Journal of
Low Power Electronics and Applications

MDPI

Article

Software and DVFS Tuning for Performance and Energy-Efficiency on Intel KNL Processors

Enrico Calore [1,*], Alessandro Gabbana [1,2], Sebastiano Fabio Schifano [1] and Raffaele Tripiccione [1]

[1] Università degli Studi di Ferrara and INFN, 44122 Ferrara, Italy; alessandro.gabbana@unife.it (A.G.); schifano@fe.infn.it (S.F.S.); tripiccione@fe.infn.it (R.T.)
[2] Bergische Universität Wuppertal, 42119 Wuppertal, Germany
* Correspondence: enrico.calore@unife.it; Tel.: +39-0532-974612

Received: 29 March 2018; Accepted: 1 June 2018; Published: 11 June 2018

Abstract: Energy consumption of processors and memories is quickly becoming a limiting factor in the deployment of large computing systems. For this reason, it is important to understand the energy performance of these processors and to study strategies allowing their use in the most efficient way. In this work, we focus on the computing and energy performance of the Knights Landing Xeon Phi, the latest Intel many-core architecture processor for HPC applications. We consider the 64-core Xeon Phi 7230 and profile its performance and energy efficiency using both its on-chip MCDRAM and the off-chip DDR4 memory as the main storage for application data. As a benchmark application, we use a lattice Boltzmann code heavily optimized for this architecture and implemented using several different arrangements of the application data in memory (data-layouts, in short). We also assess the dependence of energy consumption on data-layouts, memory configurations (DDR4 or MCDRAM) and the number of threads per core. We finally consider possible trade-offs between computing performance and energy efficiency, tuning the clock frequency of the processor using the Dynamic Voltage and Frequency Scaling (DVFS) technique.

Keywords: energy; KNL; MCDRAM; memory; lattice Boltzmann; HPC; DVFS

1. Introduction

Energy consumption is quickly becoming one of the most critical issues in modern HPC systems. Correspondingly, much interest is now focused on attempts to increase energy efficiency using a variety of different hardware- and software-based approaches. A further driver of research towards energy efficiency is also given by the need to lower the total cost of ownership of HPC installations, increasingly depending on the electricity bill. Processors and accelerators are the main sources of power drain in modern computing systems [1]; for this reason, assessing their energy efficiency is of paramount importance for the design and deployment of large energy-efficient parallel systems. Recent hardware developments show a definite trend of improving energy efficiency, as measured typically by the increasing peak-FLOPs/watts ratio of recent architectures [2]. However, measuring and profiling the energy performance of actual applications is obviously relevant, as very energy-efficient processors (according to the peak-FLOPs/watts ratio) may be highly inefficient when running codes unable to exploit a large fraction of their peak performance. For this reason, in this work, we study the energy efficiency of the Intel Knights Landing (KNL) architecture, using as a benchmarking code a real HPC application that has been heavily optimized for several architectures and routinely used for production runs of fluid-dynamics simulations based on the lattice Boltzmann method [3]. This application is a good representative of a wider class of lattice-based stencil codes, including also HPC Grand Challenge applications such as Lattice Quantum Chromodynamics (LQCD) [4–8].

In this work, we evaluate the impact of a variety of software options—different number of threads per core, different memory configurations (i.e., MCDRAM and DDR4) and different data layouts—on energy consumption of the KNL as it runs different kernels of our benchmark application. Performance evaluations for HPC workloads on this processor were already done in [9], where the impact on the applications performance of the different memory configurations has been considered. In addition, here, we assess the impact of different data structures and of different memory modes, both on performance and energy efficiency. These aspects are becoming increasingly important and recently have been taken into account in [10] for applications of interest for the National Energy Research Scientific Computing Center. Different from this, we focus on the area of computational fluid-dynamics taking into account an application based on Lattice Boltzmann (LB) methods, a common example of the stencil algorithm widely used by several scientific communities.

This paper is an extended version of a proceedings paper presented at the "Mini-symposium on energy aware scientific computing on low power and heterogeneous architectures" held at the ParCo 2017 conference [11]. In addition to the previous results, we further analyze the trade-offs between performance and energy efficiency made possible by processor frequency tuning and assess the corresponding impact on the time-to-solution of our application. To this effect, we introduce a new section discussing the clock-frequency tuning of the KNL using the DVFS (Dynamic Voltage and Frequency Scaling) technique. Here, we measure the time-to-solution and the energy-to-solution of our benchmark application for different processor frequencies, following a strategy developed to study other architectures [12]. We also added a section to better describe the experimental setup used for the tests.

The remainder of the paper is organized as follows: Section 2 gives a brief overview of lattice Boltzmann methods and of the experimental setup; Section 3 presents an overview of the KNL processor; Section 4 describes the technique used to monitor the energy consumption; Section 5 summarizes the performance and energy efficiency results using default frequency governors; while Section 6 presents a study of the performance-energy trade-off using the DVFS technique; finally, in Section 7, we provide our concluding remarks and some ideas for future works.

2. Lattice Boltzmann Methods

Lattice Boltzmann (LB) methods are a class of numerical schemes routinely used in many physics and engineering contexts, to study the dynamics of fluid flows in several different regimes. LB mehtods [13] are based on the synthetic dynamics of populations sitting at the sites of a discrete lattice. Over the years, many different LB methods have been developed; while there are significant differences among them in terms of their detailed structure, physical and numerical accuracy and application domains, all LB methods are discrete in position and momentum spaces; they all share a basic algorithmic structure in which each discrete time-step is obtained by first moving populations from lattice-sites to neighboring lattice-sites (the propagate step) and then recomputing the values of all populations at each site (the collide step). Macroscopic physical quantities (e.g., density, velocity, temperature) are computed as appropriate weighted averages of all population values.

In this work, we consider a state-of-the-art $D2Q37$ LB model that has been extensively used for large-scale simulations of convective turbulence [14] on HPC systems [15]. This model uses 37 populations per lattice site and reproduces the thermo-hydrodynamical evolution of fluids in two dimensions, enforcing the equation of state of a perfect gas ($p = \rho T$) [16,17].

A lattice Boltzmann simulation code starts with an initial assignment of the populations values and then evolves the system in time for as many time-steps as needed, spending most of its computing time in the execution of the propagate and collide kernels. In particular, propagate accounts for ≈30% and collide for ≈70%, according to the different implementations, as shown in detail later.

As already remarked, propagate moves populations across lattice sites according to a stencil that depends on the LB model used. This kernel only performs a large number of sparse memory accesses, and for this reason, it is strongly memory-bound. On the other hand, the collide kernel uses as input

the populations gathered by propagate and performs all the mathematical steps associated with the computation of the new population values. This function is strongly compute-bound, making heavy use of the floating-point units of the processor.

To easily grant the independent execution of both the functions on all the lattice sites, exposing all the available parallelism, we adopt in this work a two-lattice approach, where propagate reads from one copy of the lattice, writing to a second one, while collide does the opposite. Over the years, we have developed several implementations of this LB model, using them for studies of convective turbulence [18] and as benchmarks, profiling the performance of recent HPC hardware architectures based on several commodity CPUs and GPUs [19–23]. In this work, we use an implementation initially developed for Intel CPUs [24] and later ported and optimized for the Intel Knights Corner (KNC) processor [25,26].

Experimental Methodology

For the rest of the paper, we report results collected on a single stand-alone KNL 7230 processor, described in Section 3. We measure the execution time of the main functions of our application, and at the same time, we measure also the energy consumption of processor and memory, as described in Section 4.

In general, we measure the execution time and energy consumption of 1000 iterations of each function or of the full iteration executing both of them, without taking into account lattice initialization and the result correctness check. Collected performance metrics are later averaged over iterations and normalized per lattice site. Having size-independent metrics allows us to compare the performances of simulations over different lattice sizes, in order to test both domain sizes, which are able and which are not able to fit in the high bandwidth memory, as detailed later.

We take into account different implementations of the *D*2*Q*37 LB model, as detailed in Section 5, but for all the tests, we always run on the same GNU/Linux machine, hosting a CentOS 7 distribution with Kernel 3.10.0. The compute node has been always allocated exclusively to our tests. Code is compiled with the Intel compiler Version 17.0.1 and run with one Message Passing Interface (MPI) process and multiple threads. Threads affinity has been set using the Intel compiler environment variables adopting the configuration that led the best performance in each case.

3. The Knights Landing Architecture

The Knights Landing (KNL) architecture is the first generation self-bootable processor based on the Many Integrated Cores (MIC) architecture developed by Intel. This processor integrates an array of 64, 68 or 72 cores together with four high-speed Multi-Channel DRAM (MCDRAM) memory banks, providing an aggregate raw peak bandwidth of more than 450 GB/s [27]. It also integrates 6 DDR4 channels supporting up to 384 GB of memory with a peak raw bandwidth of 115.2 GB/s. Two cores are bonded together into a tile sharing an L2-cache of 1 MB. Tiles are connected by a 2D-mesh of rings and can be clustered in several Non-uniform memory access (NUMA) configurations. In this work we only use the Quadrantcluster configuration where the array of tiles is partitioned into four quadrants, each connected to one MCDRAM controller. This configurations is the one recommended by Intel to run applications using the KNL as a symmetric multi-processor, as it reduces the latency of L2-cache misses, and the 4 blocks of MCDRAM appear as contiguous blocks of memory addresses [28]. Additionally, the MCDRAM can be configured at boot time in *Flat*, *Cache* or *Hybrid* mode. The *Flat* mode configures the MCDRAM as a separate addressable memory, while the *Cache* mode configures the MCDRAM as a last-level cache; the *Hybrid* mode allows one to use the MCDRAM partly as addressable memory and partly as last-level cache [29]. In this work, we only consider the *Flat* and *Cache* configurations.

KNL exploits two levels of parallelism, task parallelism built onto the array of cores and data parallelism using the AVX 512-bit vector (SIMD) instructions. Each core has two out-of-order Vector Processing Units (VPUs) and supports the execution of up to 4 threads. The KNL has a peak theoretical

performance of 6 TFLOPs in single precision and 3 TFLOPs in double precision, and the typical Thermal Design Power (TDP) is 215 W, including MCDRAM memories. For more details on this architecture, see [30].

4. Measuring the Energy Consumption of the KNL

As other Intel processors (from Sandy Bridge architecture onward), the KNL integrates energy meters and a set of Machine-Specific Registers (MSR) that can be read through the Running Average Power Limit (RAPL) interface. In this work, we use the *PACKAGE_ENERGY* and *DRAM_ENERGY* counters to monitor respectively the energy consumption of the processor package (accounting for cores, caches and MCDRAM) and of the DRAM memory system.

A popular way to access these counters is through a library named PAPI [31] providing a common API for energy/power readings for different processors, partially hiding architectural details. The use of this technique to read energy consumption data from Intel processors is an established practice [32], validated by several third parties studies [33,34]. On top of the PAPI library, we have developed a custom library to manage power/energy data acquisition from hardware registers. This library allows benchmarking codes to directly start and stop measurements, using architecture-specific interfaces, such as RAPL for Intel CPUs and the NVIDIA Management Library (NVML) for NVIDIA GPUs [12]. Our library also lets benchmarking codes place markers in the data stream in order to have an accurate time correlation between the running kernels and the acquired power/energy values. This library is available for download as free software (https://baltig.infn.it/COKA/PAPI-power-reader). We use the energy-to-solution (E_s) as our basic metric to measure the energy efficiency of processors running our application. This is defined as a product of time-to-solution (T_s) and average power (P_{avg}) drained while computing the workload: $E_s = T_s \times P_{avg}$. To measure P_{avg} we sample the RAPL hardware counters at 100 Hz thanks to our wrapper library and then convert readout values to watts.

RAPL counters in fact already provide the energy consumption, in Joules, between the sampling intervals, thus to obtain the average power drain during the interval, it is enough to divide the sampled value (in Joules) by the sampling period (in seconds). Then, it is enough to further average over the execution time, to derive P_{avg} of a piece of code, e.g., of a function. The package and DRAM power drains can then be summed to obtain the overall value, or can be analyzed separately.

In the following, we measure this metric for the two most time-consuming kernels of our application, propagate, which is strongly memory-bound, and collide, which is strongly compute-bound.

5. Energy Optimization of Data Structures

The LB application described in Section 2 has been originally implemented using the AoS (Array of Structure) data layout, showing a good performance on CPU processors. Later, its data layout was re-factored, porting the application to GPUs, leading to another implementation based on the SoA (Structure of Array) data layout, giving better performance on these processors.

More recently, two slightly more complex data layouts were introduced [35] in the quest of a single data structure to be used for a portable implementation, able to obtain high performance on most available architectures. We have considered two hybrid data structures, mixing AoS and SoA properties, which we call CSoA (Clustered Structure of Array) and CAoSoA (Clustered Array of Structure of Array). CSoA is a modified SoA scheme that guarantees that vectorized read and writes are always aligned. This is obtained by clustering a set of consecutive elements of each population array in clusters of VL elements, where VL is a multiple of the hardware vector length. The CAoSoA layout is a further optimization of the former, further improving population data locality, as it keeps close, and not only aligned, the addresses of all the population data needed to process each lattice site. A more detailed description of both layouts can be found in [35].

In this work, we test all of the above data layouts on the KNL architecture, measuring the energy consumption of both the package and DRAM. We also run with various numbers of threads and

with different memory configurations: (i) allocating the lattice data only on the DRAM using the Flat/Quadrant configuration; (ii) allocating the lattice only on the MCDRAM using the Flat/Quadrant configuration; or (iii) allocating the lattice only on the DRAM, but using the MCDRAM as a last level cache, using the Cache/Quadrant configuration. The last case is relevant when using very large lattices, which do not fit in the MCDRAM. As we consider different lattice sizes, we present all our results normalizing them to one lattice site. Our aim is to analyze and highlight possible differences in performance, average power and energy efficiency.

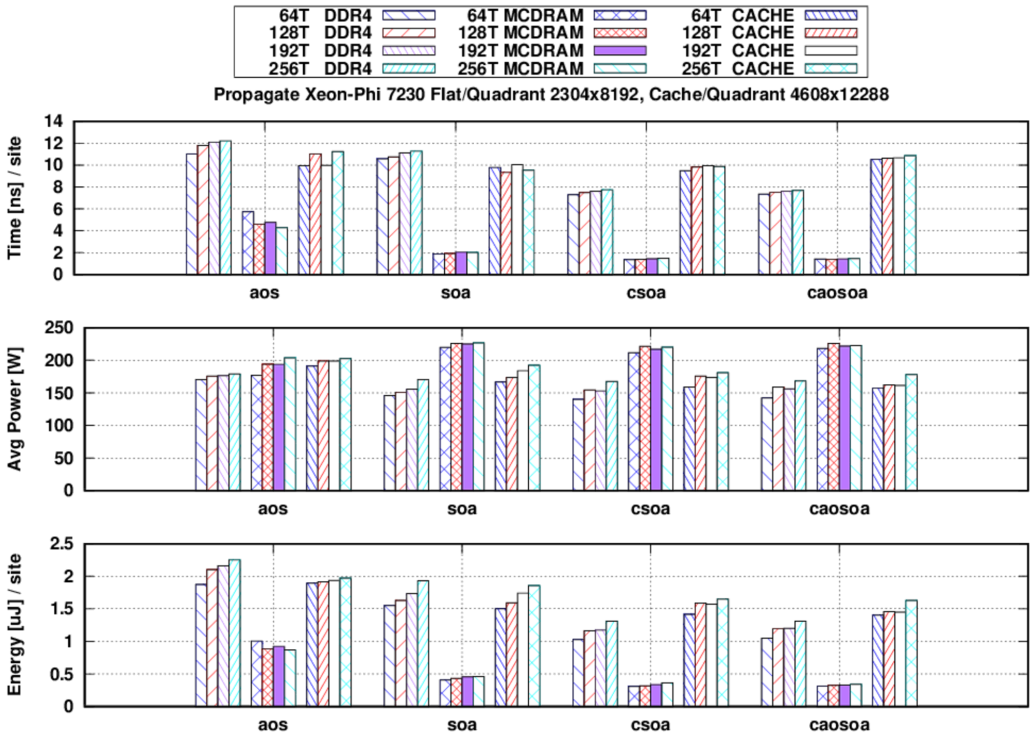
Figure 1a,b shows the main results of our tests, where power and energy values account for the sum of package and DRAM contributions and energy is normalized per lattice site.

Concerning the propagate function, as shown in Figure 1a, using Flat-mode and allocating the lattice in the MCDRAM memory, the maximum bandwidth using the AoS data layout is 138 GB/s, while using SoA, it can be increased to 314 GB/s (i.e., 2.3×), and then further increased to 433 GB/s (3.1× wrt AoS) using the CSoA layout. When using the main DRAM, bandwidth drops for all data layouts: 51 GB/s for AoS, 56 GB/s for SoA and 81 GB/s for CSoA.

Finally, when using the Cache-mode and a larger lattice size, which does not fit in the MCDRAM, we measure an almost constant value of 59, 60 and 62 GB/s respectively for AoS, SoA and CSoA.

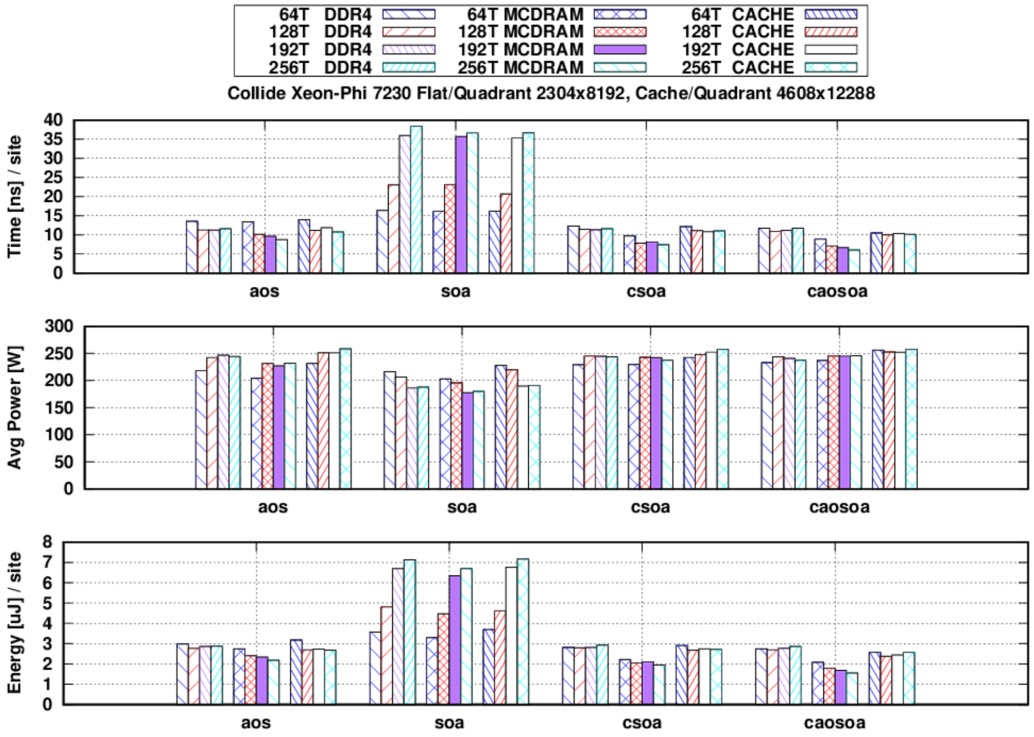
The CAoSoA data layout does not improve over the CSoA for the propagate function, but as we discuss later (see however Figure 1b), it improves the performance of the collide kernel.

Our best E_S figure is obtained using a Flat-MCDRAM configuration and the CSoA data layout, giving an energy reduction of ≈2.5× wrt the AoS data layout.



(**a**)

Figure 1. *Cont.*



(**b**)

Figure 1. Tests run using three different memory configurations (DDR4 Flat, MCDRAM Flat and Cache) and numbers of threads (64, 128, 192 and 256). We show three metrics: time-to-solution (T_s), average power drain (P_{avg}) and energy-to-solution (E_s). Average values over 1000 iterations. (**a**) refers to the propagate and (**b**) to the collide kernels. (**a**) propagate function: T_s in nanoseconds per site (**top**), P_{avg} in watts (**middle**) and E_s in microjoules per site (**bottom**); (**b**) collide function: T_s in nanoseconds per site (**top**), P_{avg} in watts (**middle**) and E_s in microjoules per site (**bottom**). AoS, Array of Structure; SoA, Structure of Array; CSoA, Clustered Structure of Array; CAoSoA, Clustered Array of Structure of Array.

From the point of view of all the evaluated metrics, using just 64 threads is the best choice, since it gives the best performance plus the lowest power drain and energy consumption. However, we see that the performance of the propagate kernel is, within a range of 10%, largely independent of the number of threads used. This is justified by the fact that the propagate function is completely memory-bound, and 64 threads are enough to keep the memory controllers busy at all times.

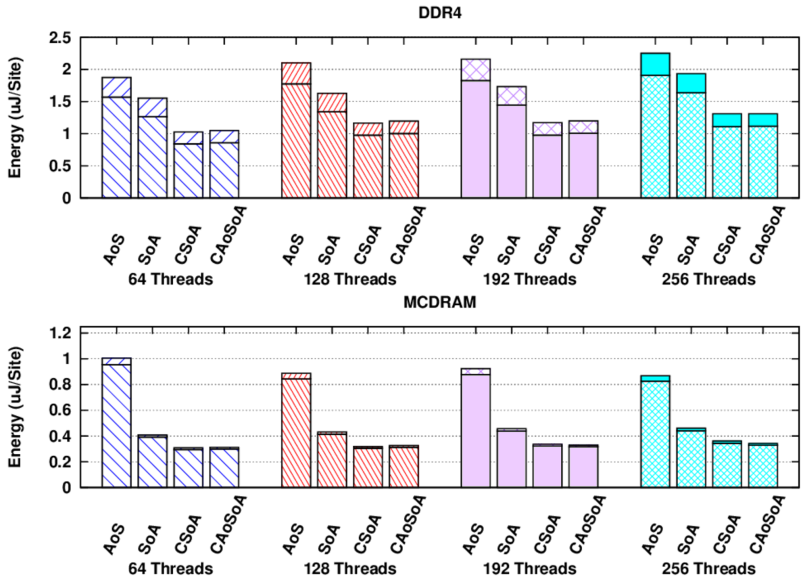
Concerning the collide function, similar plots are shown in Figure 1b, where again the average power drain and E_S are given as the sum of package and DRAM contributions and E_S is normalized per lattice site. For the Flat-MCDRAM configuration, performance increases when changing the data layout from AoS to CSoA, and in this case, it can be further increased changing to CAoSoA. On the other hand, SoA yields the worst performance, as vectorization can be very poorly exploited in this case, since for a given lattice site, the various populations are stored far from each other at non unit-stride addresses [3,35]. When using CAoSoA, we measure a sustained performance of ≈1 TFLOP, corresponding to ≈37% of the raw peak performance of the KNL. It is very nice to remark that the CAoSoA layout also gives the best E_S, ≈2× better than AoS, both for performance and E_S.

Finally, we remark that, opposite to the behavior of propagate, for the collide function, E_S decreases using more threads per CPU (due to a higher computational intensity), apart for the case using the SoA data-structure, for which vectorization is harmed by misaligned memory accesses.

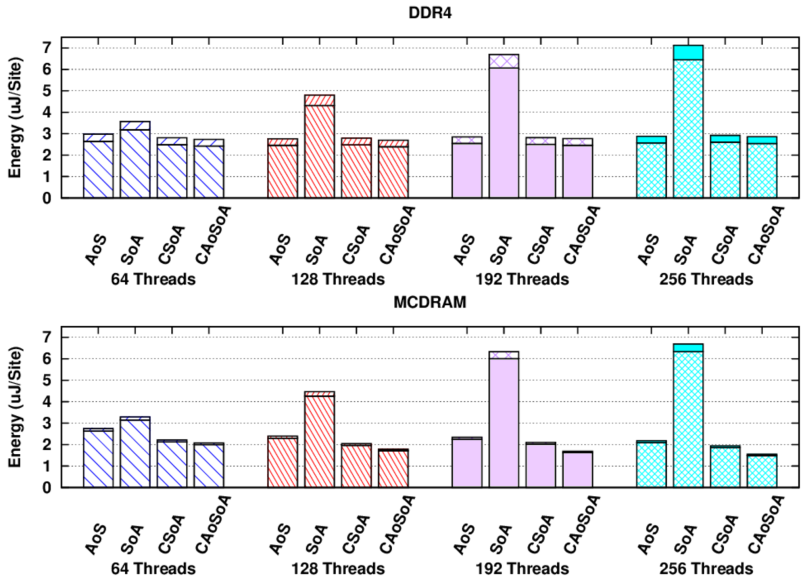
For both functions, we see that the E_S differences, between the various tests performed, are mainly driven by T_S. Despite the fact that the average power drain shows differences up to ≈30%, it seems

always convenient to choose the best performing configuration from the T_S point of view, to obtain also the best energy efficiency.

In Figure 2, we highlight the E_S metrics for all the different data layouts, using the Flat configuration and thus using either the off-chip (DDR4) or the on-chip (MCDRAM) memory, to allocate the whole lattice.



(**a**)



(**b**)

Figure 2. Energy consumption in nano-joules per lattice site, using different memory data layouts and different numbers of threads. Each bar represents the package energy (**bottom**), plus the DRAM energy (top). When using the MCDRAM, the latter is just the DRAM idle energy consumption. (**a**) The propagate function. Flat configuration, using the DDR4 system memory (**top**) and the MCDRAM (**bottom**). Notice the different scales on the y-axes. (**b**) The collide function. Flat configuration, using the DDR4 system memory (**top**) and the MCDRAM (**bottom**).

Here, we display separately the package and DRAM contributions, where for each bar in the plot, package energy is at the bottom and DRAM energy at the top. When using the MCDRAM, its energy consumption is accounted along with the rest of the package, and thus, what is displayed as DRAM energy is just due to the idle power drain.

The main advantage of using the MCDRAM is clearly for the propagate function where we save $\approx 2/3$ of the energy wrt the best performing test run using the DDR4 system memory (take note of the different scales in the y-axes of Figure 2a). Anyhow, also for the collide function, shown in Figure 2b, we can halve the energy consumption. In both cases, the saving of energy is mainly by the reduced execution time, since the DDR4 average power drain accounts for at most $\approx 10\%$ of the total.

6. Energy Efficiency Optimization Using DVFS

To further investigate possible optimization towards energy efficiency, we then select the version of our LBM code giving the best computing performance (i.e., the one using the CAoSoA data layout) and use it to explore the further available trade-off between time-to-solution and energy-to-solution, on the KNL processor. A handy way for users to tune power-related parameters of processors is to use the DVFS support. This is available on most recent processors, giving the possibility to set a specific processor clock frequency [36].

In the previous version of the Xeon Phi (code-named KNC), which was usable just as an accelerator in conjunction with a CPU, DVFS was not directly accessible. Consequently, in the past, it could be used only to reduce the host processor frequency while only the accelerator was in use for computations, leading to energy savings [37]. On the other side, on the KNL, DVFS can be used, as for other Intel CPUs, giving the possibility to set a power cap or to select a specific processor frequency.

Some research works following this opportunity focused on the use of DVFS on the KNL, mainly setting different power caps while running HPC benchmarks [38], or popular linear algebra kernels [39]. In our work, although following a similar approach, we do not set any power cap, but we select a specific processor frequency, aiming to expose the available trade-off between performance and energy efficiency for our application and draw a general conclusion for lattice Boltzmann simulations.

We have recently used the same LB application for similar studies on different architectures [12,21], explicitly setting different processor frequencies in order to highlight interesting trade-offs between performance and energy efficiency. Starting from these previous results, we then investigate the trade-offs available on the KNL processor, using the LB application described in Section 2 as a representative of a wider class of lattice-based simulations.

According to the roofline model [40], any processor based on the von Neumann architecture hosts two different subsystems working together to perform a given computation: a compute subsystem with a given computational performance C (FLOPS/s) and a memory subsystem, providing a bandwidth B (Byte/s) between the processor and memory. The ratio $M_b = C/B$, specific to each hardware architecture, is known as machine-balance [41]. On the other side, every computational task performed by an application is made up of a certain number of operations O, operating on D data items to be fetched from and written into memory. Thus, for any software function, the corresponding ratio $I = O/D$ is referred as the arithmetic intensity, computational intensity or operational intensity.

However, in general, this model tells us that, given a hardware architecture with a machine-balance M_b, the performance of a software function with a computational intensity I would be limited by the hardware memory subsystem if $I < M_b$ or by the compute subsystem if $I > M_b$. Software optimizations attempting to change I in order to mach the available M_b are indeed a well-known practice to maximize application performance, which often translate also into an increase in energy efficiency [12].

Default OS frequency governors commonly set the higher available processor frequency, when an application is running, in order to maximize the system M_b (and thus performance), independent of the actual application needs. As an example, a memory-bound application with $I << M_b$ would not appreciate any performance benefit from a higher processor clock, but more

energy would be spent. On the other side, the clock frequency of the processor can be decreased to lower M_b, in order to match a function-specific I. This would not lead to a performance increase either, but lowering the processor clock (and correspondingly lowering also the processor supply voltage) gives a lower power dissipation (theoretically without impacting performances), and thus a higher energy efficiency.

The roofline model has been recently used to study in detail the KNL architecture [42], highlighting the fact that the M_b value is obviously different if we use the MCDRAM or the DDR4 memory. In our case, the KNL 7230 has a $M_{b,MC} \approx \frac{2662}{450} = 5.9$ FLOPs/byte using the MCDRAM and $M_{b,DDR} \approx \frac{2662}{115.2} = 23.1$ FLOPs/byte using the DDR4 main memory.

Concerning the LB application described in Section 2, the propagate function is completely memory-bound, while the collide function has an arithmetic intensity $I_{\text{collide}} \approx 13.3$. This suggests that on this architecture, $M_{b,MC} < I_{\text{collide}} < M_{b,DDR}$, and thus, the collide function, which is commonly compute-bound on most architectures, should become memory-bound when using the DDR4 main memory.

6.1. Function Benchmarks

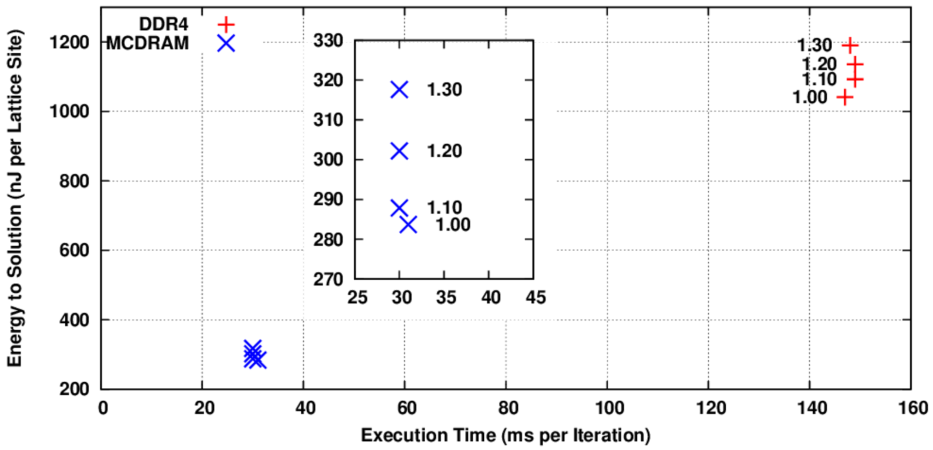
We start benchmarking both the propagate and collide functions, sweeping all the possible frequencies, highlighting the available trade-off between performance and energy efficiency for each of them and selecting optimal frequencies for both E_S and T_S metrics.

Our benchmark application is instrumented to change the processor clock frequencies from within the application itself. It uses the acpi_cpufreq driver of the Linux kernel (as of Linux Kernel 3.9, the default driver for Intel Sandy Bridge and newer CPUs is intel_pstate, so we disabled it in order to be able to use the acpi_cpufreq driver instead.), able to set a specific frequency on each core by calling the cpufreq_set_frequency () function. The Userspace cpufreq governor has to be loaded in advance, in order to disable the governors managing the dynamic frequency scaling and to be able to manually select a fixed processor frequency.

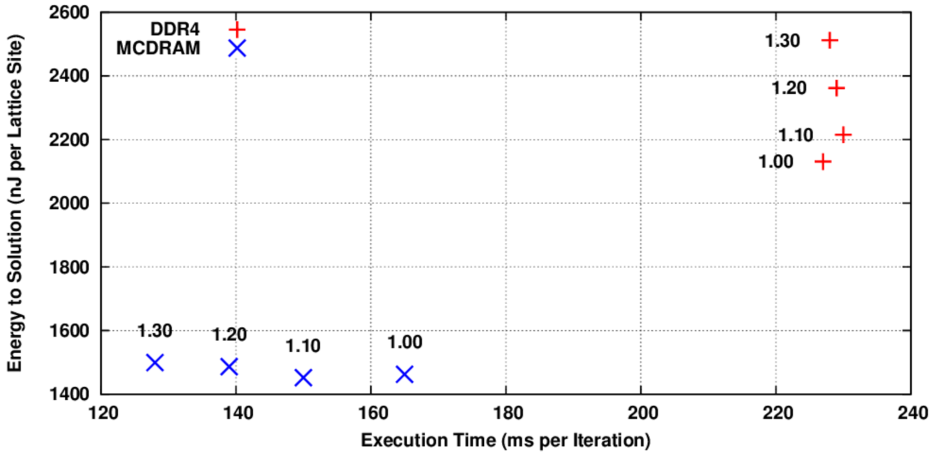
In Figure 3a, we show the energy-to-solution E_S metric as a function of the time-to-solution T_S for the propagate function using the CAoSoA data layout. As expected, this function being completely memory-bound, a decrease in the processor frequency does not lead to an increase in the execution time, but allows one to save $\approx$ 15% of the energy, both for the configurations using the external DDR4 memory or the internal MCDRAM.

In Figure 3b, we show the results of the same test for the collide function. In this case, as predicted by the roofline model, this function is strongly compute-bound using the MCDRAM memory, while it becomes memory-bound when using the slower DDR4 memory. In the former case, in fact, its T_S is heavily impacted by a processor frequency decrease, while there is a negligible E_S benefit. On the other hand, in the latter case, an $\approx$20% saving of energy can be appreciated with almost no impact on performance.

These findings suggest that the optimal frequency for the propagate function, as expected, is always the lowest value (i.e., 1.0 GHz), independent of the memory configuration used, either MCDRAM or DDR4. This holds true also for the collide function, when using the DDR4 memory, but not when using the MCDRAM. In this latter case, a Pareto front is present, from which to look for a trade-off between E_S and T_S, although the possible energy-saving is in the order of just $\approx$5%, practically suggesting running at full throttle also from the E_S point of view.



(**a**)



(**b**)

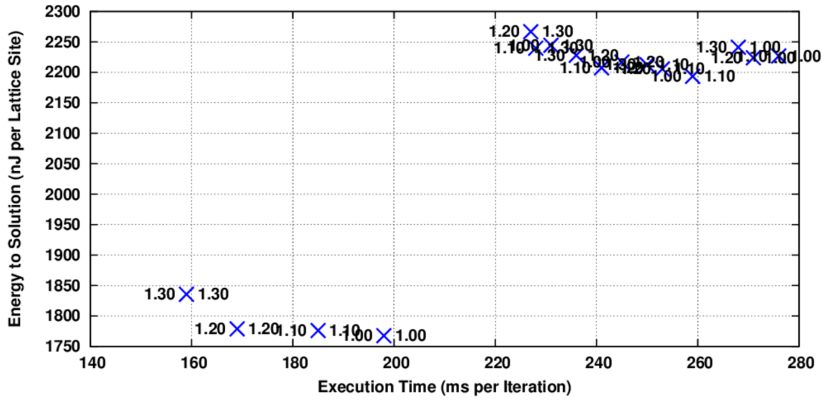
Figure 3. E_S (energy-to-solution) versus T_S (time-to-solution) for the propagate and collide functions adopting the CAoSoA data layout and using 256 threads. KNL cores' frequencies are shown in GHz as labels. For each data point, we show the average over 100 iterations. This measure has been performed three times, and the results are the same within a range of 1.5% in the E_S. (**a**) The propagate function using respectively the DDR4 (red) and MCDRAM (blue) memories; (**b**) the collide function using respectively the DDR4 (red) and MCDRAM (blue) memories.

6.2. Full Application Results

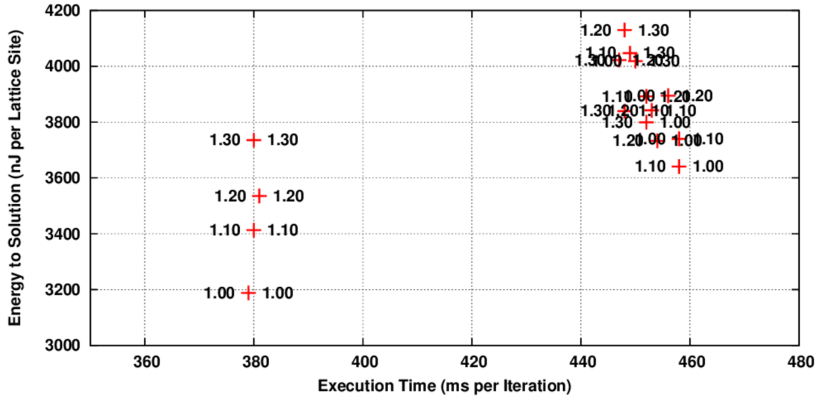
Using the results from the previous section, we conclude that, at least for the case of our application, the best strategy for computing performance and energy efficiency is to store the lattice in MCDRAM and use two different clock frequencies for the two main functions—propagate and collide—of the application. In particular, the processor frequency should be lowered to the minimum value before running the propagate kernel and then increased to the maximum value before executing the collide kernel. To test the feasibility and the corresponding benefits of this strategy, we run a real simulation test changing the clock frequency of the KNL processor from the application itself and try all available clock frequencies for the two functions.

In Figure 4a, we show our results, allocating the lattice to the MCDRAM memory and changing the KNL cores' frequency before launching each function. No frequency changes are made for the tests

in which both functions are run at the same frequency. We see that the performance of all runs adopting different frequencies for the two functions is badly impacted. This is due to the fact that the latency time associated with a change in core frequency is large, of the order of tens of milliseconds (two changes for each iteration). This phenomena is the same as already observed for NVIDIA GPUs [12], where the time cost of each clock change has been measured as ≈10 ms, in sharp contrast to Intel Haswell CPUs, where it is ≈10 µs. In Figure 4b, we show the results of the same tests allocating the lattice to the DDR4. Similar conclusions can be drawn concerning the time cost of each clock change, but we also see that the whole application behavior is mainly memory-bound when running on the DDR4, while it is compute-bound when running on MCDRAM. This suggest that for relatively small lattice sizes, able to fit in the MCDRAM, there is a possible trade-off between performance and energy efficiency, but at first approximation, the highest frequency would be desirable. On the contrary, for large lattice sizes, not able to fit in the MCDRAM, almost ≈20% of the energy could be saved, lowering the KNL cores' frequency to the minimum value, without impacting the performance of the whole application.



(**a**)



(**b**)

Figure 4. E_S (energy-to-solution) versus T_S (time-to-solution) for the whole simulation adopting the CAoSoA data layout and using 256 threads. KNL cores' frequencies for the two functions are shown in GHz; the label on the left for propagate and the label on the right for collide. (**a**) Full simulation storing the lattice in the MCDRAM memory; (**b**) full simulation storing the lattice in the DDR4 memory.

7. Conclusions and Future Works

In this work, we have investigated the energy efficiency of the Intel KNL for lattice Boltzmann applications, assessing the energy to solution for the most relevant compute kernels, i.e., propagate and collide. Based on our experience related to our application in using the KNL and to the experimental measures we have done, some concluding remarks are in order:

1. Applications previously developed for ordinary x86 multi-core CPUs can be easily ported and run on KNL processors. However, the performance is strongly related to the level of vectorization and core parallelism that applications are able to exploit;
2. For LB (and for many other) applications, appropriate data layouts play a relevant role to allow for vectorization and for an efficient use of the memory sub-system, improving both computing and energy efficiency;
3. If application data fit within the MCDRAM, the performance of KNL is very competitive with that of recent GPUs in terms of both computing and energy efficiency; unfortunately, if this is not the case, computing performance is strongly reduced;
4. Given the machine-balance reduction when using DDR4, instead of MCDRAM, functions, which are commonly compute-bound on most architectures, may become memory-bound in this condition;
5. To simulate large lattices that do not fit in the MCDRAM, it is then important to be able to split them across several KNLs to let every sub-lattice fit in the MCDRAM, as is commonly done when running LB applications on multiple GPUs [23];
6. If it is not possible to split the data domain across several processors, the performance degradation could be compensated by an energy savings of up to 20% using DVFS to reduce the cores' frequency;
7. As for GPU devices [12], also on the KNL, due to the time needed to change the frequency of all the cores, a function by function selection of core frequencies is not viable for LB applications.

In the future, we plan to further investigate the energy efficiency of the KNL, comparing it also to other recent architectures and to different LBM implementations. As an example, we plan to compare this implementation with others, having the propagate and collide functions fused together [43], in order to analyze the impact on the trade-off between performance and energy efficiency, given the different computational intensities. We also plan to compare this implementation to others, aiming to reduce the memory footprint [44], in order to evaluate how the data "compression" techniques helping to fit in the MCDRAM impact the performance and energy efficiency of this architecture.

Moreover, we would like to adopt more handy tools for performance and energy profiling [45], allowing us to correlate processor performance counters with performance and energy metrics, fostering finer grained analysis.

Author Contributions: The authors have jointly contributed to the development of this project and to the writing of this research paper. In particular: E.C. focused on the development and implementation of tools to set the processor clock frequencies and analysis of the corresponding energy consumption; A.G. and S.F.S. have contributed to the design and implementation of the different data-structures and to the analysis of computing performances; and R.T. has designed and developed the lattice Boltzmann algorithm.

Funding: This work was done in the framework of the COKA and COSA projects of INFN. This research was funded by MIUR under PRIN2015 programme and A.G. has been supported by Marie Sklodowska-Curie Action, Grant Agreement No.642069.

Acknowledgments: We would like to thank CINECA (Italy) for the access to their HPC systems.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ge, R.; Feng, X.; Song, S.; Chang, H.C.; Li, D.; Cameron, K.W. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Paral. Distrib. Syst.* **2010**, *21*, 658–671, doi:10.1109/TPDS.2009.76.
2. Attig, N.; Gibbon, P.; Lippert, T. Trends in supercomputing: The European path to exascale. *Comput. Phys. Commun.* **2011**, *182*, 2041–2046, doi:10.1016/j.cpc.2010.11.011.
3. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccione, R. Early experience on using Knights Landing processors for Lattice Boltzmann applications. In Proceedings of the 12th International Parallel Processing and Applied Mathematics Conference, Lublin, Poland, 10–13 September 2017; Volume 1077, pp. 1–12, doi:10.1007/978-3-319-78024-5_45.
4. Bernard, C.; Christ, N.; Gottlieb, S.; Jansen, K.; Kenway, R.; Lippert, T.; Lüscher, M.; Mackenzie, P.; Niedermayer, F.; Sharpe, S.; et al. Panel discussion on the cost of dynamical quark simulations. *Nuclear Phys. B Proc. Suppl.* **2002**, *106*, 199–205, doi:10.1016/S0920-5632(01)01664-4.
5. Bilardi, G.; Pietracaprina, A.; Pucci, G.; Schifano, F.; Tripiccione, R. *The Potential of on-Chip Multiprocessing for QCD Machines*; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2005; Volume 3769, pp. 386–397, doi:10.1007/11602569_41.
6. Bonati, C.; Calore, E.; Coscetti, S.; D'Elia, M.; Mesiti, M.; Negro, F.; Schifano, S.F.; Tripiccione, R. Development of scientific software for HPC architectures using OpenACC: the case of LQCD. In Proceedings of the 2015 International Workshop on Software Engineering for High Performance Computing in Science (SE4HPCS), Florence, Italy, 18 May 2015; pp. 9–15, doi:10.1109/SE4HPCS.2015.9.
7. Bonati, C.; Coscetti, S.; D'Elia, M.; Mesiti, M.; Negro, F.; Calore, E.; Schifano, S.F.; Silvi, G.; Tripiccione, R. Design and optimization of a portable LQCD Monte Carlo code using OpenACC. *Int. J. Mod. Phys. C* **2017**, *28*, doi:10.1142/S0129183117500632.
8. Bonati, C.; Calore, E.; D'Elia, M.; Mesiti, M.; Negro, F.; Sanfilippo, F.; Schifano, S.; Silvi, G.; Tripiccione, R. Portable multi-node LQCD Monte Carlo simulations using OpenACC. *Int. J. Mod. Phys. C* **2018**, *29*, doi:10.1142/S0129183118500109.
9. Peng, I.B.; Gioiosa, R.; Kestor, G.; Cicotti, P.; Laure, E.; Markidis, S. Exploring the Performance Benefit of Hybrid Memory System on HPC Environments. In Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, USA, 29 May–2 June 2017; pp. 683–692, doi:10.1109/IPDPSW.2017.115.
10. Allen, T.; Daley, C.S.; Doerfler, D.; Austin, B.; Wright, N.J. Performance and Energy Usage of Workloads on KNL and Haswell Architectures. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*; Jarvis, S., Wright, S., Hammond, S., Eds.; Springer: New York, NY, USA, 2018; pp. 236–249, doi:10.1007/978-3-319-72971-8_12.
11. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccione, R. Energy-efficiency evaluation of Intel KNL for HPC workloads. In *Parallel Computing is Everywhere*; Advances in Parallel Computing; IOS: Amsterdam, The Netherlands, 2018; Volume 32, pp. 733–742, doi:10.3233/978-1-61499-843-3-733.
12. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccione, R. Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications. *Concurr. Comput. Pract. Exp.* **2017**, *29*, 1–19, doi:10.1002/cpe.4143.
13. Succi, S. *The Lattice-Boltzmann Equation*; Oxford University Press: Oxford, UK, 2001.
14. Biferale, L.; Mantovani, F.; Sbragaglia, M.; Scagliarini, A.; Toschi, F.; Tripiccione, R. Second-order closure in stratified turbulence: Simulations and modeling of bulk and entrainment regions. *Phys. Rev. E* **2011**, *84*, 016305, doi:10.1103/PhysRevE.84.016305.
15. Biferale, L.; Mantovani, F.; Pivanti, M.; Sbragaglia, M.; Scagliarini, A.; Schifano, S.F.; Toschi, F.; Tripiccione, R. Lattice Boltzmann fluid-dynamics on the QPACE supercomputer. *Procedia Comput. Sci.* **2010**, *1*, 1075–1082, doi:10.1016/j.procs.2010.04.119.
16. Sbragaglia, M.; Benzi, R.; Biferale, L.; Chen, H.; Shan, X.; Succi, S. Lattice Boltzmann method with self-consistent thermo-hydrodynamic equilibria. *J. Fluid Mech.* **2009**, *628*, 299–309, doi:10.1017/S002211200900665X.

17. Scagliarini, A.; Biferale, L.; Sbragaglia, M.; Sugiyama, K.; Toschi, F. Lattice Boltzmann methods for thermal flows: Continuum limit and applications to compressible Rayleigh–Taylor systems. *Phys. Fluids* **2010**, *22*, 055101, doi:10.1063/1.3392774.
18. Biferale, L.; Mantovani, F.; Sbragaglia, M.; Scagliarini, A.; Toschi, F.; Tripiccione, R. Reactive Rayleigh-Taylor systems: Front propagation and non-stationarity. *EPL* **2011**, *94*, 54004, doi:10.1209/0295-5075/94/54004.
19. Biferale, L.; Mantovani, F.; Pivanti, M.; Pozzati, F.; Sbragaglia, M.; Scagliarini, A.; Schifano, S.F.; Toschi, F.; Tripiccione, R. A Multi-GPU Implementation of a D2Q37 Lattice Boltzmann Code. In Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics, Torun, Poland, 11–14 September 2011; Revised Selected Papers, Part I; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2012; pp. 640–650, doi:10.1007/978-3-642-31464-3_65.
20. Calore, E.; Schifano, S.F.; Tripiccione, R. On Portability, Performance and Scalability of an MPI OpenCL Lattice Boltzmann Code. In *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, 25–26 August 2014;* Revised Selected Papers, Part II; Lecture Notes in Computer Science; Springer: Cham, Switherland, 2014; pp. 438–449, doi:10.1007/978-3-319-14313-2_37.
21. Calore, E.; Schifano, S.F.; Tripiccione, R. *Energy-Performance Tradeoffs for HPC Applications on Low Power Processors*; Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Springer: Berlin/Heidelberg, Germany, 2015; Volume 9523, pp. 737–748, doi:10.1007/978-3-319-27308-2_59.
22. Calore, E.; Gabbana, A.; Kraus, J.; Schifano, S.F.; Tripiccione, R. Performance and portability of accelerated lattice Boltzmann applications with OpenACC. *Concurr. Comput. Pract. Exp.* **2016**, *28*, 3485–3502, doi:10.1002/cpe.3862.
23. Calore, E.; Gabbana, A.; Kraus, J.; Pellegrini, E.; Schifano, S.F.; Tripiccione, R. Massively parallel lattice-Boltzmann codes on large GPU clusters. *Paral. Comput.* **2016**, *58*, 1–24, doi:10.1016/j.parco.2016.08.005.
24. Mantovani, F.; Pivanti, M.; Schifano, S.F.; Tripiccione, R. Performance issues on many-core processors: A D2Q37 Lattice Boltzmann scheme as a test-case. *Comput. Fluids* **2013**, *88*, 743–752, doi:10.1016/j.compfluid.2013.05.014.
25. Crimi, G.; Mantovani, F.; Pivanti, M.; Schifano, S.F.; Tripiccione, R. Early Experience on Porting and Running a Lattice Boltzmann Code on the Xeon-phi Co-Processor. *Procedia Comput. Sci.* **2013**, *18*, 551–560, doi:10.1016/j.procs.2013.05.219.
26. Calore, E.; Demo, N.; Schifano, S.F.; Tripiccione, R. Experience on Vectorizing Lattice Boltzmann Kernels for Multi- and Many-Core Architectures. In Proceedings of the 11th International Conference on Parallel Processing and Applied Mathematics, Krakow, Poland, 6–9 September 2015; Revised Selected Papers, Part I; Lecture Notes in Computer Science; Springer: Cham, Switherland, 2016; pp. 53–62, doi:10.1007/978-3-319-32149-3_6.
27. McCalpin, J.D. *STREAM: Sustainable Memory Bandwidth in High Performance Computers*; University of Virginia: Charlottesville, VA, USA; A Continually Updated Technical Report. Available online: http://www.cs.virginia.edu/stream/ (accessed on 3 June 2018).
28. Colfax. Clustering Modes in Knights Landing Processors. Available online: https://colfaxresearch.com/knl-numa/ (accessed on 3 June 2018).
29. Colfax. MCDRAM as High-Bandwidth Memory (HBM) in Knights Landing Processors: Developers Guide. Available online: https://colfaxresearch.com/knl-mcdram/ (accessed on 3 June 2018).
30. Sodani, A.; Gramunt, R.; Corbal, J.; Kim, H.S.; Vinod, K.; Chinthamani, S.; Hutsell, S.; Agarwal, R.; Liu, Y.C. Knights landing: Second-generation Intel Xeon Phi product. *IEEE Micro* **2016**, *36*, 34–46, doi:10.1109/MM.2016.25.
31. Dongarra, J.; London, K.; Moore, S.; Mucci, P.; Terpstra, D. Using PAPI for hardware performance monitoring on Linux systems. In Proceedings of the Conference on Linux Clusters: The HPC Revolution, Champaign, IL, USA, 25–27 June 2001; Volume 5.
32. Weaver, V.; Johnson, M.; Kasichayanula, K.; Ralph, J.; Luszczek, P.; Terpstra, D.; Moore, S. Measuring Energy and Power with PAPI. In Proceedings of the 1st International Conference on Parallel Processing Workshops (ICPPW), Pittsburgh, PA, USA, 10–13 September 2012; pp. 262–268, doi:10.1109/ICPPW.2012.39.

33. Hackenberg, D.; Schone, 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), Hyderabad, India, 25–29 May 2015; pp. 896–904, doi:10.1109/IPDPSW.2015.70.
34. Desrochers, S.; Paradis, C.; Weaver, V.M. A Validation of DRAM RAPL Power Measurements. In Proceedings of the Second International Symposium on Memory Systems, Alexandria, VA, USA, 3–6 October 2016; pp. 455–470, doi:10.1145/2989081.2989088.
35. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccione, R. Optimization of lattice Boltzmann simulations on heterogeneous computers. *Int. J. High Perform. Comput. Appl.* **2017**, doi:10.1177/1094342017703771.
36. Etinski, M.; Corbalán, J.; Labarta, J.; Valero, M. Understanding the future of energy-performance trade-off via DVFS in HPC environments. *J. Paral. Distrib. Comput.* **2012**, *72*, 579–590, doi:10.1016/j.jpdc.2012.01.006.
37. Lawson, G.; Sosonkina, M.; Shen, Y. Performance and Energy Evaluation of CoMD on Intel Xeon Phi Co-processors. In Proceedings of the 2014 Hardware-Software Co-Design for High Performance Computing, New Orleans, LA, USA, 17–17 November 2014; pp. 49–54, doi:10.1109/Co-HPC.2014.12.
38. Lawson, G.; Sundriyal, V.; Sosonkina, M.; Shen, Y. Runtime Power Limiting of Parallel Applications on Intel Xeon Phi Processors. In Proceedings of the 2016 4th International Workshop on Energy Efficient Supercomputing (E2SC), Salt Lake City, UT, USA, 14–14 November 2016; pp. 39–45, doi:10.1109/E2SC.2016.011.
39. Haidar, A.; Jagode, H.; YarKhan, A.; Vaccaro, P.; Tomov, S.; Dongarra, J. Power-aware computing: Measurement, control, and performance analysis for Intel Xeon Phi. In Proceedings of the 2017 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 12–14 September 2017; pp. 1–7, doi:10.1109/HPEC.2017.8091085.
40. Williams, S.; Waterman, A.; Patterson, D. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* **2009**, *52*, 65–76, doi:10.1145/1498765.1498785.
41. McCalpin, J.D. Memory Bandwidth and Machine Balance in Current High Performance Computers. In Proceedings of the IEEE Technical Committee on Computer Architecture (TCCA) Newsletter, Santa Margherita Ligure, Italy, 22–24 June 1995; pp. 19–25.
42. Doerfler, D.; Deslippe, J.; Williams, S.; Oliker, L.; Cook, B.; Kurth, T.; Lobet, M.; Malas, T.; Vay, J.L.; Vincenti, H. Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor. In *High Performance Computing*; Taufer, M., Mohr, B., Kunkel, J.M., Eds.; Kluwer Academic/Plenum Press: Dordrecht, The Netherlands, 2016; pp. 339–353, doi:10.1007/978-3-319-46079-6_24.
43. Valero-Lara, P.; Igual, F.D.; Prieto-Matías, M.; Pinelli, A.; Favier, J. Accelerating fluid–solid simulations (Lattice-Boltzmann & Immersed-Boundary) on heterogeneous architectures. *J. Comput. Sci.* **2015**, *10*, 249–261, doi:10.1016/j.jocs.2015.07.002.
44. Valero-Lara, P. Reducing memory requirements for large size LBM simulations on GPUs. *Concurr. Comput. Pract. Exp.* **2017**, *29*, e4221, doi:10.1002/cpe.4221.
45. Mantovani, F.; Calore, E. Multi-Node Advanced Performance and Power Analysis with Paraver. In *Parallel Computing is Everywhere*; Advances in Parallel Computing; Springer: Berlin, Germany, 2018; Volume 32, pp. 723–732, doi:10.3233/978-1-61499-843-3-723.