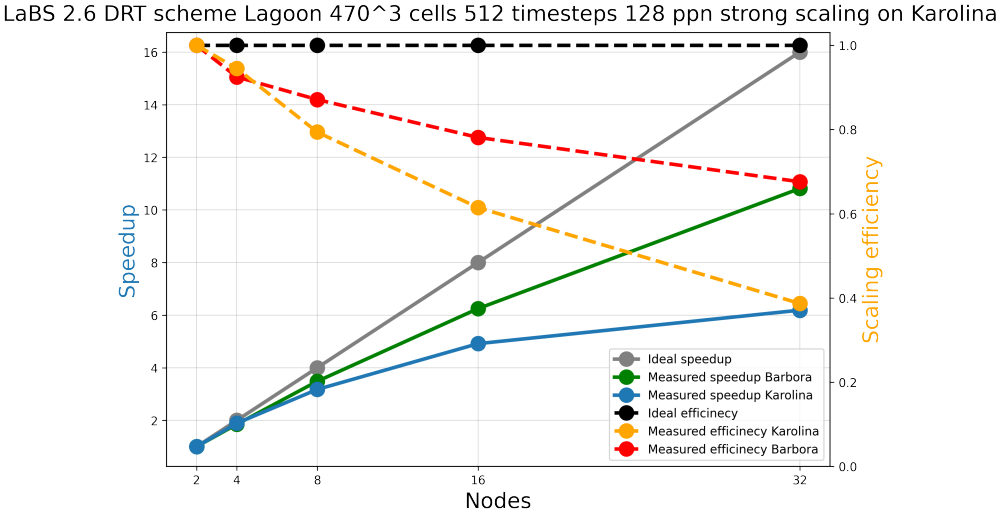


Figure 24: LABS/ProLB original Lagoon scaling on 2 - 32 nodes (128 processes per node).

The scaling analysis in Figure 25 again confirms the strong correlation between the bandwidth and the performance in MLups. The grey line is the extrapolated stream benchmark bandwidth, which represents the upper achievable limit of any memory-bound code. The total maximum bandwidth measured on 32 nodes reaches 36 % of the stream. The black line represents the upper limit of LABS/ProLB performance in terms of linear scaling based on the base run. The 32 nodes run then gives 38 % of that linear scaling. The overall ideal theoretical performance would be even further – 1349 MLups per node as stated in Equation 2, or 1138 MLups per node as measured by the stream benchmark in section 3.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

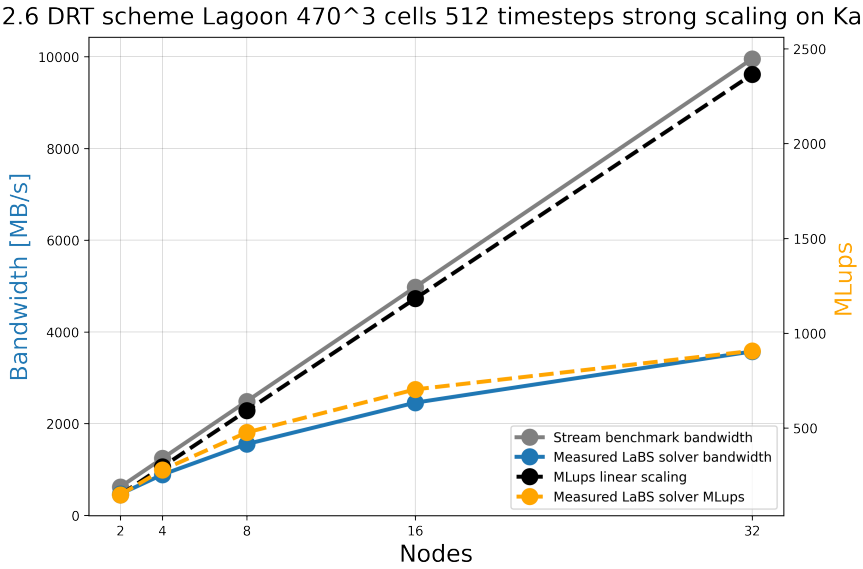


Figure 25: LABS/ProLB original Lagoon DRAM bandwidth vs MLups on 2 - 32 nodes (128 processes per node).

Due to large disk, memory, and time consumption of the tracing and analysis, the POP metrics presented in Figure 26 were obtained from only 20 consecutive time steps (256 - 275). This was ensured by manual instrumentation of the LABS/ProLB code using Extrae API. The metrics for the beginning, middle, and the end part of the solver were experimentally compared and the obtained values were very similar, thus we assume the selection of 20 time steps is representative for the POP metrics evaluation. The main limiting factors of the scaling according the POP efficiency model is Load balance and Transfer efficiency. The same, except the significant frequency degradation, was observed also on Barbora cluster as reported in D2.2. Table 10, which is the base source for the Computation scalability section of the POP metrics, shows a decrease from 3.2 to 1.87 GHz. Note the dynamic frequency scaling (turbo frequency mode) was enabled on both machines. We can also see quite low IPC values that grows with scale, but still only up to 1.

Table 10: Overview of the Speedup, IPC and Frequency of LABS/ProLB original Lagoon 20 time steps on 2 - 32 nodes.

Number of processes	256	512	1024	2048	4096
Number of nodes	2	4	8	16	32
Elapsed time [s]	0.88	0.44	0.27	0.17	0.14
Speedup	1.00	2.01	3.22	5.18	6.41
Scaling efficiency	1.00	1.01	0.81	0.65	0.40
Average IPC	0.66	0.70	0.77	0.85	1.02
Average frequency [GHz]	3.20	3.15	2.87	2.46	1.87

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

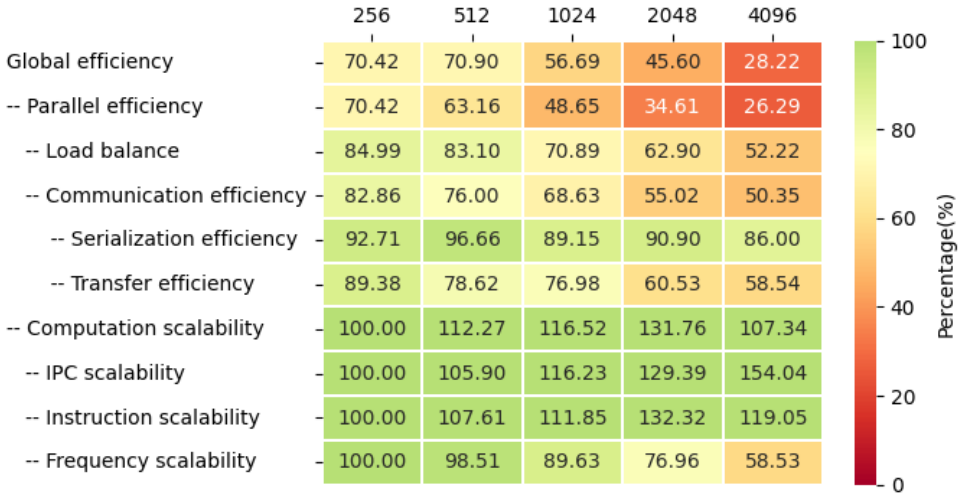


Figure 26: LABS/ProLB original Lagoon Efficiency metrics of 20 time steps on 2 - 32 nodes (128 processes per node.)

Aside from the efficiencies, when we directly compare the solver execution times on Barbora and Karolina as we see in Table 11, the full Karolina nodes perform 1.39x - 2.49x faster over a full Barbora nodes. Due to large number of total processes, the scaling efficiency drops faster on Karolina cluster.

Table 11: Comparison original Lagoon Strong Scaling Barbora CPU (36 MPI processes per node) - Karolina CPU (128 MPI processes per node).

Nodes	Barbora 20 timesteps [s]	Karolina 20 timesteps [s]	Speedup over Barbora	Barbora speedup efficiency	Karolina speedup efficiency
2	15.01	6.16	**2.44**	**1.00**	**1.00**
4	8.12	3.26	**2.49**	**0.92**	**0.94**
8	4.31	1.94	**2.22**	**0.87**	**0.79**
16	2.40	1.25	**1.92**	**0.78**	**0.61**
32	1.39	1.00	**1.39**	**0.68**	**0.39**

4.2 LABS/ProLB: Analysis on the scaled up use case

The first set of measurements with the enlarged Lagoon test case used 128 processes per node, since that configuration turned out to be the most efficient for the original test case.

Figure 27 shows that the scaling drops below 80 % efficiency with 32 and more full nodes. This is ~2x higher scaling efficiency than in case of the original Lagoon test case with ~4.3x less fluid nodes.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

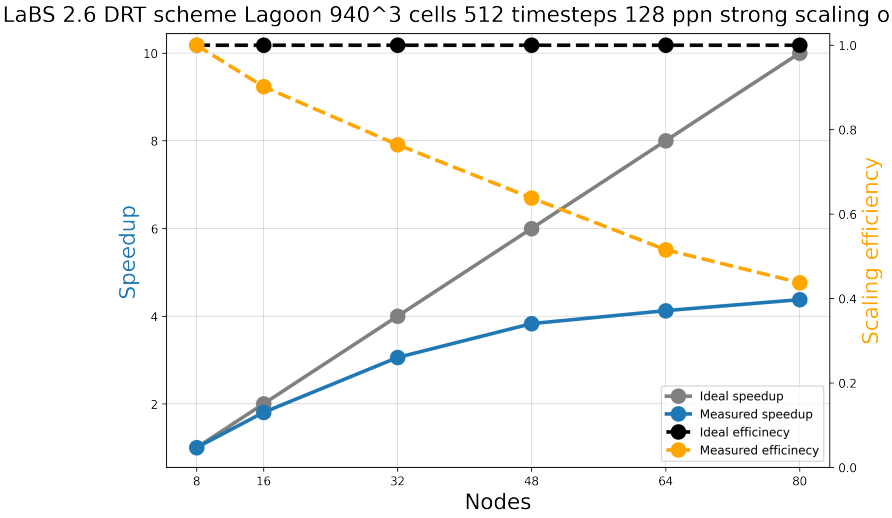


Figure 27: LABS/ProLB scaled up Lagoon scaling on 8 - 80 nodes (128 processes per node).

Figure 28 proves that the correlation between performance in MLups and memory bandwidth holds even for the scaled up test case, with just small deviation on 32 and 48 nodes. We obtained the maximum per node performance 111 MLups on 8 nodes, which is ~10 % of the stream-based theoretical peak value, while the maximum total performance was reached on 80 nodes, i.e 10240 MPI processes, with near to 4000 MLups corresponding to ~4 % of the stream-based theoretical peak value. We also observed quite significant variability in computation time for large node-count runs, that in turn also affects the MLups values. There is no clear justification, but it could be explained by network contention.

From the bandwidth point of view, we measured the maximum 222 GB/s per node on 8 nodes that is ~71 % of the stream benchmark value. The maximum total bandwidth 9528 GB/s obtained on 80 nodes is ~38 % of the stream benchmark.

The large distance to the theoretical performance roof and much smaller distance to the bandwidth roof indicates that the real memory traffic is significantly higher than the theoretical value assumed in Section 3. Thus, the further performance analysis and optimisations should be conducted with respect to this finding.

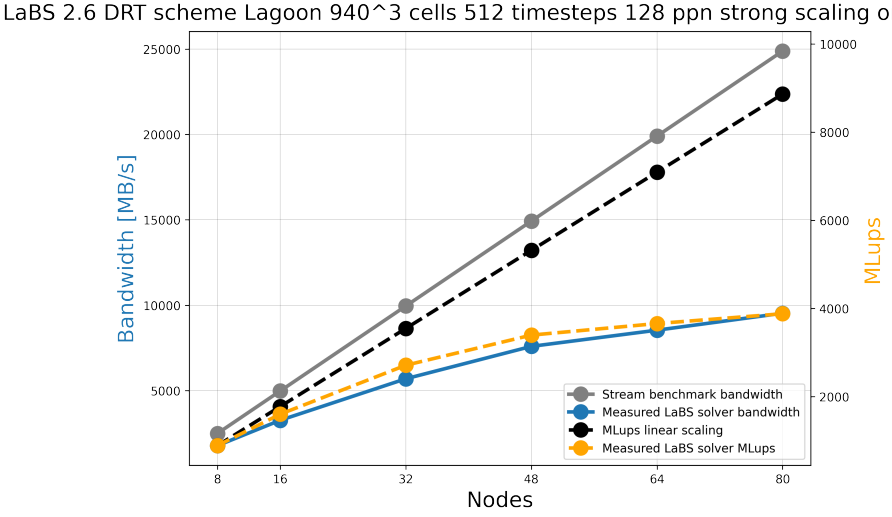


Figure 28: LABS/ProLB scaled up Lagoon DRAM bandwidth vs MLups on 8 - 80 nodes (128 processes per node).

To verify if the single-node memory bandwidth scaling presented in the beginning of the previous subsection holds even for the scaled up test case on much higher node count, we analysed the second set of measurements, this time with 64 processes per node, i.e. every other core of a node was idle. Please note that this leads to doubling the number of nodes while keeping the number of active cores identical to the first set.

Figure 29 illustrates that the scalability plateaus with more than 96 nodes. That is also the point where the scaling efficiency drops below 80 %.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

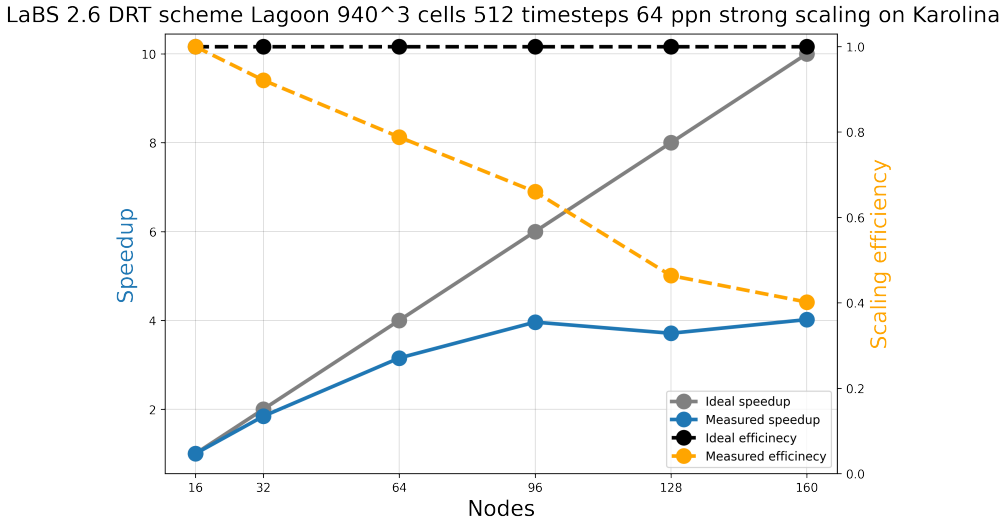


Figure 29: LABS/ProLB scaled up Lagoon scaling on 16 - 160 nodes (64 processes per node).

In Figure 30 one can clearly see a divergence between bandwidth and performance mainly on 64 and 96 nodes. As noted above, there is quite significant variance of measured solver execution times that affects the calculated performance on such a high number of nodes. On the other hand, the bandwidth values collection is performed on per process basis using HW counters, thus not so strongly affected by any load imbalance or communication synchronisation.

Again the maximum per node bandwidth 162 GB/s was measured on the smallest run, 16 nodes this time, corresponding to ~49 % of the stream benchmark value. The maximum total bandwidth 15 579 GB/s was reached on 160 nodes, which is ~30 % of the stream benchmark.

The maximum performance per node 82 MLups measured on 16 nodes is ~7 % of the stream-based theoretical peak value, while the maximum total performance 5294 MLups on 160 nodes is ~3 % of the referred peak.

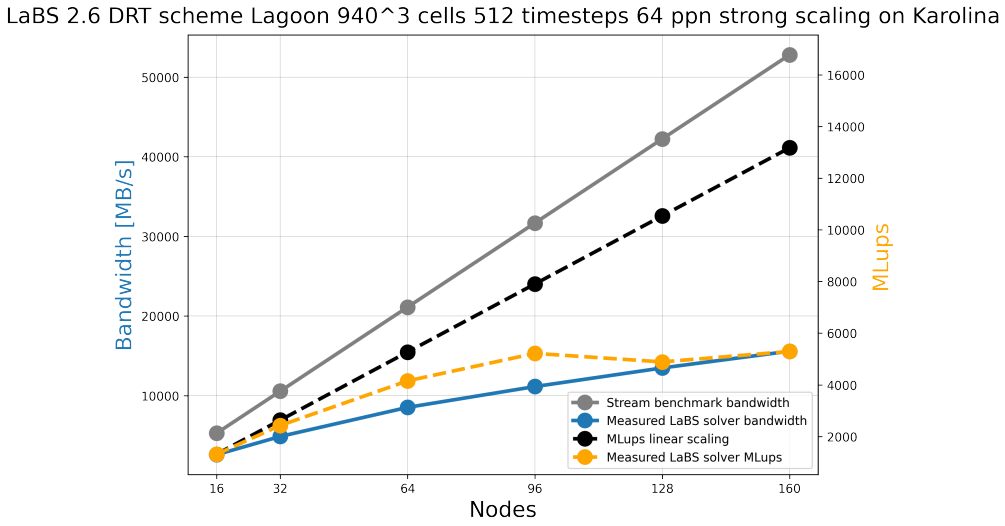


Figure 30: LABS/ProLB scaled up Lagoon DRAM bandwidth vs MLups on 16 - 160 nodes (64 processes per node).

We compare the solver execution times of both 128 and 64 processes per node measurement sets for the same number of nodes in Table 12. It shows that up to 32 nodes, it is overall faster to use 128 processes per node, while with higher node count it is worth to decrease the number of processes per node to 64. Such a comparison identifies the point where the number of processes, thus amount of communication, outweighs the benefit of extra cores for computation.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Table 12: Comparison scaled up Lagoon Strong Scaling Karolina CPU 128 ppn - 64 ppn.

Nodes	128 ppn solver time [s]	64 ppn solver time [s]
8	**165.4**	211.40
16	**91.7**	111.30
32	**54.1**	60.40
48	43.2	**42.80**
64	40.1	**35.30**
80	37.8	**30.70**
96	-	28.1
128	-	30
160	-	27.7

As we reached higher maximum total performance with 64 processes per node, we used that configuration also for the POP analysis of the scaled up Lagoon test case. Comparing with the original test case, we can observe higher IPC values in Table 13. Also the Frequency scalability degradation is much milder – the lowest average CPU frequency is 2.36 GHz on 160 nodes.

Nevertheless, the main limiting factors identified by POP metrics in Figure 31 are still the Load balance and Transfer efficiency as we observed in the original Lagoon test case. Revelation and proof of the root causes of these inefficiencies require deeper analysis and testing.

Table 13: Overview of the Speedup, IPC and Frequency of LaBS scaled up Lagoon 20 time steps on 16 - 160 nodes.

Number of processess	1024	2048	4096	6144	8192	10240
Number of nodes	16	32	64	96	128	160
Elapsed time [s]	0.62	0.32	0.18	0.17	0.15	0.13
Speedup	1.00	1.95	3.41	3.72	4.26	4.86
Scaling Efficiency	1.00	0.98	0.85	0.62	0.53	0.49
Average IPC	1.04	1.09	1.12	1.19	1.25	1.24
Average frequency [GHz]	3.23	3.17	2.99	2.81	2.54	2.36

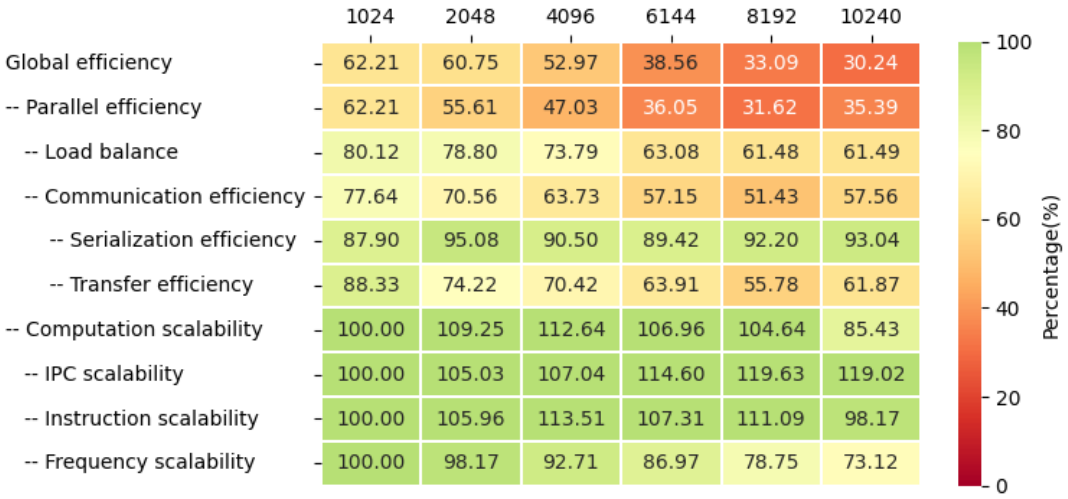


Figure 31: LABS/ProLB scaled up Lagoon Efficiency metrics of 20 time steps on 16 - 160 nodes (64 processes per node).

5 Conclusion

This deliverable contains the latest performance measurements and analysis of the LABS/ProLB and waLBerla codes on CPU and GPU whenever possible. Additionally, energy efficiency and tunning experiments have been performed on CPU and GPU yielding significant energy economies without compromising runtime

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

efficiency.
The next phase will include comparisons with the updated versions of the codes resulting from WP3 and WP5 developments.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

References

[1] M Bauer, H. Köstler, and U. Rüde. "lbmpy: Automatic code generation for efficient parallel lattice Boltzmann methods." In: *Journal of Computational Science* (2021). DOI: 10.1016/j.jocs.2020.101269.

[2] M. Bauer et al. "waLBerla: A block-structured high-performance framework for multiphysics simulations." In: *Computers & Mathematics with Applications* (2021). DOI: 10.1016/j.camwa.2020.01.007.

[3] R. Ge et al. *Power Measurement Tutorial for the Green500 List*. Tech. rep. June 2007. URL: https://www.top500.org/files/green500/tutorial.pdf.

[4] M. Geier and M. Schönherr. "Esoteric Twist: An Efficient in-Place Streaming Algorithmus for the Lattice Boltzmann Method on Massively Parallel Hardware." In: *Computation* 5.2 (2017). DOI: 10.3390/computation5020019.

[5] M. Geier et al. "The cumulant lattice Boltzmann equation in three dimensions: Theory and validation." In: *Computers & Mathematics with Applications* (2015). DOI: 10.1016/j.camwa.2015.05.001.

[6] Julian Hammer et al. "Automatic Loop Kernel Analysis and Performance Modeling with Kerncraft." In: *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*. PMBS '15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450340090. DOI: 10.1145/2832087.2832092. URL: https://doi.org/10.1145/2832087.2832092.

[7] M. Holzer et al. "Highly efficient lattice Boltzmann multiphase simulations of immiscible fluids at high-density ratios on CPUs and GPUs through code generation." In: *The International Journal of High Performance Computing Applications* (2021). DOI: 10.1177/10943420211016525.

[8] Jonas Latt, Christophe Coreixas, and Joël Beny. "Cross-platform programming model for many-core lattice Boltzmann simulations." In: *PLOS ONE* 16.4 (2021), pp. 1–29. DOI: 10.1371/journal.pone.0250306. URL: https://doi.org/10.1371/journal.pone.0250306.

[9] Moritz Lehmann. "Esoteric Pull and Esoteric Push: Two Simple In-Place Streaming Schemes for the Lattice Boltzmann Method on GPUs." In: *Computation* 10.6 (2022). ISSN: 2079-3197. DOI: 10.3390/computation10060092. URL: https://www.mdpi.com/2079-3197/10/6/92.

[10] Moritz Lehmann et al. "Accuracy and performance of the lattice Boltzmann method with 64-bit, 32-bit, and customized 16-bit number formats." In: *Phys. Rev. E* 106 (1 2022), p. 015308. DOI: 10.1103/PhysRevE.106.015308. URL: https://link.aps.org/doi/10.1103/PhysRevE.106.015308.

[11] LLNL. *msr-safe*. https://github.com/LLNL/msr-safe. 2021.

[12] G. Wellein et al. "On the single processor performance of simple lattice Boltzmann kernels." In: *Computers & Fluids* 35.8 (2006). Proceedings of the First International Conference for Mesoscopic Methods in Engineering and Science, pp. 910–919. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2005.02.008. URL: https://www.sciencedirect.com/science/article/pii/S0045793005001532.

[13] Samuel Williams, Andrew Waterman, and David Patterson. "Roofline: An Insightful Visual Performance Model for Multicore Architectures." In: *Commun. ACM* 52.4 (2009), pp. 65–76. ISSN: 0001-0782. DOI: 10.1145/1498765.1498785. URL: https://doi.org/10.1145/1498765.1498785.

[14] M. Wittmann et al. "Comparison of different propagation steps for lattice Boltzmann methods." In: *Computers & Mathematics with Applications* 65.6 (2013). Mesoscopic Methods in Engineering and Science, pp. 924–935. DOI: 10.1016/j.camwa.2012.05.002.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.