

## Article

# Energy-Efficient Implementation of the Lattice Boltzmann Method

Ondrej Vysocky <sup>1,\*</sup> , Markus Holzer <sup>2,3</sup> , Gabriel Staffelbach <sup>3</sup> , Radim Vavrik <sup>1</sup>  and Lubomir Riha <sup>1</sup> 

<sup>1</sup> IT4Innovations National Supercomputing Center, VŠB—Technical University of Ostrava, 708 00 Ostrava-Poruba, Czech Republic; radim.vavrik@vsb.cz (R.V.); lubomir.riha@vsb.cz (L.R.)

<sup>2</sup> Chair for System Simulation, Friedrich-Alexander-Universitat Erlangen-Nurnberg, 91058 Erlangen, Germany; markus.holzer@fau.de

<sup>3</sup> CERFACS, 31057 Toulouse Cedex 1, France; gabriel.staffelbach@cerfacs.fr

\* Correspondence: ondrej.vysocky@vsb.cz

**Abstract:** Energy costs are now one of the leading criteria when procuring new computing hardware. Until recently, developers and users focused only on pure performance in terms of time-to-solution. Recent advances in energy-aware runtime systems render the optimization of both runtime and energy-to-solution possible by including hardware tuning depending on the application's workload. This work presents the impact that energy-sensitive tuning strategies have on a state-of-the-art high-performance computing code based on the lattice Boltzmann approach called WALBERLA. We evaluate both CPU-only and GPU-accelerated supercomputers. This paper demonstrates that, with little user intervention, when using the energy-efficient runtime system called MERIC, it is possible to save a significant amount of energy while maintaining performance.

**Keywords:** HPC; GPU accelerators; DVFS; MERIC; energy-aware runtime system; dynamic resource management



**Citation:** Vysocky, O.; Holzer, M.; Staffelbach, G.; Vavrik, R.; Riha, L. Energy-Efficient Implementation of the Lattice Boltzmann Method. *Energies* **2024**, *17*, 502. <https://doi.org/10.3390/en17020502>

Academic Editor: Orlando Ayala

Received: 29 November 2023

Revised: 8 January 2024

Accepted: 16 January 2024

Published: 19 January 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

With the increasing challenges in developing faster hardware, the industry has shifted its focus from a pure performance perspective, as dictated by Moore's law, to the metric of performance per Watt. This paradigm shift, introduced by Intel in the mid-2000s with their first multi-core processors, prioritized energy efficiency and power consumption as pivotal factors in new chip design. Despite the adoption of performance per Watt metrics by various vendors, similar to the theoretical peak performance, these metrics are often based on undisclosed and nonstandardized benchmarks. Consequently, they do not accurately reflect the true power consumption of an application. Moreover, the diverse ways in which applications utilize standardized hardware make it essential to customize default processor settings to enhance performance per Watt on a per-application basis.

In this study, our focus lies in optimizing the energy consumption of the lattice Boltzmann method (LBM)-based massively parallel multiphysics framework WALBERLA [1]. WALBERLA stands as contemporary open-source C++ software designed to harness the full potential of large-scale supercomputers to address intricate research questions in the area of Computational Fluid Dynamics (CFD). The WALBERLA is one of EuroHPC Center of Excellence for Exascale CFD (CEEC) [2] applications. The framework development process prioritizes performance and efficiency, leading to strategic choices such as fully distributed data structures on an octree of blocks. Each data block contains information only about itself and its nearest neighbors, allowing efficient distribution across supercomputers through the Message Passing Interface (MPI) [1,3,4].

Optimizing hardware efficiency begins at the individual chip and core level, necessitating low-level architecture-specific optimizations, like vectorization with Single Instruction, Multiple Data (SIMD) instructions. Challenges escalate with code porting to accelerators, such as GPUs, demanding compatibility adjustments. WALBERLA addresses this complexity through meta-programming techniques within the *lbmpy* and *pystencils* Python

frameworks [5–8]. These techniques enable the formulation of algorithms in a symbolic form close to a mathematical representation. Subsequently, automated processes handle discretization and the generation of low-level C code, substantially elevating the level of abstraction and separation of concerns.

LBM-based applications are interesting to analyze for their dynamic behavior—in general, every solver iteration consists of two phases with different requirements on hardware resources. Calore et al. analyzed these kernels on various hardware architectures for possible energy savings [9,10] but using a very simple C code [11]. We build on their findings, especially the effective usage of an energy-efficient runtime system [12].

This paper provides an overview of the WALBERLA framework, elucidating its theoretical underpinnings and technologies. We integrate this understanding with performance tuning, pinpointing scenarios conducive to power efficiency gains while minimizing the impact on the time-to-solution for both CPU and GPU hardware configurations.

## 2. Lattice Boltzmann Method—Theoretical Background

The lattice Boltzmann method is a mesoscopic approach situated between macroscopic solutions of the Navier–Stokes equations (NSEs) and microscopic methods. Its origins can be traced back to an extension of lattice gas automata; however, more modernly, the theory is derived by discretizing the Boltzmann equation [13,14]. From this, the lattice Boltzmann equation (LBE) emerges that can be stated as:

$$f_i(\mathbf{x} + \mathbf{c}_i \Delta t, t + \Delta t) = f_i(\mathbf{x}, t) + \Omega_i(\mathbf{x}, t).$$

It describes the evolution of a local particle distribution function (PDF)  $f$  with  $q$ -entries stored in each lattice site. Typically, the grid is a  $d$ -dimensional Cartesian lattice with grid spacing  $\Delta x \in \mathbb{R}^+$ , giving the method its name. The PDF vector describes the probability of a virtual fluid particle in position  $\mathbf{x} \in \mathbb{R}^d$  and time,  $t \in \mathbb{R}^+$  traveling with discrete lattice velocity  $\mathbf{c}_i \in \Delta x / \Delta t \{-1, 0, 1\}^d$  [14]. Thus, instead of tracking individual real existing particles as microscopic approaches do, ensembles of virtual particles are simulated in the LBM approach.

The LBM can be separated in a streaming step, where PDFs are advected according to their velocities, and a collision step that rearranges the population cell locally. Thus, in the emerging algorithm, all nonlinear operations are cell local, while all nonlocal operations are linear. This gives the method its algorithmic simplicity and ease of the parallelization process. The collision operator  $\Omega_i(\mathbf{x}, t) \in \mathbb{R}$ , for redistribution of the PDFs, can be stated as

$$\Omega_i(\mathbf{x}, t) = \mathcal{T}^{-1}(\mathcal{T}(f) + S(f^{\text{eq}} - \mathcal{T}(f)))$$

where the PDFs are transformed to the collision space with a bijective mapping  $\mathcal{T}$  [8]. In the collision space, the collision is resolved by subtracting the equilibrium of the PDFs  $f^{\text{eq}}(\rho, \mathbf{u}) \in \mathbb{R}^q$  from the PDFs. Therefore, each entry in the emerging vector corresponds to different physical properties. Thus, to model distinct physical processes, different relaxation rates are applied to each quantity, which are stored in a diagonal relaxation matrix  $S$ . Typically, each relaxation rate  $\omega_i < 2/\Delta t$ , the inverse of which is referred to as the relaxation time  $\tau_i = 1/\omega_i$ . For example, to recover the correct kinematic viscosity  $\nu$  of a fluid, the relaxation time for the corresponding collision quantities can be obtained through

$$\nu = c_s^2 \left( \tau - \frac{\Delta t}{2} \right).$$

The basis for most LBM formulations is the Maxwell–Boltzmann distribution that defines the equilibrium state of the particles [14]

$$\Psi(\rho, \mathbf{u}, \mathbf{c}) = \rho \left( \frac{1}{2\pi c_s^2} \right)^{3/2} \exp \left( -\frac{\|\mathbf{c} - \mathbf{u}\|^2}{2c_s^2} \right),$$

where  $\rho \equiv \rho(x, t)$  and  $\mathbf{u} \equiv \mathbf{u}(x, t) \in \mathbb{R}^d$  describe the macroscopic density and velocity, respectively. Furthermore, the speed of sound  $c_s$  is defined as  $c_s = \sqrt{1/3} \Delta x / \Delta t$ .

### 3. Code Generation of LBM Kernels

Writing highly performant and flexible software is a severe challenge in many frameworks. On one side, the problem arises in describing the equations to solve in a way that is close to the mathematical description, while on the other side, the code needs to be specialized for different processing units, like SIMD, and accelerators, such as GPUs. In the massively parallel multiphysics framework, WALBERLA this is solved by employing meta-programming techniques. An overview of the approach is depicted in Figure 1. At the highest level, the Python package *lbmpy* encapsulates the complete symbolic representation of the lattice Boltzmann method. For this, the open-source library *SymPy* is used and extended by [15]. This workflow allows for the systematic dissection of the LBM into its constituent parts, subsequently modularizing and streamlining each step. However, modularization occurs directly on the mathematical level to form a final optimized update rule. A detailed description of this process can be found in [8]. Finally, this leads to highly specialized, problem-specific LBM compute kernels with minimal floating point operations (FLOPs), all while maintaining a remarkable degree of modularity within the source code.

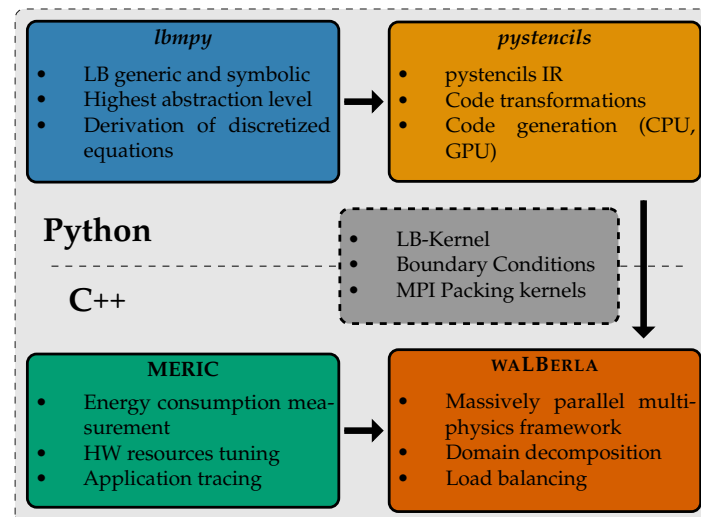


**Figure 1.** Overview of the software stack with *lbmpy*, *pystencils*, WALBERLA and software tuning. With the high-level Python packages *lbmpy* and *pystencils*, the numerical equations are derived, discretized, and, finally, lower-level C-Code is generated from this symbolic representation. The generated code can be combined with the C++ framework WALBERLA and compiled. The executable is optimized in terms of energy consumption.

From the symbolic description, an Abstract Syntax Tree (AST) is constructed within the *pystencils* Intermittent Representation (IR). This tree-based representation incorporates architecture-specific AST nodes and pointer access in subsequent kernels. Within this representation, spatial access particulars are encapsulated through *pystencils fields*. Additionally, constant expressions or fixed special model parameters can be directly evaluated to reduce the computational overhead. Given that the LBM compute kernel is symbolically defined, encompassing all field data accesses, the automated derivation of compute kernels naturally extends to encompass boundary conditions. This process also involves the creation of kernels for packing and unpacking. This suite of kernels plays a pivotal role in populating communication buffers for MPI operations.

Finally, the intermediate representation of the compute, boundary and packing/unpacking kernels is printed by the C or the CUDA backend of *pystencils* to a clearly defined interface. Each function takes raw pointers for array accesses together with their representative shape and stride information as well as all remaining free parameters. This simple and

consistent interface makes it possible to easily integrate the kernels in existing C/C++ software structures. Furthermore, with Python C-API, the low-level kernels can be mapped to Python functions, which enables interactive development by utilizing *lbmpy*/*pystencils* as stand-alone packages.

LBM is known for its high memory demand, and thus it was often shown that highly optimized compute kernels are only limited by the memory bandwidth of a processor or accelerator [6,7]. Thus, naturally, the question arises as to whether it is possible to reduce the energy consumption by reducing the frequency of the CPU compute units (CPU cores) while maintaining the full memory subsystem performance. Furthermore, the high level of optimization employed to *lbmpy* leads to especially low FLOP numbers in the hotspot of the code [8].

#### 4. Energy-Aware Hardware Tuning—Theoretical Background

Energy efficiency is commonly defined as the performance achieved per unit of power consumption, typically expressed as floating point operations per second per Watt. However, when dealing with codes based on the lattice Boltzmann method (LBM), performance is better characterized by the number of Lattice Updates executed Per Second (LUPS). In this study, we quantify the energy efficiency of the WALBERLA application as Millions of Lattice Updates per Second per Watt (MLUPS/W).

To accurately measure the energy consumption of an application, a high-frequency power monitoring system is imperative. This system should provide real-time power or energy consumption readings for the entire computational node or, at a minimum, its key computational components.

The total energy consumed (in Joules) can be calculated from power samples (in Watts) obtained at a specific sampling frequency (in Hertz) as depicted in the following equation:

$$Energy(t) = \int_0^t Power(x), dx \approx \frac{\sum_{i=0}^n PowerSample_i}{SamplingFrequency}.$$

There are two fundamental approaches to increase energy efficiency: (1) optimizing applications to fully exploit computational resources, ensuring that the workload aligns with the upper limits defined by the hardware's roofline model, or (2) judiciously limiting unused resources to prevent power wastage.

Modern high-performance CPUs and GPUs offer at least one tunable parameter controllable from the user space. Typically, it is the frequency of computation units (CPU cores) which directly impacts the peak performance of the chip, and it is crucial for compute-intensive computing tasks. These parameters can be adjusted either statically or dynamically.

Static tuning involves configuring specific hardware settings at the start of an application execution and maintaining this configuration until its completion. However, such static setups are rarely optimal for complex applications, leading to suboptimal energy savings. Static tuning lacks adaptability to workload changes during application execution, hindering the achievement of maximum available efficiencies.

In contrast, dynamic tuning adjusts parameters continuously during application execution. This functionality is achieved by energy-aware runtime systems that can identify optimal settings for different phases of the application and modify hardware configuration.

One such system is COUNTDOWN [16], maintained by CINECA and the University of Bologna. COUNTDOWN dynamically scales CPU core frequency during the MPI communication and synchronization phases, while ensuring that the application's performance is preserved.

The Barcelona Supercomputing Center develops EAR [17], a library that iteratively adjusts the CPU core frequency or powercap based on binary instructions and performance counters values.

LLNL Conductor [18] employs a power limit approach, identifying critical communication paths and allocating more power to slower processes to reduce waiting times, thus enhancing overall performance. Similarly, LLNL Uncore Power Scavenger [19] dynami-

cally tunes the Intel CPU configuration by sampling RAPL DRAM power consumption and instructions per cycle variation. Optimal energy savings are achieved with a 200 ms sampling interval.

Furthermore, the Runtime Exploitation for Application Dynamism for Energy-efficient eXascale computing (READEX) project [20] introduced a dynamic tuning methodology [21] and its implementation. The tools developed in this project provide HPC application developers with ways to exploit the dynamic behavior of their applications. The methodology is based on the assumption that each region of an application may require specific hardware configurations. The READEX approach identifies these requirements for each region and dynamically adjusts the hardware configuration when entering a region. MERIC [22], an implementation of the READEX approach developed at IT4Innovations, defines a minimum runtime of the region as 100 ms to ensure reliable energy measurements and to accommodate latency when changing hardware configurations.

These tools are able to bring significant energy savings without or with limited performance penalty. However, these tools are designed to work well on non-accelerated machines, and they do not tune GPU parameters. However, in modern accelerated HPC clusters, GPUs consume the majority of the compute node energy. The energy efficiency of data center GPUs is significantly higher when compared to that of server CPUs. This is confirmed by the fact that all the top-ranked supercomputers in Green500 [23] (the list of the most energy-efficient HPC systems) are based on Nvidia or AMD GPUs. Taking all this into account, it is still possible to improve the energy efficiency of these GPUs by around tens of percent as presented in [24].

A survey of GPU energy-efficiency analysis and optimization techniques [25] refers to various approaches to identify the optimal configuration of the GPU frequency to obtain energy savings, but in each listed case, the paper presents a single configuration for the whole execution of the application, which we refer to as static tuning. Similarly, Kraljic et al. [26] identified the execution phases of an analyzed application by sampling GPU energy consumption. Ghazanfar et al. [27] trained a neural network model to identify the optimal GPU SM frequency. However, in all cases above, the authors did not try to change the configuration dynamically during the execution of an application.

## 5. Results of the waLBerla Energy Consumption Optimization

For studying the energy consumption of waLBerla in an industrially relevant test case, the so-called LAGOON (Landing-Gear nOise database for CAA validation), which is simplified Airbus plane landing gear, was chosen [28,29]. The test setup consists of the LAGOON geometry (see Figure 2) in a virtual wind tunnel with a uniform resolution of  $512 \times 512 \times 512$  cells. For the inflow wall bounce back, boundary conditions with an inflow velocity of 0.05 in lattice units were used, while the outflow was modeled with non-reflecting outflow boundaries. For the purpose of the energy tuning, the benchmark case was simulated for 100 time steps.

Benchmarking and performance and energy measurements were performed at the following machines of the IT4Innovations supercomputing center: (1) Barbora non-accelerated partition, and (2) Karolina GPU accelerated partition.

The Barbora system is equipped with two Intel Xeon Gold 6240 CPUs (codename Cascade lake) per node. Each CPU has 18 cores (the hyper-threading is disabled) and is designed to work at 150 W TDP. The nominal frequency of the CPU is 2.6 GHz, but it can reach up to (i) 3.9 GHz of the turbo frequency when only two cores are active or (ii) 3.3 GHz of the turbo frequency when all cores are active and execute SSE instructions. The CPU core frequency can be reduced all the way to 1.1 GHz by a user or operating system. Since the Nehalem architecture, Intel has been using ‘uncore’ to refer to the frequency of the subsystems in the physical processor package that are shared by multiple processor cores e.g., last-level cache, on-chip ring interconnect or integrated memory controllers. Uncore regions overall occupy approximately 30% of a chip area [30]. While the core frequency is critical for compute-bound regions, the memory-bound regions are much more sensitive

to uncore frequency [31]. The Barbora CPUs can scale the uncore frequency between 1.2 and 2.4 GHz.

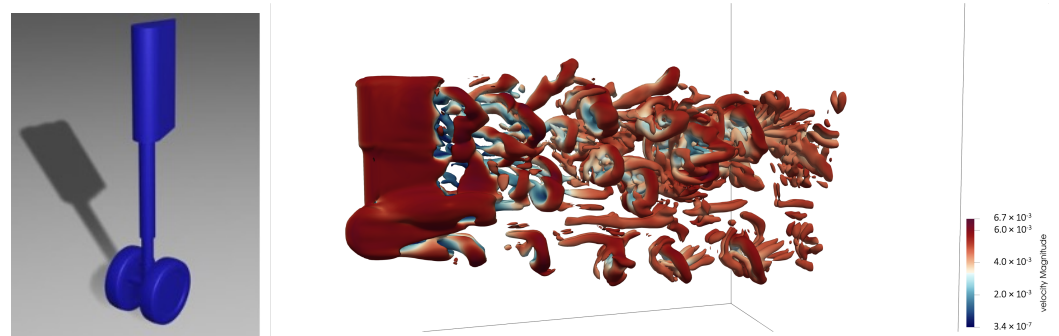


**Figure 2.** Geometry of the LAGOON plane landing gear (left). The geometry was studied in a virtual wind tunnel (right).

Barbora’s computational nodes are equipped by the on-board Atos | Bull High Definition Energy Efficient Monitoring (HDEEM) system [32], which reads power consumption from the mainboard hardware sensors and stores the data to dedicated memory. The sensor that monitors the consumption of the whole node provides 1000 power samples per second, and the rest of the sensors that monitor the compute node sub-units provide 100 samples per second. Both aggregated values and power samples can be read from the user space using a dedicated library or command-line utility. Since Sandy Bridge generation, Intel processors have integrated a Running Average Power Limit (RAPL) hardware power controller that provides a power measurement and mechanism to limit the power consumption of several domains of the CPU [33]. Intel RAPL controls the CPU core and uncore frequencies to keep the average power consumption of the CPU package below the TDP. The Intel RAPL interface allows a reduction in this power limit but not an increase.

Karolina cluster (rank 71. in Top500 11/2021, 8. in Green500 11/2021 [34]) nodes are equipped with two AMD EPYC 7763 CPUs, and eight Nvidia A100-SXM4 GPUs. One MPI process per GPU is used, four per CPU. To improve the energy efficiency of the application, we have specified the frequency of the GPU streaming multiprocessors (SMs), which is analogous to the CPU core frequency. For this purpose, we use the Nvidia Management Library (NVML), which provides the function `nvmlDeviceSetApplicationsClocks()` that sets a specific clock speed of a target GPU for both (i) memory and (ii) streaming multiprocessors. However, the A100-SXM4 uses HBM2 memory, whose frequency cannot be tuned as it is possible in the case of the GDDR memory. Therefore, on data-center-grade GPUs, like A100, only the frequency of SMs can be controlled.

The energy consumption of the GPU-accelerated application executions was measured using performance counters of the GPU (accessed using the Nvidia Management Library) and CPU (AMD RAPL with a similar power monitoring interface as Intel RAPL without the support for power capping).

To improve the energy efficiency of the GPU-accelerated executions of the WALBERLA, we performed the static tuning of the GPUs. The MERIC runtime system supports the dynamic tuning of GPU-accelerated applications based on CPU regions only if the GPU workload is synchronized. This limitation comes from the requirement that the GPU frequency cannot be controlled from a kernel. It is the CPU that creates the request to change the frequency through the GPU driver.

By default, when running a workload, the A100-SXM4 GPU (400 W TDP) uses the maximum turbo frequency of 1.410 GHz (if not forced to reduce the frequency by the power consumption exceeding the power limit or by thermal throttling) and switches to the nominal frequency of the GPU (1.095 GHz) when copying the data to/from the GPU memory. The frequency can be reduced to 210 MHz in 81 steps.

During the execution of the CUDA kernels on the GPUs, we also evaluated the impact of the CPU core frequency tuning. The nominal frequency of the AMD EPYC 7763 (280 W

TDP) is 2.45 GHz, while the CPU can run up to 3.525 GHz boost frequency. To reduce the number of tests, the 100 MHz step was used instead of the 25 MHz step, which is the highest resolution supported.

To control the hardware parameters mentioned above and to measure the resource consumption of the executed application, we used the MERIC runtime system. In the case of the non-accelerated version of WALBERLA, both static (single hardware configuration for the entire application execution) and dynamic tuning (a specific hardware configuration for each part of the application) were used. In the case of the GPU-accelerated version, static tuning was used only since the MERIC does not have support to identify which CUDA kernel is running on a GPU. A runtime system with support for dynamic GPU tuning is still a work in progress.

### 5.1. Static Tuning of the CPU Parameters

The WALBERLA energy efficiency analysis started with the static tuning of its non-accelerated version. We performed an exhaustive state-space search, testing all possible CPU core and uncore frequency configurations, using the 0.2 GHz step for both the CPU core and the uncore frequency. The lowest frequencies were omitted since one can expect high performance penalty in these configurations.

Table 1 (performance penalty), Table 2 (HDEEM energy savings) and Table 3 (Intel RAPL energy savings) show the consumption of resources of the WALBERLA solver in various configurations, using color coding to indicate which values are better (green) or worse (red).

From all the evaluated configurations, the highest energy savings based on the HDEEM measurements are 22.8%. These were reached for the following configuration: CF 1.9 GHz and UCF 1.8 GHz. For the same configuration, the savings calculated from the RAPL measurements are 29.6%. However, in this configuration, the performance drops by about 13.9%. The major difference in energy savings between HDEEM and RAPL comes from the set of power domains monitored by them. RAPL only monitors the power consumption of the CPU, which is the only component that brings energy savings due to tuning. The power consumption of the remaining node components remains unchanged, which has a major impact on energy savings if the runtime is extended. Since HDEEM monitors the power consumption of the entire node, its results are more representative.

Static tuning usually provides a limited possibility to obtain major energy savings without a performance penalty. WALBERLA reached 10.2% energy savings based on HDEEM measurements and 12.1% energy savings based on RAPL measurements at a cost of 1.6% performance degradation when the core frequency was reduced to 2.8 GHz and the uncore frequency remained without any limitations.

The performance impact of the evaluated CPU frequencies configurations on the WALBERLA solver is shown in Table 1. The respective energy savings are in Table 2 for the HDEEM measurements and in Table 3 for the Intel RAPL measurements. Based on the measurements in these tables, we identified configurations that cause up to 2, 5, 10 and unlimited runtime extension while bringing the maximum possible energy savings based on HDEEM measurements. The summary of the results is in Table 4, which presents the best configurations for various performance penalty limits. It also shows one hand-picked configuration which provides 19.6% energy savings based on HDEEM measurements while extending the runtime only about 6.9%.

**Table 1.** Impact of the static tuning on the overall runtime in [%] of WALBERLA when running the Lagoon use case.

| $\frac{\text{core [GHz]}}{\text{uncore [GHz]}}$ | 1.9    | 2.1    | 2.3    | 2.5    | 2.6   | 2.7    | 2.8    | 2.9    | 3.0    | 3.1    | 3.2    | 3.3    |
|-------------------------------------------------|--------|--------|--------|--------|-------|--------|--------|--------|--------|--------|--------|--------|
| 1.6                                             | −17.46 | −15.91 | −14.6  | −13.68 | −14.2 | −13.06 | −13.93 | −14.35 | −11.71 | −11.49 | −11.26 | −10.82 |
| 1.8                                             | −13.9  | −11.75 | −10.35 | −9.97  | −9.24 | −8.83  | −8.45  | −8.34  | −7.64  | −8.01  | −7.32  | −6.93  |
| 2                                               | −10.89 | −10.03 | −7.79  | −6.86  | −6.37 | −5.78  | −5.57  | −5.36  | −4.79  | −4.27  | −4.44  | −4.01  |
| 2.2                                             | −8.87  | −6.88  | −5.85  | −4.75  | −4    | −3.84  | −3.38  | −3     | −2.94  | −2.22  | −1.85  | −1.89  |
| 2.4                                             | −7.3   | −5.17  | −4.24  | −3.1   | −2.81 | −3.69  | −1.59  | −1.33  | −1.06  | −0.96  | −0.27  | −0.09  |

**Table 2.** Energy savings for different configurations using HDEEM measurements in [%] of WALBERLA when running the Lagoon use case.

| $\frac{\text{core [GHz]}}{\text{uncore [GHz]}}$ | 1.9   | 2.1   | 2.3   | 2.5   | 2.6   | 2.7   | 2.8   | 2.9   | 3.0   | 3.1  | 3.2  | 3.3   |
|-------------------------------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|------|------|-------|
| 1.6                                             | 22.61 | 21.39 | 19.78 | 16.63 | 15    | 14.29 | 11.81 | 9.22  | 9.53  | 7.07 | 4.86 | 2.45  |
| 1.8                                             | 22.84 | 22.28 | 20.84 | 17.3  | 16.63 | 15.36 | 14.18 | 12.18 | 10.59 | 7.68 | 6.02 | 3.56  |
| 2                                               | 21.95 | 20.51 | 19.62 | 16.64 | 15.79 | 14.75 | 13.35 | 11.59 | 9.92  | 8.14 | 5.71 | 3.21  |
| 2.2                                             | 20.19 | 19.61 | 17.74 | 15.15 | 14.47 | 13.14 | 11.99 | 10.49 | 8.14  | 6.54 | 4.72 | 1.82  |
| 2.4                                             | 17.69 | 17.26 | 15.54 | 12.89 | 11.81 | 9.56  | 10.17 | 8.33  | 6.36  | 4.18 | 2.68 | −0.27 |

**Table 3.** Energy savings for different configurations using Intel RAPL measurements in [%] of WALBERLA when running the Lagoon use case.

| $\frac{\text{core [GHz]}}{\text{uncore [GHz]}}$ | 1.9   | 2.1   | 2.3   | 2.5   | 2.6   | 2.7   | 2.8   | 2.9   | 3.0   | 3.1   | 3.2  | 3.3   |
|-------------------------------------------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|------|-------|
| 1.6                                             | 31.25 | 28.8  | 26.42 | 22.46 | 20.46 | 19.34 | 16.38 | 13.61 | 13.62 | 10.44 | 7.81 | 4.63  |
| 1.8                                             | 29.63 | 29.2  | 26.93 | 22.23 | 21.68 | 19.95 | 18.42 | 16.05 | 13.77 | 10.5  | 8.08 | 5.16  |
| 2                                               | 28.04 | 26.1  | 24.99 | 21    | 19.96 | 18.57 | 16.67 | 14.28 | 12.38 | 10.06 | 7.37 | 4.34  |
| 2.2                                             | 25.4  | 24.6  | 21.84 | 18.77 | 17.82 | 16.08 | 14.58 | 12.92 | 9.88  | 7.81  | 5.84 | 2.36  |
| 2.4                                             | 22.35 | 21.28 | 19.48 | 15.61 | 14.42 | 11.66 | 12.06 | 9.88  | 7.69  | 4.97  | 3.33 | −0.32 |

**Table 4.** Results of the static tuning of the WALBERLA solver when running the Lagoon use case. The table presents energy savings from both the HDEEM and Intel RAPL energy measurements for various performance degradation trade-offs. In addition, one hand-picked configuration that brings meaningful energy savings with modest performance penalty is also presented.

|               | −2%<br>Limit | −5%<br>Limit | −10%<br>Limit | No<br>Limit | Selected<br>Configuration |
|---------------|--------------|--------------|---------------|-------------|---------------------------|
| Runtime [%]   | −1.6         | −4.8         | −8.9          | −13.9       | −6.9                      |
| HDEEM [%]     | 10.2         | 15.2         | 20.2          | 22.8        | 19.6                      |
| RAPL [%]      | 12.1         | 18.8         | 25.4          | 29.6        | 24.6                      |
| CF; UCF [GHz] | 2.8; 2.4     | 2.5; 2.2     | 1.9; 2.2      | 1.9; 1.8    | 2.1, 2.2                  |

Finally, Figure 3 shows the power consumption timeline of the entire compute node (blade) and selected node components. Samples were collected by HDEEM during the execution of the Lagoon test case. One can see dynamic changes in power consumption, which indicates that a static hardware configuration is not optimal for the whole application run because the hardware requirements change over time.

### 5.2. Dynamic Tuning of the CPU Parameters

This section presents a WALBERLA analysis using dynamic tuning, which sets hardware configurations that best suit each instrumented section of the code. MERIC supports automatic binary instrumentation, which generates a copy of the application executable binary file that includes the MERIC API calls at the beginning and the end of all selected regions. Due to the exception of usage in the WALBERLA code, it was not possible to use fully automatic binary instrumentation because the execution then resulted in a runtime error of uncaught exceptions. We manually instrumented the WALBERLA source code with the MERIC function calls, which resulted in less fine-grain instrumentation than would be possible with full binary instrumentations. Despite the fact that the instrumentation consists of only eight regions, which is not optimal, we were able to cover 99% of the application runtime.

In the case of the identification of an optimal dynamic configuration, we also executed the application in various hardware configurations, while for each instrumented region, we identified its optimal configuration. The state-space search was performed twice—one to obtain a configuration that does not cause any performance degradation, and another one to bring maximum HDEEM energy savings without any runtime extension limitation.

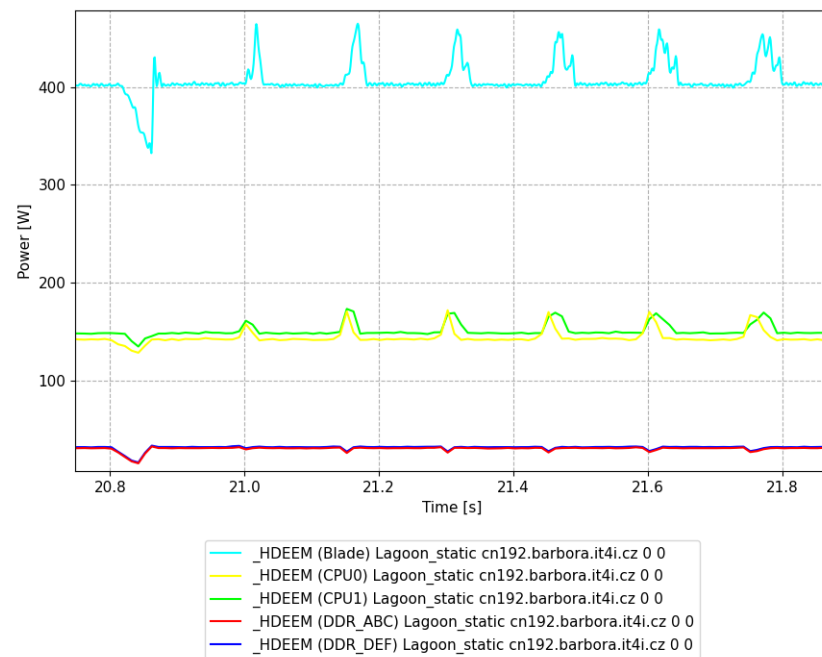


**Figure 3.** Power timeline based on HDEEM measurements of WALBERLA when running the Lagoon use case. The data are collected for entire compute node (blade), and its components (two CPUs and two groups of memory channels). The time windows shows one synchronization phase followed by the six solver iterations.

Table 5 compares four different executions of WALBERLA, the default hardware configuration, the compromise static configuration with 2.1 GHz core and 1.8 GHz uncore frequency, and two executions using dynamic tuning with and without performance penalty. The WALBERLA execution Lagoon configuration has changed to show that these parameters have a major impact on static execution optimal configuration and what savings it brings. In contrast to the previous section, here, we present values for the whole application runtime because the dynamic tuning optimized the whole runtime. In this case, the solver takes 3/4 of the runtime. While in the previous Section 5.1, we present a runtime extension of about 12.6% in the compromise static configuration, now the same configuration extends the runtime by just about 5.8%. The problem that each execution configuration may result in a different optimal static configuration is solved by using dynamic tuning since each region of the application may take a different time; however, the regions' hardware requirements are the same, and thus the optimal configuration is the same.

Table 5 shows that dynamic tuning can achieve higher energy savings than static tuning. The dynamically tuned execution of WALBERLA consumed 7.9% less energy without extending the runtime. The highest energy savings achieved with dynamic tuning is 19.1% at a cost of extending the runtime by 16.2%.

Please note that in this case, the solver consists of a single region only. In Figure 3, it is visible that the solver should be split into at least two different regions because these regions have different hardware requirements. However, their runtime is very short (up to 5 ms), while MERIC requires regions of at least 100 ms. Investigating possible energy savings gained from the dynamic tuning of the solver is our goal when a new release of the MERIC that brings better support for fine-grain tuning becomes available.

**Table 5.** Comparison of static and dynamic tuning results of WALBERLA when running the Lagoon use case. The table presents their impact on runtime, energy consumption and performance per Watt. Values are evaluated from the whole application execution.

|                                    | Default | Static Tuning | Dynamic Tuning | Dynamic Const. Time |
|------------------------------------|---------|---------------|----------------|---------------------|
| Runtime [s]                        | 66.86   | 70.75         | 77.69          | 66.64               |
| HDEEM [kJ]                         | 99.04   | 90.88         | 80.17          | 91.24               |
| Solver energy efficiency [MLups/W] | 0.548   | 0.667         | 0.734          | 0.595               |
| Runtime extension [%]              | -       | 5.8           | 16.2           | −0.4                |
| HDEEM savings [%]                  | -       | 9.2           | 19.1           | 7.9                 |

### 5.3. Static Tuning of the GPU Parameters

In this last experiment, we executed the WALBERLA Lagoon use case on a GPU accelerated node, with 8x Nvidia A100-SXM4 GPUs, while all GPUs were set to a specific frequency at the beginning of the application execution. For static tuning, it is not necessary to use a runtime system to control the hardware configuration. In the simplest case, Nvidia utility `nvidia-smi` integrated into the CUDA toolkit can be used. However, as well as in the case of static CPU tuning, the MERIC provides additional resource consumption measurement.

Figure 4 shows the application runtime and energy consumption of the GPUs measured using the NVML interface together with the CPUs energy consumption measured using AMD RAPL (Package power domain) for various streaming multiprocessor (SM) frequency settings. The power consumption of the remaining active components of the compute node or server (mainboard, cooling fans, NICs, etc.) is not measured during the application execution. The server does not provide any high-frequency power monitoring system. However, the total server power consumption can be measured using the HPE Redfish implementation called Integrated Lights Out (iLO) [35] once every ten seconds.

For this GPU-accelerated server, iLO reports on average extra 600 W in addition to the power consumption reported by AMD RAPL and NVML. This implies that the runtime extension resulting from underclocking the GPUs might be very harmful to the overall server energy consumption despite the fact that the performance counters (RAPL and NVML) report energy savings.

Figure 5 shows the energy efficiency for various GPU frequencies, where the energy consumption is evaluated as a sum of AMD RAPL measurements, NVML measurements, and additional 600 W to account for the remaining on-node components.

Based on the measurements presented, we identified that the optimal configuration of the A100 SM frequency is 1.005 GHz. Table 6 compares the resource consumption for the default and for the optimal frequency, in which the application runtime is extended by about 2.2% only. For these settings, the energy efficiency expressed in MLUPs/W is improved by 22.7% for RAPL + NVML measurements only (19.8% energy savings), respectively, and 15% if the 600 W static power consumption of the node is included (9.3% energy savings).

For the optimal SM frequency, we also evaluated the impact of the CPU core frequency scaling. We expected that the maximum boost frequency is not necessary to reach the maximum performance that GPUs deliver. The AMD EPYC 7763 nominal frequency is 2.45 GHz, while the CPU can run up to the 3.525 GHz boost frequency. The CPU power consumption when idle is approximately 90 W. During WALBERLA execution, while the CPU only controls the GPU execution, the power consumption increases by 10 W only. This gives very little space for power savings.

Contrary to Intel processors in Barbora, the CPU core frequency of AMD EPYC CPUs in Karolina cannot be locked, but it can be limited from the top. Figure 6 shows the runtime and energy consumption (NVML + RAPL only) for various CPU core frequency limits for GPU-accelerated executions of WALBERLA. Scaling the frequency in the range of turbo

frequencies does not have any significant impact on the application runtime; however, it also does not bring any energy savings. Scaling below the nominal frequency has a negative impact on both the runtime and energy consumption.

**Table 6.** Accelerated execution of WALBERLA performance in the default configuration on the hardware compared to static configuration 1.005 GHz SM frequency of A100 GPUs.

|                             | Default | Static Tuning |
|-----------------------------|---------|---------------|
| Runtime [s]                 | 60.3    | 61.1          |
| Runtime extension [%]       | -       | 2.2           |
| NVML + RAPL                 |         |               |
| energy [kJ]                 | 95.0    | 75.2          |
| energy efficiency [MLups/W] | 15.4    | 19.4          |
| energy savings [%]          | -       | 19.8          |
| NVML + RAPL + 600 W         |         |               |
| energy [kJ]                 | 125.2   | 113.5         |
| energy efficiency [MLups/W] | 11.2    | 13.1          |
| energy savings [%]          | -       | 9.3           |

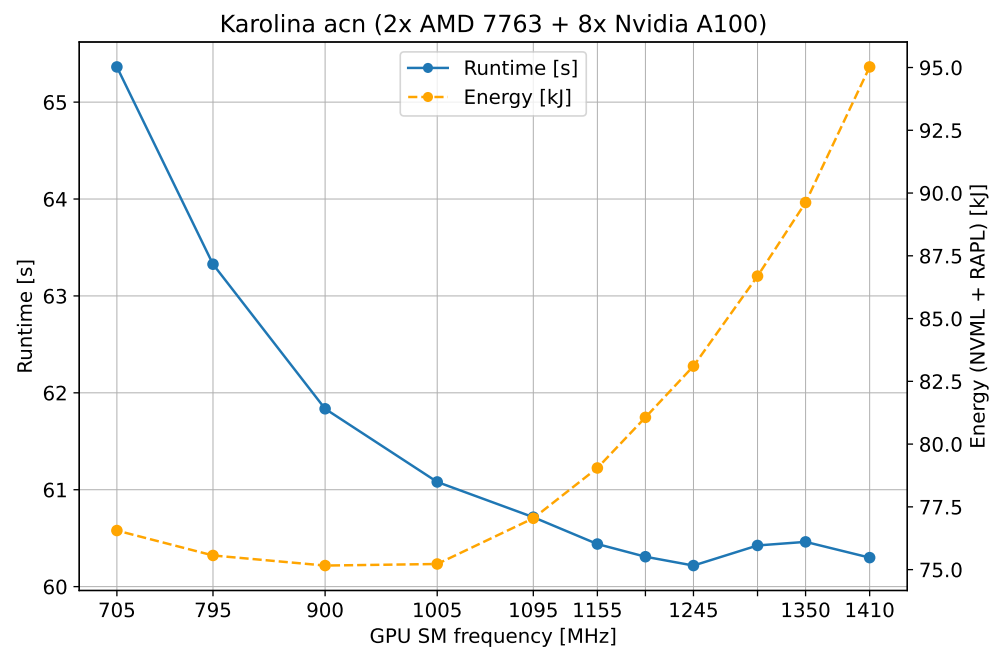


**Figure 4.** Energy consumption and runtime of WALBERLA when running the Lagoon use case on GPU accelerated node. The results are for different GPUs streaming multiprocessor frequencies.

For the purpose of this analysis, it seems that only the GPU frequency matters when looking at improving the energy efficiency. This might be different for other applications that can fully utilize both CPU and GPU resources.

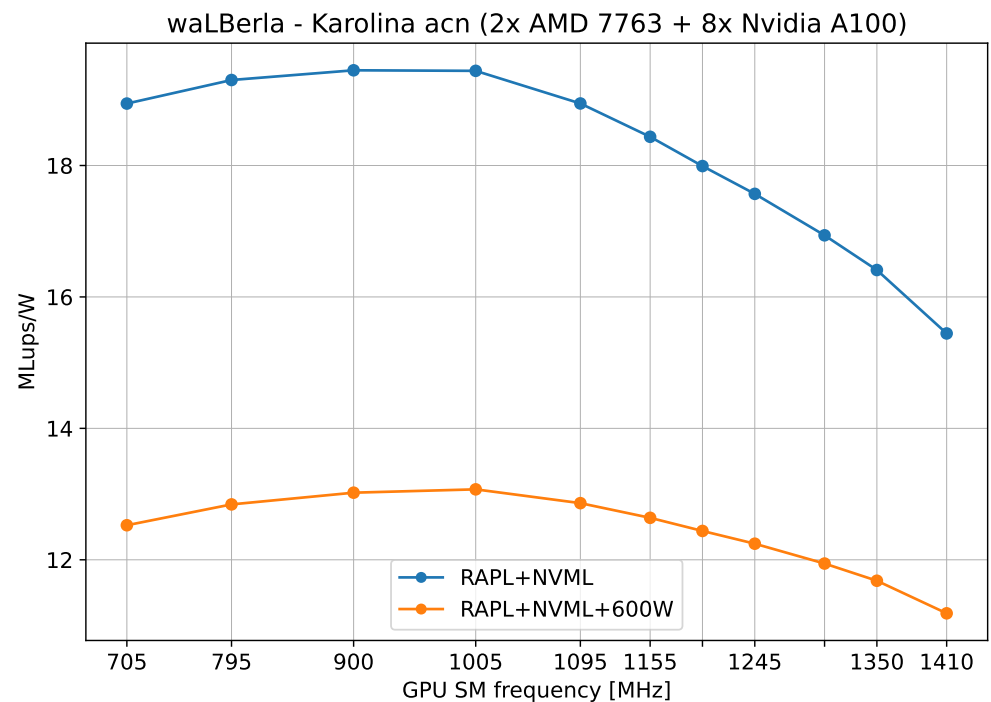


**Figure 5.** Energy efficiency of the GPU-accelerated version of WALBERLA for different GPU streaming multiprocessor frequencies. RAPL + NVML reports the power consumption of the CPUs and GPUs only, while RAPL + NVML + 600 W shows the power consumption of the entire server.

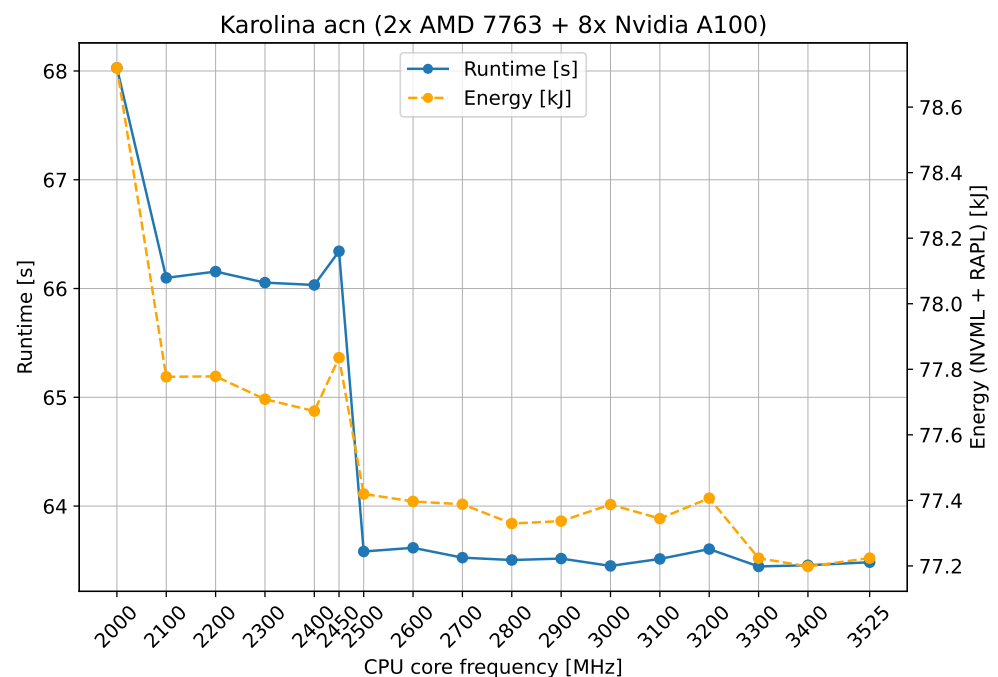


**Figure 6.** Runtime and energy consumption of the WALBERLA using the optimal (1.005 GHz) GPU streaming multiprocessor frequency during the CPU core frequency scaling.

## 6. Conclusions

This study showcases improvements in energy efficiency achieved through a detailed understanding of the intricacies of the application. Notable reductions, up to 20%, in energy consumption were demonstrated for both the CPU-only cluster as well as accelerated machines with Nvidia A100 GPUs. Impressively, these gains were achieved with minimal

user intervention. However, a challenge lies in the current limitations associated with user permissions, restricting the ability to optimize processor or GPU behaviors in most HPC centers. Despite this hurdle, given the observed advantages, we are optimistic that HPC centers will implement solutions that benefit both users and hosting entities.

In particular, the accelerated version of WALBERLA showed significantly higher energy efficiency than the CPU version of the code, even in the default hardware configuration. This notable high efficiency stems from the inherent energy efficiency of Nvidia's top HPC GPUs. While our comparison faced slight discrepancies due to the usage of different power monitoring systems, we compensated for them by incorporating modeled power consumption data for the remaining on-node components of the GPU accelerated node.

The presented measurements align closely with expected figures for such advanced hardware, indicating the highly effective utilization of GPUs by the accelerated implementation of WALBERLA. The reduced GPU power consumption is caused by the fact that WALBERLA is memory-bound code, and the streaming multiprocessor is constantly waiting for data from the GPU memory. By underclocking the SM frequency, the power consumption is reduced, and at the same time the performance is not impacted.

This research not only signifies a path to energy-efficient computing but also underlines the potential for even greater strides in the future. As technologies continue to evolve and collaborative efforts drive innovation, the landscape of high-performance computing stands poised for a transformative era of unparalleled energy efficiency and computational power.

**Author Contributions:** Conceptualization, O.V., M.H. and G.S.; methodology, O.V.; software, O.V. and M.H.; validation, R.V. and L.R.; investigation, O.V. and R.V.; writing—original draft preparation, O.V., M.H. and G.S.; writing—review and editing, R.V. and L.R.; supervision, L.R. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the SCALABLE project. This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 956000. The JU receives support from the European Union's Horizon 2020 research and innovation program and France, Germany, and the Czech Republic. This project has received funding from the Ministry of Education, Youth and Sports of the Czech Republic (ID: MC2103). This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID: 90254).

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results. Authors Markus Holzer and Gabriel Staffelbach were employed by the company CERFACS. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## References

1. Bauer, M.; Eibl, S.; Godenschwager, C.; Kohl, N.; Kuron, M.; Rettinger, C.; Schornbaum, F.; Schwarzmeier, C.; Thönnies, D.; Köstler, H.; et al. waLberla: A block-structured high-performance framework for multiphysics simulations. *Comput. Math. Appl.* **2021**, *81*, 478–501. [\[CrossRef\]](#)
2. EuroHPC. Center of Excellence CEEC. 2021. Available online: <https://ceec-coe.eu/> (accessed on 2 January 2024).
3. Bauer, M.; Schornbaum, F.; Godenschwager, C.; Markl, M.; Anderl, D.; Köstler, H.; Rüde, U. A Python extension for the massively parallel multiphysics simulation framework waLberla. *Int. J. Parallel Emergent Distrib. Syst.* **2015**, *31*, 529–542. [\[CrossRef\]](#)
4. Godenschwager, C.; Schornbaum, F.; Bauer, M.; Köstler, H.; Rüde, U. A Framework for Hybrid Parallel Flow Simulations with a Trillion Cells in Complex Geometries. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13), New York, NY, USA, 17–21 June 2013. [\[CrossRef\]](#)
5. Bauer, M.; Hötzer, J.; Ernst, D.; Hammer, J.; Seiz, M.; Hierl, H.; Hönig, J.; Köstler, H.; Wellein, G.; Nestler, B.; et al. Code generation for massively parallel phase-field simulations. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, CO, USA, 17–19 November 2019; ACM: New York, NY, USA, 2019. [\[CrossRef\]](#)
6. Bauer, M.; Köstler, H.; Rüde, U. lbmpy: Automatic code generation for efficient parallel lattice Boltzmann methods. *J. Comput. Sci.* **2021**, *49*, 101269. [\[CrossRef\]](#)

7. Holzer, M.; Bauer, M.; Köstler, H.; Rude, U. Highly efficient lattice Boltzmann multiphase simulations of immiscible fluids at high-density ratios on CPUs and GPUs through code generation. *Int. J. High Perform. Comput. Appl.* **2021**, *35*, 413–427. [\[CrossRef\]](#)
8. Hennig, F.; Holzer, M.; Rude, U. Advanced Automatic Code Generation for Multiple Relaxation-Time Lattice Boltzmann Methods. *SIAM J. Sci. Comput.* **2023**, *45*, C233–C254. [\[CrossRef\]](#)
9. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccion, R. Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications. *Concurr. Comput. Pract. Exp.* **2017**, *29*, e4143. [\[CrossRef\]](#)
10. Giroto, I.; Schifano, S.F.; Calore, E.; Di Staso, G.; Toschi, F. Performance and Energy Assessment of a Lattice Boltzmann Method Based Application on the Skylake Processor. *Computation* **2020**, *8*, 44. [\[CrossRef\]](#)
11. Mantovani, F.; Pivanti, M.; Schifano, S.; Tripiccion, R. Performance issues on many-core processors: A D2Q37 Lattice Boltzmann scheme as a test-case. *Comput. Fluids* **2013**, *88*, 743–752. [\[CrossRef\]](#)
12. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccion, R. Energy-Efficiency Tuning of a Lattice Boltzmann Simulation Using MERIC. In Proceedings of the International Conference on Parallel Processing and Applied Mathematics, Bialystok, Poland, 8–11 September 2019; Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 169–180.
13. Chen, H.; Chen, S.; Matthaeus, W.H. Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method. *Phys. Rev. A* **1992**, *45*, R5339–R5342. [\[CrossRef\]](#)
14. Krüger, T.; Kusumaatmaja, H.; Kuzmin, A.; Shardt, O.; Silva, G.; Viggen, E.M. *The Lattice Boltzmann Method*; Springer International Publishing: Cham, Switzerland, 2017. [\[CrossRef\]](#)
15. Meurer, A.; Smith, C.P.; Paprocki, M.; Čertík, O.; Kirpichev, S.B.; Rocklin, M.; Kumar, A.; Ivanov, S.; Moore, J.K.; Singh, S.; et al. SymPy: Symbolic computing in Python. *PeerJ Comput. Sci.* **2017**, *3*, e103. [\[CrossRef\]](#)
16. Cesarini, D.; Bartolini, A.; Bonfa, P.; Cavazzoni, C.; Benini, L. COUNTDOWN: A Run-time Library for Performance-Neutral Energy Saving in MPI Applications. *IEEE Trans. Comput.* **2020**, *1*, 1–14. [\[CrossRef\]](#)
17. Corbalan, J.; Alonso, L.; Aneas, J.; Brochard, L. Energy Optimization and Analysis with EAR. In Proceedings of the 2020 IEEE International Conference on Cluster Computing (CLUSTER), Kobe, Japan, 14–17 September 2020; Volume 2020, pp. 464–472.
18. Marathe, A.; Bailey, P.E.; Lowenthal, D.K.; Rountree, B.; Schulz, M.; de Supinski, B.R. A Run-Time System for Power-Constrained HPC Applications. In Proceedings of the High Performance Computing, Frankfurt, Germany, 12–16 July 2015; Kunkel, J.M., Ludwig, T., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 394–408.
19. Gholkar, N.; Mueller, F.; Rountree, B. Uncore Power Scavenger: A Runtime for Uncore Power Conservation on HPC Systems. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, New York, NY, USA, 17–19 November 2019; SC '19. [\[CrossRef\]](#)
20. READEX. Horizon 2020 READEX Project. 2018. Available online: <https://www.readex.eu> (accessed on 15 January 2024).
21. Schuchart, J.; Gerndt, M.; Kjeldsberg, P.G.; Lysaght, M.; Horák, D.; Říha, L.; Gocht, A.; Sourouri, M.; Kumaraswamy, M.; Chowdhury, A.; et al. The READEX formalism for automatic tuning for energy efficiency. *Computing* **2017**, *99*, 727–745. [\[CrossRef\]](#)
22. Vysocky, O.; Beseda, M.; Riha, L.; Zapletal, J.; Lysaght, M.; Kannan, V. MERIC and RADAR Generator: Tools for Energy Evaluation and Runtime Tuning of HPC Applications. In Proceedings of the High Performance Computing in Science and Engineering, Karolinka, Czech Republic, 22–25 May 2017; Kozubek, T., Cermak, M., Tichy, P., Blaheta, R., Sistek, J., Lukas, D., Jaros, J., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 144–159.
23. Ge, R.; Feng, X.; Pyla, H.; Cameron, K.; Feng, W. Power Measurement Tutorial for the Green500 List; Technical Report; 2007. Available online: <https://www.top500.org/files/green500/tutorial.pdf> (accessed on 15 January 2024).
24. Spetko, M.; Vysocky, O.; Janský, B.; Riha, L. DGX-A100 Face to Face DGX-2—Performance, Power and Thermal Behavior Evaluation. *Energies* **2021**, *14*, 376. [\[CrossRef\]](#)
25. Mittal, S.; Vetter, J.S. A Survey of Methods For Analyzing and Improving GPU Energy Efficiency. *arXiv* **2014**, arXiv:1404.4629.
26. Kraljic, K.; Kerger, D.; Schulz, M. Energy Efficient Frequency Scaling on GPUs in Heterogeneous HPC Systems. In Proceedings of the Architecture of Computing Systems, Heilbronn, Germany, 13–15 September 2022; Schulz, M., Trinitis, C., Papadopoulou, N., Pionteck, T., Eds.; Springer International Publishing: Cham, Switzerland, 2022; pp. 3–16.
27. Ali, G.; Side, M.; Bhalachandra, S.; Wright, N.J.; Chen, Y. Performance-Aware Energy-Efficient GPU Frequency Selection Using DNN-Based Models. In Proceedings of the 52nd International Conference on Parallel Processing (ICPP '23), New York, NY, USA, 7–10 August 2023; pp. 433–442. [\[CrossRef\]](#)
28. Manoha, E.; Bulté, J.; Caruelle, B. Lagoon: An Experimental Database for the Validation of CFD/CAA Methods for Landing Gear Noise Prediction. In Proceedings of the 14th AIAA/CEAS Aeroacoustics Conference (29th AIAA Aeroacoustics Conference), Vancouver, BC, Canada, 5–7 May 2008. [\[CrossRef\]](#)
29. Manoha, E.; Bulte, J.; Ciobaca, V.; Caruelle, B. LAGOON: Further Analysis of Aerodynamic Experiments and Early Aeroacoustics Results. In Proceedings of the 15th AIAA/CEAS Aeroacoustics Conference (30th AIAA Aeroacoustics Conference), Miami, FL, USA, 11–13 May 2009. [\[CrossRef\]](#)
30. Hill, D.L.; Bachand, D.; Bilgin, S.; Greiner, R.; Hammarlund, P.; Huff, T.; Kulick, S.; Safranek, R. The uncore: A modular approach to feeding the high-performance cores. *Intel Technol. J.* **2010**, *14*, 30–49.
31. Hackenberg, D.; Schöne, R.; Ilsche, T.; Molka, D.; Schuchart, J.; Geyer, R. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW'15), Hyderabad, India, 25–29 May 2015; pp. 896–904. [\[CrossRef\]](#)

32. Hackenberg, D.; Ilsche, T.; Schuchart, J.; Schöne, R.; Nagel, W.; Simon, M.; Georgiou, Y. HDEEM: High Definition Energy Efficiency Monitoring. In Proceedings of the Energy Efficient Supercomputing Workshop (E2SC), New Orleans, LA, USA, 16–21 November 2014; pp. 1–10. [[CrossRef](#)]
33. Gough, C.; Steiner, I.; Saunders, W. CPU Power Management. In *Energy Efficient Servers: Blueprints for Data Center Optimization*; Apress: Berkeley, CA, USA, 2015; pp. 21–70. [[CrossRef](#)]
34. TOP500. TOP 500 Supercomputing Sites. Available online: <https://www.top500.org/> (accessed on 11 April 2023).
35. HPE. *HPE Integrated Lights-Out (iLO)*; Technical Report; HPE: Hong Kong, 2023.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.