The Journal of Supercomputing
https://doi.org/10.1007/s11227-024-06066-0

Energy efficiency and performance analysis of a legacy atomic scale materials modeling simulator (VASP)

Isidoro Nieves-Pírez[1] · Alfonso Muñoz[2] · Francisco Almeida[1] · Vicente Blanco[1]

Accepted: 12 March 2024

Abstract

This work tackles the performance and energy consumption analysis of a legacy scientific application, the VASP (Vienna Ab-initio Simulation Package), an application commonly used by physicists and chemists for modeling materials at the atomic scale. Many of these scientific applications have been implemented in Fortran, where energy metrics instrumentation is not straightforward. We obtained performance figures (execution time and energy consumption) by instrumenting the source code using EML. This energy measurement library has been modified to introduce Fortran interfaces for these metrics. The analysis was carried out using different matrix algebra libraries, parallelization techniques, and hardware platforms, emphasizing on the MPI, OpenMP, and CUDA parallel implementations of the algorithms used in VASP. We employ various material specifications (atomic structures) and molecular sizes of a silicon-based crystal to create a set of benchmarks for these specifications, leading to some recommendations for final users regarding performance improvements. The proposed benchmarking technique assists the user in selecting the right combination of problem size, compilers, and parallelization options available in VASP. For a given system platform, the user will be able to determine not only the architecture to use (GPU or multicore processors), but also the appropriate library and parallelization according to the atomic structure and molecular size.

Isidoro Nieves-Pírez, Alfonso Muñoz, Francisco Almeida, and Vicente Blanco contributed equally to this work.

✉ Vicente Blanco
vblanco@ull.es

Isidoro Nieves-Pírez
alu0100284829@edu.ull.es

Alfonso Muñoz
amunoz@ull.es

Francisco Almeida
fameida@ull.es

[1] Computer Science and Systems Department, Universidad de La Laguna (ULL), San Francisco de Paula s/n, La Laguna 38270, Spain

[2] Instituto de Materiales y Nanotecnología, Universidad de La Laguna (ULL), La Laguna, Spain

Published online: 16 April 2024

Keywords Applied computing physics · Performance · Energy aware computing

1 Introduction

General-purpose graphics processing accelerators (GPGPUs) are designed to run simulations and scientific applications, which constitute a new low-cost, high-performance computing platform. To take advantage of this new hardware infrastructure, it is necessary to define paradigms that adapt to the new model. The unified computer device architecture (CUDA [1]) is the answer to this challenge presented by NVIDIA. CUDA is both a parallel computing architecture and a programming model that helps to use the multiple processing cores of the GPGPU to run a vast number of threads in parallel. Applications designed with and for CUDA will be able to assign independent tasks to each thread, significantly increasing computational performance, a transcendental issue for research in computational chemistry, bioinformatics, and other fields.

This programming model has been incorporated into other parallel programming approaches used in high-performance computing, which involve the construction of applications with hybrid parallel programming techniques [2], combining MPI, OpenMP, and CUDA models. It is not simple to program and maintain applications with a large base code and diverse programming models. Therefore, some developers opt to write code for GPU architectures using OpenACC [3], a high-level OpenMP-inspired programming standard for coding applications for these architectures.

This work focuses on analyzing a scientific application that needs to benefit from the performance offered by parallel processing, both on multicore CPUs and GPUs, to obtain efficient and accurate results. We try to analyze the application code to identify the sections where we apply instrumentation to capture detailed performance and energy efficiency measures. Specifically, we study the performance of VASP (Vienna Ab-initio Simulation Package) [4], a legacy software package written in Fortran that allows materials to be modeled at the atomic scale. As an instrumentation tool, we use the EML library [5] to measure power consumption. We developed a specific Fortran interface to communicate EML with VASP and to perform energy measurements in real time. We obtained performance and energy efficiency results for several materials and various molecular sizes with parallel CUDA, MPI, and OMP implementations of VASP.

As objectives, we have proposed characterizing the performance of the VASP application in an MPI/OMP and GPU environment. We compile the legacy source code with the selected matrix algebra libraries and optimization parameters suitable for the chosen parallel environment. We conducted performance and energy consumption measurements that allowed us to compare MPI and GPU environments, primarily in the *Verode* cluster of the ULL. We used the EML library to instrument the legacy Fortran code of the VASP application, allowing us to obtain detailed energy consumption measurements. We analyzed performance and consumption by comparing the results

obtained with the different libraries and configuration parameters, while attempting to maintain or improve the quality of the results. The results of parallel applications must be at least as good as those of their sequential counterparts, particularly in terms of energy and pressure values in the study of solids.

In this paper, we offer three primary contributions.

- We conducted a performance analysis of VASP versions 5.4.4 and 6.2.1, focusing on the most significant configuration parameters, to compare the various parallel execution capabilities when studying molecules with different numbers of atoms using the same sampling mesh. We aimed to identify any potential performance improvements related to the different types of parallelism available.
- We conducted an analysis of energy efficiency by integrating a new EML-Fortran interface into the legacy VASP code. This interface enabled us to measure the energy consumption of each socket and the GPU. We included calls to the EML library in the main loop, or ionic loop, of the program so that measurements could be taken at each iteration.
- We propose a working methodology that helps to make decisions about the hardware architecture and type of parallelism to use with VASP, depending on the size of the molecular structure to be worked on. For example, when dealing with molecules of 40 atoms, it is beneficial from both a performance and energy efficiency standpoint to use a node with two sockets and a GPU. MPI–OMP yields intermediate results, but requires a proper combination of MPI threads and OMP tasks.

The paper is organized into six sections: Sect. 2 describes the research problem of *ab initio* simulations, Sect. 3 describes the computational platform where we perform the performance analysis and detail the instrumentation of legacy code using the EML library, and Sect. 4 collects the most important aspects of the software used and shows the results obtained. Finally, Sects. 5 and 6 introduce the related work and summarize the findings reached with this study.

2 VASP

VASP is a simulation software package that works with plane waves based on density functional theory (DFT) and solves the *Kohn-Sham's* equations. This approach facilitates the theoretical study of compounds through energy and force calculations, allowing for optimization of geometry, simulations of molecular dynamics, and determination of a wide range of physicochemical properties for solids or surfaces for industrial application. The program uses Davidson [6] and RMMDIIS [7, 8] algorithms to perform these calculations. We address the use of the application in heterogeneous CPU–GPU systems, optimizing the hardware, libraries, and compiler selection to improve performance and energy efficiency.

2.1 Algorithms in VASP

Several approaches for parallelization of the algorithms used in VASP can be found in [9], where most of the computational work consists of efficiently solving the *Schrödinger's* equation:

$$H\Psi_{nk} = \varepsilon_{nk}\Psi_{nk} \tag{1}$$

for a system with a large number of electrons, N. Using the lattice periodicity (*Bloch's theorem*) the problem is reduced to working with the primitive crystal cell and a smaller number of electrons, which decreases the quantum numbers n and k characterizing the bands and the integration at k is reduced to localized values within the *first Brillouin Zone* (1BZ). Density functional theory (DFT) makes this problem tractable and allows the problem to be solved self-consistently using the *Kohn-Sham's* equations. The *Kohn-Sham's* equations are similar to a non-interacting *Schrödinger* equation for a fictitious system (the "*Kohn-Sham* system") of non-interacting electrons that generate the same density as any given system of interacting electrons.

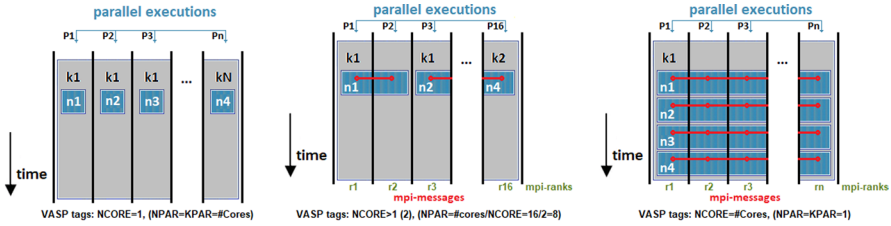
The *k-points* are sampling points in the 1BZ of the material. In periodic calculations, you want to sample the *k-space* essentially to consider the effect of neighboring unit cells: k corresponds to lattice vectors in reciprocal space.

In a previous work, we developed the *Bandas* [10] desktop application, which allows visualizing the material's band structure in a series of points k in high directions of symmetry of the 1BZ. Iterative matrix diagonalization algorithms, such as *Blocked-Davidson* or RMM-DIIS, are used to solve the *Schrödinger's* equation. Because the Hamiltonian H depends on the set of solutions (eigenfunctions Ψ), this must be resolved iteratively until convergence is reached through an iterative matrix diagonalization that requires considerable computational work. To solve the *Schrödinger's* equation, we must perform two large groups of calculations:

(a) *Hamiltonian* The steps to solve the Hamiltonian of the *Schrödinger's* equation for each of its solutions (eigenfunctions Ψ) in each wave vector k are composed of the following calculations:

1. Fourier Transforms
2. PAW projections with matrix-matrix operations
3. Computation of the local potential by product of elements in real space
4. Computation of the nonlocal potential through matrix-matrix operations
5. Computation of kinetic energy by product of elements in reciprocal space
6. Sum of results in reciprocal space and Fourier transforms

(b) *Orthogonalization and diagonalization* Once the *Hamiltonian* is obtained, the solutions to this equation must be orthonormal by applying *Gram-Schmidt* orthogonalization. This group of computations involves global MPI communication across all logical groups of processes (MPI-ranks).



(a) over k–points **(b)** over orbitals **(c)** over plane–wave coeficients

Fig. 1 VASP MPI parallelization

The parallelization strategy implemented by VASP using the message passing interface (MPI) was described by Wende et al in [9]. The hybrid MPI-CUDA implementation of the VASP was described by [11–13]. VASP can apply three levels of parallelism, from a basic level to an advanced level, as follows:

1. *Bloch wave-vectors, k-points (basic-level)*. To avoid heavy computations, the user usually decreases the number of *k-points* when the size of the system increases. All bands are processed in parallel.

   VASP tags: KPAR = #Cores. KPAR is the number of *k-points* that are to be treated in parallel. Several N = #Cores / KPAR compute cores work together on an individual *k-point* where #Cores is the total number of cores. KPAR must be an integer divisor of the total number of cores. The default value of KPAR is 1. The NPAR indicates the number of bands processed in parallel. Its default value is equal to #Cores. This implies that each orbital is processed by a core. The NCORE parameter, which is easier to use, has taken over from the NPAR. NCORE = 1 (default). This parameter determines how many cores work on an orbital (or band) of electrons. Orbitals describe the behavior of an electron within an atom. For simplicity, in the figures we consider 4 orbitals labeled: n1–n4. See Fig. 1a.
2. *The one-electron orbitals at each k-point: n1–n4 (medium-level)*. In fact, the *Schrödinger's* equation can be solved by working directly on each orbital using the algorithms of *Blocked-Davidson* or RMM-DIIS. The number of orbitals increases proportionally to the size of the system. Several bands are processed in parallel. We can refer to a medium-sized system as a structure with 50 atoms. **VASP tags:** For NCORE > 1 (or NPAR=#cores), #cores is the number of cores selected. The relationship between both parameters is established as NCORE =#Cores/NPAR, where #Cores is the total number of cores. As an illustrative example, in Fig. 1b we have set NCORE=2 and #Cores=16. Because each orbital is processed by more than one core, it is necessary to intercommunicate the processing performed by all the cores. That is, among all the MPI-ranks. This communication is carried out by passing messages. MPI allows you to create logical groups of processes. In each group, a process is identified by a *rank* number.
3. *The plane-wave and local basis set coefficients (or equivalently fast Fourier transforms [FFTs]), that is, over the PAW projections used to represent each orbital in multiple MPI-ranks (advanced-level)*. Projector-augmented-wave (PAW) projec-

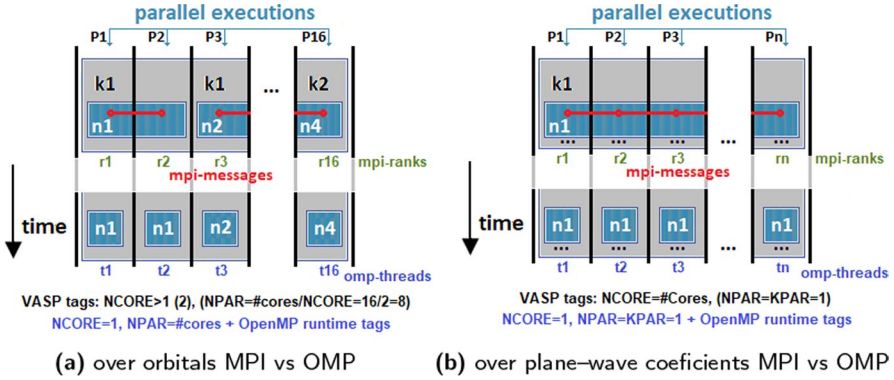


Fig. 2 VASP MPI/OMP parallelization

tion is a mapping of the set of solutions of the *Schrödinger's* equation on a set of localized functions. Large systems require this level even though it adds overhead due to the necessary intercommunication between *MPI-ranks*. MPI message passing is required to calculate the fast Fourier transforms and PAW projections of the orbital. All cores work in parallel on a single band. The plane wave coefficients of each band are distributed among all cores. This level is understood only from a theoretical point of view, since in practice this way of working is slow.

VASP tags: NCORE=#Cores. NPAR=KPAR=1. This is the extreme case for the basic level. Each orbital to be handled by all cores requires a high MPI message to pass between all cores. See Fig. 1c.

For the best results, a mixture of all three levels of parallelism is often applied at the same time. Starting with version 6, VASP offers the possibility of using OpenMP [14] to communicate cores via shared memory instead of MPI messages. The partial or complete replacement of MPI processes by OpenMP (OMP) tasks redefines the VASP parameters for the medium and advanced levels as follows:

2. *The one-electron orbitals at each k-point: n1–n4 (medium-level).*

   VASP tags with OpenMP:

   NCORE = 1. For a fixed value when using OMP and NPAR=#cores, #cores is the number of selected cores. Additionally, we have to set the following OMP parameters at runtime: OMP_NUM_THREADS (number of OMP tasks), OMP_STACKSIZE (stack size used by each task) and OMP_PLACES and OMP_PROC_BIND (to set the affinity policy of the tasks in relation to the hardware). Figure 2a graphically shows how the 16 *MPI-ranks* can be replaced by 16 OMP tasks.
3. *The plane-wave and local basis set coefficients (or equivalently fast Fourier transforms—FFTs).*

   VASP tags with OpenMP:

   NCORE=1. Fixed value when using OMP and NPAR=KPAR=1. In addition, the OMP parameters described in the previous medium level must be established

at runtime. Figure 2b illustrates how to replace the *n MPI-ranks* with *n* OMP tasks.

Since VASP 6, the use of OMP parallelism or GPU parallelism involves setting the parameter NCORE = 1.

Additionally, we can consider other VASP parameters: `ALGO = Normal` (default) applies the *Blocked-Davidson* algorithm, `ALGO = Very fast` the RMM-DIIS algorithm or `ALGO = Fast` for a combination of both (*Davidson* is used for the initial phase, and then VASP switches to RMM-DIIS).

The parameter `NSIM` specifies the number of bands that will be optimally treated at the same time. Its default value is 4. According to the VASP guide, specifying NSIM > 1 allows the program to perform matrix-matrix operations, instead of vector-matrix operations, for evaluations of nonlocal projection operators in real space, as it is able to speed up calculations on some machines. The article *Running VASP on Nvidia GPUs* [15] recommends increasing the value of the parameter to improve the efficiency of the computation on GPUs, although this increase generates greater memory allocation on these devices.

3 Performance and energy analysis

3.1 Computational environment

We have employed a variety of computing systems composed of multiple processor architectures and software components. Experiments were conducted on a high-performance computing cluster called *Verode* and two individual machines with a high number of cores. The final analysis was focused on node *Verode21* for the latest benchmarks due to its powerful CPU, superior GPU, and its ability to provide results for comparison. We used several compilers and libraries to generate a VASP executable, beginning with standard libraries and eventually utilizing optimized libraries from the Intel oneAPI framework [16]. Table 1 summarizes the main hardware characteristics of all the platforms used.

The following matrix algebra libraries have been taken into account for the performance analysis: BLAS, LAPACK, ScaLAPACK, OpenBLAS, and MKL. From a computational standpoint, the VASP workload involves three-dimensional Fast Fourier Transforms (FFTs), matrix-matrix multiplication (BLAS), matrix diagonalization, solving eigenvalue problems (LAPACK and ScaLAPACK), and MPI communications. To be more precise, the two major blocks of calculation needed to solve the *Schrödinger's* equation make use of these algebraic libraries in the following way:

Table 1 Platforms used for the evaluation of VASP

Nodes	CPU-type	Cores	GPU-type	Cores
Asterix	Intel Xeon E5-2699 v3 (Haswell)@2.30 GHz	36	–	
Tenerife	Intel Xeon E5-2699 v4 (Broadwell)@2.20 GHz	44	–	
Marengo	Intel Xeon Gold 6152 (Skylake)@ 2.10 GHz	44	–	
Verode17	Intel Xeon E5-2660 (Sandy Bridge)@ 2.20 GHz	16	Nvidia Tesla K20	2496
Verode18	Intel Xeon E5-2660 (Sandy Bridge)@ 2.20 GHz	16	Nvidia Tesla K40	2880
Verode20	Intel Xeon E5-2698 v3 (Haswell)@ 2.30 GHz	32	Nvidia Tesla K20	2496
Verode21	Intel®Xeon®Gold 6230N (Cascade Lake)@ 2.30 GHz	40	Nvidia Tesla V100	5120

(All nodes with two sockets)

a) Steps 1) and 6): FFT. *Hamiltonian*. Steps 2) to 6): BLAS.

b) *Orthogonalization and diagonalization*. FFT, LAPACK, ScaLAPACK, and communications between *MPI–ranks*.

Parallel VASP runs have been tested in a Message Passing Interface (MPI) environment, which is the current standard for communication between processes. This environment provides the necessary primitives for efficient communications between processes that, combined with computing tools, such as the matrix libraries mentioned above, allows the build of efficient supercomputing software. The MPI implementations tested were OpenMPI [17] version 3.1.5 and IntelMPI [18] oneAPI version 2021.4. Additionally, VASP version 6 has extended the range of possibilities for parallelism in CPU cores using shared memory through the OpenMP standard. OpenMP is included in Intel oneAPI 2021 [16] and Nvidia HPC 23.3 [19].

VASP runs take advantage of GPUs with CUDA runtime and its libraries: cuBLAS (a GPU-accelerated version of BLAS) and cuFFT (GPU-accelerated FFT). We have used CUDA-C 9.1 with VASP 5.4.4 and CUDA-C/OpenACC version 12 with VASP 6.2.1. However, the CUDA-C connection will no longer be supported in VASP 6.3.0 and later versions.

3.2 Instrumentation

VASP provides performance metrics for execution times. However, for energy metrics, the source code must be properly instrumented. For that purpose, we have extended the Energy Measurement Library (EML) [20] to allow use in Fortran codes.

The utilization of EML requires that applications include the calls to the EML routines before and after the sections of code being examined. The usage pattern requires that to recognize all devices supported by EML, the application must first initialize the library by invoking `emlInit`. Before each target section, it will initiate monitoring by

invoking `emlStart`, and then it will terminate monitoring by invoking `emlStop`. This procedure returns the measurements obtained from each of the identified devices. Finally, the application can free up resources by invoking `emlShutdown`.

In our experimentation setting, EML recognizes the RAPL interfaces in Intel processors (one for each socket) and the NVML interfaces for NVIDIA GPU devices as devices.

We have developed an interface module for the EML library to enable C-routines to be called from a Fortran code such as VASP. This module, which includes other legacy subroutines, defines the call interfaces to the EML library (see Listing 1). Additionally, four new subroutines have been added to the VASP Fortran code for initialization, measurement start, and measurement end (see Listing 2) and deallocation of resources. These subroutines call their EML counterparts. The EML routine (`emlStop`) returns a complex data structure which is simplified by a wrapper routine (`emlWrapperStop`). This wrapper routine then returns the consumption data for each CPU socket and GPU to VASP, which is linked with the wrapper routine (see Fig. 1). Consumption data, along with original time data, is written in an output file identified with the `EML` tag.

```
1  module c2f_interface
2  ...
3  interface
4      ! legacy routines
5      ...
6  !----interface EML LIB routines
7      ! emlError emlStart()
8      integer (c_int) function EMLMeter_emlStart()
9                          & bind(c,name="emlStart")
10         import
11     end function EMLMeter_emlStart
12
13     ! emlError_t emlStop(emlData_t *data)
14     integer (c_int) function EMLMeter_emlStop(emlData)
15                               & bind(c,name="emlStop")
16         import
17         type (c_ptr), value :: emlData
18     end function EMLMeter_emlStop
19
20     ! int emlWrapperStop(char *s2f)
21     integer (c_int) function EMLMe_emlWrapperStop(s2f)
22                         & bind(c,name="emlWrapperStop")
23         import
24         character (c_char) :: s2f
25     end function EMLMeter_emlWrapperStop
26     ...
27 !----end interface EML Lib
28 end interface
29 end module c2f_interface
```

Listing 1 C to Fortran interface

```
!-EML Stop measure
SUBROUTINE EML_STOP(IU6)
USE c2f_interface, ONLY : EMLMeter_emlWrapperStop
INTEGER IU6

INTEGER :: emlError
CHARACTER (LEN=120) S2F

S2F = ''
emlError = EMLMeter_emlWrapperStop(S2F)

WRITE(IU6,'(A)') S2F

END SUBROUTINE
```

Listing 2 Fortran routine to log power consumption

3.3 VASP instrumentation

We have compiled and tested various configurations of VASP versions 5.4.4 [13, 21] and 6.2.1. have been compiled and tested. All theese versions are currently employed by different users according to their necesities or resource capabilities usage, and they show different performance patterns that must be analyzed. In detail:

- VASP 5.4.4 with Intel oneAPI: MPI and MPI-GPU
- VASP 6.2.1 with Intel oneAPI: MPI, MPI-OMP, and MPI-GPU
- VASP 6.2.1 with Nvidia-HPC: MPI, MPI-OMP, MPI-GPU, and MPI-OMP-GPU

The MPI-OMP-GPU configuration obtains better performance on the CPU processing side compared to the MPI-GPU configuration, which is reflected in the results. This version of VASP allows multiple combinations of algebraic libraries. Since the Verode cluster incorporates Intel processors, we will find it interesting to test this configuration using the MKL library.

The VASP documentation [22] suggests that when hardware has more than 64 cores sharing the memory, OpenMP should be used. It is essential that MPI processes and OMP tasks are allocated to physical cores in a way that allows them to share the same block of memory within the same socket, so that inter-process communication is optimized. Wende et al. [9] discuss the use of OpenMP in VASP. With regards to OpenACC and GPUs, VASP version 6 only permits one MPI thread per GPU due to the use of NVIDIA Collective Communications Library (NCCL). This limitation can be overcomed by replacing the MPI range with multiple OMP tasks, which can enhance the parallelization of CPU-side processing. However, if the MPI range is replaced with a single OMP task, there is no significant benefit.

To accelerate experimentation and avoid waiting for long execution times, we modified the VASP source code to include the `NITE` parameter. This new feature enable us to set the number of iterations for the main loop (ionic loop) to be executed and allow us to specify the number of iterations below the convergence number of the main loop that is to be executed. Its default value is zero, so the number

of convergence iterations does not change. The parameter can be included in the VASP parameter file (INCAR). To obtain precise readings of energy consumption per iteration, we connected the EML library to the VASP MPI and GPU versions, as described in the previous section. We added to the VASP code the start and end measure calls for each iteration of the main loop, which were already linked to the existing timing start and end calls. We have ensured that all energy measurement routines are protected from parallel executions, so that the measurement and recording of results in the output file are exclusive to each iteration.

4 Experimental results

We created the experiment using a combination of VASP application parameters, architectures, libraries, and optimization parameters. The most important aspects in terms of performance and energy consumption are the following.

- We built the VASP executables using the GCC compiler, Intel Fortran compiler, and Portland Group, Inc (PGI) compiler included in the Nvidia HPC SDK. The GCC compiler was used only in pre-testing for the results presented in Table 3. From this point on, the Intel compiler and the nvHPC compiler have been used. Additionally, OpenMP and OpenACC options were included when available (VASP version 6.2.1). However, along the paper, the compiler used on each computational experiment will be fixed.
- The WAVECAR file, generated by VASP, is a binary file that stores the final wave functions. It contains the number of bands, cut-off energy, basis vectors defining the supercell, eigenvalues, Fermi-weights, and wave functions. VASP reads this file as the first input on each main loop iteration to accelerate the computations. At the end of each iteration, the file is replaced. On the first run of VASP, the WAVECAR file does not exist, so we carry out multiple runs of the initial two or three iterations to observe their effect on performance and energy consumption.
- VASP GPU runs are managed by a combination of MPI and MPI-OMP parallelism, which helps to divide the workload between cores in the CPU. VASP utilizes a hybrid programming model to manage parallelism at both the CPU and GPU level. As discussed in Sect. 2, VASP allows three levels of MPI parallelism: over *k-points*, orbitals, and plane wave coefficients. The GPU level of parallelism is achieved by replacing subroutine calls from standard linear algebra libraries with faster GPU-accelerated implementations, such as by replacing vector operations with matrix operations.
- Table 2 compiles all input sets used in the experiments carried out, both for preliminary tests and for the final results. Data related to each input material can be looked at *The Materials Project* [23, 24] website.

Table 2 Input sets for VASP experimentation

Material	# Atoms	k-points	Input files (Materials project) [25]
ZnSe	2	24/40/60	mp-1190
$GdPO_4$	24	8/24	mp-3735
$Gd_2P_4O_{13}$	76	G*2 2 1	mp-779989
Si	4	G*3 3 3	mp-165
Si	16	G*3 3 3	mp-1079297
Si	24	G*3 3 3	mp-1095269
Si	40	G*3 3 3	mp-1196961
Si	46	G*3 3 3	mp-971662
Si	58	G*3 3 3	mp-1202745
Si	68	G*3 3 3	mp-1203790
Si	100	G*3 3 3	mp-1245041
Si	232	G*3 3 3	mp-1201492

∗ Gamma centred mesh

Table 3 Preliminary experimentation: ZnSe - OpenBLAS vs BLAS

k-points	Serial		MPI(16p)		MPI(32p)		GPU	
	O.B	BLAS	O.B	BLAS	O.B	BLAS	O.B	BLAS
24	68.5	35.9	94.7	7.4	107.3	**5.6**	47.5	30.9
40	233.4	121.5	315.5	22.7	385.3	**16.8**	136.8	83.8
60	588.9	304.7	862.3	58.7	971.1	**44.2**	348.9	210.9

Time in sec - VASP 5.4.4 GCC

4.1 Library acceleration

We analyze the performance of the algebra libraries used in VASP. Several options were available, OpenBLAS, BLAS, or MKL for basic algebraic operations, and we needed to determine which one was best suited for our architectures.

First, we conducted experiments on three systems to compare OpenBLAS and BLAS using the VASP 5.4.4 version compiled with GCC. Table 3 displays the execution times in seconds for a serial version on the *Verode20* system, for an MPI parallel version on the *Verode18* (16 cores) and *Verode20* (32 cores) systems, and for a GPU version on the *Verode18* system (with an NVIDIA GPU K40). For this test we used a 2-atom molecule of a type II-VI semiconductor (Zinc Selenide). The BLAS library proved to be more efficient than OpenBLAS in all cases, with the best performance being achieved with the 32 cores configuration on *Verode20*. Consequently, the GPU version was not considered a viable option for this example.

Next, we focus on comparing the BLAS and MKL libraries. We conducted a study that varied the number of cores (Fig. 3) using the VASP 5.4.4 version

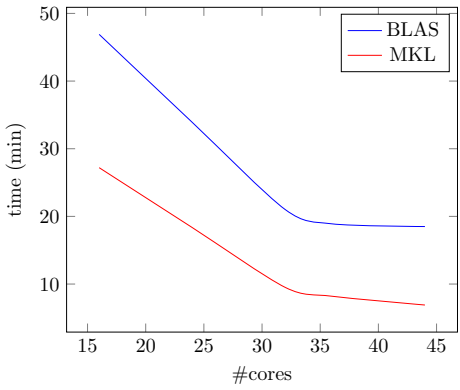


Fig. 3 BLAS vs MKL library—VASP 5.4.4 IoA

compiled with the Intel oneAPI fortran compiler (labeled IoA in Figures). We collected data to create a VASP MPI plot for a monazite ($GdPO_4$) with 24 *k-points* in different multicore system configurations using all the cores of each node. Specifically, measurements were taken with *Verode18* for the 16 core point, with *Verode20* for the 32 core point, with *Asterix* for the 36 core point, and with *Marengo* for 44 cores. Regardless of whether machines with more cores have faster architectures (see Table 1), the performance of the MKL library is always better than the BLAS library.

From the analysis carried out, we conclude that for all the cases studied we have obtained the best performance with the MKL library, and, therefore, this will be the library that we will use for the rest of the experimentation.

4.2 NSIM and time for iteration (LOOP+)

In order to make a fair comparison between different architectures, we must identify parameters that require similar resources from the computational systems. Solving the *Schrödinger's* equation necessitates the execution of an iterative algorithm with a convergence criteria (e.g. different number of iterations for different executions). The execution time for one of these steps is identified in VASP as LOOP+, which is often used as a performance metric in many studies [26].

The `NSIM` parameter has a major impact on performance when using a GPU implementation in VASP 5.4.4. By default, it is set to 4, but increasing it to more than 1 allows the program to perform matrix-matrix operations instead of vector-matrix operations for the evaluation of non-local projection operators in real space, which can speed up calculations on some machines. According to [15], increasing the parameter's value can improve computation efficiency on GPUs, although it also requires more memory allocation on these devices.

We conducted experiments to validate whether the execution time would improve when increasing NSIM for the same number of threads in parallel MPI executions. To carry out these experiments, we used the VASP 5.4.4 version compiled with the Intel compiler. The results in Table 4 for $GdPO_4$ material with the MKL library show that for the best-performing MPI threads, 32, the execution time does not

Table 4 Performance evolution varying the NSIM parameter

Threads	NSIM	8 k-points		24 k–points	
		MPI	GPU	MPI	GPU
(a) $GdPO_4$					
4	4	7.6	5.2	101.2	42.0
4	8	6.3	4.8	81.3	39.8
4	16	7.9	4.5	109.1	42.4
8	4	4.6	6.4	60.8	65.2
8	8	4.5	4.6	50.0	50.2
8	16	4.8	5.2	65.3	52.7
16	4	3.6	6.7	49.4	82.5
16	8	3.5	6.7	40.0	81.1
16	16	3.7	6.3	51.7	68.8
32	4	2.6	10.2	38.7	112.6
32	8	3.3	10.4	34.6	120.4
32	16	3.3	9.9	45.7	110.7

NSIM	MPI(32threads)	GPU(4threads)
(b) $Gd_2P_4O_{13}$		
4	21.5	9.4
8	32.2	7.6
16	31.1	6.9
32	32.3	6.6

Verode21 node, LOOP+ in sec - VASP 5.4.4 IoA

Verode21, LOOP+ in sec - VASP 5.4.4 IoA

decrease as NSIM increases. However, for four threads, better GPU results indicate that execution time can be improved by increasing NSIM.

Previously, in [27], we studied a molecule of 24 atoms and found that the best performance was achieved with MPI. However, GPU performance was slower as a result of the large amount of communication between the CPU and GPU for a compound of this size or smaller. In another gadolinium phosphate with a structure of 76 atoms, $Gd_2P_4O_{13}$, Table 4b shows that the GPU execution has the best performance with NSIM=32. If this parameter is increased too much, it will cause a CUDA memory error on the GPU.

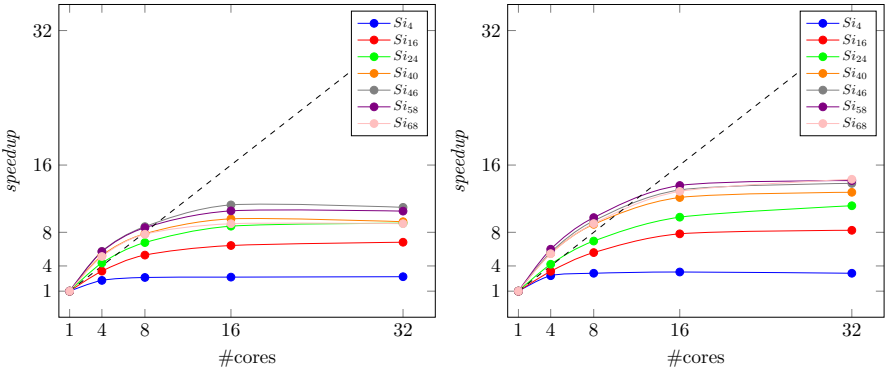
Using the time-per-iteration (LOOP+) as a reference, we have tested the influence of the NSIM parameter in MPI and GPU executions. We conclude that this parameter improves performance for large molecules using GPU parallelism.

4.3 Performance evaluation

To assess the speedup of using MPI and GPU, we performed time measurements on the *sockets* and the GPU for each iteration of the main loop (ionic loop) of VASP. We ran the first ten iterations of the loop for seven silicon molecules of varying sizes (4, 16, 24, 40, 46, 58, and 68 atoms) and obtained the data from the VASP silicon input files from the *The Materials Project* website. This website provides open-access information on materials and tools used by physicists and chemists. Generally, when the number of atoms in the molecule being studied increases, smaller sampling grids (*k-points*) are employed. We have chosen a fixed mesh (3 3 3) to make a valid comparison. This mesh is Gamma centered, i.e., a *k-point* mesh (grid) that is centered around the gamma point of the *Brillouin Zone*. All tests were carried out on the *Verode21* node for two versions of the application: VASP 5.4.4 and VASP 6.2.1, which introduces new parallel execution modes facilitated by OpenMP and OpenACC and the support of a new compiler (Nvidia-HPC). In what follows, the experimental results have been obtained by compiling the VASP 5.4.4 code using the Intel oneAPI fortran compiler and the nvHPC compiler for the 6.2.1 version.

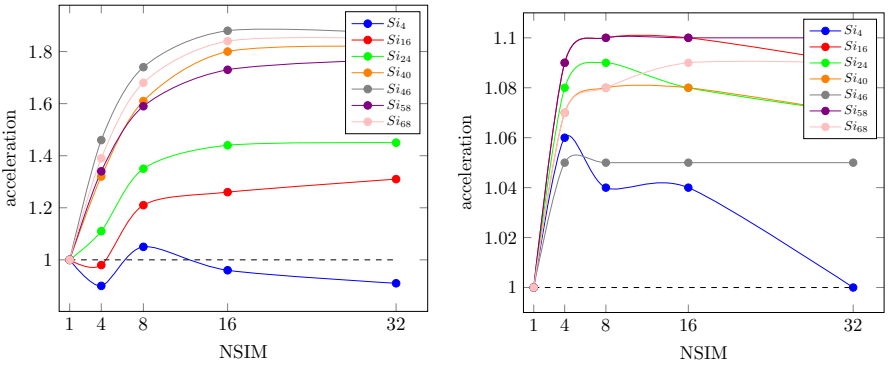
The graph in Fig. 4 illustrates the *speedup* in relation to the number of cores used in the execution. To calculate the *speedup* values, we used a reference sequential execution time, which was determined by measuring the first two MPI iterations of a thread with the parameters NCORE=NSIM=1. Figure 4a reveals that the *speedup* obtained up to 16 cores is quite substantial for most molecule sizes (e.g., 11.26 for 46 atoms and 16 cores). However, with 32 cores, the improvement becomes negligible (e.g., 10.97 for 46 atoms). This is likely due to increased communication between *sockets*. A similar pattern can be seen for VASP 6.2.1 in Fig. 4b.

We examined the speedup generated on GPU runs by increasing the NSIM parameter, which allows the program to substitute vector products for matrix products, thereby increasing the degree of parallelism. We calculate the reference time for GPU by averaging the first two iterations executed with one thread and the parameter NSIM=1. GPU executions were conducted with 4 MPI threads. Figure 5a shows the GPU acceleration relative to the NSIM parameter for the same silicon



(a) MPI version. VASP 5.4.4 IoA **(b)** MPI version. VASP 6.2.1 nvHPC

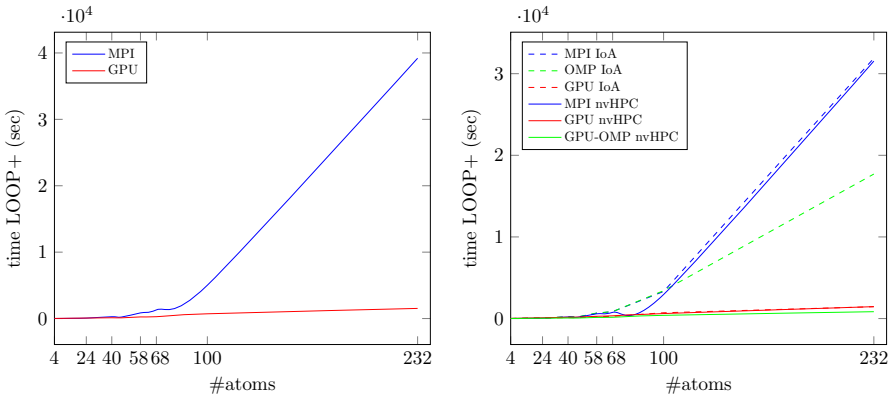
Fig. 4 Speedup for Si structure simulations



(a) VASP 5.4.4 IoA **(b)** VASP 6.2.1 nvHPC

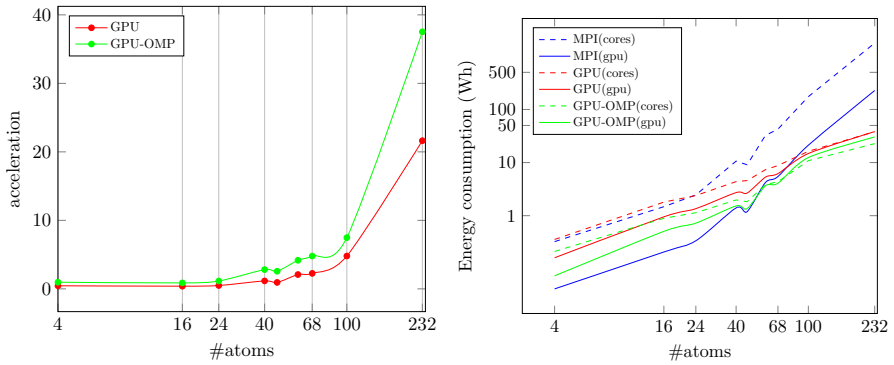
Fig. 5 NSIM acceleration impact for GPU version

molecules used in *speedup* MPI test for VASP 5.4.4. We can see that for molecules with more than four atoms, the increase of NSIM up to a value of 16 produces a significant acceleration (e.g., 1.88 for 46 atoms and NSIM=16). NSIM=32 does not improve the results due to increased communications between CPU and GPU (e.g., 1.87 for 46 atoms). In the case of VASP 6.2.1, GPU acceleration with respect to increase of the NSIM parameter is observed to improve with Intel oneAPI, but surprisingly, it does not improve performance with Nvidia-HPC (Fig. 5b). We have depicted in Fig. 6a the time per iteration of the ionic loop after including the molecule sizes of 100 and 232. The aim is to compare the performance of MPI and GPU when working with the exact sizes of silicon molecules and the same *k-points*. The best MPI times, obtained with 32 threads, and the best GPU times, with four threads and NSIM=32, are shown. Experimentally, it was found that performance decreased when using less than four threads with a GPU. With more than 4, the performance was hindered due to overhead in communications. Regarding the time per iteration



(a) VASP 5.4.4 IoA **(b)** VASP 6.2.1

Fig. 6 LOOP+ execution time



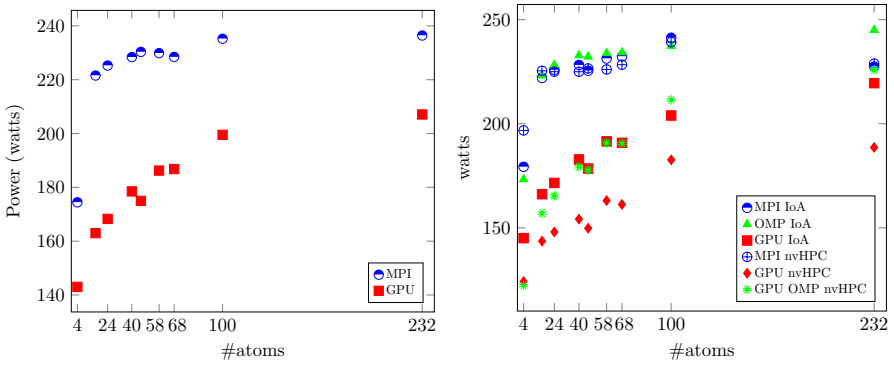
(a) GPU acceleration relative to MPI **(b)** Energy consumption in gpu and cores

Fig. 7 VASP 6.2.1 build with Nvidia HPC SDK framework

in VASP 6.2.1, OMP provides intermediate performance between MPI and GPU (Fig. 6b).

The figure demonstrates that the performance of MPI and GPU is equivalent to up to 40 atoms. After this point, GPU executions are more effective than MPI ones. The MPI–OMP–GPU combination offers the best results from a size of 24 atoms, presenting itself as the fastest option from a practical point of view (Fig. 6b). More in detail, in Fig. 7a we can see the acceleration that GPU and GPU–OMP bring to MPI execution.

We can see that *speedup* with MPI has improved significantly in VASP version 6.2.1 compared to the previous version, 5.4.4. In version 6.2.1, no better results are obtained by increasing the NSIM parameter beyond 8. Hybrid MPI/OMP implementation obtains better performance than the MPI-only version, but far from the GPU implementation with molecules larger than 40 atoms. The best performance results



(a) VASP 5.4.4 IoA **(b)** VASP 6.2.1

Fig. 8 Dissipated power

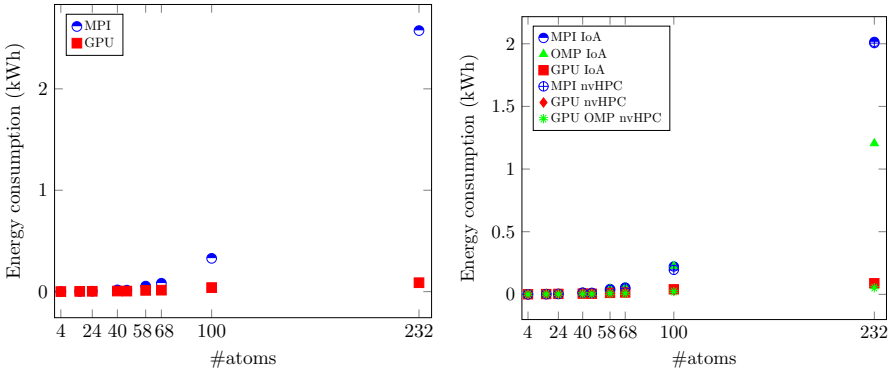
are obtained with GPU–OMP parallelism with molecule sizes greater than about 24 atoms, notably improving the GPU results.

4.4 Energy consumption measures

The increasing attention to measure the energy consumed by the calculations necessary to get the VASP results has been addressed by using the EML library. This library is able to obtain the power consumption on each *socket* and GPU device. We measure the energy consumption and computation time simultaneously using VASP linked with the EML library. We obtain the energy consumption on the *sockets* and GPU at each iteration of the main VASP loop. We conducted experiments with different sample times from the EML library: 1ms, 10ms, 100ms, and 1s. And we find that with a low sampling rate (1s), we can maintain precision without significantly impacting performance measurement.

The experiments were designed to measure the energy consumption of the same silicon structures studied in the previous subsection. Our aim was to compare the energy consumption of MPI, OMP, and GPU executions for different sizes of Si molecules. We obtain the average total consumption of the two *sockets* and the GPU for the first ion loop iteration of each silicon structure ten times. As mentioned in the last paragraph of the 4.3 section, 32 threads were used for the best MPI performance results and four threads and NSIM=32 for the best GPU results. Therefore, we obtained the dissipated power and energy consumption measurements for the executions with these parameters. These experiments were carried out with the VASP 5.4.4 version compiled with Intel oneAPI and the 6.2.1 version compiled with Intel oneAPI and Nvidia HPC SDK frameworks.

Figure 8a represents the instantaneous power dissipated by the calculations of the selected silicon molecules and Fig. 9a their energy consumption measured on node *Verode21* with VASP 5.4.4. For each molecule size, we have used the same *k-points* and input parameters in the MPI and GPU calculations. For both MPI and GPU tests, we add the total energy of both sockets and that of the GPU,



(a) VASP 5.4.4 IoA **(b)** VASP 6.2.1

Fig. 9 Energy consumption

which is not operational in the case of MPI runs (but has an impact on power consumption). In Fig. 8a, we can see that the power dissipated by the calculations in the GPU for any molecule size is lower than that obtained using MPI parallelism. From the point of view of energy efficiency, we can see in Fig. 9a that the use of a GPU with VASP represents a clear advantage over MPI/OMP parallelism from molecular structures of about 40 atoms and beyond.

VASP 6.2.1 addresses similar power dissipation (Fig. 8b) and energy consumption (Fig. 9b) of various sizes of silicon molecules. In this case, we can compare MPI, OMP, and GPU implementations compiled with Intel oneAPI, and MPI, GPU and GPU–OMP implementations compiled with Nvidia HPC SDK framework. Figure 7b displays a comparison of energy consumption between CPU cores and GPU for MPI, GPU, and GPU-OMP, depicted on a logarithmic scale to facilitate visualization across various molecular sizes. In general, the multicore processor consumes more energy for all structures, while GPU-OMP version consumes less power for large molecule sizes.

From an energy perspective, OpenMP (OMP) has slightly higher instantaneous power values than MPI. For large molecules, GPU computations result in very low power values. The Nvidia HPC SDK framework brings to VASP 6.2.1 lower power dissipation (Fig. 8b) compared to VASP 5.4.4, as well as lower energy consumption (Fig. 9b), which highlights the importance of assessing VASP's performance with different compilers.

Research on energy consumption has yielded similar results in the work of Stegailov [28–30]. The energy consumption of GPU calculations is lower than MPI for molecules larger than 40 atoms. OMP improves energy efficiency compared with MPI-only implementation, but GPU–OMP combination brings us the best performance in terms of energy consumption.

5 Related work

HPC systems are becoming more and more popular due to the increasing use in various areas of knowledge of simulation techniques. According to the global energy development policy, these systems must optimize their performance and, at the same time, make efficient use of the energy consumed. Simulations and scientific applications for computational chemistry have taken advantage of the increase in performance offered by new generations of supercomputers.

The article by Hacene et al. [21] presents a way to improve the performance of the VASP program in material simulations using *ab initio* methods. The authors discuss how systems equipped with GPUs can greatly reduce program computation time. Yasuda [31] and Genovese [32] studied how to speed up these calculations using GPUs. Since VASP version 6, OMP parallelism combined with MPI and GPU can be used. Wende et al. [9] describe different parallelization strategies for *ab initio* algorithms in multicore systems using MPI/OMP communications between processes. Zhao et al. [33] add to these aspects various memory access methods. Several authors discuss MPI communications [34] and the binding of processes to cores in VASP [12, 35]. According to Peter Larsson [36], one of the most important parameters to improve performance with GPUs is NSIM. This parameter manages the number of bands[1] that are optimized in parallel using matrix-matrix operations. The author recommends increasing its value as long as possible and avoiding consuming the memory of these devices. Maniopoulou et al. [11] present a parallelization strategy for *k-point* workload distribution using KPAR (number of *k-points* treated in parallel) and NPAR (number of bands treated in parallel) parameters. In other articles, more parameters for performance improvement are considered [13, 37]. Articles [38–40] introduce performance improvements using certain libraries, reducing MPI communications, and moving computation to the GPU. Although other metrics have been presented to make performance results independent of the different HPC architectures used [41], the measurement of the time per iteration (LOOP+) of the main loop of the *Kohn-Sham* equation resolution algorithm is considered an accepted reference. Articles [9, 26, 28–30, 33] consider this measurement as a reference to measure performance. As in the mentioned articles, we perform our performance measurements using LOOP+ with different molecule sizes using the same *k-point* mesh.

Currently, it is essential to align high performance computing with contained energy consumption. Calore et al. [42] study the power consumption and performance of mobile processors running HPC applications. They study the power performance of a Tegra K1 mobile processor that runs an HPC application on the CPU and GPU. They analyze current measurements obtained through a customized monitor.

Stegailov et al. make several comparisons between different Intel and AMD CPUs [29, 30] for HPC solutions related to efficiency and energy consumption with

[1] In VASP, the bands represent the energy levels of the electrons in a material, while the orbitals represent the individual electron wavefunctions that contribute to those energy levels.

VASP. Also, Stegailov et al. [28] use a reference metric that makes it possible to compare different types of Intel, AMD and ARM processors together with GPUs. They compared the efficiency of VASP taking a GaAs molecule of 80 atoms. To do this, they use a metric based on the balance between maximum floating point performance and memory bandwidth that allows them to compare different systems with CPUs and GPUs. In addition, they measure energy consumption using digital wattmeters.

Most of these research studies focus on analyzing the performance or energy consumption of a specific version, typically with a fixed size. However, our study takes an integrated approach by analyzing different versions, varying parameter values, and jointly examining both performance and energy consumption. We also include an analysis of version 6, which has not been previously considered, due in part to installation difficulties and significant changes to the GPU version that has now migrated from CUDA-C to OpenACC. Our research group has extensive experience in analyzing and modeling the performance of large applications such as Linpack [43], as well as assessing energy efficiency in heterogeneous architectures multicore-GPU [44]. We have developed a Fortran interface for EML to measure the energy consumption of VASP. Our experiments involve different molecule sizes with the same sampling mesh from the VASP code itself, without the need for external physical instrumentation. We explore a range of parallelism models used in VASP, including Message Passing with MPI, OpenMP, and GPU, with CUDA-C and OpenACC implementations. The methodology used to plan our experiments can provide guidance to other researchers looking to optimize the performance of VASP for their specific hardware.

6 Conclusions

An energy and performance analysis of the VASP legacy application has been developed for a single two-socket HPC node. The results of our study are valuable to developers and researchers looking to select the best combination of compilers, parallelism model, and hardware for a high-performance and energy-efficient computing of VASP.

The VASP code has been compiled using Intel and Nvidia (Portland Group) compilers and several other libraries that provide different performance. It was also instrumented with the EML library for energy efficiency measurements.

Initial experiments carried out with Intel processors showed that the best execution times were achieved with the MKL library, so it was used for the following tests. Furthermore, it was found that the main factor in improving MPI performance is increasing the number of threads executed, but exceeding the number of threads in a *socket* leads to a decrease in performance due to communications between *sockets*. In GPU calculations, increasing the value of the NSIM parameter significantly improves performance with Intel oneAPI in VASP version 5.4.4, but a high value can cause memory failure on the GPU. For version 6.2.1 of VASP, the OpenACC implementation does not use the NSIM parameter anymore. Problems that benefit

from computing with GPUs are those with the highest workload caused by the size of the molecule and/or the number of *k-points*.

Our findings indicate that the combination of MPI, OpenMP, and OpenACC for GPUs shows the best performance and energy efficiency if the number of atoms in the structure is high enough. For small molecules, using multicore-only hardware with MKL and the Intel compiler will provide similar performance, and using a GPU is not necessary in such cases. Ultimately, our overarching objective is to thoroughly examine the different parallel implementations and to pinpoint areas where each can be improved, ultimately contributing to greater energy efficiency.

Ab initio algorithms in the VASP code are CPU bounded and energy consumption values are related to execution times. Therefore, structure sizes greater than 40 atoms suggest the use of GPU for this node. Experiments to collect energy consumption showed that the energy consumption of the MPI and OMP code versions increases significantly with the size of the molecule, while high-performance GPU executions have a much lower prolonged energy consumption against increasing molecular size, especially with version 6.2.1.

Author Contributions These authors contributed equally to this work.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work has been partially funded by the Ministry of Science and Innovation of Spain through the projects PID2019-107228RB-I00, TED2021-131019B-I00, and PDC2022-134013-I00; and by the Spanish Network CAPAP-H.

Availability of Data and Materials The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

Ethical Approval Not applicable.

References

1. Corporation N (2023) CUDA Nvidia parallel computing and programming platform. http://www.nvidia.com/object/cuda_home_new.html

2. Diaz J, Muñoz-Caro C, Niño A (2012) A survey of parallel programming models and tools in the multi and many-core era. IEEE Trans Parall Distrib Syst 23(8):1369–1386. https://doi.org/10.1109/TPDS.2011.308
3. Developer N (2023) OpenACC. https://developer.nvidia.com/openacc
4. Kresse G, Furthmüler J (2016) VASP the Guide. http://cms.mpi.univie.ac.at/vasp/vasp/vasp.html
5. Cabrera A, Almeida F, Arteaga J, Blanco V (2014) Measuring energy consumption using eml (energy measurement library). Comput Sci Res Develop. https://doi.org/10.1007/s00450-014-0269-5
6. Davidson ER (1983) Methods in computational molecular physics. G.H.F. Diercksen and S. Wilson, p 95
7. Wood DM, Zunger A (1985) A new method for diagonalising large matrices. J Phys A 18:1343
8. Pulay P (1980) Convergence acceleration in iterative sequences: the case of SCF iteration. Chem Phys 73:393
9. Wende F, Marsman M, Kim J, Vasilev F, Zhao Z, Steinke T (2019) OpenMP in VASP: threading and SIMD. Int J Quantum Chem 119:25851
10. Alfonso Muñoz ULL pages: Bandas. A Program for Teaching Solid State Physics (2018). http://amunoz.webs.ull.es/indexen.htm
11. Maniopoulou A, Davidson ERM, Grau-Crespo R, Walshd A, Busha IJ, Catlow CRA, Woodley SM (2012) Introducing k-point parallelism into VASP. Comput Phys Commun 183:1696–1701
12. Shainer G, Lui P, Hilgeman M, Layton J, Stevens C, Stemple W, Schultz S, Ludden G, Mora J, Kresse G (2013) Maximizing application performance in a multi-core, NUMA-aware compute cluster by multi-level tuning. Lect Notes Comput Sci 7905
13. Hutchinson M, Widom M (2012) VASP on a GPU: application to exact-exchange calculations of the stability of elemental boron. Comput Phys Commun 183:1422–1426
14. OpenMP: OpenMP ARB (Architecture Review Boards) (2023). https://www.openmp.org/
15. at Linköping University, N.-N.S.C.: Running VASP on Nvidia GPUs (2018). https://nsc.liu.se/pla/blog/2015/11/16/vaspgpu/
16. Zone ID (2022) Intel Fortran Compiler for oneAPI. https://www.intel.com/content/www/us/en/developer/articles/release-notes/oneapi-fortran-compiler-release-notes.html
17. Computing OSHP (2018) Open MPI. https://open-mpi.org/
18. Zone ID (2021) Intel MPI Library - OneAPI. https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/mpi-library.html#gs.1sth91
19. 22.11 NH (2023) Nvidia HPC SDK documentation. https://docs.nvidia.com/hpc-sdk/archive/22.11/compilers/hpc-compilers-user-guide/index.html
20. ULL HPCG-(2021) Energy Measurement Library. https://github.com/HPC-ULL/eml
21. Hacene M, Anciaux-Sedrakian A, Rozanska X, Klahr D, Guignon T, Fleurat-Lessard P (2012) Accelerating VASP electronic structure calculations using graphic processing units. J Comput Chem 33(32):2581
22. Combining MPI and OpenMP (2023). https://www.vasp.at/wiki/index.php/Combining_MPI_and_OpenMP
23. Jain A, Ong SP (2022) The Materials Project. https://materialsproject.org/
24. Jain A, Ong SP, Hautier G, Chen W, Richards WD, Dacek S, Cholia S, Gunter D, Skinner D, Ceder G, Ka Persson (2013) The Materials Project: a materials genome approach to accelerating materials innovation. APL Mater 1(1):011002. https://doi.org/10.1063/1.4812323
25. The Materials Project: Data retrieved from the Materials Project for various materials (mp-#) from database version v2023.11.1 (2024). https://materialsproject.org/
26. Wende F, Marsman M, Zhao Z, Kim J (2017) Porting VASP from MPI to MPI + OpenMP [SIMD]. ResearchGate 319138405
27. Marqueño T, Pellicer-Porres J, Errandonea D, Santamaria-Perez D, Martinez-Garcia D, Rodríguez-Hernández P, Muñoz A, Nieves-Pérez I, Achary S, Betinelli M (2022) Lattice dynamics of zircon-type NdVO4 and Scheelite-type PrVO4 under high-pressure. J Phys: Condens Matter 34:025404
28. Stegailov V, Smirnov G, Vecher V (2019) VASP hits the memory wall: processors efficiency comparison. Wiley, Hoboken, p 31
29. Stegailov V, Vecher V (2018) Efficiency analysis of intel, AMD and Nvidia 64-Bit hardware for memory-bound problems: a case study of ab initio calculations with VASP. Springer International Publishing AG 10778:81–90
30. Stegailov V, Vecher V (2017) Efficiency analysis of intel and AMD x86 64 architectures for Ab initio calculations: a case study of VASP. Springer International Publishing AG 793:430–441

31. Yasuda K (2008) Accelerating density functional calculations with graphics processing unit. J Chem Theory Comput 4:1230–1236
32. Genovese L, Ospici M, Deutsch T, Méhaut J-FM, Neelov A, Goedecker S (2009) Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures. J Chem Phys 131:034103
33. Zhao Z, Marsman M, Wende F, Kim J (2017) Performance of hybrid MPI/OpenMP VASP on cray XC40 based on intel knights landing many integrated core architecture. In: Proceedings, vol 134s2
34. Dinan J, Balaji P, Goodell D, Miller D, Snir M, Thakur R (2013) Enabling MPI interoperability through flexible communication endpoints. In: Proceedings of the 20th European MPI Users' Group Meeting, EuroMPI '13, pp 13–18
35. Sun G, Kürtia J, Rajczy P, Kertesz M, Hafner J, Kresse G (2002) Performance of the Vienna ab initio simulation package (VASP) in chemical applications. J Mol Struct (Theochem) 624:37–45
36. Larsson P (2015) Running VASP on Nvidia GPUs. National Supercomputer Centre at Linköping University
37. Maintz S, Eck B, Dronskowski R (2011) Speeding up plane-wave electronic-structure calculations using graphics-processing units. Comput Phys Commun 182:1421–1427
38. Jia W, Cao Z, Wang L, Fu J, Chi X, Gao W, Wang L-W (2012) The analysis of a plane wave pseudopotential density functional theory code on a GPU machine. Comput Phys Commun 184:9–18
39. Jia W, Fu J, Cao Z, Wang L, Chi X, Gao W, Wang L-W (2013) Fast plane wave density functional theory molecular dynamics calculations on multi-GPU machines. J Comput Phys 251:102–115
40. Borzilov V, Lozhnikov V, Prudnikov P, Mamonova M, Mamonov A, Sorokin A, Baksheev G (2019) Ab initio calculation of multilayer magnetic structures by VASP on OpenPOWER high performance system. International Conference on Computer Simulation in Physics and Beyond, vol 1163
41. Stegailov V, Orekhov ND, Smirnov GS (2015) HPC hardware efficiency for quantum and classical molecular dynamics. Springer International Publishing Switzerland 9251:469–473
42. Calore E, Schifano SF, Tripiccione R (2015) Energy-performance tradeoffs for HPC applications on low power processors. Springer International Publishing Switzerland 9523:737–748
43. Cabrera A, Acosta A, Almeida F, Blanco V (2020) A dynamic multi-objective approach for dynamic load balancing in heterogeneous systems. IEEE Trans Parallel Distrib Syst 31(10):2421–2434. https://doi.org/10.1109/TPDS.2020.2989869
44. Cabrera A, Almeida F, Blanco V, Giménez D (2013) Analytical modeling of the energy consumption for the high performance linpack. In: 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2013, Belfast, United Kingdom, February 27–March 1, 2013. IEEE Computer Society, USA, pp 343–350. https://doi.org/10.1109/PDP.2013.56